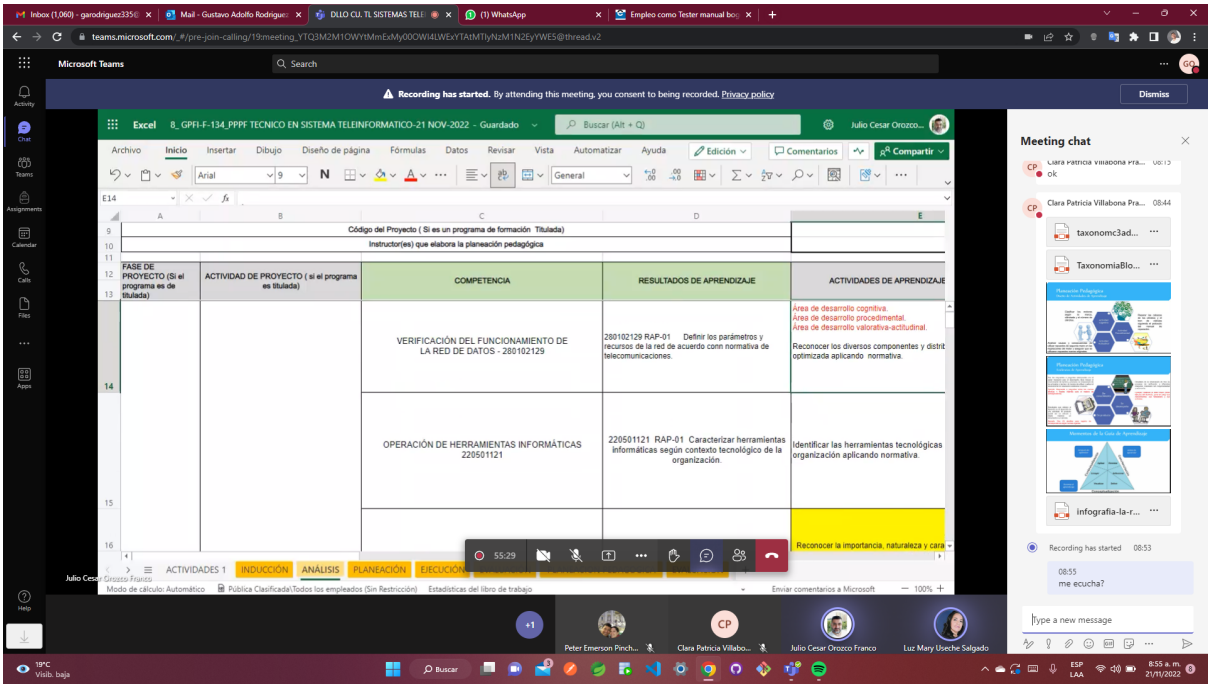
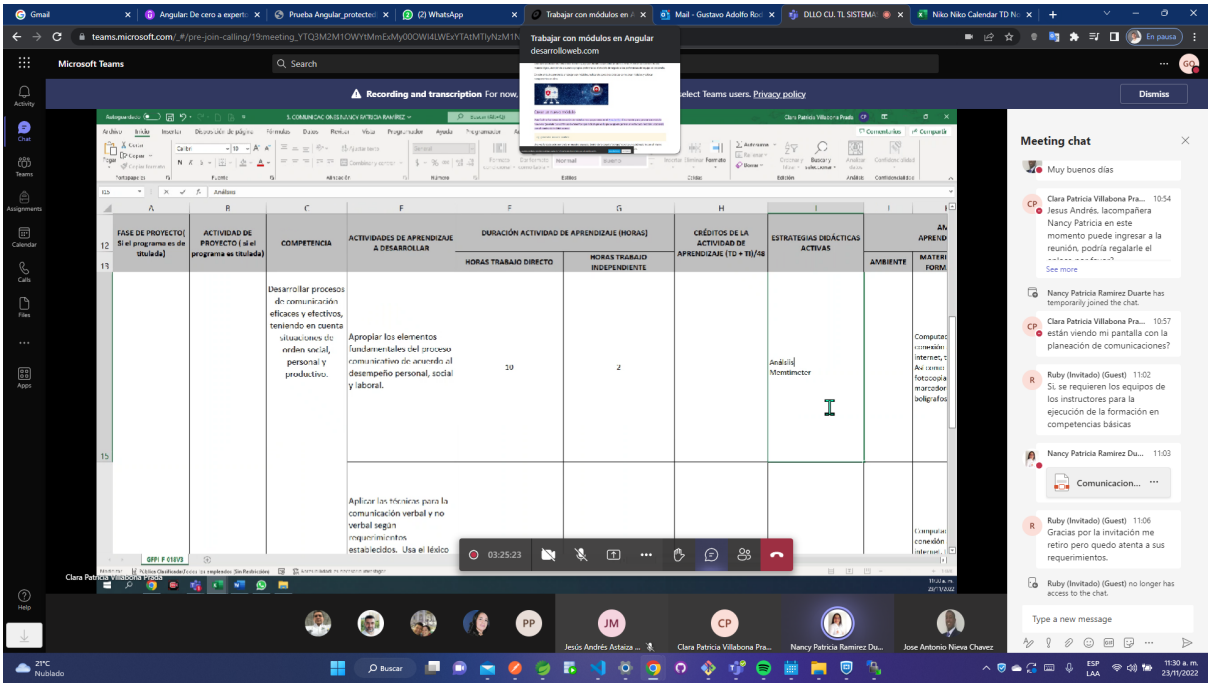
21/11/2022



23/11/2022



PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: **TECNICO EN SISTEMAS TELEINFORMATICO**
- Código del Programa de Formación: **233108 v 1.0**
- Nombre del Proyecto: **Fortalecimiento de los servicios teleinformáticos de equipos y redes en las MiPymes de la región de Santander.**
- Fase del Proyecto: **ANALISIS**
- Actividad de Proyecto: **DIAGNOSTICAR EL ESTADO DE LA INFRAESTRUCTURA DE REDES Y EQUIPOS INFORMATICOS Y SOLICITUDES DE SERVICIO TÉCNICO PRESENTE EN EL ENTORNO DE TRABAJO.**
- Competencia: **OPERAR HERRAMIENTAS INFORMÁTICAS Y DIGITALES DE ACUERDO CON PROTOCOLOS Y MANUALES TÉCNICOS**
- Resultados de Aprendizaje Alcanzar:
    - RAP1 - Caracterizar herramientas informáticas según contexto tecnológico de la organización.
- Duración de la Guía: 40 horas

2. PRESENTACIÓN

El manejo y diagnostico de programas y herramientas que hacen parte del funcionamiento de la infraestructura técnica de las empresas, representan un conjunto clave en las actividades cotidianas de personal técnico, dado que es en estas que se detectan fallas, errores o daños reportados por los usuarios. Es por esto que es fundamental adquirir habilidades necesarias para instalación y ejecución, de sistemas operativos, software, parches, reparaciones físicas, entre otros, con la finalidad de permitir dar continuidad en la prestación del servicio.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

3.1 Actividades de Reflexión inicial.

Analisis de caso: Llego a su entorno laboral un grupo de 15 computadores nuevos para ensamblar con todos sus componentes, finalizado el proceso se encienden para verificar el funcionamiento, pero arroja el mensaje que se visualiza a continuación:



Como técnico tiene que verificar que elementos se encuentran faltantes y completar su instalacion para la total operatividad de los equipos, resuelva las siguientes inquietudes en base a sus conocimientos

¿A qué elemento faltante hace referencia el mensaje en pantalla?

¿Cómo determina el elemento faltante sea el apropiado, en base a los demás componentes?

¿Existe la posibilidad de que el computador sea funcional en este momento? ¿Por qué?

¿Cuáles son los elementos físicos con los que ecuenta la entidad?

¿Existe una lista blanca de componentes software permitidos? ¿Cuáles son?

Participe en el foro temático dispuesto por su instructor en el cual aportara brindando respuesta a las inquietudes y opinando sobre los aportes de sus compañeros

NOTA: Tenga muy presente la ortografía, buena redacción y gramática para redactar, sea claro y coherente con la información que transmite para que esta no sea distorsionada.

3.2 Actividades de contextualización e identificación de conocimientos

Realice la búsqueda en la sopa de letras las palabras asociadas a los sistemas operativos, en específico a la temática de restauración y formateo.

w	i	p	i	x	d	a	d	i	c	a	p	a	c
h	z	p	y	p	c	f	e	r	r	o	r	s	p
y	j	u	r	o	o	t	d	i	o	r	d	n	a
p	b	t	p	r	m	j	a	j	a	l	e	f	n
e	r	e	s	t	a	u	r	a	c	i	o	n	t
r	u	s	i	a	n	x	y	z	t	n	s	g	a
v	s	t	s	b	d	m	n	o	e	u	j	o	l
i	u	a	t	l	o	n	e	s	r	x	e	h	l
s	a	l	e	e	s	t	p	d	m	t	p	l	a
o	r	v	m	v	a	v	q	v	i	r	u	s	a
r	i	e	a	m	a	c	o	s	n	o	k	h	z
a	o	p	r	o	g	r	a	m	a	s	c	g	u
h	c	o	n	s	o	l	a	i	l	d	a	v	l
i	f	m	s	d	o	s	s	m	s	q	b	b	x
n	s	s	o	i	b	w	i	n	d	o	w	s	w
o	i	o	i	r	a	u	s	u	r	e	p	u	s

restauración
virus
root
medio
portable
pantalla azul
backup
bios
consola
terminal
comandos
Setup
iso
Windows
Linux
Android

programas
Mac os
hypervisor
capacidad
sistema
ms dos
usuario
superusuario
formateo

3.3 Actividades de apropiación del conocimiento

Actividad 1: A continuación, en la tabla que se muestra a continuación encontrara una serie d de archivo, para cada una disponga un logotipo o imagen del programa con el cual puede ejecutarlo y una breve descripción de esta

.doc	Hace referencia a documentos con contenido tipo texto, puede contener imágenes e hipervínculos	.xlsx		.accdb	
.mpp		.ppt		.pub	
.cda		. ogv		.tar	
.exe		.pdf		.rar	
.wma		. vssm		.txt	
.sys		.png		.html	

Actividad 2: En el recuadro se relacionan varios equipos de cómputo de los que se indica el sistema operativo, en base a sus conocimientos, investigacion y orientación de su instructor, indique cuales son los requerimientos mínimos para su ejecución, de forma tal que permita que sea funcional por un largo plazo de tiempo.

NOTA: Tenga en cuenta que solo se permite el uso e instalacion de herramientas licenciadas

	SISTEMA OPERATIVO	DESCRIPCION	
Ejemplo	Linux Lite	Procesador. RAM. Espacio en disco duro.	Intel Centrino o superior 1Gb 100Gb
	ArcaOS	Procesador. RAM. Espacio en disco duro.	
	Windows 10	Procesador. RAM. Espacio en disco duro.	
	Ubuntu	Procesador. RAM. Espacio en disco duro.	
	macOS Mojave	Procesador. RAM.	

	Espacio en disco duro.	
Windows 8	Procesador. RAM. Espacio en disco duro.	
Chromium OS	Procesador. RAM. Espacio en disco duro.	
Mac OS X Leopard	Procesador. RAM. Espacio en disco duro.	

3.4 Actividades de transferencia del conocimiento.

Actividad 1: Realice un manual bajo los parámetros designados por su instructor en el cual se evidencie la instalación de:

- Distribución Linux designada por su instructor al igual que la tabla de particiones con home, swap, boot y /, el proceso deberá ejecutarse en la herramienta VMware, de cada etapa del proceso tome una captura de pantalla para ilustrar el documento
- Sistema Operativo Windows designada por su instructor al igual que las particiones primaria, lógica y extendida según sea el caso el proceso deberá ejecutarse en la herramienta VirtualBox, de cada etapa del proceso tome una captura de pantalla para ilustrar el documento

Actividad 2: Realice las actividades relacionadas en **taller 1_guia1** complementarias a la guía de aprendizaje, al finalizar adjunte los resultados al documento evidencia

Descripción del Informe técnico:

Documento bajo normas IEEE, que incluya todo el desarrollo del taller, asigne el nombre al documento de:

Guia1_Taller1_Nombre Aprendiz.doc

- Abstract
- Keywords
- Introducción (¿De qué trata el documento?)
- Contenido: (desarrollo de las actividades planteadas)
- Conclusiones (¿Qué aprendí?)
- Bibliografia (bajo normas IEEE)

Recuerde realizar la verificación de resultados, las respuestas a todos los puntos y ortografía antes de enviar el taller resuelto. Envíe el archivo al enlace dispuesto por su instructor en la plataforma

- Ambiente Requerido
  Taller o Aula con computadores de escritorio o portátiles con acceso a internet, sistemas operativos, paquete office, memorias USB, ISOs, antivirus.
- Materiales
  Torres de computador de escritorio para instalacion y prueba de sistemas operativos con todos sus componentes internos funcionales

- **Agregar instalación de base de datos**
- **Backups hacia el rap 2**
- **Manipular herramientas ofimáticas**
- **Contextualizar**

4. ACTIVIDADES DE EVALUACIÓN

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento : Establece 5 herramientas ofimaticas para el registro de informacion	Diferencia herramientas tic, tipos de software y servicios de internet, de acuerdo con las necesidades de la organización.	
Evidencias de Desempeño: Clasifica las diferentes herramientas para la gestion y control de la informacion requeridas para la organizacion	Diferencia herramientas tic, tipos de software y servicios de internet, de acuerdo con las necesidades de la organización.	
Evidencias de Conocimiento : Establece el sistema operativo para los equipos provistos por la organizacion	Reconoce las características del sistema operativo, los sistemas informáticos según el tipo de dispositivo y su función en la gestión de la información.	
Evidencias de Producto: Establece un documento de control (backlog) para relacionar el sistema operativo versus el equipo fisico	Emplea herramientas ofimáticas de apoyo de acuerdo con las necesidades requeridas por el usuario.	

5. GLOSARIO DE TÉRMINOS

Bios: (sigla en inglés de basic input/output system) software que localiza y reconoce todos los dispositivos necesarios para cargar el sistema operativo en la memoria RAM; es un software muy básico instalado en la placa base, usualmente está escrito en lenguaje ensamblador.

Fichero: conjunto de bytes que son almacenados en un dispositivo. Un archivo es identificado por un nombre y la descripción de la carpeta o directorio que lo contiene. archivo

Línea de comandos: método que permite a los usuarios dar instrucciones a algún programa informático por medio de una línea de texto simple.

Hipervisor: o monitor de máquina virtual es una plataforma que permite aplicar diversas técnicas de control de virtualización para utilizar, al mismo tiempo, diferentes sistemas operativos en una misma computadora

Restauración: herramienta implementada en Windows que permite devolver al ordenador a un estado anterior. Recopila los archivos para quitarse y el estado que estaba el sistema y luego se ve un cargador que va devolviendo al equipo su estado anterior. Luego reinicia el sistema.

Superusuario o root: es el nombre convencional de la cuenta de usuario que posee todos los derechos en todos los modos. Normalmente es la cuenta de administrador en sistemas operativos del tipo Unix

UEFI: Unified Extensible Firmware Interface es una especificación que define una interfaz entre el sistema operativo y el firmware, reemplaza la antigua interfaz del Sistema Básico de Entrada y Salida estándar presentado en las computadoras personales IBM PC como IBM PC ROM BIOS

Unix sistema operativo, es decir, es una colección de programas que ejecutan otros programas en una computadora. UNIX nació en los Laboratorios Bell de AT&T en 1969, desarrollado por Ken Thompson y Dennis Ritchie (también creador del lenguaje de programación C)

6. REFERENTES BILBIOGRÁFICOS

Bellido, Q. E. (2013). Instalación y actualización de sistemas operativos (uf0852). Retrieved from https://ebookcentral-proquest-com.bdigital.sena.edu.co

Bellido, Q. E. (2013). Instalación y configuración de sistemas operativos. Retrieved from https://ebookcentral-proquest-com.bdigital.sena.edu.co

Canonical (2020). Ubuntu 20.04 LTS. Canonical Ltd. Recuperado de: https://ubuntu.com/download/desktop

Carvajal, P. F. (2017). Instalación y actualización de sistemas operativos transversal: Uf0852. Retrieved from https://ebookcentral-proquest-com.bdigital.sena.edu.co

La, R. M. D. L. (2004). Sistemas operativos. Retrieved from https://ebookcentral-proquest-com.bdigital.sena.edu.co

Microsoft (2020). Descargar imagen de disco de Windows 8.1. Microsoft (archivo ISO). Recuperado de https://www.microsoft.com/es-es/software-download/windows8ISO

Romero, R., Pérez, R., & Escrivá, G. (2013). Sistemas operativos en red. Retrieved from https://ebookcentral-proquest-com.bdigital.sena.edu.co

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	**Gustavo Adolfo Rodriguez Quinayas**	**Experto temático**	**Centro de comercio y servicios, regional Cauca**	**11/11/22**

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

Taller

Gestión de inventario y selección de herramientas de la organización.

Las empresas pueden contar un sin número de elementos hardware y/o software para la correcta ejecución de operaciones y actividades, lo cual es requerido que todos los elementos estén funcionando correctamente de acuerdo con las políticas de la entidad. Las empresas suelen tener un inventario de herramientas y licencias, las cuales son permitidas para la ejecución de actividades de sus colaboradores, aunque es común no tener el estado actual de las mismas, permitiendo así tener un riesgo al aumentar la probabilidad de fallo.

Caso de estudio

La fábrica de software del SENA, de la regional cauca, ***CaucaSoft-SENA***, opera en uno de los centros de formación de la regional cauca, la cual funciona como una unidad productiva que presta los servicios de desarrollo de software, esta unidad cuenta con diferentes herramientas de desarrollo tecnológico tales como equipos de cómputo MAC, portátiles, notebooks, tablets, dispositivos móviles, Smart tvs, Smart waths, lectores de NFC, impresoras convencionales, impresoras 3D, de la misma manera licencias de Windows, OSX, Herramientas de diseño, SDK's de desarrollo, paquetes de ofimática, entre otros.

Esta unidad fue lanzada el primero de enero de 2022, por lo cual muchas de las actividades de mantenimiento, control y seguimiento de la hoja de vida de las herramientas no han sido alimentada de la manera más adecuada, por lo cual su actividad como técnico en mantenimiento es realizar las siguientes actividades.

Desarrollo de la actividad

Para cada uno de los siguientes literales dar respuesta a partir de la lectura de la información de apoyo para el componente formativo.

Taller

1. Realizar un inventario de las herramientas hardware con las que cuenta la entidad
2. Construir la hoja de vida de cada elemento, haciendo uso de las plantillas y herramientas ofimáticas (Word, Excel, Ppt)
3. Realizar un inventario de las herramientas software con las que cuenta la unidad, (Sistemas operativos, paquetes ofimáticos, herramientas de desarrollo)
4. Realizar un inventario de las licencias con las que cuenta la unidad
5. Validar o establecer la lista blanca de las herramientas software permitidas por la entidad, identificando versiones permitidas, licencias de la entidad o licencias opensource
6. Proponer un mecanismo de almacenamiento (drive, svn, git) para la gestión de los recursos establecidos en la guía.
7. Realice un diagnóstico de las herramientas hardware teniendo en cuenta los paquetes y software instalados establecidas en el inventario.

PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: **TECNICO EN SISTEMAS TELEINFORMATICO**
- Código del Programa de Formación: **233108 v 1.0**
- Nombre del Proyecto: **Fortalecimiento de los servicios teleinformáticos de equipos y redes en las MiPymes de la región de Santander.**
- Fase del Proyecto: **EVALUACION**
- Actividad de Proyecto: **VALIDAR LOS PROCEDIMIENTOS REALIZADOS EN LOS SISTEMAS TELEINFORMÁTICOS..**
- Competencia: **OPERAR HERRAMIENTAS INFORMÁTICAS Y DIGITALES DE ACUERDO CON PROTOCOLOS Y MANUALES TÉCNICOS**
- Resultados de Aprendizaje Alcanzar:
    - RAP4 - Documentar la gestión de la información acuerdo a protocolos establecidos.

Duración de la Guía: 48 horas

2. PRESENTACIÓN

El informe técnico es una herramienta indispensable en el entorno laboral, dado que es en esta que se estipula la información y los datos mas importantes sobre un proceso, procedimiento, equipo o evento que se presenta a diario en las entidades, de tal manera que es un mecanismo de defensa y precaución ante cualquier mantenimiento o proceso llevado a cabo, dado que consignan las acciones y protocolos ejecutados durante un soporte.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

3.1 Actividades de Reflexión inicial.

Analisis de caso: ha sido ascendido al cargo de coordinador en su area por su buen desempeño, se requiere realizar reportes diarios del avance en los mantenimientos realizados a los equipos para notificar al cliente quien desconoce por completo el funcionamiento de un computador, al verificar un informe enviado antes de su ascenso, correspondiente a un equipo de gerencia al cual se le realizaron varios cambios de hardware y actualización de programas, el cliente no se encuentra muy satisfecho porque según él se le está facturando partes y suministros como la pasta térmica, limpia contactos entre otros, que no se encuentran relacionados en el informe técnico del mantenimiento de dicho equipo, razón por la cual el desea cancelar el contrato.

- Indique la importancia del informe técnico en este caso
- ¿Cuál fue el error del personal que realizo el mantenimiento al equipo?
- ¿Como se puede evitar los conflictos de este tipo en un entorno laboral?
- ¿Cuándo se debe realizar el informe técnico?

Participe en el debate organizado, previo apunte de sus ideas, para lo cual tenga muy presente la ortografía, buena redacción y gramática para redactar, sea claro y coherente con la información que transmite para que esta no sea distorsionada.

Participe en el foro temático dispuesto por su instructor en el cual aportara brindando respuesta a las inquietudes y opinando sobre los aportes de sus compañeros

NOTA: Tenga muy presente la ortografía, buena redacción y gramática para redactar, sea claro y coherente con la información que transmite para que esta no sea distorsionada.

3.2 Actividades de contextualización e identificación de conocimientos

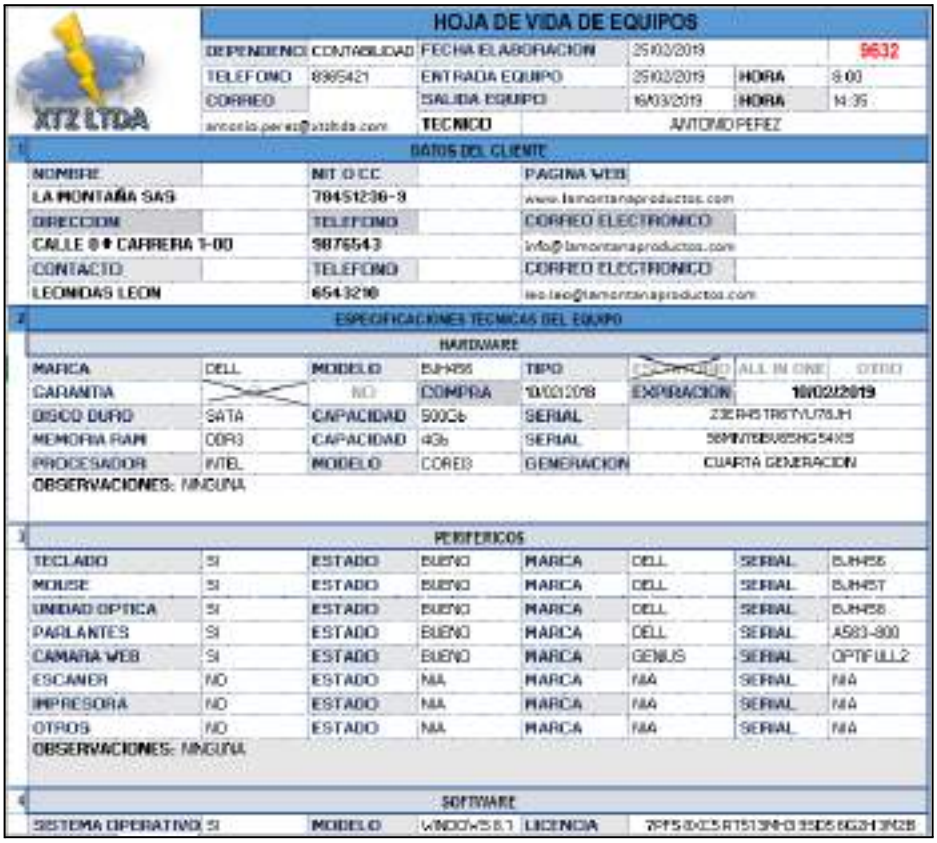
En la nube de palabras que se muestra a continuación identifique mínimo 10 palabras relacionadas con la realización de informes técnicos y describa para cada una su importancia dentro del ámbito del mantenimiento de equipos



Fuente: Abir Zubiria

3.3 Actividades de apropiación del conocimiento

Dentro de los informes técnicos encontramos los formatos, que nos permiten unificar la informacion de manera organizada y estandarizada para todas las actividades que realiza la empresa, por tanto se hace necesario la creación de formatos fundamentales según sea la necesidad. Elabore un formato de hoja de vida en MS Excel para facilitar el proceso, agregue tantas secciones como sea necesario, inserte formulas, funciones y celdas desplegables para los casos que asi lo requieran. Adicional elabore un form bitácora, otro formato con frecuencia para el registro y reporte de novedades. A continuación s parte de un formato como guía del que se busca construir.



HOJA DE VIDA DE EQUIPOS

XTZ LTDA

DEPENDENCIA	CONTABILIDAD	FECHA ELABORACION	25/02/2019		9632
TELEFONO	8965421	ENTRADA EQUIPO	25/02/2019	HORA	8:00
CORREO		SALIDA EQUIPO	16/03/2019	HORA	14:35
antonio.perez@xtzltda.com		TECNICO	ANTONIO PEREZ		

DATOS DEL CLIENTE

NOMBRE	NIT O CC	PAGINA WEB	
LA MONTAÑA SAS	70451236-3	www.lamontanaproductos.com	
DIRECCION	TELEFONO	CORREO ELECTRONICO	
CALLE 8 # CARRERA 1-00	5875543	info@lamontanaproductos.com	
CONTACTO	TELEFONO	CORREO ELECTRONICO	
LEONIDAS LEON	6543210	leo.leo@lamontanaproductos.com	

ESPECIFICACIONES TECNICAS DEL EQUIPO

HARDWARE

MARCA	DELL	MODELO	BJHK56	TIPO	~~COMPUTADOR~~	ALL IN ONE	OTRO
GARANTIA	~~SI~~	NO	COMPRA	10/03/2018	EXPIRACION	10/03/2019	
DISCO DURO	SATA	CAPACIDAD	500Gb	SERIAL	23ERHSTRSTYU76JH		
MEMORIA RAM	DDR3	CAPACIDAD	4Gb	SERIAL	56MNTSBV65HG54XS		
PROCESADOR	INTEL	MODELO	CORE3	GENERACION	CUARTA GENERACION		

OBSERVACIONES: NINGUNA

PERIFERICOS

TECLADO	SI	ESTADO	BUENO	MARCA	DELL	SERIAL	BJHK56
MOUSE	SI	ESTADO	BUENO	MARCA	DELL	SERIAL	BJHK57
UNIDAD OPTICA	SI	ESTADO	BUENO	MARCA	DELL	SERIAL	BJHK58
PARLANTES	SI	ESTADO	BUENO	MARCA	DELL	SERIAL	A583-900
CAMARA WEB	SI	ESTADO	BUENO	MARCA	GENIUS	SERIAL	OPTIFULL2
ESCANER	NO	ESTADO	NA	MARCA	NA	SERIAL	NA
IMPRESORA	NO	ESTADO	NA	MARCA	NA	SERIAL	NA
OTROS	NO	ESTADO	NA	MARCA	NA	SERIAL	NA

OBSERVACIONES: NINGUNA

SOFTWARE

SISTEMA OPERATIVO	SI	MODELO	WINDOWS 8.1	LICENCIA	7PF5-0XC5-RT51-3NH0-35D5-6G2H-3N28

NOTA: Recuerde que todos los formatos deben entregarse diligenciados para evidenciar su adecuado uso. Verifique la ortografía buena redacción y gramática para redactar, sea claro y coherente con la información que transmite para que esta no sea distorsionada.

3.4 Actividades de transferencia del conocimiento.

A continuación se presenta un analisis de caso en base al cual deberá elaborar un informe técnico, describiendo la incidencia, diagnostico de falla y procedimiento de solución

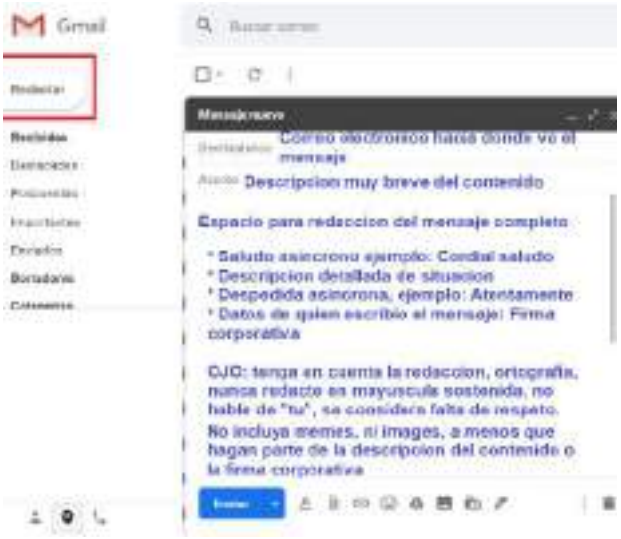
Analisis de caso:

En la empresa XTZ Ltda, con sedes en Bogota, Medellin y California en USA, se dedica a prestar el servicio de mantenimiento de equipos de computo para empresas estatales a modo de proyecto, se designo a uno de los técnicos para realizar los mantenimientos de 10 de los equipos provenientes de la Alcaldia. El coodinador de mantenimientos viaja para una capacitación y deja el técnico encargado de todo el proceso. Llega una solicitud de entrega urgente de 4 equipos que se requieren para la Alcaldia, el técnico inicia el mantenimiento del equipo 1, encontrando que esta quemada la fuente de alimentación porque se encuentra sellada con cinta la rejilla de ventilación de la misma, para poder justificar el reemplazo de la parte el técnico deberá informar a su coordinador por correo la novedad que presenta el equipo, a la Alcadia para autorizar el cobro de una fuente de alimentación nueva en el formato designado y a la sede en USA en el mismo formato de informe pero con la novedad de que quien recibe no habla español.

Elabore el correo electrónico dirigido al coordinador indicando exponiendo la situacion y solucion, un informe técnico para el personal de la alcaldia indicando su diagnostico y las partes en falla, por ultimo

deberá hacer una copia del informe redactado y realizar su traduccion al ingles para reportarlo a la sede en California.

Tips para la elaboración de correos electronicos



Formato sugerido para el informe tecnico



Descripción del Informe técnico:

Documento bajo normas APA edición 6, que incluya todo el desarrollo del taller, asigne el nombre al documento de:

GuiaXX_Nombre Aprendiz.doc

- Portada
- Introducción (¿De qué trata el documento?)
- Objetivos (¿Qué espero aprender?)
- Contenido: (desarrollo de las actividades planteadas)
- Conclusiones (¿Qué aprendí?)
- Bibliografia (bajo normas APA)

Recuerde realizar la verificación de resultados, las respuestas a todos los puntos y ortografía antes de enviar el taller resuelto. Envíe el archivo al enlace dispuesto por su instructor en la plataforma designada

3.4 Actividades de transferencia del conocimiento.

Actividad 1

Haciendo uso del material de apoyo realice un cuestionario que permita evaluar los ítems expuestos en el documento y ejecute la encuesta a 3 entidades.

Descripción del Informe técnico:

Documento bajo normas APA edición 6, que incluya todo el desarrollo del cuestionario, asigne el nombre al documento de:

GuiaXX_SolucionCuestionario.doc

- Portada
- Introducción (¿De qué trata el documento?)
- Objetivos (¿Qué espero aprender?)
- Contenido: (desarrollo de las actividades planteadas)
- Resultado de la encuesta
  - Por pregunta
  - Por entidad
- Recomendaciones
- Conclusiones (¿Qué entendí de los procesos de las entidades?)
- Bibliografia (bajo normas APA)

Recuerde realizar la verificación de resultados, las respuestas a todos los puntos y ortografía antes de enviar el taller resuelto. Envíe el archivo al enlace dispuesto por su instructor en la plataforma designada

Actividad 2

Haciendo uso del material de apoyo instale una de las herramitas HelpDesk expuestas y realice un manual de instalación, explicando como es el proceso de búsqueda, descarga, instalación, configuración y acceso a la herramienta. El manual debe indicar los ítems principales para crear nuevos soportes y registros en la herramienta.

Descripción del Informe técnico:

Documento bajo normas APA edición 6, que incluya todo el desarrollo del cuestionario, asigne el nombre al documento de:

GuiaXX_ManualInstalacion_HelpDesk.doc

- Portada
- Introducción (¿De qué trata el documento?)
- Objetivos (¿Qué espero aprender?)
- Contenido: (desarrollo de las actividades planteadas)
- Característica de la herramienta
- Descarga de la herramienta y documentación oficial
- Wizard de instalación
- Configuración

- Acceso
- Módulos de la herramienta
- Flujo grama para registro y seguimiento de incidencias
- Conclusiones (¿Qué entendí de los procesos de las entidades?)
- Bibliografia (bajo normas APA)

Recuerde realizar la verificación de resultados, las respuestas a todos los puntos y ortografía antes de enviar el taller resuelto. Envíe el archivo al enlace dispuesto por su instructor en la plataforma designada

4. ACTIVIDADES DE EVALUACIÓN

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento:	● Conocimiento relacionado con herramientas para elaboración de informes tecnicos ● Conocimiento sobre términos técnicos de los informes ● Conocimiento acerca de elaboración de formatos Conocimiento referente a elaboración de informes en formato y por correo	● Evaluación conceptual acerca de los conceptos estudiados
Evidencias de Desempeño	● Analisis del caso en base a conocimientos previos ● Identificacion de palabras relacionadas con informes tecnicos ● Elaboracion de formatos Elaboracion de correos y redacción de informes	● Observación y valoración de los resultados suministrados en cada uno de ítems creados
Evidencias de Producto:	● Participación en debate basado en el analisis de caso ● Descripcion de palabras en un ámbito tecnico ● Formato de hoja de vida de computador y bitácora diligenciado	Informe escrito bajo normas APA edición 6 que registre toda la informacion realizada tanto en la guía como en el taller.

	Informe técnico en base a analisis de caso en español, ingles y a traves de correo electronico	

5. GLOSARIO DE TÉRMINOS

Inspeccion: implica las mediciones, pruebas y medidores aplicados a ciertas características con respecto a un objeto o actividad. Técnica analítica de seguridad que consiste en un análisis, realizado mediante observación directa de las instalaciones, equipos y procesos productivos (condiciones, características, metodología del trabajo, actitudes, aptitudes, comportamiento humano…)

Manual: guía de instrucciones que sirve para el uso de un dispositivo, la corrección de problemas o el establecimiento de procedimientos de trabajo. Los manuales son de enorme relevancia a la hora de transmitir información que sirva a las personas a desenvolverse en una situación determinada.

MS Excel: hoja de cálculo desarrollada por Microsoft para Windows, macOS, Android e iOS. Cuenta con cálculo, herramientas gráficas, tablas calculares y un lenguaje de programación macro llamado Visual Basic para aplicaciones.

Reporte: Documento informativo que sirve para comunicar información que sea relevante, este puede ser un material creado por una empresa, organización o un trabajo de clases que sirve para dar mayor información sobre un tema en específico

Service Manual: va dirigido a un público con conocimientos técnicos sobre algún área. Se debe presentar una breve descripción del sistema desarrollado, que contemple el ámbito abarcado, cual es su función principal y un detalle de las funciones macros o partes que lo componen, con contenidos múltiples que trata de principios y técnicas de una actividad determinada. Se elabora como fuente básica de referencia para el órgano responsable de la actividad y como información generales para todo el personal interesado en esa actividad.

User Manual: documento que busca dar alguna información sobre cómo realizar algo en particular. Estos textos sirven como una guía para que el individuo sepa la manera correcta de realizar una acción o para que se informe sobre algo.

6. REFERENTES BILBIOGRÁFICOS

Calderón García, R. (2011). Manual para la elaboración de trabajos académicos y de investigación. Guadalajara, Jal, Mexico: Universidad de Guadalajara. Recuperado de https://elibro-net.bdigital.sena.edu.co/es/lc/senavirtual/titulos/39885.

Díaz Barriga Martínez, R. (). Redacción técnica. México, Instituto Politécnico Nacional. Recuperado de https://elibro-net.bdigital.sena.edu.co/es/lc/senavirtual/titulos/101778

Gutiérrez Madrigal, F. (2016). Técnicas de comunicación. San José, Costa Rica, Costa Rica: Editorial Secade. Recuperado de https://elibro-net.bdigital.sena.edu.co/es/lc/senavirtual/titulos/79262.

Mariscal, M. (2009). Manual de proceso de la información. México D.F, Mexico: Editorial Miguel Ángel Porrúa. Recuperado de https://elibro-net.bdigital.sena.edu.co/es/lc/senavirtual/titulos/38114.

Sosa Flores, M. (. (2006). Manual Básico de Excel. Santa Fe, Argentina: El Cid Editor. Recuperado de https://elibro-net.bdigital.sena.edu.co/es/lc/senavirtual/titulos/35033.

Tiburcio Pintos, G. y Álvarez Gutiérrez, L. C. (2020). Manual para la elaboración y presentación de anteproyectos, proyectos de investigación y tesis. Editorial Universo Sur. Recuperado de https://elibro-net.bdigital.sena.edu.co/es/lc/senavirtual/titulos/131890.

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)	**Gustavo Adolfo Rodriguez Quinayas**	**Experto temático**	**Centro de comercio y servicios, regional Cauca**	**12/12/22**

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

Material de apoyo

Tabla de contenidos

Introducción

1. **Conceptos generales**

    1.1. Beneficios de la gestión TI
    1.2. ¿Cómo y cuándo empezar la gestión TI?
    1.3. Buenas prácticas de la gestión TI
    1.4. Acuerdo de nivel de servicio

2. **Mesa de servicio**

    2.1. Diagnóstico y preparación
    2.2. Definición de la estructura de servicios
    2.3. Selección de roles y propietarios de roles
    2.4. Análisis de procesos existentes
    2.5. Definición de la estructura de procesos
    2.6. Definición de interfaces de ITIL
    2.7. Establecimiento de controles
    2.8. Diseño de los procesos
    2.9. Selección e implantación de sistemas
    2.10. Implementación de procesos y capacitación

    a. **Desarrollo de contenidos**

Introducción

En este componente formativo se estudian algunos conceptos, beneficios, buenas prácticas y acuerdos de nivel de servicio en gestión TI, relacionados con el tratamiento, almacenamiento y protección de la información a través de los servicios tecnológicos. Finalmente, se presentan las prácticas obligantes en la gestión de TI, sus marcos de trabajo y otras definiciones necesarias para la planeación, implementación, control y seguimiento de procesos.

1. Conceptos generales

La Gestión de Tecnologías de la Información (TI) es la disciplina enfocada en procesos para alinear los servicios de TI con las necesidades de las entidades y sus usuarios, centrándose en garantizar los beneficios que puede percibir el cliente final de las tecnologías gestionadas.

La gestión de servicios de TI se consolida en una integración de tres componentes: personas, procesos y tecnología como se muestra en el siguiente recurso:

El escenario general de los servicios de TI está asociado con el *outsoursing* (subcontratación o tercerización de procesos), servicios compartidos, por ejemplo, cuando la empresa usa GSuit, Teams, y también sistemas de

información particulares de la empresa o sistemas legados. Sea cual sea el caso, es importante mantener una base de conocimiento de la gestión de los servicios ya que se ha probado que esas prácticas son exitosas.

1.1. Beneficios de la gestión de TI

El concepto anterior brinda una idea general de lo que es la gestión de TI y sus componentes, la cual es importante revisar para establecer estrategias y prácticas en las corporaciones, de forma que se puedan aprovechar los siguientes beneficios:

- **Mejora en la productividad:** la implementación de herramientas como lo propone la gestión de TI hace que el trabajo sea más ágil, eliminando tareas repetitivas y demoradas. De esta forma, el talento humano se puede enfocar en aquellas actividades que hacen mejor o en las que son especialistas.
- **Mejora en la seguridad informática**: la información es muy importante para las empresas de cualquier sector. La gestión eficaz de TI también se encarga de velar por la ejecución operativa de muchos de los controles de seguridad informática.
- **Más calidad en la relación con los clientes**: permite establecer mejores y diferentes canales de atención al cliente o usuario. La tendencia es que estos canales se integren, por ejemplo, si un cliente entró en contacto vía *e-mail* o por WhatsApp y, luego decide llamar a la empresa para el mismo caso se espera que no necesite repetir todas las informaciones que dio en el primer contacto. De esta forma, una gestión de TI efectiva hace que desde cualquier área de la empresa se tenga acceso a la información importante para la atención que los usuarios esperan.
- **Optimización de procesos:** cuando la entidad no está usando el *software* adecuado o no ha mejorado sus procesos adaptando nuevas tecnologías, esto hace que la empresa esté atrasada frente al mercado en el sector económico al que pertenece. La gestión de TI puede ahorrar tiempo y dinero adoptando nuevas tecnologías en los procesos de la entidad, a la vez que la impulsa en su crecimiento.
- **Reducción de costos:** hace que los procesos puedan ser innovados constantemente, adoptando nuevas o mejores herramientas de *hardware* y *software* que optimicen los procesos de negocio haciendo a la empresa más productiva, debido a la reducción de costos o maximización de ganancias.
- **Identificar nuevas oportunidades:** posibilita saber en qué momento los clientes o usuarios cambian a un nuevo perfil para actualizar los servicios tecnológicos. Por tanto, se puede identificar el punto para ofrecer productos complementarios o nuevos servicios TI.

1.2. ¿Cómo y cuándo empezar la gestión de TI?

Muchos especialistas están de acuerdo en responder a la pregunta "**¿cuándo es el momento de implementar la gestión de TI?**", y la respuesta es "**lo antes posible**", desde el principio de ser posible.

Casi todo el *software* que existe en el mercado para hacer gestión de TI es escalable en funcionalidad y costos. Esto quiere decir que existen herramientas para la gestión de TI con costos que van desde cero pesos hasta el valor que tenga dentro de una organización, y también existen herramientas de funcionalidades básicas que pueden configurarse para soportar las más complejas. Según esto, el costo en la adquisición de herramientas para la gestión TI no es una limitante; sin embargo, el desconocimiento de cómo usarlas o adoptar estrategias para la integración de estas herramientas con la atención a los usuarios y la gestión de los servicios, sí lo puede ser.

Ahora, el escenario es el siguiente: existen herramientas para la gestión de TI de las que se puede disponer y también existen buenas prácticas ya definidas para la integración de estas herramientas con los procesos de

atención al usuario y la gestión de TI. Por lo tanto, la conclusión es que para mejorar la estrategia de gestión de TI las actividades a realizar son:

- **Estudiar las guías de buenas prácticas de TI:** accediendo a las librerías de buenas prácticas.
- **Búsqueda y selección herramientas de TI:** seleccionar la necesidad de gestión y las herramientas que se ajusten al presupuesto (aunque el presupuesto sea cero).

A la agrupación de mejores prácticas establecidas, su implementación e integración en los procesos en una entidad es lo que se denomina como **Gobierno de TI o Gobernanza de TI** y muchas de estas prácticas están documentadas en las librerías ITIL (Information Technology Infrastructure Library), librería que se ha internacionalizado y estandarizado a nivel mundial y ahora está en la Norma ISO/IEC 20000.

1.3. Buenas prácticas en la gestión de TI

La necesidad de innovación y transformación digital que cada vez más crece en las entidades crea un aumento en la importancia de la administración de servicios y procesos de tecnología. Para ello, existe una variedad de marcos de trabajo, los más empleados son:

- **ITIL**: es la sigla en inglés de Information Technology Infrastructure Library (ITIL), en español Biblioteca de Infraestructura de Tecnología de la Información. Es el modelo de gestión de TI más conocido y es la colección de mejores prácticas para acoplar TI a las necesidades específicas de una empresa. De esta forma proporciona una mejor optimización de los servicios prestados, con el objetivo de reducir los costos y mejorar la eficacia de los sectores involucrados.
  La serie ITIL está compuesta por cinco libros denominados Estrategia de servicio, Diseño del servicio, Transición del servicio, Operación del servicio y Mejora continua del servicio.
- **COBIT**: es la sigla en inglés de Control Objectives for Information and Related Technology, en español Objetivos de Control para las Tecnologías de la Información y Relacionadas. Es un marco de gobierno de TI cuyo propósito es divulgar conceptos y prácticas de TI; centra su atención en que las inversiones en TI estén alineadas con las necesidades de la empresa.
- **PMBOOK:** es la sigla en inglés de Project Management Book of Knowledge, en español Guía de los fundamentos para la dirección de proyectos. Es una de las referencias mejor calificadas en lo referente a la gestión de proyectos y cuya aplicabilidad en proyectos TI ha sido comprobada; esta metodología es a menudo aplicada en distintos sectores de la gestión de TI.
  PMBOK divide cada proyecto en cinco fases: estructuración, planeación, implementación, mantenimiento y gestión, cada una de ellas con actividades y contextos definidos. El propósito es apoyar la estructuración, planeación, implementación y mantenimiento de los procesos de la gestión de proyectos.

1.4. Acuerdos de nivel de servicio

Un acuerdo de nivel de servicio (ANS) o también conocido por las siglas en inglés SLA (Service Level Agreement), es un acuerdo escrito o verbal entre un proveedor de servicio TI y su cliente, con el fin de fijar el nivel acordado para la calidad en la prestación de dicho servicio. El ANS permite tener un consenso en términos del nivel de calidad del servicio para temas como tiempo de respuesta, documentación disponible, disponibilidad horaria, personal asignado al servicio, entre otros.

Cuando se elabora una propuesta técnica de un sistema para un cliente no basta con poner la descripción del producto, el costo, el tiempo de entrega, forma de pago, etc. También es importante agregar un ítem que describa cómo se clasifica el nivel servicio.

A continuación, se presenta un ejemplo de clasificación del nivel de servicios. En la primera tabla se definen los tipos de incidencias o solicitudes de soporte y en la segunda, el nivel de atención para cada tipo de incidencia, con lo cual queda descrito un acuerdo de nivel de servicio dentro de la propuesta técnica para un sistema informático web.

Tabla 1
Tipos de incidencias o solicitudes de soporte

Tipos de incidencias o solicitudes de soporte	
Normal	Hace referencia a una solicitud hecha por el director de tecnología o a quien delegue, donde no se ve comprometido o afectado de manera considerable el servicio para el cual se solicita el soporte. El nivel de criticidad no afecta de manera directa el *core* del servicio de agendamiento.
Prioritaria	Hace referencia a una solicitud hecha por el director de tecnología o a quien delegue, donde se ve afectado de manera directa el correcto funcionamiento de las herramientas o servicios definidos en el alcance. Estas fallas pueden afectar directamente el *core* del servicio de agendamiento.
Programada	Hace referencia a una solicitud hecha por el director de tecnología o a quien delegue, donde el acceso a los servicios o herramientas no se ve afectado, pero es necesario programar alguna actividad de ajustes, cambios, parametrizaciones o actualizaciones.

Toda incidencia o solicitud de soporte será evaluada y atendida de acuerdo con el medio de acceso y forma de asistencia, y al tiempo de respuesta, según la siguiente clasificación:

Tabla 2
Nivel de atención para cada tipo de incidencia

Nivel de atención para cada tipo de incidencia		
Tipo	Medio de acceso y forma de asistencia	Tiempo de respuesta
Normal	Telefónico: 5 x 8 a través de celular 315 5008002 Vía *e-mail*: al correo soporte@quanther.com WhatsApp: 315 5008002 Gestión remota	De 2 - 8 horas
Prioritaria	Telefónico: 7 x 8 celular 315 5008002 Vía *e-mail*: al correo soporte@quanther.com WhatsApp: 315 5008002 Presencial de acuerdo con la disponibilidad del personal técnico.	De 1 - 3 horas
Programada	Telefónico: 7 x 8 Celular 315 5008002 Vía *e-mail*: al correo peter@quanther.com WhatsApp: 315 5008002 Presencial de acuerdo con la disponibilidad del personal técnico. Se requiere previa programación de la ventana de mantenimiento solicitada y la disponibilidad del personal de la IPS involucrada en el ajuste o cambio a realizar.	De 6 - 24 horas

2. Mesa de servicio

Trabajar desde una oficina o desde casa requiere el uso de infraestructura tecnológica y servicios de TI. Los sistemas, dispositivos y servidores permiten a los operarios realizar sus funciones; cuando uno de los servicios o la infraestructura falla, o impide realizar las labores (a esto se denomina incidente), es la mesa de servicio la que interviene para realizar las actividades correctivas y las operaciones preventivas, con el fin de garantizar que no vuelva a ocurrir. Las mesas de servicio ofrecen una ruta claramente definida para que los usuarios al momento de informar solicitudes, consultas y problemas se puedan identificar, clasificar, apropiar y procesar.

Es preciso que la mesa de servicio sea un único punto de contacto entre el usuario, el equipo de TI y los funcionarios de la empresa.

La mesa de servicio es una de las prácticas obligantes en la gestión de TI; hace falta estudiar y seguir un marco de trabajo para ofrecer una guía completa de una estrategia de establecimiento, por ejemplo, de un mes de servicio, la cual debe responder a los alcances de generación del valor de los servicios TI; sin embargo, en este punto se exponen solo algunos de los conceptos de las bibliotecas ITIL para la planeación, implementación, control y seguimiento de procesos, orientados a mejorar el área de mesa de servicio y restablecer la satisfacción del cliente que solicita servicios de soporte, una entidad que en principio puede ser unipersonal. Se propone la aplicación y estructuración de actividades, políticas y roles de los procesos.

Según ITIL para establecer una mesa de servicio, independientemente del tamaño de la empresa, así sea una empresa grande, mediana, pequeña, unipersonal o un trabajador autónomo se deben implementar los pasos que se ilustran en la siguiente figura:

Figura 1
Pasos para establecer la mesa de ayuda



2.1. Diagnóstico y preparación

Antes de realizar la implementación de la mesa de servicio se debe proceder con un diagnóstico de la empresa, la cual puede ser o no unipersonal, o un trabajador autónomo. Dicho diagnóstico puede ser llevado a cabo mediante la realización de entrevistas y aplicación de encuestas de satisfacción a los clientes, en donde se determinen las causas con sus respectivos problemas a soportar. Las causas con los problemas más comunes que estas generan se encuentran en la siguiente tabla:

Tabla 3
Causas y problemas más comunes

Causa	Problemas generados
La recepción de los incidentes tiene varios canales de acceso, *e-mail*, WhatsApp, celular etc.	• El cliente se queja por la indisponibilidad del servicio. • El soporte que se brinda no es el adecuado. • El servicio postventa hace que el producto o servicio pierda valor. • Atención desordenada de incidentes.
Dificultad en la comunicación con el usuario final que reporta el problema.	
Olvido o descuido en comunicar los incidentes reportados.	

Desorden en el manejo del tiempo y falta de orden de prioridades de los incidentes reportados.	• Incidentes que no se resuelven o se resuelven inoportunamente.
No se capacita adecuadamente al nuevo personal y a los nuevos clientes. Se improvisa constantemente en la atención del incidente. Soporte de requerimientos que están fuera de la solicitud inicial o el alcance del sistema y esto dilata el tiempo asignado para dicho soporte.	• Sobrecarga operativa de soporte. • Insatisfacción del cliente a razón de soluciones temporales. • La imagen del proveedor TI se debilita frente al cliente. • El cliente se acostumbra a no seguir el conducto regular definido para resolver incidencias.
Reuniones no programadas.	• Pérdida de tiempo. • Las reuniones derivan en temas no enfocados o definidos.
Suspender trabajos que están marchando para asumir nuevos incidentes.	• Se crea una cultura de incumplimiento e irresponsabilidad por parte del proveedor de TI en el quehacer cotidiano de sus labores. • Se crea la costumbre de pedir disculpas por el incumplimiento. • Cliente insatisfecho y desmotivado.

2.2. Definición de la estructura de servicios

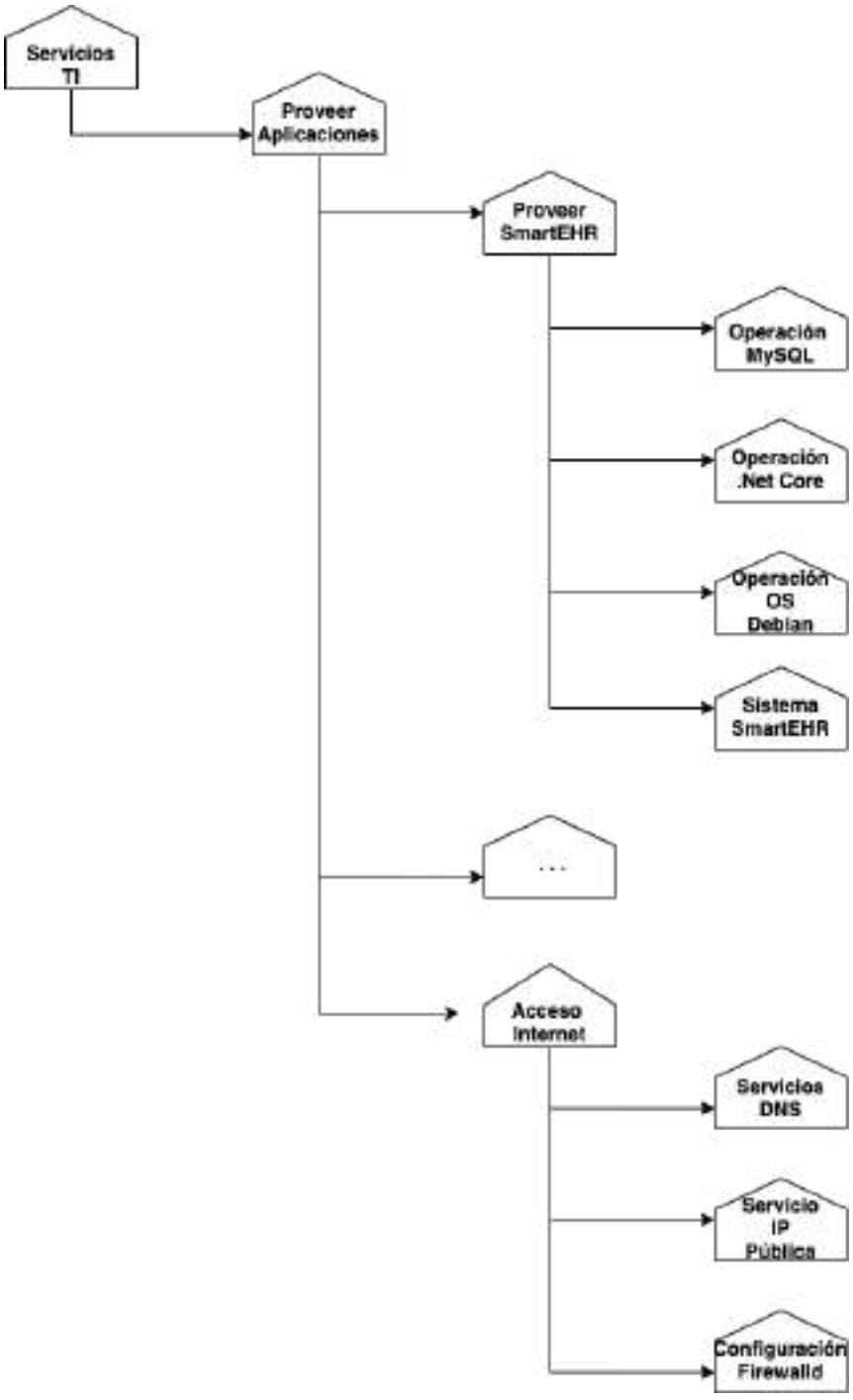
En este paso se garantiza la efectividad y eficacia en la provisión y el soporte de servicios, con el fin de generar valor para el cliente y el proveedor del servicio (Granizo, 2009).

ITIL recomienda definir inequívocamente los servicios que debe ofrecer la empresa a sus clientes como:

- SmartAppoinmet: en español, Sistema de citas y agendamiento.
- SmartEHR: en español, Sistema de historias clínicas.
- SmartPOS: en español, Sistema de facturación de ferreterías.
- SmartHPS: en español, Sistema de facturación de droguerías.
- Implementación de los productos.
- Capacitaciones.
- Soporte a usuarios.

Para cada uno de los servicios se debe especificar sus componentes, la relación de dependencia de uso y otros, por ejemplo, en la siguiente figura se muestra cómo para proveer ampliaciones se necesita que los sistemas tengan acceso a Internet y no cualquier servicio o tipo Internet, sino uno con unas particularidades específicas, tales como DNS, dirección IP pública, configuración de *firewalls* de seguridad, proveedores de envío de correo (SMTP), etc.

Figura 2
Estructura de servicios



2.3. Selección de roles y propietarios de roles

La identificación y asignación de los roles internos y externos necesarios para el correcto funcionamiento de la mesa de servicio se derivan directamente de las disciplinas ITIL.

Los roles internos están representados por el personal de la empresa encargada de brindar el servicio de soporte y asesoramiento a los roles externos, que son los que realizan las peticiones o solicitudes y son conocidos como clientes o usuarios.

En las tablas que están a continuación, se muestran los roles internos y externos para los casos de soporte que puede prestar una empresa, que puede o no ser unipersonal o un trabajador autónomo.

Tabla 4

Roles internos

Cargo	Propietario	Rol ITIL
Gerente de soporte	Nombre completo	Monitorización y control
Soporte Senior	Nombre completo	• Gestor de incidencias • Peticiones, problemas Nivel 2 • Centro de servicio al usuario
Soporte Junior	Nombre completo	• Gestor de incidencias • Peticiones • Problemas Nivel 1 • Centro de servicio al usuario
Área desarrollo	Nombre completo	• Solicitudes Nivel 4

Tabla 5

Roles externos

Propietario	Rol ITIL
Droguería + salud	• Cliente • Usuario
Dr. Manuel Parra	• Cliente • Usuario
Amazon Web Services	• Soporte Nivel 3 • Proveedor de infraestructura

En una organización grande estos roles pueden ser complejos. También puede incrementar esta complejidad el número de clientes externos que tenga la empresa, por ejemplo, si se desea implementar un servicio de gestión de incidentes se debe nombrar un gestor de problemas (IT Process Wiki, 2021).

2.4. Análisis de procesos existentes

ITIL recomienda elaborar una evaluación enfocada en los procesos existentes, para lo cual se recomienda aplicar instrumentos de medidas como encuestas, que permitan validar aspectos de la prestación del servicio de soporte, basados en los siguientes criterios:

- Gestión de capacidad el proceso.
- Control de calidad.
- Gestión de la información.
- Relación con el cliente.
- Nivel de satisfacción.
- Oportunidad en la prestación del servicio.

En caso tal, que como resultado de la aplicación del instrumento anterior se encuentren hallazgos similares o iguales a estos:

- Poca rapidez en la gestión de las incidencias, porque no cuenta con una bitácora automatizada, en la que se registre el estado de cada una. Los incidentes llegan por *e-mail*, WhatsApp o llamada telefónica, pero a veces se olvidan y no se resuelve el incidente.
- El cliente no sabe cómo están siendo gestionadas sus incidencias. En ocasiones tiene la impresión de que no se está haciendo nada para resolver su problema.
- No existe un nivel de prioridad para dar tratamiento al más urgente de los incidentes.
- No existe un control adecuado de tiempos de gestión de incidentes.
- No existe un proceso de control de calidad.
- No hay forma de documentar las incidencias que se convierten posteriormente en problemas.
- No es posible visualizar la carga laboral que se tiene en determinado momento.
- No se puede obtener un informe de gestión con incidentes de fechas pasadas.
- No se conoce el nivel de satisfacción del cliente respecto a sus solicitudes.
- Los usuarios reclaman por el mal servicio y no se tienen registros de las quejas.

2.5. Definición de estructura de procesos

Con el análisis de los procesos existentes y sus efectos en la empresa se puede identificar qué se debe estructurar para lograr su mejora. Por lo tanto, es necesario crear dos estructuras, una para los procesos internos de atención al usuario y otra para los acuerdos entre el proveedor del servicio – cliente, esto con el fin de controlar el área de soporte y el manejo de incidentes, de tal manera que estas labores no interrumpan las tareas de desarrollo de nuevos productos de *software*.

Un ejemplo de proceso interno es:

1. La gestión de incidentes inicia con la petición del usuario, quien ingresa a la plataforma de gestión y luego registra la solicitud que será evaluada por el soporte Junior o soporte Senior.

2. Según los acuerdos de tiempo de entrega se le asigna la prioridad al incidente, los cuales constan en los acuerdos entre el proveedor del servicio – cliente.

3. Se realizan las actividades que den respuesta al incidente.

4. Si los incidentes son comunes o repetitivos deben reposar en una base de conocimiento para ingresarlos al proceso de control de calidad, donde se encuesta el nivel de servicio con sus respectivos informes.

2.6. Definición de interfaces de ITIL

Las interfaces de procesos de incidentes son los canales de comunicación con el área de soporte, y pueden ser llevados a cabo telefónicamente, por correo electrónico, chats, plataformas web, entre otros; pero independiente de cuál sea el medio de entrada, las salidas serán comunes a todas las entradas, de modo que todo se reporte en un sitio centralizadamente. En la tabla que está a continuación, se presentan las entradas y salidas más comunes del proceso.

Tabla 6

Entradas y salidas más comunes del proceso

Entrada	Salidas
• **Formulario web de incidentes.** Los incidentes se activan cuando se registran las peticiones en este formulario. • **Correo electrónico.** Se debe proveer un correo electrónico para que el cliente pueda informar su solicitud. • **Llamada telefónica.** Proporcionar un número de contacto para que el cliente pueda manifestar su petición. • **Chat.** Establecer un canal de chat como WhatsApp o Telegram para que el cliente indique su petición.	• Informes de incidencias gestionadas. • Nivel de satisfacción. • Reporte de oportunidad de atención.

Una vez identificada la entrada es importante definir la clasificación del incidente como lo indica la escala básica enunciada en la tabla 7:

Tabla 7
Escala básica de incidentes

Escala básica de incidencias	
Normal	Hace referencia a una solicitud hecha por el director de tecnología o a quien delegue, donde no se ve comprometido o afectado de manera considerable el servicio para el cual se solicita el soporte. El nivel de criticidad no afecta de manera directa el *core* del servicio de agendamiento.
Prioritaria	Hace referencia a una solicitud hecha por el director de tecnología o a quien delegue, donde se ve afectado de manera directa el correcto funcionamiento de las herramientas o servicios definidos en el alcance. Estas fallas pueden afectar directamente el *core* del servicio de agendamiento.
Programada	Hace referencia a una solicitud hecha por el director de tecnología o a quien delegue, donde el acceso a los servicios o herramientas no se ve afectado, pero es necesario programar alguna actividad de ajustes, cambios, parametrizaciones o actualizaciones.

Los incidentes también se clasifican por tipo de solicitud:

- Falla general del sistema.
- No registra datos.
- Muestra información incompleta.
- Ausencia de información.
- Falla de conectividad.
- El sistema está lento.

La escala básica y los tipos de solicitud de incidencias permiten contar con reportes para identificar los aspectos que más fallas registran y proceder con la mejora del servicio.

2.7. Establecimiento de controles

En este paso se definen los controles y el tiempo de su aplicación en la mesa de servicio, con el fin realizar seguimiento al cumplimiento de los procesos internos de atención al usuario y a los acuerdos entre el proveedor del servicio – cliente, que permitan mejorar el servicio. Algunos ejemplos de controles son:

- Revisión mensual de los indicadores de gestión para identificar el tipo de servicio que presenta más fallas o no está correctamente gestionado.
- Revisión quincenal de los incidentes retrasados no gestionados para intervenir en cada caso y llegar a un acuerdo de nivel con el cliente.
- Revisión trimestral del tipo de falla que más reporta incidencias para establecer un plan de mejora sobre el servicio.

Para materializar esos controles se deben utilizar los indicadores de gestión, los cuales van a estar definidos por los incidentes resueltos y sin resolver, tal como se describe en la siguiente tabla:

Tabla 8
Indicadores de gestión

Indicador	Descripción

Número de incidentes solicitados.	Cantidad de registros de solicitudes de incidentes.
Número de incidentes en ejecución.	Cantidad de incidentes que no se han resuelto aún.
Número de incidentes retrasados no cerrados.	Cantidad de incidentes que se han pasado del tiempo o nivel de servicio acordado con el cliente y aún no se han resuelto.
Tipo medio de incidencia por tipo incidencia.	Cantidad de tiempo promedio empleado en resolver las incidencias reportadas durante determinado rango de horas, días, meses.
Tipo medio de incidencia por clasificación de la falla.	Cantidad de tiempo promedio empleado en resolver incidencias para cada tipo de clasificación.
Número de incidentes cerrados fuera de tiempo por clasificación.	Cantidad de incidentes que se resolvieron por fuera del tiempo acordado con el cliente.

Los anteriores indicadores no son los únicos, es posible que con la implementación y ejecución de los servicios se evidencian otros, según la dinámica del servicio.

2.8. Diseño de los procesos

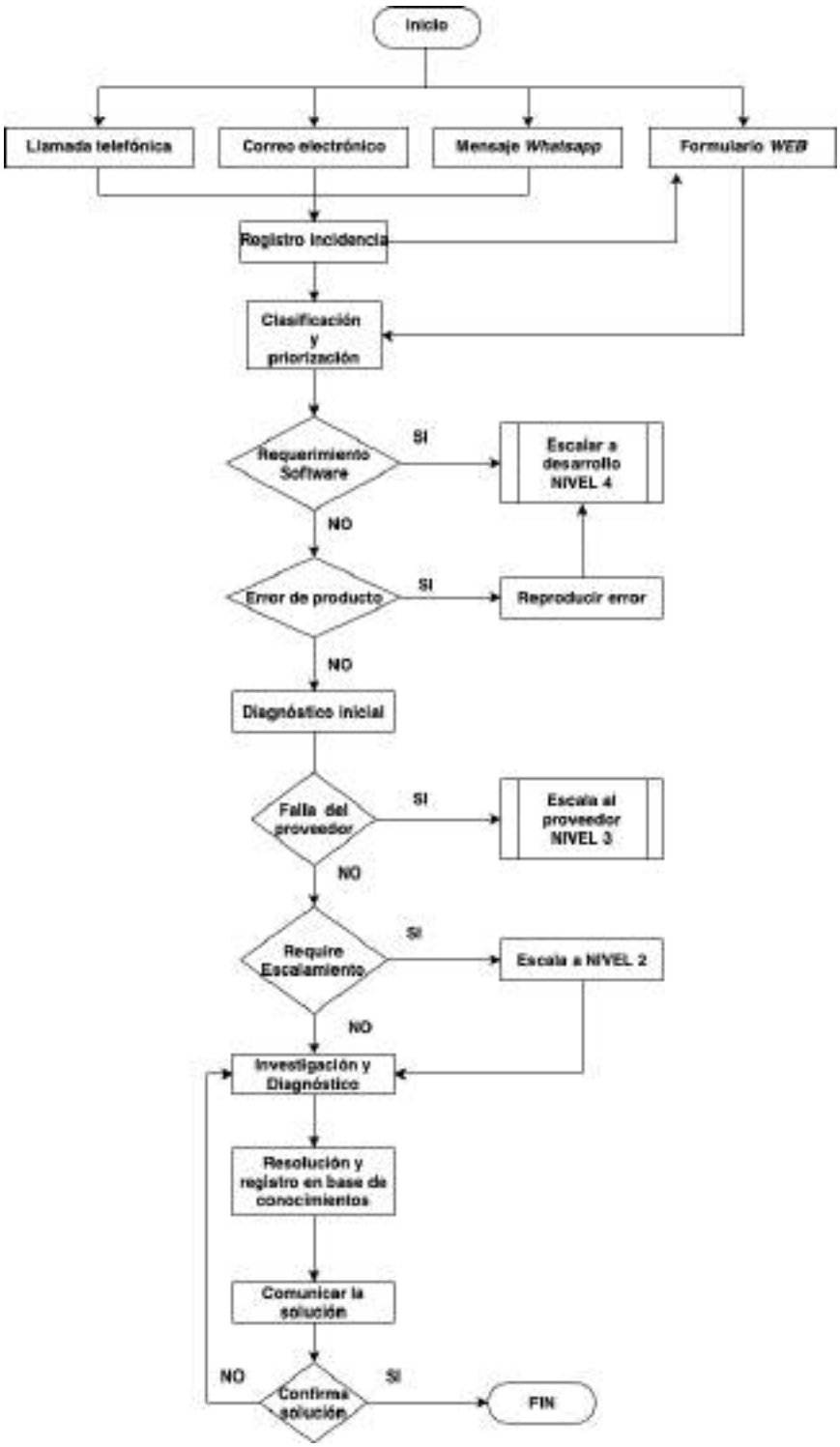
Generalmente, en el diseño de procesos de gestión de incidentes deben existir al menos cuatro niveles:

- **Nivel 1.** Se da comienzo con una asistencia inicial, este nivel es el responsable de las incidencias básicas del cliente, es el primer contacto con el usuario; proporciona asistencia física y/o remota a reportes relacionados con el uso de los dispositivos de cómputo, impresión, conectividad, entre otros. Atiende el requerimiento y brinda solución a las fallas básicas; un ejemplo de falla básica: la falta de capacitación del usuario en el manejo de la herramienta bien sea de *hadware o software*.
- **Nivel 2.** Si no se logra brindar una solución en el nivel anterior se escala el incidente a este nivel para que el técnico de desarrollo del sistema atienda este requerimiento.
- **Nivel 3.** Cuando el incidente se escala a este nivel es porque la falla tiene su origen en el servicio o en algún proveedor de servicios en la nube, y solo ellos pueden dar solución a esta situación. También se escala a terceros los daños que requieren de personal especializado en soporte técnico y ellos serán los responsables.
- **Nivel 4.** Cuando la solicitud no es una falla y atiende a un requerimiento que el sistema actual no tiene, indica que se debe intervenir con el desarrollo de nueva funcionalidad o modificación de la función actual.

La siguiente figura muestra un ejemplo del modelo de proceso que debe ser documentado y ampliado para los subprocesos de escalado a nivel 3 y 4.

Figura 3

Modelo del proceso de mesa de servicio



A medida que se van implementando los anteriores procesos se pueden ir modificando las acciones y modelos, de tal forma que se adapten cada vez más a la satisfacción de las necesidades del cliente.

2.9. Selección e implementación de sistemas

En el mercado existen muchas herramientas de pago y gratuitas, las cuales pueden ser seleccionadas según la necesidad. En la siguiente figura se muestra algunas de las herramientas de pago con sus diferentes características:

Figura 4

Servidores desk software de pago



Nota. Tomado de Bonillaware (2021).
https://www.bonillaware.com/wp-content/uploads/comparativa_helpdesk.png

En la siguiente tabla se muestran las herramientas más comunes bajo la licencia Open Source, es decir, aplicaciones de código abierto para su modificación y, las cuales en su mayoría, no requieren de pago para su uso:

Tabla 8

Servidores desk software Open Source y no pago

Proyecto	**Web**
GLPI (Gestionnaire Libre de Parc Informatique)	https://glpi-project.org/es/
osTicket	https://osticket.com/
((OTRS)) Community Edition	https://otrs.com/
NetBox	https://netbox.readthedocs.io/en/stable/

Al igual que en el listado de Servidores *desk software* de pago, los Servidores desk *software* Open Source no pago presentan características propias, las cuales pueden satisfacer diferentes necesidades de la empresa prestadora del servicio y a sus clientes.

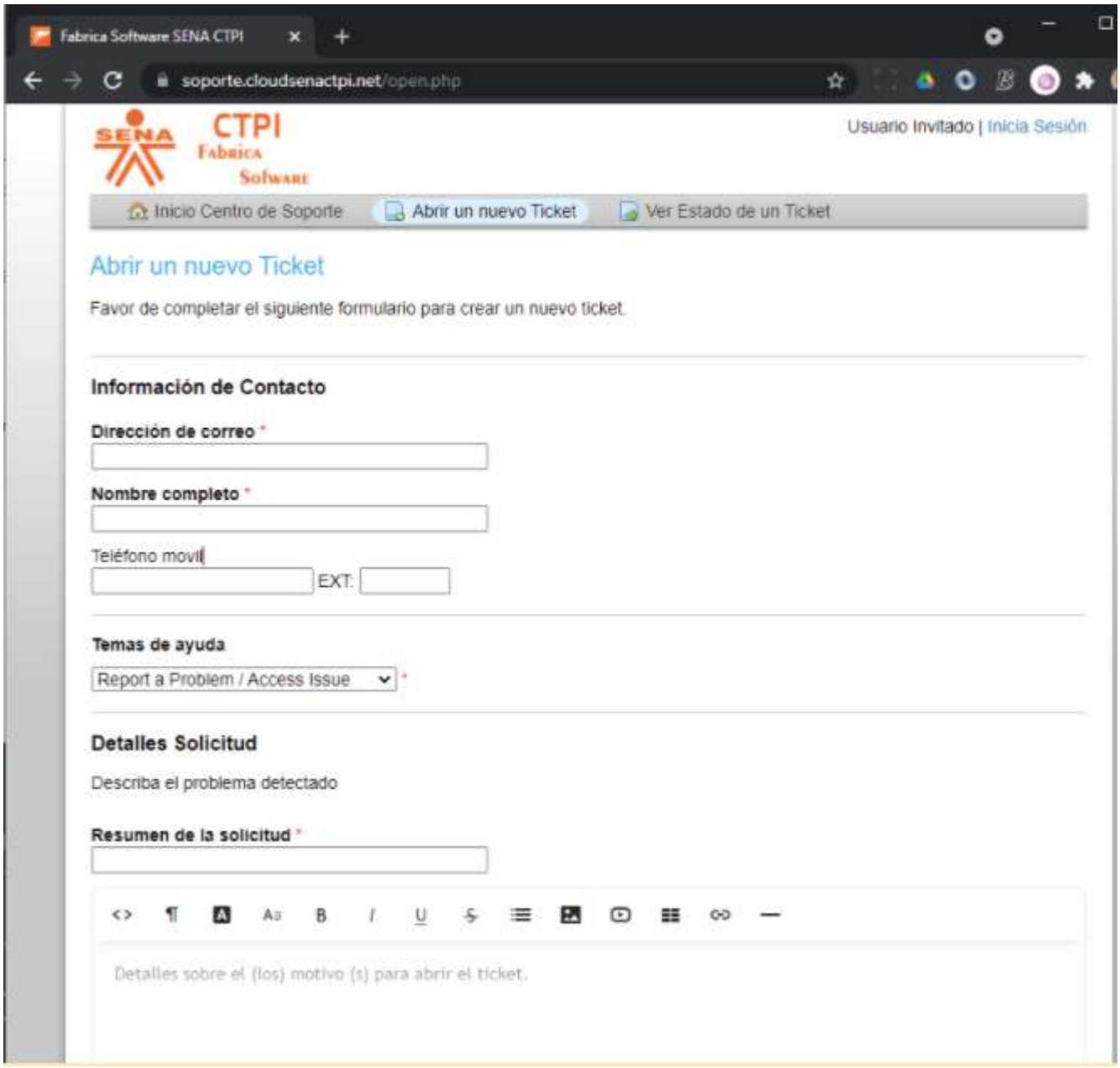
2.10. Implementación de procesos y capacitación

Una vez elegidas las herramientas se procede a la instalación y configuración, de manera que a los clientes se les permita solicitar el soporte. Se recomienda usar la herramienta osTicket, ya que el proveedor la ofrece gratuitamente en su página web.

Casi todos los proveedores de *hosting* ofrecen alguna herramienta Open Source y las de chat *online* para brindar el respectivo soporte.

En la siguiente figura se ofrece un ejemplo de una de las herramientas de soporte Open Source, llamada osTicket en algunos de sus componentes de interfaz gráfica:

Figura 5
Formulario para generar una incidencia



Nota. Tomado de https://soporte.cloudsenactpi.net

La herramienta seleccionada cuenta con los siguientes componentes y que se muestran en la figura que aparece a continuación.

- Base de datos de conocimiento.
- Control de incidencias o *tickets*
- Control de tareas y actividades.

Figura 6

Algunas funcionalidades del sistema

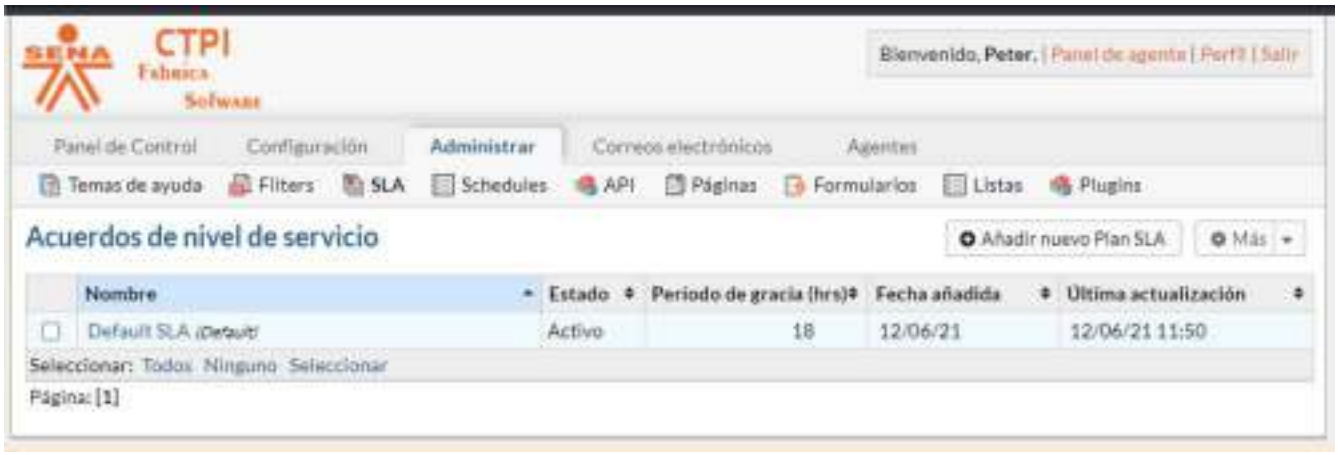


Nota. Tomado de https://soporte.cloudsenactpi.net

Como se puede observar en la siguiente figura el sistema seleccionado maneja acuerdos de nivel de servicio (SLA), configurables.

Figura 7

Gestión de acuerdos de nivel de servicio



Nota. Tomado de https://soporte.cloudsenactpi.net

Después de instalar e implementar la herramienta es necesario proceder con la capacitación de su equipo colaborador y de sus clientes para que se registren y se haga el seguimiento de las solicitudes.

Finalmente, debido a que existen muchas herramientas para todo tipo de presupuesto que permiten implementar buenas prácticas de gestión de TI de usuarios, y existe una base de recomendaciones amplia para establecer estos procesos de implementación, es posible concluir que solo por la falta de compromiso con el progreso, una entidad dejaría de establecer estrategias que faciliten la gobernanza de TI.

C. Actividades didácticas (opcionales si son sugeridas)

Descripción de la actividad didáctica	
Nombre de la actividad	N/A
Objetivo de la actividad	
Tipo de actividad sugerida	
Archivo de la actividad (Anexo donde se describe la actividad propuesta)	

d. Material complementario

Tema	Referencia APA del material	Tipo de material (video, capítulo de libro, artículo, otro)	Enlace del recurso o archivo del documento o material
Definición de estructura de procesos y de interfaces de ITIL	Baud, J. (2016). *ITIL® V3. Entender el enfoque y adoptar las buenas prácticas*. Ediciones ENI.	Libro	
Establecimiento de controles	Baud, J. (2020). ITIL® 4 *Entender el enfoque y adoptar las buenas prácticas.* Ediciones ENI.	Libro	
Implementación de procesos y capacitación	Setiawan, A. (2019). *Tutoriall instalation OSTicket on Windows10 with XAMPP* [video] YouTube. https://www.youtube.com/watch?v=KCz1ZKGOp28	Video	https://www.youtube.com/watch?v=KCz1ZKGOp28
Mesa de servicio	IT Process Wiki. (2021). *El Wiki de ITIL® | IT Process Maps.* https://wiki.es.it-processmaps.com/index.php/Portada	Wiki	https://wiki.es.it-processmaps.com/index.php/Portada

e. Glosario

Término	Significado
Gobierno de TI	Capacidad de controlar los aspectos inherentes a la gestión de tecnologías de la información.
Incidente	Falla o solicitud de nivel técnico sobre un sistema o servicios de infraestructura o aplicación.
Outsorsing	Hace referencia al proceso de subcontratación o externalización de algún proceso o parte del proceso empresarial, de tal forma que se transfieren los recursos y responsabilidades referentes al cumplimiento de ciertas tareas a una sociedad o persona externa.
Sistemas legados	Sistemas de información poco actualizados o que emplean tecnologías que ya no están en constante actualización y son susceptibles a ser reemplazados.

f. Referencias bibliográficas

- Bonillaware. (2021). *Helpdesk: 7 herramientas de gestión de atención al cliente.* Bonillaware. https://www.bonillaware.com/7-herramientas-de-helpdesk

- Granizo, I. (2009). *Diseño e implementación de un modelo de simulación para el Gobierno de las TI basado en ITIL v3, MOF y CMMI for services.* Universidad Carlos III de Madrid.

- IT Process Wiki. (2021). *El Wiki de ITIL® | IT Process Maps.* https://wiki.es.it-processmaps.com/index.php/Portada

g. Control del documento

	Nombre	**Cargo**	**Dependencia** *(Para el SENA indicar Regional y Centro de formación)*	**Fecha**
Autor (es)	Peter Emerson Pinchao	Experto temático	Regional Cauca - Centro de Teleinformática y Producción Industrial	Diciembre de 2022
	Gustavo Adolfo Rodriguez	Experto temático	Regional Cauca - Centro de Comercio y Servicios	Diciembre de 2022

h. Control de cambios

(Diligenciar únicamente si realiza ajustes a la Unidad temática)

	Nombre	Cargo	Dependencia	Fecha	Razón del cambio
Autor (es)					

PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Desarrollo de aplicaciones informáticas aplicando Inteligencia Artificial
- Código del Programa de Formación:
- Nombre del Proyecto: Construcción de software de integración aplicando inteligencia artificial y tecnologías orientadas a servicios
- Fase del Proyecto : Análisis
- Actividad de Proyecto: Determinar las especificaciones funcionales del sistema de información
- Competencia: 220501092 - Establecer requisitos de la solución de software de acuerdo con estándares y procedimiento técnico
- Resultados de Aprendizaje Alcanzar:
    - 220501092-1 Recolectar la información de acuerdo con las necesidades del cliente.
    - 220501092-2 Documentar el informe de requisitos de acuerdo con la identificación de las necesidades del cliente.
    - 220501092-3 Validar el informe de los requisitos del software con el cliente, aplicando técnicas y protocolos definidos por la organización.
- Duración de la Guía

2. PRESENTACIÓN

La presente guía de aprendizaje orienta al desarrollo del Establecimiento de requisitos para la solución del software", en ella, el aprendiz abordará temáticas que le permitirán de acuerdo con una problemática, poder identificar los requisitos necesarios para el desarrollo de sistemas de información, teniendo en cuenta los requisitos del cliente y la aplicación de buenas prácticas de calidad en proceso de desarrollo de software; adicionalmente, la actividad de proyecto en su visión integradora.

La Recolección de Requerimientos cumple un papel primordial en el proceso de desarrollo de software: la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, el comportamiento del sistema; de esta manera, se pretenden minimizar los problemas relacionados con el desarrollo de sistemas.

Permanece atento, y ¡APROVECHA ESTA OPORTUNIDAD DE APRENDIZAJE!

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

3.1 Actividad de Aprendizaje AA1. Identificar conceptos básicos de Teoría General de Sistemas y Enfoque Sistémico.

Esta actividad se centra en la identificación de fuentes de requisitos
Duración: 12 horas

Materiales de formación a consultar: para el desarrollo de esta actividad es importante la lectura y análisis del material de formación: **"Material de apoyo formativo 1: Ingeniería de Requisitos."**

Evidencias: a continuación, se describen las acciones y las correspondientes evidencias que conforma la actividad de aprendizaje**:**

- **220501092_01 – AA1_EV01 infografía sobre Teoría General de Sistemas requisitos**

- Realizar un mapa mental con los conceptos sobre ingeniería de requisitos vistos en el **Material de apoyo formativo 1: Ingeniería de Requisitos**
- Utilizar una herramienta Tic para la realización del mapa mental.
- El mapa debe tener las principales características de los mapas mentales.

Lineamientos generales para la entrega de la evidencia:

- **Productos para entregar:** documento informe técnicas de elicitación.
- **Extensión:** Libre.
- **Formato:** PDF o Word.

- **220501092_01 – AA1_EV01 Listado**: Fuentes de requisitos

- Conteste las preguntas planteadas en el foro temático con referencia al tema de identificación de fuentes de requisitos y de la opinión de las respuestas de por lo menos 3 de sus compañeros.
- Responda la pregunta: Cuáles son los tipos de fuentes y de un ejemplo de cada una, ¿ en qué caso uso cada tipo de fuente?
- Se debe responder de forma concisa donde el aprendiz de la opinión con referencia al tema.
- El aprendiz debe dar su opinión a la respuesta de por lo menos 3 compañeros.
- Se deben cumplir con normas ortográficas en las respuestas realizadas.

Lineamientos generales para la entrega de la evidencia:

- **Productos para entregar:** argumentación del foro.
- **Extensión:** Libre.
- Para hacer el envío de la evidencia remítase al área de la actividad correspondien GFPI-F-135 V01 espacio: **infografía**: Fuentes de requisitos**.AA1_EV02**.

Tomando como referencia la planeación pedagógica y las orientaciones para elaborar guías de aprendizaje citado en la guía de desarrollo curricular.

3.2 Actividad de Aprendizaje AA2. Elaborar instrumentos de recolección de datos, de acuerdo con técnicas que permitan el procesamiento de la información.

Esta actividad se centra en el estudio de los componentes de la fase de elicitación de requisitos, técnicas y herramientas de captura de requisitos.

Duración: 24 horas

Materiales de formación a consultar: para el desarrollo de esta actividad es importante la lectura y análisis del material de formación: "**Material de Apoyo formativo : Fase de Elicitación de requisitos**".

Evidencias: a continuación, se describen las acciones y las correspondientes evidencias que conforma la actividad de aprendizaje**:**

- **AP01- 220501092_01- AA2_EV01 Diseño del instrumento de recolección de información.**

Seleccione una de las técnicas utilizadas para la elicitación de requisitos y realice el diseño de un instrumento de recolección de información, teniendo en cuenta las características del software a construir.
Elementos a tener en cuenta en el documento de requisitos:

- Se deben seguir las normas básicas de presentación de un documento escrito, es decir el documento debe tener como mínimo una portada, introducción, objetivo, preguntas claras y alcance

Lineamientos generales para la entrega de la evidencia:

- **Productos para entregar:**documento de diseño.
- **Extensión:** Libre.
- **Formato:** PDF o Word.
- Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Diseño del instrumento de recolección de información.AA2_EV01**.

- **220501092_01 – AA2_EV02 Formulación de requisitos del proyecto de Software.**

Teniendo en cuenta las características del Software a construir seleccione una de las herramientas para captura de requisitos y defina sus funcionalidades.

Elementos a tener en cuenta en el documento de requisitos:

- Se deben seguir las normas básicas de presentación de un documento escrito, es decir el documento debe tener como mínimo una portada, introducción, objetivo, preguntas claras y alcance
- Seleccione una de las herramientas para captura de requisitos mencionada en el componente (Diagramas de casos de uso, Historias de Usuario, Storyboard) para realizar la formulación de los requisitos.
- Utilice una de las herramientas o plantillas (starUml, Plantilla Historias de Usuario) que se describieron en el componente formativo.

Lineamientos generales para la entrega de la evidencia:

- **Productos para entregar:** documento de requisitos.

- **Extensión:** Libre.
- **Formato:** PDF o Word.
- Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Formulación de requisitos del proyecto de Software.AA2_EV02**.

❖ **220501092_01 – AA2_EV03 Formulario de recolección de información.**

Teniendo en cuenta las características del *software* a construir elaborar un documento donde se pueda identificar las técnicas de elicitación para el levantamiento de requisitos utilizada.

Elementos a tener en cuenta en el documento de requisitos:

- Se deben seguir las normas básicas de presentación de un documento escrito, es decir el documento debe tener como mínimo una portada, introducción, las técnicas utilizadas para el levantamiento de requisitos, y el análisis arrojado después de aplicadas las técnicas.

- Respecto a la técnica deberá adjuntar el formato o la estructura de la técnica realizada; si es entrevista el guión utilizado; si es encuesta las preguntas realizadas, etc.
- Respecto al análisis se deberá realizar un informe de los resultados de la técnica aplicada.

Lineamientos generales para la entrega de la evidencia:

- **Productos para entregar:** documento con formulario
- **Extensión:** Libre.
- **Formato:** PDF o Word.
- Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Formulario de recolección de información Software.AA2_EV03**.

3.3 Actividad de Aprendizaje AA3. Determinar los requisitos funcionales y no funcionales del software de acuerdo con los requerimientos del cliente.

Esta actividad se centra en dos grandes bloques, (i) análisis de las diferentes técnicas de análisis de requisitos y (ii) estándares o modelos para el proceso de documentación de requisitos del *software*

Duración: 36 horas.

Materiales de formación a consultar: para el desarrollo de esta actividad es importante la lectura y análisis del material de formación: "**Componente formativo 4: la fase de análisis y especificación de requisitos**".

Evidencias: a continuación, se describen las acciones y las correspondientes evidencias que conforma la actividad de aprendizaje**:**

❖ **220501092_03 – AA3_EV01 Especificación de los requerimientos funcionales y no funcionales del software.**:

Construir el documento de requisitos que especifique los requisitos funcionales y no funcionales teniendo en cuenta las características del software a realizar.

Elementos para tener en cuenta en el documento técnico de validación

Se deben seguir las normas básicas de presentación de un documento escrito, es decir el documento debe tener como mínimo una portada, introducción, y la lista de:

.

- Requisitos funcionales
- Requisitos no funcionales

Lineamientos generales para la entrega de la evidencia:

- **Productos para entregar:** informe técnico.
- **Extensión:** Libre.
- **Formato:** PDF o Word.

Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Especificación de los requerimientos funcionales y no funcionales del software.AA4_EV01**.

❖ **220501092_03 – AA3_EV02 Documento con especificación de requerimientos.**:

Tomando de referencia lo realizado en la evidencia **AA4_EV01**, **Especificación de los requerimientos funcionales y no funcionales del software,** construir el documento de requisitos que tenga una estructura ordenada y fácil de leer donde por cada requisito referencia se establezcan descripciones claras, usuarios afectados, propósito de cada requisito y criterios de aceptación.

.

Elementos a tener en cuenta en el documento técnico de validación

- Se deben seguir las normas básicas de presentación de un documento escrito, es decir el documento debe tener como mínimo una portada, introducción, alcance, lista de requerimientos y versión del documento. Los requerimientos serán redactados usando el modelo IEEE830 y también el modelo de descripción de requisitos por medio de historias de usuario.
- Respecto a lista de requerimientos el aprendiz deberá agregar una sección donde se describa cada requisito usando los siguientes elementos del estándar IEEE830
  - Perspectiva del producto
  - Funciones del producto
  - Características de los usuarios
  - Restricciones
  - Requisitos funcionales (formato de casos de uso)
  - Requisitos no funcionales
- Respecto a la lista de requerimientos el aprendiz deberá agregar una sección donde se describa cada requisito usando la estructura de historias de usuario con los siguientes elementos por historia:
  - Número de historia (priorizada).
  - Nombre de la historia.
  - Usuario.
  - Puntos estimados de esfuerzo.
  - Descripción de la historia de usuario.
  - Observaciones.
  - Criterios de aceptación.

Lineamientos generales para la entrega de la evidencia:

- **Productos para entregar:** informe técnico.
- **Extensión:** Libre.
- **Formato:** PDF o Word.

Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Documento con especificación de requerimientos.AA3_EV02**.

3.4 Actividad de Aprendizaje AA5. Aplicar técnicas de validación de requisitos del software.

En esta actividad de aprendizaje se abordan las técnicas de validación de requisitos: revisiones, los prototipos y los casos de prueba.
Duración: 24 horas

Materiales de formación a consultar: para el desarrollo de esta actividad es importante la lectura y análisis del material de formación: "**Componente formativo 5: la fase de validación de requisitos**".

Evidencias: a continuación, se describen las acciones y las correspondientes evidencias que conforma la actividad de aprendizaje**:**

- **220501092_04 - AA5_EV01 Uso de herramientas de gestión de requisitos**: (se cambia por la del cuestionario que está en la planeación)

A partir de la revisión de las diferentes herramientas tecnológicas disponibles para la gestión de requisitos, seleccionar una para configurar los requerimientos del proyecto seleccionado.

Lineamientos generales para la entrega de la evidencia:

- **Productos para entregar:** documento informe de fuentes.
- **Extensión:** Libre.
- **Formato:** PDF o Word.
- Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **Uso de herramientas de gestión de requisitos. AA5_EV01**.

- **220501092_04 - AA5_EV02 Informe de evaluación de los requerimientos**:

Teniendo en cuenta las características de las técnicas de validación de requisitos abordadas en la actividad de aprendizaje, construir los prototipos del sistema y los casos de prueba asociados a cada interfaz desarrollada.

Elementos a tener en cuenta en el documento técnico de validación

- Se deben seguir las normas básicas de presentación de un documento escrito, es decir el documento debe tener como mínimo una portada, introducción, alcance, lista de requerimientos y versión del documento.
- Deberá seleccionar 4 historias de usuario o requisitos del proyecto seleccionado para representarlos por medio de prototipos y realizar sus respectivos casos de prueba.
- Para la construcción de prototipos deberá usar una de las herramientas propuestas en el componente formativo y exportar dichos elementos desarrollados como imágenes para ser incluidos dentro del documento.
- Por cada interfaz desarrollada deberá construir un caso de prueba el cual debe tener los siguientes elementos.
    - Objetivo del caso de prueba.
    - Identificador.
    - Nombre del requerimiento o historia de usuario asociada.
    - Precondiciones.
    - Lista de pasos con los resultados esperados.

Lineamientos generales para la entrega de la evidencia:

- **Productos para entregar:** informe técnico de validación.
- **Extensión:** Libre.
- **Formato:** PDF o Word.

Para hacer el envío de la evidencia remítase al área de la actividad correspondiente y acceda al espacio: **informe de evaluación de los requerimientos. AA4_EV01**.

4. ACTIVIDADES DE EVALUACIÓN

Tome como referencia la técnica e instrumentos de evaluación citados en la guía de Desarrollo Curricular

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento: **220501092_01 - AA1- EV01 - Infografía sobre Teoría General de Sistemas.** **Evidencias de producto:** **220501092_01 - AA2 EVO2 - Listado de Fuentes de requisitos.**	**Selecciona las técnicas de recolección de acuerdo con las necesidades del cliente.**	**Lista de Chequeo**
Evidencia de desempeño: **220501092_01 - AA2 - EV01 - Diseño del instrumento de recolección de información.** **220501092_01 - AA2 - EV02 - Formulacion del proyecto de software.** **Evidencia de producto:** **2220501092_01 - AA2 - EV03 - Formulario de recoleccion de información.**	**Determina los instrumentos de recolección de información según las técnicas seleccionadas.** **Utiliza las técnicas de captura de requisitos de acuerdo con las fuentes identificadas.**	**Lista de chequeo**
Evidencias de Desempeño: **220501092_02- AA3 - EV01 - Especificación de los requerimientos funcionales y no funcionales del software.** **Evidencia de producto.**	**Elabora documento de especificación de requisitos, según necesidades del cliente, metodología, técnicas y estándares de especificación y características técnicas del proyecto.**	**Lista de chequeo**

220501092_02 - AA3 - EV02 - Documento con especificación de requerimientos.	**Administra los cambios en los requisitos de acuerdo con las estrategias de control de versiones.**	
Evidencias de Conocimiento: **220501092_03 - AA4 - EV01 -Cuestionario Técnico** **Evidencia de producto.** **220501092_03 - AA4 - EV02 - Informe de evaluación de los requerimientos.**	**Valida el documento de especificación de requisitos, según las necesidades del cliente y las técnicas de validación.** **Ajusta el documento de especificación de requisitos, según los hallazgos encontrados en la validación.**	**Lista de chequeo**

5. GLOSARIO DE TÉRMINOS

***Hardware*:** corresponde a todas las partes físicas y tangibles de una computadora: sus componentes eléctricos, electrónicos, electromecánicos y mecánicos, sus cables, gabinetes o cajas, periféricos de todo tipo y cualquier otro elemento físico involucrado
Metodología: ciencia que consta de métodos y técnicas, que se aplican sistemáticamente durante un proceso de investigación o para solucionar una problemática.
Requerimientos: es una descripción completa del comportamiento del sistema que se va a desarrollar. Incluye un conjunto de casos de uso que describe todas las interacciones que tendrán los usuarios con el software.
Requisitos de Usuarios: Necesidades que los usuarios expresan verbalmente
Requisitos del Sistema: Son los componentes que el sistema debe tener para realizar determinadas tareas
Requisitos Funcionales: Servicios que el sistema debe proporcionar al finalizar el sistema
Stakeholders: interesados o participantes en un proyecto.
***Software*:** Soporte lógico, programas, parte no mecánica de un sistema. Serie de instrucciones necesarias para ejecutar diversas aplicaciones y tareas.
Teoría General de sistemas: es un esfuerzo de estudio interdisciplinario que trata de encontrar las propiedades comunes a entidades, los sistemas, que se presentan en todos los niveles de la realidad, pero que son objetivo tradicionalmente de disciplinas académicas diferentes**.**

6. REFERENTES BIBLIOGRÁFICOS

Pressman, R. (2010). Ingeniería del software, un enfoque práctico (Séptima edición). Bogotá: McGraw-Hill.

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)				

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO

PROGRAMA DE FORMACIÓN	Tecnólogo Desarrollo de Soluciones Informáticas con Inteligencia Artificial

COMPETENCIA	XXX	XXX	XXX

NÚMERO DEL COMPONENTE FORMATIVO	CF01.
NOMBRE DEL COMPONENTE FORMATIVO	CARACTERIZACION DE PROCESOS
BREVE DESCRIPCIÓN	La definición de procesos ayuda al entendimiento del funcionamiento de una organización el análisis, moldeamiento, diagramación y mejora continua son actividades esenciales en la estructura de una empresa que quiere aplicar desde una orden lógico las actividades diarias de una empresa.
PALABRAS CLAVE	Procesos, sistemas, teoría general de sistemas,diagrama.

ÁREA OCUPACIONAL	9 - PROCESAMIENTO, FABRICACIÓN Y ENSAMBLE
IDIOMA	Español

A. **TABLA DE CONTENIDOS:**

Introducción
1. Teoría General de Sistemas.
2. Enfoque de sistemas.
3. Definición de los sistemas.
3.1 Tipos de sistemas.
3.1.1 Sistema, Supersistema y Subsistema.
3.1.2 Reales físicos, ideales y abstracciones.
3.1.3 De la Naturaleza, artificiales y compuestos.
3.1.4 Cerrados y abiertos.
3.2 Sistemas de Información.
3.2.1 Datos e Información.
4 Análisis de los procesos a nivel de negocio.
4.1 Identificar los procesos.
4.2 Determinar el equipo de trabajo.
4.3 Generar los Diagramas de los procesos de negocio.
4.3.1 Diagrama de procesos.
4.3.2 Elementos de un diagrama de procesos.
4.3.3 Un buen diagrama de procesos.
4.3.4 Dividendos de los diagramas de procesos.
4.4 Definir la situación actual de cada proceso.
4.5 Puntos de mejora.
4.6 Nuevo proceso mejorado.
4.7 Caso de estudio.

INTRODUCCION

La caracterización de procesos es una técnica que permite a partir de la observación y el análisis de funciones de una organización el poder determinar, definir incluso en caso de ser necesario abstraer los pasos que de manera lógico - secuencial se deben llevar a acabo para la realización de alguna actividad, esta caracterización permite a las organizaciones generar mecanismos de control, vigilancia y seguimiento unidos a una gestión documental que pretenden principalmente lograr la optimización en la utilización de los recursos tanto materiales como no materiales para ordenar la toma decisiones, delegar responsabilidades, facilitar las dinámicas de capacitación, formar al personal que interviene de manera periódica generando un idóneo ambiente de trabajo que al final se traduce en la la satisfacción de sus usuarios y para la generación de valor de la organización.

1. Teoría general de sistemas

Es a un campo de la ciencia que pretende examinar las propiedades que definen a los sistemas, es decir, categorías formadas por partes interrelacionadas que llevan al cumplimiento de una acción u objetivo.

Desde el punto de vista histórico las primeras contribuciones a este campo y su definición vienen de la teoría general de sistemas de Ludwig von Bertalanffy**.** Este modelo ha perdurado como herramienta para la aplicación del método científico y sigue siendo utilizado como apoyo fundamental en el análisis de sistemas, organizaciones incluso para grupos humanos entre otros usos.

Desde la biología Karl Ludwig von Bertalanffy (1968) “propone la teoría general de sistemas como un instrumento amplio que se aplica y es común su uso para la resolución de problemas hoy en día y en todas las áreas del saber”.

Esta teoría se constituyó como un nuevo aporte dentro del conjunto de paradigmas científicos para el abordaje de problemas que se basa en la correspondencia entre los elementos que conforman los sistemas. Antes de esta propuesta se pensaba que los sistemas desde el punto de vista de su conformación global se asemejaban a la sumatoria de de sus elementos integrantes, y que podían ser entendidos disgregan-dolos para realizar un estudio individual de cada componente; el pensamiento de Bertalanffy presenta objeciones a tales creencias.

Desde sus inicios, la teoría general de sistemas se ha utilizado en diferentes áreas de la ciencia, la ingeniería, las matemáticas, la biología, a la política, las ciencias sociales, a las ciencias de la computación, a la economía, y a otras ciencias exactas y sociales, primordialmente en el marco del análisis de las correlaciones que se presentan en sus componentes y su interoperatividad.

2. Enfoque de sistemas

Un puntal básico unido a los respectivos análisis (Simulación, Teoría de comportamientos, Teoría de colas, Teoria de juegos) que se pueden aplicar para la resolución de problemas usando el método científico es el enfoque de sistemas; Centro Tic Junta de Andalucía(2021) “se trata de comprender el funcionamiento de una organización u objeto de estudio desde una perspectiva holística e integradora, en donde lo importante son las relaciones entre los componentes. Se llama holismo al punto de vista que se interesa más por el todo que por la suma de las partes. El enfoque sistémico no concibe la posibilidad de explicar un elemento si no es precisamente en su relación con el todo. Metodológicamente, por tanto el enfoque sistémico es lo opuesto al individualismo metodológico, aunque esto no implique necesariamente que estén en contradicción”.

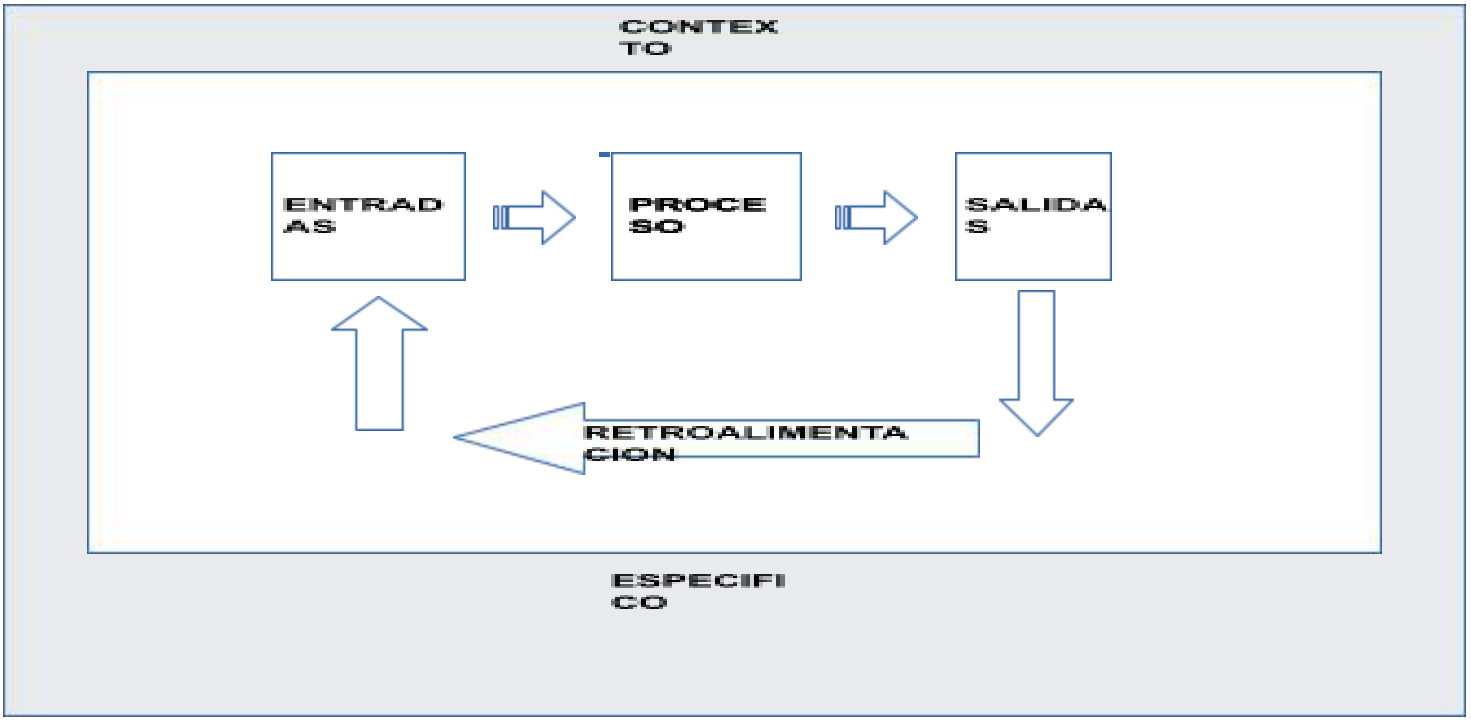
3. Definiendo los sistemas.

Para Ludwig von Berthalanffy(1968) “Sistema es un conjunto organizado de elementos que interactúan entre sí con estructura lógica o que son interdependientes, formando un todo complejo, identificable y distinto. Por los elementos de un sistema se entienden no solo sus conformación física sino las funciones que estos desempeñan. Algún conjunto de elementos de un sistema puede ser considerado un subsistema si mantienen una relación entre sí que los hace también un conjunto identificable y distinto”.

Tales sistemas se caracterizan por su capacidad de recepción de elementos los cuales son denominados como entradas estas entradas a su vez se pueden tipificar como tipos de energía, información a manera de mensajes, datos y señales también recursos físicos. Estas entradas se someten a un conjunto de actividades las cuales alimentan una serie de acciones que las transforman y como consecuencia se generan unos resultados o salidas; una situación muy interesante se presenta cuando una salida o parte de ella se convierte nuevamente en entrada cuando esto sucede se dice que existe retroalimentación. La retroalimentación ayuda a elevar el grado de perfeccionamiento en las respuestas y en el propio comportamiento del sistema logrando de manera inmediata altos niveles de control principio básico de la automatización de los mismos.

Así mismo cuando de un subsistema se conocen solo las entradas y las salidas pero no las actividades internas se dice que es una caja negra en ese caso se aconseja utilizar la observación y guiarse por el sentido común para proponer las actividades desde cero que resuelven el mismo problema.

Figura 1. Ejemplo de los elementos del sistema.



3.1 Clasificación de sistemas.

Bertalanffy y otros autores posteriores han definido distintas formas de clasificar a los sistemas en función de su conformación estructural y a sus funcionalidades**.** Estas son las definiciones más importantes.



3.1.1 Sistema, supersistema y subsistemas.

Los sistemas se pueden agrupar teniendo en cuenta su grado de complejidad se componen a su vez de distintos niveles de jerarquía en que pueden dividirse y que interactúan entre ellos intercambiando algún tipo de información, de modo que estos al final no son autónomos unos de otros.

Si se entiende por sistema como un conjunto de partes, podemos hablar de *subsistemas* para referirse a tales elementos; un ejemplo de ello es: la conformación de una familia como un sistema y cada miembro en ella es un *subsistema*. El *supersistema* es el contexto externo que rodea al sistema y en el que éste se encuentra inmerso; en los grupos o comunidades de humanos es identificable con la sociedad.

3.1.2 Reales físicos, ideales y abstracciones.

De acuerdo a su esencia los sistemas se pueden clasificar en reales físicos a su vez como producto del pensamiento en ideales y modelos. Los sistemas reales son aquellos que existen físicamente y que pueden ser objeto de observación y por tanto objetos de medición, mientras que los sistemas ideales son las representaciones simbólicas extraídas de la abstracción que produce el pensamiento y el lenguaje que se traducen en diseños y modelos. Los abstracciones y sus correlaciones pretenden representar características reales e ideales aquí es importante tener en cuenta el conjunto de requerimientos, solicitudes, la definición de limites con las cuales se pretende llegar a los objetivos finales.

3.1.3 De la Naturaleza, artificiales y mixtos.

Cuando un sistema esta inmerso o corresponde unicamente al funcionamiento de la naturaleza, como lo es un lago, un rio, la selva el cuerpo humano los planetas o las galaxias, entonces ahí que referirse a ellos como *sistemas naturalez*. Por el contrario, los sistemas artificiales son aquellos que emergen de la creación del hombre es decir como productos de la acción humana; dentro de este tipo de sistema podemos encontrar los construidos por el hombre aviones,trenes,maquinarias de transformación de las energías incluso de transformación de la información tangibles e intangibles. .

Los sistemas mixtos se dan por la combinación de elementos naturales y artificiales. Cualquier entorno de orden físico que sufre alteraciones dadas por el ser humano, como lo es los pueblos, las ciudades, los rios en donde se construyen represas, es considerado un sistema mixto; el grado de participación de elementos naturales y artificiales varía en cada caso concreto en proporciones diferentes.

3.1.4 Cerrados y abiertos.

Torres (2021) “El criterio principal que define a un sistema es el grado de interacción con el supersistema y otros sistemas. Los sistemas abiertos intercambian materia, energía y/o información con el entorno que los rodea, adaptándose a éstos e influyendo en él. En cambio, los sistemas cerrados se encuentran teóricamente aislados de las influencias ambientales; en la práctica se habla de sistemas cerrados cuando están altamente estructurados y la retroalimentación es mínima, puesto que ningún sistema es completamente independiente de su supersistema”.

3.2 Sistemas de Información.

Teniendo en cuenta los tipos de sistemas estudiados en donde se puede observar que este concepto es genérico en las diferentes áreas del saber vale la pena hacer énfasis dada la temática a la que esta orientado este componente se debe identificar claramente los sistemas de información gestionados por procesamiento electrónico de datos que entran a solucionar problemas de la vida real inmersos en la hola actual denominada como la tercera revolución industrial en donde convergen las últimas técnicas de procesamiento de datos (Analítica de datos, Big Data, Inteligencia Artificial, Minería de datos, Domótica entre otros) unido definitivamente al desarrollo de software en donde se recomienda de manera primordial identificarlas para analizar su posible inter-operabilidad como un camino expedito para aprovechar funcionalidades que se pueden reutilizar minimizando así muchos recursos y generando incluso oportunidades de negocio.

3.2.1 Datos e Información.

Estos términos datos e información tienden a confundirse pero a la luz de los sistema que estamos tratando en este apartado vale la pena definirlos.

La definición de dato es según Deusto(2019) ”*Un dato no es otra cosa que una representación simbólica de alguna situación o suceso, sin ningún sentido semántico, describiendo un hecho concreto. O lo que es lo mismo, sin transmitir mensaje ninguno, un ejemplo de dato podría ser una letra o un hecho.*”

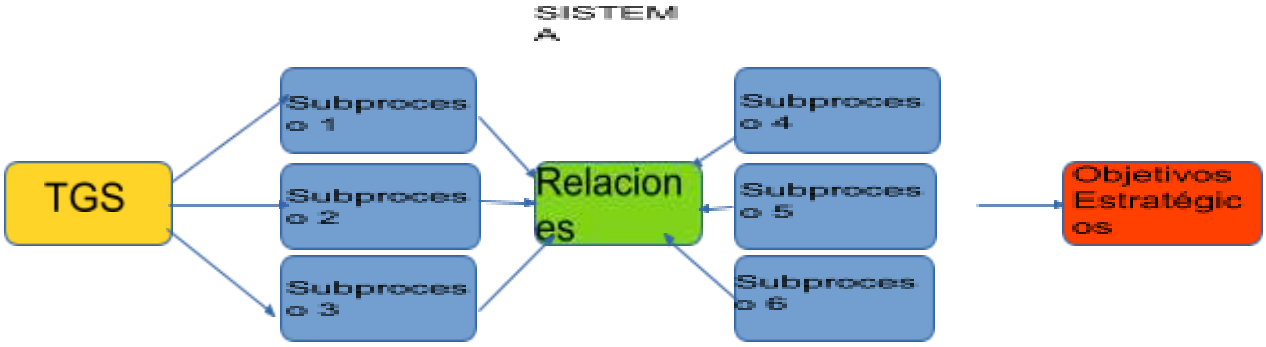
La definición de información es según Deusto(2019) “*La información se define como un conjunto de datos los cuales son adecuadamente procesados, para que de esta manera, puedan proveer un mensaje que contribuya a la toma de decisiones a la hora de resolver un problema o afrontar una situación cualquiera en la que se requiera de la toma de decisiones de cualquier tipo.*”



4 Análisis de los procesos a nivel de negocio.

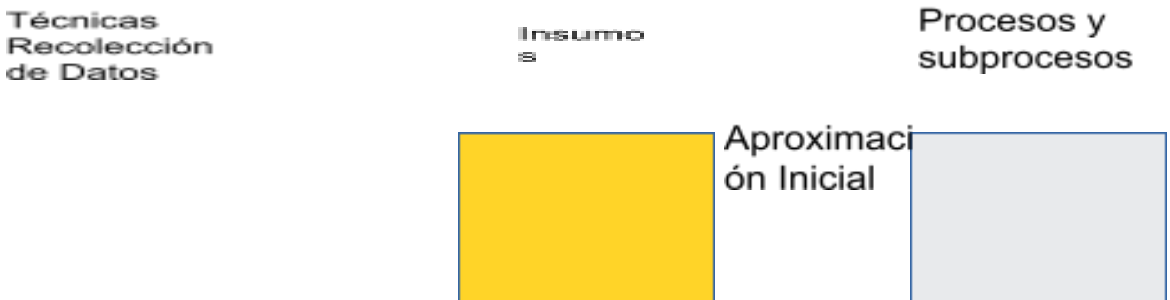
Con base en los conceptos de la teoría general de sistemas y partiendo de la visión del todo se debe comprender el funcionamiento real a partir de desestructurar ese todo en procesos, subprocesos, relaciones y

actores que intervienen con el objetivo de identificar,comprender, evaluar y resolver problemas que ayudan a las instituciones a mejorar dichos procesos, reformar las opciones de negocio y brindar a sus consumidores a experimentar una mejor calidad de productos o servicios. Por esto, es necesario que realicen de manera constante , frecuente y repetitiva un análisis de sus procesos de negocio desde la TGS donde visualicen y comprendan cada uno de los pormenores que transforman el negocio con el objetivo de mejorar sus operaciones y el servicio al cliente. Una de las técnicas que permite conseguir estos objetivos es el análisis de procesos de negocio que significa llegar al conocimiento detallado de cada labor que se hace abordando la operación central de valor hasta las que pueden considerarse como labores rama o periféricas no menos importantes. Ese es el nombre que se le da a la acción que realizan las organizaciones para revisar, documentar y entender sus procesos. Tiene como objetivo el desglose de cada uno de los componentes en cada proceso (entradas, salidas, procedimientos, controles, aplicaciones, datos, etc.) comprender sus interrelaciones para alcanzar los objetivos de una organización.



4.1 Identificar los procesos.

Para realizar una excelente identificación de procesos se recomienda realizar las siguientes actividades en dos escenarios en donde se puede partir antes de tener algún tipo de cercanía con la entidad como para después de obtener acceso con la organización, para antes de llegar a la organización se debe investigar sobre el contexto de negocio en el que se mueve la empresa por ejemplo determinar si se esta hablando de un banco o un empresa de energía o una empresa que presta servicios y en ese caso que tipo de servicios presta o por el contrario si es una empresa que fabrica productos que productos fabrica es decir que hace; conocer de ella también sus sedes o instalaciones ubicarse al máximo en la relevancia que ella tienen en el contexto que opera acudiendo a información que en primera medida es externa. Cuando exista cercanía, acceso o contacto con la entidad, planear entonces la recolección de información aplicando las técnicas que existen para ello como lo es el método de la observación, la elaboración de formatos de encuestas, el enfoque de tyrabajo (focus group) y las entrevistas; como resultado de estas técnicas se deben establecer claramente y como mínimo lo siguientes items: roles de los intervinientes, información de contactos con responsabilidades, misión , visión , objetivos estratégicos, estructura organizacional un organigrama de le entidad su distribución de dependencias u oficinas en donde se puede ubicar de manera ágil a los responsables que seguramente se convertirán en futuros miembros de los equipos de trabajo donde se presentara interacción (Sinergia laboral); con lo anterior dispuesto finalmente se puede dar el primer paso fundamental que corresponde a identificar, conocer, incluso ubicar los sitios en donde suceden en el mundo real aquellos procesos que necesitan ser mejorados, descritos, analizados o propuestos para generar mejoras.





4.2 Determinar el equipo de trabajo.

Con base en la información recolectada y a la obtención de los requerimientos que permitieron identificar los procesos que necesitan mejorarse, se procede a formar el equipo de trabajo que los identificará, analizará y documentará. Lo ideal es que los responsables que conocen a detalle los procesos conformen el equipo. Por lo tanto , el equipo debería estar integrado por el siguiente grupo humano:

- El líder del proyecto o el proceso elegido.
- Un alto miembro de la organización que conoce los objetivos estratégicos.
- Los empleados vinculados al proceso.
- Los trabajadores de otras áreas que cuentan con la actual experiencia y que desean conocer las mejoras que se llevarán a cabo.

4.3 Generar el diagrama de los procesos de negocio

Se trata de representar en una secuencia lógica los elementos expresados en la TGS nombre del proceso el cual debe ser un verbo (fabricar, validar, informar) que permite modelar en un dibujo el proceso de manera rápida, dinámica y sencilla acerca de como fluyen las entradas en cada elemento y hacia donde se deben dirigir sus salidas. En el diagrama es fundamental definir las competencias de cada uno de los integrantes del equipo en cada tarea.

El diagrama de procesos es también conocido como diagrama de flujo y en el mundo de los sistema orientados por procesamiento electrónico de datos obedece a un algoritmo es decir al conjunto de pasos lógicos para resolver un problema, es una de las herramienta mas útil para cualquier entidad que quiere gestionar proyectos de todo tipo tanto a nivel interno como externo a ella. Permite abordar la descomposición y conocimiento de los procesos empresariales en orden lógico plasmados dentro de un único documento con sus relaciones, visualizar oportunidades de mejora y, en general para dar importancia y generar control hasta el mas mínimo detalle a todos los procesos de una compañía, por pequeños que puedan considerarse.

4.3.1 Diagrama de procesos

Teniendo en cuenta la teoría general de sistemas antes relacionada un diagrama de procesos es la forma en la cual se aplica el enfoque sistémico para representar de manera gráfica los componentes de las principales actividades que pueden inferirse a su vez como subsistemas y que conforman el todo interoperante en un

orden lógico guiados por sus interrelaciones. Muestra el flujo secuencial de interacción de las actividades de un proceso a través de símbolos gráficos para proporcionar una mejor visualización del funcionamiento del mismo, ayudando a su entendimiento y haciendo su descripción más didáctica, visual e intuitiva.

Así, el diagrama de procesos se convierte en una herramienta primordial para llevar acabo el análisis de manera global y ver en qué acciones del flujo de trabajo se pueden introducir métricas, controles o descubrir oportunidades para mejorar, especialmente pretende orientar los aspectos que permiten como primera medida aumentar la productividad de los empleados, delimitar y asignar las competencias para cada tarea, ver cuellos de botella es decir posibles atascos en el devenir del flujo y en general para aclarar el propio flujo de trabajo.

En definitiva, es una de las herramientas que mas ayuda en la consecución de una mejor gestión empresarial integral, especialmente en aquellas entidades en donde los procesos suelen ser acoplados, de alta complejidad, bastante rígidos o están muy interrelacionados.

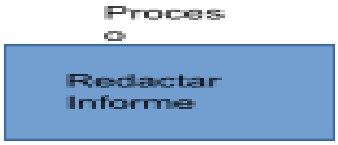
Teniendo en cuanta las labores a ejecutar vs un tiempo estimado o empleado los diagramas de procesos más utilizados para la gestión de proyectos son el diagrama de Gantt y el diagrama de Pert. Los dos diagramas permiten planificar, simplificar, programar actividades, asignar responsabilidades, pesos por valor de actividad de forma sencilla y muy visual que facilitan su entendimiento.



4.3.2 Elementos de un diagrama de procesos

A la hora de construir es decir de representar y dibujar un diagrama de procesos, existe un lenguaje de escritura que se conforma por una serie de símbolos comúnmente aceptados que han sido establecidos por el estándar ANSI.
Estos diagramas están compuestos por elementos gráficos como lo son terminadores, símbolos de procesos, de subprocesos y de decisiones, líneas con flechas y conectores. Los principales elementos están representados de la siguientes formas:

- Procesos o actividades. Su representación es un rectángulo, se refieren a una acción en un proceso de negocios. Son, posiblemente, los elementos más importantes de un diagrama en cada uno de estos se recomienda de manera detalla aplicar el análisis sistémico en cada actividad, son los que siempre deben aparecer, y por eso deben describirse de forma clara y concisa. Deben ser descritos con una frase única de verbo y sustantivo. Por ejemplo: “Redactar un nuevo informe para presentar a la gerencia de la empresa”.



- Subprocesos. Estos se representan con un rectángulo y líneas dobles a cada lado. Deben ser considerados como partes de un proceso padre de mayor complejidad y sirven para determinar funciones más simples al diagrama de flujo incluso reutilizables.



- Nodos de decisión. Son los que se representan mediante un triangulo doble (Diamante) y son los que muestran los puntos de decisión a seguir bien sea positivo si se responde a la decisión de “sí” o negativo “no”.



- Conectores. Representados por círculos o cuadros conectores que se deben acompañar con la utilización de etiquetas o nombres de caracteres comúnmente alfanuméricos. Aseguran que todos los procesos están conectados de forma lógica y correcta para la correcta navegación entre páginas.



- Líneas de flecha. Utilizadas gráficamente para mantener el hilo conductor del flujo de trabajo la coherencia y claridad en un diagrama de procesos. Se pueden dibujar hacia todas las direcciones, normalmente representa el camino por el que fluirá el diagrama.



- Conector para elemento fuera de pagina. Permite establecer relaciones para elementos que no están en la misma página.



- Entrada manual. Permite representar un acción la cual requiere intervención humana



- Almacenamiento. Representa el sitio en donde normalmente se aloja la información a manera de datos que comúnmente son de carácter estático.



- Múltiples documentos. Representa la necesidad de tener en cuenta varios elementos de información o normas, especialmente son considerados como entradas.



- Terminadores. Se representan por un rectángulo con esquinas curvas. Son muy importante se consideran como los mayores delimitadores del flujo y deben aparecer siempre al inicio y al final de un diagrama de flujo para indicar la parte del documento o gráfico en donde inicia y en donde finaliza un diagrama.



4.3.3 Un buen diagrama de procesos

Al momento de crear un diagrama de procesos se recomienda guiarse por al menos cumplir los siguientes pasos:

1. Ubicar o según sea el caso abstraer los componentes del proceso. En este paso se aclaran cuáles son los parámetros de entrada al proceso y sus salidas, así como las actividades que se desarrollan en el mismo.

2. Ordenar de manera lógica las actividades. Dado que el diagrama de procesos es un flujo, es necesario ordenar las actividades para saber cuáles es su secuencia.

3. Elegir los símbolos adecuados conforme el tipo de tarea para que cualquier usuario pueda diferenciar los distintos elementos que forman el diagrama.

4. Hacer la conexión y relación entre las actividades, para lo cual se utilizan flechas y líneas de puntos o continuas.

5. Representar el comienzo y el final de cada proceso. Aunque pueda parecer trivial e irrelevante, lo cierto es que este es un punto muy importante para establecer los límites de cada procesos u actividad de cara a los propietarios de los procesos, gerentes y supervisores.

6. Validar, socializar con los integrantes del equipo y revisar el diagrama de procesos para asegurarse que su representación obedece a un gráfico acorde con lo que se quiere reflejar.

4.3.4 Dividendos de los diagramas de procesos

Utilizar un diagrama de flujo le aporta los siguientes beneficios a tu empresa:

- Permite obtener la primera aproximación desde el punto de vista algorítmico de cada actividad o proceso así como también puntos de interoperabilidad.
- Favorece la comprensión del proceso que trata de simular. Dado que normalmente están representadas a través de algún elemento gráfico y sus nodos están interconectados entre sí, gracias a un diagrama de este tipo es más fácil saber cuáles son las tareas que se dibujan, así como sus relaciones de dependencia.
- Permite identificar de una forma rápida los inconvenientes y dónde se producen. Así, es más fácil saber dónde se crean o se presentan posibles cuellos de botella, las responsabilidades de cada área y los puntos de decisión. Asimismo, permite analizar de una forma mucho más exhaustiva el problema, lo cual hace posible que aparezcan nuevas soluciones y alternativas.
- Permite definir claramente las fronteras de cada proceso, ya que todas las tareas están lo suficientemente delimitadas, incluso aunque estén interrelacionadas. De esta manera, se puede saber qué tareas suponen un mayor valor añadido y cuáles no aportan tanto.
- También es una buena herramienta de capacitación, instrucción y formación para nuevos empleados que se integran dentro de la empresa facilitando su proceso de adaptación a la organización

4.4 Realizar un diagnóstico de cada proceso

Esta labor permite a las empresas detectar por me dio la implementación de estrategias para detectar los problemas de funcionamiento y deficiencias estas estrategias son: entender las causas propias que maneja los casos de cada proceso ejecutar simulacros de relevamiento de procesos unido a sus responsables , apoyarse en herramientas de software para registrar resultados, controles, métricas, análisis de tiempos, costos y materias primas.

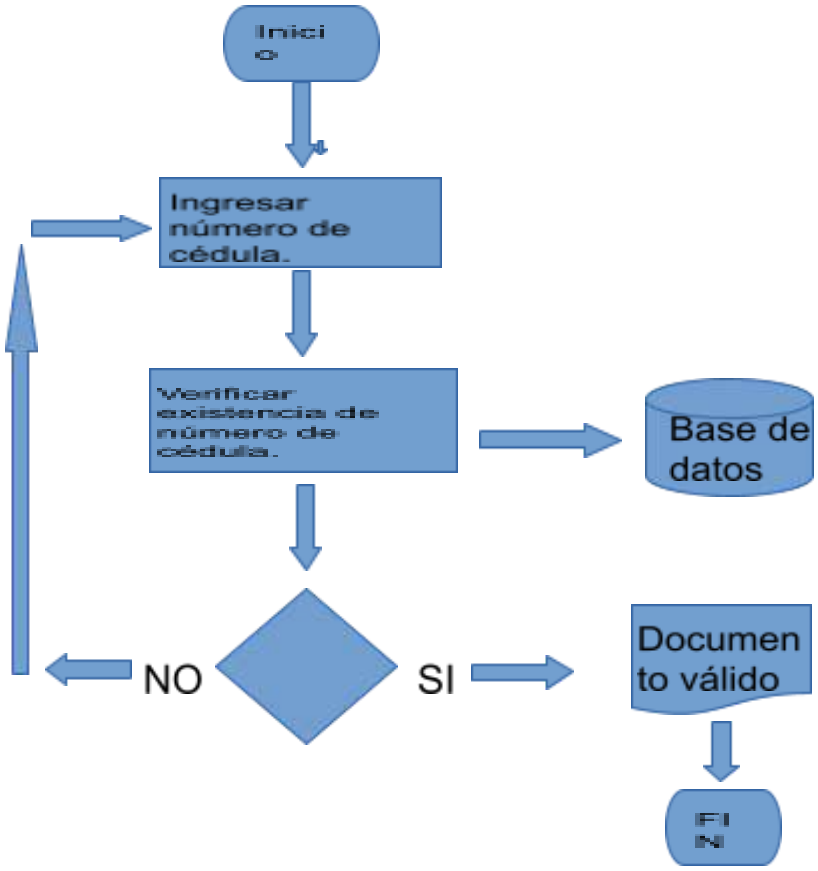
4.5 Puntos de mejora

Luego del diagnóstico de la situación actual, es importante construir un informe que de a conocer el estado anterior y el estado actual hacia donde se quiere llegar con las mejoras. En este informe se recomienda de manera obligatoria el poder relacionar los alcances de las soluciones unidos en forma transversal con los objetivos estratégicos de la organización.

4.6 Nuevo proceso mejorado.

Los resultados obtenidos de la aplicación de cada uno de los pasos anteriores se debe compilar y almacenar esta información de salida con el objetivo de llevar trazabilidad y de la reunión de dicha información sale el diseño de un nuevo proceso, el cual estará destinado al logro de los objetivos estratégicos de la compañía y así llegar a un estadio de mayor eficiencia y eficacia.

Ejemplo de diagrama.



4.7 Caso de estudio.

Identificar el conjunto de elementos que intervienen en un sistema de compras online de una fabrica aplicando la TGS para finalmente proponer las actividades representadas en un diagrama de procesos.

SISTEMA DE COMPRAS EN LINEA

ENTRADAS	SUB-PROCESO O ACTIVIDAD	SALIDAS
• Datos del solicitante (Nombres, apellidos, identificación, dirección de contacto). • Producto a comprar (Valor , Cantidad). • Número único de identificación de la compra.	Recepción de solicitud de compra. Almacén	• Número único de identificación de la compra. • Factura. • Envió. • Producto.

• Insumos • Mano de obra • Facturas	Fabrica Contabilidad	• Producto elaborado. • Informes contables

Puesto que el sistema de compras en linea no intercambia elementos de información con el medio ambiente que lo rodea se considera un sistema cerrado.

Se pueden inferir las siguiente relaciones:

Usuario solicitante contra el proceso de solicitud de compra en donde un usuario puede generar una o muchas solicitudes de compras.
El almacén maneja el inventario de productos.
La fabrica provee de muchos productos al almacén.
El almacén aprovisiona el producto para la solicitud.
El usuario realiza el pago delproducto.
Contabilidad registra la factura de compra generada por la solicitud.

Actores Responsables:

Usuario que genera la solicitud.
Jefe de ventas.
Administrador del sistema Online.
Jefe de contabilidad.
Jefe del control de operaciones de fábrica.
Jefe de inventarios.
Empleados de la fábrica.
Gerencia de Área.

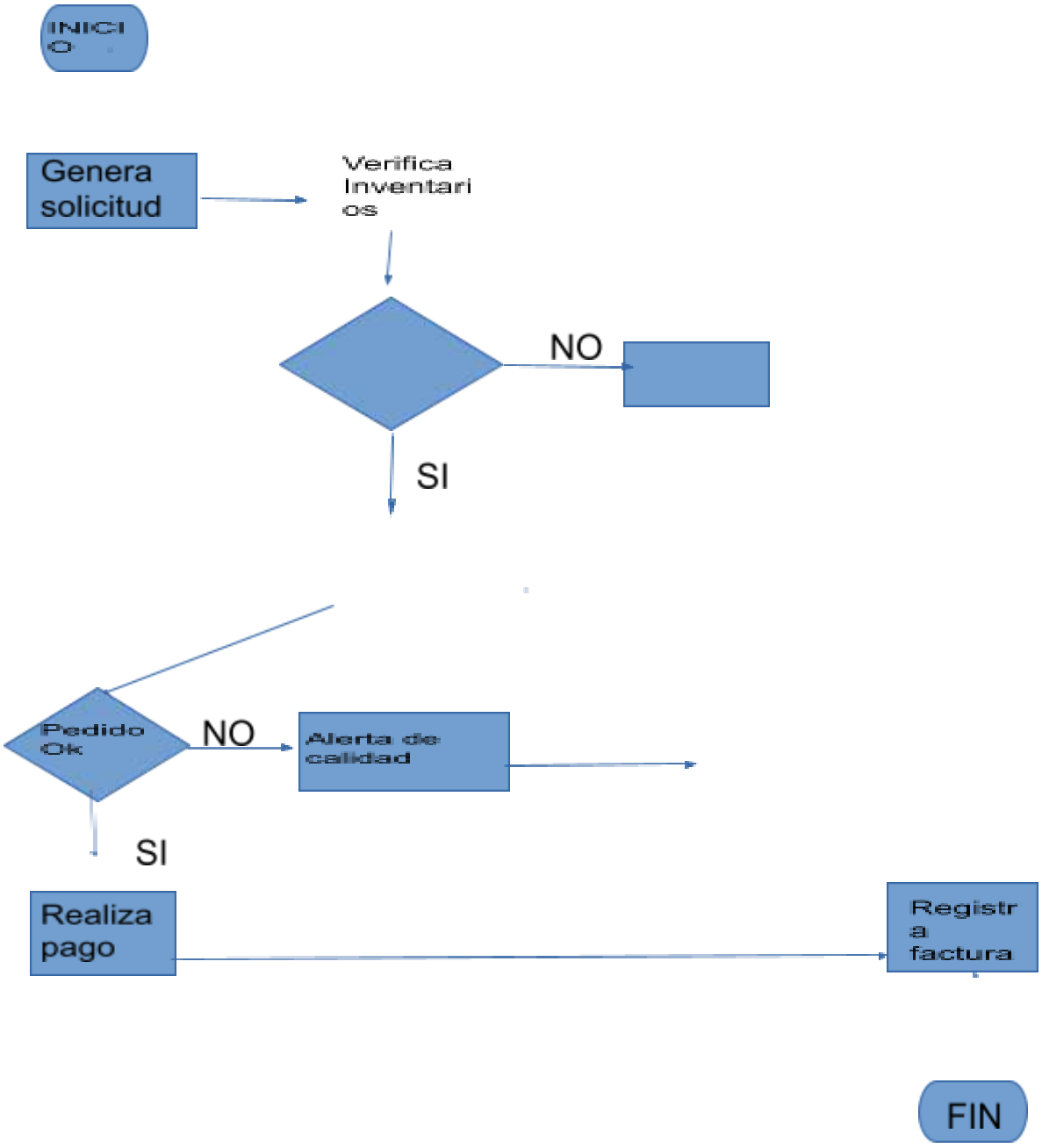
Diagrama de procesos:

SISTEMA DE COMPRAS EN LINEA



SUBPROCESO

SOLICITUD DE CLIENTE	ALMACEN	FABRICA	CONTABILIDAD



A. **ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)**

DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA	
Nombre de la Actividad	Conceptos de Teoría general de sistemas.

Objetivo de la actividad	Identificar los conceptos de la teoría general de sistemas para ser aplicados a la caracterización de procesos.
Tipo de actividad sugerida	Selección múltiple.
Archivo de la actividad (Anexo donde se describe la actividad propuesta)	Anexo documento en Word llamado Actividad didáctica 1

DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA	
Nombre de la Actividad	Pasos lógicos de aplicación de conceptos.
Objetivo de la actividad	Afianzar algunos de los conceptos más importantes de la elaboración de un diagrama de procesos.
Tipo de actividad sugerida	Secuencia didáctica.
Archivo de la actividad (Anexo donde se describe la actividad propuesta)	Anexo documento en Word llamado Actividad didáctica 2

B. **MATERIAL COMPLEMENTARIO:**

Tema	Referencia APA del Material	Tipo de material (Video, capítulo de libro, artículo, otro)	Enlace del Recurso o Archivo del documento o material
Crear un diagrama de flujo.	Barraza.A,Felix.L,Leyva.M,Martínez.L,Sáenz.B,Sánchez.K,Flores.V (2014).Mapas Conceptuales Diagramas de Flujo y Esquemas	LIBRO	https://d1wqtxts1xzle7.cloudfront.net/43720567/mapasesquemasflujo-with-cover-page-v2.pdf?Expires=1633992867&Signature=EJF5szZVIc3XTWMUHDZvzfa2Se~Uyz0IqjYyQaYDuvTUyCouWnA9n6Ux7AExLjQbkqDC~Ma3-QTiA5uyrcgCH2ubbiZcl8SSihrGqXzi8z04tZl0eGXo1PBoVcKOgMXREqTLep8FrFCxzRMNS8rfNqwEnNkWF~fXfDfKhfJ9hbPA54dQ9z9SVE64qHiWREYZK-wS1cnDY6fI1AU5qNt4W

			lUh0xOSUJ3epJhsGK~Immrd~~xHhT2Jaw37M4PROT1xGjmnleYeUht8ds1cssKxXrV2zsvzzpWH7GXxCQoOV8-LJrjlNRMXMsq~Ufx-c9rutSzXScF4T19fDh2Xb4p3A__&Key-Pair-Id=APKAJLOHF5GGSLRBV4ZA
Crear un diagrama de flujo online.	Lucidchart Software (2021).	Página web	https://www.lucidchart.com/pages/es

C. **GLOSARIO:**

TÉRMINO	SIGNIFICADO
ANSI	(American National Standards Institute).
GANTT	Método popularizado por Henry Gantt
PERT	Program Evaluation Research Task
HOLISMO	Capacidad de analizar un componente como un todo.
RETROALIMENTACION	Cuando parte de un mensaje de salida se convierte nuevamente en entrada.
SINERGIA	Cuando varios elementos de una organización actúan de manera concertada.
TEORIA DE COLAS	Teoría que estudia los tiempos de espera dentro de un sistema
SIMULACION	Corresponde a la posibilidad de conducir experimentos en una computadora.
TEORIA DE COMPORTAMIENTOS	Orientada al estudio del comportamiento humano en la administración.
TEORIA DE JUEGOS	Corresponde a la aplicación de un modelo matemático juego para entender la toma de decisiones.
TGS	Teoría General de sistemas.

D. **REFERENCIAS BIBLIOGRÁFICAS:**

- Tamayo. A (1999). Teoría general de sistemas.
- Torres. A (2021). La Teoría General de Sistemas, de Ludwig von Bertalanffy. *https://psicologiaymente.com/psicologia/teoria-general-de-sistemas-ludwig-von-bertalanffy*
- Centro Tic de Andalucía (2019). El enfoque sistémico. *https://www.juntadeandalucia.es/averroes/centros-tic/14002996/helvia/aula/archivos/repositorio/250/271/html/economia/1c/sistemico.htm*

- Deusto(2019). ¿Datos e información es lo mismo ?. *https://blogs.deusto.es/master-informatica/datos-e-informacion-son-lo-mismo/*

- ESAN Graduate School of Business (2019). ¿Qué es el análisis de procesos de negocio y cómo aplicarlo en mi empresa?). *https://www.esan.edu.pe/apuntes-empresariales/2019/11/que-es-el-analisis-de-procesos-de-negocio-y-como-aplicarlo-en-mi-empresa/*

- E.Ekon (2020).¿Qué es un diagrama de procesos y por qué es tan importante para tu empresa?. https://www.ekon.es/diagrama-procesos-empresa/

- I. Espinal, A.Gimeno, F.González, (1998). El Enfoque Sistémico En Los Estudios Sobre La Familia.(Articulo). *https://www.uv.es/jugar2/Enfoque%20Sistemico.pdf*

- L. Bertalanffy (1968). Theory General Systems

E. **CONTROL DEL DOCUMENTO**

	Nombre	**Cargo**	**Dependencia**	**Fecha**
Autor (es)				

F. **CONTROL DE CAMBIOS**

	Nombre	**Cargo**	**Dependencia**	**Fecha**	**Razón del Cambio**
Autor (es)					

ANEXO
Actividad didáctica 1
Opción múltiple

Favor escoger la respuesta correcta:

1. La Teoría General de Sistema fue propuesta por:

a. Einstein
b. Tesla
X c. Bertalanfy
d.Von Newman

2. Un sistema se define como:

a. Conjunto de elementos interrelacionados
b. Susbsistemas aislados con interés dispersos
c. Conjunto de elementos de comportamiento caótico
d. Pensamientos que solo viven en el universo de las ideas del hombre

3. Holismo se puede definir como:

a. Conjunto de partes que se encuentran solas
b. Partes que se agrupan y son más que el todo
c. Es una comportamiento del proceso
X d. El todo es más que la suma de las partes

4. En orden cuales son los elementos de un sistema:

a. Retroalimentación, salidas,entradas, procesos,contexto
b. Contexto, entradas, retroalimentación, proceso, salidas
X c. Contexto,entradas, proceso, salidas, retroalimentación
d. Retroalimentación, entradas, salidas,proceso

5. Un sistema cerrado se caracteriza por:

a. En ese sistema nadie puede entrar
X b. No existe intercambio de información con el medio ambiente
c. Los elementos no se pueden dividirse
d. Las partes están desordenadas

6. Cual de los siguientes sistemas puede considerarse como natural:

a. El sistema de trenes
b. Una ciudad
c. Una piscina
X d. Las aguas del mar

7. Cúal de las siguientes no es una técnica para recolectar información

a. Observación y entrevistas
b. Enfoque de grupos
c. Encuestas o formularios
X d. La edición de texto

8. Cúal de siguiente no pertenece a los diagramas de proceso

a. Diagrama de gantt
b. Diagrama de pert
X c. Diagrama de estados
d. Diagrama de flujo

PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Desarrollo de aplicaciones informáticas aplicando Inteligencia Artificial
- Código del Programa de Formación:
- Nombre del Proyecto: Construcción de software de integración aplicando inteligencia artificial y tecnologías orientadas a servicios
- Fase del Proyecto : Planeación
- Actividad de Proyecto: Establecer procesos lógicos para el desarrollo de algoritmos
- Competencia: Diseñar la solución de software de acuerdo con procedimientos y requisitos técnicos
- Resultados de Aprendizaje Alcanzar: 220501095-01 Plantear algoritmos de acuerdo con el problema a solucionar.
- Duración de la Guía: 96 horas

2. PRESENTACIÓN

Los algoritmos se utilizan para encontrar soluciones a un problema. No siguen reglas para la búsqueda de la solución, simplemente una búsqueda sistemática, que más o menos viene a significar que hay que probar todo lo posible hasta encontrar la solución o encontrar que no existe solución al problema. Para conseguir este propósito, se separa la búsqueda en varias búsquedas parciales o subtareas. Asimismo, estas subtareas suelen incluir más subtareas, por lo que el tratamiento general de estos algoritmos es de naturaleza recursiva.

Puesto que a veces nos interesa conocer múltiples soluciones de un problema, estos algoritmos se pueden modificar fácilmente para obtener una única solución (si existe) o todas las soluciones posibles (si existe más de una) al problema dado.

El objetivo general de esta guía es representar bosquejos de soluciones mediante el uso de pseudocódigo y diagramación estructurada.

EL ENTUSIASMO Y COMPROMISO EN SU PROCESO DE FORMACIÓN ES DECISIVO PARA SU DESARROLLO PROFESIONAL

¡ÁNIMO!

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

3.1 Actividades de Reflexión Inicial

3.1.1 Participo en una charla introductoria con el instructor, para analizar y meditar sobre las actividades que vamos a desarrollar, propongo ideas y estrategias para la consecución de los resultados de aprendizaje.

3.1.2 Expongo mi concepto sobre el desarrollo de las actividades del proyecto y los problemas presentados hasta el momento, propongo ideas para mejorar.

3.2 Actividades de Contextualización e Identificación de Conocimientos Necesarios para el Aprendizaje

Los primeros algoritmos registrados datan de Babilonia, originados en las matemáticas como un método para resolver un problema usando una secuencia de cálculos más simples.

3.2.1 Realizo un trabajo de investigación dándole respuesta a los siguientes interrogantes:

- ¿Qué es un algoritmo?
- ¿Qué es pseudocódigo?
- ¿Qué es un diagrama de flujo de una aplicación software?
- ¿Cuáles son los pasos a seguir para la solución de un problema por medio de un algoritmo?

3.2.2 Realizo la lectura y ejercicios propuestos en el documento "**Introducción a la algoritmia.pdf**".

En plenaria, reflexione sobre los conceptos construidos por el grupo y genere las conclusiones de la actividad.

Duración: 1 hora

3.3 Actividades de apropiación del conocimiento

Actividad de aprendizaje 1: Desarrollar algoritmos que den respuesta a problematicas planteados.

Participo de la socialización que realiza el instructor sobre las diferentes temáticas relacionadas con algoritmia.

3.3.2 Realizo la lectura "**Componente formativo Análisis y solución de problemas aplicando algoritmos**" que se encuentra en el material de apoyo suministrado y realizo ejercicios propuestos. En plenaria socialice los ejercicios propuestos y genere conclusiones de la actividad.

3.3.3 Realizo la lectura "Estructuras condicionales.pdf" que se encuentra en el material de apoyo suministrado.

3.3.4 Realizo la lectura "Estructuras repetitivas - Mientras.pdf" que se encuentra en el material de apoyo suministrado.

3.3.5 Realizo la lectura "Estructuras repetitivas - Para.pdf" que se encuentra en el material de apoyo suministrado.

3.3.6 Realizo la lectura "Estructuras repetitivas - Repita.pdf" que se encuentra en el material de apoyo suministrado.

- Autoestudio con realización de ejercicios de comprensión (cuestionarios, resúmenes, mapas conceptuales, esquemas, etc.).
- Socialización con el equipo de trabajo y con el instructor.
- Realización de las prácticas de laboratorio en el momento pertinente.
- Realizar la retroalimentación, síntesis y conclusiones sobre la actividad.

Ambiente requerido para la actividad: mesa, silla y conectividad a Internet alámbrica o inalámbrica.

Materiales para la actividad: computador de escritorio o portátil, sistema operativo Windows (recomendable), software ofimático y material de apoyo.

3.4 Actividades de Transferencia del Conocimiento

Después de realizadas las lecturas y socializar con el instructor las temáticas de análisis de algoritmos, estructuras secuenciales, repetitivas y condicionales realizar los siguientes talleres.

- Desarrollo los ejercicios planteados en el archivo "**Taller de aplicación de conceptos iniciales algoritmia.pdf**". Tenga en cuenta las condiciones de entrega.
- Desarrollo los ejercicios planteados en el archivo **"Taller de aplicación de diseño de algoritmos.pdf**". Tenga en cuenta las condiciones de entrega.
- Me preparo para presentar una evaluación sobre estructuras algorítmicas por medio de la construcción de algoritmos para solucionar problemas especificados según requerimientos establecidos por el instructor.
- Desarrollo los ejercicios planteados en el archivo **"Taller de aplicación de codificación algoritmos.pdf**". Tenga en cuenta las condiciones de entrega.
- Reflexiono: ¿Cómo pueden las temáticas vistas aportarme en la elaboración de la solución de mi proyecto formativo y en la elaboración de soluciones en el entorno laboral

4. ACTIVIDADES DE EVALUACIÓN

Tome como referencia la técnica e instrumentos de evaluación citados en la guía de Desarrollo Curricular

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento **Evidencias de Desempeño** ● Evaluación(es). ● Taller(es)		**TECNICA:** Observación directa **INSTRUMENTO**: Lista chequeo

Evidencias de Producto: **Codigo fuente-** Taller de solución de algoritmos empleando herramientas y/o legunaje de programación seleccionado**.**		

5. GLOSARIO DE TÉRMINOS

Software: Soporte lógico, programas, parte no mecánica de un sistema. Serie de instrucciones necesarias para ejecutar diversas aplicaciones y tareas.

Lenguaje de programación: Un lenguaje de programación es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado de un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones.

Compilador: Un compilador es un programa informático que traduce un programa escrito en un lenguaje de programación a otro lenguaje de programación, generando un programa equivalente que la máquina será capaz de interpretar. Usualmente el segundo lenguaje es lenguaje de máquina, pero también puede ser simplemente texto. Este proceso de traducción se conoce como compilación.

Intérprete: es un programa informático capaz de analizar y ejecutar otros programas, escritos en un lenguaje de alto nivel. Los intérpretes se diferencian de los compiladores en que mientras estos traducen un programa desde su descripción en un lenguaje de programación al código de máquina del sistema, los primeros (los intérpretes) sólo realizan la traducción a medida que sea necesaria, típicamente, instrucción por instrucción, y normalmente no guardan el resultado de dicha traducción.

Algoritmo: es un conjunto preescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien lo ejecute. Dados un estado inicial y una entrada, siguiendo los pasos sucesivos se llega a un estado final y se obtiene una solución.

Programa: conjunto unitario de instrucciones que permite a una computadora realizar funciones diversas, como el tratamiento de textos, el diseño de gráficos, la resolución de problemas matemáticos, el manejo de bancos de datos (RAE, 2021).

6. REFERENTES BILBIOGRÁFICOS

- López, J. C. (2009). Educación básica algoritmos y programación. Guía para docentes. Segunda edición. Fundación Gabriel Piedrahita Uribe. www.eduteka.org https://libros.metabiblioteca.org/bitstream/001/169/8/AlgoritmosProgramacion.pdf

- Medina, H. M. (2018). Estrategias metodológicas para el desarrollo del pensamiento lógico-matemático. *Didasc@lia: Didáctica y Educación*, 9(1), p. 125-132. https://dialnet.unirioja.es/descarga/articulo/6595073.pdf

- Real Academia Española. (2020). *Algoritmo*. Diccionario de la lengua española. RAE. https://dle.rae.es/algoritmo

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)				

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

Introducción

El producto principal de la industria del *software* es un conjunto de instrucciones de computador que, correctamente implantadas, pueden llamarse programa, módulo de programa, aplicación, servicio o simplemente producto *software*. Codificar este conjunto de instrucciones es la tarea que trata de convertir algoritmos en un programa o módulo que compone un programa y que una computadora o teléfono inteligente (*smartphone*) es capaz de ejecutar. Estas instrucciones están escritas en un lenguaje de programación que puede ser interpretado o compilado por una máquina.

El lenguaje de programación es como un sistema de comunicación entre la persona (programador) y la máquina que consiste en un conjunto de reglas gramaticales bien definidas, que le ofrecen al programador la capacidad de escribir (programar) los algoritmos de los que está compuesto un sistema informático. En particular se emplea el lenguaje JavaScript, ya que casi todos los navegadores lo interpretan, y se puede usar para publicar servicios y aplicaciones en internet.

1. Conceptos básicos de programación

Programar es el proceso de crear *software* escribiendo, probando, depurando y dando mantenimiento a las instrucciones del computador en un lenguaje de programación. A este conjunto de instrucciones se les denomina código fuente del *software* creado.

Existen centenares de lenguajes de programación, muchos de ellos fueron creados para dar respuesta a problemas particulares o máquinas específicas. No es de interés ahora discutir todos los tipos de lenguajes existentes, sino, más bien, conocer aquellos que pueden ser interpretados por computadores, *smartphones* o que pueden ser empleados para proveer servicios informáticos a través de internet y su clasificación más general.

Un lenguaje de programación es diferente al lenguaje de códigos que puede entender la máquina (lenguaje de máquina). Los lenguajes de programación, pueden dividirse en dos categorías: lenguajes interpretados o lenguajes compilados, a saber:

Tabla 1
Lenguajes de programación

Compilado	Interpretado
Definición	
El código creado debe traducirse de manera que el procesador del computador o *smartphone* pueda comprenderlo, a este proceso se le llama compilación.	**Un programa escrito en un lenguaje interpretado, necesita de otro programa auxiliar (el intérprete), que traduce los códigos fuentes de un programa y los interpreta para la máquina.**
El lenguaje compilado necesita que antes de ser ejecutada una compilación se convierta el código fuente escrito a un lenguaje de máquina.	**Un lenguaje interpretado es convertido a lenguaje de máquina cada vez que es ejecutado (para ello requiere del intérprete), que traduce las instrucciones.**
Características	

Para un lenguaje compilado luego de compilar un programa se debe crear ejecutables para cada uno de los sistemas operativos en los que lo vaya a utilizar. Es decir, un programa compilado en Linux no servirá en Windows o en Mac OS.	**Por lo general, en el ciclo de desarrollo de *software* (tiempo entre el momento en que escribe el código y lo prueba) es más corto en un lenguaje interpretado dado que no se necesita realizar el proceso de compilación, cada vez que cambia el código fuente.**
Cuando se está ejecutando un programa escrito en un lenguaje compilado es mucho más rápido que uno interpretado, dado a que se encuentra en código de máquina.	**Un lenguaje interpretado para ser ejecutado, debe tener instalado el interpretador.**
Los lenguajes compilados están optimizados para el momento de la ejecución, pero esto significa más trabajo para el programador (compilarlo).	**El código fuente en un lenguaje interpretado está optimizado para hacerle la vida más fácil al programador, aunque esto signifique una carga adicional de procesamiento para la máquina.**

JavaScript, al igual que cualquier otro lenguaje de programación, tiene algunas características especiales: sintaxis, modelo de datos, etc. JavaScript **es un lenguaje esencial para cualquier principiante de programación y también para quienes ya saben programar**, debido a que las diferencias con otros lenguajes de programación son numerosas siendo un lenguaje necesario para el desarrollo de aplicaciones web y móviles hoy en día.

En adelante, JavaScript es el lenguaje de programación abordado y es un lenguaje interpretado que inicialmente fue creado para que los navegadores lo interpreten y hoy en día muchos dispositivos tienen el intérprete que permite correr programas escritos en él **por lo que será más sencillo el trabajo tanto para la máquina como para el programador.**

2. Entornos de codificación e instalación JavaScript

En su mayoría los navegadores (Chrome, Mozilla, Edge) tienen intérprete de JavaScript y adicional se puede instalar el intérprete para el sistema operativo de preferencia (Framebits, 2020).

Dado lo anterior, existen herramientas, manuales o videos que se emplean para el desarrollo de programas en JavaScript y a continuación se comparten recursos de acceso y material audiovisual de apoyo de cada herramienta, tal como se indica en la siguiente tabla.

Tabla 2
Entornos de codificación

Herramienta	Recurso Web	Uso y/o instalación
Editor y compilador en línea PlayCode	https://playcode.io/	**Woodrow, J (2018) How to use playcode.io. https://www.youtube.com/watch?v=9rtsvbc4KZI**
Intérprete de JavaScript: Node js	https://nodejs.org/es/	**Framebits. (19 de enero. 2020). Descargar e instalar Node Js en Windows 10. Youtube. https://youtu.be/v0x1Ku5Tgac**
Editor de código Visual Studio Code	https://code.visualstudio.com/	**Code Compadre. (30 de junio. 2020). How to Download and Install Visual Studio Code for Windows 10. Youtube. https://youtu.be/KpzkPlh_HsU**
El navegador: Google Chrome	https://www.google.com/chrome/	**Choque, R. C. (2020, May 10). Cómo usar la consola de google chrome para Javascript.**

		Youtube. Retrieved from https://www.youtube.com/watch?v=Hf3n-p3VYx4

3. Sintaxis del lenguaje JavaScript

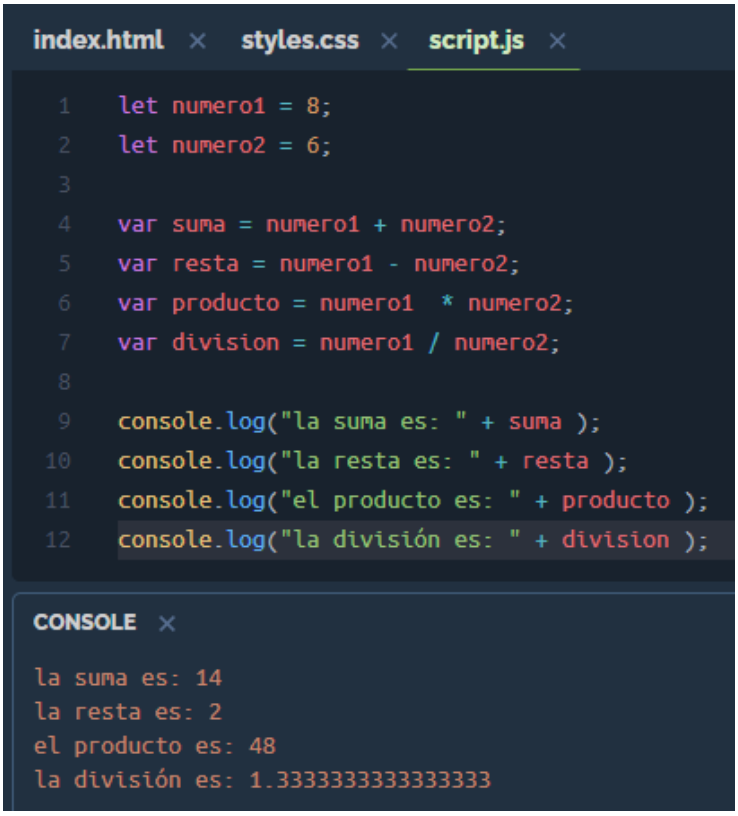
En adelante, se presentan las características básicas de la sintaxis de JavaScript, codificando algoritmos básicos, para ejemplificar la estructura propia del lenguaje de programación, sin embargo, se recomienda que siempre que se esté codificando algoritmos, se pueda tener a disposición la documentación técnica oficial del lenguaje y material complementario propuesto.

3.1. Funciones y operaciones matemáticas

En JavaScript se pueden utilizar muchas funciones matemáticas; por ejemplo, un algoritmo que toma dos números (8 y 6) y le aplica las 4 operaciones básicas y en el editor de código online PlayCode *(*disponible en https://playcode.io/*)* se revisan las características de la sintaxis del JavaScript y a la vez se conocen las operaciones matemáticas del lenguaje.

Ingresar en la ruta del navegador a https://playcode.io/new/, escribir el código de la línea 1 a la 12 en la pestaña script.js como se muestra en la siguiente figura:

Figura 3
Operaciones aritméticas con JavaScritpt



Explicación: en las líneas 1 y 2 aparece la palabra reservada **let** que sirve para indicar que se define una variable accesible en el contexto donde se crea (cuando se vea programación modular quedará más claro esto del contexto).

```
let numero1 = 8;
let numero2 = 6;
```

Las palabras reservadas **numero1** y **numero2**, se les asigna el valor correspondiente. Es de observar que se definen y se inicializa la variable en una misma línea, esto es una característica del lenguaje.

```
var suma = numero1 + numero2;
var resta = numero1 - numero2;
```

Luego en las líneas 4 al 7 se crean o se definen unas variables con la palabra reservada **var,** a diferencia de la palabra **let** estas variables se pueden ver desde cualquier parte o módulo del script.

```
var producto = numero1 * numero2;
var division = numero1 / numero2;
```

Una vez se crean las variables con identificadores **suma**, **resta**, **producto**, **division**, también son inicializados sus valores empleando las operaciones matemáticas respectivas.

Una de las formas que JavaScript permite mostrar los resultados es con la función **log** del objeto **console** y se accede **console.log**() (con el operador punto **.**) y entre paréntesis lo que se desea mostrar en la pantalla de la consola.

```
console.log("la suma es: " + suma);
console.log("la resta es: " + resta);
```

Validación: dentro de los paréntesis "la suma es: " + suma, aparece la operación más + usado para sumar dos números antes, solo que acá no hay dos números sino una cadena de texto y un número, pero en este caso lo que hace el operador + es concatenar o juntar el texto “la suma es:” y el valor de la variable suma. A esta característica de los operadores se les llama sobrecarga de operadores, porque según el tipo de datos suma y según otra concatena.

Cuando las operaciones matemáticas son más complejas como la raíz cuadrada, la función exponencial, las funciones trigonométricas, existe un objeto que agrupa estas funciones se llama **Math,** puede ver la documentación técnica de la clase completa en Math (MDN, 2021b), a continuación, un ejemplo de su uso.

Se debe crear un algoritmo que muestre el seno del valor PI (3,1416) multiplicado por 3 la función, y también que nos eleve 6 a la tercera potencia mostrando su resultado:

Figura 4
Uso del objeto Math



Se debe escribir el código fuente de las figuras 3 y 4, y obtener los mismos resultados en la consola, como se muestra en cada imagen, de esta forma se familiariza con la herramienta de trabajo.

A continuación, puede revisar el listado de funciones de la clase Math.

Tabla 1
Métodos del objeto Math
Nota: Tomada de MDN (2021b)

Todas las funciones matemáticas se pueden combinar con otras operaciones aritméticas o emplear operadores de evaluación de condición y previo a ello es preciso conocer los diferentes tipos de datos que existen.

3.2. Tipos de datos, operadores y orden de evaluación

Existen muchos tipos de operadores para JavaScript, cuya función es realizar una operación entre dos o más valores contenidos en variables, constantes o acumuladores:

A. Tipos de datos

JavaScript tiene los siguientes tipos de operadores (se describirán los principales para aprender a programar).

Según sea el tipo que las variables o constantes almacenan, se pueden clasificar en seis (6) tipos de datos primitivos la referencia del lenguaje dice que son:

- ***Undefined***: indeterminado o indefinido
- ***Boolean***: tipo booleano los valores posibles son *true* o *false*.
- ***Number***: números enteros, o decimales.
- ***String***: cadenas de texto.
- ***BigInt***: números enteros grandes.
- ***Symbol***: referencia a otros datos

Otros tipos de datos más complejos o abstractos de datos:

- ***Null***: tipo primitivo especial que tiene un uso adicional para su valor: si el objeto no se hereda, se muestra *null*.
- ***Object***: tipo estructural especial que no es de datos, pero para cualquier instancia de objeto construido que también se utiliza como estructuras de datos (*new Object, new Array, new Map, new Set, new WeakMap, new WeakSet, new Date* y casi todo lo hecho con la palabra clave *new*).
- ***Function***: una estructura sin datos, aunque también responde al operador t.

Cuando se declaran variables se debe considerar los siguientes tipos, ya se han usado dos (2) de ellas:

- ***var***: declara una variable, opcionalmente la inicia a un valor.
- ***let*****:** declara una variable local con ámbito de bloque, opcionalmente la inicia a un valor.
- ***const***: declara un nombre de constante de solo lectura y ámbito de bloque.

Para ampliar la información de acuerdo con los tipos de datos y estructuras, se debe revisar el material complementario dispuesto:

	*Tipos de datos y estructuras en JavaScript***.** (MDN, 2021e). https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Grammar_and_Types#tipos_de_datos

Una variable en JavaScript puede tener cualquier tipo de dato, entero, real, cadena de texto etc. JavaScript no es un lenguaje fuertemente tipado de manera que queda en el programador la responsabilidad de saber qué tipo de dato está almacenando en cada variable.

B. Operadores

Se considera una expresión, aunque de un tipo distinto, el asignar un valor a una variable. Para todas estas operaciones se emplean los denominados '**operadores**'. Se listan a continuación algunos de los operadores que se utilizan con mucha frecuencia.

❖ **Operadores de asignación**

Un operador de asignación asigna un valor a la variable a la izquierda basándose en el valor de su operando derecho. El operador de asignación más simple es igual (=), que asigna el valor de su operando derecho a su operando izquierdo. Es decir, x = y asigna el valor de y a x. A continuación, se presenta una lista de operadores más usados.

También hay operadores de asignación compuestos que son una abreviatura de las operaciones enumeradas en la siguiente tabla:

Tabla 2

Operadores de asignación básicos

Nombre	Operador abreviado	Significado
Asignación	**x = y**	**x = y**
Asignación de adición	**x += y**	**x = x + y**
Asignación de resta	**x -= y**	**x = x - y**
Asignación de multiplicación	**x *= y**	**x = x * y**
Asignación de división	**x /= y**	**x = x / y**
Asignación de residuo	**x %= y**	**x = x % y**
Asignación de exponenciación	**x **= y**	**x = x ** y**

Nota: Tomada de MDN (2021a)

❖ Operadores de comparación

Son operadores cuya función es comparar dos expresiones y como resultado de la comparación devuelven un valor falso o verdadero (booleano), que representa la relación de sus valores comparados. Existen operadores para comparar valores numéricos, pero también operadores para comparar cadenas y operadores para comparar otros tipos de datos (tabla 3).

Tabla 3

Operadores lógicos

Operador	Descripción	Ejemplos que devuelven true
Igual (==)	**Devuelve true si los operandos son iguales.**	**3 == var1 "3" == var1 3 == '3'**
No es igual (!=)	**Devuelve true si los operandos no son iguales.**	**var1 != 4 var2 != "3"**
Estrictamente igual (===)	**Devuelve true si los operandos son iguales y del mismo tipo. Consulta también Object.is y similitud en JS.**	**3 === var1**
Desigualdad estricta (!==)	**Devuelve true si los operandos son del mismo tipo, pero no iguales, o son de diferente tipo.**	**var1 !== "3" 3 !== '3'**
Mayor que (>)	**Devuelve true si el operando izquierdo es mayor que el operando derecho.**	**var2 > var1 "12" > 2**
Mayor o igual que (>=)	**Devuelve true si el operando izquierdo es mayor o igual que el operando derecho.**	**var2 >= var1 var1 >= 3**
Menor que (<)	**Devuelve true si el operando izquierdo es menor que el operando derecho.**	**var1 < var2 "2" < 12**
Menor o igual (<=)	**Devuelve true si el operando izquierdo es menor o igual que el operando derecho.**	**var1 <= var2 var2 <= 5**

Nota: Tomada de MDN (2021a)

❖ Operadores de aritméticos

Los operadores aritméticos toman valores numéricos (ya sean literales o variables) como sus operandos y devuelve un solo valor numérico. Los operadores aritméticos básicos son suma (+), resta (-), multiplicación (*) y división (/). Estos operadores funcionan como en la mayoría de los otros lenguajes de programación cuando se usan con números de punto flotante, teniendo en cuenta que la división por cero produce un error (tabla 4).

Tabla 4

Operadores aritméticos

Operador	Descripción	Ejemplo
Residuo (%)	**Operador binario. Devuelve el resto entero de dividir los dos operandos.**	**12 % 5 devuelve 2.**
Incremento (++)	**Operador unario. Agrega uno a su operando. Si se usa como operador prefijo (++x), devuelve el valor de su operando después de agregar uno; si se usa como operador sufijo (x++), devuelve el valor de su operando antes de agregar uno.**	**Si x es 3, ++x establece x en 4 y devuelve 4, mientras que x++ devuelve 3 y, solo entonces, establece x en 4.**
Decremento (--)	**Operador unario. Resta uno de su operando. El valor de retorno es análogo al del operador de incremento.**	**Si x es 3, entonces --x establece x e n 2 y devuelve 2, mientras que x-- devuelve 3 y, solo entonces, establece x en 2.**
Negación unaria (-)	**Operador unario. Devuelve la negación de su operando.**	**Si x es 3, entonces -x devuelve -3.**
Positivo unario (+)	**Operador unario. Intenta convertir el operando en un número, si aún no lo es.**	**+"3" devuelve 3. +true devuelve 1.**
Operador de exponenciación (**)**	**Calcula la base a la potencia de exponente, es decir, base exponente**	**2 ** 3 retorna 8. 10 ** -1 retorna 0.1.**

Nota: Tomada de MDN (2021a)

❖ Operadores lógicos

Con los operadores lógicos, se pueden crear condiciones compuestas, por ejemplo, cuando se deben cumplir dos o más condiciones para elegir las operaciones ejecutar; además, se pueden describir estas combinaciones de condiciones (tabla 5 y figura 5).

Tabla 5

Operadores lógicos

Operador	Uso	Descripción
AND Lógico (&&)	**expr1 && expr2**	**Devuelve expr1 si se puede convertir a false; de lo contrario, devuelve expr2. Por lo tanto, cuando se usa con valores booleanos, && devuelve true si ambos operandos son true; de lo contrario, devuelve false.**
OR lógico (||)	**expr1 || expr2**	**Devuelve expr1 si se puede convertir a true; de lo contrario, devuelve expr2. Por lo tanto, cuando se usa con valores booleanos, || devuelve true si alguno de los operandos es true; si ambos son falsos, devuelve false.**
NOT lógico (!)	**!expr**	**Devuelve false si su único operando se puede convertir a true; de lo contrario, devuelve true.**

Nota: Tomada de MDN (2021a)

Figura 5

Ejemplos de operadores lógicos



Se han presentado los operadores más comunes, aunque existen otros tipos de operadores que tal vez no son los más empleados y una referencia completa la puede obtener desde el recurso web denominado *Expresiones y operadores* (MDN, 2021a).

www	La lista completa de los principales operadores de JavaScript se puede revisar en *Expresiones y operadores* (MND, 2021a). https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators

C. Orden de evaluación de operadores

Es importante recordar que los computadores ejecutan los operadores en un orden establecido. El siguiente es el orden (jerarquía), de acuerdo con los expuestos anteriormente, y deben conservar el orden de precedencia de operadores:

1. Paréntesis (se ejecutan primero los más internos).
2. Signo (-2), si un valor es positivo o negativo.
3. Potencias (^) y Raíces y otras funciones (Math.sqrt()); Productos y Divisiones (* y /), en este mismo orden.
4. Sumas y Restas (+ y -).
5. Concatenación (+).
6. Relacionales (=, <, >).
7. Negación (no).
8. Conjunción (y).
9. Disyunción (o).

3.3. Expresiones y comentarios

JavaScript permite poner comentarios en el código fuente, como la mayoría de los lenguajes de programación y existen dos tipos de comentarios en línea que comienzan con una doble barra: //, y los comentarios multilínea, que comienzan con /* y terminan con */ tal como se observa en la siguiente figura.

Figura 6

Tipos de comentarios



Los comentarios son importantes porque le dan mantenibilidad al programa, es decir que otro programador, o el mismo, tiempo después puede revisar el código y apoyarse en los comentarios para saber qué hace el algoritmo y como lo hace.

En adelante se procura que exista al menos un comentario en cada código fuente empleado para ejemplificar los elementos constitutivos del lenguaje de programación JavaScript y puede hacer lo mismo durante la creación de los diferentes códigos.

3.4. Estructuras de selección

La estructura de selección se necesita cuando el código de tu programa ejecuta uno de varios resultados posibles, basado en el valor de una condición.

Ejemplo: un aprendiz aprueba un examen cuando la calificación de este es mayor o igual a 3. Elaborar un programa en JavaScript donde que dada una calificación, aplique el criterio de aprobación e imprima "Aprobado" o "Reprobado", según sea el caso.

Como resultado del **análisis** a este problema tenemos el diseño del siguiente algoritmo:

Figura 7

Seudocódigo calificaciones.

```
ALGORITMO Calificación del usuario;
VAR
    REAL calificacion
```

```
INICIO
   calificacion = 4;
   SI ( calificación >= 3)
            ESCRIBIR( "Aprobo" );
   SINO
            ESCRIBIR( "Reprobo" );
   FINSI
FIN
```

El mismo algoritmo se puede visualizar, pero ya en lenguaje de programación:

Figura 8
Algoritmo calificaciones en JavaScript



Como la línea 1 empieza con /* quiere decir que toda continuación será un comentario hasta que aparezca los caracteres */ (línea 3), en él se debe escribir que hace el algoritmo y otra información que se verá más adelante.

En la línea 4 se define un valor constante (es decir no se modificará más en todo el algoritmo donde definimos) se ha puesto por identificador la palabra calificacion.

Para recordar: una palabra reservada es una palabra que no se puede utilizar como identificador de algún dato.

En la línea 6 ocurren varias cosas que se deben tener en cuenta:

1. Se usa la palabra reservada **if** y entre paréntesis la condición que la se quiere comparar.
2. Se emplea el símbolo corchete abierto **{** se llama abrir corchete.
3. En la línea 7 se envía la instrucción de que imprima el resultado de que aprobó.
4. En la línea 8 se cierra un bloque de código con el símbolo corchete cerrado **}**, esto quiere decir que hasta este punto termina el bloque. A todo lo que esté entre estos corchetes se le llamará CONTEXTO, en el primer contexto están agrupadas las operaciones que se ejecutarán si la condición **calificacion >= 3** es verdadera.
5. También en la línea 8 se oberva la palabra reservada **else**, que sirve para indicarle al intérprete de JavaScript que se creará un bloque o contexto para cuando la condición evaluada por el **if** no se cumpla, y enseguida se abre el contexto con el carácter corchete abierto **{**.
6. Luego, en la línea 9 se ha puesto la operación de escribir en la consola que ha reprobado.
7. Para finalizar el contexto, se cierra con el carácter corchete cerrado **}** .

Ahora se debe escribir el programa y cambiar los valores de la constante para validar que el algoritmo está bien diseñado.

De la misma forma, se puede realizar el algoritmo que, dada una edad en años, evalúe e imprima si es mayor de edad:

Figura 9 ***Escribir en pantalla si es o no mayor de edad***	**De acuerdo con la figura 9, no existe un contexto si no se cumple la condición de mayoría de edad a esta estructura se denomina CONDICIONAL SIMPLE.** **El flujo de ejecución del algoritmo anterior es básico y puede haber otros mucho más complejos como las estructuras de repetición o iterativas que también tienen una sintaxis particular.**



3.5. Estructuras de repetición

Las estructuras de repetición permiten repetir un bloque de instrucciones, determinado o no, un número de veces. A continuación, se ejemplifican los más comunes (*FOR*, *WHILE*) que se pueden codificar también en la sintaxis de JavaScript.

A. Estructura de repetición FOR

Ejemplo 1: escribir un procedimiento que muestre siete (7) veces en pantalla la frase “Esto es un algoritmo”. Mostrar en pantalla la salida del ejercicio:

Figura 9 ***Imprimir 7 veces un mensaje***	**Como se puede observar en la línea 4 de la figura 9, se ha usado la palabra reservada *for* y dentro del paréntesis aparecen tres (3) instrucciones (separadas por punto y coma).** **La primera instrucción *let* i = 0; es la creación e inicialización de una variable con identificador i, como se usó la palabra reservada *let* se está indicado que la variable i solo podrá ser accedida dentro del contexto del *for*.** **Luego se ve una condición i < 7;, esta condición significa que mientras que i tenga valores de 0 a 6 debe ejecutar el bloque o contexto definido en el for (desde el corchete abierto hasta el corchete cerrado).**



Una vez ha terminado de ejecutar el contexto (línea 6) se ejecutará la tercera sentencia de la línea 7 o sea i++, esta línea es equivalente a i = i + 1; incrementar en 1 la variable i, luego de incrementarla procede a evaluar nuevamente la condición y si se cumple volverá a ejecutar las instrucciones que están dentro el contexto (lo que está dentro de paréntesis).

Es de notar que los valores toman la variable i son 0, 1, 2, 3, 4, 5, 6. En total 7 valores el número de veces que se necesita que se repita el bucle.

Para recordar: el carácter punto y coma se usa para decirle al computador, o intérprete, que ha finalizado una instrucción.
La estructura de repetición PARA (FOR) se usa cuando se sabe cuántas veces se debe repetir un grupo de instrucciones.

Ejemplo 2: se necesita elaborar un algoritmo en JavaScript que, dado un número entero, sume todos los números naturales que hay hasta ese número. Por ejemplo, si el usuario digita 3, el programa debe sumar: 1 + 2 + 3, si el usuario digita 5 el programa debe sumar 1+2+3+4+5. Al finalizar, debe imprimir el resultado.

Figura 10
Sumar los números desde 1 hasta un número



Como se puede ver, se decide una constante con valor 5, y una variable que será para el algoritmo un acumulador donde se almacenarán las sumas. También, note que los valores de i van desde 1 hasta exactamente el valor que tiene la constante (es decir 5).

B. Estructura de repetición WHILE

A través del ejemplo del cálculo de factorial se explicará el uso y comportamiento de la sentencia **while**. Recordando el algoritmo diseñado:

Figura 10
Seudocódigo algoritmo cálculo de factorial

```
ALGORITMO cálculo de factorial;
VAR
     ENTERO contador;
     ENTERO factorial;
INICIO
   contador = 1;
   factorial = 1;
   ESCRIBIR( "Escriba el número" );
   LEER(  numero );
   MIENTRAS  contador <= numero  HACER
       factorial = factorial * contador;
       contador = contador + 1;
   FINMIENTRAS
   ESCRIBIR( factorial );
FIN
```

De acuerdo con el anterior algoritmo, el resultante para el cálculo de factorial es:

Figura 11
Seudocódigo algoritmo cálculo de factorial



Es ideal poder transcribir el código usando https://playcode.io/new/, para lo cual se debe escribir la condición del *while,* es decir, **contador <= numero** de último, para que el sistema PlayCode no entre a un ciclo que nunca termina. Una vez funcione correctamente, se propone el código que está entre comentarios (//) para practicar las dos formas de sintaxis.

Es de recordar que las estructuras de repetición básicas y su sintaxis pueden pasar su aplicación más común para recorrer la estructura de datos.

3.6. Estructuras de datos

Las estructuras de datos también conocidas como arreglos, son importantes en el desarrollo de algunos algoritmos son obligatorias a la hora de hacer aplicaciones web o móviles; por ello, se explica su sintaxis en JavaScript para los vectores, matrices y registros.

A. Vectores

Los vectores se definen en JavaScript como un tipo de dato con métodos, y atributos que lo definen. Es ideal analizar lo siguiente:

Figura 12
Definición de un vector



En la línea 1 se ve cómo se define un vector con identificador frutas, de tres (3) elementos, cada uno de ellos es una cadena de texto con una fruta.

En la línea 3 se manda imprimir en la consola todo el vector con su contenido, como se ve en la consola.

En la línea 4 se le llama al atributo frutas.length que contiene el tamaño del vector (es decir 3).

❖ **Acceder a los elementos de un vector**

Los índices de los vectores se comienzan en cero, en otras palabras, el índice del primer elemento de un vector es 0, y el del último elemento es igual al valor de la propiedad *length* del *array* restándole 1. Si se utiliza un número de índice no válido, se obtendrá ***undefined***.

Figura 13
Indexar un vector



Ahora, se debe crear un programa que guarde los resultados de la tabla de 5 en un vector, con estos resultados recorrer el vector e imprimir la tabla del 5:

Figura 15
Crear la tabla del 5



Es ideal revisar la lista de los métodos más comunes en los arreglos:

Tabla 6
Funciones comunes de un vector

Función	Descripción

indexOf()	**Devuelve el índice del primer elemento del *array* que sea igual a elemento Buscado, o -1 si no existe.**
join()	**Concatena en un *string* todos los elementos de un *array*.**
push()	**Añade uno o más elementos al final de un *array* y devuelve el nuevo valor de su propiedad *length*.**
***pop*()**	**Elimina el último elemento de un *array*, y devuelve dicho elemento.**
***sort*()**	**Ordena los elementos de un *array*, modificando este, y devuelve el *array* ordenado.**
***shift*()**	**Elimina el primer elemento de un *array*, y devuelve dicho elemento.**

	**Finalmente, desde el material complementario se encuentra una lista completa o accediendo desde Array - JavaScript (MDN, 2021c). https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions**

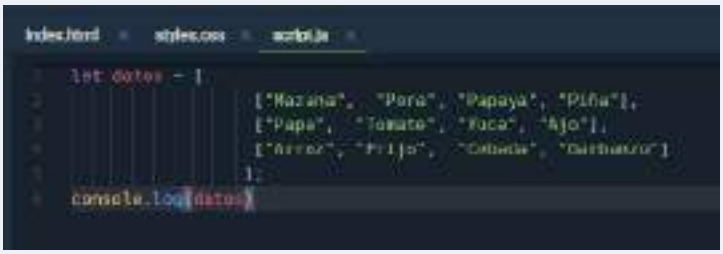
B. Matrices

Las matrices se pueden ver lógicamente como un vector, y cada uno de sus elementos es otro vector, así la representación de la matriz queda codificada como muestra la figura 16:

Figura 16
Crear la tabla del 5

“Mazana”	**“Pera”**	**“Papaya”**	**“Piña”**
“Papa”	**“Tomate”**	**“Yuca”**	**“Ajo”**
“Arroz”	**“Frijo”**	**“Cebada”**	**“Garbanzo”**

Matriz de 3x3



C. Registros

Hasta ahora se ha visto cómo un arreglo es la colección de datos del mismo tipo, pero un registro es una colección de datos de diferente tipo que se relacionan entre sí:

Tabla 7
Registros

Nombre	Correo	Edad	Saldo
Juan	**juan@sindato.com**	**19**	**36.234**
Luis	**luis@sindato.com**	**18**	**23.234**
Andrea	**andrea@sindato.com**	**22**	**0**
Pedro	**pedro@sindato.com**	**12**	**65.234**

Maria	maria@sindato.com	**16**	**123**

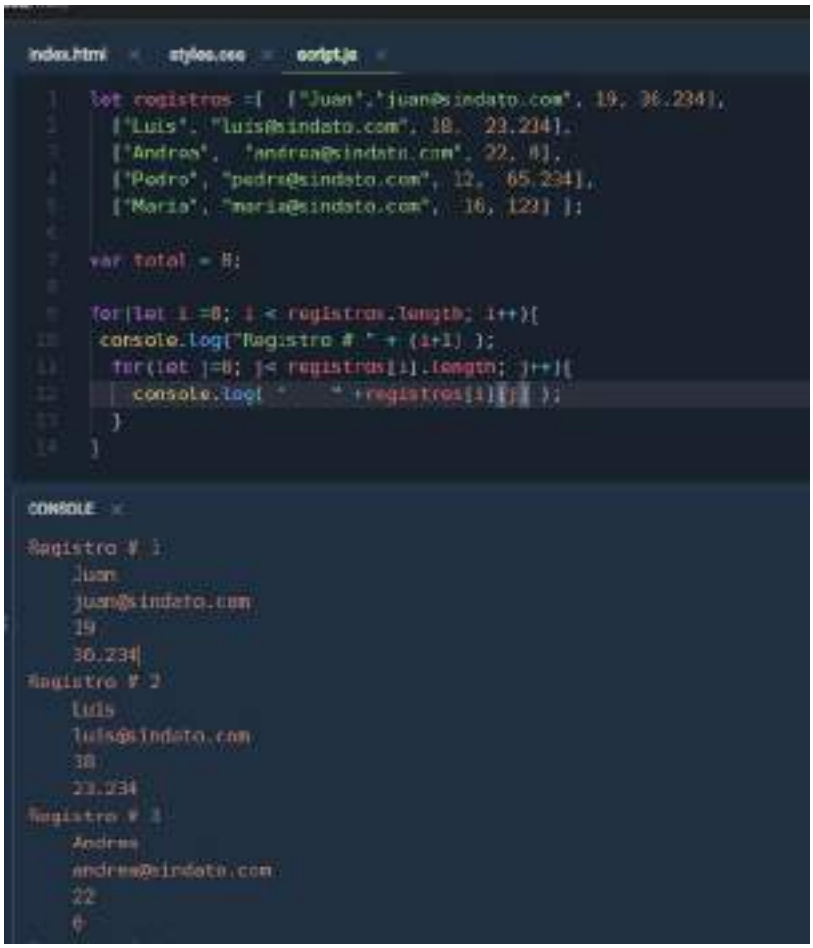
Exceptuando el encabezado de la tabla 7, cada fila representa un registro y cada dato de un registro puede tener un tipo de dato diferente. Si se está interesado en codificar la tabla 7, en una estructura con JavaScript el código sería como se tiene en la figura 17:

Figura 17
Registros



Para recorrer todos los registros se precisan dos (2) ciclos ***for***, uno dentro de otro (anidado) como se muestra a continuación:

Figura 18
Recorrer registros



Como se puede observar en la figura 18, se requieren dos ciclos ***for*** anidados, y cada ciclo tiene una variable de índice de nombre diferente **i** y **j** y la manera de indexar la matriz vista anteriormente en el ejemplo de **registros [i][j]**.

3.7. Estructuras de salto

Estas estructuras son instrucciones que permiten romper la ejecución natural o secuencial de los programas, permitiendo que se salte a otro punto de la ejecución del programa, estas instrucciones tienen las palabras reservadas (*continue, break* y *return*):

- **Sentencia *continue***

La sentencia ***continue*** se salta a la siguiente iteración del ciclo *for* o *while* como se presenta en el siguiente ejemplo y teniendo en cuenta los resultados.

Figura 19
Sentencia continue



Como se observa, cuando la variable **i** tiene el valor 3, se ejecuta la sentencia ***continue*** se salta la sentencia donde debe imprimir el valor de 3, por ese motivo no aparece en la consola.

- **Sentencia *break***

Por otro lado, la sentencia break, detiene la ejecución del ciclo independientemente de cuántas veces este configurando el ciclo. Mire el siguiente ejemplo en la Figura 20, prestando especial atención en la salida de consola.

Figura 20
Sentencia break



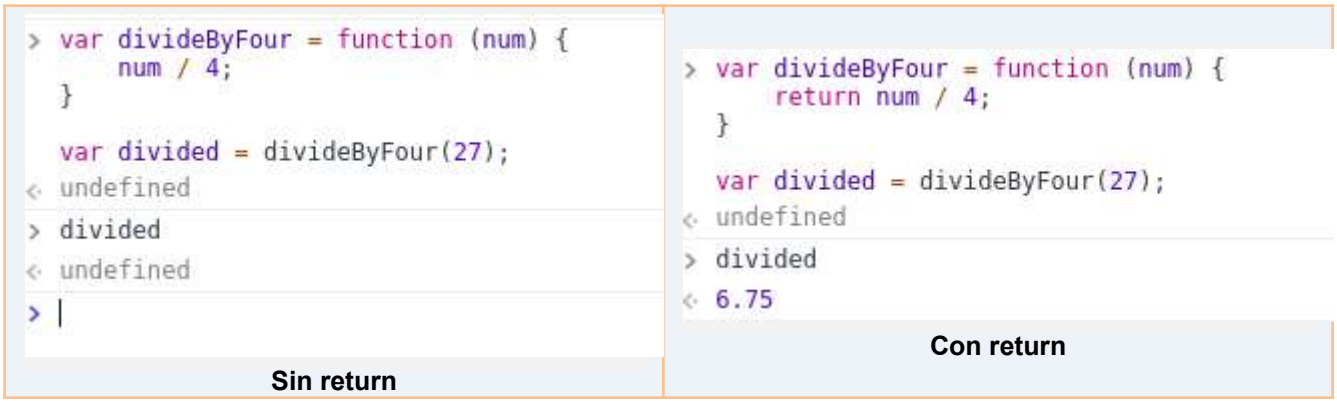
En la anterior figura, cuando **i** tiene el valor de 3 ya no se sigue ejecutando más el ciclo, aunque esté programado para ejecutarse 5 veces, solo ejecuta 3 iteraciones y, ya que en el tercero luego de evaluar la condición **i == 3** ejecuta la sentencia **break** haciendo que termine todo.

- **Sentencia *return***

Sirve para terminar la ejecución de un bloque de instrucciones, se usa mucho en programación modular cuando se quiere retornar un resultado, se usará más adelante.

Por ejemplo, se tiene una función que divide un número entre 4 y se ha creado para obtener resultados. En el primer caso se le pasa 27 para que lo divida entre 4 pero como no hay *return* dentro, devuelve "*undefined*". No accede al número porque el número solo vive dentro de la función y no hay nada que dé el valor. Si se le agrega *return*, efectivamente se obtiene el resultado de la división.

Figura 21
Sentencia return.



Nota. Tomada de Vedia (2018)

3.8. Métodos de ordenamiento y búsqueda

Existen varias técnicas de ordenamiento de vectores las cuales son importantes para realizar búsquedas y, al igual que en la vida real, es más fácil buscar en un lugar que esté ordenado que en uno que no lo esté, como JavaScript está diseñado para pequeños algoritmos que resuelven pequeños problemas y convertirlos en servicios, el uso de esas técnicas no tiene mucha aplicabilidad, pues la diferencia en tiempos computacionales de ejecución es considerable solo cuando se manejan volúmenes de datos grandes, mientras que para volúmenes de datos cortos o pequeños los tiempos computacionales no tienen diferencias comparables. Es por eso que, si en JavaScript el volumen de datos en los arreglos no será grande, no es necesario la aplicación de esas técnicas.

JavaScript provee un *stack* de funciones básicas para realizar el ordenamiento y búsqueda que es muy útil para la implementación de servicios basados en algoritmos.

A. Ordenamiento de cadenas de texto

Se ordenará un vector de mayor a menor valor y de acuerdo con los datos que tiene, se debe tener en cuenta que el arreglo se modifica al ordenarlo, devolviendo la misma matriz ordenada, pero no una nueva. Esto es importante, si se quiere mantener inmutable el vector y obtener otro ordenado, lo que se tendría que hacer es una copia del arreglo antes de ordenarlo.

Por defecto el método *Array*.*sort*() ordena los elementos del *array* como si fueran cadenas inclusive si los datos son de tipo entero, como en este ejemplo:

Figura 22
Ordenamiento de números como cadenas de texto



La figura 22 muestra el arreglo a ordenar en la línea 1, en la línea 2, se muestra en consola el vector, se ha usado la función ***join*(', ')** que sirve para concatenar los datos del vector o arreglo poniendo entre ellos una coma (,) y espacio (línea 2). Luego se aplica la función ***sort*()** en la tercera línea de código y, posteriormente, se muestra el vector ya ordenado. Como se observa, el resultado en consola es un ordenamiento como si los elementos fueran cadenas de texto.

Figura 23
Ordenamiento de cadenas de texto



El ejemplo de la figura anterior muestra un ordenamiento de cadenas de texto. También podemos ordenar el arreglo de forma descendente con el método ***Array.reverse*()** como se muestra en la figura 24.

Figura 24
Invertir el orden de los elementos



La función ***Array.reverse*()** no ordena los elementos, simplemente toma los elementos y les invierte el orden, es decir, el primero pasa a ser el último, el segundo, el penúltimo y así hasta que el último pasa al primer puesto, es por esto que primero se ordena en orden ascendente y luego se invierte para lograr el orden descendente.

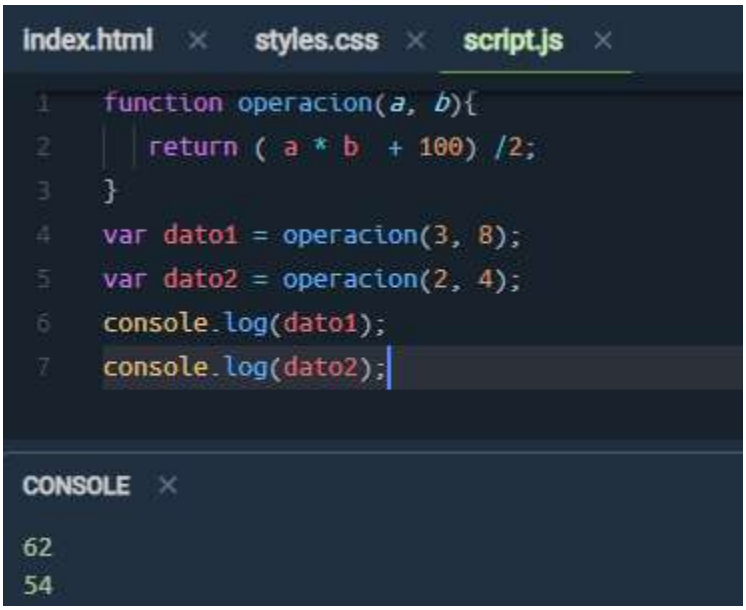
B. Ordenamiento de datos numéricos

Como se ve el ordenamiento con la función ***Array.sort*()** se hace considerando los elementos del arreglo como cadenas de texto. Antes de ver cómo se ordenan con datos numéricos se explicará que es una función en *JavaScript*.

Las funciones son uno de los bloques de construcción fundamentales en JavaScrip*t*. Una función en JavaScript es similar a un procedimiento o “un conjunto de instrucciones que realiza una tarea o calcula un valor, pero para que un procedimiento califique como función, debe tomar alguna entrada y devolver una salida donde hay alguna relación obvia entre la entrada y la salida”. Para usar una función, se debe definir en algún lugar del ámbito en el que se desea llamarla (MDN, 2021d).

Ejemplo: se quiere construir una función que reciba dos (2) datos numéricos, hay que multiplicar estos dos números, al resultado se le suma el valor de 100 y, por último, retorna la mitad de este resultado.

Figura 25
Definición de una función



Como se puede observar en la línea 1 de la figura 25, se usa la palabra reservada ***function*** y luego se escoge identificador para la función que el caso es **operacion**, luego entre paréntesis se seleccionan nombres para las variables de entrada (en el ejemplo **a**, y **b**), a continuación, se define un contexto (espacio de código entre corchetes) para indicar cuáles son las instrucciones que ejecutará la función. En la línea 2 se muestra que la función retorna la multiplicación de a * b sumando 100 y a este resultado dividido entre 2. En la línea 4 se ve como invocamos la función **operacion**, y le pasamos los parámetros 3 y 8 para las variables a y b respectivamente, el resultado es almacenado en la variable **dato1**. Y como se observa la función puede invocarse más de una vez (las que se necesiten).

Para ordenar un vector con datos numéricos se debe pasar como argumento una función al método ***Array.sort*()**, esta función indica qué operación debe hacerse sobre los datos a comparar entre dos datos del arreglo. Para comparar si un número es mayor que otros es muy común usar la resta porque si se resta un número solo existen tres tipos de respuestas. Cuando a un número **a** le restamos un número **b** (**a - b**), si el resultado es negativo es porque **b** es mayor que **a**, si el resultado es positivo porque **a** es mayor que **b** si el resultado es cero solo si **a** y **b** son iguales.

Figura 26
Ordenamiento de números



Como se puede ver en la figura 26 se pasa una función (sin identificador) que recibe dos datos del arreglo (a y b) y los compara con la diferencia (resta) de ellos. A las funciones que no se les pone identificador se les llaman funciones anónimas, también existe otra forma de sintaxis para funciones anónimas, se denomina funciones flecha. Se verá, con la sintaxis de función flecha, cómo ordenar un arreglo número de manera inversa en la figura 27.

Figura 27
Ordenamiento inverso de números



La figura 27 muestra cómo no se usa la palabra reservada ***function***, tampoco se usan la identificación de contexto (corchetes) porque la función tiene una sola línea y se reemplaza por el operador **=>**. Y por defecto se retorna el resultado de la operación **b – a,** que es la diferencia de los dos números, pero en el otro orden (este orden de la operación es el que determina si el ordenamiento es de mayor a menor o viceversa).

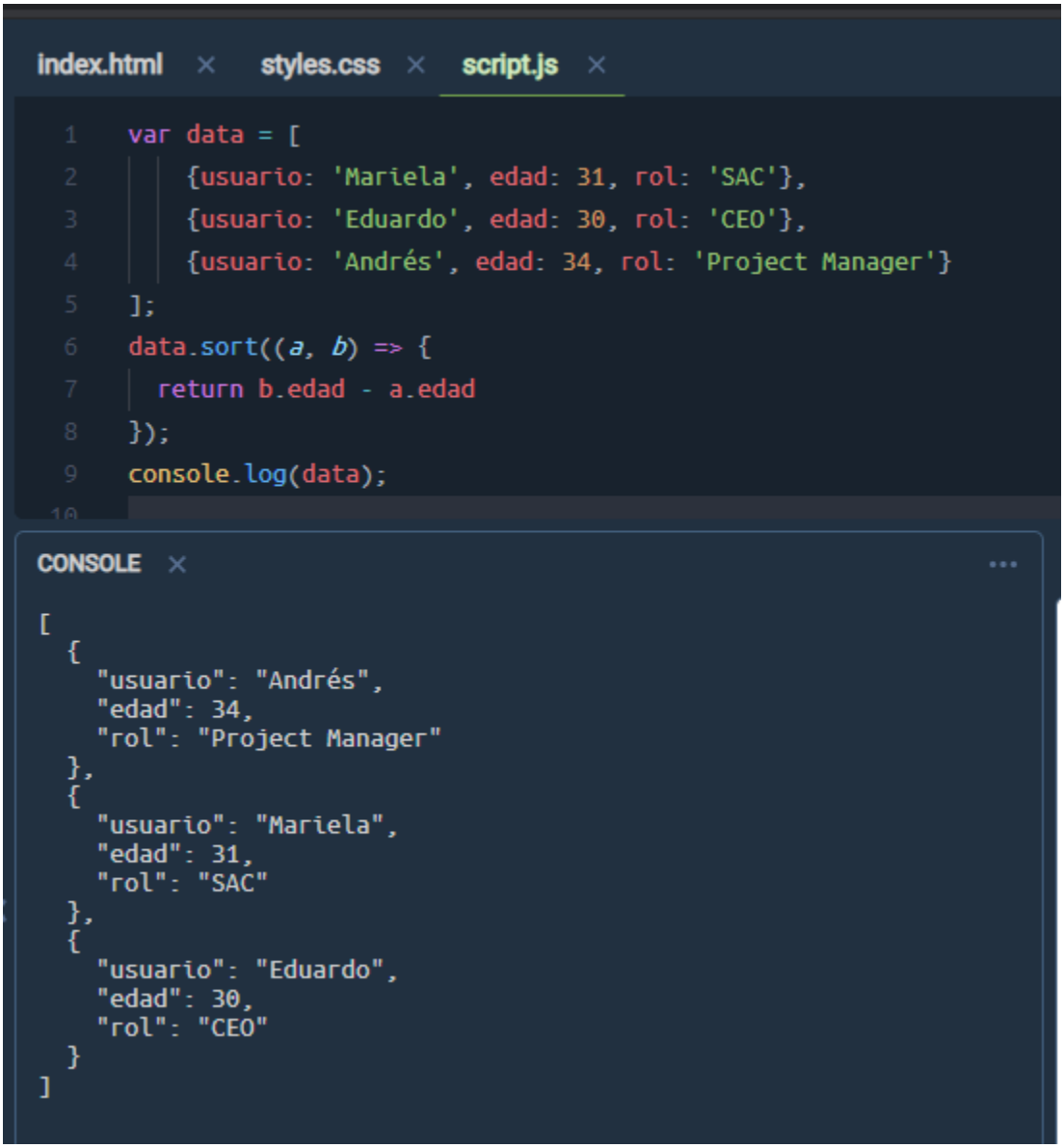
C. Ordenamiento de registros

Suponiendo que se tienen los registros de la tabla, se desea ordenarlos de mayor edad a menor edad.

Tabla 8
Registros a ordenar

Usuario	Edad	Rol
Mariela	31	SAC
Eduardo	30	CEO
Andrés	34	Project Manager

Figura 28
Ordenamiento de registros



En esta oportunidad en la figura 28 se muestra en las líneas 6 y 8, que se puede usar contexto (corchetes) a pesar de que dentro solo hay una sentencia (línea 7), el resultado a retornar es la diferencia entre **b.edad** y **a.edad** para que el ordenamiento sea de mayor a menor.

D. Buscar un elemento

El método *find*() devuelve el valor del primer elemento del *array* que cumple la función de prueba proporcionada (Array.prototype.find() – JavaScript, MDN, 2021).

Figura 29
Buscar un elemento



Como se puede ver en la figura 29, se usa el identificar elemento para referenciar a la variable que se va a aplicar el criterio que en el ejemplo es encontrar el primer elemento que sea mayor que 10.

Muchas veces es importante que, en lugar de retornar un solo valor, el resultado sea más de uno, por ejemplo, si en el arreglo de la figura 30, se retornan los valores que son mayores que 10, por lo tanto, debe retornar otro vector con los valores resultados. Para esto existe la función *Array.*filter(), que crea un nuevo arreglo con los datos que cumplan con la condición.

Figura 30
Buscar varios elementos



Es de interés notar que la consola solo retornó los datos que cumplen la condición que el elemento es mayor que 10.

En la medida que se va avanzando en el conocimiento de la sintaxis del lenguaje, el código fuente se va enriqueciendo de elementos y por tanto de complejidad, esto hace que sea necesario conocer herramientas para la depuración de fallas y corregir los fallos en la redacción del código.

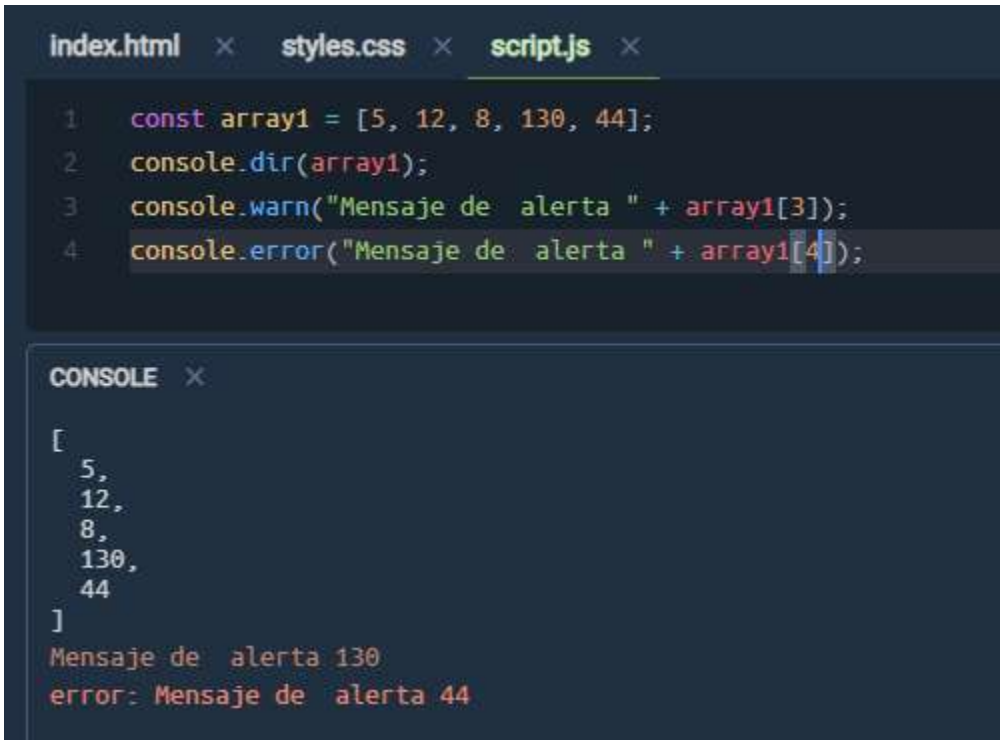
4. Depuración y fallas de sintaxis

La principal funcionalidad que se requiere en la depuración del código, es la mostrar salidas parciales de datos de depuración del estado de las variables, o tal vez se tenga la necesidad de mostrar que se está ejecutando determinada línea de código. Por lo general, la salida de JavaScript está dirigida hacia un navegador o hacia un flujo de datos de un servidor web, dejando la consola para realizar estas depuraciones.

Hasta ahora se ha usado la consola para mostrar datos de depuración cuando se emplea ***console.log()*,** que es un método usado para mostrar mensajes de confirmación de operaciones o mitificaciones de éxito en las operaciones del algoritmo. Pero también existen más:

- ***console.log()*** para enviar y registrar mensajes generales.
- ***console.dir()*** para registrar un objeto y visualizar sus propiedades.
- ***console.warn()*** para registrar mensajes de alerta.
- ***console.error()*** para registrar mensajes de error.

Figura 31
Mensaje de depuración



Se puede revisar el procedimiento de despliegue y uso de consola para el navegador *Google Chrome* en el siguiente enlace https://youtu.be/Hf3n-p3VYx4 (Choque, R. C, 2020).

5. Fallas de lógica

Para realizar las depuraciones y correcciones en la lógica, es preciso realizar la ejecución paso a paso de algunas líneas de código, para poder apreciar los cambios que las variables tienen en el transcurso de la ejecución. A continuación, se indican algunas instrucciones para realizar una revisión paso a paso, cuando la depuración se hace desde un navegador web siguiente los siguientes pasos.

A. Chrome DevTools

Se puede ir a las herramientas de desarrollo (DevTools) pulsando "Command+Option+I" en Mac o "Control+Shift+I" en Windows y Linux para ver el panel de inspección en la parte izquierda de la ventana del navegador. Al inspeccionar el elemento ofrece diferentes opciones para trabajar.

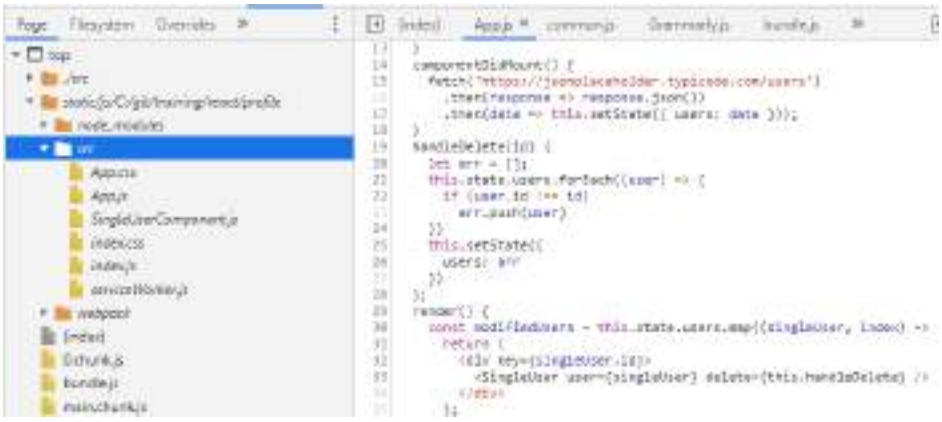
Aparece varias pestañas las opciones presentes para depuración en Chrome son:

- **Console**: la consola se utiliza para imprimir registros, muestra errores y advertencias.
- **Elements**: se pueden seleccionar los elementos para inspeccionar los elementos HTML y hacer estilos en ellos.
- **Sources**: permite navegar los archivos que están presentes en la página cargada y se puede elegir el archivo que se desea consultar.
- **Network**: empleado para inspeccionar todas las llamadas de red que realizan los clientes al servidor, como las cabeceras HTTP, el contenido y el tamaño.

B. Seleccionar fuente (*sources*)

Una vez se tiene que ir a la sección del archivo con el código fuente a depurar, se puede navegar a través de él e identificar la porción de código en la que se sospecha ocurre un error en la lógica.

Figura 32
Seleccionar fuente



C. Marcar punto de interrupción

Un punto de interrupción es una línea de código donde el programa se detendrá. Ya seleccionado el archivo a ser depurado, se puede ver que hay números de línea escritos por cada línea de código. Se debe hacer clic en este número de línea y esta línea actuará como un punto de interrupción y la ejecución se detendrá una vez que el control llegue a esta línea.

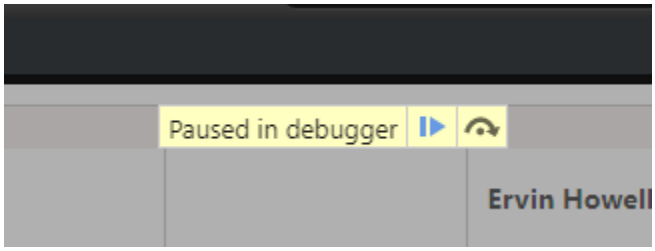
Figura 33
Marcar un break point



D. Avanzar línea a línea

Cuando la ejecución se detiene en el punto de interrupción, se puede reanudar la ejecución haciendo clic en el botón azul que se muestra en la figura 34, así la ejecución avanza línea por línea. Aquí se puede inspeccionar cada línea de código junto con los valores contenidos en las variables.

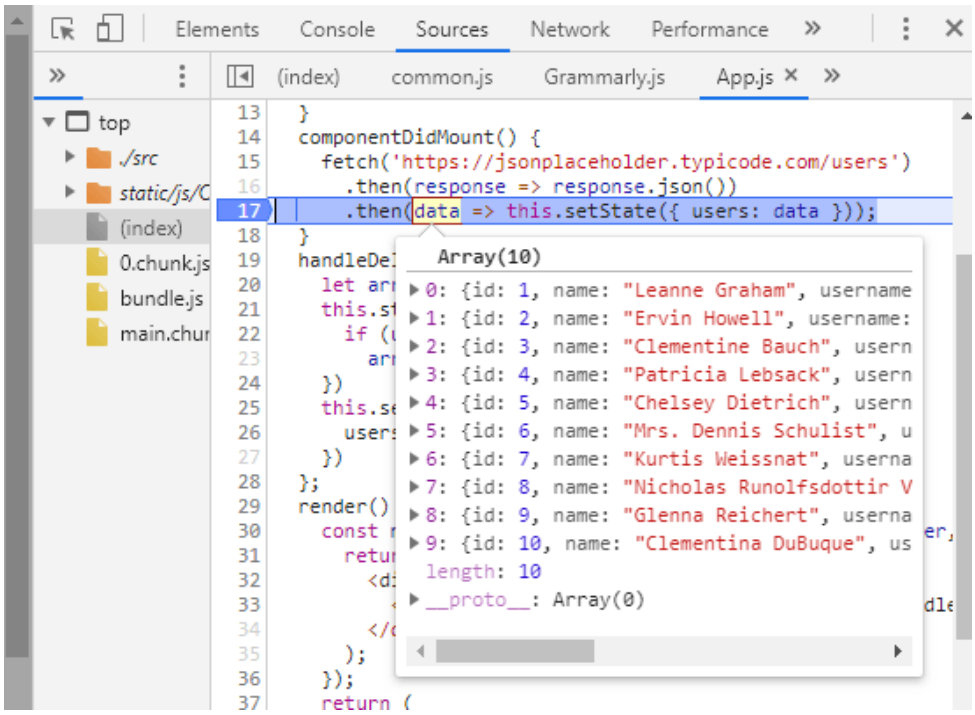
Figura 34
Ejecución línea a línea



E. Comprobar el estado de las variables

El punto más importante es conocer los valores de las variables, esto se hace pasando el ratón por encima de la variable. La principal ventaja de usar puntos de ruptura es que se pueden conocer los valores de más de una variable al mismo tiempo, lo que no es posible en el caso de las consolas de registro, donde se debe escribir *console.log()* para cada variable.

Figura 35
Valores de las variables



www	**Se puede ver un ejemplo en el siguiente enlace: https://youtu.be/l388w3wDkjc (Autodidacta F, 2017).**

6. Manejo de errores y excepciones

Las excepciones son errores imprevistos que ocurren en la ejecución de un programa; son anormalidades que impiden o perturban el comportamiento o el flujo normal de las instrucciones de un programa.

El objetivo principal en el manejo de errores es separar el código de la lógica del programa, del código de manejo de excepciones: de esta forma, al producirse la anomalía, si ocurre un error y existe un manejador de excepciones, él toma el control de la ejecución del programa.

En JavaScript, el control de errores resulta muy necesario tal vez más que en otros lenguajes debido a la dificultad natural de este para testear aplicaciones o los cambios de versión de intérprete en múltiples navegadores.

La forma más sencilla de disparar un error es con la palabra reservada ***throw***, este comando permite enviar al intérprete de JavaScript el evento de que ha ocurrido un error, generalmente permite enviar cualquier tipo de dato, pero lo más común que envíe un “Error” como se muestra en la siguiente figura.

Figura 36

Lanzar una excepción

```
throw new Error( "No se puede dividir por cero" );
```

Existe un tipo de estructura de control en JavaScript que está pendiente de revisar en el flujo de ejecución de un programa para detectar comportamientos inesperados, la estructura es ***try … catch*** que consta de un bloque que intenta la ejecución de una sesión de código (contexto), a la espera de que pueda ocurrir una posible excepción. Finalizado el contexto existe otro denominado ***catch*** que tiene un argumento o variable de contexto, donde se captura el elemento disparado.

Figura 37
Bloque de captura de excepción



En la figura 37 se muestra un algoritmo que divide dos números y, como se sabe, en este proceso se debe validar que no exista una división por 0 al no estar definida aritméticamente.

Para eso existe el contexto ***try*** que va desde la línea 4 a la línea 10, y dentro de él se verifica el caso que el divisor si es igual a 0 debe disparar una excepción. Ella es capturada en la línea 10 por el contexto ***catch*** en la variable **e**, de esta forma, el contexto de catch de la línea 10 a la 13 maneja la excepción, mostrando un mensaje de error y volviendo la variable resultada a nulo. Así, la excepción es lanzada y capturada y el programa no es interrumpido abruptamente, sino que es controlado a pesar de los problemas que se puedan presentar.

FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO

PROGRAMA DE FORMACIÓN	

NÚMERO DEL COMPONENTE FORMATIVO	3
NOMBRE DEL COMPONENTE FORMATIVO	Solución de problemas mediante la utilización de algoritmos
BREVE DESCRIPCIÓN	**Máximo 150 palabras en un solo párrafo**
PALABRAS CLAVE	**Máximo 5 palabras clave**

COMPETENCIA	**220501093** **Crear algoritmos para solución de problemas.**	**RESULTADOS DE APRENDIZAJE**	**220501093 -01** **Definir conceptos de Algoritmos para aplicarlos en la solución de casos problemas**

AREA OCUPACIONAL	**2 - CIENCIAS NATURALES, APLICADAS Y RELACIONADAS**

IDIOMA	**Español**

A. TABLA DE CONTENIDOS:

Introducción

1. Introducción

1.1 Metodología para el desarrollo de Algoritmos.
2. Jerarquía de operadores
2.1 Tipos de Datos
2.2 Datos numéricos
2.3 Datos de tipo número real.
2.4 Datos Alfanuméricos (String)
2.5 Caracteres.
2.6 Cadenas.
3. Operadores y expresiones.
3.1 Aritméticos.
3.2 Relacionales
3.3 Lógicos.
3.4 Datos lógicos o booleanos
4. Jerarquía de Operadores
5. Simbología Grafica de un algoritmo (Diagramas de flujo)
5.1 Diagramas de flujo
5.2 Reglas para la elaboración de diagramas de flujo
6. Qué es LPP
6.1 Programación con LPP
6.2 Ejemplo hola mundo.
6.3 diagrama de flujo y seudocódigo en LPP

B. DESARROLLO DE CONTENIDOS:

Introducción

1. Metodología para el desarrollo de Algoritmos.

Metodología para la solución de un problema mediante el uso de un Algoritmo Definición del Problema Esta fase está dada por el enunciado del problema, el cual requiere una definición clara y precisa. Es importante que se conozca con detalle lo que se desea realizar, mientras esto no se conozca del todo no tiene mucho caso continuar con la siguiente etapa.

Análisis del Problema

Una vez que se ha comprendido lo que se desea realizar, es necesario definir:

Los datos de entrada.
La información de salida que se desea producir

Los métodos y fórmulas que se necesitan para procesar los datos.
Diseño o Elaboración del Algoritmo
Las características de un buen algoritmo son:

Debe tener un punto particular de inicio.
Debe ser definido, no debe permitir dobles interpretaciones.
Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.
Debe ser finito en tamaño y tiempo de ejecución.
Depuración

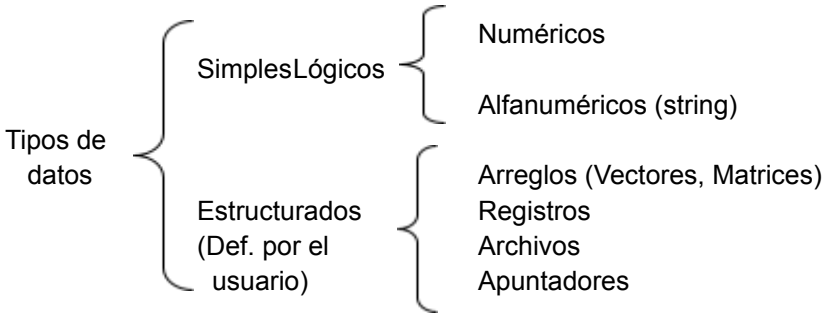
prueba de escritorio es la comprobación que se hace de un algoritmo para saber si cumple con los requerimientos. Esta prueba consiste en tomar datos específicos como entrada y seguir la secuencia indicada en el algoritmo hasta obtener un resultado, el análisis de estos resultados indicará si el algoritmo es correcto o si es necesario corregirlo o hacerle ajustes.

Pseudocódigo
Es la descripción de un algoritmo que asemeja a un lenguaje de programación, pero utilizando algunas convenciones del lenguaje natural. Tiene ventajas con respecto a los diagramas de flujo, requiere menos espacio para su presentación y permite presentar instrucciones complejas. El pseudocódigo no está regido por ningún estándar.

El principal objetivo del pseudocódigo es representar la solución a un algoritmo de la forma más detallada posible y a su vez lo más parecida posible al lenguaje que posteriormente se utilizará para la codificación del mismo.

2. Jerarquía de operadores

2.1 Tipos de Datos



2.2 Datos numéricos

Los datos de tipo numérico son aquellos que representan cantidades o información cuantificable, Datos de tipo número entero**.** Son datos que se expresan mediante un número exacto; es decir, sin componente fraccionario y pueden ser positivos o negativos. Este tipo se utiliza para representar elementos que no pueden encontrarse de forma fraccionada en el dominio del problema como, por ejemplo:

el número de estudiantes de un curso	**25**

el sueldo de un empleado	825344
la edad de una persona	22
el valor de un electrodoméstico	980000
la extensión en kilómetros cuadrados que tiene un país	25680000
la nota de un estudiante	8
La cantidad de empleados de una empresa	200
Las asignaturas que cursa un estudiante	8
El número de goles anotados en un partido de fútbol	4
La cantidad de votos que obtiene un candidato	105314

Es importante anotar que el rango de valores enteros comprendido entre el infinito negativo y el infinito positivo no puede ser manejado en los lenguajes de programación, por razones de almacenamiento, los valores mínimos y máximos a manejar varían dependiendo del lenguaje. Algunos lenguajes de programación reservan dos bytes para los datos de tipo entero, por lo tanto, el rango de valores admitidos está entre -32.768 y 32.767.

2.3 Datos de tipo número real.

Son datos que se expresan mediante un número que incluye una fracción y pueden ser positivos o negativos. Este tipo de datos se utiliza para representar elementos que en el dominio del problema tienen valores formados por una parte entera y una parte decimal, por ejemplo:

La estatura de una persona (expresada en metros)	**1.7**
La temperatura ambiente (en grados centígrados)	18.5
La nota de un estudiante (con base 5.0)	3.5
La tasa de interés mensual	2.6

Los lenguajes de programación reservan un número fijo de bytes para los datos de tipo real, por ello el tamaño de estos números también es limitado.

2.4 Datos Alfanuméricos (String)

Se entiende como datos alfanuméricos aquellos que no representan una cantidad o valor numérico y por ende no se utilizan para cuantificar, sino para describir o cualificar un elemento al que hacen referencia. Es una secuencia de caracteres alfanuméricos que permiten representar valores identificables de forma descriptiva, esto incluye nombres de personas, direcciones, Es posible representar números como alfanuméricos, pero estos pierden su propiedad matemática, es decir no es posible hacer operaciones con ellos. Este tipo de datos se representan encerrados entre comillas.

Los datos alfanuméricos pueden estar formados por caracteres del alfabeto, por números y por otros símbolos; sin embargo, aunque incluyan dígitos no pueden ser operados matemáticamente.

color de una fruta	**rojo**
Año	"2021"

dirección de una casa	cr 77 # 24 – 89
el cargo de un empleado	Gerente
el género	Masculino

2.5 Caracteres.

Los caracteres son cada uno de los símbolos incluidos en un sistema de codificación, pueden ser dígitos, letras del alfabeto y símbolos. El sistema más conocido actualmente es ASCII (*American Standard Code for Information Interchange*). Este requiere un *byte* de memoria para almacenar cada carácter e incluye un total de 256 caracteres.
Son ejemplos de caracteres:

Caracteres
1
a
%
+
B
3

Una tabla que será muy útil para el futuro programador permite identificar los caracteres especiales y muestra cómo escribirlos con la combinación del teclado

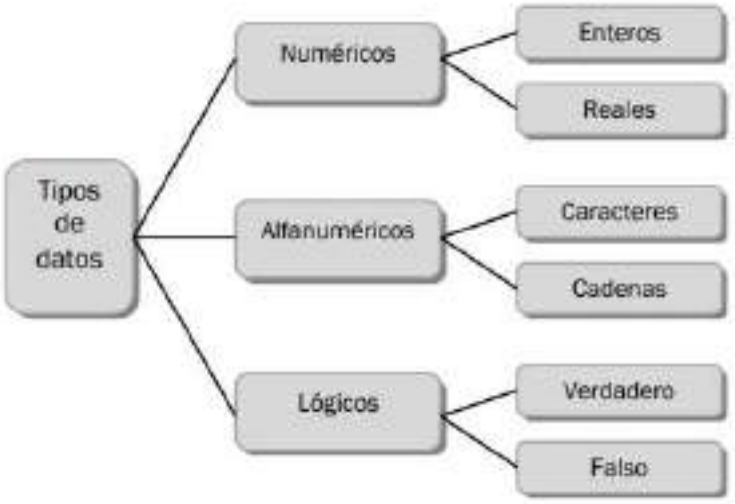


2.6 Cadenas.

Son conjuntos de caracteres encerrados entre comillas (“ “) que son manipulados como un solo dato, por ejemplo:

El nombre de una persona	“José”
La ubicación de una universidad	“Calle 19 con 22”
El título de un libro	“Diseño de algoritmos”

Veremos cómo se realiza el análisis de los problemas dividiéndolos por componentes los principales son:



Numéricos	**Alfanuméricos**	**Lógicos**	**Constantes**
Son aquellos valores que solo pueden contener números, ya sean enteros, fraccionarios, positivos, negativos, reales o imaginarios con la finalidad de realizar operaciones aritméticas comunes.	Es una secuencia de caracteres que pueden ser números, letras o signos, estos pueden ser nombres, direcciones y en comparación con los numéricos, no se pueden realizar operaciones aritméticas con estos.	Son aquellos que solo pueden tener un valor, cierto o falso, ya que representan el resultado de una comparación de valores.	Son datos que su valor no va a cambiar durante todo el desarrollo del algoritmo, las constantes pueden tener o no un identificador

3. Operadores y expresiones.

Son símbolos que tienen una función predefinida (suma, resta, multiplicación etc.) y que recibe sus argumentos de manera infija (luzumisu, s.f.), los operadores más utilizados son:

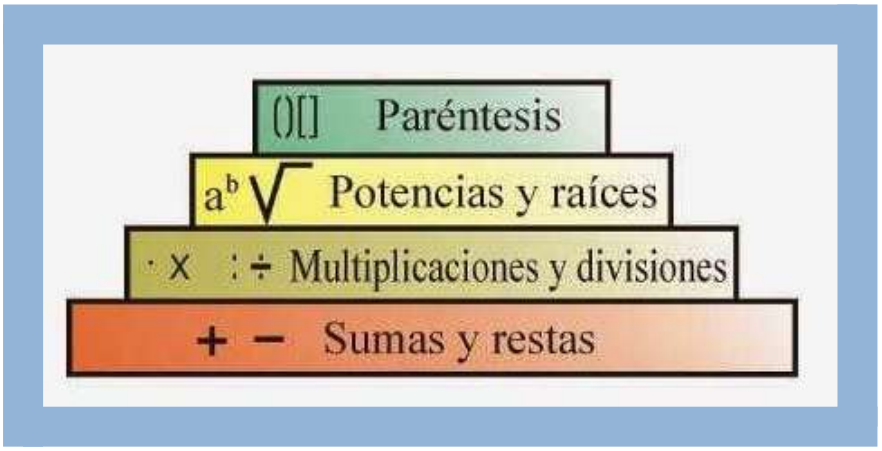
Aritméticos	**Relacionales**	**Lógicos**

Son los operadores que permiten realizar operaciones aritméticas	Estos operadores permiten evaluar la relación entre un par de operando (constantes o variables) y permiten devolver valores de cierto o falso.	Son operadores que permiten conectar un par de propiedades, es decir, nos permiten obtener una respuesta lógica de la comparación de dos operaciones.

3.1 Aritméticos.

Son los operadores que permiten realizar operaciones aritméticas y se enlistan en la siguiente tabla:

Operador	Acción	Ejemplo
+	Suma	X = 5 + 4 | x= 9
-	Resta	X = 10 – 3 | x=7
*	Multiplicación	X = 3 * 2 | x = 6
/	División	X = 10 / 2 | x = 5
%	División Modular	X = 1 / 3 | x = .66
++	Incremento	X = 1; X++ | x = 2
--	Decremento	X = 1; X-- | x = 0
^	Potencia	X = 2 ^ 3 | x = 8



3.2 Relacionales

Estos operadores permiten evaluar la relación entre un par de operando (constantes o variables) y permiten devolver valores de cierto o falso.

Operador	Acción
>	Mayor que
>=	Mayor o igual que
<	Menor que
<=	Menor o igual que

==	Igual / idéntico
¡=	Diferente / distinto

Son operadores que se encargan de unir y comparar dos o más valores, se utilizan en comparaciones de parejas por los símbolos:

== : igual que
!= : diferente a
> : mayor que
< : menor que
>= : mayor igual que
<= : menor igual que

Estos operadores se usan para comparar valores de variables por pares, es decir, no se pueden comparar más de 2 valores al mismo tiempo:
a > b > c //ERROR
(a > b) && (b > c) //BIEN

Operador Relacional	Descripción
=	Compara los dos operandos y evalúa si son iguales
<>	Compara los dos operandos y evalúa si son diferentes
>	Evalúa si el operando izquierdo es mayor que el derecho
<	Evalúa si el operando izquierdo es menor que el derecho
>=	Evalúa si el operando izquierdo es mayor o igual que el derecho
<=	Evalúa si el operando izquierdo es menor o igual que el derecho

Ejemplo	Explicación
x = y	Devuelve verdadero si los valores son iguales y falso si no lo son
x <> y	Devuelve verdadero si los valores no son iguales y falso en caso contrario
x > y	Devuelve verdadero si el primero es mayor que el segundo y falso en caso contrario
x < y	Devuelve verdadero si el primero es menor que el segundo y falso en caso contrario
x >= y	Devuelve verdadero si el primero es mayor o es igual que el segundo y falso en caso contrario
x <= y	Devuelve verdadero si el primero es menor o es igual que el segundo y falso en caso contrario

3.3 Lógicos.

Son operadores que permiten conectar un par de propiedades, es decir, nos permiten obtener una respuesta lógica de la comparación de dos operaciones.

Operador	**Acción**
&&	Conjunción, and, Y
||	Disyunción, or, O
!	Negación, no

Son operadores de unión, también llamados compuertas lógicas, estos operadores pueden unir dos o más pares de valores comparados por medio de los operadores relaciones:

&& : Operador AND (Y) todas las condiciones deben ser verdaderas para que se ejecute una acción.
|| : Operador OR (O) de todas las condiciones solo una debe ser verdadera y con hace determinada acción.
! : Operador NOT (NO) se niega la afirmación para cambiar su valor, es decir cambia de verdadero a falso y de falso a verdadero.

Operador Lógico	Descripción	Ejemplo
AND	Operador lógico AND	x < 7 AND y = 10
OR	Operador lógico OR	x < 7 OR y = 10
NOT	Operador Lógico de negación	NOT x = 10

3.4 Datos lógicos o booleanos

Los datos de este tipo solo pueden tomar dos valores: verdadero o falso. En programación se utilizan para hacer referencia al cumplimiento de determinadas condiciones, por ejemplo: la existencia de un archivo, la validez de un dato, la relación existente entre dos datos.

Operador And

Operando1	*Operador*	*Operando2*	*Resultado*
T	AND	T	T
T		F	F
F		T	F
F		F	F

Recuerde que con and se cumple si todas cumplen.

Ejemplo:

Antioquia gana si Nacional Gana Medellín Gana y Envigado Gana.

Nacional	Medellín	Envigado	Antioquia
G	G	G	G
P	G	G	P
G	P	P	P

Operador Or

Operando1	*Operador*	*Operando2*	*Resultado*
T	OR	T	T
T		F	T
F		T	T
F		F	F

Recuerde que con or se cumple si alguna cumple. Utilice el mismo ejemplo.

Operador Not

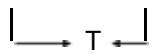
Operando	*Resultado*
T	F
F	T

Ejemplos:

(a < b) and (b < c)

(10<20) and (20<30)

T and T



Prioridad de los Operadores Lógicos

Not

And
Or

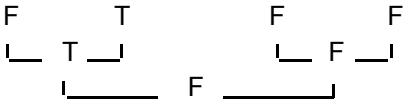
Prioridad de los Operadores en General

1.- ( )
2.- ^
3.- *, /, Mod, Not
4.- +, -, And
5.- >, <, > =, < =, < >, =, Or

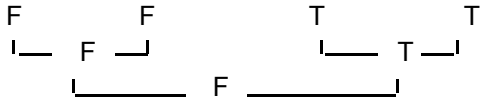
Ejemplos:

a = 10 b = 12 c = 13 d =10

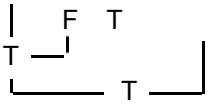
1) ((a > b)or(a < c)) and ((a = c) or (a > = b))



2) ((a > = b) or (a < d)) and ((a > = d) and (c > d))

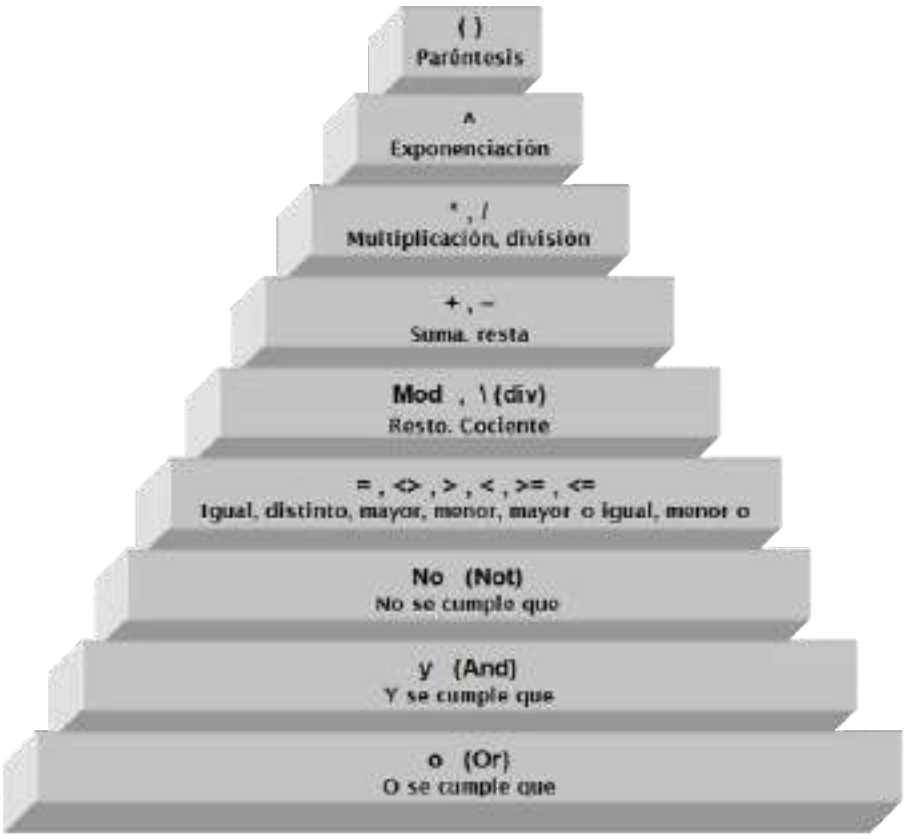


3) not (a = c) and (c > b)



4. Jerarquía de Operadores.

Para realizar operaciones como la anterior, se debe seguir un orden para la realización de operaciones:



Primero se realizan las operaciones que se encuentran ente paréntesis, si hay más de una operación dentro de un paréntesis de aplica el mismo criterio para realizar las operaciones en el orden indicado, posteriormente las potencias y raíces, multiplicaciones y divisiones, sumas y restas en ese orden, si un operador se encuentra dos veces, se realiza la operación que se encuentre primero de izquierda a derecha, posteriormente y también de izquierda a derecha se aplican los operadores relacionales y por último los operadores lógicos.

	Operadores Matemáticos
1	Paréntesis
2	Potencias y Raíces
3	Multiplicaciones y Divisiones
4	Sumas y restas
5	Operadores Relacionales
6	<,>,<=,>=,==,!=
7	Operadores Lógicos
8	Not
9	AND , OR

5. Simbología Grafica de un algoritmo (Diagramas de flujo)

5.1 Diagramas de flujo

La forma más común de representación de los algoritmos es por medio de diagramas de flujo, seudocódigo y lenguaje de programación. Estas representaciones tienen una estructura, una sintaxis y representación estándar definida que permite la comprensión sencilla sin dar cabida a ambigüedades o malos entendidos. Ahora veremos cada una de estas de manera detallada para comprender su estructura y funcionamiento

Un diagrama de flujo es la representación gráfica de un algoritmo. También se puede decir que es la representación detallada en forma gráfica de cómo deben realizarse los pasos en la computadora para producir resultados.
Esta representación gráfica se da cuando varios símbolos, se relacionan entre sí mediante líneas que indican el orden en que se deben ejecutar los procesos. Los símbolos utilizados han sido normalizados por el instituto norteamericano de normalización (ANSI):

Los diagramas de flujo son la representación gráfica del algoritmo de manera visual, se representa gráficamente el paso a paso que muestra la solución al problema por medio de elementos gráficos estandarizados que permiten una fácil comprensión. Esta representación gráfica se da cuando varios símbolos, se relacionan entre sí mediante líneas que indican el orden en que se deben ejecutar los procesos.
Los diagramas de flujo se usan para capacitar a los aprendices en el desarrollo de algoritmos por su facilidad de lectura. Sin embargo, los diagramas de flujo se usan principalmente para representar algoritmos pequeños.

Los principales elementos de un DFD son:
Terminadores: Indican Inicio y Final de un diseño. Se representan con rectángulos de esquinas redondeadas.
Procesos: Contienen las acciones generalmente una o varias asignaciones. Se representan con un rectángulo.
Lectura: Contiene las variables de entrada. Tiene varias representaciones según el dispositivo de entrada. La del teclado es un rectángulo con la esquina superior izquierda cortada.
Escritura: Contiene las variables de salida. Tiene varias representaciones una para pantalla, y otra para impresora.
Flujos: Están representados por flechas que indican la secuencia a seguir.
Conectores: Son pequeños círculos con un número al interior, que permiten mostrar la continuidad de un diagrama, cuando éste se encuentra en varias hojas.

SIMBOLO	NOMBRE	FUNCION

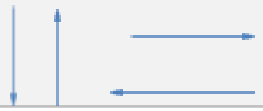
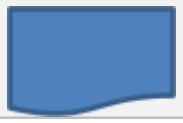
	línea de flujo conexiones de pasos por medio de flechas	muestra la dirección y sentido del flujo del proceso conectando los símbolos
	terminador comienzo o fin del proceso	muestra materiales, información o acciones para comenzar el proceso o para mostrar el resultado en el final
	proceso actividad	tarea o actividad llevada a cabo durante el proceso
	conector conexión con otro proceso	proceso independiente que en algún momento aparece relacionado con el proceso principal
	datos entrada/ salida información de apoyo	información necesaria para alimentar una actividad (datos para realizarla)
	decisión bifurcación	indica puntos en los que se toma decisiones. si o no abierto /cerrado
	documento	hace referencia a la generación o consulta de un documento especifico en un punto del proceso

Figura. Símbolos básicos diagramas de flujo.

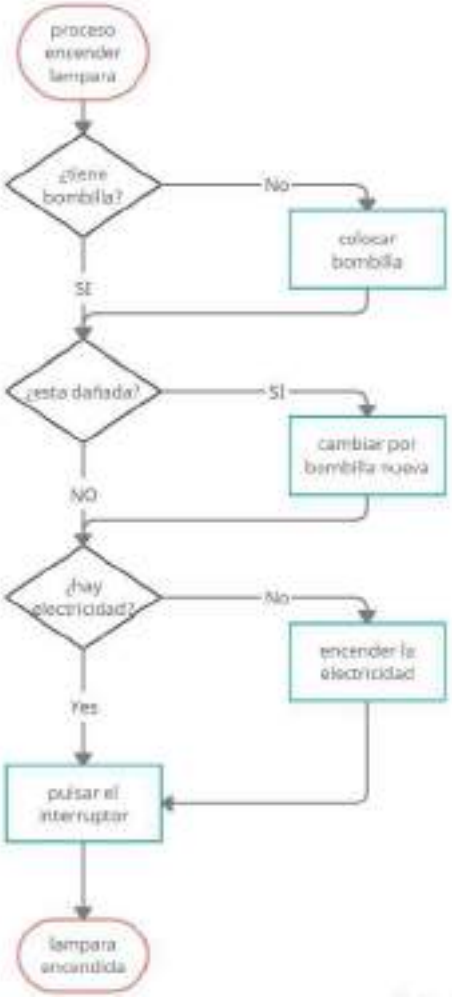
5.2 Reglas para la elaboración de diagramas de flujo

Recomendaciones para el diseño de Diagramas de Flujo

Se deben se usar solamente líneas de flujo horizontales y/o verticales.
Se debe evitar el cruce de líneas utilizando los conectores.
Se deben usar conectores solo cuando sea necesario.
No deben quedar líneas de flujo son conectar.
Se deben trazar los símbolos de manera que se puedan leer de arriba hacia abajo y de izquierda a derecha.
Todo texto escrito dentro de un símbolo deberá ser escrito claramente, evitando el uso de muchas palabras.

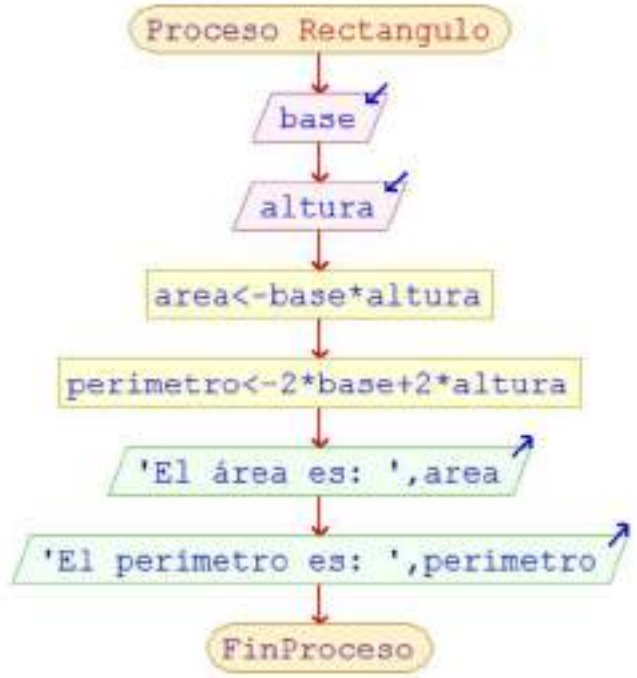


Realizar un diagrama de flujo que permita encender una lámpara



Continuamos trabajando con el ejemplo para calcular el área y perímetro de un rectángulo, el diseño sería el siguiente:

Paso 1: Solicitar datos (base y altura)
Paso 2: Calcular área y perímetro con las fórmulas
Paso 3: Imprimir los resultados (area y perimetro)

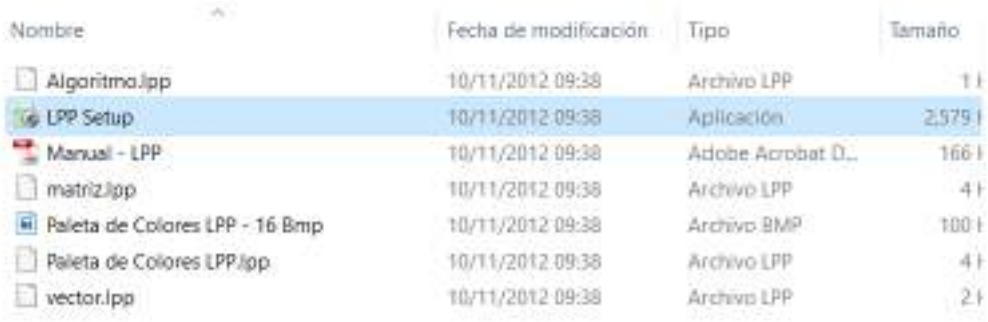


6. Qué es LPP

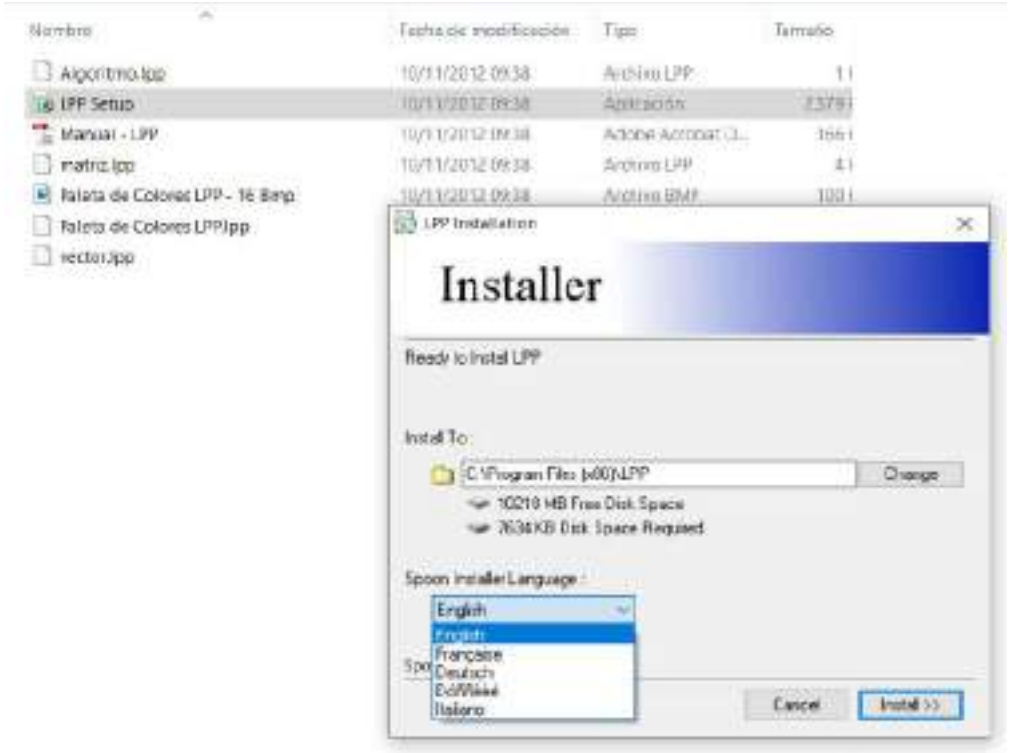
Para realizar nuestras actividades de programación básica iniciaremos con este lenguaje de programación que fue creado como proyecto de graduación del Ingeniero Iván Deras. Lpp es un lenguaje de programación para principiantes, el cual fue diseñado con la idea de facilitar el proceso de enseñanza-aprendizaje de un lenguaje de programación en nuestro idioma, este contiene la mayoría de instrucciones que tienen los lenguajes de programación.

Instalacion pasa a paso del software LPP.

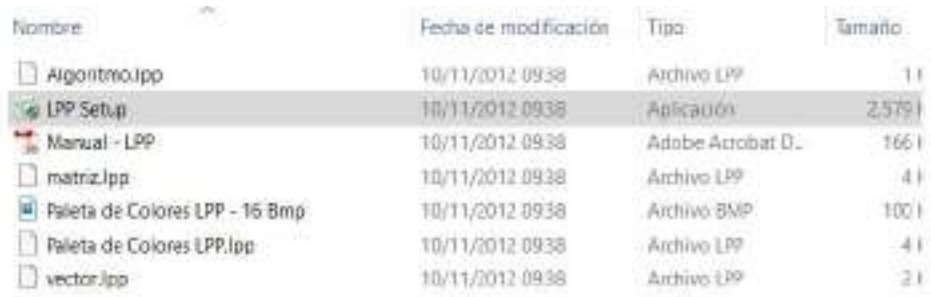
Instalacion pasa a paso del software LPP lenguaje de programación para principiantes:
1.Descargar el archvo comprimido y descomprimir en la carpera buscar el archivo LPP Setup



Se verá una ventana como la siguiente sin realizar modificaciones seleccionar el botón de Install

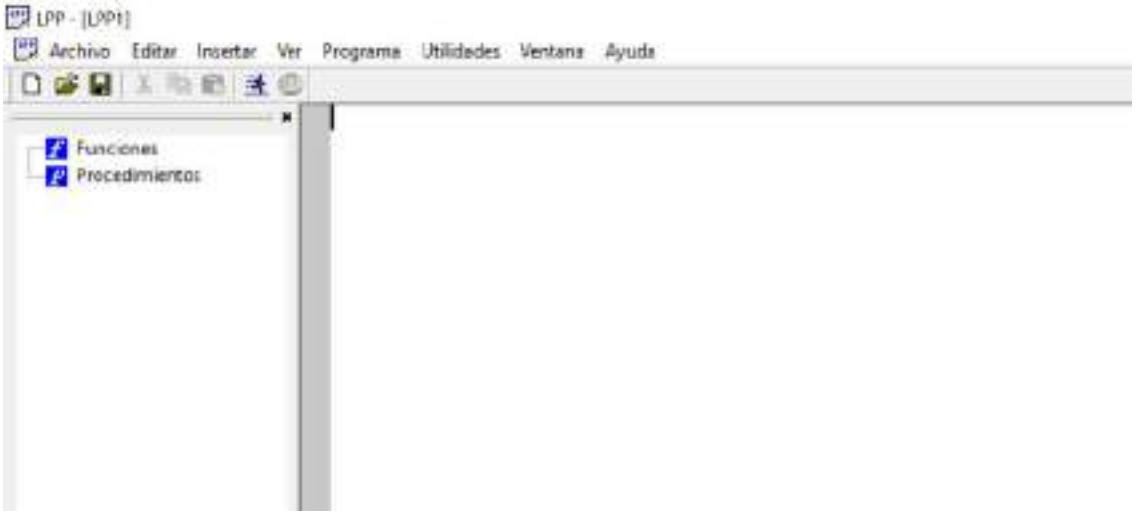


Al terminar la instalación seleccionar la opción de Finish





Y este será el entorno con el cual iniciaremos a trabajar nuestros programas utilizando seudocódigo



Video complementario para realizar la instalación del programa:
[Instalación LPP - YouTube](https://www.youtube.com/watch?v=X_zAV7weBOc&t=2s) https://www.youtube.com/watch?v=X_zAV7weBOc&t=2s

6.1 Programación con LPP

a continuación, dejare una serie de palabras reservadas del lenguaje para principiantes LPP para ir contextualizándonos con el entorno LPP

toda la estructura de código debe tener un 'inicio' y un 'fin'

inicio = para declarar que abrimos el programa
fin = indica que termina el programa
entero = para definir una variable de tipo entero
cadena [25] = para definir una variable con 25 caracteres
lea = para capturar por teclado

escriba = para mostrar en pantalla
si = condición (si)
entonces = siempre va después de la pregunta
sino = negación de la condición (si)
y = condición que obliga
o = condición
// Comentario de línea va al comienzo y al final //
/* Comentario de bloque de texto va al comienzo
Varias líneas y va al final del bloque de texto ***/**

OPERADORES LOGICOS

+ suma
- resta
***** multiplicación
/ división
<- Asignación (siempre se asigna hacia la izquierda) *Nota: Primero se definen las variables

Con esta información que tenemos ya podemos empezar a hacer nuestros primeros programas en LPP vamos a escribir y explicar el funcionamiento con algunos programas sencillos.

Realiza tu primer programa, aprende a utilizar las instrucciones:
inicio. escriba. lea. fin
veremos cómo se declaran y asignan los valores a las variables en LPP

6.2 Ejemplo hola mundo.

Este pequeño ejemplo os permite escribir en pantalla Hola mundo podemos copiar este texto y llevarlo al programa LPP y ejecutarlo o compilarlo  (este es el icono que encontramos en el Programa) para ver su resultado.

```
/*
  ESTO ES UN COMENTARIO:
  En este primer programa vamos a aprender:
    1. Las instrucciones de inicio y fin
    2. Escribir en pantalla “Hola Mundo”
    3. Escribir en pantalla “Hola Mundo” y en otra línea “Este es mi primer programa!”

*/

inicio
escriba "Hola Mundo"
llamar nueva_linea
escriba "Este es mi primer programa!"
fin
```

6.3 diagrama de flujo y seudocódigo en LPP

video de apoyo Operadores Aritméticos
https://www.youtube.com/watch?v=BIHTzKN6Rfw&t=704s

realizaremos el siguiente Problema explicando paso a paso lo que se debe hacer para solucionarlo por medio de un diagrama de flujo y luego convertirlo seudocódigo en LPP

un estudiante del colegio Albert Einstein para conocer su nota final del periodo debe realizar las siguientes operaciones con las notas obtenidas de la siguiente manera

La nota del Saber tendrá un valor de 50%
La nota del Hacer tendrá un valor de 30%
La nota del Ser tendrá un valor de 20%

La nota final será la suma de estos valores

Definimos ahora variables y el tipo de información que manejará que nos servirán para la solución del problema

Datos de entrada		
Nombre	Cadena	Nombre del Estudiante
NotConocimiento	Real	Nota para el Saber
NotHacer	Real	Nota para el Hacer
NotSer	Real	Nota para el Ser

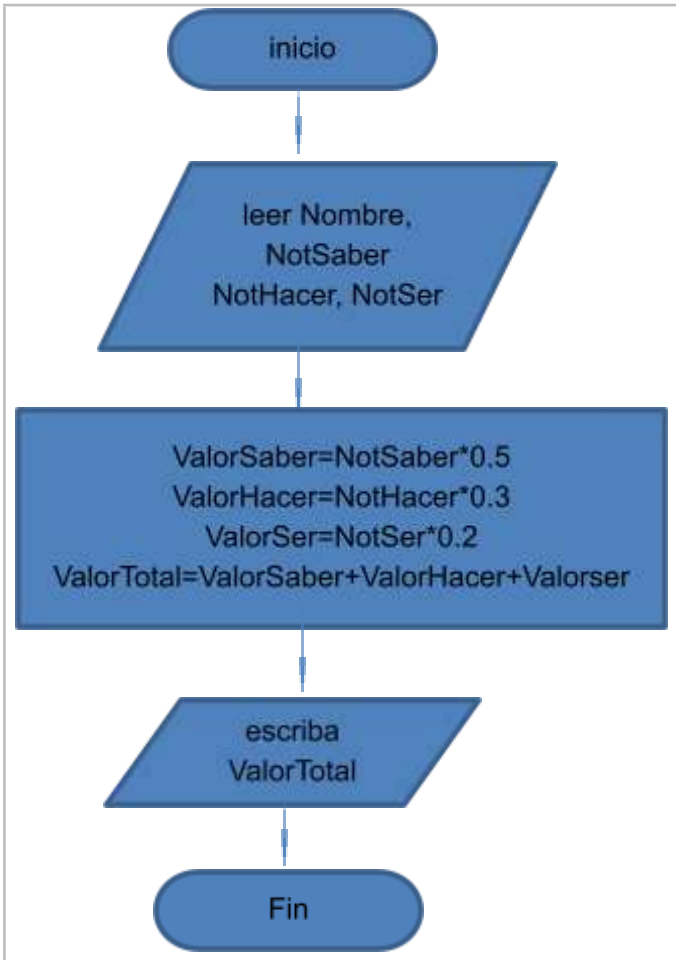
Datos de Salida		
Variable	**Tipo**	**Descripción**
ValorTotal	Real	Valor Total de la Nota Final

Datos de Proceso		
Variable	**Tipo**	**Descripción**
Valor Saber	Real	Valor de la nota Saber
Valor Hacer	Real	Valor de la nota Hacer
ValorSer	Real	Valor de la nota Ser

Procesos

ValorSaber	=	NotSaber * 0.5
ValorHacer	=	NotHacer * 0.3
ValorSer	=	NotSer * 0.2
ValorTotal	=	ValorSaber + ValorHacer + ValorSer

Este sería el análisis identificando los datos de entrada, los datos de salida, los datos del proceso y el proceso para darle solución a este problema
Ahora procedemos a crear el diagrama de flujo.



Inicio de algoritmo
Lectura de datos
Procesos
Resultado
Fin algoritmo

Descripción del paso a paso del seudocódigo

Descripción	Pseudocódigo
Definición de variables	**Cadena[25]** Nombre **Real** NotSaber, NotHacer, NotSer **Real** ValorSaber, ValorHacer, ValorSer, ValorTotal
Inicio de algoritmo	**Inicio**
Lectura de datos	**Escriba** “Digite su nombre” **Lea** Nombre **Escriba** "Digite la Nota de Saber: " **lea** NotSaber **Escriba** "Digite la Nota de Hacer: " **lea** NotHacer **Escriba** "Digite la Nota de Ser: " **lea** NotSer

Procesos	ValorSaber=NotSaber*0.5 ValorHacer=NotHacer*0.3 ValorSer=NotSer*0.2 ValorTotal=ValorSaber+ValorHacer+Valorser
Escritura de resultados	**Escriba** "La Nota Final es: " **Escriba** ValorTotal Fin **Fin**

Para una mejor visualización del código se lo pondrá a continuación para comprobar su funcionamiento se puede copiar pegar y ejecutar en el programa LPP y comprobar su funcionamiento

```
Cadena [25] Nombre
Real NotSaber, NotHacer, NotSer
Real ValorSaber, ValorHacer, ValorSer, ValorTotal


Inicio
        Escriba "Digite su nombre"
        Lea Nombre

        Escriba "Digite la Nota de Saber: "
        lea NotSaber

        Escriba "Digite la Nota de Hacer: "
        lea NotHacer

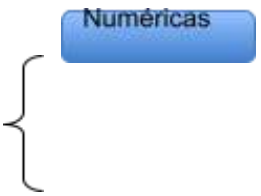
        Escriba "Digite la Nota de Ser: "
        lea NotSer

        ValorSaber <-NotSaber*0.5
        ValorHacer<-NotHacer*0.3
        ValorSer<-NotSer*0.2
        ValorTotal<-ValorSaber+ValorHacer+Valorser


        Escriba "La Nota Final es: "
        Escriba ValorTotal
Fin
```

7. Tipos de variables

Clasificación de las Variables





Las variables pueden ser clasificadas con base a tres criterios: el tipo de dato que guardan, la función que cumplen y el ámbito en el que se utiliza.

De acuerdo al tipo de dato que almacenan, las variables pueden ser de tipo entero, real, carácter, cadena, lógico y de cualquier otro tipo que el lenguaje de programación defina.

En cuanto a la funcionalidad, las variables pueden ser: variables de trabajo, contadores, acumuladores y conmutadores.

Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como 'b', un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.

El ámbito determina el espacio en el que las variables existen, pueden ser globales o locales.

Declaración de variables

Esta operación consiste en reservar en memoria el espacio suficiente para almacenar un dato del tipo especificado y a la vez incluir el identificador en la lista de variables del programa.
Para declarar una variable se escribe el tipo de dato seguido del identificador de la variable, de la forma:

Entero	Real	Cadena
edad	estatura, peso, sueldo	nombre, apellido, dirección

Aunque hay algunos lenguajes de programación que permiten declarar las variables en el momento en que se las necesita, es aconsejable, en favor de los buenos hábitos de programación, siempre declarar las variables antes de utilizarlas y el sitio más adecuado es el inicio del programa o de la función.

Asignación de valores
Esta operación consiste en almacenar un dato en una posición de memoria haciendo uso de una variable previamente declarada. Es necesario que el dato asignado corresponda al tipo para el cual se declaró la variable. Una expresión de asignación tiene la forma:
variable = dato

Ejemplo:

Declaración de variables:
Entero: base, altura
Real: área
Asignación:
base = 10
altura = 5
área = base * altura

Variable Numéricas: Son aquellas en las cuales se almacenan valores numéricos, positivos o negativos, es decir almacenan números del 0 al 9, signos (+ y -) y el punto decimal. Ejemplo:

iva=0.15 pi=3.1416 costo=2500

Variables Lógicas: Son aquellas que solo pueden tener dos valores (cierto o falso) estos representan el resultado de una comparación entre otros datos.

Variables Alfanuméricas: Está formada por caracteres alfanuméricos (letras, números y caracteres especiales). Ejemplo:

letra='a' apellido='lopez' direccion='Av. Libertad #190'

Por su Uso

Variables de Trabajo: Variables que reciben el resultado de una operación matemática completa y que se usan normalmente dentro de un programa. Son las variables que se declaran con el fin de guardar los valores leídos o calculados durante la ejecución del programa.
Ejemplo:

Real: área
Entero: base, altura
Altura = 10
Base = 20
Área = base * altura / 2

Contadores
Son variables que se utilizan para registrar el número de veces que se ejecuta una operación o un grupo de ellas. El uso más frecuente es como variable de control de un ciclo finito, en cuyo caso guardan el número de iteraciones, pero también pueden registrar el número de registros de un archivo o la cantidad de datos que cumplen una condición. Los contadores se incrementan o decrementan con un valor constante, por lo regular es de uno en uno.

Ejemplo: se desea ingresar las notas definitivas de los estudiantes de un grupo para calcular el promedio del mismo. Dado que el tamaño del grupo puede variar, se requiere una variable (contador) para registrar el número de notas ingresadas, para luego poder calcular el promedio.

Acumuladores.

También conocidos como totalizadores. Son variables que se utilizan para almacenar valores que se leen o se calculan repetidas veces. Por ejemplo, si se quisiera calcular el promedio de notas de un grupo de estudiantes, lo primero que se hace es leer las notas y cada una se suma en una variable (acumulador) de modo que después de leer todas las notas se divide la sumatoria de las mismas sobre el número de estudiantes para obtener el promedio.

En este ejemplo se ha hecho referencia a un contador para conocer el número de estudiantes y a un acumulador para sumar las notas. El acumulador se diferencia del contador en que éste no tiene incrementos regulares, sino que se incrementa de acuerdo al valor leído o al resultado de una operación.

Conmutadores.
También se les llama interruptores, *switchs*, banderas, centinelas o *flags*. Son variables que pueden tomar diferentes valores en la ejecución de un programa y dependiendo de dichos valores el programa puede variar la secuencia de instrucciones a ejecutar, es decir, tomar decisiones.

Ejemplo 1: un conmutador puede utilizarse para informarle a cualquier módulo del programa si un determinado archivo ha sido abierto. Para ello se declara la variable de tipo lógico y se le asigna el valor falso y en el momento en que se abre el archivo se cambia a verdadero. Así en cualquier momento que se desee averiguar si el archivo ha sido abierto basta con verificar el estado del conmutador.

Ejemplo 2: si se busca un registro en un archivo se declara una variable de tipo lógico y se inicializa en falso, para indicar que el registro no ha sido encontrado. Se procede a hacer la búsqueda, en el momento en que el registro es localizado se cambia el valor de la variable a verdadero. Al finalizar la búsqueda, la variable (conmutador) informará si el registro fue encontrado o no, dependiendo si su valor es verdadero o falso, y según éste el programa sabrá que instrucciones ejecutar.

Ejemplo 3: es común que un programa esté protegido con un sistema de seguridad. En algunos casos se tienen diferentes niveles de usuarios, donde dependiendo del nivel tiene acceso a determinados módulos. El programa habilitará o deshabilitará ciertas opciones dependiendo del nivel de acceso, el cual estará almacenado en una variable (conmutador).

Locales.

Una variable es de tipo local cuando solo existe dentro del módulo o función en el que fue declarada. Este tipo de variable es útil cuando se aplica el concepto de programación modular, ya que permite que en cada procedimiento o función se tengan sus propias variables sin que entren en conflicto con las declaradas en otros módulos. Por defecto todas las variables son locales al ámbito donde se declaran, si se desea declararlas como globales es necesario especificarlas como tales.

Globales.
Son variables que se declaran para ser utilizadas en cualquier módulo* del programa. Existen desde que se declaran hasta que se termina la ejecución de la aplicación.

Una expresión de asignación tiene tres partes, una variable, el signo igual y el valor o la expresión cuyo resultado se asigna a la variable. La variable siempre va a la izquierda del signo igual, mientras que el valor o la expresión siempre estará a la derecha.

Constantes
Una constate hace referencia a una posición de memoria y se forma de igual manera que una variable, con la diferencia de que el dato almacenado no cambia durante la ejecución del programa.

Las constantes se utilizan para no tener que escribir los mismos valores en diversas partes del programa, en vez de ello se utiliza una referencia al mismo. De manera que cuando se requiera cambiar dicho valor basta con hacer el cambio en una sola línea de código, en la que se definió la constante.
Otro motivo para definir una constante es que resulta más fácil recordar y escribir un identificador que un valor. Por ejemplo, es más fácil escribir *pi* que 3.1416, en especial si este
valor hay que utilizarlo repetidas veces en el programa.

C. ACTIVIDADES DIDÁCTICAS

DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA	
Nombre de la Actividad	Resolución de problemas
Objetivo de la actividad	● Afianzar algunos de los conceptos más importantes para el manejo de la lógica en la resolución de problemas. ● Desarrollo de la creatividad elementos, modelos, fases y sus objetivos.
Tipo de actividad sugerida	Solución de problemas

Archivo de la actividad	Anexo documento en Word llamado Actividad didáctica 3

D. MATERIAL COMPLEMENTARIO:

Este material complementario permitirá reforzar los conceptos sobre la resolución de problemas como base para la contrición

Referencia APA del Material	**Tipo de material** **(Video, capítulo de libro, articulo, otro)**	**Enlace del Recurso o** **Archivo del documento o material**
Curso de Programación para Principiantes o Aprendices - Parte 3 – Operadores Aritméticos	Video	https://www.youtube.com/watch?v=BIHTzKN6Rfw
Curso de Programación para Principiantes o Aprendices - Parte 2 – Escriba - Lea y Variables	video	https://www.youtube.com/watch?v=Up6wnLfhb3k

E. GLOSARIO:

TERMINO	**SIGNIFICADO**

F. REFERENCIAS BIBLIOGRÁFICAS:

AHO, Alfred V.; HOPCROFT, John E.; ULLMAN, Jeffrey D. (1998). Estructuras de datos y algoritmos. México: Addison Wesley.

BRASSARD, G.; BRATLEY, P. (1997). Fundamentos de Algoritmia. Madrid: Prentice-Hall.

COLLADO MACHUCA, M.; MORALES FERNÁNDEZ, R.; MORENO NAVARRO, J. J. (1987). Estructuras de datos. Realización en Pascal. Madrid: Ediciones Díaz de Santos.

GARCÍA MOLINA, J. J.; MONTOYA DATO, F. J.; FERNÁNDEZ ALEMÁN, J. L.; MAJADO ROSALES, M. J. (2005). Una introducción a la programación. Un enfoque algorítmico. Madrid: Thomson-Paraninfo.

JOYANES, L. (1990). Problemas de Metodología de la Programación. Madrid: McGraw-Hill.

JOYANES, L.; ZAHOHERO, I. (2005). Programación en C : metodología, algoritmos y estructura de datos, 2ª Edición. Madrid: McGraw-Hill

G. CONTROL DEL DOCUMENTO

	Nombre	**Cargo**	**Dependencia**	**Fecha**
Autor (es)				

H. CONTROL DE CAMBIOS

(Diligenciar únicamente si realiza ajustes a la Unidad Temática)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA	
Nombre de la Actividad	Resolución de problemas
Objetivo de la actividad	Afianzar algunos de los conceptos más importantes para el manejo de la lógica en la resolución de problemas
Tipo de actividad sugerida	Solución de problemas
Archivo de la actividad (Anexo donde se describe la actividad propuesta)	Anexo documento en Word llamado Actividad didáctica 1

Desarrollo de la actividad:

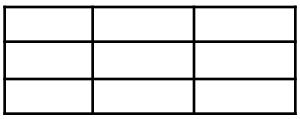
Realice los siguientes retos:

A través de una serie de ejemplos y ejercicios, el aprendiz realiza un acercamiento a las diferentes técnicas de resolución de problemas los aprendices deben ser capaces de utilizar la lógica para resolver problemas específicos, comenzando con los más sencillos, ejercitando su lógica y razonamiento.

A partir del conocimiento adquirido durante el desarrollo de las actividades de apropiación, presente un documento PDF con la respuesta a las siguientes preguntas y situaciones:

Existen muchas clases de problemas como, por ejemplo: matemáticos, estadísticos, lógicos, etc. A continuación, usted encontrara problemas que pretenden despertar la lógica o el fundamento matemático para poder ser resueltos.

1. Ordene los números del 1 al 9 en el cuadro, de manera que la suma en las filas, columnas y diagonal sea igual.

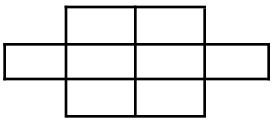


2. Una botella con su tapa cuesta $ 110. Si la botella cuesta $100 más que la tapa. ¿cuánto cuesta la botella?

3. Coloque los signos +, -, *, / entre los números para obtener las igualdades siguientes:
2 2 2 = 6
3 3 3 = 6
5 5 5 = 6
7 7 7 = 6

4. ¿Cuáles son los 5 siguientes números de la serie?
0, 1, 1, 2, 3, ...

5. Ordene los números del 1 al 8 de tal manera que dos números consecutivos no queden unidos en forma horizontal, vertical o diagonal.

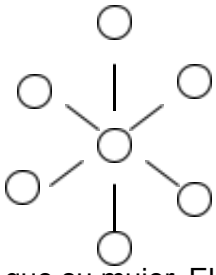


6. Ordene los números del 1 al 7 de forma tal su suma sea 100.

7. Utilice ocho veces los números ocho (8) y el símbolo + cuantas veces sea necesario para obtener como respuesta o resultado el valor mil (1000).

8. Cuantas veces aparece el 9 en los números naturales entre 1 y 100.

9. En la figura coloque las cifras del 1 al 7 de modo que la suma de cualquiera de las filas sea el mismo valor.



10. Un individuo pesa el doble que su mujer. Ella pesa el doble que su hijo y el peso total es de 154 kilos. Cuánto pesa cada miembro de la familia.

11. Cada letra representa un número primo diferente. Hállelos.
a * b * c = 1001

Solución.

1.)

2	9	4
7	5	3
6	1	8

2)

105

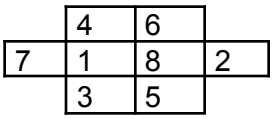
3)

2 + 2 + 2 = 6
3 * 3 - 3 = 6
5 / 5 + 5 = 6
7 - 7 / 7 = 6

4)

0, 1, 1, 2, 3, 5, 8, 13, 21, 34

5)

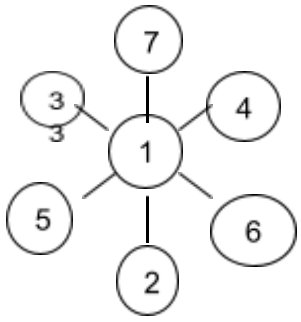


6)
13+56+24+7

7)
888+88+8+8+8

8)
20

9)



10)
88+44+22

11)
7*11*13

PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Desarrollo de aplicaciones informáticas aplicando Inteligencia Artificial
- Código del Programa de Formación:
- Nombre del Proyecto: Construcción de software de integración aplicando inteligencia artificial y tecnologías orientadas a servicios
- Fase del Proyecto (si es formación Titulada): Ejecución
- Actividad de Proyecto (si es formación Titulada): Diseñar la solución de software de acuerdo con procedimientos y requisitos técnicos
- Competencia: Procesar los datos del modelo de inteligencia artificial
- Resultados de Aprendizaje Alcanzar:
    - Crear base de datos con base a los requermientos del cliente.
    - Manipular datos en el Sistema Administrador de Bases de Datos.

Duración de la Guía: 144 Horas

2. PRESENTACIÓN

La información es el bien más preciado de una organización, es por ello que los datos deben estar almacenados en un Sistema Manejador de Bases de Datos, en esta guía nos centraremos en el manejo de Bases de Datos a través de instrucciones SQL.

Para SQL, se realizará la construcción de diferentes bases de datos y tablas mediante utilización de sentencias DDL (Lenguaje de Definición de Datos) y DML (Lenguaje de Manipulación de Bases de Datos), estas sentencias se apicaran específicamente con el gestor de bases de datos MySQL; aunque esté trabajando en plataformas diferentes, la sintaxis básica de SQL es estándar.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

3.1. Actividades de Reflexión inicial.

Vamos a jugar stop por equipos de cinco aprendices, tenga en cuenta las siguientes indicaciones:

- Organice equipos de cinco integrantes
- Cada integrante del equipo debe contar con una hoja y lápiz o lapicero
- Como equipo deben seleccionar cinco temas(palabras), *ej nombre, color, animal, entre otros*Una vez elegidos los temas se sortea una letra entre los jugadores y comienza la ronda
- Por medio de votación de los jugadores, las respuestas de cada tema serán marcadas como validas o invalidadas
- Los puntajes serán los siguientes 10 puntos para cada respuesta valida, 5 para respuestas repetidas y 0 para las invalidas
- Se repite el mismo proceso para cada ronda, se siguieren 4 rondas.

- Al finalizar las 4 rondas realicen diligencien la siguiente tabla:

Letra/ Ronda 1	Palabra 1	Palabra 2	Palabra 3	Palabra 4	Palabra 5
Cantidad de Respuestas Validas					
Cantidad de Respuestas Repetidas					
Cantidad de Respuestas invalidas					
Letra/ Ronda 2	Palabra 1	Palabra 2	Palabra 3	Palabra 4	Palabra 5
Cantidad de Respuestas Validas					
Cantidad de Respuestas Repetidas					
Cantidad de Respuestas invalidas					

Una vez finalizada la recolección de información por equipo, con su instructor construyan una tabla similar con la información de todos los equipos, una vez finalizada la actividad realicen un conversatorio sobre los resultados obtenidos, específicamente los datos coincidentes, valores máximos, mínimos, entre otros.

3.2 Actividades de contextualización e identificación de conocimientos necesarios para el aprendizaje.)

Actividad 1. Fundamentar conceptos para diferenciar Bases de Datos Relacionales y no Relacionales
Tipo de actividad: Individual
Tiempo estimado para la actividad: 10 horas

Tomando como referencia el material suministrado por su instructor y sus investigaciones realizadas, en un documento de texto solucione los siguientes interrogantes:

1. Que tipos de bases de datos existen, mencione sus características, ventajas y desventajas
2. Qué es un Sistema manejador de bases de Datos
3. Cuáles son los principales motores de bases de datos existentes y sus características.
4. Que es un Modelo Entidad Relación y modelo relacional
5. Mencione y explique los tipos de relaciones que existen
6. ¿Para qué sirve la normalización?
7. ¿Qué es la integridad de los datos?

Entregable Documento de texto con la solución a los interrogantes planteados

3.3. Actividades de apropiación del conocimiento (Conceptualización y Teorización).

Actividad 2. Construir Instrucciones SQL para solucionar requerimientos del cliente
Tipo de Actividad: Individual
Tiempo estimado para la actividad: 50 horas

Para esta actividad nos centraremos en el documento “ENUNCIADOS.doc”, recuerde que dicho documento contiene diversos casos empresariales que requieren ser solucionados, para esta actividad su instructor asignará gradualmente los enunciados para que sean realizadas las siguientes actividades:

- Elaboración del DER, (Diagrama Entidad Relación), estos diagramas deben encontrarse en un UNICO documento de texto.
- DDL: Bloc de notas por cada enunciado desarrollado, con el código SQL utilizado para crear la Base de datos y las respectivas tablas
- DML: En los blocs de notas construidos anteriormente, incluir las instrucciones de inserción de TRES registros por tabla(entidad) y las instrucciones construidas para dar solución a las consultas planteadas por su instructor, es decir que por cada enunciado asignado por su instructor se tendrá un documento txt con la creación de las entidades y la inserción de los registros respectivos.

Actividad 3. Construir consultas SQL a partir de enunciados dados
Tipo de Actividad: Individual
Tiempo estimado para la actividad: 20 horas

Lo invito a que revise el documento adjunto denominado “LaboratorioSQL.doc” donde encontrará diversas actividades, las cuales le permitirán construir sentencias SQL para la definición y manipulación del modelo de base de datos.

3.4. Actividades de transferencia del conocimiento.

Actividad 4. Solucionar requerimientos codificando instrucciones SQL
Tipo de Actividad: Individual
Tiempo estimado para la actividad: 15 horas

Estas dos actividades se desarrollarán con control de tiempo simulando una competencia, se plantea la asignación de cada uno de los ítems con periodos específicos de tiempo, al finalizar el primer ítem se valora el resultado y se procede a realizar la siguiente actividad y así sucesivamente.

1. Realice las actividades propuestas en el documento denominado “Actividad3.doc”, ubicado en el espacio actividades.
2. Realice las actividades propuestas en el documento denominado “Actividad4.doc”, ubicado en el espacio actividades.

Actividad 5. Aplicar DML Y DDL al Caso de estudio Alquiler de Motocicletas
Tipo de Actividad: Por equipos de Desarrollo
Tiempo estimado para la actividad: 50 horas

Tomando como referencia el documento denominado “CasodeEstudioAlquilerMotocicletas” ubicado en la carpeta Actividades/ 4_Transferencia, realicen las siguientes actividades en documento de texto:

1. Obtención de los datos Relevantes
2. Elaboración del diagrama Entidad Relación
3. Elaboración del Diagrama Entidad Relación
4. Relación de las tablas con sus respectivos campos
5. Implementación de la base de datos y tablas en MYSQL
6. Implementación del proyecto, colección y documentos en firebase

4. ACTIVIDADES DE EVALUACIÓN

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento: **Fundamentar conceptos para diferenciar Bases de Datos Relacionales y no Relacionales:** Documento de texto con la solución a los interrogantes planteados **Evidencias de Desempeño** **Solucionar requerimientos codificando instrucciones SQL:** Documento de Word incluya la instrucción utilizada junto con la respuesta arrojada por el sistema. **Evidencias de Producto:** **Construir Instrucciones SQL para solucionar requerimientos del cliente:** UNICO documento de texto con el DER, (Diagrama Entidad Relación) de los requerimientos planteados. Bloc de notas por cada requerimiento planteado con las instrucciones DDL y DML utilizadas. **Construir consultas SQL a partir de enunciados dados:** Documento de texto con nombre “Solucion_Laboratorio_SQL” y Archivo txt con el nombre “Script_Laboratorio_SQL”.	Crea bases de datos teniendo en cuenta lenguaje de definición de datos, requerimientos del sistema y tecnología seleccionada. Realiza operaciones sobre los objetos de la base de datos aplicando las instrucciones SQL.	**Lista de verificación**

Aplicar DML Y DDL al Caso de estudio Alquiler de Motocicletas: Documento de texto con la solución a las actividades planteadas		

5. GLOSARIO DE TÉRMINOS

Administración de Base de Datos Relacional: Técnica de administración de bases de datos en la que la información se guarda en tablas de datos bidimensionales. El programa que funciona con dos tablas de datos al mismo tiempo relaciona la información mediante vinculaciones establecidas por una columna o campos comunes.

Acceso: La manera en la cual los archivos o conjunto de datos son referenciados por la computadora.

Base de Datos: Es una serie de datos organizados y relacionados entre sí, los cuales son recolectados y explotados por los Sistemas de Información de una empresa o negocio en particular.

Campo o atributo: Unidad básica de una base de datos, un campo puede ser, por ejemplo, el nombre de una persona. Los nombres de los campos no pueden empezar con espacios en blanco y caracteres especiales. No pueden llevar puntos, ni signos de exclamación o corchetes. Si pueden tener espacios en blanco en el medio.

Clave candidata: Conjunto de atributos de una tabla que identifican unívocamente cada tupla de la tabla.

Clave primaria: Clave candidata que se escoge como identificador de las tuplas.

Clave alternativa: Cualquier clave candidata que no sea primaria.

Clave externa o secundaria: Atributo de una tabla relacionado con una clave de otra tabla

Datos estadísticos: estos almacenan información estadística sobre los datos en la base de datos.

Dominio: El dominio de un campo es el conjunto de valores aceptables que puede contener. Por ejemplo, el dominio del campo sexo estará ajustado a los dos valores que puede contener este dato, es decir, femenino y masculino.

Esquema: Es la aplicación de un Modelo de Datos a la realidad, esa representación puede ser en forma gráfica o lingüística y va a representar los objetos, sus interrelaciones, las operaciones que se pueden realizar sobre ellos y las restricciones que afecten al sistema. Resume las características más importantes del sistema de información para el objetivo que se persigue.

Entidad: Es la representación de un objeto o concepto del mundo real que se describe en una base de datos.

Foreing Key: En el contexto de bases de datos relacionales una clave foránea o clave ajena (o foreing key FK) es una limitación referencial entre dos tablas.

El DBMS: es un conjunto de programas que se encargan de manejar la creación y todos los accesos a las bases de datos.

ELIMINACION: Es una solicitud de eliminación que se expresa de forma muy parecida a una consulta. Sin embargo, en vez de presentar tuplas al usuario, quitamos las tuplas seleccionadas de la base de datos. Sólo puede eliminar tuplas completas; no se puede eliminar únicamente valores de determinados atributos.

Facilidad de Consultas: Permitir al usuario hacer cuestiones sencillas a la base de datos. Este tipo de consultas tienen como misión proporcionar la información solicitada por el usuario de una forma correcta y rápida.

Gestor de Base de Datos: Es un conjunto de programas que permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad

Independencia de los datos: Se refiere a la protección contra los programas de aplicaciones que pueden originar modificaciones cuando se altera la organización física y lógica de las bases de datos.

La manipulación de datos: la manipulación de datos es la recuperación de información almacenada en la base de datos

Lenguaje de consulta: Son los lenguajes en el que los usuarios solicitan información de la base de datos. Estos lenguajes son generalmente de más alto nivel que los lenguajes de programación. Los lenguajes de consulta pueden clasificarse como procedimentales y no procedimentales

Manipulación de Base de Datos: Usando la base de Datos -- el usuario puede añadir, borrar y modificar información a la base de datos así como también hacer consultas

Modelo de base de datos orientado a objetos: es una adaptación a los sistemas de bases de datos. Se basa en el concepto de encapsulamiento de datos y código que opera sobre estos en un objeto.

Modelos de Red: Este modelo permite la representación de muchos a muchos de una Base de Datos. El modelo de red evita redundancia en la información, a través de la incorporación de un tipo de registro denominado el conector.

Nivel lógico: definición de las estructuras de datos que constituyen la base de datos

Reglas de Integridad: Son restricciones que definen los estados de consistencias de las bases de datos.

DBMS: Es un conjunto de programas que se encarga de manejar la cremación y todos los acceso a las bases de datos.

DDL: Es un lenguaje de definición de datos

Recuperación: Proporcionar como mínimo el mismo nivel de recuperación que los sistemas de bases de datos actuales. De forma que, tanto en caso de fallo de hardware como de fallo de software, el sistema pueda retroceder hasta un estado coherente de los datos.

Relaciones: Las relaciones son conexiones lógicas que existen entre las tablas y los atributos y se pueden clasificar por diferentes criterios.

Sistema de Administración de Base de Datos: Es el software que controla la organización, almacenamiento, recuperación, seguridad e integridad de los datos en una base de datos

SISTEMA GESTOR DE BASE DE DATOS: Es un conjunto de programas que permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad.

Transacción: es una secuencia de operaciones de acceso a la base de datos que constituye una unidad lógica de ejecución.

Transacciones compartidas: Las transacciones compartidas soportan grupos de usuarios en estaciones de trabajo, los cuales desean coordinar sus esfuerzos en tiempo real, los usuarios pueden compartir los resultados intermedios de una base de datos. La transacción compartida permite que varias personas intervengan en una sola transacción.

Tuplas: Es la representación de una fila en una de las tablas que se está almacenando datos. Y las cuales serán llamadas por los administradores de Base de Datos en el tiempo de ejecución de un sistema.

Usuario final: es quien accesa a las bases de datos por medio de un lenguaje de consulta o de programas de aplicación.

6. REFERENTES BILBIOGRÁFICOS

- Abraham Silberschatz, et. Al. Fundamentos de Bases de Datos. 4ª Edición. 2004.
- Sánchez, J. Principios sobre bases de datos. http://www.jorgesanchez.net/bd/bdrelacional.pdf
- Pons, O. et al. Introducción a los sistemas de bases de datos. Paraninfo. 2008.
- Elmasri, R., Navathe, S.B., Fundamentos de Sistemas de Bases de Datos, 5ª edición, Pearson Education, 2008.
- https://dev.mysql.com/doc/refman/8.0/en/
- Reglas de normalización: http://www.youtube.com/watch?v=p_UGm8UYaew
- http://www.youtube.com/watch?v=n8PMCxxuXr4
- http://www.youtube.com/watch?v=TSeN_BtuEKM
- Craig Larman. UML y patrones, editorial Prentice Hall, 2007.
- G. Booch, J. Rumbaugh, I. Jacobson. El lenguaje unificado de modelado: manual de referencia. Addison-Wesley, 1999.
- James Rumbaugh. Object Oriented Modeling and Design. Pearson Education, 2002
- Schmuller, Joseph. Aprendiendo UML en 24 horas. Prentice Hall
- Scott W. Ambler. Introducción a los Objetos, 3ra. edición, 2003
  Disponible en la Biblioteca digital SENA (Solo está autorizado y disponible desde la red del SENA)
- Kimmel, Paul. Manual de UML. Mexico: McGraw-Hill España, 2010. p i.
- Ejercicios Realización de consultas SQL . https://josejuansanchez.org/bd/ejercicios-consultas-sql/

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)				

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

1. **Cree una base de datos llamada Aseguramiento**
2. **En la base de datos aseguramiento cree las siguientes tablas**

Compania	Tipo	Longitud
nitCompania (pk)	Varchar	11
nomCompania	Varchar	30
fecFunCompania	Date	
repLegCompania	Varchar	100
TipoAutomotor	Tipo	Longitud
conTipoAutomotor (pk)	int autoincrementable	
nomTipoAutomotor	Varchar	30
Aseguramiento	Tipo	Longitud
codAseguramiento (pk)	Int autoincrementable	
plaAseguramiento (fk)	Varchar	6
fecIniAseguramiento	Date	
fecFinAseguramiento	Date	
valAseAseguramiento	Int	
estAseguramiento	Varchar	7
costAseguramiento	Int	

Automotor	Tipo	Longitud
plaAutomotor (pk)	Varchar	6
tipAutomotor (fk)	Int	
marAutomotor	Varchar	30

modAutomotor	Int	
numPasAutomotor	Int	
cilAutomotor	Int	
numChaAutomotor	Varchar	20

Incidente	Tipo	Longitud
codIncidente (pk)	Int autoincrementable	
fecIncidente	Date	
plaIncidente (fk)	varchar	6
lugIncidente	varchar	40
canHerIncidente	Int	
canFatIncidente	Int	
canAutInvIncidente	Int	

3. En cada una de las tablas, inserte los siguientes registros:

Tabla: TipoAutomotor		
	conTipoAutomotor	nomTipoAutomotor
Registro 1	1	Automóviles
Registro 2	2	Camperos
Registro 3	3	Camiones

Tabla: Compania				
	nitCompania	nomCompania	fecFunCompania	repLegCompania

Registro 1	800890890-2	Seguros Atlantida	1998	Carlos López
Registro 2	899999999-1	Aseguradora Rojas	1991	Luis Fernando Rojas
Registro 3	899999999-5	Seguros del Estadio	2001	María Margarita Pérez

Tabla: Automotor

	plaAutomotor	marAutomotor	tipAutomotor	modAutomotor	numPasAutomotor	cilAutomotor	numChaAutomotor
Registro 1	FLL420	chevrolet corsa	1	2003	5	1.400	wywzzz167kk009d25
Registro 2	DKZ820	renault stepway	1	2008	5	1600	wywwzz157kk009d45
Registro 3	KJQ920	kia sportage	2	2009	7	2000	wywzzz157kk009d25

Tabla: Aseguramiento

	codAseguramiento	fecIniAseguramiento	fecFinAseguramiento	valAseAseguramiento	estAseguramiento	costAseguramiento	plaAseguramiento
Registro 1	1	2023-09-30	2024-09-30	30.000.000	Vigente	500.000	FLL420
Registro 2	2	2023-09-27	2024-09-27	35.000.000	Vigente	600.000	DKZ820
Registro 3	3	2023-09-28	2024-09-28	50.000.000	Vencido	800.000	KJQ920

Tabla: Incidentes

	codIncidente	fecIncidente	Pla Incidente	lugIncidente	canHerIncidente	can FatIncidente	canAutInvIncidente
Registro 1	1	2023-09-30	DKZ820	Bucaramanga	0	0	2
Registro 2	2	2023-09-30	FLL420	Girón	1	0	1
Registro 3	3	2023-09-28	FLL420	Bucaramanga	1	0	2

4. Realice las siguientes consultas

1) Visualizar los datos de las empresas fundadas entre el año 1991 y 1998
2) Listar los todos datos de los automotores cuya póliza expira en octubre de 2023, este reporte debe visualizar la placa, el modelo, la marca, número de pasajeros, cilindraje nombre de automotor, el valor de la póliza y el valor asegurado.
3) Visualizar los datos de los incidentes ocurridos el 30 de septiembre de 2023, con su respectivo número de póliza, fecha de inicio de la póliza, valor asegurado y valor de la póliza.
4) Visualizar los datos de los incidentes que han tenido un(1) herido, este reporte debe visualizar la placa del automotor, con los respectivos datos de la póliza como son fecha de inicio, valor, estado y valor asegurado
5) Visualizar todos los datos de la póliza más costosa.
6) Visualizar los incidentes con el mínimo número de autos involucrados, de este incidente visualizar el estado de la póliza y el valor asegurado.
7) Visualizar los incidentes del vehículo con placas " FLL420", este reporte debe visualizar la fecha, el lugar, la cantidad de heridos del incidente, la fecha de inicio la de expiración de la póliza y el valor asegurado.
8) Visualizar los datos de la empresa con nit 899999999-5

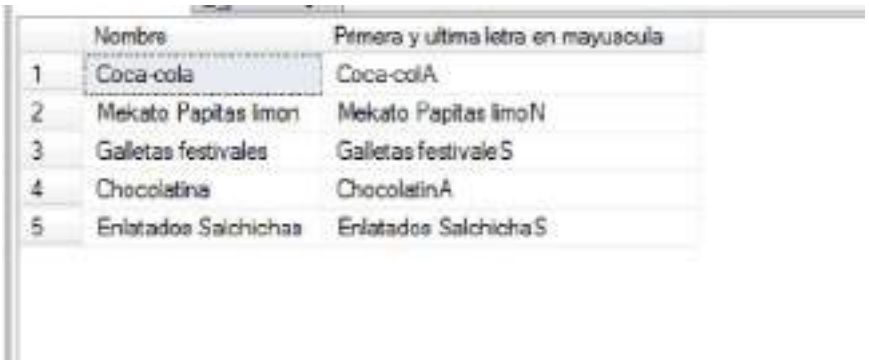
9) Visualizar los datos de la póliza cuyo valor asegurado es el más costoso, este reporte además de visualizar todos los datos de la póliza, debe presentar todos los datos del vehículo que tiene dicha póliza.
10) Visualizar los datos de las pólizas de los automotores tipo 1, este reporte debe incluir placa, marca, modelo, cilindraje del vehículo junto con la fecha de inicio, de finalización y estado de la póliza

Evidencias a Entregar

Documento en formato txt con el nombre Solucion_Laboratorio_SQL que contenga el código SQL para la construcción y manipulación de las Bases de Datos solicitadas en el presente laboratorio y que permita obtener los resultados solicitados.

Tomando como referente las entidades y los datos presentados en el documento de Excel denominado “entidadesActividad4” realice las siguientes actividades:

1. Construya una base de datos denominada “SQLActividad4”
2. Cree cada una de las entidades(tablas) propuestas, tenga en cuenta realizar el análisis correspondiente e identificar las llaves primarias y foráneas en cada una de las entidades
3. Inserte los registros propuestos en la hoja “datos”, del documento “entidadesActividad4”
4. Realice las siguientes consultas, y en un documento de Word incluya la instrucción utilizada junto con la respuesta arrojada por el sistema.
    1. Realizar un query donde retorne todos los nombres originales de los productos en una columna y en otra los nombres de los productos con la primera y la última letra en Mayúscula (como se muestra en la imagen).



	Nombre	Primera y ultima letra en mayuscula
1	Coca-cola	Coca-colA
2	Mekato Papitas limon	Mekato Papitas limoN
3	Galletas festivales	Galletas festivaleS
4	Chocolatina	ChocolatinA
5	Enlatados Salchichas	Enlatados SalchichaS

    2. Devolver la cantidad de participantes con Rol Experto y Género Femenino por cada una de las delegaciones. (10 Puntos)
    3. Devolver la placa, marca y modelo de los vehículos que estuvieron en la ruta 'Centro de servicios y gestión financiera - Hotel El Gran Marquez'. (10 Puntos)
    4. Realizar un query donde se visualice cuál fue el producto más consumido por los participantes de la delegación de C.S.E.T y que pertenecen a la habilidad de Bases de Datos. Se debe mostrar la Delegación y Habilidad en los resultados. . (10 Puntos)
    5. Realizar un query donde se pueda visualizar los números de las habitaciones en las cuales se encuentran los participantes con rol de expertos, que pertenecen a la delegación de C.S.E.T., donde el género sea femenino y donde el año de la fecha de nacimiento sea igual a 1988; Mostrar también el Id de las participantes y su nombre. (10 puntos)
    6. Mostrar todos los datos del participante nacido el 13/05/1996 y a su vez calcular la edad que tiene actualmente, la consulta debe mostrar los siguientes datos identificación, nombre, apellido, genero, fecha nacimiento, edad, teléfono y email
    7. Visualizar todos los participantes que tengan como rol Experto, que pertenezcan a la delegación del C.S.E.T y hayan estado en la ruta número 2

siendo transportados en el vehículo de placa 123-BND y que las fechas en la que fueron transportados estén entre 2023-09-11 y 2023-09-24. (10 puntos)

8. Listar los Nombres de las delegaciones (centros) que se encuentran en el hotel “El gran Márquez” (10 puntos)
9. Listar el Nombre del hotel y número de la habitación donde se encuentra hospedado el participante Pedro Rojas. (10 puntos)
10. A partir del número de identificación de un participante (1098754322), conocer si se encuentra en alguno de los vehículos, en ese caso mostrar los datos completos del vehículo y la ruta del mismo (10 puntos)

PROCESO DE GESTIÓN DE FORMACIÓN PROFESIONAL INTEGRAL

FORMATO GUÍA DE APRENDIZAJE

IDENTIFICACIÓN DE LA GUIA DE APRENDIZAJE

- Denominación del Programa de Formación: Desarrollo de aplicaciones informáticas aplicando Inteligencia Artificial
- Código del Programa de Formación: 233104 V.2.0.0
- Nombre del Proyecto: Construcción de software de integración aplicando inteligencia artificial y tecnologías orientadas a servicios
- Fase del Proyecto: Ejecución
- Actividad de Proyecto: Construir un servicio web para consumir el modelo de IA.
- Competencia: 220501096 - Diseñar la solución de software de acuerdo con procedimientos y requisitos técnicos
- Resultados de Aprendizaje Alcanzar: Crear servicios web
- Duración de la Guía: 144

2. PRESENTACIÓN

A través de las actividades de esta guía de aprendizaje, **interpretará** algunos ejemplos de código fuente desarrollado empleando las mejores prácticas adoptadas por la industria de Software contrastados con ejemplos de código incorrecto. Esto lo llevará a discutir por qué se necesita estándares, principios y metodologías de codificación. Luego **empleará** las convenciones para nombrar, comentar y dar formato al código fuente, incluidas las clases, los métodos y las variables.

Un programa grande puede ser bastante difícil de entender y mantener. Para los programadores junior, no conocer el código y lo que hace puede ser un aspecto desalentador. Los equipos pueden tener dificultades para trabajar juntos en tales proyectos. Y desde el punto de vista de las pruebas, puede hacer las cosas bastante difíciles. Con el desarrollo de las actividades de la guía de aprendizaje **comprenderá** algunas pautas de diseño de programación, principalmente: KISS, YAGNI, DRY, SOLID y Occam's Razor, entre otras. El proposito primordial de los estándares y principios de codificación es que los programadores mejoren en su oficio programando código que sea más eficiente y fácil de mantener.

3. FORMULACIÓN DE LAS ACTIVIDADES DE APRENDIZAJE

3.1 Actividades de Reflexión Inicial

3.1.1 Actividad de Reflexión Inicial No.1: Describir los efectos que tienen las malas prácticas de programación en el proceso de desarrollo de software.

A partir de la observación del video "Siete (7) desastres del desarrollo de software | La importancia de la calidad", describa los efectos que tuvieron las malas prácticas de codificación en los proyectos expuestos y los efectos sobre sus destinatarios.

URL: https://youtu.be/6FsmLwUlTV4

- **Estrategia Didáctica**:

  Aprendizaje interactivo.

- **Técnicas Didácticas:**

  Observación, expresión oral, debate.

  Descripción:

El instructor propiciara un debate entre los aprendices a cerca de las consecuencias que tienen las malas prácticas de codificación, la ausencia de control y aseguramiento de la calidad en el proceso de desarrollo de software.

- **Ambiente requerido para la actividad:**

  Ambiente de formación, Ayudas audiovisuales (Televisor o Video Beam, conectividad a Internet alámbrica o inalámbrica), mesas, sillas.

- **Materiales para la actividad:**

  Computador de escritorio o portátil, sistema operativo Windows (recomendable), software ofimático y material de apoyo.

- **Tiempo asignado para el desarrollo de la actividad**

  Tres (3) Horas

3.2 Actividades de Contextualización

3.2.1 Actividad No. 1 de contextualización: - Identificar los errores más comunes que comenten los desarrolladores de software al programar.

A partir de la observación del video "15 Errores que cometes al programar", identificar los errores más comunes que comenten los desarrolladores y el impacto que tienen en los proyectos de software en términos de **tiempo**, **costo** y **alcance.**

URL: https://youtu.be/GoAxsdg0Xbs

- **Estrategia Didáctica**:

  Aprendizaje interactivo.

- **Técnicas Didácticas:**

  Observación, expresión oral, debate.

  Descripción:

Tras la observación del video, el instructor propiciara un debate entre los aprendices presentando proposiciones que ellos deben apoyar o rebatir con la intención de concientizarlos acerca de la importancia de emplear buenas prácticas de programación y las ventajas que ellas ofrecen.

- **Ambiente requerido para la actividad:**

  Ambiente de formación, Ayudas audiovisuales (Televisor o Video Beam, conectividad a Internet alámbrica o inalámbrica), mesas, sillas.

- **Materiales para la actividad:**

  Computador de escritorio o portátil, sistema operativo Windows (recomendable), software ofimático y material de apoyo.

- **Tiempo asignado para el desarrollo de la actividad**

  Tres (3) Horas

- **Entregables y Medio de Entrega.**

  Documento en Microsoft Word con normas APA con por lo menos una página de las conclusiones del debate. LMS Territorium.

3.3 Actividades de Apropiación

3.3.1 Actividad No. 1 de Apropiación: Conocer los paradigmas de programación sus tipos y enfoques.

Realizar lecturas comprensivas que le permitan conocer los tipos y enfoques de programación disponibles, para lo cual debe remitirse al material de apoyo denominado “MA-PP-Programming Paradigms.md”.

- **Estrategia Didáctica**:

  Auto aprendizaje

- **Técnicas Didácticas:**

  Estudio individual, Elaboración conceptual.

  Descripción:

El Instructor suministrará el material de apoyo MA-PP-Programming Paradigms.md”. Dará las pautas para la elaboración conceptual y la ejemplificación de los conceptos asociados a los enfoques de programación.

- **Material de apoyo:**

  MA-PP-Programming Paradigms.md

- **Ambiente requerido para la actividad:**

  Ambiente de formación, Ayudas audiovisuales (Televisor o Video Beam, conectividad a Internet alámbrica o inalámbrica), mesas, sillas.

- **Materiales para la actividad:**

  Computador de escritorio o portátil, sistema operativo Windows (recomendable), software ofimático y material de apoyo.

- **Tiempo asignado para el desarrollo de la actividad**

  Seis (6) Horas

- **Entregables y Medio de Entrega.**

a) Un (1) mapa mental que represente los siguientes conceptos y sus relaciones a partir de las lecturas realizadas (enfoques de programación). Formato JPG. PNG o PDF. LMS Territorium.

Tipo Imperativo

Enfoques:

- Procedural
- Orientado a objetos.
- Procesamiento en paralelo.

Tipo Declarativo

Enfoques:

- Lógico
- Funcional
- Flujo de Datos (Dataflow)
- Database.

b) Documento en Microsoft Word con normas APA con un ejemplo práctico de cada uno de los enfoques estudiados. Diferente a los contenidos en el material de apoyo. LMS Territorium.

3.3.2 Actividad No. 2 de Apropiación: Interpretar términos base del enfoque de programación Orientada a Objetos (POO)

Empleando herramientas CASE (Visual Parading, Enterprise Architec, Modelio) elaborar diagramas de Clases y Objetos (Notación UML) que faciliten la interpretación de los siguientes conceptos de POO:

a) Clases concretas y sus partes (Nombre, campos, propiedades, métodos).
b) Objetos.
c) Prototipos de objetos en JS.

d) Constructores y Destructores.
e) Encapsulamiento (get y set).
f) Clases estáticas.
g) Visibilidad y Modificacores de acceso.
h) Eventos.

- **Estrategia Didáctica**:

  Exposición.

- **Técnicas Didácticas:**

  Exposiciones del docente.

  Descripción:

El Instructor expondrá a los aprendices los conceptos de clases concretas, clases estáticas, visibilidad y modificadores de acceso.
Empleando herramientas CASE, realizara ingeniería inversa para generar el código fuente correspondiente a cada uno de los artefactos UML elaborados.
Recalcara la importancia del modelado para la correcta construcción del código fuente.

- **Material de apoyo:**

  "MA-POO-PROGRAMACIÓN ORIENTADA A OBJETOS"

- **Ambiente requerido para la actividad:**

  Ambiente de formación, Ayudas audiovisuales (Televisor o Video Beam, conectividad a Internet alámbrica o inalámbrica), mesas, sillas.

- **Materiales para la actividad:**

  Computador de escritorio o portátil, sistema operativo Windows (recomendable), software ofimático y material de apoyo.

- **Tiempo asignado para el desarrollo de la actividad**

  Diez (10) Horas

- **Entregables y Medio de Entrega.**

1. Diagramas de clases en notación UML realizado en herramientas CASE (Visual Parading, Enterprise Architech o equivalente) ejemplificando cada uno de los siguientes conceptos:

   Clase (Constructor, destructor, atributos (Campos y/o Propiedades), Métodos)

   Clase(s) estática(s),
2. Una tabla comparativa entre los distintos tipos de modificacores de acceso.
3. Código fuente generado a través del proceso de Ingeniería inversa correspo diagramas elaborados en el entregable uno(1) de esta Actividad.

Plataforma LMS Territorium. Actividad “Diagramas de clases en notación UML”

3.3.3 Actividad No. 3 de Apropiación: Comprender los tipos de asociación y herencia entre clases en el enfoque de programación Orientada a Objetos (POO)

Empleando herramientas CASE (Visual Parading, Enterprise Architec, Modelio) elaborar diagramas de Clases y Objetos (Notación UML) con distintos tipos de asociación que permitan la comprensión de los siguientes conceptos de POO:

Asosicación Unidireccional.
Asociación Bidireccional.
Herencia.
Clases Abstractas.
Composición.
Agregación.
Interfaces.

- **Estrategia Didáctica**:

  Exposición.

- **Técnicas Didácticas:**

  Exposiciones del docente.

  Descripción:

  El Instructor expondrá a los aprendices los conceptos de asociación entre clases, herencia, clases abstractas, composición, agrgación e interfaces.
  Empleando herramientas CASE, realizara ingeniería inversa para generar el código fuente correspondiente a cada uno de los artefactos UML elaborados.
  Resaltará las ventajas de usar abstracciones y no concreciones para procurar un bajo acoplamiento y una alta cohesión entre clases.

- **Material de apoyo:**

  MA-POO-PROGRAMACIÓN ORIENTADA A OBJETOS”

- **Ambiente requerido para la actividad:**

  Ambiente de formación, Ayudas audiovisuales (Televisor o Video Beam, conectividad a Internet alámbrica o inalámbrica), mesas, sillas.

- **Materiales para la actividad:**

  Computador de escritorio o portátil, sistema operativo Windows (recomendable), software ofimático y material de apoyo.

- **Tiempo asignado para el desarrollo de la actividad**

20 Horas

- **Entregables y Medio de Entrega.**
  a) Un diagrama de clases que represente cada uno de los siguientes tipos de asociación: Asociación Unidireccional. Asociación Bidireccional. Herencia. Composición. Agregación. Clases abstractas e interfaces.
  b) Código fuente generado a través del proceso de Ingeniería inversa correspondiente a los diagramas elaborados en el entregable uno(1) de esta Actividad.

Nota: El Diagrama debe ser construido con herramientas CASE (Visual Parading, Enterprise Architech, etcétera).

Plataforma LMS Territorium. Actividad Diagrama de clases y tipos de asociación.

3.3.4 Actividad No. 4 de Apropiación: Resolver problemas simples de programación Orientada a Objetos

En el entorno de programación Node JS desarrolle el código fuente que dé solución a los ejercicios propuestos en el material de apoyo denominado: MA_Práctica_ POO1_Guía1_AA3.3.4.1 a 3.3.4.5

- **Estrategia Didáctica**:

  Método basado en problemas.

- **Técnicas Didácticas:**

  Situaciones problema.

  Descripción:

El Instructor ayudara a los aprendices a reflexionar e identificar necesidades de información y los motivará a resolver cada uno de los problemas planteados.

Sin embargo, no es un observador pasivo, debe estar orientando el proceso de aprendizaje asegurándose de que el grupo no pierda el objetivo trazado, y de que identifiquen los temas pertinentes de POO para que cumplan con la solución de los problemas.

- **Material de apoyo:**

  MA_Práctica_ POO1_Guía1_AA3.3.4.1 a 3.3.4.5

- **Ambiente requerido para la actividad:**
  Ambiente de formación, Ayudas audiovisuales (Televisor o Video Beam, conectividad a Internet alámbrica o inalámbrica), mesas, sillas.

- **Materiales para la actividad:**

Computador de escritorio o portátil, sistema operativo Windows (recomendable), software ofimático y material de apoyo. IDE de programación (Node, Express, Mongo DB, React)

- **Tiempo asignado para el desarrollo de la actividad**

  30 Horas

- **Entregables y Medio de Entrega.**

  Código fuente desarrollado en el entorno Node JS para cada ejercicio propuesto en la práctica (Práctica_ POO1_Guía1_AA3.3.4.1 a 3.3.4.5)

Plataforma LMS Territorium. Práctica_ POO1_Guía1_AA3.3.4.1

3.3.5 Actividad 5 de Apropiación (Interpretativa)

Nombre de la actividad: Construir una API REST empleando Node JS, Mongo DB y Express.

Descripción:

En el entorno de programación Node JS construya una API REST para los recursos ***CURSOS*** y ***USUARIOS*** referidos en el laboratorio (AA-3.3.5-Laboratorio-Construcción de una API REST).

Construya una base de datos Not Only SQL orientada a documentos en Mongo DB.

Para el control de versiones utilice un repositorio de código fuente (Github). Implemente la estrategia GitFlow.

Construya la documentación de la API REST empleando Swagger o Postman.

Despliegue la API en un ambiente Cloud (Heroku, Azure, AWS, IBM Cloud, Oracle Cloud, Huawei Cloud).

- **Estrategia Didáctica**:

  Método basado en situaciones o casos.

- **Técnicas Didácticas:**

  Estudio de Caso. Prácticas Guiadas.

  Descripción:

El Instructor ayudara a los aprendices a reflexionar e identificar necesidades de información y los motivará a resolver cada uno de los requerimientos planteados en el caso de estudio.

Sin embargo, no es un observador pasivo, debe estar orientando el proceso de aprendizaje asegurándose de que el grupo no pierda el objetivo trazado, y de que identifiquen los temas pertinentes de POO para que cumplan con la solución de los problemas.

- **Material de apoyo:**

- **Ambiente requerido para la actividad:**

  Ambiente de formación, Ayudas audiovisuales (Televisor o Video Beam, conectividad a Internet alámbrica o inalámbrica), mesas, sillas.
- **Materiales para la actividad:**

  Computador de escritorio o portátil, sistema operativo Windows (recomendable), software ofimático y material de apoyo. IDE de programación (Node, Express, Mongo DB, React)
- **Tiempo asignado para el desarrollo de la actividad**

  60 Horas
- **Entregables y Medio de Entrega.**
  1) URL del Repositorio de código fuente de la API REST que da solución al caso de estudio Alquiler de Motocicletas..
  2) Un video a través del cual se evidencie el funcionamiento de los EndPoint de cada uno de los recursos expuesto en la API. Debe usar POSTMAN para demostrar el funcionamiento de los End Point en el ámbito local y remoto (Producción).
  3) Documentación generada por Postman de los End Point probados.

Territorium. Actividad “Link de acceso a repositorio público API REST en Node JS Motocicletas”.

3.4 Actividades de Transferencia del conocimiento:

Nombre de la actividad: Desarrollar el Backend que dé solución al caso de estudio Motocicletas. Véase material de apoyo “MA-CE-CASO DE ESTUDIO MOTORCYCLE COMPANY”.

Descripción:

Desarrollar el Backend que de solución al caso de estudio empleando el Stack de Programación MEAN, el paradigma de APIS “Request Response” y el tipo de API “REST” para cada uno de los recursos del problema abordado.

- **Estrategia Didáctica**:

  Método Masado en Proyectos.

- **Técnicas Didácticas:**

  Estudio de Caso. Prácticas Guiadas.

  Descripción:

El Instructor ayudara a los aprendices a reflexionar e identificar necesidades de información y los motivará a resolver cada uno de los requerimientos del proyecto formativo..

Sin embargo, no es un observador pasivo, debe estar orientando el proceso de aprendizaje asegurándose de que el grupo no pierda el objetivo trazado, y de que identifiquen los temas pertinentes de POO para que cumplan con la solución de los problemas.

- **Material de apoyo:**

  MA_Instructivo1_Guía1_AC.3.3.5. “*Construcción de una API REST con NodeJS, Express y MongoDB” (Cortes Arévalo, 2022, p. 1 - 59).*

- **Ambiente requerido para la actividad:**
  Ambiente de formación, Ayudas audiovisuales (Televisor o Video Beam, conectividad a Internet alámbrica o inalámbrica), mesas, sillas.
- **Materiales para la actividad:**
  Computador de escritorio o portátil, sistema operativo Windows (recomendable), software ofimático y material de apoyo. IDE de programación (Node, Express, Mongo DB, React)
- **Tiempo asignado para el desarrollo de la actividad**
  28 Horas
- **Entregables y Medio de Entrega.**
  1) Código fuente de la API REST asociada al proyecto formativo

El Backend debe estar codificado en NODE JS, debe compilar 100% y se deben evidenciar buenas prácticas de programación.

El código Fuente debe estar en un Repositorio Público tipo Github. Se debe evidenciar buenas practicas de integración de código, a saber: Git Semántico. GitFlow.

2) Debe realizar un video a través del cual se evidencie el funcionamiento de los EndPoint de cada uno de los recursos expuesto en la API. Debe usar POSTMAN para demostrar el funcionamiento de los End Point en el ámbito local y remoto (Producción).
3) Documentación generada por Postman de los End Point probados.
4) URL de los End Point Bublicados (Producción)

Territorium. Actividad “Solución al Proyecto Formativo”.

4. ACTIVIDADES DE EVALUACIÓN

Tome como referencia la técnica e instrumentos de evaluación citados en la guía de Desarrollo Curricular

Evidencias de Aprendizaje	Criterios de Evaluación	Técnicas e Instrumentos de Evaluación
Evidencias de Conocimiento : Codificar la API REST de acuerdo a los requerimientos de software del caso de estudio “Alquiler de motocicletas” **Evidencias de Desempeño** Despliegue de la API REST en ambiente Cloud (Heroku) **Evidencias de Producto:** Construcción de la documentación de la API REST	Utiliza herramientas y entornos de desarrollo, para la codificación de los módulos del sistema, aplicando la sintaxis propia del lenguaje de programación seleccionado, de acuerdo con las necesidades del sistema de información. Construye un repositorio de código fuente para el versionamiento del proyecto de software. Crea la base de datos en el motor de base de datos seleccionado, siguiendo especificaciones técnicas de del informe, según normas y protocolos de la empresa.	**Lista de chequeo IE-** AA.3.3.5.a. Subactividad 1: Codificar la API REST de acuerdo a los requerimientos de software. **Lista de chequeo IE-** AA.3.3.5.b. Subactividad 2: Desplegar la API REST en ambiente Cloud (Heroku, AWS, AZURE). **Lista de chequeo IE-** AA.3.3.5.c Subactividad 3: Construir la documentación de la API REST.

GFPI-F-135 V01

5. GLOSARIO DE TÉRMINOS

API: traduce interfaz de programación de aplicaciones. Este recurso permite que productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementado.
Base de Datos: Es una colección de datos organizados de forma ordenada y almacenada sistemáticamente para operaciones posteriores.

JSON: JavaScript Object Notation (JSON) Es un formato basado en texto estándar que se utiliza para representar datos estructurados en la sintaxis de objetos de JavaScript. Se utiliza principalmente para transmitir datos en aplicaciones web (como es el caso de enviar algunos datos desde el servidor al cliente, por lo tanto, estos datos pueden ser mostrados en páginas web, o viceversa).

Localhost: en el contexto de las redes de computadoras localhost es una computadora que ejecuta un programa. La computadora que se tiene al frente funciona también como un servidor virtual; este modo es comúnmente utilizado para realizar pruebas en el desarrollo web.

Manejadores o handlers: de eventos: en la programación con JavaScript un evento por sí mismo carece de utilidad; para que los eventos resulten útiles o aplicables se debe asociar funciones o código JavaScript a cada evento. De esta forma, cuando se produce un evento de manera inmediata se ejecuta el código indicado, por lo que la aplicación responde ante cualquier evento que se produzca durante su ejecución. Las funciones o código JavaScript que se realizan para cada evento se denominan "manejador de eventos".

Middleware: es un bloque de código que se ejecuta entre la petición que hace el usuario (request) hasta que esa petición llega al servidor. De esta manera son funciones que tienen acceso al objeto de solicitud (req), el objeto de respuesta (res) y la siguiente función de middleware en el ciclo de solicitud-respuesta de la aplicación.

Servidor Web: Una computadora de grandes especificaciones hardware que aloja un sitio web o plataformas con todos sus recursos en la Internet.

Servlet: es una clase hecha en Java utilizada para extender las capacidades de los servidores que alojan aplicaciones que siguen el modelo petición-respuesta

6. REFERENTES BILBIOGRÁFICOS

BIBLIOGRAFÍA

¿Qué es una API? (31 de Octobre de 2017). Obtenido de RedHat: https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces

and, J. A. (2021). *C# 9.0 Pocket Reference.* O'Reilly Media, Inc.

Antani, V. (2017). *Object-Oriented JavaScript - Third Edition.* Packt Publishing.

Bouna, P. (2022). *C# Object-Oriented Programming for Beginners in C# and .NET.* Packt Publishing.

Gabriel Baptista, F. A. (2022). *Software Architecture with C# 10 and .NET 6 - Third Edition.* Packt Publishing.

Herron, D. (2020). *Node.js Web Development.* Packt Publisind.

Kereki, F. (2018). *Modern JavaScript Web Development Cookbook.* Packt Publising.

Laurence Svekis, M. v. (2021). *JavaScript from Beginner to Professional.* Packt Publising.

Padolsey, J. (2020). *Clean Code in JavaScript.* Packt Publising.

Wexler, J. (2019). *Get Programming with Node.js.* Manning Publications.

CIBERGRAFÍA

¿Qué es una API? (2017, Octobre 31). Retrieved from RedHat: https://www.redhat.com/es/topics/api/what-are-application-programmin (Herron, 2020)g-interfaces

Choque, J. C. (2016). Adición de estándares de codificación node.js. GitLab https://gitlab.agetic.gob.bo/dbarra/bolivia-libre/-/commit/1ff84b6ae70f0e9436f6e6ddc7f3e060bc9556ad

Capacho Portilla, J. R., Nieto Bernal, W.(2017). Diseño de base de datos. Editorial Universidad del Norte. https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=9740

Cortes Arévalo, V. C. V. (2022, abril). Construcción de una API REST con NodeJS, Express y MongoDB.

Coronel, C., Rob, P., Morris, S.(2011). Bases de datos: diseño, implementación y Administración. Cengage Learning. https://www-ebooks7-24-com.bdigital.sena.edu.co/?il=1280

Express. (2022). Cómo empezar. https://expressjs.com/es/starter/installing.html

López, A. (2021). Qué es Postman y para qué sirve. OpenWebinars https://openwebinars.net/blog/que-es-postman/

Mozilla. (2022). Await. MDN Web Docs. https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Operators/await

Mozilla. (2022). Introducción a Express/Node. MDN Web Docs. https://developer.mozilla.org/es/docs/Learn/Server-side/Express_Nodejs/Introduction

Nodejs.org. (2022). Acerca de Node.js. Node.js https://nodejs.org/es/about

npm. (2022). About npm. https://www.npmjs.com/about

7. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia	Fecha
Autor (es)				

8. CONTROL DE CAMBIOS (diligenciar únicamente si realiza ajustes a la guía)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

Classes

In this chapter, we are going to discuss JavaScript classes. We have seen JavaScript objects already, and classes are a blueprint or template for object creation. So, many of the things discussed here should not sound too unfamiliar or revolutionary.

Classes enable object-oriented programming, which was one of the most important design advancements in software development. This development reduced the complexity of applications and increased maintainability by a huge margin.

So, object-oriented programming and classes are of great importance for computer science in general. This is not necessarily the case when we apply it to JavaScript though. JavaScript classes are something special compared to other programming languages. Beneath the surface, classes are wrapped in some sort of special function. This means that they are actually an alternative syntax for defining objects using a constructor function. In this chapter, we will learn what classes are and how we can create and use them. Along the way, we will cover the following topics:

- Object-oriented programming
- Classes and objects
- Classes
- Inheritance
- Prototypes

> Note: exercise, project, and self-check quiz answers can be found in the *Appendix*.

Object-oriented programming

Before we start diving right into the fun of classes, let's briefly say something about **object-oriented programming** (**OOP**). OOP is a very important programming paradigm wherein code is structured in objects, leading to more maintainable and reusable code. Working with OOP teaches you to really try to think of all sorts of topics in objects, by bundling properties in such a way that they can be wrapped in a blueprint called a class. This in turn might be inheriting properties from a parent class.

For example, if we are thinking of an animal, we can come up with certain properties: name, weight, height, maximum speed, colors, and a lot more. And then if we think of a specific species of fish, we can reuse all the properties of "animal" and add a few fish-specific properties in there as well. The same for dogs; if we then think of a dog, we can reuse all the properties of "animal" and add a few dog-specific ones to it. This way we have reusable code of our animal class. And when we realize we forgot a very important property for the many animals in our application, we only need to add it to the animal class.

This is very important for Java, .NET, and other classic object-oriented ways of writing code. JavaScript doesn't necessarily revolve around objects. We will need them and we will use them, but they are not the star of our code, so to speak.

Classes and objects

As a quick refresher, objects are a collection of properties and methods. We saw them in *Chapter 3*, *JavaScript Multiple Values*. The properties of an object should have sensible names. So for example, if we have a `person` object, this object could have properties called `age` and `lastName` that contain values. Here is an example of an object:

```
let dog = { dogName: "JavaScript",
            weight: 2.4,
            color: "brown",
            breed: "chihuahua"
          };
```

Copy

Classes in JavaScript encapsulate data and functions that are part of that class. If you create a class, you can later create objects using that class using the following syntax:

```
class ClassName {
  constructor(prop1, prop2) {
    this.prop1 = prop1;
    this.prop2 = prop2;
  }
```

```
}

let obj = new ClassName("arg1", "arg2");
```

Copy

This code defines a class with `ClassName` as a name, declares an `obj` variable, and initializes this with a new instance of the object. Two arguments are provided. These arguments will be used by the constructor to initialize the properties. As you can see, the parameters of the constructor and the properties of the class (`prop1` and `prop2`) have the same name. The properties of the class can be recognized by the `this` keyword in front of them. The `this` keyword refers to the object it belongs to, so it is the first property of the instance of `ClassName`.

Remember we said that classes are just some special function beneath the surface. We could create the object with a special function like this:

```
function Dog(dogName, weight, color, breed) {
  this.dogName = dogName;
  this.weight = weight;
  this.color = color;
  this.breed = breed;
}
let dog = new Dog("Jacky", 30, "brown", "labrador");
```

Copy

The dog example could have been made using a class syntax as well. It would have looked like this:

```
class Dog {
  constructor(dogName, weight, color, breed) {
    this.dogName = dogName;
    this.weight = weight;
    this.color = color;
    this.breed = breed;
  }
```

```
}

let dog = new Dog("JavaScript", 2.4, "brown", "chihuahua");
```

Copy

This results in an object with the same properties. If we do some logging as follows, we will be able to see it:

```
console.log(dog.dogName, "is a", dog.breed, "and weighs", dog.weight,
"kg.");
```

Copy

This will output:

```
JavaScript is a chihuahua and weighs 2.4 kg.
```

Copy

In the next section, we will dive into all the parts of classes.

Classes

You may wonder, if classes do the exact same thing as simply defining an object, why do we even need classes? The answer is that classes are essentially blueprints for object creation. This means that we need to do much less typing if we need to create 20 dogs when we have a `dog` class. If we have to create the objects, we will have to specify all the properties' names each time. And it would be easy to make a typo and misspell a property name. Classes come in handy in these sorts of situations.

As shown in the previous section, we use the `class` keyword to tell JavaScript we want to create a class. Next, we give the class a name. It is the convention to start class names with a capital letter.

Let's have a look at all the different elements of a class.

Constructors

The `constructor` method is a special method that we use to initialize objects with our class blueprint. There can only be one constructor in a class. This constructor contains properties that will be set when initiating the class.

Here you can see an example of a constructor in a `Person` class:

```
class Person {
  constructor(firstname, lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }
}
```

Copy

Beneath the surface, JavaScript creates a special function based on this constructor. This function gets the class name, and it will create an object with the given properties. With this special function, you can create instances (objects) of the class.

Here is how you can create a new object from the `Person` class:

```
let p = new Person("Maaike", "van Putten");
```

Copy

The `new` word is what tells JavaScript to look for the special constructor function in the `Person` class and create a new object. The constructor gets called and returns an instance of the person object with the specified properties. This object gets stored in the `p` variable.

If we use our new `p` variable in a logging statement, you can see that the properties are really set:

```
console.log("Hi", p.firstname);
```

Copy

This outputs:

```
Hi Maaike
```

Copy

What do you think will happen when we create a class without all of the properties? Let's find out:

```
let p = new Person("Maaike");
```

Copy

Many languages would crash, but not JavaScript. It just sets the remaining properties to `undefined`. You can see what happens by logging it:

```
console.log("Hi", p.firstname, p.lastname);
```

Copy

This results in:

```
Hi Maaike undefined
```

Copy

You can specify default values in `constructor`. You would do it like this:

```
constructor(firstname, lastname = "Doe") {
    this.firstname = firstname;
    this.lastname = lastname;
  }
```

Copy

This way, it would not have printed `Hi Maaike undefined`, but `Hi Maaike Doe`.

Practice exercise 7.1

Take the following steps to create a person class, and print instances of friends' names:

1. Create a class for `Person` including the constructor for `firstname` and `lastname`.
2. Create a variable and assign a value of the new `Person` object using your friend's first and last names.
3. Now add a second variable with another friend's name using their first name and last name.
4. Output both friends into the console with a greeting of `hello`.

Methods

In a class, we can specify functions. This means that our object can start doing things using the object's own properties—for example, printing a name. Functions on a class are called methods. When defining these methods, we don't use the `function`keyword. We start directly with the name:

```
class Person {
  constructor(firstname, lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }
  greet() {
    console.log("Hi there! I'm", this.firstname);
  }
}
```

Copy

We can call the `greet` method on a `Person` object like this:

```
let p = new Person("Maaike", "van Putten");
p.greet();
```

Copy

It will output:

```
Hi there! I'm Maaike
```

Copy

You can specify as many methods on a class as you want. In this example, we are using the `firstname` property. We do so by saying `this.property`. If we had a person with a different value for the `firstname` property, for example, `Rob`, it would have printed:

```
Hi there! I'm Rob
```

Copy

Just like functions, methods can also take parameters and return results:

```
class Person {
  constructor(firstname, lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }
  greet() {
    console.log("Hi there!");
  }
  compliment(name, object) {
    return "That's a wonderful " + object + ", " + name;
  }
}
```

Copy

The `compliment` function does not output anything itself, so we are logging it

```
let compliment = p.compliment("Harry", "hat");
console.log(compliment);
```

Copy

The output will be:

```
That's a wonderful hat, Harry
```

Copy

In this case we are sending parameters into our method, because you don't usually compliment your own properties (that's a nice sentence, Maaike!). However, whenever the method doesn't require external input but only the properties of the object, no parameters will work and the method can use its object's properties. Let's do an exercise and then move on to using the properties of classes outside the class.

Practice exercise 7.2

Get your friend's full name:

1. Using the `Person` class from *Practice exercise 7.1*, add a method called `fullname`, which returns the concatenated value of `firstname` and `lastname` when invoked.
2. Create values for `person1` and `person2` using two friends' first and last names.
3. Using the `fullname` method within the class, return the full name of one or both people.

Properties

Properties, sometimes also called fields, hold the data of the class. We have seen one kind of property already, when we created them in our constructors:

```
class Person {
  constructor(firstname, lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }
}
```

Copy

Here, the `Person` class gets two properties from the constructor: `firstname` and `lastname`. Properties can be added or removed just like we did for objects. These properties can be accessed from outside the class, as we saw when we logged them outside the class by accessing them on the instance:

```
let p = new Person("Maaike", "van Putten");
console.log("Hi", p.firstname);
```

Copy

Often, it is not desirable to provide direct access to our properties. We want our class to be in control of the values of properties for several reasons—perhaps we want to

do validation on a property to assure it has a certain value. For example, imagine wanting to validate an age as not being lower than 18. We can achieve this by making direct access to the property from outside the class impossible.

This is how to add properties that aren't accessible from outside. We prefix them with a `#` symbol:

```
class Person {
  #firstname;
  #lastname;
  constructor(firstname, lastname) {
    this.#firstname = firstname;
    this.#lastname = lastname;
  }
}
```

Copy

Right now, the `firstname` and `lastname` properties cannot be accessed from outside the class. This is done by adding `#` in front of the property. If we try it:

```
let p = new Person("Maria", "Saga");
console.log(p.firstname);
```

Copy

We'll get:

```
undefined
```

Copy

If we wanted to make sure we could only create objects with names starting with an "M," we could modify our constructor a bit:

```
constructor(firstname, lastname) {
    if(firstname.startsWith("M")){
      this.#firstname = firstname;
    } else {
```

```
      this.#firstname = "M" + firstname;
    }
    this.#lastname = lastname;
  }
```

Copy

Now when you try to create a person that has a `firstname` value that doesn't start with an "M," it will add an `M` in front. So for example, the value of the following first name is `Mkay`:

```
let p = new Person("kay", "Moon");
```

Copy

This is a very silly example of validation. At this point, we cannot access it from outside the class at all after the constructor. We can only access it from inside the class. This is where getters and setters come into play.

Getters and setters

Getters and setters are special properties that we can use to get data from a class and to set data fields on the class. Getters and setters are computed properties. So, they are more like properties than they are like functions. We call them accessors. They do look a bit like functions, because they have `()` behind them, but they are not!

These accessors start with the `get` and `set` keywords. It is considered good practice to make fields private as much as possible and provide access to them using getters and setters. This way, the properties cannot be set from outside without the object itself being in control. This principle is called **encapsulation**. The class encapsulates the data, and the object is in control of its own fields.

Here is how to do it:

```
class Person {
  #firstname;
  #lastname;
  constructor(firstname, lastname) {
    this.#firstname = firstname;
```

```
    this.#lastname = lastname;
  }
  get firstname() {
    return this.#firstname;
  }
  set firstname(firstname) {
    this.#firstname = firstname;
  }
  get lastname() {
    return this.#lastname;
  }
  set lastname(lastname) {
    this.#lastname = lastname;
  }
}
```

Copy

The getter is used to get the property. Therefore, it doesn't take any parameters, but simply returns the property. The setter is the other way around: it takes a parameter, assigns this new value to the property, and returns nothing. The setter can contain more logic, for example, some validation, as we'll see below. The getter can be used outside the object as if it were a property. The properties are no longer directly accessible from outside the class, but can be accessed via the getter to get the value and via the setter to update the value. Here is how to use it outside the class instance:

```
let p = new Person("Maria", "Saga");
console.log(p.firstname);
```

Copy

This will output:

```
Maria
```

Copy

We have created a new `Person` object with a first name of `Maria` and last name of `Saga`. The output is showing the first name, which is only possible because we have a getter accessor. We can also set the value to something else, because there is a setter. Here is how to update the first name, so the name is no longer `Maria`, but `Adnane`.

```
p.firstname = "Adnane";
```

Copy

At this point, nothing special is happening in the setter. We could do a similar validation as in the constructor before, like this:

```
set firstname(firstname) {
    if(firstname.startsWith("M")){
      this.#firstname = firstname;
    } else {
      this.#firstname = "M" + firstname;
    }
  }
```

Copy

This will check whether `firstname` starts with an `M`, and if it does it will update the value to whatever the `firstname` parameter is. If it doesn't, it will concatenate an `M` in front of the parameter.

Please note that we do not access `firstname` as if it was a function. If you put two parentheses `()` after it, you would actually get an error telling you that it is not a function.

Inheritance

Inheritance is one of the key concepts of OOP. It is the concept that classes can have child classes that inherit the properties and methods from the parent class. For example, if you needed all sorts of vehicle objects in your application, you could specify a class named `Vehicle` in which you specify some shared properties and methods of vehicles. You would then go ahead and create the specific child classes based on this `Vehicle` class, for example, `boat`, `car`, `bicycle`, and `motorcycle`.

This could be a very simple version of the `Vehicle` class:

```
class Vehicle {
  constructor(color, currentSpeed, maxSpeed) {
    this.color = color;
    this.currentSpeed = currentSpeed;
    this.maxSpeed = maxSpeed;
  }
  move() {
    console.log("moving at", this.currentSpeed);
  }
  accelerate(amount) {
    this.currentSpeed += amount;
  }
}
```

Copy

Here we have two methods in our `Vehicle` class: `move` and `accelerate`. And this could be a `Motorcyle` class inheriting from this class using the `extends` keyword:

```
class Motorcycle extends Vehicle {
  constructor(color, currentSpeed, maxSpeed, fuel) {
    super(color, currentSpeed, maxSpeed);
    this.fuel = fuel;
  }
  doWheelie() {
    console.log("Driving on one wheel!");
   }
}
```

Copy

With the `extends` keyword we specify that a certain class is the child of another class. In this case, `Motorcycle` is a child class of `Vehicle`. This means that we'll have access

to properties and methods from `Vehicle` in our `Motorcycle` class. We have added a special `doWheelie()` method. This is not something that makes sense to add to the `Vehicle` class, because this is an action that is specific to certain vehicles.

The `super` word in the constructor is calling the constructor from the parent, the `Vehicle` constructor in this case. This makes sure that the fields from the parent are set as well and that the methods are available without having to do anything else: they are automatically inherited. Calling `super()` is not optional, you must do it when you are in a class that is inheriting from another class, else you will get a `ReferenceError`.

Because we have access to the fields of `Vehicle` in `Motorcycle`, this will work:

```
let motor = new Motorcycle("Black", 0, 250, "gasoline");
console.log(motor.color);
motor.accelerate(50);
motor.move();
```

Copy

And this is what it will output:

```
Black
moving at 50
```

Copy

We cannot access any `Motorcycle` specific properties or methods in our `Vehicle` class. This is because not all vehicles are motorcycles, so we cannot be sure that we would have the properties or methods from a child.

Right now, we don't use any getters and setters here, but we clearly could. If there are getters and setters in the parent class, they are inherited by the child class as well. This way we could influence which properties could be fetched and changed (and how) outside our class. This is generally a good practice.

Prototypes

A prototype is the mechanism in JavaScript that makes it possible to have objects. When nothing is specified when creating a class, the objects inherit from

the `Object.prototype` prototype. This is a rather complex built-in JavaScript class that we can use. We don't need to look at how this is implemented in JavaScript, as we can consider it the base object that is always on top of the inheritance tree and therefore always present in our objects.

There is a `prototype` property available on all classes, and it is always named "prototype." We can access it like this:

```
ClassName.prototype
```

Copy

Let's give an example of how to add a function to a class using the `prototype` property. In order to do so, we'll be using this `Person` class:

```
class Person {
  constructor(firstname, lastname) {
    this.firstname = firstname;
    this.lastname = lastname;
  }
  greet() {
    console.log("Hi there!");
  }
}
```

Copy

And here is how to add a function to this class using `prototype`:

```
Person.prototype.introduce = function () {
  console.log("Hi, I'm", this.firstname);
};
```

Copy

`prototype` is a property holding all the properties and methods of an object. So, adding a function to `prototype` is adding a function to the class. You can use `prototype` to add properties or methods to an object, like we did in the above example in our code with the `introduce` function. You can also do this for properties:

```
Person.prototype.favoriteColor = "green";
```

Copy

And then you can call them from instances of `Person`:

```
let p = new Person("Maria", "Saga");
console.log(p.favoriteColor);
p.introduce();
```

Copy

It will output:

```
green
Hi, I'm Maria
```

Copy

And it will be as if you had defined the class with a favorite color holding a default value, and a function, `introduce`. They have been added to the class and are available for all instances and future instances.

So the methods and properties defined via `prototype` are really as if they were defined in the class. This means that overwriting them for a certain instance doesn't overwrite them for all instances. For example, if we were to have a second `Person` object, this person could overwrite the `favoriteColor` value and this wouldn't change the value for our object with `firstname` as `Maria`.

This is something you should not be using when you have control over the class code and you want to change it permanently. In that case, just change the class. However, you can expand existing objects like this and even expand existing objects conditionally. It is also important to know that the JavaScript built-in objects have prototypes and inherit from `Object.prototype`. However, be sure not to modify this prototype since it will affect how our JavaScript works.

Practice exercise 7.3

Create a class that contains properties for different animal species and the sound that each species makes, and create two (or more) animals:

1. Create a method that prints a given animal and its sound.

2.Add a prototype with another action for the animal.

3.Output an entire animal object into the console.

Chapter projects

Employee tracking app

Create a class to track the employees of a company:

1.Use first names, last names, and the number of years worked as values in the constructor.

2.Create two or more people with values for their first names, last names, and the number of years they've worked at the company. Add the people into an array.

3.Set up a prototype to return the details of the person's first and last names and how long they've worked at the company.

4.Iterate the contents of the array to output the results into the console, adding some text to make the output a full sentence.

Menu items price calculator

Create a class which will allow you to work out the combined price of a number of items, and interact with it to work out the total cost of different orders.

1.Create a class that contains the prices of two menu items as private field declarations.

2.Use the constructor in the class to get the argument values (how many of each item are being bought).

3.Create a method to calculate and return the total cost depending on how many of each item the user selects.

4.Use a getter property to grab the value output by the calculation method.

5. Create two or three objects with different combinations of menu selections, and output the total cost in the console.

Self-check quiz

1. What is the keyword used to create a class?
2. How would you set up a class for a person's first and last names that could include `first` and `last` as initial properties?
3. What is the concept of one thing gaining the properties and behaviors of another thing called?
4. Which of the following are true about the `constructor` method?
    - It gets executed automatically when a new object is created.
    - It should only be added afterward.
    - It has to include the `constructor` keyword.
    - It is used to initialize object properties.
    - It can be used when you have multiple values.
5. Troubleshoot the following code so that the prototype outputs the first and last name of the `Person` into the console. Which is the correct syntax for the `Person` prototype?

```
function Person(first,last) {
  this.first = first;
  this.last = last;
}
      // What should go here: A, B, or C?
      const friend1 = new Person("Laurence", "Svekis");
      console.log(friend1.getName());
```

Copy

A)

```
Person.prototype.getName = (first,last) {
```

```
  return this.first + " " + this.last;
};
```
Copy

B)

```
Person.prototype.getName = function getName() {
return this.first + " " + this.last;
};
```
Copy

C)

```
Person.prototype = function getName() {
return this.first + " " + this.last;
};
```
Copy

Summary

In this chapter, we introduced you to the concept of OOP. This means that we structure our code in such a way that objects are the central players of the logic. Classes are blueprints for objects. We can make a template for an object and create an instance easily by using the `new` keyword.

We then saw that classes can inherit from each other by using the `extends` keyword. Classes that extend from another class will have to call the constructor of this class with `super()` and will then automatically have access to all the properties and methods of the parent. This is great for reusable and highly maintainable code.

Lastly, we encountered prototypes. This is the built-in JavaScript concept that makes classes possible. By adding properties and methods to a class using `prototype`, we can modify the blueprint of that class.

In the next chapter, we will consider some of JavaScript's built-in methods, which can be used to manipulate and add complexity to your code!

Join our book’s Discord space

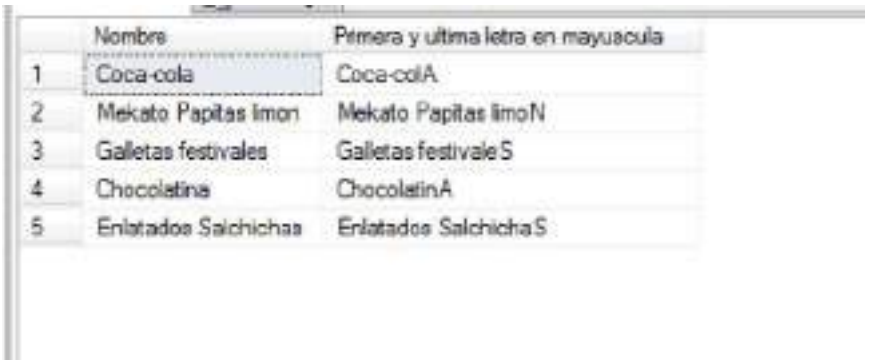
Join the book’s Discord workspace for a monthly *Ask me Anything* session with the authors:

https://packt.link/JSBook



Tomando como referente las entidades y los datos presentados en el documento de Excel denominado “entidadesActividad4” realice las siguientes actividades:

1. Construya una base de datos denominada “SQLActividad4”
2. Cree cada una de las entidades(tablas) propuestas, tenga en cuenta realizar el análisis correspondiente e identificar las llaves primarias y foráneas en cada una de las entidades
3. Inserte los registros propuestos en la hoja “datos”, del documento “entidadesActividad4”
4. Realice las siguientes consultas, y en un documento de Word incluya la instrucción utilizada junto con la respuesta arrojada por el sistema.
    1. Realizar un query donde retorne todos los nombres originales de los productos en una columna y en otra los nombres de los productos con la primera y la última letra en Mayúscula (como se muestra en la imagen).



	Nombre	Primera y ultima letra en mayuscula
1	Coca-cola	Coca-colA
2	Mekato Papitas limon	Mekato Papitas limoN
3	Galletas festivales	Galletas festivaleS
4	Chocolatina	ChocolatinA
5	Enlatados Salchichas	Enlatados SalchichaS

    2. Devolver la cantidad de participantes con Rol Experto y Género Femenino por cada una de las delegaciones. (10 Puntos)
    3. Devolver la placa, marca y modelo de los vehículos que estuvieron en la ruta 'Centro de servicios y gestión financiera - Hotel El Gran Marquez'. (10 Puntos)
    4. Realizar un query donde se visualice cuál fue el producto más consumido por los participantes de la delegación de C.S.E.T y que pertenecen a la habilidad de Bases de Datos. Se debe mostrar la Delegación y Habilidad en los resultados. . (10 Puntos)
    5. Realizar un query donde se pueda visualizar los números de las habitaciones en las cuales se encuentran los participantes con rol de expertos, que pertenecen a la delegación de C.S.E.T., donde el género sea femenino y donde el año de la fecha de nacimiento sea igual a 1988; Mostrar también el Id de las participantes y su nombre. (10 puntos)
    6. Mostrar todos los datos del participante nacido el 13/05/1996 y a su vez calcular la edad que tiene actualmente, la consulta debe mostrar los siguientes datos identificación, nombre, apellido, genero, fecha nacimiento, edad, teléfono y email
    7. Visualizar todos los participantes que tengan como rol Experto, que pertenezcan a la delegación del C.S.E.T y hayan estado en la ruta número 2

siendo transportados en el vehículo de placa 123-BND y que las fechas en la que fueron transportados estén entre 2023-09-11 y 2023-09-24. (10 puntos)

8. Listar los Nombres de las delegaciones (centros) que se encuentran en el hotel “El gran Márquez” (10 puntos)
9. Listar el Nombre del hotel y número de la habitación donde se encuentra hospedado el participante Pedro Rojas. (10 puntos)
10. A partir del número de identificación de un participante (1098754322), conocer si se encuentra en alguno de los vehículos, en ese caso mostrar los datos completos del vehículo y la ruta del mismo (10 puntos)

Manejo de clases en ES6 JavaScript

Francisco Javier Arce Anguiano

ES6-0901-Clases

Dentro de ECMAScript vamos a tener definición de clases. Nos alejamos de lo que son los prototipos y ahora nos acercamos más a la programación orientada a objetos más apegada a Java o C++ y ese tipo de lenguajes.

Si nos vamos a nuestro editor de código vamos a guardar nuestro archivo con el nombre JSOOP-Clases y vamos a crear nuestras etiquetas HTML. Pulsamos la tecla tabulador vamos a poner aquí llave script clases y creamos nuestro script y aquí pues vamos a crear nuestra clase vamos a tener la palabra reservada ***class***. Vamos a definir generalmente pues ya habíamos visto que las clases la primera letra se ponen mayúscula.

Nombremos a nuestra clase como ***Gato*** y abrimos lo que son nuestros llaves y con eso nosotros ya vamos a tener una clase. Ya no necesitamos hacerla como función. Para crear una instancia podemos utilizar la palabra reservada ***var*** o también podemos utilizar la nueva palabra reservada ***let***, la cual define un objeto o variable dentro de dentro de llaves, es decir, un bloque. En este ejemplo sería indistinto y vamos a utilizar lo que es el var y nombramos a nuestra instancia como **Benito** es igual a new Gato.

En este caso como no tenemos ningún parámetro, ninguna función pues podemos utilizarla sin necesidad de los paréntesis característicos. Una clase, opcionalmente, no precisa argumentos en su definición, por lo tanto no le acompañan los paréntesis habituales.

No es necesario tampoco lo que es el punto y coma al final, pero es una buena práctica y como habíamos dicho es una buena práctica poner el nombre de clase en mayúsculas para diferenciarlo de otros tipos de objetos como funciones o como variables.

Esta clase no hay que confundir no es una función el navegador no lo va a permitir utilizar como navegador sino que se reserva con un constructor. El contenido de la clase se ejecuta automáticamente en modo estricto.

Las declaraciones de las clases no siguen la regla de las funciones de lo que es el ***hoisting*** y de forma implícita una clase se comporta como una constante. No es posible declararla más adelante en el código en el mismo programa, que es el ambiente el alcance o también en inglés se conoce como el ***scope***.

Como ya habíamos comentado, realmente no es un nuevo modelo de programación, sino este lo estamos agrupando de manera diferente internamente y funciona exactamente igual qué es lo que hemos visto en los clases anteriores. Más adelante vamos a seguir analizando lo que es esta poderosa forma de programar.

Vamos a guardar nuestro archivo y vamos ejecutarla en un navegador. Obviamente aquí no tenemos mayor salida.

Pero bueno, aquí no es más que por el encoding perdón que está puesto pero si le damos digamos Benito pues vamos a ver que lo estamos manejando exactamente como un objeto que de momento está vacío y si bien ahí tenemos nuestro viejo amigo el prototipo.

Así que en los próximas clases vamos a ver cómo podemos utilizar esta nueva estructura de las clases dentro de ECMA6.

Es muy importante que su navegador debe de ser compatible con ECMA6. Si entre por ejemplo a la página caniuse.com y buscamos Class vamos a ver por ejemplo que dentro de lo que es ECMA 6, dentro de las clases Internet Explorer ya está prácticamente eliminado. Lo que es el Opera Mini también prácticamente ya van de salida y todos los demás navegadores pues funcionan bastante bien.

Entonces hay que tener cuidado porque esta estructura no es compatible con Internet Explorer prácticamente en ninguna de sus versiones, pero con Edge funciona sin mayor problema.

ES6-0902-Instancias

Vamos a ver poco a poco lo que son las partes de nuestra programación orientada a objetos con lo que es el ES6 que temen le llaman ECMA 2015.

Si tomamos el ejemplo anterior vamos a poner aquí nuestro charset. Estamos viendo que estamos creando lo que es nuestra clase y estamos haciendo una instancia de Gato.

Por ejemplo, vamos a hacer otro con var que se llame Cucho y que sea igual a la instancia de un Gato… punto y coma. En este caso como no estamos pasando parámetros, a lo que es no hay una función constructora lo vamos a hacer más adelante.

Pues entonces no es necesario los paréntesis aquí. Sí también le ponemos los paréntesis no pasa absolutamente nada.

Podemos mandar, digamos lo que sería un typeof Benito y un ***typeof()*** de Cucho y ambos nos deben decir de qué son objetos.

También podemos utilizar digamos en Benito podemos utilizar el ***instanceOf*** de gato y un instanceof igual de Perro. Guardamos

Nos vamos a nuestro navegador y obviamente nos va a decir que es una referencia que no existe, la de perro. Vamos a mandarlo a la consola aquí a poner console.log() y aquí console.log() igual Consol punto.

Obviamente nos dice que [la clase] perro no existe. No podemos creer vamos a creer rápidamente lo que sería la instancia perro... perdón la clase de Perro, entonces simplemente le vamos a decir aquí Perro. Nos vamos a nuestro navegador, y refrescamos.

Obviamente nos va a decir que es objeto objeto las luces de los tipos con el typeof() nos van a decir que son objetos. No nos dice que son clase sino objetos y después nos dice true (verdadero) si es instancia de Gato y nos dice false (falso) cuando preguntamos si es instancia de Perro. Así que vamos a utilizar lo que es el new... vamos a utilizar lo que es el new para crear nuestros objetos o sus estancias. En este caso si no lleva parámetros podemos omitir los paréntesis y podemos utilizar lo que es el ***typeof()*** y lo que es instanceof para poder reconocer lo que son nuestras instancias de clases dentro de ECMA 6.

ES6-0903-Constructor

Vamos a ver poco a poco lo que son las partes de nuestra programación orientada a objetos con lo que es el ES6 que también le llaman ECMA 2015.

Así que lo vamos a ver cómo hacer una clase. Si tomamos el ejemplo anterior vamos a poner aquí nuestra etiqueta charset. Estamos viendo que estamos creando lo que es nuestra clase y estamos haciendo una instancia de ***Gato***.

Por ejemplo, vamos a hacer otra instancia con la palabra reservada ***var*** que se llame Cucho y que sea igual a la instancia de un ***Gato***. En este caso, como no estamos pasando parámetros, a lo que es no hay una función constructora, la cual la vamos a hacer más adelante. Pues entonces no es necesario los paréntesis aquí, pero sí también le ponemos los paréntesis no pasa absolutamente nada.

Podemos mandar, digamos lo que sería un ***typeof*** a la instancia Benito y un typeof() a la instancia de Cucho y ambos nos deben decir de qué son ***objetos***.

También podemos utilizar, digamos, en Benito podemos utilizar el ***instanceOf*** de Gato y un ***instanceof*** igual de Perro. Guardamos el archivo y nos vamos a nuestro navegador. Obviamente nos va a decir que es una referencia que no existe, la de Perro. Vamos a mandarlo a la consola aquí a poner ***console.log()*** y aquí ***console.log()***. Obviamente nos dice que [la clase] Perro no existe.
Vamos a crear rápidamente lo que sería la instancia perro... perdón la clase de Perro,. Entonces simplemente le vamos a decir aquí ***Perro***. Nos vamos a nuestro navegador, y lo refrescamos.

Obviamente nos va a decir que es objeto los tipos con el ***typeof()*** nos van a decir que son objetos. No nos dice que son ***clase*** sino objetos y después nos dice ***true*** (verdadero) si es instancia de ***Gato*** y nos dice ***false*** (falso) cuando preguntamos si es instancia de ***Perro***. Así que vamos a utilizar lo que es la palabra reservada ***new***... vamos a utilizar lo que es ***new*** para crear nuestros objetos o sus estancias. En este caso, si no lleva parámetros podemos omitir los paréntesis y podemos utilizar lo que es el ***typeof()*** y lo que es ***instanceof*** para poder reconocer lo que son nuestras instancias de clases dentro de ECMA6.

ES6-0904-GettersSetters

Otro punto de la programación orientada a objetos que nos va a simplificar mucho la nueva especificación ES6 van a ser los setter y los objetos que se utilizan para obtener y asignar valores a los atributos de nuestros objetos respectivamente.

En JavaScript ambos métodos se corresponden con los atajos sintácticos, que son funciones finalmente que no añaden funcionalidad adicional adicional alguna simplemente pues hay que tener cuidado con las referencias circulares así que si nos vamos a nuestro editor de código, vamos a partir de un archivo nuevo y vamos a guardarlo.

En este caso como JavaScript Getters y Setters, punto HTML y creamos lo que son nuestras etiquetas HTML. Vamos a poner aquí como Java Script setter y getter y ponemos nuestro charset con el meta charset ponemos nuestras etiquetas de script y vamos a hacer nuestra nuestra clase de Gato y vamos a poner lo que es el nombre es igual a Minino por omisión si viene vacío, pues va a tomar minino y lo que es el pelo, pues, vamos a poner que es por omisión negro.

Llave que abre y llave que cierra... para evitar referencias circulares. Vamos a utilizar el this, punto, con un guion bajo y vamos a escribir el nombre. Hay personas que le gusta ponerlo al inicio. Hay personas que les gusta ponerlo al final incluso llegan a poner doble guion bajo pues para diferenciar lo que va a hacer después las diferencias con el getter y el setter y vamos a decir que esto va a ser igual a nombre, punto y coma.

Y después el this, punto, guión bajo pelo vendría siendo igual a con la misma lógica que este que estamos siguiendo. Vamos a crear lo que es el setter que es cuando nosotros podemos modificar. El setter finalmente por lo que va a ser es una equivalente a la función set y después le decimos nombre y se observa pues ahí no ponemos el guión bajo ahí lo vamos a tomar y después recibimos el valor o la cadena y aquí entonces lo vamos a designar a this, punto.

Y aquí si ponemos el guion bajo el nombre va a ser igual al valor. Por lo general en esas funciones pues nosotros podemos hacer validaciones, podemos hacer muchas cosas, después vamos a utilizar lo que es el get y vamos a poner lo que es el set igual de pelo, vamos a ponerlo que es el valor llave que abre y que cierra, vamos a poner aquí this punto que un bajo pelo en ese caso si va con el guion bajo y pasabamos lo que es el valor.. punto y coma. Vamos a utilizar lo que es el toString(), que ya lo habíamos hecho entonces toString(), y va a recibir cuando pasamos a los dos llaves y vamos a decir return this, punto, guión bajo nombre... concatenamos con.. "tiene el pelo color"... concatenamos con this, punto y guión bajo, pelo… con punto y coma… así qué bueno fuera de lo que es nuestra clase, vamos a crear las dos instancias, vamos a decir let Benito es igual a new Gato y vamos a ponerlo.

Así, sin valores, después let Cucho es igual a new Gato y vamos a llamarlo como mucho Cucho. Y después de color rosa y su punto y coma, y vamos a poner lo que es el console.log() de Benito punto toString(), con los valores con punto y coma, y vamos a copiar esto mismo para Cucho.

Entonces pesamos aquí la cadena y escribimos Cucho. Bueno, como deberá de suponer, pues este vamos a abrirlo que es la consola y tenemos minino tiene el pelo color negro y Cucho tiene el color rosa como que hubiéramos pensado.

Como lo habíamos pensado ahora vamos a cambiar de Benito, punto pelo, es igual azul, punto y coma y Benito, punto, nombre, es Benito Bodoque y su punto y coma.

Y volvemos a utilizar, pues, la información de Benito. Aquí, repetimos, podemos hacer este validaciones de todo tipo aquí en el setters, guardamos nos vamos a nuestra página, refrescamos y entonces pues Benito Bodoque tiene el pelo azul. De acuerdo, entonces esa es una manera mucho más limpia de trabajar con las variables, simplemente hay que tratar de evitar las referencias circulares. Si ponemos nombres, nombres, nombres nuevos por todos lados pues a veces caemos aquí en ciertos problemas entonces eso lo solucionamos, pues poniendo a lo que es la variable interna de la clase con algún pequeño diferenciador, por ejemplo un guion bajo podemos hacer también una función que se llame get() que en este caso no es exactamente necesario lo que es el get().

Pero podemos decir color pelo ponemos llave que abre y llaves que cierra vamos a copiar esto mismo y en vez de lo que es el get() vamos a utilizar lo que es ahora color pelo y pero no lo vamos a utilizar con un paréntesis vamos a poner color pelo como si fuera una variable.

Aquí lo estamos recuperando por medio del get() guardamos nos vamos a nuestro navegador y refrescarnos y el resultado pues es lo mismo. Benito Bodoque tiene el pelo color azul. Esto quiere decir que pudimos manejarlo como si fuera una propiedad sin necesidad de pasarle parámetros y sin necesidad de pasarle lo que son los paréntesis porque lo estamos haciendo por medio de un get(), este color de pelo.

Como ve pues es una manera muy limpia de trabajar lo que son los gettes y los setters conocidos pues en el ambiente del desarrollo orientado a objetos de la programación orientada a objetos simplemente hay que tener precaución del problema de no caer en referencias circulares entonces si va a utilizar setter y getters, pues lo más sano es que haga una pequeña diferenciación en sus variables internas o viceversa una pequeña variación en lo que son los nombres de sus propiedades settlers y getters siempre nos van a regresar, siempre nos van a regresar como propiedades, aunque bueno sabemos que son métodos.

ES6-0906-MetodoEstatico

Otra característica que vamos a tener en la programación orientada a objetos dentro de lo que es el ES6 va a ser los métodos estáticos, los métodos estáticos. nosotros podemos llamarlos sin necesidad de crear una instancia. En los ejercicios anteriores creamos una instancia con el neón y después utilizamos propiedades y utilizamos métodos.

En este caso nosotros podemos sin necesidad de crear una instancia llamar alguno de los métodos por ejemplo la clase Math pues no necesitamos crear un objeto de tipo ni un math para utilizarlo.

Para poder utilizar el coseno el seno etcetc y Math viene dentro de lo que es el paquete estándar de JavaScript.

Nosotros también podemos hacer entonces métodos estáticos. Así que si nos vamos a nuestro editor de código vamos a crear un archivo nuevo y vamos a guardarlo como JavaScript orientado a objetos métodos estáticos punto HTML.

Guardamos creamos nuestras etiquetas HTML. Vamos a escribir como título JavaScript métodos estáticos y creamos nuestro meta charset, vamos a crear nuestro script nuestras etiquetas de script y vamos a crear nuestra clase de datos Class Gato, llave que abre llave que cierra y vamos a crear un método estático por medio de Static y el nombre del método.

Por ejemplo vamos a poner this y podemos pasar lo que es un valor y vamos a escribir aquí.

Vamos a suponer como dice y vamos a poner un retorno que diga los gatos dicen "miau", de acuerdo, para poder llamar a nuestra clase y entonces aquí vamos a escribir, sin necesidad de crear el objeto, vamos a decir gato, punto, dice y es un método estático, de acuerdo.

Y aquí vamos a mandarlo a la consola. console.log(), punto y coma y lo ponemos antes del punto y coma. Guardamos y nos vamos a nuestro navegador.

Abrimos nuestra consola. En este caso vamos a Web Developer y web Console y entonces ahí están "los gatos dicen miau". Como puede ver pues es una de las muchísimas aplicaciones que podemos nosotros definir lo que es un método de forma estática y lo podemos llamar sin necesidad de crear la instancia dentro de JavaScript ECMA 6.

ES6-0907-Herencia

Vamos a ver ahora lo que es el proceso de la herencia en lo que es la programación orientada a objetos con el ECMA 6 y en este caso vamos a utilizar la sentencia extends, que se utiliza pues en programación orientada a objetos como PHP o en otros lenguajes.

Así que si nos vamos a nuestro editor de código. Vamos a trabajar con un archivo en blanco y vamos a guardarlo como JavaScript orientado a objetos herencia punto HTML y vamos a hacer nuestras etiquetas HTML.

Vamos a poner aquí como título JavaScript herencia y ponemos también lo que es nuestro charset con meta charset escribimos nuestras este etiquetarse script y ebrias aquí vamos a crear una clase class que sea "mascota" y dentro de "mascota" vamos a crear en el constructor constructor lo que hemos visto del constructor que es parte del ES6.

Vamos a poner los parámetros para hacer cualquier nombre pero bueno para el común como un objeto vacío en el caso de que no nos mande ninguna información pues subjetivación y aquí vamos a utilizar el constructor entonces vamos a poner el paréntesis y las llaves es igual a parms, digamos este es el constructor y vamos a poner los datos.

Vamos a poner por ejemplo el nombre del animalito nombre dos puntos va a ser igual a this y vamos a utilizar un bajo nombre o lo que vemos visto perder y íbamos a ponerle sin nombre por omisión, coma, género.

Vamos a poner aquí un punto y un bajo género y ponemos sin género y después especie, dos puntos de un bajo especie es igual sin especies sin especial y no lleva la última como aquí lo que estamos haciendo entonces es pasándolo por el constructor vamos a hacer también lo que son los getter y los setters. Vamos a utilizar por ejemplo get nombre y vamos a poner un return de this, punto, de un bajo nombre que también le habíamos visto lo que son setters y getters,

vamos a copiar esto es getter y vamos a pegarlo aquí y ponemos género y aquí abajo sería un bajo género.

Pegamos, también lo que sería especial especie y aquí bajo especia... especie y especie especie, especie especie, muy bien.

Entonces ahí tenemos nuestros Getter aquí vas a ponerle comentario que son los getters y vamos a copiar tú mismo para ser nuestros Settlers. Ese va a ser digamos que nuestra función pues base y aquí vamos a poner el valor y nombre sería igual valor, valor igual aquí ponemos este setter y también setter, género vamos a copiar esto mismo… entonces aquí vamos a borrar el return y vamos a poner aquí valor y pesamos valor y entonces que hoy tenemos nuestra es nuestra clase base o madre y vamos a hacer entonces al menos una de las [clases], no llevamos mucho tiempo Class en Gato y decimos extends Mascota, llave que abre y cierra y vamos a poner aquí un constructor, constructor que tenga igual vamos a poner los parámetros es igual a vacío, llave que abre que cierra y vamos a utilizar lo que super().

Más adelante lo que hace superes de que la clase madre y recupera los parámetros que tengamos en la función correspondiente en este caso del constructor, entonces decimos súper pasamos como parámetro, género dos puntos y vamos a poner que es un Gato() y su ahora que ya tenemos nuestras nuestras clases y la clase heredada pues vamos a generar un objeto vamos a ponerle Benito y este va a ser igual a new Gato y ya por omisión pues debe de ser un gato porque aquí lo estamos pasando como género, vamos a pasar consola.log() y vamos a imprimir primero que sea Benito, punto, nombre nos lo hemos modificado. después Benito, punto, género y después Benito punto, especie y su punto y coma.

Y ahora vamos a modificarlo vamos a poner Benito en tu nombre. Ahí estamos utilizando los getters y los setters de este de la clase madre y vamos a poner que se llama Benito Bodoque como Benito punto género no sé decir que es un macho, punto y coma. Podemos copiar esta misma línea pero a ver antes y después el género no lo modificamos porque sigue siendo un gato. Guardamos.

Vamos a revisarlo y bueno aquí me faltó...l perdón. Igual aquí le puse género Gato y el y no en el género pues macho y hembra no gato aquí es especial que bueno gato creo que son especies una subespecie pero bueno especie un gato o felino y entonces ahora si generó macho.

Guardamos, nos vamos a nuestro navegador, refrescamos y ahora se nos dice sin nombre, sin género. Gato, Benito Bodoque, macho, gato.

Vamos a crear pues también lo que es una leyenda lo podemos hacer desde desde la clase madre. Vamos a poner aquí enunciado y vamos a modificar el link Trunk llave que abre y cierra y vamos a poner algo similar a lo que tenemos aquí y aquí podemos por ejemplo return. "Mi nombre es", espacio concatenamos con lo que sería this, punto, guión bajo, nombre más,

coma, "soy un ", contactemos con el más, ***this***, punto, guión bajo especie, más "y soy" con él más que un bajo this perdón, this punto que un bajo género, punto y coma, así que vamos a revisar muy bien, y aquí entonces vamos a poner Benito toString() y aquí también Benito toString(), guardamos nos vamos a nuestro navegador refrescamos y "mi nombre es sin nombre Soy un gato y soy sin género y después nos dice Mi nombre es Benito Bodoque Soy un gato y soy macho.

Así que bueno con eso es una de las muchas muchas posibilidades que vamos a tener por medio del extends de heredarse una clase que generalmente se llama clase padre o clase madre con lo que serían diferentes subclases que es una de las facilidades que vamos a tener en ES6 con JavaScript.

ES6-0908-Super

Vamos a ver ahora a más detalle lo que es la sentencia súper cuando utilizamos la herencia. Entonces super va a llamar tanto propiedades o métodos de la clase padre a la clase hijo.

Hemos comentado de que en español la clase debería de ser hija y madre pero tomando como es en inglés pues dicen "Son and Father" pero bueno se puede ser como se quiera llamarlo la clase superior y la clase inferior heredada. En este caso nosotros podemos llamar de desde el método constructor de la clase padre con super() y después los argumentos, que fue lo que hicimos en el ejercicio anterior pero también podemos llamar a cualquier otro tipo de método que se encuentre en la clase padre hacia la clase hijo.

Aquí lo importante es que cuando estamos utilizando super() desde el constructor debe de utilizarse antes de la palabra clave this. Esto es muy importante porque muchas veces nos da error y lo primero que tenemos que hacer es llamar a super() y bueno también como hemos visto puede ser utilizado para llamar a otras funciones, es decir métodos desde el objeto padre hacia la función hacia la clase heredada.

Así que si nos vamos a nuestro editor de código, vamos a crear lo que es un un ejemplo muy sencillo. De hecho viene en lo que es la documentación de Mozilla que creo que es la mejor documentación para JavaScript y vamos a guardarlo como script orientado a objetos super() junto a ello lo guardamos creamos nuestras etiquetas HTML.

Vamos a poner aquí este script super y creamos nuestro charset con tabulador y ahora aquí vamos a escribir el script y abrimos aquí pues vamos a hacer una clase. Como ya hemos dicho la programación orientada a objetos explica con gatos o con geometría, entonces vamos a hacer algo de geometría y vamos a decir clásico de rectángulo, rectángulo y bueno hay hasta una teoría bien fundamentada de que este no un rectángulo no puede quedar a un cuadrado pero es simplemente un ejemplo constructor.

Abrimos paréntesis y vamos a tomar altura y ancho o anchura y abrimos las llaves y vamos a poner con el this.

Aquí sí podemos utilizar this. nombre es igual a rectángulo rectángulo y su punto y coma. Después vamos a poner this, punto, altura es igual altura y después this, punto, ancho es igual a ancho punto y coma. Entonces ahí tenemos nuestra función constructora vamos a mandar el clásico saludo, cerramos llave que abre y cierra y vamos a poner para el console, punto, los, vamos escribir "Hola soy un" y aquí podemos poner coma, this, punto, nombre rectángulo cuadrado, lo que sea. Cerramos. Vamos a tener también por medio del set un área y en este caso pues vamos a regresar return, this, punto, altura por this, punto, ancho o anchura y ahí lo tenemos y también vamos a poner con el set, área el valor value o es escribir y vamos a decir que entonces la altura, this, punto, altura siendo igual a lo que es this, punto, ancho menos lo que es el cuadrado del valor Math junto a sqrt(). Ya tenemos nuestra clase.

Ahora vamos a utilizar lo que es la clase Cuadrado Cuadrado y extends y heredamos de rectángulo. Abrimos y vamos a hacer lo que es el constructor y vamos a tomar solamente que en ese caso es un cuadrado pues de tener los dos lados iguales [obviamente] y vamos a creer lo que es este clásico.

Vamos a llamar a un this simplemente como altura, punto y coma, no necesitamos modificarlo y después vamos a tomar un super. Que en este caso pues le vamos a pasar a altura y el ancho pues que ser lo mismo entonces altura, coma, altura con super, punto y coma, y después le vamos a decir de this, punto, name, cuadrado y punto y coma que es el. Este es el ejemplo clásico que tenemos y crear una instancia vamos a decir este podemos utilizar let o var como quiera usted y puede poner mi cuadrado y cuadrado es igual a new cuadrado y estamos pasando digamos 10. Lo guardamos y nos vamos a nuestro navegador refrescamos y bueno esta mi consola y entonces este es muy muy común que lo hagamos que es el “reference error” o que debe de llamarse el constructo rantes de cualquier this.

Entonces regresamos y vamos a comentar la primera instrucción debe de ser súper buena en ese caso pues ya no tenemos el error que es algo muy muy común que tengamos vamos a decir mi cuadrado punto saludo, que como se ve pues lo tenemos aquí en el en el rectángulo en la clase padre. “Hola soy un rectángulo” o porque no lo modificamos si bien aunque tengamos aquí name cuadrado en realidad no lo estamos modificando porque lo estamos pasando después, entonces como lo estamos llamando aquí pues entonces no este no lo está modificando

Si por ejemplo utilizamos área en vez de saludo vamos a utilizar aquí un console punto log y vamos a llamar a área como lo estamos haciendo como set y get pues no lo hacemos como una función entonces aquí lo aplicamos y nos dice efectivamente que es 100 si se está modificando lo que dices de lo que es la altura y lo que es el ancho, la anchura pero no el no las las propiedades que tengamos aquí, por ejemplo vamos a modificar ahora por medio de mi

cuadrado, punto, altura, área, perdón, área y le voy a poner y es igual a 50 recordemos que lo estamos haciendo con set y get entonces no es su función sino lo manejamos como una propiedad vamos a revisarlo y aquí tengo un error pues aquí value o valor value, guardamos y lo ejecutamos en nuestro navegador a esta calculando el área entonces
es hay que simplemente tener cuidado con el súper, el súper nos sirve para llamar lo de lo que tengamos en la clase padre ya sea propiedades podemos llamar también métodos.

Así que ese sí de las nuevas características que vamos a tener en ES6 para la programación orientada a objetos en JavaScript.

Functions

You have seen quite a lot of JavaScript already, and now you are ready for functions. Soon you will see that you have been using functions already, but now it is time to learn how to start writing your own. Functions are a great building block that will reduce the amount of code you will need in your app. You can call a function whenever you need it, and you can write it as a kind of template with variables. So, depending on how you've written it, you can reuse it in many situations.

They do require you to think differently about the structure of your code and this can be hard, especially in the beginning. Once you have got the hang of this way of thinking, functions will really help you to write nicely structured, reusable, and low-maintenance code. Let's dive into this new abstraction layer!

Along the way, we will cover the following topics:

- Basic functions
- Function arguments
- Return
- Variable scope in functions
- Recursive functions
- Nested functions
- Anonymous functions
- Function callbacks

> Note: exercise, project and self-check quiz answers can be found in the *Appendix*.

Basic functions

We have been calling functions for a while already.
Remember `prompt()`, `console.log()`, `push()`, and `sort()` for arrays? These are all functions. Functions are a group of statements, variable declarations, loops, and so on that are bundled together. Calling a function means an entire group of statements will get executed.

First, we are going to have a look at how we can invoke functions, and then we will see how to write functions of our own.

Invoking functions

We can recognize functions by the parentheses at the end. We can invoke functions like this:

```
nameOfTheFunction();
functionThatTakesInput("the input", 5, true);
```

Copy

This is invoking a function called `nameOfTheFunction` with no arguments, and a function called `functionThatTakesInput` with three required arguments. Let's have a look at what functions can look like when we start writing them.

Writing functions

Writing a function can be done using the `function` keyword. Here is the template syntax to do so:

```
function nameOfTheFunction() {
    //content of the function
}
```

Copy

The above function can be called like this:

```
nameOfTheFunction();
```

Copy

Let's write a function that asks for your name and then greets you:

```
function sayHello() {
  let you = prompt("What's your name? ");
  console.log("Hello", you + "!");
}
```

Copy

We add a space after the question mark to ensure the user starts typing their answer one space away from the question mark, rather than directly afterward. We call this function like this:

```
sayHello();
```

Copy

It will prompt:

```
What's your name? >
```

Copy

Let's go ahead and enter our name. The output will be:

```
Hello Maaike!
```

Copy

Take a moment to consider the relationship between functions and variables. As you have seen, functions can contain variables, which shape how they operate. The opposite is also true: variables can contain functions. Still with me? Here you can see an example of a variable containing a function (`varContainingFunction`) and a variable inside a function (`varInFunction`):

```
let varContainingFunction = function() {
    let varInFunction = "I'm in a function.";
    console.log("hi there!", varInFunction);
};
varContainingFunction();
```

Copy

Variables contain a certain value and *are* something; they do not *do* anything. Functions are actions. They are a bundle of statements that can be executed when they get called. JavaScript will not run the statements when the functions do not get invoked. We will return to the idea of storing functions in variables, and consider some of the benefits, in the *Anonymous functions* section, but for now let's move on to look at the best way to name your functions.

Naming functions

Giving your function a name might seem like a trivial task, but there are some best practices to keep in mind here. To keep it short:

- Use camelCase for your functions: this means that the first word starts with a lowercase letter and new words start with a capital. That makes it a lot easier to read and keeps your code consistent.
- Make sure that the name describes what the function is doing: it's better to call a number addition function `addNumbers` than `myFunc`.
- Use a verb to describe what the function is doing: make it an action. So instead of `hiThere`, call it `sayHi`.

Practice exercise 6.1

See if you can write a function for yourself. We want to write a function that adds two numbers.

1. Create a function that takes two parameters, adds the parameters together, and returns the result.
2. Set up two different variables with two different values.
3. Use your function on the two variables, and output the result using `console.log`.
4. Create a second call to the function using two more numbers as arguments sent to the function.

Practice exercise 6.2

We are going to create a program that will randomly describe an inputted name.

1. Create an array of descriptive words.
2. Create a function that contains a prompt asking the user for a name.
3. Select a random value from the array using `Math.random`.
4. Output into the console the prompt value and the randomly selected array value.

5.Invoke the function.

Parameters and arguments

You may have noticed that we are talking about parameters and arguments. Both terms are commonly used to mean the information that is passed into a function:

```
function tester(para1, para2){
    return para1 + " " + para2;
}
const arg1 = "argument 1";
const arg2 = "argument 2";
tester(arg1, arg2);
```

Copy

A parameter is defined as the variable listed inside the parentheses of the function definition, which defines the scope of the function. They are declared like so:

```
function myFunc(param1, param2) {
  // code of the function;
}
```

Copy

A practical example could be the following, which takes `x` and `y` as parameters:

```
function addTwoNumbers(x, y) {
  console.log(x + y);
}
```

Copy

When called, this function will simply add the parameters and log the result. However, to do this, we can call the function with arguments:

```
myFunc("arg1", "arg2");
```

Copy

We have seen various examples of arguments; for example:

```
console.log("this is an argument");
prompt("argument here too");
let arr = [];
arr.push("argument");
```

Copy

Depending on the arguments you are calling with the function, the outcome of the function can change, which makes the function a very powerful and flexible building block. A practical example using our `addTwoNumbers()` function looks like this:

```
addTwoNumbers(3, 4);
addTwoNumbers(12,-90);
```

Copy

This will output:

```
7
-78
```

Copy

As you can see, the function has a different outcome for both calls. This is because we call it with different arguments, which take the place of `x` and `y`, that are sent to the function to be used within the function scope.

Practice exercise 6.3

Create a basic calculator that takes two numbers and one string value indicating an operation. If the operation equals add, the two numbers should be added. If the operation equals subtract, the two numbers should be subtracted from one another. If there is no option specified, the value of the option should be `add`.

The result of this function needs to be logged. Test your function by invoking it with different operators and no operator specified.

1. Set up two variables containing number values.
2. Set up a variable to hold an operator, either + or -.

3. Create a function that retrieves the two values and the operator string value within its parameters. Use those values with a condition to check if the operator is + or -, and add or subtract the values accordingly (remember if not presented with a valid operator, the function should default to addition).
4. Within `console.log()`, call the function using your variables and output the response to the console.
5. Update the operator value to be the other operator type—either plus or minus—and call to the function again with the new updated arguments.

Default or unsuitable parameters

What happens if we call our `addTwoNumbers()` function without any arguments? Take a moment and decide what you think this should do:

```
addTwoNumbers();
```
Copy

Some languages might crash and cry, but not JavaScript. JavaScript just gives the variables a default type, which is undefined. And `undefined` + `undefined` equals:

```
NaN
```
Copy

Instead, we could tell JavaScript to take different default parameters. And that can be done like this:

```
function addTwoNumbers(x = 2, y = 3) {
  console.log(x + y);
}
```
Copy

If you call the function with no arguments now, it will automatically assign `2` to `x` and `3` to `y`, unless you override them by calling the function with arguments. The values that are used for invoking are prioritized over hardcoded arguments. So, given the above function, what will the output of these function calls be?

```
addTwoNumbers();
addTwoNumbers(6, 6);
addTwoNumbers(10);
```
Copy

The output will be:

```
5
12
13
```
Copy

The first one has the default values, so `x` is `2` and `y` is `3`. The second one assigns `6` to both `x` and `y`. The last one is a bit less obvious. We are only giving one argument, so which one will be given this value? Well, JavaScript does not like to overcomplicate things. It simply assigns the value to the first parameter, `x`.
Therefore, `x` becomes `10` and `y` gets its default value `3`, and together that makes `13`.

If you call a function with more arguments than parameters, nothing will happen. JavaScript will just execute the function using the first arguments that can be mapped to parameters. Like this:

```
addTwoNumbers(1,2,3,4);
```
Copy

This will output:

```
3
```
Copy

It is just adding 1 and 2 and ignoring the last two arguments (`3` and `4`).

Special functions and operators

There are a few special ways of writing functions, as well as some special operators that will come in handy. We are talking about arrow functions and the spread and rest operators here. Arrow functions are great for sending functions around as parameters and using shorter notations. The spread and rest operators make our lives easier and are more flexible when sending arguments and working with arrays.

Arrow functions

Arrow functions are a special way of writing functions that can be confusing at first. Their use looks like this:

```
(param1, param2) => body of the function;
```

Copy

Or for no parameters:

```
() => body of the function;
```

Copy

Or for one parameter (no parentheses are needed here):

```
param => body of the function;
```

Copy

Or for a multiline function with two parameters:

```
(param1, param2) => {
  // line 1;
  // any number of lines;
};
```

Copy

Arrow functions are useful whenever you want to write an implementation on the spot, such as inside another function as an argument. This is because they are a shorthand notation for writing functions. They are most often used for functions that consist of only one statement. Let's start with a simple function that we will rewrite to an arrow function:

```
function doingStuff(x) {
  console.log(x);
}
```

Copy

To rewrite this as an arrow function, you will have to store it in a variable or send it in as an argument if you want to be able to use it. We use the name of the variable to execute the arrow function. In this case we only have one parameter, so it's optional to surround it with parentheses. We can write it like this:

```
let doingArrowStuff = x => console.log(x);
```

Copy

And invoke it like this:

```
doingArrowStuff("Great!");
```

Copy

This will log `Great!` to the console. If there is more than one argument, we will have to use parentheses, like this:

```
let addTwoNumbers = (x, y) => console.log(x + y);
```

Copy

We can call it like this:

```
addTwoNumbers(5, 3);
```

Copy

And then it will log `8` to the console. If there are no arguments, you must use the parentheses, like this:

```
let sayHi = () => console.log("hi");
```

Copy

If we call `sayHi()`, it will log `hi` to the console.

As a final example, we can combine the arrow function with certain built-in methods. For example, we can use the `foreach()` method on an array. This method executes a certain function for every element in the array. Have a look at this example:

```
const arr = ["squirrel", "alpaca", "buddy"];

arr.forEach(e => console.log(e));
```

Copy

It outputs:

```
squirrel
alpaca
buddy
```

Copy

For every element in the array, it takes the element as input and executing the arrow function for it. In this case, the function is to log the element. So the output is every single element in the array.

Using arrow functions combined with built-in functions is very powerful. We can do something for every element in the array, without counting or writing a complicated loop. We'll see more examples of great use cases for arrow functions later on.

Spread operator

The spread operator is a special operator. It consists of three dots used before a referenced expression or string, and it spreads out the arguments or elements of an array.

This might sound very complicated, so let's look at a simple example:

```
let spread = ["so", "much", "fun"];
let message = ["JavaScript", "is", ...spread, "and",
"very","powerful"];
```

Copy

The value of this array becomes:

```
['JavaScript', 'is', 'so', 'much', 'fun', 'and', 'very', 'powerful']
```

Copy

As you can see, the elements of the spread operator become individual elements in the array. The spread operator spreads the array to individual elements in the new array. It can also be used to send multiple arguments to a function, like this:

```
function addTwoNumbers(x, y) {
  console.log(x + y);
}
```

```
let arr = [5, 9];
addTwoNumbers(...arr);
```

Copy

This will log `14` to the console, since it is the same as calling the function with:

```
addTwoNumbers(5, 9);
```

Copy

This operator avoids having to copy a long array or string into a function, which saves time and reduces code complexity. You can call a function with multiple spread operators. It will use all the elements of the arrays as input. Here's an example:

```
function addFourNumbers(x, y, z, a) {
  console.log(x + y + z + a);
}
let arr = [5, 9];
let arr2 = [6, 7];
addFourNumbers(...arr, ...arr2);
```

Copy

This will output `27` to the console, calling the function like this:

```
addFourNumbers(5, 9, 6, 7);
```

Copy

Rest parameter

Similar to the spread operator, we have the rest parameter. It has the same symbol as the spread operator, but it is used inside the function parameter list. Remember what would happen if we were to send an argument too many times, as here:

```
function someFunction(param1, param2) {
  console.log(param1, param2);
}
```

```
someFunction("hi", "there!", "How are you?");
```

Copy

That's right. Nothing really: it would just pretend we only sent in two arguments and log `hi there!`. If we use the rest parameter, it allows us to send in any number of arguments and translate them into a parameter array. Here is an example:

```
function someFunction(param1, ...param2) {
  console.log(param1, param2);
}
someFunction("hi", "there!", "How are you?");
```

Copy

This will log:

```
hi [ 'there!', 'How are you?' ]
```

Copy

As you can see, the second parameter has changed into an array, containing our second and third arguments. This can be useful whenever you are not sure what number of arguments you will get. Using the rest parameter allows you to process this variable number of arguments, for example, using a loop.

Returning function values

We are still missing a very important piece to make functions as useful as they are: the return value. Functions can give back a result when we specify a return value. The return value can be stored in a variable. We have done this already – remember `prompt()`?

```
let favoriteSubject = prompt("What is your favorite subject?");
```

Copy

We are storing the result of our `prompt()` function in the variable `favoriteSubject`, which in this case would be whatever the user specifies. Let's see what happens if we store the result of our `addTwoNumbers()` function and log that variable:

```
let result = addTwoNumbers(4, 5);
```

```
console.log(result);
```

Copy

You may or may not have guessed it—this logs the following:

```
9
undefined
```

Copy

The value `9` is written to the console because `addTwoNumbers()` contains a `console.log()` statement. The `console.log(result)` line outputs `undefined`, because nothing is inserted into the function to store the result, meaning our function `addTwoNumbers()` does not send anything back. Since JavaScript does not like to cause trouble and crash, it will assign `undefined`. To counter this, we can rewrite our `addTwoNumbers()` function to actually return the value instead of logging it. This is much more powerful because we can store the result and continue working with the result of this function in the rest of our code:

```
function addTwoNumbers(x, y) {
  return x + y;
}
```

Copy

`return` ends the function and sends back whatever value comes after `return`. If it is an expression, like the one above, it will evaluate the expression to one result and then return that to where it was called (the `result` variable, in this instance):

```
let result = addTwoNumbers(4, 5);
console.log(result);
```

Copy

With these adjustments made, the code snippet logs `9` to the terminal.

What do you think this code does?

```
let resultsArr = [];
for(let i = 0; i < 10; i ++){
  let result = addTwoNumbers(i, 2*i);
```

```
  resultsArr.push(result);
}
console.log(resultsArr);
```

Copy

It logs an array of all the results to the screen. The function is being called in a loop. The first iteration, `i`, equals `0`. Therefore, the result is `0`. The last iteration, `i`, equals `9`, and therefore the last value of the array equals `27`. Here are the results:

```
[
   0,  3,  6,  9, 12,
  15, 18, 21, 24, 27
]
```

Copy

Practice exercise 6.4

Modify the calculator that you made in *Practice exercise 6.2* to return added values instead of printing them. Then, call the function 10 or more times in a loop, and store the results in an array. Once the loop finishes, output the final array into the console.

1. Set up an empty array to store the values that will be calculated within the loop.
2. Create a loop that runs 10 times, incrementing by 1 each time, creating two values each iteration. For the first value, multiply the value of the loop count by 5. For the second value, multiply the value of the loop counter by itself.
3. Create a function that returns the value of the two parameters passed into the function when it is called. Add the values together, returning the result.
4. Within the loop, call the calculation function, passing in the two values as arguments into the function and storing the returned result in a response variable.

5. Still within the loop, push the result values into the array as it iterates through the loop.
6. After the loop is complete, output the value of the array into the console.
7. You should see the values `[0, 6, 14, 24, 36, 50, 66, 84, 104, 126]` for the array in the console.

Returning with arrow functions

If we have a one-line arrow function, we can return without using the keyword `return`. So if we want to rewrite the function, we can write it like this to make an arrow function out of it:

```
let addTwoNumbers = (x, y) => x + y;
```

Copy

And we can call it and store the result like this:

```
let result = addTwoNumbers(12, 15);
console.log(result);
```

Copy

This will then log `27` to the console. If it's a multiline function, you will have to use the keyword `return` as demonstrated in the previous section. So, for example:

```
let addTwoNumbers = (x, y) => {
  console.log("Adding...");
  return x + y;
}
```

Copy

Variable scope in functions

In this section, we will discuss a topic that is often considered challenging. We will talk about scope. Scope defines where you can access a certain variable. When a variable is *in scope*, you can access it. When a variable is *out of scope*, you cannot access the variable. We will discuss this for both local and global variables.

Local variables in functions

Local variables are only in scope within the function they are defined. This is true for `let` variables and `var` variables. There is a difference between them, which we will touch upon here as well. The function parameters (they do not use `let` or `var`) are also local variables. This might sound very vague, but the next code snippet will demonstrate what this means:

```
function testAvailability(x) {
  console.log("Available here:", x);
}
testAvailability("Hi!");
console.log("Not available here:", x);
```

Copy

This will output:

```
Available here: Hi!
ReferenceError: x is not defined
```

Copy

When called inside the function, `x` will be logged. The statement outside of the function fails, because `x` is a local variable to the function `testAvailability()`. This is showing that the function parameters are not accessible outside of the function.

They are out of scope outside the function and in scope inside the function. Let's have a look at a variable defined inside a function:

```
function testAvailability() {
  let y = "Local variable!";
  console.log("Available here:", y);
}
testAvailability();
console.log("Not available here:", y);
```

Copy

This shows the following on the console:

```
Available here: Local variable!
ReferenceError: y is not defined
```

Copy

Variables defined inside the function are not available outside the function either.

For beginners, it can be confusing to combine local variables and `return`. Right now, we're telling you the local variables declared inside a function are not available outside of the function, but with `return` you can make their values available outside the function. So if you need their values outside a function, you can return the values. The key word here is *values*! You cannot return the variable itself. Instead, a value can be caught and stored in a different variable, like this:

```
function testAvailability() {
  let y = "I'll return";
  console.log("Available here:", y);
  return y;
}
let z = testAvailability();
console.log("Outside the function:", z);
console.log("Not available here:", y);
```

Copy

So, the returned value `I'll return` that was assigned to local variable `y` gets returned and stored in variable `z`.

This variable `z` could actually also have been called `y`, but that would have been confusing since it still would have been a different variable.

The output of this code snippet is as follows:

```
Available here: I'll return
Outside the function: I'll return
ReferenceError: y is not defined
```

Copy

let versus var variables

The difference between `let` and `var` is that `var` is function-scoped, which is the concept we described above. `let` is actually not function-scoped but block-scoped. A block is defined by two curly braces `{ }`. The code within those braces is where `let` is still available.

Let's see this distinction in action:

```
function doingStuff() {
  if (true) {
    var x = "local";
  }
  console.log(x);
}
doingStuff();
```

Copy

The output of this snippet will be:

```
local
```

Copy

If we use `var`, the variable becomes function-scoped and is available anywhere in the function block (even before defining with the value undefined). Thus, after the `if` block has ended, `x` can still be accessed.

Here is what happens with `let`:

```
function doingStuff() {
  if (true) {
    let x = "local";
  }
  console.log(x);
}
```

```
doingStuff();
```

Copy

This will produce the following output:

```
ReferenceError: x is not defined
```

Copy

Here we get the error that `x is not defined`. Since `let` is only block-scoped, `x` goes out of scope when the `if` block ends and can no longer be accessed after that.

A final difference between `let` and `var` relates to the order of declaration in a script. Try using the value of `x` before having defined it with `let`:

```
function doingStuff() {
  if (true) {
    console.log(x);
    let x = "local";
  }
}
doingStuff();
```

Copy

This will give a `ReferenceError` that `x` is not initialized. This is because variables declared with `let` cannot be accessed before being defined, even within the same block. What do you think will happen for a `var` declaration like this?

```
function doingStuff() {
  if (true) {
    console.log(x);
    var x = "local";
  }
}
doingStuff();
```

Copy

This time, we won't get an error. When we use a `var` variable before the `define` statement, we simply get `undefined`. This is due to a phenomenon called hoisting, which means using a `var` variable before it's been declared results in the variable being `undefined` rather than giving a `ReferenceError`.

Hoisting, and how to negate its effects if needed, are more complex topics that we will cover in *Chapter 12*, *Intermediate JavaScript*.

const scope

Constants are block-scoped, just like `let`. This is why the scope rules here are similar to those for `let`. Here is an example:

```
function doingStuff() {
  if (true) {
    const X = "local";
  }
  console.log(X);
}
doingStuff();
```

Copy

This will produce the following output:

```
ReferenceError: X is not defined
```

Copy

Using a `const` variable before having defined it will also give a `ReferenceError`, just as it does for a `let` variable.

Global variables

As you might have guessed, global variables are variables declared outside a function and not in some other code block. Variables are accessible in the scope (either function or block) where they're defined, plus any "lower" scopes. So, a variable defined outside of a function is available within the function as well as inside any functions or other code blocks inside that function. A variable defined at the top

level of your program is therefore available everywhere in your program. This concept is called a global variable. You can see an example here:

```
let globalVar = "Accessible everywhere!";
console.log("Outside function:", globalVar);
function creatingNewScope(x) {
  console.log("Access to global vars inside function." , globalVar);
}
creatingNewScope("some parameter");
console.log("Still available:", globalVar);
```

Copy

This will output:

```
Outside function: Accessible everywhere!
Access to global vars inside function. Accessible everywhere!
Still available: Accessible everywhere!
```

Copy

As you can see, global variables are accessible from everywhere because they are not declared in a block. They are *always* in scope after they have been defined—it doesn't matter where you use them. However, you can hide their accessibility inside a function by specifying a new variable with the same name inside that scope; this can be done for `let`, `var`, and `const`. (This is not changing the value of the `const` variable; you are creating a new `const` variable that is going to override the first one in the inner scope.) In the same scope, you cannot specify two `let` or two `const` variables with the same name. You can do so for `var`, but you shouldn't do so, in order to avoid confusion.

If you create a variable with the same name inside a function, that variable's value will be used whenever you refer to that variable name within the scope of that particular function. Here you can see an example:

```
let x = "global";
function doingStuff() {
  let x = "local";
```

```
  console.log(x);
}
doingStuff();
console.log(x);
```

Copy

This will output:

```
local
global
```

Copy

As you can see, the value of `x` inside the `doingStuff()` function is `local`. However, outside the function the value is still `global`. This means that you'll have to be extra careful about mixing up names in local and global scopes. It is usually better to avoid this.

The same is also true for parameter names. If you have the same parameter name as a global variable, the value of the parameter will be used:

```
let x = "global";

function doingStuff(x) {
  console.log(x);
}

doingStuff("param");
```

Copy

This will log `param`.

There is a danger in relying on global variables too much. This is something you will come across soon when your applications grow. As we just saw, local variables override the value of global variables. It is best to work with local variables in functions; this way, you have more control over what you are working with. This might be a bit vague for now, but it will become clear when coding in the wild as things get bigger and more lines and files of code get involved.

There is only one more very important point to be made about scopes for now. Let's start with an example and see if you can figure out what this should log:

```
function confuseReader() {
  x = "Guess my scope...";
  console.log("Inside the function:", x);
}
confuseReader();
console.log("Outside of function:", x);
```

Copy

Answer ready? Here is the output:

```
Inside the function: Guess my scope...
Outside of function: Guess my scope...
```

Copy

Do not close the book—we'll explain what is going on. If you look carefully, the `x` in the function gets defined without the keyword `let` or `var`. There is no declaration of `x` above the code; this is all the code of the program. JavaScript does not see `let` or `var` and then decides, "this must be a global variable." Even though it gets defined inside the function, the declaration of `x` within the function gets global scope and can still be accessed outside of the function.

We really want to emphasize that this is a terrible practice. If you need a global variable, declare it at the top of your file.

Immediately invoked function expression

The **immediately invoked function expression** (**IIFE**) is a way of expressing a function so that it gets invoked immediately. It is anonymous, it doesn't have a name, and it is self-executing.

This can be useful when you want to initialize something using this function. It is also used in many design patterns, for example, to create private and public variables and functions.

This has to do with where functions and variables are accessible from. If you have an IIFE in the top-level scope, whatever is in there is not accessible from outside even though it is top level.

Here is how to define it:

```
(function () {
  console.log("IIFE!");
})();
```

Copy

The function itself is surrounded by parentheses, which makes it create a function instance. Without these parentheses around it, it would throw an error because our function does not have a name (this is worked around by assigning the function to a variable, though, where the output can be returned to the variable).

`();` executes the unnamed function—this must be done immediately following a function declaration. If your function were to require a parameter, you would pass it in within these final brackets.

You could also combine IIFE with other function patterns. For example, you could use an arrow function here to make the function even more concise:

```
(()=>{
    console.log("run right away");
})();
```

Copy

Again, we use `();` to invoke the function that you created.

Practice exercise 6.5

Use IIFE to create a few immediately invoked functions and observe how the scope is affected.

1. Create a variable value with `let` and assign a string value of 1000 to it.

2. Create an IIFE function and within this function scope assign a new value to a variable of the same name. Within the function, print the local value to the console.
3. Create an IIFE expression, assigning it to a new `result` variable, and assign a new value to a variable of the same name within this scope. Return this local value to the `result` variable and invoke the function. Print the `result` variable, along with the variable name you've been using: what value does it contain now?
4. Lastly, create an anonymous function that has a parameter. Add logic that will assign a passed-in value to the same variable name as the other steps, and print it as part of a string sentence. Invoke the function and pass in your desired value within the rounded brackets.

Recursive functions

In some cases, you want to call the same function from inside the function. It can be a beautiful solution to rather complex problems. There are some things to keep in mind though. What do you think this will do?

```
function getRecursive(nr) {
  console.log(nr);
  getRecursive(--nr);
}
getRecursive(3);
```

Copy

It prints `3` and then counts down and never stops. Why is it not stopping? Well, we are not saying when it should stop. Look at our improved version:

```
function getRecursive(nr) {
  console.log(nr);
  if (nr > 0) {
```

```
    getRecursive(--nr);
  }
}
getRecursive(3);
```

Copy

This function is going to call itself until the value of the parameter is no longer bigger than `0`. And then it stops.

What happens when we call a function recursively is that it goes one function deeper every time. The first function call is done last. For this function it goes like this:

- `getRecursive(3)`
  - `getRecursive(2)`
    - `getRecursive(1)`
      - `getRecursive(0)`
      - done with `getRecursive(0)` execution
    - done with `getRecursive(1)` execution
  - done with `getRecursive(2)` execution
- done with `getRecursive(3)` execution

The next recursive function will demonstrate that:

```
function logRecursive(nr) {
  console.log("Started function:", nr);
  if (nr > 0) {
    logRecursive(nr - 1);
  } else {
      console.log("done with recursion");
  }
  console.log("Ended function:", nr);
}
logRecursive(3);
```

Copy

It will output:

```
Started function: 3
Started function: 2
Started function: 1
Started function: 0
done with recursion
Ended function: 0
Ended function: 1
Ended function: 2
Ended function: 3
```

Copy

Recursive functions can be great in some contexts. When you feel the need to call the same function over and over again in a loop, you should probably consider recursion. An example could also be searching for something. Instead of looping over everything inside the same function, you can split up inside the function and call the function repeatedly from the inside.

However, it must be kept in mind that in general, the performance of recursion is slightly worse than the performance of regular iteration using a loop. So if this causes a bottleneck situation that would really slow down your application, then you might want to consider another approach.

Have a look at calculating the factorial using recursive functions in the following exercise.

Practice exercise 6.6

A common problem that we can solve with recursion is calculating the factorial.

Quick mathematics refresher about factorials:

The factorial of a number is the product of all positive integers bigger than 0, up to the number itself. So for example, the factorial of seven is 7 * 6 * 5 * 4 * 3 * 2 * 1. You can write this as 7!.

How are recursive functions going to help us calculate the factorial? We are going to call the function with a lower number until we reach 0. In this exercise, we will use recursion to calculate the factorial result of a numeric value set as the argument of a function.

1. Create a function that contains a condition within it checking if the argument value is `0`.
2. If the parameter is equal to `0`, it should return the value of `1`. Otherwise, it should return the value of the argument multiplied by the value returned from the function itself, subtracting one from the value of the argument that is provided. This will result in running the block of code until the value reaches `0`.
3. Invoke the function, providing an argument of whatever number you want to find the factorial of. The code should run whatever number is passed initially into the function, decreasing all the way to `0` and outputting the results of the calculation to the console. It could also contain a `console.log()` call to print the current value of the argument in the function as it gets invoked.
4. Change and update the number to see how it affects the results.

Nested functions

Just as with loops, `if` statements, and actually all other building blocks, we can have functions inside functions. This phenomenon is called nested functions:

```
function doOuterFunctionStuff(nr) {
  console.log("Outer function");
  doInnerFunctionStuff(nr);
  function doInnerFunctionStuff(x) {
    console.log(x + 7);
    console.log("I can access outer variables:", nr);
  }
}
```

```
doOuterFunctionStuff(2);
```

Copy

This will output:

```
Outer function
9
I can access outer variables: 2
```

Copy

As you can see, the outer function is calling its nested function. This nested function has access to the variables of the parent. The other way around, this is not the case. Variables defined inside the inner function have function scope. This means they are accessible inside the function where they are defined, which is in this case the inner function. Thus, this will throw a `ReferenceError`:

```
function doOuterFunctionStuff(nr) {
  doInnerFunctionStuff(nr);
  function doInnerFunctionStuff(x) {
    let z = 10;
  }
  console.log("Not accessible:", z);
}
doOuterFunctionStuff(2);
```

Copy

What do you think this will do?

```
function doOuterFunctionStuff(nr) {
  doInnerFunctionStuff(nr);
  function doInnerFunctionStuff(x) {
    let z = 10;
  }
}
```

```
doInnerFunctionStuff(3);
```

Copy

This will also throw a `ReferenceError`. Now, `doInnerFunctionStuff()` is defined inside the outer function, which means that it is only in scope inside `doOuterFunctionStuff()`. Outside this function, it is out of scope.

Practice exercise 6.7

Create a countdown loop starting at a dynamic value of `10`.

1. Set the `start` variable at a value of `10`, which will be used as the starting value for the loop.
2. Create a function that takes one argument, which is the countdown value.
3. Within the function, output the current value of the countdown into the console.
4. Add a condition to check if the value is less than 1; if it is, then return the function.
5. Add a condition to check if the value of the countdown is not less than 1, then continue to loop by calling the function within itself.
6. Make sure you add a decrement operator on the countdown so the preceding condition eventually will be true to end the loop. Every time it loops, the value will decrease until it reaches 0.
7. Update and create a second countdown using a condition if the value is greater than 0. If it is, decrease the value of the countdown by 1.
8. Use `return` to return the function, which then invokes it again and again until the condition is no longer true.
9. Make sure, when you send the new countdown value as an argument into the function, that there is a way out of the loop by using the `return` keyword and a condition that continues the loop if met.

Anonymous functions

So far, we have been naming our functions. We can also create functions without names if we store them inside variables. We call these functions anonymous. Here is a non-anonymous function:

```
function doingStuffAnonymously() {
  console.log("Not so secret though.");
}
```
Copy

Here is how to turn the previous function into an anonymous function:

```
function () {
  console.log("Not so secret though.");
};
```
Copy

As you can see, our function has no name. It is anonymous. So you may wonder how you can invoke this function. Well actually, you can't like this!

We will have to store it in a variable in order to call the anonymous function; we can store it like this:

```
let functionVariable = function () {
  console.log("Not so secret though.");
};
```
Copy

An anonymous function can be called using the variable name, like this:

```
functionVariable();
```
Copy

It will simply output `Not so secret though.`.

This might seem a bit useless, but it is a very powerful JavaScript construct. Storing functions inside variables enables us to do very cool things, like passing in functions

as parameters. This concept adds another abstract layer to coding. This concept is called callbacks, and we will discuss it in the next section.

Practice exercise 6.8

1. Set a variable name and assign a function to it. Create a function expression with one parameter that outputs a provided argument to the console.
2. Pass an argument into the function.
3. Create the same function as a normal function declaration.

Function callbacks

Here is an example of passing a function as an argument to another function:

```
function doFlexibleStuff(executeStuff) {
  executeStuff();
  console.log("Inside doFlexibleStuffFunction.");
}
```

Copy

If we call this new function with our previously made anonymous function, `functionVariable`, like this:

```
doFlexibleStuff(functionVariable);
```

Copy

It will output:

```
Not so secret though.
Inside doFlexibleStuffFunction.
```

Copy

But we can also call it with another function, and then our `doFlexibleStuff` function will execute this other function. How cool is that?

```
let anotherFunctionVariable = function() {
```

```
  console.log("Another anonymous function implementation.");
}

doFlexibleStuff(anotherFunctionVariable);
```

Copy

This will produce the following output:

```
Another anonymous function implementation.
Inside doFlexibleStuffFunction.
```

Copy

So what happened? We created a function and stored it in the `anotherFunctionVariable` variable. We then sent that in as a function parameter to our `doFlexibleStuff()` function. And this function is simply executing whatever function gets sent in.

At this point you may wonder why the writers are so excited about this callback concept. It probably looks rather lame in the examples you have seen so far. Once we get to asynchronous functions later on, this concept is going to be of great help. To still satisfy your need for a more concrete example, we will give you one.

In JavaScript, there are many built-in functions, as you may know by now. One of them is the `setTimeout()` function. It is a very special function that is executing a certain function after a specified amount of time that it will wait first. It is also seemingly responsible for quite a few terribly performing web pages, but that is definitely not the fault of this poor misunderstood and misused function.

This code is really something you should try to understand:

```
let youGotThis = function () {
  console.log("You're doing really well, keep coding!");
};

setTimeout(youGotThis, 1000);
```

Copy

It is going to wait for `1000`ms (one second) and then print:

```
You're doing really well, keep coding!
```

Copy

If you need more encouragement, you can use the `setInterval()` function instead. It works very similarly, but instead of executing the specified function once, it will keep on executing it with the specified interval:

```
setInterval(youGotThis, 1000);
```

Copy

In this case, it will print our encouraging message every second until you kill the program.

This concept of the function executing the function after having been called itself is very useful for managing asynchronous program execution.

Chapter projects

Create a recursive function

Create a recursive function that counts up to 10. Invoke the function with different start numbers as the arguments that are passed into the function. The function should run until the value is greater than 10.

Set timeout order

Use the arrow format to create functions that output the values `one` and `two` to the console. Create a third function that outputs the value `three` to the console, and then invokes the first two functions.

Create a fourth function that outputs the word `four` to the console and also use `setTimeout()` to invoke the first function immediately and then the third function.

What does your output look like in the console? Try to get the console to output:

```
Four
Three
One
Two
```

```
One
```

Copy

Self-check quiz

1. What value is output into the console?

```
let val = 10;
function tester(val){
    val += 10;
    if(val < 100){
        return tester(val);
    }
    return val;
}
tester(val);
console.log(val);
```

Copy

12. What will be output into the console by the below code?

```
let testFunction = function(){
    console.log("Hello");
}();
```

Copy

16. What will be output to the console?

```
(function () {
    console.log("Welcome");
})();
(function () {
    let firstName = "Laurence";
})();
let result = (function () {
```

```
24.             let firstName = "Laurence";
25.             return firstName;
26.         })();
27.         console.log(result);
28.         (function (firstName) {
29.             console.log("My Name is " + firstName);
30.         })("Laurence");
```

Copy

31. What will be output to the console?

```
32.         let test2 = (num) => num + 5;
33.         console.log(test2(14));
```

Copy

34. What will be output to the console?

```
35.         var addFive1 = function addFive1(num) {
36.         return num + 2;
37.         };
38.         let addFive2 = (num) => num + 2;
39.         console.log(addFive1(14));
```

Copy

Summary

In this chapter, we have covered functions. Functions are a great JavaScript building block that we can use to reuse lines of code. We can give our functions parameters, so that we can change the code depending on the arguments a function gets invoked with. Functions can return a result; we do so using the `return` keyword. And we can use `return` at the place where we call a function. We can store the result in a variable or use it in another function, for example.

We then met with variable scopes. The scope entails the places from where variables are accessible. Default `let` and `const` variables can be accessed inside the block where they're defined (and the inner blocks of that block) and `var` is just accessible from the line where it was defined.

We can also use recursive functions to elegantly solve problems that can be solved recursively by nature, such as calculating the factorial. Nested functions were the next topic we studied. They are not a big deal, just functions inside functions. Basic functions inside functions are not considered very pretty, but anonymous functions and arrow functions are not uncommon to see. Anonymous functions are functions without a name and arrow functions are a special case of anonymous functions, where we use an arrow to separate the parameters and the body.

In the next chapter, we'll consider classes, another powerful programming construct!

Join our book's Discord space

Join the book's Discord workspace for a monthly *Ask me Anything* session with the authors:

https://packt.link/JSBook

