Videos Youtube de Paradigma de Programación Orientada a Objetos

1. Clases y Objetos. https://www.youtube.com/embed/cfGrtrdDGRk
2. Abstracción. https://www.youtube.com/embed/oBO01Cx_YwQ
3. Herencia. https://www.youtube.com/embed/9NynVRpZzv4
4. Encapsulamiento. https://www.youtube.com/embed/gR0EssHrl24
5. Polimorfismo. https://www.youtube.com/embed/tjjecfz9Cvk

TALLER DE ESTRUCTURAS DE CONTROL EN LA SOLUCIÓN DE ALGORITMOS

1. Para los siguientes ejercicios de análisis realizar el pseudocodigo en lenguaje natural (papel y lapiz) y adjuntar la imagen del análisis de los siguientes acertijos:

    1.1. En este ejercicio, se debe comprender y solucionar el acertijo, después de solucionarlo, se debe realizar el pseudocodigo de la solución en lenguaje natural y adjuntar la imagen del análisis del acertijo

Un pastor tiene un inconveniente, necesita pasar al otro lado del rio una paca de heno, un lobo y una oveja, para trasladar cada uno de los elementos cuenta con una barca en la que solo puede transportar uno de los elementos y el pastor.

¿ Como el pastor puede trasladar al lobo, el heno y la oveja evitando que los elementos se hagan daño entre si?, teniendo en cuenta que el lobo come ovejas y la oveja come heno.

    1.2. Número de 3 dígitos. El dígito que está en el medio es 4 veces mayor que el tercer y último dígito. Además, el primero es 3 unidades más pequeño que el segundo. ¿Qué número es?.

    1.3. La suma de un palo y una pelota de golf es de 1'10 euros. El palo supera el precio de la pelota en un euro. ¿Cuánto cuesta la pelota?.

    1.4. Un granjero tiene 40 cerdos, 10 conejos y 20 caballos. Si llamas 'caballos' a los 'cerdos', ¿cuántos nuevos caballos tendrá ahora?

    1.5. Un señor y sus 2 hijos quieren pasar un río en una balsa, pero la balsa solo aguanta 80kg. Él pesa 80Kg y cada uno de sus hijos 40kg ¿Cómo pasarán?¿teniendo que llevar la balsa otra vez de vuelta para que pasen todos?

    1.6. Si 5 máquinas hacen 5 artículos en 5 minutos, ¿cuánto tiempo dedicarán 100 máquinas en hacer 100 artículos?.

    1.7. Juan se levanta por la mañana y descubre que la luz de la habitación no funciona. Abre el cajón de los guantes, en el que hay diez guantes negros y diez azul oscuro. ¿Cuántos debe coger para asegurarse de que obtiene un par del mismo color?

    1.8. Tu conduces un autobús, en el que se montan 18 personas. En la siguiente parada, se bajan 5 pero suben otras 13. Al llegar a la siguiente estación, se bajan 21 y se suben otras 4. ¿De qué color son los ojos del conductor?

2. Realizar el diagrama de flujo adjuntando la imagen y escriba el pseudocodigo de los siguientes ejercicios utilizando Pseint.

    2.1. Sumar dos números ingresados por teclado y mostrar la suma de ellos.

    2.2. Leer una secuencia de números hasta que se ingrese un numero negativo, debe calcular y mostrar la cantidad de números ingresados.

2.3. Determina el área de un cuadrado.

2.4. Determina el perímetro de un cuadrado.

2.5. Realiza el algoritmo necesario para que el ordenador te pregunte tu nombre y te responda con un saludo personalizado concatenando el nombre ingresado.

2.6. Ingresar 10 números positivos, y realizar la suma.

2.7. Ingresar caracteres por teclado hasta que se ingrese el carácter (-) y imprimir todas las letras ingresadas.

2.8. Ingresar un numero entero positivo y validar si es para o impar.

2.9. Ingresar un numero entero positivo y validar si es primo.

3. Para los siguientes ejercicios utilizar la herramienta Pseint.

3.1. Leer 10 numero y imprimir la suma.

3.2. Leer 10 numeros y imprimir la suma de los pares e impares.

3.3. Ingresar 20 numeros y calcular el promedio de los numeros.

3.4. Calcular el area de un ciculo.

3.5. Calcular el area de un cuadrado.

3.6. Adivinar un numero, el programa genera un numero aleatorio del 1 al 100, el usuario debe ingresar numeros y va validando por cada numero si es mayor o menor al numero que se debe adivinar, el programa debe permitir maximo 10 intentos y terminara al momento de hallar el numero a adivinar con un mensaje de éxito y si se ingresan los 10 numeros maximos debe mostrar el numero seleccionado por el sistema.

3.7. El programa debe pedir el numero de elementos que se van a ingresar, despues debe ingresarse numero por numero y debe ir contando los numeros mayoresa a cero, numero menores a cero e iguales a cero, debe mostrar cuantos numeros se ingresaron según las condiciones dadas.

4. Utilizar Pseint y escribir el pseudocodigo de los siguientes ejercicios:

4.1. Se quiere realizar un programa que lea por teclado las 5 notas obtenidas por un alumno (comprendidas entre 0 y 10). A continuación debe mostrar todas las notas, la nota media, la nota más alta que ha sacado y la menor.

4.2. Crear un vector de 5 elementos de cadenas de caracteres, inicializa el vector con datos leídos por el teclado. Copia los elementos del vector en otro vector pero en orden inverso, y muéstralo por la pantalla.

4.3. Hacer un programa que inicialice un vector de números con valores aleatorios, y posterior ordene los elementos de menor a mayor.

4.4. Crea un array o arreglo unidimensional con un tamaño de 5, asignale los valores numéricos manualmente (los que tu quieras) y muestralos por pantalla.

4.5. Crea un array o arreglo unidimensional donde tu le indiques el tamaño por teclado y crear una función que rellene el array o arreglo con los múltiplos de un numero pedido por teclado. Por ejemplo, si defino un array de tamaño 5 y eligo un 3 en la función, el array contendrá 3, 6, 9, 12, 15. Muestralos por pantalla usando otra función distinta.

TALLER DE APLICACIONES DE PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

Realizar correctamente los ejercicios de programación básica utilizando funciones y procedimientos según como se especifique en cada uno de los ejercicios, para resolver estos ejercicios resolverlos en Pseint y adjuntar los ejercicios en una carpeta comprimida como entregale.

1. Crea un programa que pida dos número enteros al usuario y diga si alguno de ellos es múltiplo del otro. Crea una función EsMultiplo que reciba los dos números, y devuelve si el primero es múltiplo del segundo.

2. Crea un función “ConvertirEspaciado”, que reciba como parámetro un texto y devuelve una cadena con un espacio adicional tras cada letra. Por ejemplo, “Hola, tú” devolverá “H o l a , t ú “. Crea un programa principal donde se use dicha función.

3. Crea una función “calcularMaxMin” que recibe una arreglo con valores numérico y devuelve el valor máximo y el mínimo. Crea un programa que pida números por teclado y muestre el máximo y el mínimo, utilizando la función anterior.

4. Diseñar una función que calcule el área y el perímetro de una circunferencia. Utiliza dicha función en un programa principal que lea el radio de una circunferencia y muestre su área y perímetro.

5. Crear una subrutina llamada “Login”, que recibe un nombre de usuario y una contraseña y te devuelve Verdadero si el nombre de usuario es “usuario1” y la contraseña es “asdasd”. Además recibe el número de intentos que se ha intentado hacer login y si no se ha podido hacer login incremente este valor.
   Crear un programa principal donde se pida un nombre de usuario y una contraseña y se intente hacer login, solamente tenemos tres oportunidades para intentarlo.

6. Crear una calculadora que en el programa principal pida dos números, y cree una función llamada sumarNumeros, debe recibir dos números enteros y devolver el resultado, una función restarNumeros, debe recibir los dos números y devolver la diferencia, una función multiplicarNumeros, debe recibir dos números y devolver el resultado y la función dividirNumeros, debe recibir dos números y devolver el resultado.

   En el programa principal debe pedirse los dos números y seleccionar que operación dese realizar el usuario (suma, resta, multiplicación y división).

7. Programa que muestre un saludo por pantalla, utilizar un procedimiento.

8. Programa que mediante una función calcule el cuadrado de un número cualquiera y lo muestre en pantalla. Programa que muestre por pantalla los números del 1 al 100, utilizar recursividad.

9. Programa que realice la media de dos números, utilizar una función.

10. Programa que mediante un procedimiento calcule el cuadrado de un número cualquiera y lo muestre en pantalla.

11. Programa que mediante un procedimiento calcule el cuadrado de un número cualquiera y lo muestre en pantalla.

12. Programa que mediante un procedimiento calcule el cuadrado de un número cualquiera y lo muestre en pantalla.

13. Programa que pida un nombre, un apellido y un lugar y a raíz de ahí muestre en pantalla una pequeña historia, se debe usar un procedimiento.

14. Programa que muestre un menú con las opciones sumar, restar, multiplicar y dividir, el programa solicitará una opción y realizará la tarea elegida, se debe usar un procedimiento para mostrar el menú, pedir los datos en el main y después usar funciones para realizar los cálculos.

15. Programa que pida por pantalla un número del 1 al 10 y mediante un procedimiento muestre por pantalla el número escrito en letras.

FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO

PROGRAMA DE FORMACIÓN	IMPLEMENTAR ALGORITMOS EN SOLUCIONES SOFTWARE

COMPETENCIA	Desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo.	RESULTADOS DE APRENDIZAJE	220501093-03 Codificar el algoritmo de la solución del problema de acuerdo al lenguaje de programación seleccionado

NÚMERO DEL COMPONENTE FORMATIVO	003
NOMBRE DEL COMPONENTE FORMATIVO	Codificación de algoritmos
BREVE DESCRIPCIÓN	La realización de programas informáticos es una de las industrias de mayor crecimiento en los últimos años considerando que el producto final de un programa informático es un conjunto de instrucciones que el computador puede interpretar. En este componente aprenderá de las instrucciones para crear y realizar programas informáticos.
PALABRAS CLAVE	Codificación, JavaScript, Código fuente, interprete, compilación.

ÁREA OCUPACIONAL	No aplica
IDIOMA	Español

A. TABLA DE CONTENIDOS:

Introducción

1. **Conceptos básicos de programación.**
2. **Entornos de codificación e instalación.**
3. **Sintaxis del lenguaje JavaScript.**
   - 3.1. Funciones y operaciones matemáticas
   - 3.2. Tipos de datos, operadores y orden de evaluación
   - 3.3. Expresiones y comentarios
   - 3.4. Estructuras de selección
   - 3.5. Estructuras de repetición
   - 3.6. Estructuras de datos
   - 3.7. Estructuras de salto
   - 3.8. Métodos de ordenamiento y búsqueda
4. **Depuración y fallas de sintaxis.**
5. **Fallas de lógica.**
6. **Manejo de errores y excepciones.**

B. DESARROLLO DE CONTENIDOS:

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 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 desde este punto discutir todos los tipos de lenguajes existentes sino más bien aquellos que pueden ser interpretados por computadores, *Smartphone* 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:

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 va a servir en *Windows* o en *Mac OS.*	Generalmente 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.

Tabla 1. Lenguajes de programación

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 adicionalmente 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:

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

Tabla 2. Entornos de codificación

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 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; veamos 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.

Ingrese en la ruta del navegador a https://playcode.io/new/, escriba el código de la línea 1 a la 12 en la pestaña *script.js* como se muestra en la siguiente imagen:

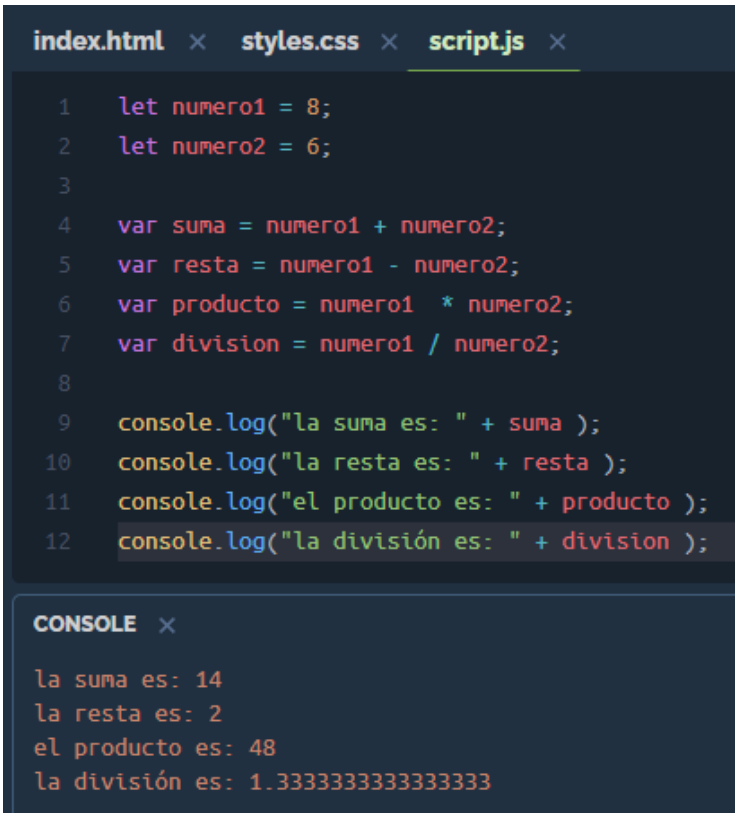


Ilustración 3. Operaciones aritméticas con JavaScritpt

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. Note 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 crea 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;
```

Note que 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 – JavaScript, 2021) veremos un ejemplo de su uso.

Creemos 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 mostrándonos su resultado:

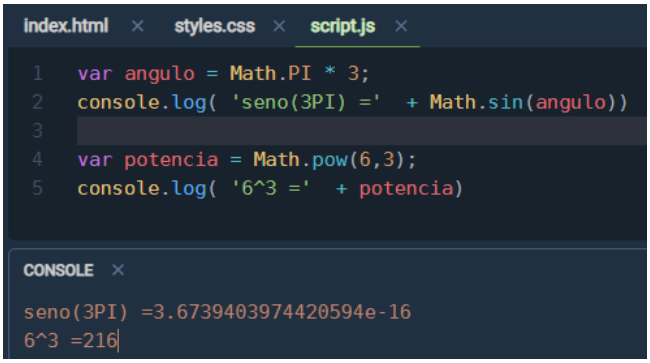


Ilustración 4. Uso del Objeto *Math*

Escriba el código fuente de las Ilustraciones 3 y 4, y obtenga 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 revise el listado de funciones de la clase Math.

PDF	Tabla 1. Métodos del objeto Math Fuente (Math - JavaScript | MDN. 2021)

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 *Javascrip*t, su 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 6 tipos de datos primitivos la referencia del lenguaje nos 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 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 a los tipos de datos y estructuras puede revisar el material complementario dispuesto:

	Tipos de datos y estructuras en JavaScript | MDN, 2021. 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 Lista 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, una lista de operadores más usados.

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

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

Tabla 2. Operadores de asignación básicos (Expresiones y operadores – JavaScript, 2021)

❖ **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.

Operador	Descripción	Ejemplos que devuelventrue
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

Tabla 3. Operadores lógicos. Fuente (Expresiones y operadores – JavaScript, 2021)

❖ **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.

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 en 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.

Tabla 4. Operadores aritméticos. Fuente (Expresiones y operadores – JavaScript, 2021)

❖ **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. Con los operadores lógicos, puede describir estas combinaciones de condiciones.

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.

Tabla 4. Operadores lógicos. Fuente (Expresiones y operadores – JavaScript, 2021)

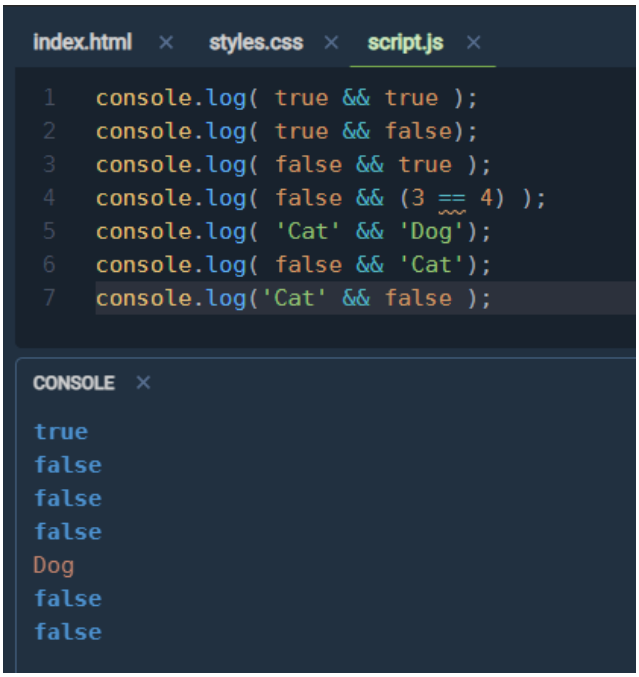


Ilustración 5. Ejemplos de operadores lógicos

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” – *JavaScript*, 2021.

www	Ampliemos los principales operadores de JavaScript la lista completa que se puede revisar en (Expresiones y operadores – *JavaScript*, 2021). 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 a los revisados anteriormente y que se debe 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. Relaciónales (=, <, >)
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 */:



Ilustración 6. Tipos de comentarios

Los comentarios son importantes porque le dan mantenibilidad al programa, es decir 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:

```
ALGORITMO Calificación del usuario;
VAR
    REAL calificacion
INICIO
   calificacion = 4;
   SI ( calificación >= 3)
            ESCRIBIR( “Aprobo” );
   SINO
            ESCRIBIR( “Reprobo” );
   FINSI
FIN
```

Ilustración 7: Seudocódigo calificaciones.

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



Ilustración 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**.

Recuerde: 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 notar:

1. Se usa la palabra reservada **if** y entre paréntesis la condición que la se quiere comparar.
2. Note que se usa el símbolo corchete abierto **{** se llama abrir corchete.
3. En la línea 7 se puede ver que mandamos la instrucción de que imprima el resultado de que aprobó.
4. Mire que en la línea 8 se cierra un bloque de código con el símbolo corchete cerrado **}**, con esto decimos que hasta este punto termina el bloque. A todo lo que esté entre estos corchetes le

llamaremos 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 vemos la palabra reservada **else**, que sirve para indicarle al intérprete de JavaScript que crearemos un bloque o contexto para cuando la condición evaluada por el **if** no se cumpla, y enseguida abrimos 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 cerramos con el carácter corchete cerrado **}** .

Escriba el programa y cambie 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 evalué e imprima si es mayor de edad:



Ilustración 9: Escribir en pantalla si es o no mayor de edad.

De acuerdo en la Ilustración 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 7 veces en pantalla la frase “Esto es un algoritmo”. Mostrar en pantalla la salida del ejercicio:



Ilustración 9. Imprimir 7 veces un mensaje.

Como se puede observar en la línea 4 de la Ilustración 9, se ha usado la palabra reservada ***for*** y dentro del paréntesis aparecen 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 ósea **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).

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

Recuerde: el carácter punto y coma se usa para decirle al computador o, interprete que ha finalizado una instrucción.
La estructura de repetición PARA (FOR) se usa cuando sabemos cuántas veces debemos 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:



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).

Ilustración 10. Sumar los números desde 1 hasta un número.

B. Estructura de repetición WHILE.

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

```
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
```

Ilustración 10: Seudocódigo algoritmo cálculo de factorial.

De acuerdo al anterior algoritmo el resultante para el cálculo de factorial:

```
// Cáculo del factorial
const numero = 5;
var contador = 1;
var factorial = 1;

while (contador <= numero){
    factorial = factorial * contador; // se puede abreviar factorial *= contador;
    contador ++;  // es equivalente a contador = contador + 1;
}
console.log(factorial);
```

```
CONSOLE
120
```

Ilustración 11: Seudocódigo algoritmo cálculo de factorial.

Es ideal que pueda trascribir el código usando https://playcode.io/new/, para lo cual 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 le funcione se propone el código que está entre comentarios (//) para practicar las dos formas de sintaxis.

Recuerde 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 *JavaScritp* 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 los siguiente:

```
let frutas = ["Manzana", "Banana", "Pera"]

console.log(frutas)
console.log(frutas.length)
```

```
CONSOLE
[
  "Manzana",
  "Banana",
  "Pera"
]
3
```

Ilustración 12: definición de un vector.

En la línea 1 se ve cómo se define un vector con identificador **frutas**, de 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 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***.



Ilustración 13: indexar de un vector.

Cree 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:



Ilustración 15: Crear la tabla del 5.

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

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 éste, y devuelve el *array* ordenado.
***shift*()**	Elimina el primer elemento de un *array*, y devuelve dicho elemento.

Tabla 5. Funciones comunes de un vector.

	Finalmente, desde el material complementario encuentra una lista completa o accediendo desde (*Array - JavaScript* | MDN, 2021). 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 Ilustración 16:

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

Matriz de 3x3

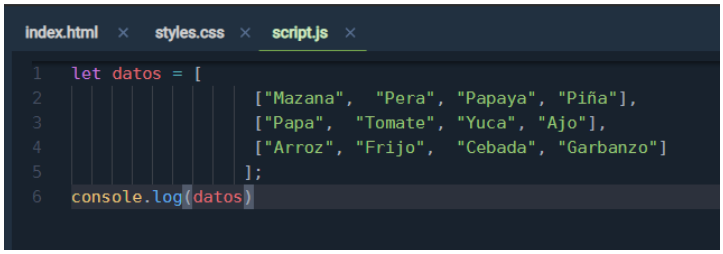


Ilustración 16: Crear la tabla del 5.

C. Registros

Hasta ahora hemos visto como 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:

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

Tabla 6. Registros.

Exceptuando por el encabezado de la tabla 6, cada fila representa un registro, y como se ve cada dato de un registro puede tener un tipo de dato diferente. Si se está interesado en codificar la Tabla 6, en una estructura con JavaScript el código sería como se tiene en la ilustración 17:

```
let registros =
[
  ["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]
];
```

Ilustración 17: Registros

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

```
let registros =[  ["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] ];

var total = 0;

for(let i =0; i < registros.length; i++){
 console.log("Registro # " + (i+1) );
  for(let j=0; j< registros[i].length; j++){
    console.log( "    " +registros[i][j] );
  }
}
```

```
CONSOLE
Registro # 1
    Juan
    juan@sindato.com
    19
    36.234
Registro # 2
    Luis
    luis@sindato.com
    18
    23.234
Registro # 3
    Andrea
    andrea@sindato.com
    22
    0
```

Ilustración 18: Recorrer registros

Como se puede observar en la Ilustración 18, se requiere de dos ciclos ***for*** anidados, note que cada ciclo tiene una variable de índice de nombre diferente **i** y **j** y la manera de indexar la matriz visto 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 nuestros 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, por favor mire los resultados.

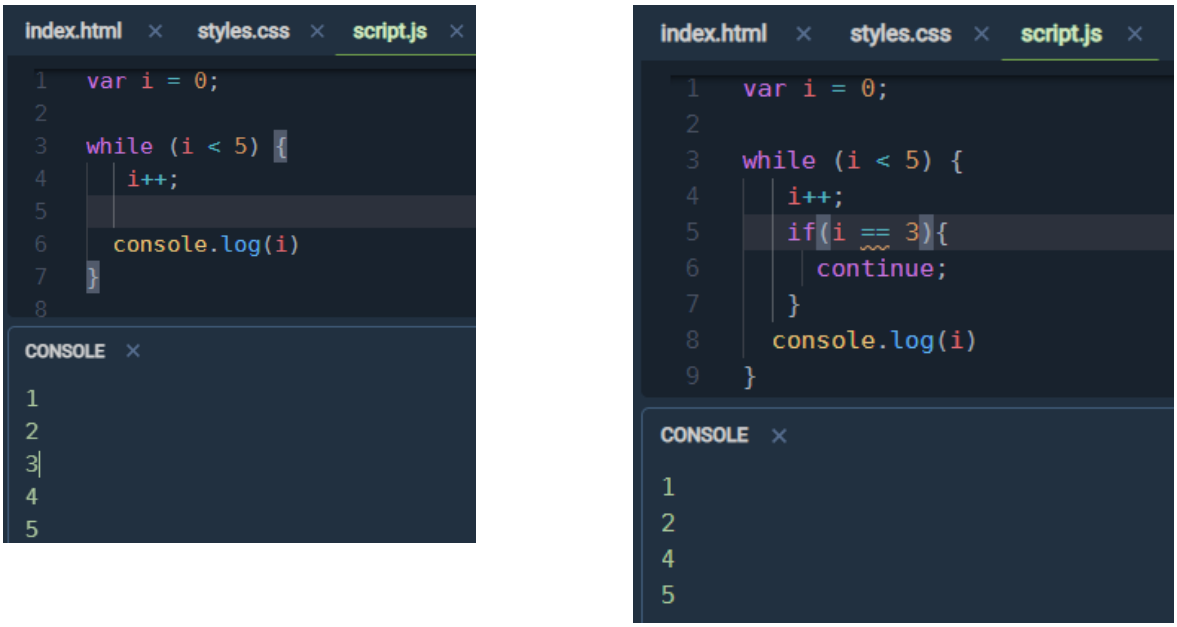


Ilustración 19: Sentencia ***continue***

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 Ilustración 20, prestando especial atención en la salida de consola.



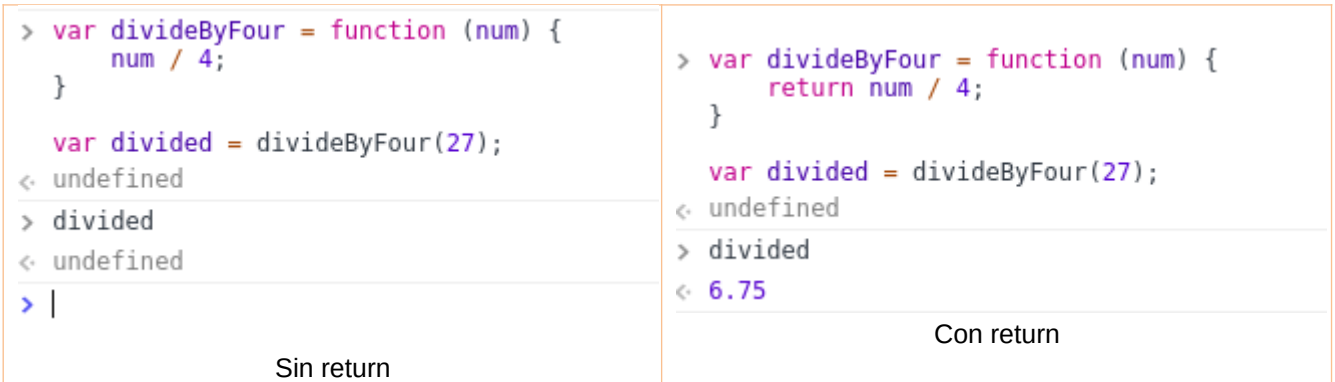
Ilustración 20: Sentencia **break**

Mire en la anterior ilustración, 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 sólo 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.



Sentencia *return*. Fuente: Vedia, Maria (2018). https://medium.com/@mvtercero85/cuando-usar-return-en-javascript-7b80e025eb7f

3.8. Métodos de ordenamiento y búsqueda

Existen varias técnicas de ordenamiento de vectores las cuales son importantes sobre todo para realizar búsquedas, al igual que en la vida real, es más fácil buscar en un lugar que esté ordenado que en uno que no lo este, como *JavaScript* está diseñado para pequeños algoritmos que resuelven pequeños problemas y convertirlos en servicios, el uso de esas técnicas no tiene mucha aplicabilidad, ya que 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 grandes, 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.

Ordenaremos un vector de acuerdo a los datos que tiene de mayor a menor valor, se deben tener en cuenta el arreglo se modifica al ordenarlo, devolviendo la misma matriz ordenada, pero no una nueva. Esto es importante, si queremos mantener inmutable el vector y obtener otro ordenado, lo que tendríamos que hacer 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 miremos un ejemplo:



Ilustración 21: Ordenamiento de números como cadenas de texto.

La Ilustración 21 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 son cadenas de texto.

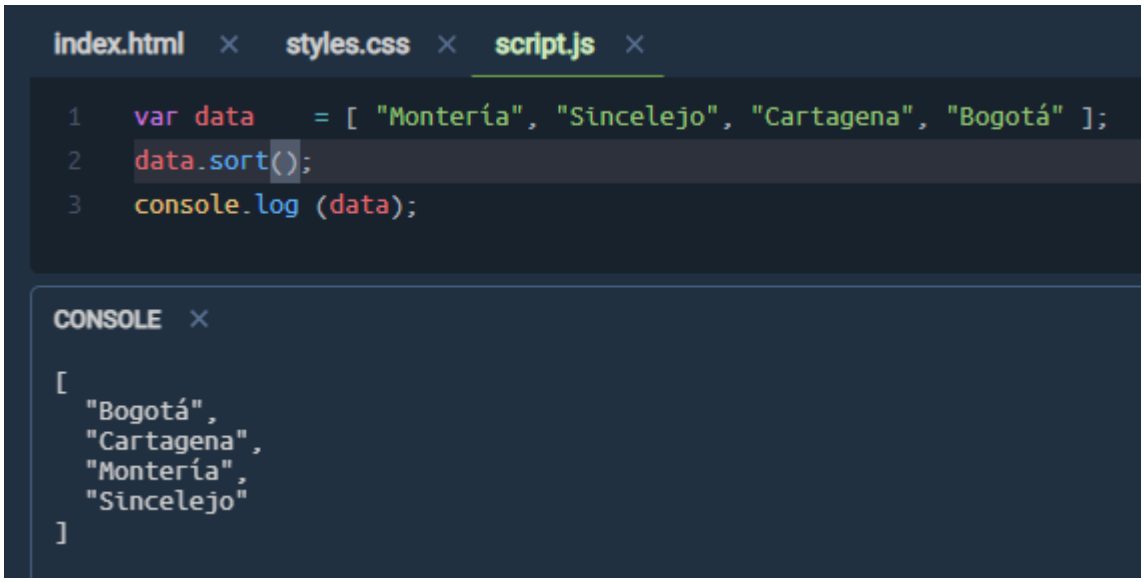


Ilustración 22: Ordenamiento de cadenas de texto.

El ejemplo de la ilustración 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 ilustración 23.

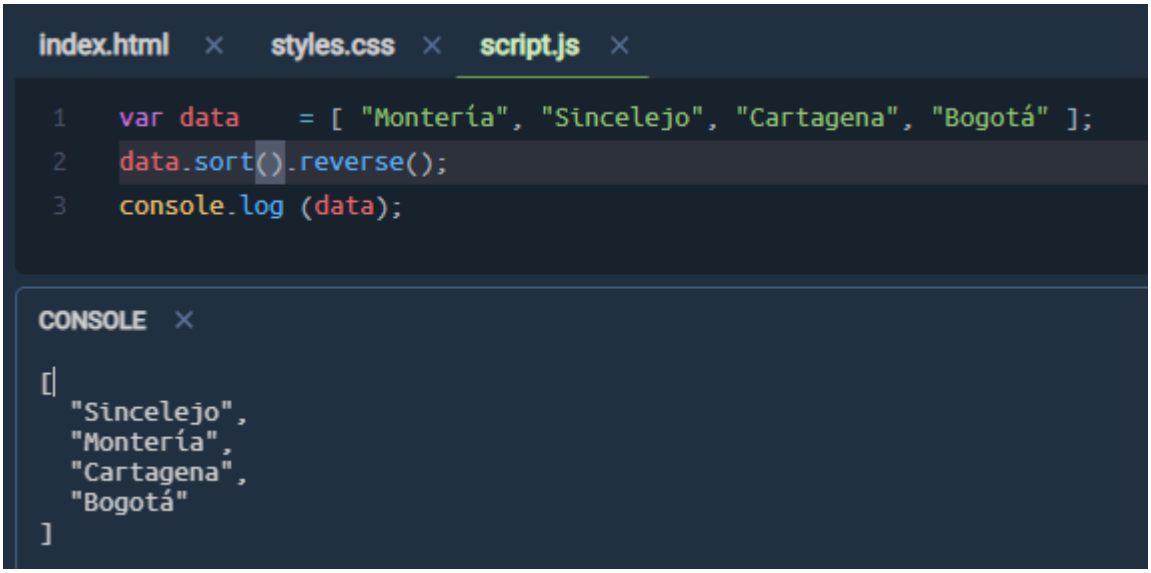


Ilustración 23: Invertir el orden de los elementos.

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 última pasa al primer puesto, es por esto que primero lo ordenamos en orden ascendente y luego lo invertimos, 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 *JavaScript*. 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 (Funciones - *JavaScript* | MDN, 2021).

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

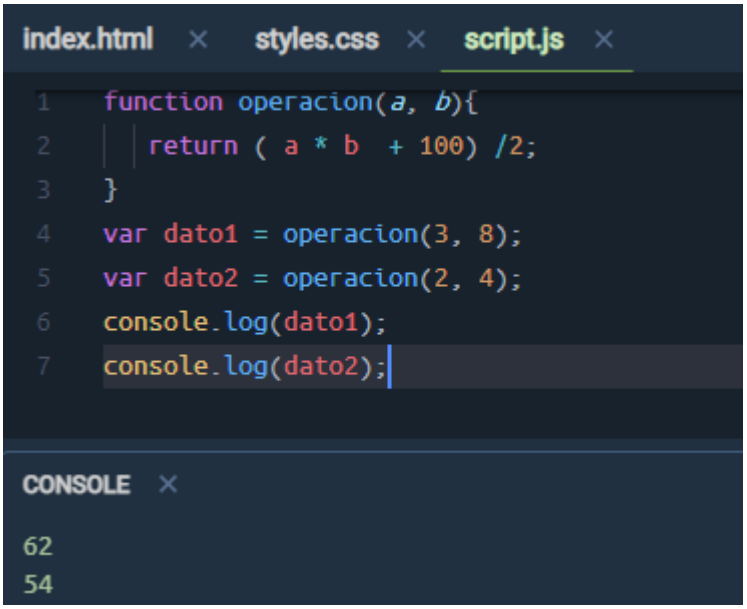


Ilustración 24: Definición de una función.

Como se puede observar en la línea 1 de la ilustración 24, se usa la palabra reservada ***funtion*** 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 necesitemos).

Para ordenar un vector con datos numéricos debemos pasarle 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 comunes 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.



Ilustración 25: Ordenamiento de números.

Como se puede ver en la Ilustración 25 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, como ordenar un arreglo número de manera inversa en la ilustración 26.

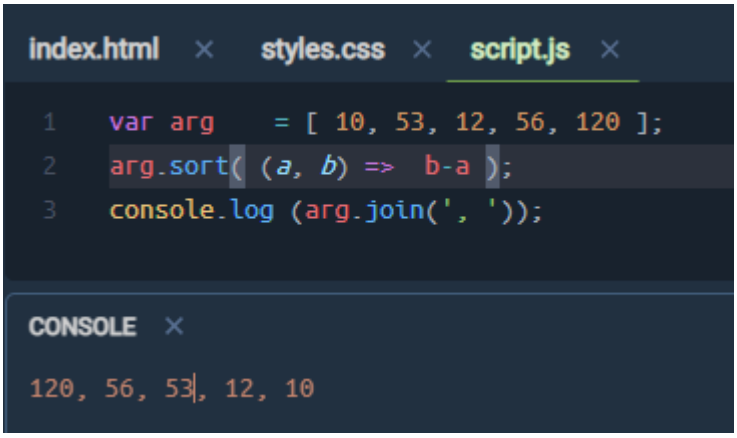


Ilustración 26: Ordenamiento inverso de números.

La ilustración 26 muestra como 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.

Suponga que tiene los registros de la tabla 7. Y desea ordenarlos por la edad de mayor edad a menor edad.

Usuario	Edad	Rol
Mariela	31	SAC

Eduardo	30	CEO
Andrés	34	Project Manager

Tabla 7. Registros a ordenar.

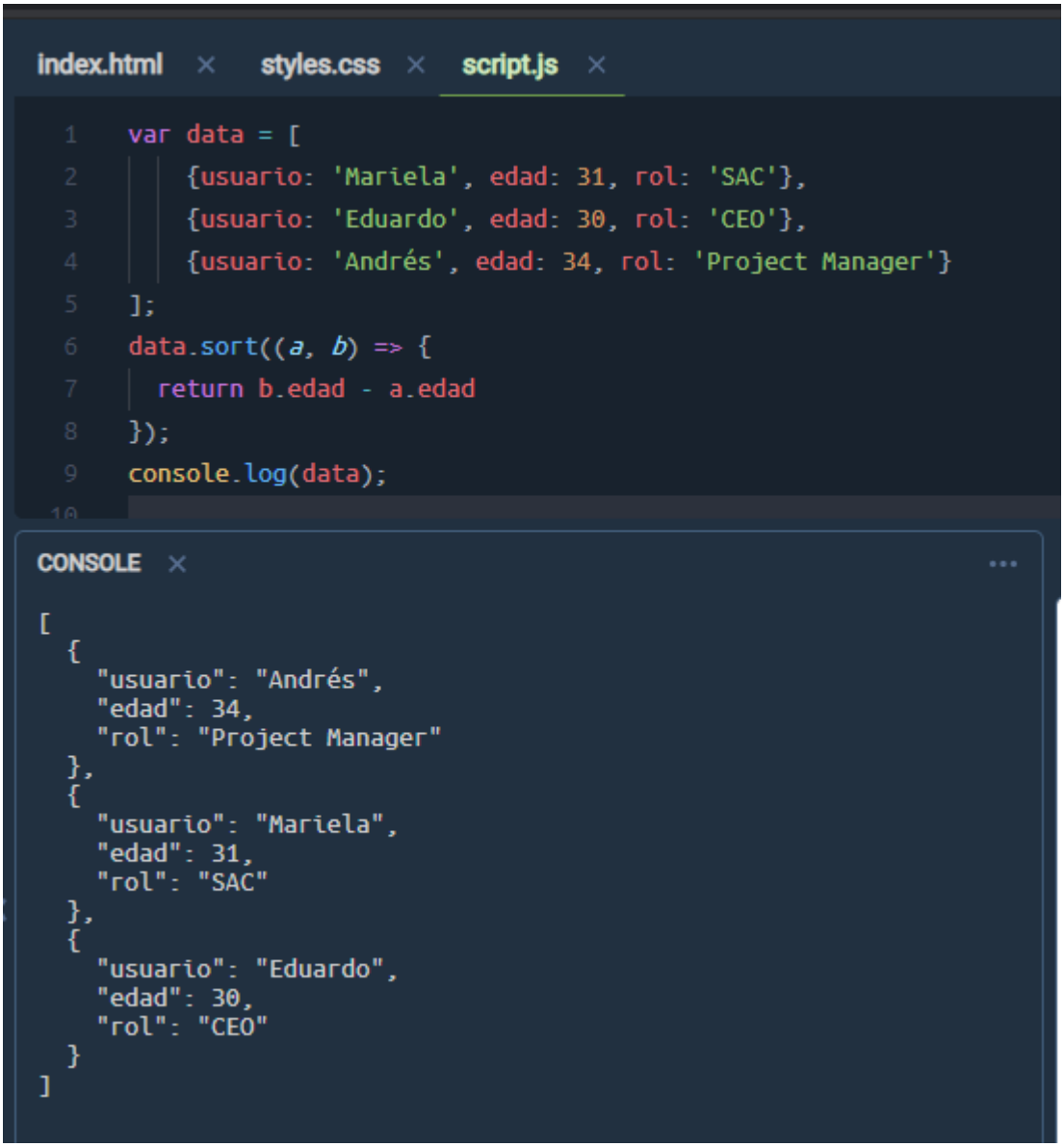


Ilustración 27: Ordenamiento de registros.

En esta oportunidad en la ilustración 27 se muestra (línea 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).

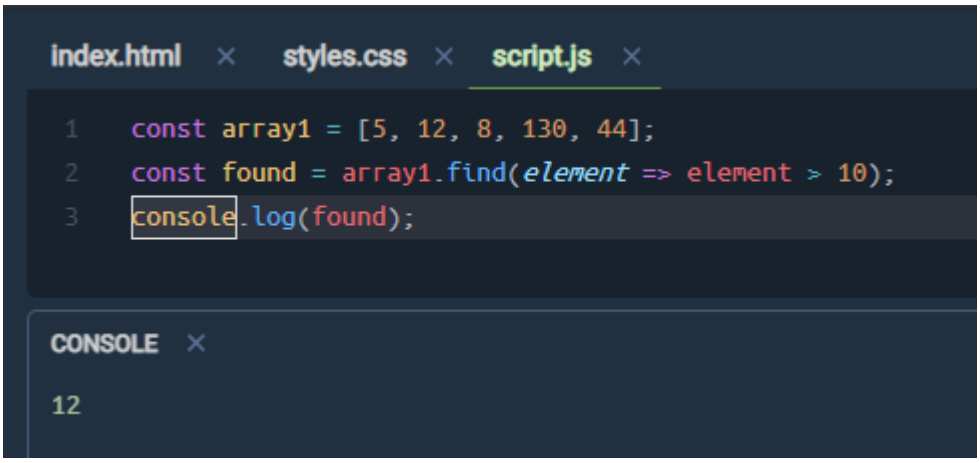


Ilustración 28: Buscar un elemento.

Como se puede ver en la ilustración 28, 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 Ilustración 28, se retorne 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.

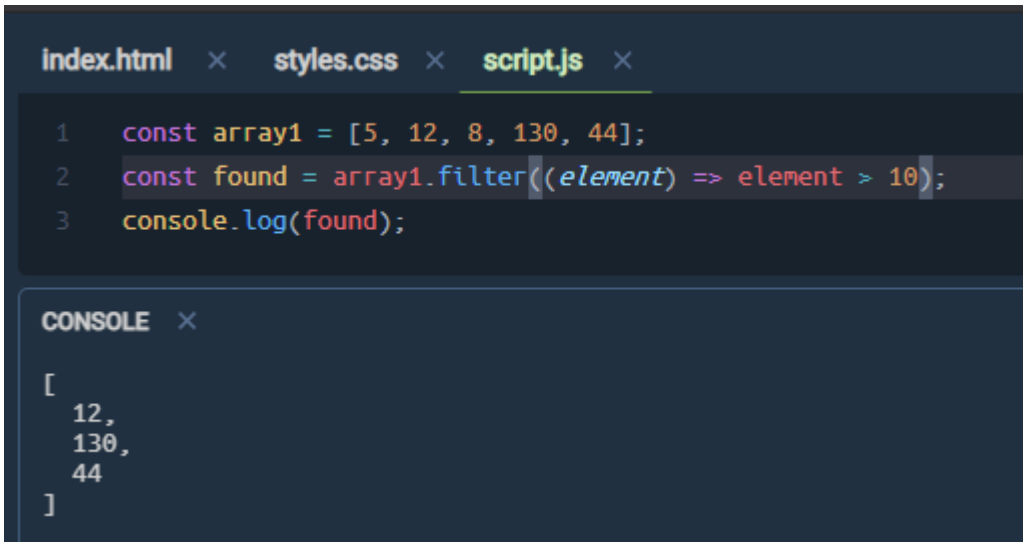


Ilustración 29: Buscar varios elementos.

Note en la consola que 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 de 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, dejándonos la consola para realizar estas depuraciones.

Hasta ahora hemos usado la consola para mostrar datos de depuración cuando usamos ***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.

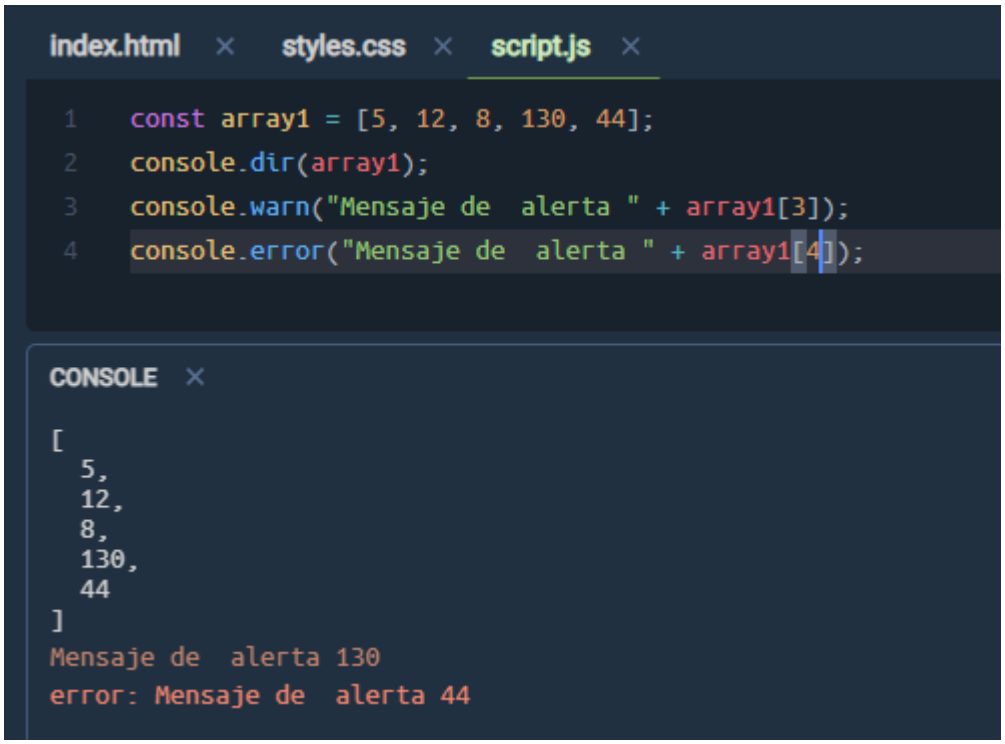


Ilustración 30: Mensaje de depuración.

www	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 dictan 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 tenemos que ir a la sección del archivo con el código fuente a depurar. Podemos navegar a través de él e identificamos la porción de código en la que se sospecha ocurre un error en la lógica.

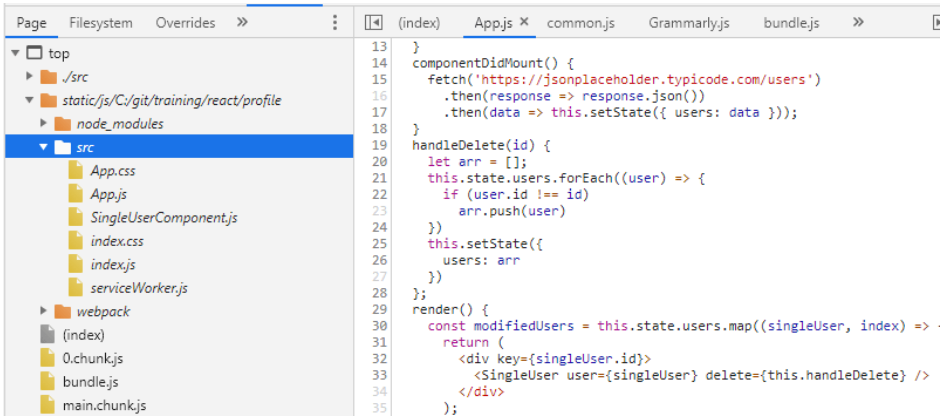


Ilustración 31: Seleccionar fuente.

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 a por cada línea de código. Se debe hacer clic en este número de línea. Ahora esta línea actuará como un punto de interrupción y la ejecución se detendrá una vez que el control llegue a esta línea.

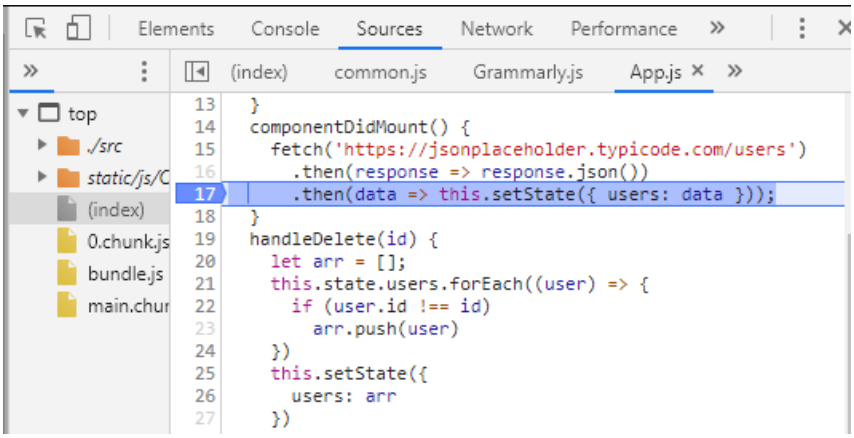


Ilustración 32: Marcar un break point.

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 Ilustración 33. Así la ejecución avanza línea por línea. Aquí puede inspeccionar cada línea de código junto con los valores contenidos en las variables.

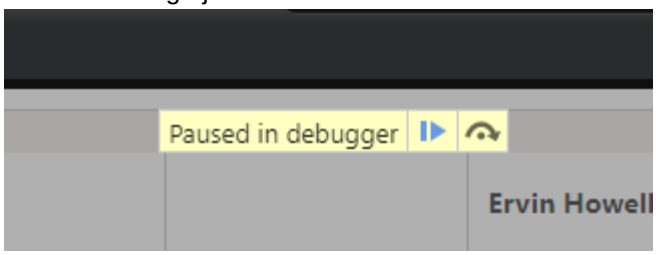


Ilustración 33: Ejecución línea a línea.

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 puede 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.

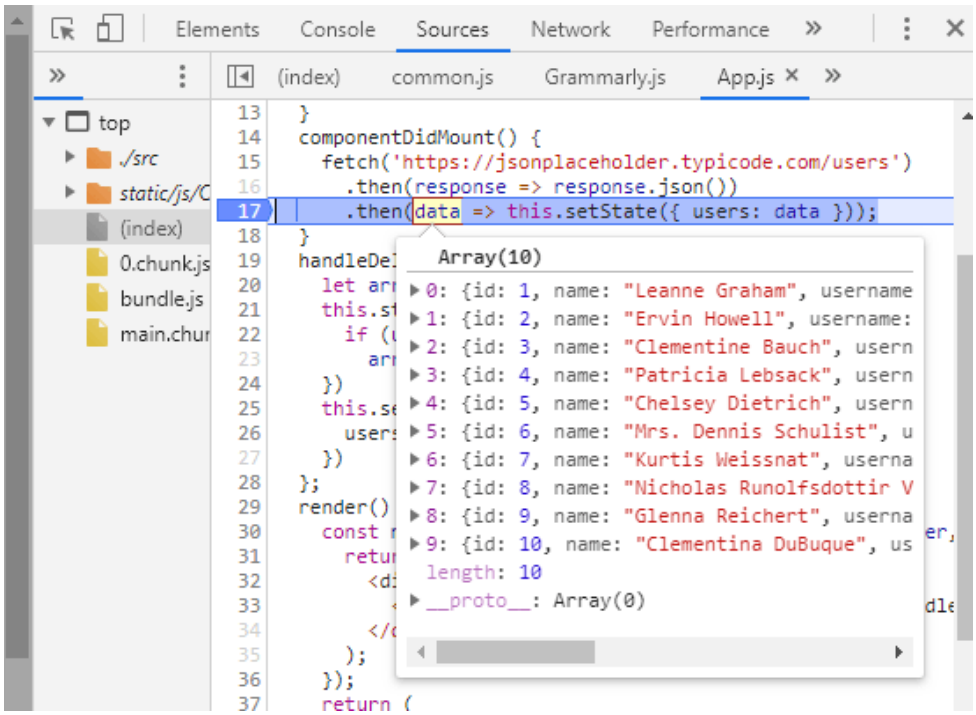


Ilustración 34: Valores de las variables.

www	Puede ver un ejemplo en el siguiente enlace: https://youtu.be/I388w3wDkjc (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, de esta manera 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 envié un “Error” como se muestra en la siguiente imagen.

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

Ilustración 35: lazar una excepción.

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*.**
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.



Ilustración 36: Bloque de captura de excepción.

En la Ilustración 36 se muestra un algoritmo que divide dos números, y como sabemos en este proceso se debe validar que no exista una división por cero 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 cero 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. De esta forma 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.

C. ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)

N/A

D. MATERIAL COMPLEMENTARIO:

Referencia APA del Material	Tipo de material (Video, capítulo de libro, artículo, otro)	Enlace del Recurso o Archivo del documento o material
Framebits. (19 de enero. 2020). Descargar e instalar	Video	https://youtu.be/

Node Js en Windows 10. Youtube. https://youtu.be/v0x1Ku5Tgac		v0x1Ku5Tgac
Code Compadre. (30 de junio. 2020). How to Download and Install Visual Studio Code for Windows 10. Youtube. https://youtu.be/KpzkPlh_HsU	Video	https://youtu.be/KpzkPlh_HsU
VideoMarketingViral. (11 de enero de 2020). Descargar Google Chrome Para PC 2021 (WINDOWS 7/8/10). Youtube. https://youtu.be/V-M2rMBhgKc	Video	https://youtu.be/V-M2rMBhgKc
Math - JavaScript | MDN. (2021, March 28). Retrieved from https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math	Documentación técnica	https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math
Expresiones y operadores - JavaScript | MDN. (2021, March 28). Retrieved from https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators	Documentación técnica	https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators
Array - JavaScript | MDN. (2021, Mar. 28). Retrieved from https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array#	Documentación técnica	https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array#
Funciones - JavaScript | MDN. (2021, Apr. 04). Retrieved from https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions	Documentación técnica	https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions
Array.prototype.find() - JavaScript | MDN. (2021, Apr. 04). Retrieved from https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/find	Documentación técnica	https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/find
Choque, R. C. (2020, May 10). Como usar la consola de google chrome para Javascript. Youtube. Retrieved from https://www.youtube.com/watch?v=Hf3n-p3VYx4	Video	https://youtu.be/Hf3n-p3VYx4
Autodidacta, F. (2017, May 11). Depurar o hacer debug a JavaScript con google Chrome. Youtube. Retrieved from https://www.youtube.com/watch?v=I388w3wDkjc&t=192s	Video	https://youtu.be/I388w3wDkjc

E. GLOSARIO:

TÉRMINO	SIGNIFICADO
Código fuente	Está compuesto por la línea de texto de un programa en determinado lenguaje de programación.
Código máquina	Es el código de instrucciones que directamente interpreta el circuito microprogramable microprocesador de una computadora.
Compilador	Es un programa de computador encargado de traducir el código fuente de otro programa escrito en un lenguaje de programación a otro lenguaje de programación.

F. REFERENCIAS BIBLIOGRÁFICAS:

Array - JavaScript | MDN. (2021, Mar. 28). Retrieved from https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array#

Array.prototype.find() - JavaScript | MDN. (2021, Apr. 04). https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Array/find

Autodidacta, F. (2017, May 11). Depurar o hacer debug a javascript con google chrome. Youtube. https://www.youtube.com/watch?v=I388w3wDkjc&t=192s

Choque, R. C. (2020, May 10). Como usar la consola de google chrome para Javascript. Youtube. https://www.youtube.com/watch?v=Hf3n-p3VYx4

Expresiones y operadores - JavaScript | MDN. (2021, March 28). https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Expressions_and_Operators

Funciones - JavaScript | MDN. (2021, Apr. 04). https://developer.mozilla.org/es/docs/Web/JavaScript/Guide/Functions

López Juan Carlos, 2009, Algoritmos y programación (guía para docentes). https://eduteka.icesi.edu.co/

Math - JavaScript | MDN. (2021, Apr. 13). https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math

Math - JavaScript | MDN. (2021, March 28). https://developer.mozilla.org/es/docs/Web/JavaScript/Reference/Global_Objects/Math

Resnick, Mitchel (2007a): Sembrando las semillas para una sociedad más creativa. Laboratorio de medios de MIT, Massachussets [Consulta en línea: Eduteka, Marzo 16, 2009, http://www.eduteka.org/ScratchResnickCreatividad.php]

Rojas A., Vicente & Ñacato C., José (1980): Técnica de flujogramas I; Editora Andina, Quito, séptima edición.

Tipos de datos y estructuras en JavaScript - JavaScript | MDN. (2021, Apr. 11). https://developer.mozilla.org/es/docs/Web/

Vedia, M. (2018). Cuando usar Return en Javascript - María Vedia - Medium. Medium. https://medium.com/@mvtercero85/cuando-usar-return-en-javascript-7b80e025eb7f

G. CONTROL DEL DOCUMENTO

	Nombre	Cargo	Dependencia *(Para el SENA indicar Regional y Centro de Formación)*	Fecha
Autor (es)	Henry Eduardo		Regional Cauca, Centro de	Abril de 2021

	Bastidas Paruma	Experto temático	teleinformática y producción industrial	
	Peter Emerson Pinchao Solis	Experto temático	Regional Cauca, Centro de teleinformática y producción industrial	Abril de 2021
	Oscar Absalón Guevara	Diseñador Instruccional	Regional Bogotá, Centro de Gestión Industrial.	Abril de 2021
	Rafael Neftalí Lizcano Reyes.	Asesor Pedagógico	Regional Santander. Centro Industrial del Diseño y la Manufactura.	Abril 2021

H. CONTROL DE CAMBIOS

(Diligenciar únicamente si realiza ajustes a la Unidad Temática)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO

PROGRAMA DE FORMACIÓN	SOLUCIÓN DE PROBLEMAS MEDIANTE LA UTILIZACIÓN DE ALGORITMOS

COMPETENCIA	220501096 Desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo.	RESULTADOS DE APRENDIZAJE	220501093-02 Diseñar algoritmos de acuerdo con el problema a solucionar.

NÚMERO DEL COMPONENTE FORMATIVO	2
NOMBRE DEL COMPONENTE FORMATIVO	Análisis y solución de problemas aplicando algoritmos
BREVE DESCRIPCIÓN	El avance tecnológico lleva implícita la frontera del conocimiento, esto a su vez genera nuevos conceptos, terminologías, notaciones, nomenclaturas y su representación a través diagramas, planos o artefactos que terminan convirtiéndose en un lenguaje natural de la tecnología en cuestión. En este componente se presentan los artefactos diagramas y nomenclaturas con los que se describen los algoritmos.
PALABRAS CLAVE	Pseudocódigo, Diagrama de flujo, traza, variables, constantes, contadores, acumuladores, condiciones.

AREA OCUPACIONAL	No aplica
IDIOMA	Español

A. TABLA DE CONTENIDOS:

Introducción

1. Metodologías de algoritmos
1.1. Análisis
1.2. Diseño
1.2.1. Notación de diseño: seudocódigo
1.2.2. Pruebas de escritorio o trazas.

2. Diagramas de flujo
2.1. Elaborar diagramas de flujo
2.2. Herramientas digitales para creación y prueba de algoritmos.

3. **Uso de identificadores y palabras reservadas.**

4. Operadores y jerarquía en los operadores

5. Estructuras de control secuencial
5.1. Ejemplo estructura de control secuencial
5.2. Estructura condicional
5.3. Estructuras de iteración o de repetitivas
5.3.1. Estructura PARA (FOR)
5.3.2. Estructura MIENTRAS (WHILE)

6. Estructura de datos (arreglos, matrices)
6.1. Vectores
6.2. Matrices

7. Programación modular

B. DESARROLLO DE CONTENIDOS:

Introducción

A lo largo de las últimas décadas, la industria del software ha podido recoger experiencias, que poco a poco se han convertido en un conjunto de técnicas y buenas prácticas en el campo de la algoritmia. A continuación, se explorará un enfoque metodológico para el diseño y especificación de algoritmos con notaciones que son ampliamente aceptadas y que posteriormente se pueden implementar en cualquier lenguaje de programación.
Este enfoque será una guía o marco de referencia a la hora de abordar cualquier problema en el que se pretenda dar solución a través de procesos computacionales.

1. Metodologías de algoritmos

De manera implícita en las distintas áreas productivas de la sociedad, se usan algoritmos y un par de ejemplos pueden ser que, en la construcción de una casa, el ingeniero o el arquitecto, analiza el terreno, diseña unos planos, realiza maquetas y así continúan siguiendo una secuencia lógica de procesos, para la resolución del problema; o un abogado antes de defender a un cliente analiza las pruebas, elabora un plan, practica sus argumentos y sigue un plan para lograr la defensa de su cliente. Se puede concluir que una solución a la medida de un problema requiere una metodología que lleve a la solución final.

En el proceso de desarrollo de programas informáticos, aparte de conocer esta metodología se necesita conocer las funciones que puede realizar un computador y la representación de las mismas.

En general existen 3 fases secuenciales para la elaboración de un algoritmo representadas en el siguiente diagrama y de las cuales se abordarán las dos primeras:

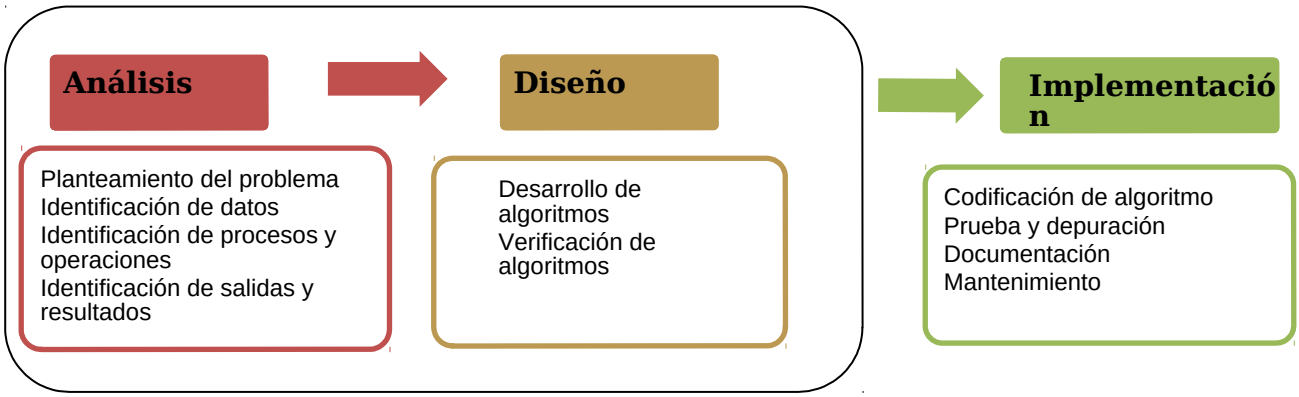


Ilustración 1: metodología de creación de algoritmos

1.1. Análisis.

Como se percibió en componentes anteriores la fase de análisis trata de responder a la pregunta: ¿qué es lo que se debe hacer?, y esto hace necesario entender el problema. Por ello es preciso relacionar los datos de entrada, salida o resultados con los procesos a ejecutar con esta información y los datos auxiliares que se puedan requerir en las tareas.

Hasta este punto parece que esas son las actividades propias del análisis, sin embargo, el éxito de un algoritmo consiste en reducir a cero la brecha de lo que el cliente necesita y los resultados que el algoritmo entrega. Para lograrlo en su totalidad, es importante indagar otros aspectos que corresponden a la naturaleza del problema, las siguientes preguntas modeladoras son de referencia casi obligante a tener en cuenta:

¿Qué es?, ¿Quienes intervienen? ¿Cómo interviene o se comporta? ¿Cuándo?, ¿Cómo lo hace?, ¿Por qué? y ¿Para qué?.

1.2. Diseño

En esta etapa, se responde básicamente a la pregunta **¿cómo se va a hacer?**, refiriéndose a cómo el sistema va a realizar la tarea solicitada. Lo importante en este punto, es que debe quedar bien definido el problema, sin dar espacio a más de una interpretación de la acción o acciones que debe hacer el sistema.

En el proceso de desarrollo del algoritmo las siguientes actividades son obligatorias, sobre todo si se está empezado en el diseño de soluciones basadas en algoritmos y no se cuenta con mucha experiencia:

- Definir las partes del algoritmo.
- Pasos para el desarrollo del algoritmo.
- La representación del algoritmo (notación, diagrama o seudocódigo).

Para la verificación del algoritmo, se aplica una técnica que denominamos pruebas de escritorio o trazas, que trata de la validación y verificación del algoritmo mediante la ejecución de las sentencias u operaciones que lo componen (proceso), y a si determinar sus resultados de salida a partir de una base determinada de elementos en la entrada del proceso.

Como el resultado del proceso de diseño es un algoritmo, es obligatorio saber representarlo de alguna manera, se empleará inicialmente con la notación de pseudocódigo para hacer este tipo de representaciones:

1.2.1. Notación de diseño: seudocódigo

Para expresar el contenido de un programa de computadora, se necesita de un lenguaje de programación que las computadoras pueden interpretar. Sin embargo, se puede expresar ese mismo contenido en el lenguaje natural lengua materna (español), empleando expresiones sencillas que lo simplifican y abrevian. Cuando se emplea esta técnica se dice que está escrito en pseudocódigo.

Ejemplo: el cliente pide que se desarrolle un programa de computadora que le solicite al usuario un número, luego el sistema debe mostrar en pantalla el número digitado.

A continuación, se aplica la metodología vista:

- **Análisis:** el dato de entrada debe ser un número ingresado por el usuario, pero antes de eso el sistema debe solicitarle al usuario que lo ingrese (digite) al programa, si no, de qué otra manera el usuario sabría que el programa está esperando el ingreso de ese dato. Por lo tanto, una posible operación inicial dentro del algoritmo sería escribir en pantalla que "*por favor se ingrese un número"*.

La segunda operación es la de recibir ese número desde el teclado, para ingresarlo al programa, pero esto implica que debemos definir un dato intermedio donde se guardará ese valor y el cual identificará el valor ingresado (a este dato se denomina variable porque puede tener cualquier valor entero según digite el usuario). Ya ocurre el primer problema: para guardar el dato (variable) se debe definir dónde se almacenará o como podría referirse al dato ingresado dentro del programa, por lo tanto, antes de empezar el algoritmo se debe indicar al computador que reserve un espacio para guardar un número entero.

Una vez definida la variable es posible referirse a ella, leer desde el teclado el número entero y almacenarlo en dicha variable.

Por último, el sistema debe imprimir el contenido de la variable (dato de salida), con lo cual se puede dar fin al programa.

- **Diseño:** en anterior análisis se contemplaron los datos de entrada, datos auxiliares (variable) operaciones (impresión en pantalla, lectura de datos e impresión en pantalla de resultados). Pero son muchas las palabras usadas en el análisis por lo tanto lo abreviamos con el siguiente código que posteriormente explicaremos:

```
ALGORITMO Mostrar número digitado;
VAR
   ENTERO entrada;
INICIO
   ESCRIBIR ("Por favor ingrese un número");
   LEER (entrada);
   ESCRIBIR (entrada);
FIN
```

Ilustración 2: seudocódigo imprimir un número

En **ALGORITMO** se indica el nombre del algoritmo, y después se declaran las variables que serán usadas en él (con **VAR**) indicando su tipo de dato (un valor de tipo **ENTERO**, para este ejemplo).

Entre **INICIO** y **FIN** se escriben las diferentes acciones que realizará el algoritmo y usamos **ESCRIBIR**() para indicar con esta palabra que se mostrará un mensaje por pantalla (el texto que se va a mostrar 'tal cual' debe ir entre paréntesis dobles), y **LEER**() para recibir el dato desde el teclado.

Se debe tener en cuenta que cada línea termina en punto y coma (excepto **INICIO** y **FIN**), el punto y coma indica que lo que se ejecuta es una sentencia que hace una única operación, mientras que lo que está entre **INICIO** y **FIN** agrupa varias sentencias que se ejecutan en un orden determinado.

1.2.2. Pruebas de escritorio o trazas.

Las pruebas de escritorio son las simulaciones de ejecución de un algoritmo que permite determinar la validez o efectividad del mismo. Se trata de escribir en una tabla con tantas columnas como variables tiene el algoritmo y seguir las instrucciones poniendo los valores correspondientes.

Ejemplo: Escribir un algoritmo que calcule el cuadrado y el cubo de un número introducido por teclado y mostrar los resultados para el cubo y el cuadrado.

- **Análisis:** nuevamente el dato de entrada debe ser un número ingresado por el usuario, y al igual que en el ejemplo anterior se debe solicitar al usuario que lo digite.

Antes de realizar las operaciones de elevar al cuadrado y al cubo, se necesita definir las dos variables que recibirán estos datos (el cuadrado, y el cubo), por lo tanto, el algoritmo debe definirlas antes de realizar las operaciones y asignarles a ellas los resultados.
Una vez realizadas las operaciones el algoritmo debe mostrar en pantalla el contenido de estas variables, con un mensaje texto que indique si es el resultado del cuadrado o del cubo.

- **Diseño:** La siguiente ilustración tiene la representación del resultado al análisis anteriormente.

```
ALGORITMO Calcular el cuadrado y el cubo;
VAR
    ENTERO entrada;
    ENTERO cuadrado;
    ENTERO cubo;
INICIO
    ESCRIBIR("Por favor ingrese un número");
    LEER( entrada );
    cuadrado <= entrada * entrada;
    cubo <= entrada * entrada * entrada;
    ESCRIBIR( "El Cuadrado " cuadrado );
    ESCRIBIR( "El Cubo " cubo );
FIN
```

Ilustración 3: Metodología de creación de algoritmos

Las pruebas de escritorio permiten verificar que el pseudocódigo realice lo que se espera al resolver el problema. Sin embargo, por ahora es importante que a todos los pseudocódigos se les aplique al menos una de prueba de escritorio o las que se consideren necesarias.

La mayoría de las pruebas de escritorio se representan en una tabla, donde en cada columna se anotan las variables y condiciones si las hay, en el orden en que aparecen en el algoritmo, para entenderlo se tiene que recorrer línea a línea el pseudocódigo y así identificar tanto las variables como condiciones e ir anotándolas en la tabla de la prueba de escritorio.

Es importante que en la tabla no aparezcan variables o condiciones repetidas y en la última columna se pueden anotar los datos de salida para entonces también anotar “Salida”.

Ejecutando línea a línea el algoritmo de la Ilustración 3 y se va llenando la tabla como muestra la siguiente figura:

ESCRIBIR("Por favor ingrese un número");

Número	Cuadrado	Cubo	Salida
			Por favor ingrese un número

LEER(entrada);

Número	Cuadrado	Cubo	Salida
3			Por favor ingrese un número 3

cuadrado <= entrada * entrada;

Número	Cuadrado	Cubo	Salida
3	9		Por favor ingrese un número 3

cubo <= entrada * entrada * entrada;

Número	Cuadrado	Cubo	Salida
3	9	27	Por favor ingrese un número 3

ESCRIBIR(cuadrado);

Número	Cuadrado	Cubo	Salida
3	9	27	Por favor ingrese un número 3 El Cuadrado 9

ESCRIBIR("Por favor ingrese un número");

Número	Cuadrado	Cubo	Salida
3	9	27	Por favor ingrese un número 3 El Cuadrado 9 El Cubo 27

Ilustración 4: Tablas de traza del algoritmo.

También es posible realizar pruebas de escritorio a otras representaciones de algoritmos, como los diagramas de flujo expuestos a continuación y que se representan con algoritmos recomendando que a cada ejemplo presentado se le realice la prueba de escritorio correspondiente.

2. Diagramas de flujo

Una forma común de presentar los algoritmos es con diagramas de flujo, que consiste en una representación por medio de un gráfico. Mientras en seudocódigo las sentencias se representaban con preposiciones en diagramas de flujo la preposición es reemplazada por un gráfico.

La representación del algoritmo de la Ilustración 2 sería:

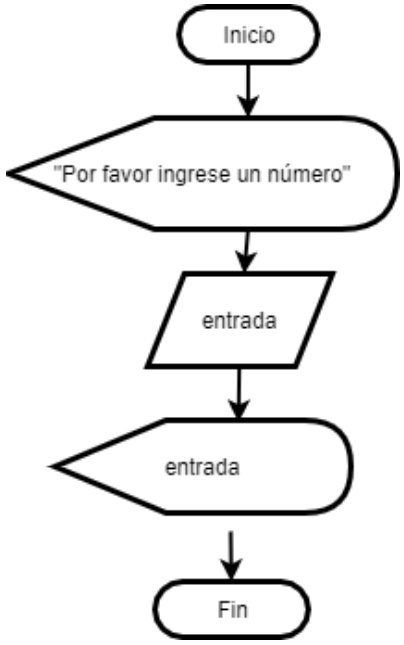
	Inicio del algoritmo Muestra en pantalla el mensaje que ingrese un número Leemos del teclado un número Muestra en pantalla el valor del número digitado Final del algoritmo

Ilustración 5: Diagrama de flujo imprimir un entero

Es ideal revisar el siguiente vídeo sobre la representación gráfica de un algoritmo a través del diagrama de flujo:

Motion Graphics
DI_CF02_2 Ventajas de los diagramas de flujo

Vídeo. Ventajas del diagrama de flujo

La representación del algoritmo de la Ilustración 3, (algoritmo que calcule el cuadrado y el cubo de un número introducido por teclado y mostrar los resultados el cubo y el cuadrado).

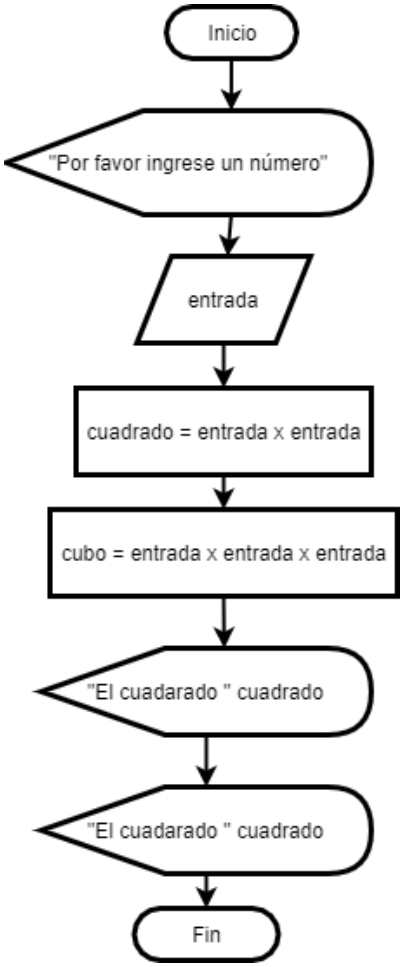
	Inicio del algoritmo Muestra en pantalla el mensaje que ingrese un número Leemos del teclado un número Calcula el cuadrado del número digitado Calcula el cubo del número digitado Muestra el cuadrado del número digitado Muestra el cubo del número digitado Final del algoritmo

Ilustración 6: Diagrama de flujo imprimir el cuadrado y el cubo de un entero

La Organización Internacional para la Estandarización (ISO, de su sigla en inglés) y el Instituto Nacional Americano de Estandarización (ANSI, de su sigla en inglés), estandarizaron los símbolos que han sido mayor mente aceptados en 1985. Con el fin de evitar la utilización de símbolos diferentes para representar procesos iguales (López, 2009).

Los siguientes son los principales símbolos para crear diagramas de flujo:

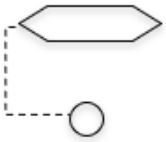
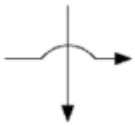
	**Inicio/Final** Se utiliza para indicar el inicio y el final de un diagrama; del Inicio sólo puede salir una línea de flujo y al Final sólo debe llegar una línea (López, 2009).		**Decisión** Indica la comparación de dos datos y dependiendo del resultado lógico (falso o verdadero) se toma la decisión de seguir un camino del diagrama u otro (López, 2009).
	**Entrada General** Entrada/Salida de datos en General (en esta guía, solo la usaremos para la Entrada) (López, 2009).		**Iteración** Indica que una instrucción o grupo de instrucciones deben ejecutarse varias veces (López, 2009).
	**Entrada por teclado** Instrucción de entrada de datos por teclado. Indica que el computador debe esperar a que el usuario teclee un dato que se guardará en una variable o constante (López, 2009).		**Salida Impresa** Indica la presentación de uno o varios resultados en forma impresa (López, 2009).
	**Llamada a subrutina** Indica la llamada a una subrutina o procedimiento determinado (López, 2009).		**Salida en Pantalla** Instrucción de presentación de mensajes o resultados en pantalla (López, 2009).
	**Acción/Proceso General** Indica una acción o instrucción general que debe realizar el computador (cambios de valores de variables, asignaciones, operaciones aritméticas, etc) (López, 2009).		**Conector** Indica el enlace de dos partes de un diagrama dentro de la misma página (López, 2009).
	**Flujo** Indica el seguimiento lógico del diagrama. También indica el sentido de ejecución de las operaciones (López, 2009).		**Conector** Indica el enlace de dos partes de un diagrama en páginas diferentes (López, 2009)

Tabla 1: Símbolos de los diagramas de flujo

2.1. Elaborar diagramas de flujo

Cuando el algoritmo se desea representar como diagrama de flujo, se deben tener en cuenta algunos principios para su elaboración (Rojas & Ñacato, 1980):

Infografías
DI_CF02_2,1 Diagramas de flujo

Infografía. Elaborar diagramas de flujo

2.2. Herramientas digitales para creación y prueba de algoritmos.

Para aprender a reconocer los elementos que componen los diagramas de flujo se puede hacer uso de la herramienta denominada Draw.io a través del enlace https://app.diagrams.net/ con la que puede usar una cuenta de correo de Google, o misena.edu.co para acceder y crear un nuevo diagrama.
Es ideal ingresar y seleccionar los diagramas que se muestran en la siguiente Ilustración.

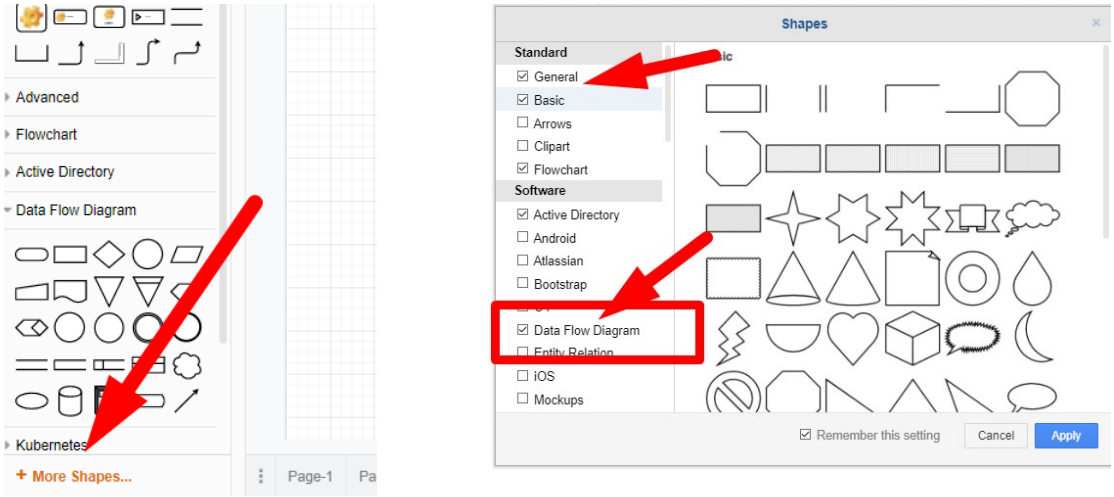


Ilustración 19: Seleccionar formas https://app.diagrams.net/

En caso de contar con intermitencia de Internet se puede realizar el ejercicio propuesto a través del programa de libre distribución disponible en http://dia-installer.de/ o puede acceder a la versión de escritorio de Draw.io disponible en https://app.diagrams.net/.

Insituto de Informatica Uach. (2020, Sept. 08). Como descargar draw io 2 para dibujar Diagramas de Flujo. Youtube. https://www.youtube.com/watch?v=encz3h8TLWA

El programa de mayor uso para el diseño y depuración de algoritmos en seudocodigo es *PSeInt* disponible en http://pseint.sourceforge.net/.

3. Uso de identificadores y palabras reservadas.

En la elaboración de diagramas de flujo o algoritmos muchas veces se necesitan datos auxiliares, a continuación, se definen algunos de estos tipos de datos auxiliares.

Los identificadores son nombres que se dan a los elementos utilizados para resolver un problema y poder diferenciar unos de otros. Al asignar nombres (identificadores) a variables, constantes y procedimientos se deben tener en cuenta algunas reglas (López, 2009):

- Pueden estar formados por una combinación de letras y números (saldoMes, salario, fecha2, etc).
- El primer carácter de un nombre debe ser una letra.
- La mayoría de los lenguajes de programación diferencian las mayúsculas de las minúsculas.
- Los nombres deben ser nemotécnicos, con solo leerlos se puede entender lo que contienen.
- Deben ser muy descriptivos; no utilizar abreviaturas, a menos que se justifique.
- Es conveniente utilizar una sola palabra para nombrar páginas, controles, variables, etc.
- No utilizar caracteres reservados (%, +, /, >, etc.) tampoco letras acentuadas (á, é, í, ó, ú).
- No utilizar palabras reservadas por los lenguajes de programación.

Ejemplo: listar los identificadores usados en los algoritmos previos.

Solución: en el algoritmo de imprimir un número entero usando el identificador de **entrada** para representar el número digitado.
En el algoritmo de imprimir el cuadrado y el cubo de un número entero empleando el identificador de **entrada** para representar el número digitado, el identificador cuadrado para representar el valor del **cuadrado** de entrada y el identificador **cubo** para almacenar el cubo de **entrada**.

Todos los lenguajes de programación tienen unas palabras para nombrar sus comandos, instrucciones y funciones. Estas palabras no se pueden usar como indicadores.

a. Variables

Para los algoritmos con diversos conjuntos de datos iniciales, se debe determinar una independencia entre los datos iniciales de un problema y la estructura de su solución. Esto se hace mediante la utilización de variables (cantidades que se suelen emplear y que durante la ejecución del algoritmo pueden tomar cualquier valor de un intervalo de valores posibles).

Las variables son espacios de trabajo (contenedores) dispuestos para almacenar datos (valores). El valor de una variable puede ir cambiando en algún paso del algoritmo o permanecer sin cambios; por lo tanto, el valor de una variable es el del último valor o dato asignado a esta. En la ilustración 6, las variables eran los indicadores **cuadrados** y **cubo**.

b. Constantes

Las constantes se crean de la misma forma que las variables a diferencia que se procura su definición o creación a inicio del algoritmo. Las constantes consisten en datos que, luego de ser asignados, no cambian en ninguna instrucción del Algoritmo. Pueden contener constantes matemáticas (pi).

c. Contadores

Un contador es una variable que está en ambos lados de una asignación (a la derecha y a la izquierda) interna, a la que se le suma un valor constante. Es necesario haberla iniciado al principio del algoritmo, ya que va a ser leído su contenido.

Ejemplo: definir un contador llamado **contador** que se incremente en uno cada vez tres veces.

```
ALGORITMO Contador;
VAR
    ENTERO contador;
INICIO
    contador  = 0;
    contador <= contador + 1;
    contador <= contador + 1;
    contador <= contador + 1;
   ESCRIBIR( contador );
FIN
```

Ilustración 6: Contador en un algoritmo

Como se puede ver en el algoritmo, tres veces se incremente en 1 (uno) el valor del de contador y al final de la ejecución su valor es de 3. Es importante notar que contador inicia con un valor el valor 0, si esta asignación inicial no se hace el computador pondría un valor aleatorio y el valor al final del algoritmo sería indeterminado.

d. Acumuladores

Los acumuladores son una versión ampliada de contadores, tienen las mismas características que un contador excepto el valor que se va incrementando no es una constante sino una que es un valor variable.

4. Operadores y jerarquía en los operadores

Los operadores son símbolos que sirven para manipular datos. En un algoritmo es necesario dejar un espacio en blanco a cada lado del signo aritmético. Los operadores y las funciones que realizan se clasifican como se indica a continuación:

- **Aritméticos**: permiten realizar operaciones entre datos de tipo numérico y dan como resultado otro valor de tipo numérico. Ejemplo: producto o multiplicación (*); división (/); suma (+); resta (-); asignación (=).
- **Alfanuméricos**: permiten operar con datos de tipo carácter (letras) o cadenas de texto. La mayoría de los lenguajes de programación admiten el operador + para realizar la concatenación (unión) de caracteres o cadenas.
- **Relaciónales**: permiten la comparación entre datos del mismo tipo de dato y dan como resultado dos valores posibles: Verdadero o Falso. Ejemplo: igual a (=); menor que (<); mayor que (>,) mayor e igual que (>=).
- **Lógicos**: posibilitan la evaluación lógica de dos expresiones de tipo lógico. Dan como resultado uno de dos valores posibles: Verdadero o Falso. Ejemplo: negación (no); conjunción (y); disyunción(o).

A. Jerarquía en los operadores

Los computadores ejecutan los operadores en un orden establecido. El siguiente es el orden (jerarquía) para ejecutar operadores:

1. Paréntesis (se ejecutan primero los más internos)
2. Signo (-2), si un valor es positivo o negativo

3. Potencias (^) y Raíces (sqrt); Productos y Divisiones (* y /), en este mismo orden
4. Sumas y Restas (+ y -)
5. Concatenación (+)
6. Relaciónales (=, <, >)
7. Negación (no)
8. Conjunción (y)
9. Disyunción (o)

Con el siguiente ejemplo se espera sea más comprensible la jerarquía de operadores:

Orden	Operación	Explicación y expresión resultante
1	7 + 8	Primero que todo lo que más valor tiene en la jerarquía son los paréntesis de manera que y en particular los más interiores de esta forma el computador se interesara en resolver primero el contenido de (7 + 8). Y el resultado será 15. De esta forma la expresión queda reducida a: (2 + 15 * 16 + 3^2) /4 +5
2	3^2	De la expresión resultante luego de la operación 1, se observa que nuevamente hay un paréntesis por reducir, de manera que la computadora intentará resolver su contenido primero, como dentro del paréntesis está la expresión de elevar al cuadrado el número 3 (3^2) que tiene como resultado 9, la expresión resultante sería: (2 + 15 * 16 + 9) /4 +5
3	15 * 16	De los paréntesis de la expresión resultante la operación de más jerarquía es la multiplicación 15 * 16 que tiene como resultado 240, la expresión resultante sería: (2 + 240 + 9) /4 +5
4	2+24+9	De la expresión resultante el que tiene mayor precedencia es el paréntesis por lo tanto la suma aritmética de su contenido data como expresión resultante: 251/4 +5
5	251/4	De la expresión resultante el operador de mayor jerarquía es la división y da como resultado 62.75 que en la siguiente operación se le sumará el valor de 5 la expresión resultante será: 62,75 +5
6	62,75 +5	67,75

Tabla 2: Orden de ejecución de una expresión matemática

A continuación, se revisan dos ejemplos:

Ejemplo 1:

un computador debe ejecutar las siguientes operaciones, se debe escribir en el orden que el computador ejecutará las operaciones de la siguiente expresión matemática: (**2 + (7 + 8) * 16 + 3^2) /4 +5.**

Solución: primero que todo lo que más valor tiene en la jerarquía son los paréntesis de manera que y en particular los más interiores de esta forma el computador se interesara en resolver primero el contenido de (**7 + 8).**

Ejemplo 2:

Escriba la siguiente ecuación como una expresión que el computador pueda interpretar en el orden adecuado.

$$\frac{\sqrt[2]{\frac{6^2}{2}+7}}{8^2}$$

Solución:

sqrt (6^2/2 + 7) / 8^2

5. Estructuras de control secuencial

Una estructura es definida como un esquema con determinada distribución y orden que permite representar una idea de forma simplificada. La estructura de control secuencial es la más sencilla. También es conocida como estructura lineal. Se compone de instrucciones que deben ejecutarse en forma consecutiva, una tras otra, siguiendo una línea de flujo. Los problemas más sencillos pueden resolverse haciendo uso únicamente de esta estructura. Generalmente, la estructura secuencial hace parte de soluciones a problemas complejos, en las que se la utilizan otras estructuras iterativas (repetir varias veces un conjunto de instrucciones) y condicionales (López, 2009).

La siguiente imagen muestra la estructura general del algoritmo más básico.

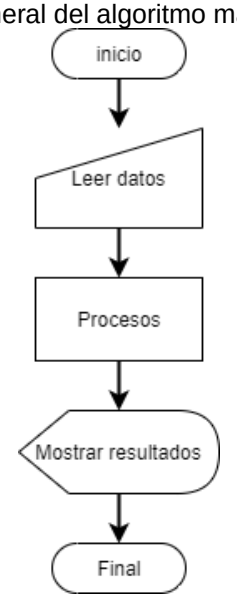


Ilustración 7: Modelo de estructura secuencial.

5.1. Ejemplo estructura de control secuencial

Escribir un algoritmo para calcular el área de cualquier triángulo rectángulo. En él se debe pedir al usuario que ingrese los valores de la altura y la base del triángulo

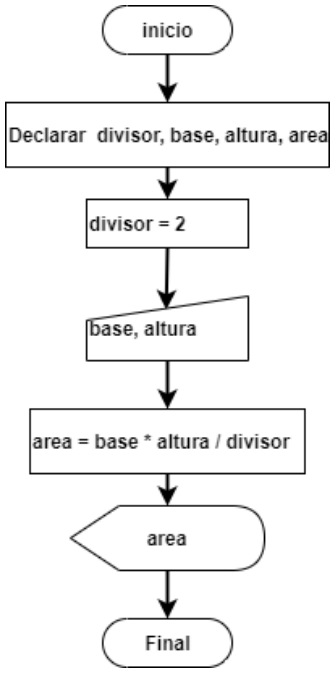
Análisis:

- **Formular el problema**: está claramente planteado consiste en calcular el área de un rectángulo.
- **Resultados esperados**: se espera que el algoritmo sea capaz de calcular el área del rectángulo y mostrar en la pantalla el resultado de ese cálculo.

- **Datos disponibles**: Dos datos números que son la base y la altura del triángulo (se deben solicitar al usuario). El aprendiz debe preguntarse si sus conocimientos actuales de matemáticas le permiten resolver este problema, de no ser así, debe plantear una estrategia para obtener los conocimientos requeridos. Para este ejemplo recordar que un triángulo rectángulo es aquel que uno de sus lados tiene un ángulo recto (90 grados)
- **Restricciones**: los valores de base y altura son variables y se deben solicitar al usuario.
- **Procesos necesarios**: definir variables; asignar el valor 2 a la constante usando para dividir (***divisor***); solicitar al usuario el valor de la altura del triángulo; solicitar al usuario el valor de la base; aplicar la fórmula de área; mostrar el resultado.

Diseño:

El algoritmo resultado del análisis se puede ver en la ilustración 8, están representados tanto el diagrama de flujo como en seudocódigo, mientras se está aprendiendo a diseñarlos, es importante hacerlos algoritmos en diagrama de flujo y en la medida que se va adquiriendo experticia con la representación de seudocódigo es más conveniente hacer esta representación en razón de la velocidad a la hora de diseñarlo.




```
ALGORITMO área de un triángulo;
VAR
    ENTERO divisor;
    REAL area;
    REAL base;
    REAL altura;
INICIO
   divisor = 2;
   ESCRIBIR( "Escriba la base ");
   LEER( base )
   ESCRIBIR( "Escriba la altura ");
   LEER( altura );
   area = base * altura / divisor;
   ESCRIBIR( area );
FIN
```

Ilustración 8: diagrama al área de un triángulo

Del diagrama de flujo y el seudocódigo podemos notar lo siguiente: el identificadora de área no tiene tilde por razón antes mencionada, el tipo de dato REAL hace referencia que es un número que puede contener decimales, es decir, que no solamente puede contener un número entero sino también un valor con cifras decimales.

5.2. Estructura condicional.

Es importante prestar atención a las estructuras que utilizan para resolver problemas y poderlas reconocer para lograr mayor control sobre la solución planteada.

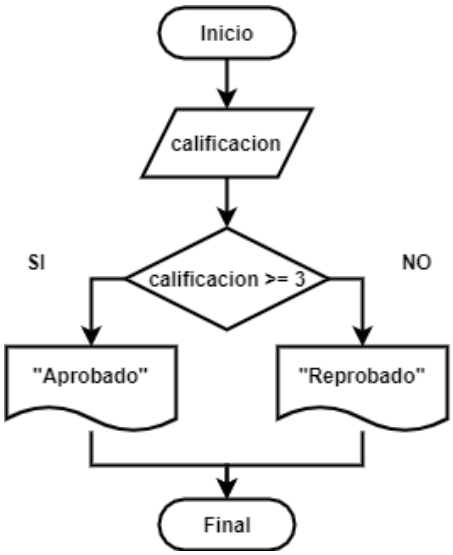
La estructura condicional se emplea para indicarle al computador que debe evaluar una condición y, según el resultado, ejecutar el bloque de instrucciones. La forma más común está compuesta por una condición que se evalúa y dos bloques de instrucciones que se ejecutan, uno cuando la condición es verdadera y otro cuando la condición es falsa.

Ejemplo: un aprendiz aprueba un examen cuando la calificación de este es mayor o igual a 3. Elaborar un algoritmo que pida al usuario una calificación, aplique el criterio de aprobación e imprima “Aprobado” o “Reprobado”, según sea el caso.

Análisis:

Formular el problema: Es un problema sencillo de condición con dos operaciones o condición doble.
Resultados esperados: Un aviso que reporte si el estudiante “Aprobado” o “Reprobado” el examen.
Datos disponibles: La calificación ingresada por el usuario. Para aprobar, la nota debe ser mayor o igual a 3.
Restricciones: Aplicar el criterio de aprobación (calificación mayor o igual a 3).
Procesos necesarios: Solicitar al usuario que ingrese la calificación. Evaluar si la calificación es igual o superior a 3; en caso de ser verdadero, reportar “Aprobado”; en caso contrario, reportar “Reprobado”.

Diseño:




```
ALGORITMO Calificación del usuario;
VAR
    REAL calificacion
INICIO
   ESCRIBIR( "Escriba la calificación");
   LEER( calificacion )
   SI ( calificación >= 3)
            ESCRIBIR( "Aprobado" );
   SI NO
            ESCRIBIR( "Reprobado" );
   FINSI
FIN
```

Ilustración 9: Estructura condicional doble.

En la Ilustración 9 se observa cómo se introduce la calificación por teclado, tanto en el diagrama de flujo (con la representación de trapecio) o el seudocódigo con la preposición **LEER**, luego este valor se compara en un condicional, en el diagrama de flujo con la representación de un rombo, mientras que en el seudocódigo se una la preposición **SI**, seguido de las preposiciones que se han de ejecutar en caso de que la condición sea verdadera (note que está escrita más a la derecha). También se usa la preposición **SI NO** para identificar el inicio de bloque de instrucciones a ejecutar si no se cumple la condición y la preposición **FINSI** para indicar que no hay más intrusiones que ejecutar si se cumple o no la condición.

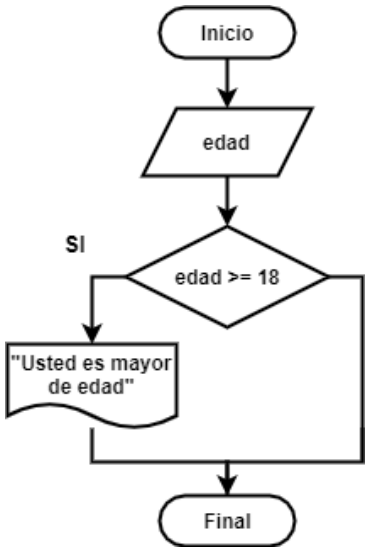
Como la condición evalúa dos condiciones si se cumple o no, y en cada caso existe un bloque de instrucciones (o alimenta una instrucción) a seguir se llama: ESTRUCTURA CONDICIONAL SIMPLE, la ilustración 9 corresponde a una de estas condicionales dobles, ya que si la condición se cumple escribe algo y si no se cumple también hace otra operación.

Ejemplo: escribir un algoritmo en donde se pida la edad del usuario; si es mayor de edad debe aparecer un mensaje indicándolo.

Análisis:

Formular el problema: Es un problema sencillo de condición simple.
Resultados esperados: Un mensaje que indique si el usuario es mayor de edad si no es mayor de edad no hacer nada.
Datos disponibles: la edad del usuario en años (número entero).
Restricciones: Aplicar el criterio de mayoría de edad (edad mayor o igual a 18).
Procesos necesarios: Solicitar al usuario que ingrese su edad. Evaluar si la edad es igual o superior a 18; en caso de ser verdadero “usted es mayor de edad”.

Diseño:




```
ALGORITMO Edad del usuario;
VAR
    REAL edad;
INICIO
   ESCRIBIR( "Escriba la calificación");
   LEER( edad )
   SI ( edad >= 18)
            ESCRIBIR( "Usted es mayor de edad" );
   FINSI
FIN
```

Ilustración 10: Estructura condicional simple.

Como se puede ver en la ilustración 10, no se ejecuta nada en caso de que NO se cumpla la condición. Este tipo de estructura condicional se denomina ESTRUCTURA CONDICIONAL SIMPLE. Como se observa en la anterior ilustración todo el proceso (pedir la edad, comprobar la condición, mostrar o no el mensaje) se realiza una única vez y finaliza la ejecución del algoritmo, si se deseara que el programa repitiera estas mismas operaciones más de una vez tendríamos que usar una estructura de iteración como se verá a continuación.

5.3. Estructuras de iteración o repetitivas.

Las estructuras iterativas o de repetición, permiten ejecutar una o varias instrucciones, un número determinado de veces o, indefinidamente, hasta que se cumpla una condición. Esta estructura ayuda a simplificar los algoritmos.

En programación existen como mínimo dos tipos de estructuras repetitivas, las cuales tienen variantes en los diferentes lenguajes de programación.

La característica común es que ambos tipos permiten ejecutar una o varias instrucciones:

- Un número conocido de veces (Estructura PARA o FOR en inglés).
- Mientras se cumpla una condición (Estructura MIENTRAS o WHILE).

5.3.1. Estructura PARA (*FOR*)

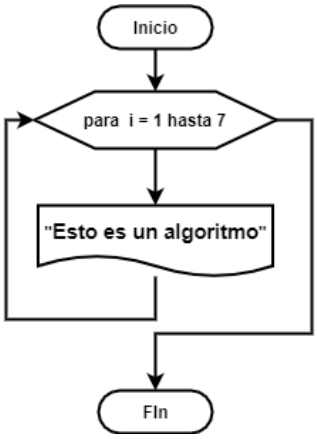
Esta estructura se usa cuando sabemos el número de veces que se debe ejecutar un conjunto de operaciones.

EJEMPLO: Escribir un procedimiento que muestre 7 veces en pantalla la frase “Esto es un algoritmo”. Mostrar en pantalla la salida del en el computador.

Análisis:

Formular el problema: Es un problema sencillo solo consiste en usar una sentencia de repetición.
Resultados esperados: el mensaje “Esto es un algoritmo” 7 veces impreso en pantalla.
Datos disponibles: constante con el número de repeticiones (7).
Restricciones: ninguna.
Procesos necesarios: hacer que se imprima en pantalla el mensaje 7 veces

Diseño:




```
ALGORITMO Ciclo para;
INICIO
   PARA  i DESDE 1 HASTA 7
       ESCRIBIR( "Esto es un algoritmo" );
   FINPARA
FIN
```

Ilustración 11: Estructura de repetición FOR.

La salida en pantalla sería:

```
Esto es un algoritmo
Esto es un algoritmo
Esto es un algoritmo
Esto es un algoritmo
Esto es un algoritmo
Esto es un algoritmo
Esto es un algoritmo
```

Como se puede ver en el algoritmo de la Ilustración 11, existe una variable **i** tendrá el valor 1, 2, 3 ... hasta 7, esta variable, se puede usar dentro del el bloque **PARA** hasta **FINPARA**.

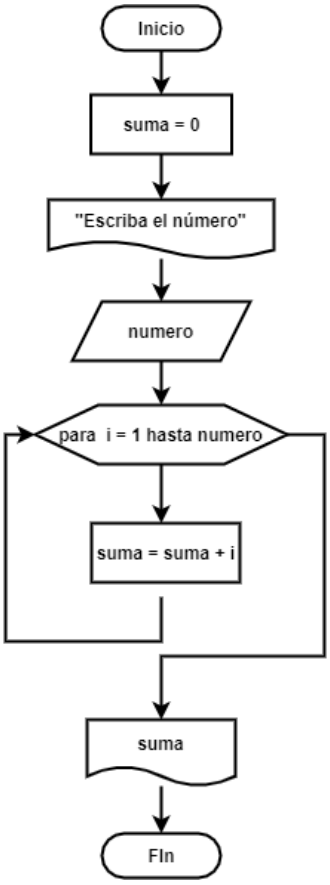
Miremos un ejemplo un poco más elaborado.

Ejemplo: se necesita elaborar un algoritmo que solicite al usuario un número entero y 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.

Análisis:

Formular el problema: necesita la sumatoria de los números en secuencia natural.
Resultados esperados: la suma de los números en sucesivos de uno hasta el número digitado por el usuario.
Datos disponibles: el dato entero que digita el usuario es la entrada.
Datos adicionales: Necesitamos un acumulador donde ir guardando el valor de las sumas sucesivas.
Restricciones: ninguna.
Procesos necesarios: definir un acumulador con valor inicial cero, imprimir para que el usuario sepa que debe digitar un número ("Escriba el número"), Leer del teclado el número del usuario, sumar cada vez el dato que se va incrementando de uno en uno al valor digitado por el usuario.

Diseño:




```
ALGORITMO sumas sucesivas;
VAR
     ENTERO número;
     ENTERO suma;
INICIO
   suma = 0;
   ESCRIBIR( "Escriba el número" );
   LEER(  número
   PARA  i DESDE 1 HASTA número
       suma = suma + i;
   FINPARA
   ESCRIBIR( suma );
FIN
```

Ilustración 12: Estructura de repetición FOR acumulador suma.

En la anterior ilustración se muestra el algoritmo en sus dos representaciones más comunes, se propone en secciones posteriores haremos una prueba de escritorio a este problema.

5.3.2.Estructura MIENTRAS (WHILE)

La instrucción "Mientras", puede impedir la ejecución de un conjunto de instrucciones, si la evaluación de una condición lógica es falsa. Esto significa que es repetitiva únicamente cuando la evaluación de la condición es verdadera.

Ejemplo: realice un programa que permita obtener la factorial de un número dado por el usuario.

Análisis:

Formular el problema: La factorial de un entero positivo n, se define como el producto de todos los números enteros positivos desde 1 hasta n.

Ejemplo: ¡Matemáticamente 5! = 5*4*3*2*1 ó a la inversa: 5! = 1*2*3*4*5.

Resultados esperados: la factorial del número digitado por el usuario.

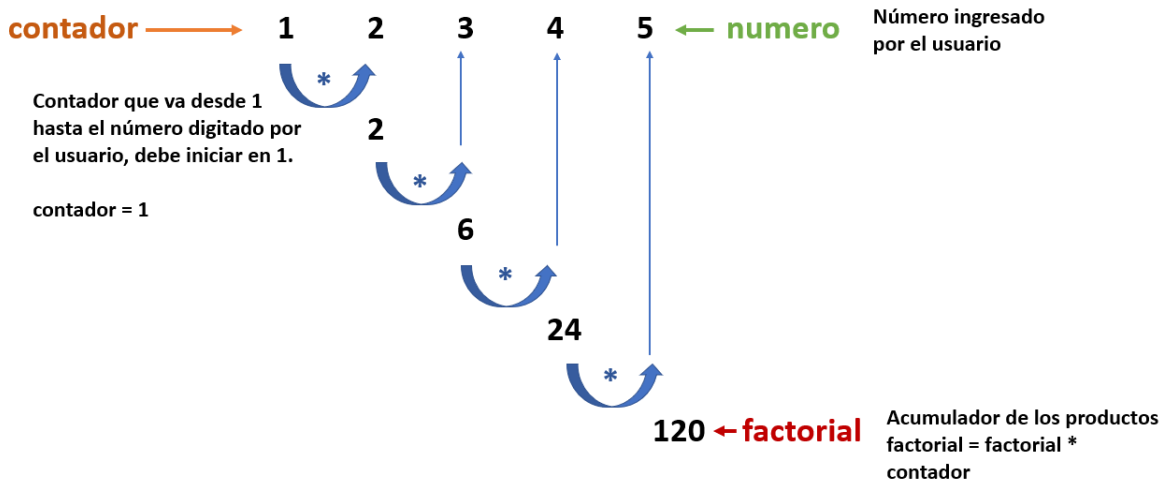
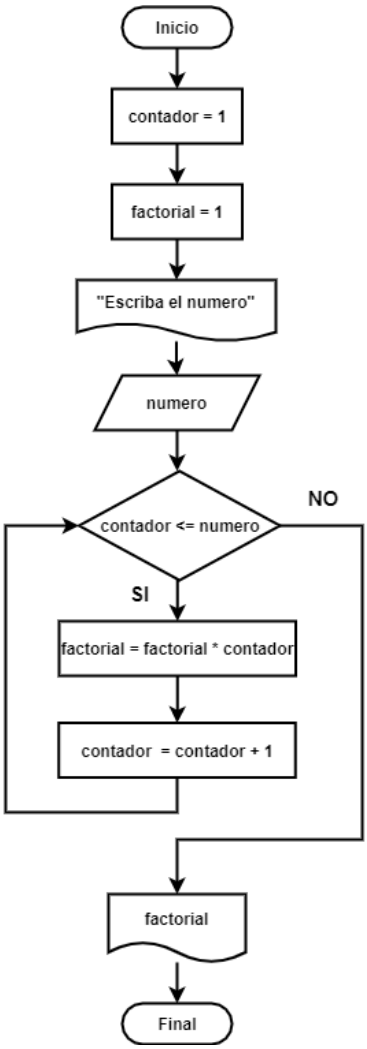


Ilustración 13: Proceso de factorial.

La anterior ilustración se observa que es preciso ir incrementando el valor en 1 (valor constante -contador-) hasta que llegue al número 5 (datos suministrados por el usuario); es decir, 1*2 = 2, este resultado se multiplica por 3 (2 * 3 = 6), 6 se multiplica por 4 (6 * 4 = 24), finalmente también 24 se multiplica por 5 y se obtiene la factorial (24 * 5 = 120) de esta forma los resultados de las multiplicaciones se van acumulando hasta obtener la factorial.

Análisis:
Datos disponibles: el dato entero que digita el usuario es la entrada.
Datos adicionales: se requiere de un acumulador donde ir guardado el valor de las multiplicaciones sucesivas, también un contador para ir contando el número secuencial (como lo hace el número i en el ciclo PARA).
Restricciones: la repetición se detiene cuando.
Procesos necesarios: definir un acumulador con valor inicial cero, imprimir para que el usuario sepa que debe digitar un número (“Por favor el número”), Leer del teclado el número del usuario, multiplicar cada vez el dato que se va incrementando de uno en uno al valor digitado por el usuario.

Diseño:




```
ALGORITMO 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
```

Ilustración 14: Algoritmo factorial ciclo mientras.

Como se puede observar si se hace la prueba de escritorio veremos cómo el proceso logra el objetivo, también se debe anotar que este algoritmo se puede hacer con una sentencia repetitiva PARA, de manera que queda como ejercicio.

Las estructuras de control repetitivas son muy útiles y necesarias a la hora de manipular tipos de datos más complejos que los hasta ahora vistos, como lo son los arreglos (vectores) o las matrices, ya que estos tipos de datos requieren realizar operaciones que se repiten más de una vez.

6. Estructura de datos

Es importante conocer el concepto de arreglos e identificar cuándo usarlos en la aplicación en los algoritmos. Así mismo ser capaz de resolver problemas básicos mediante diagramas de flujo y seudocódigo. Existen varios tipos de arreglos, los más comunes son los vectores y las matrices.

6.1. Vectores

Los arreglos o *array*, en inglés, (matriz, arreglo o vector) se definen como un conjunto finito y ordenado de elementos del mismo tipo. La propiedad "ordenado" quiere decir que el elemento primero, segundo, tercero, …, enésimo de un *array* puede ser identificado y accedido. Cuando se dice que los elementos de un arreglo son homogéneos, significa que es del mismo tipo de datos. Un arreglo puede estar compuesto de todos sus elementos de tipo letras, otro puede tener todos sus elementos de tipo entero, etc. Los arreglos se conocen como matrices en matemáticas, y tablas en temas financieros (Vazquez, 2012).

Según sus dimensiones los arreglos se clasifican en:

- **Unidimensionales:** de una dimensión, comparables con una lista de elementos.
- **Multidimensionales**: son arreglos de varias dimensiones o los más comunes son bidimensionales (dos dimensiones o tablas) o Arreglos multidimensionales (tridimensionales, por ejemplo.)

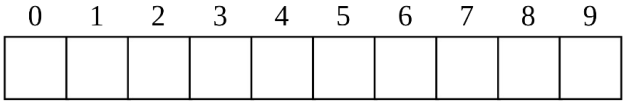


Ilustración 15: Matriz unidimensional con 10 elementos

Cada vector está compuesto de un número de elementos ya definidos, esto hace que sea una estructura estática (no puede aumentarse ni reducirse). Cada elemento está referenciado por la posición que ocupa dentro del vector. Estas posiciones son llamadas índices. Nótese que empieza a contar desde 0 como muestra la Ilustración 15.

Para referirse un vector se usa un identificador como lo hacemos con las variables o constantes, pero queda entre corchetes **[ ]** por ejemplo si el vector se llama **lista**, y queremos acceder al quinto elemento del vector de la Ilustración 15 debemos hacerlo así **lista[4]**, note que la quinta posición se accede con el índice 4, porque se empezó a contar desde cero.

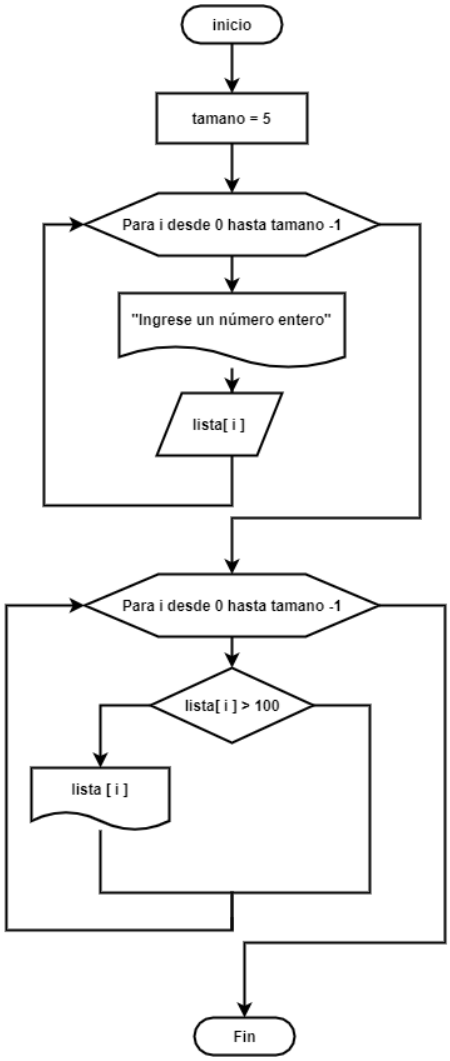
Para realizar las operaciones de asignación, lectura/escritura sobre un vector se recomienda usar las estructuras repetitivas, a través de ellas manipular los índices correspondientes a las posiciones del arreglo.

Ejemplo: acer un programa que pida 5 números enteros al usuario y los guarde en un arreglo de números en un arreglo, luego que imprima los numero mayores de 100.

Análisis:

Formular el problema: pedirle al usuario 5 números enteros almacenarlos en un arreglo y luego recorrer el arreglo para buscar los números mayores a 100.
Resultados esperados: imprimir en pantalla de los 5 números ingresados por el usuario solamente los mayores a 100.
Datos disponibles: 5 números ingresados por el usuario.
Datos adicionales: por cada uno de esos números validar si son mayores que 100 y mostrar los que lo sean en la pantalla.
Restricciones: ninguna.
Procesos necesarios: se requiere definir un arreglo de tamaño 5, se necesita pedir 5 veces un entero y almacenarlos en el vector. Luego se debe recorrer las 5 posiciones del vector para comparar en cada uno de los datos si es mayor que 100, si el número comparador lo que se debe imprimir. Como 5 es un número invariante el programa lo define como una constante de nombre **tamaño** (no se usa la ñ porque no permite ni acentos latinos ni tildes), usaremos la sentencia repetitiva PARA porque sabes el número de veces que se debe repetir cada acción.

Diseño:




```
ALGORITMO Vector de 5 enteros;
VAR
     ENTERO tamano;
     ENTERO lista[tamano];
INICIO
   tamano = 5;

   PARA  i DESDE 0 HASTA tamano -1
      ESCRIBIR( "Escriba el número" );
      LEER( lista[ i ] );
   FINPARA

   PARA  i DESDE 0 HASTA tamano -1
      SI ( lista[ i ] > 100 )
           ESCRIBIR(lista[ i ] );
      FINSI
   FINPARA
FIN
```

Ilustración 16: Vector de 5 elementos.

6.2. Matrices.

Las matrices son arreglos de más de una dimensión, en la siguiente ilustración vemos un ejemplo de una matriz de 2 dimensiones que son 4x5 (correspondiente a 4 columnas y 5 filas).

0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3
4,0	4,1	4,2	4,3

Ilustración 16: Matriz de 4 x 5

Cuando se quiere acceder de un elemento del vector si el identificador se usa el corchete doble. Por ejemplo, cuando para acceder al último elemento de la Ilustración 16, usado es matriz seria **matriz[4][3]**;

Para recorrer estas estructuras de datos se requiere ciclos anidados, y estas técnicas se irán abordando más adelante.

7. Programación modular

Existe un área que se denomina ingeniería del software que se interesa en el proceso de creación o producción de software y que concentra sus esfuerzos en aportar herramientas, procedimientos y técnicas para su construcción.

La ingeniería del software propone que la construcción de software este pensada en la descomposición de un problema en un conjunto de subproblemas independientes entre sí, más sencillos de analizar y resolver; estos subproblemas pueden ser tratados separadamente unos de otros. Esto básicamente consiste en la modularidad del software.

Debido a la modularidad se pueden probar los subprogramas o módulos de manera independiente, depurando errores antes de su uso en el programa principal y almacenándose para su posterior reutilización cuantas veces se necesite.

El enfoque básico de modularidad en un algoritmo puede estar compuesto de otros algoritmos que se especializan en una función determinada. Para saber cómo descomponer un algoritmo en pequeños algoritmos se listan los aspectos que pueden ayudar a definir la división de algoritmos:

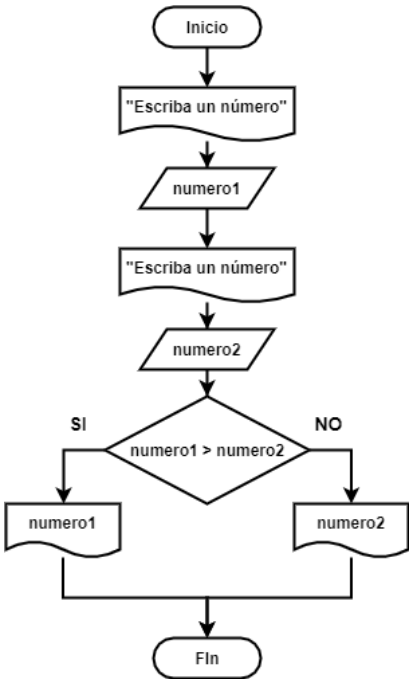
a. **Según la funcionalidad:** un algoritmo debe realizar una función específica y no ir más allá de sus responsabilidades. Por ejemplo, si tenemos un módulo que calcula la multiplicación de dos números y devuelve el resultado, no sería adecuado que además imprima mensajes en pantalla o haga otra cosa diferente de su propósito.
b. **Según su identificación:** cada módulo tiene un nombre de identificación que nos permite ejecutarlos y diferenciarlos de otros, es recomendable que elegir este nombre que sea representativo de la función que realiza, de esa forma resulta intuitivo en el momento de usarlo dentro de un algoritmo.
c. **Según sus datos de entrada:** los módulos comúnmente reciben datos de entrada, necesarios para realizar las operaciones requeridas. Se pueden clasificar los módulos de acuerdo a los parámetros que se requieren para obtener los resultados solicitados. En el momento de usar el módulo se tienen que agregar todos los parámetros necesarios, en el orden establecido y deben ser del tipo correcto.
d. **Parámetro de salida:** también se puede emplear los resultados a la salida del algoritmo para clasificar los módulos, ya que en el momento de definir un algoritmo se debe indicar el tipo de dato que devuelve al momento de la ejecución del algoritmo.
e. **Pruebas de escritorios:** como se mencionó antes las pruebas escritorio nos sirven para depurar y probar que un algoritmo solucione el problema, en esta sesión mencionaremos los objetivos principales y ejecutaremos una prueba a un algoritmo de mejor complejidad para su comprensión.

Ejemplo: realizar un algoritmo que solicite dos números e imprima cuál es el mayor de ambos, si son iguales mostrar cualquiera.

Análisis:

Formular el problema: es un algoritmo que solita y compara dos números para mostrar el mayor.
Resultados esperados: el número que es mayor de ambos.
Datos disponibles: 2 números ingresados por el usuario.
Datos adicionales: ninguno.
Restricciones: validar cuál número es mayor para saber cuál mostrar.
Procesos necesarios: se requiere definir los datos a recibir, pedirle que al usuario que digite los números, comparar para saber cuál es el mayor y mostrarlo.

Diseño:




```
ALGORITMO El mayor de dos números;
VAR
     ENTERO numero1;
     ENTERO numero2;
INICIO
   ESCRIBIR( "Escriba un número" );
   LEER( numero1 );
   ESCRIBIR( "Escriba un número" );
   LEER( numero2 );
   SI  numero1 > numero2
      ESCRIBIR( numero1 );
   SINO
      ESCRIBIR( numero2 );
   FINSI
FIN
```

Ilustración 17: Algoritmo el mayor de dos números

Diseño de prueba 1:
Selección de datos: Usaremos para la prueba los números 34 y 16
Salida esperada: debe mostrar el número 34.

1. ESCRIBIR("Escriba en número");

numero1	numero2	Salida en pantalla
		Escriba un número

2. LEER(numero1);

numero1	numero2	Salida en pantalla
34		Escriba un número
		34

3. ESCRIBIR("Escriba en número");

numero1	numero2	Salida en pantalla
34		Escriba un número
		34
		Escriba un número

4. LEER(numero1);

numero1	numero2	Salida en pantalla
34		Escriba un número
	16	34
		Escriba un número
		16

5. SI numero1 > numero2

6. ESCRIBIR(numero1);

numero1	numero2	Salida en pantalla
34		
	16	

numero1	numero2	Salida en pantalla
34		
	16	

Ilustración 18: Prueba de escritorio de algoritmo el mayor de dos números

Una prueba de escritorio no es determinante en este ejercicio, es necesario hacerla para los dos casos faltantes, en caso que el segundo número es mayor que el primero y el caso en que ambos números ingresados son iguales. Se propone que use el ejemplo de la Ilustración 18 para realizar el desarrollo final del ejemplo.

I. ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)

DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA	
Nombre de la Actividad	NA
Objetivo de la actividad	
Tipo de actividad sugerida	
Archivo de la actividad (Anexo donde se describe la actividad propuesta)	

J. MATERIAL COMPLEMENTARIO:

Referencia APA del Material	Tipo de material (Video, capítulo de libro, articulo, otro)	Enlace del Recurso o Archivo del documento o material
Material_Audiovisual_100319633. (6 de enero. 2021). App Diagrams.net (Básico para Diagramas de flujo). Youtube. https://youtu.be/jdnqmFpqTvo	Video	https://youtu.be/jdnqmFpqTvo
ENRIQUE OLIVARES. (23 de febrero 2016). introducción a utilizar el programa DIA para diagramas de flujo a nivel Bachillerato. https://youtu.be/DnRx9D7QWg8	Video	https://youtu.be/DnRx9D7QWg8
DiscoDurodeRoer. (27 de abril 2015). Ejercicios PseInt - Básicos #1 - Empezando por lo básico. https://youtu.be/DHIi4dcaMEc	Video	https://youtu.be/DHIi4dcaMEc

Insituto de Informatica Uach. (2020, Sept. 08). Como descargar draw io 2 para dibujar Diagramas de Flujo. Youtube. Retrieved from https://www.youtube.com/watch?v=encz3h8TLWA	Vídeo	https://www.youtube.com/watch?v=encz3h8TLWA

K. GLOSARIO:

TERMINO	SIGNIFICADO
Modularidad	Consiste en dividir un programa en módulos, cada uno de ellos debe poder ejecutarse por separado, pero están diseñados para trabajar o con otros módulos.
Iteración	La iteración es repetir sucesivas veces un proceso, con la finalidad de lograr una meta resultado u objetivo. a cada repetición del proceso también se le denomina una "iteración", generalmente y los resultados obtenidos en una iteración se emplean como punto de partida hacia la siguiente iteración.
Concatenar	Unir o enlazar dos o más cosas (RAE, 2021)

L. REFERENCIAS BIBLIOGRÁFICAS:

López Juan Carlos, 2009, Algoritmos y programación (guía para docentes), https://libros.metabiblioteca.org/bitstream/001/169/8/AlgoritmosProgramacion.pdf

RAE, Asale, R. (2021, Apr. 12). concatenar | Diccionario de la lengua española. Retrieved from https://dle.rae.es/concatenar

Resnick, Mitchel (2007a): Sembrando semillas para una sociedad más creativa. Laboratorio de medios de MIT, Massachussets Eduteka, marzo 16, 2009, http://www.eduteka.org/ScratchResnickCreatividad.php.

Rojas A., Vicente & Ñacato C., José (1980): Técnica de flujogramas I; Editora Andina, Quito, séptima edición. https://biblioteca.epn.edu.ec/cgi-bin/koha/opac-detail.pl?biblionumber=27333

Vazquez Juan Bernardo, 2012, ANÁLISIS Y DISEÑO DE ALGORITMOS, http://www.aliat.org.mx/BibliotecasDigitales/sistemas/Analisis_y_disenio_de_algoritmos.pdf

M. CONTROL DEL DOCUMENTO

	Nombre	**Cargo**	**Dependencia** *(Para el SENA indicar Regional y Centro de Formación)*	**Fecha**
Autor (es)	Henry Eduardo Bastidas Paruma	Experto temático	Regional Cauca, Centro de teleinformática y producción industrial	Abril 2021
	Peter Emerson Pinchao Solis	Experto temático	Regional Cauca, Centro de teleinformática y producción industrial	Abril 2021
	Oscar Absalón Guevara	Diseñador instruccional	Regional Bogotá, Centro de Gestión Industrial.	Abril 2021

N. CONTROL DE CAMBIOS

(Diligenciar únicamente si realiza ajustes a la Unidad Temática)

	Nombre	Cargo	Dependencia	Fecha	Razón del Cambio
Autor (es)					

FORMATO PARA EL DESARROLLO DE COMPONENTE FORMATIVO

PROGRAMA DE FORMACIÓN	SOLUCIÓN DE PROBLEMAS MEDIANTE LA UTILIZACIÓN DE ALGORITMOS

NÚMERO DEL COMPONENTE FORMATIVO	1
NOMBRE DEL COMPONENTE FORMATIVO	Introducción a la algoritmia
BREVE DESCRIPCIÓN	Es importante reconocer los componentes que forman parte de un problema desde un punto de vista lógico y procedimental, ya que estos son las bases del análisis y diseño de algoritmos. Este componente orienta el pensamiento y despierta la conciencia de a este tipo de análisis aportando un enfoque metodológico para la solución de problemas.
PALABRAS CLAVE	Algoritmo, Problema, Lógica, Análisis.

COMPETENCIA	Desarrollar la solución de software de acuerdo con el diseño y metodologías de desarrollo.	RESULTADOS DE APRENDIZAJE	220501096 - 01 Identificar los componentes de un algoritmo para dar solución a un problema.

ÁREA OCUPACIONAL	No aplica
IDIOMA	Español

A. TABLA DE CONTENIDOS:

Introducción

1. Introducción a la algoritmia.

1.1. Definición de algoritmo.
1.2. Pensamiento algorítmico.
1.3. Solución de problemas y programación.
1.4. Análisis de problema (entenderlo)
    1.4.1. Formular el problema.
    1.4.2. Precisar elementos de entrada
    1.4.3. Precisar resultados esperados
1.5. Desarrollo de la creatividad elementos, modelos, fases y sus objetivos.

2. **Lógica matemática.**

2.1. Componentes de la lógica matemática
2.2. Ejercicios de lógica proposicional.
    2.2.1. Ejercicio 1
    2.2.2. Ejercicio 2
    2.2.3. Ejercicio 3

B. DESARROLLO DE CONTENIDOS:

Introducción

En la vida cotidiana se debe imaginar, diseñar y planear soluciones a problemas corrientes del día a día, por ende, muchas veces la estrategia de “divide y vencerás” es empleada para afrontar la solución de estos problemas la cual consiste en tomar un problema grande y segmentarlo en pequeños sub problemas (o pequeñas metas) que son más fáciles de abordar, y la unión logra la solución a lo inicialmente planteado.

Desde un punto de vista computacional esta estrategia es la más empleada y los grandes sistemas informáticos son la unión de un conjunto de instrucciones computacionales que resuelven problemas muy puntuales.
Desde aquí se propone facilitar técnicas, para el análisis de problemas usando algoritmos que a futuro pueden llegar a ser parte constitutiva de un sistema teleinformático.

1. Introducción a la algoritmia

La aplicación apropiada de una serie de pasos muy detallados, puede garantizar una solución correcta de un problema. La programación de computadores (o construcción de software) está basada en esta práctica.



Imagen: pasos en la algoritmia básica.

1.1. Definición de algoritmo

Según la Real academia de la lengua española (RAE) un algoritmo está definido como: conjunto ordenado y finito de operaciones que permite hallar la solución de un problema (Asale, R.-, & Rae, 2021).

Entendiendo que computacionalmente las máquinas están en capacidad de hacer unas pocas operaciones matemáticas muy básicas y sencillas, pero también se considera que computan enormes cantidades de estas operaciones en unidades tiempo muy corto.

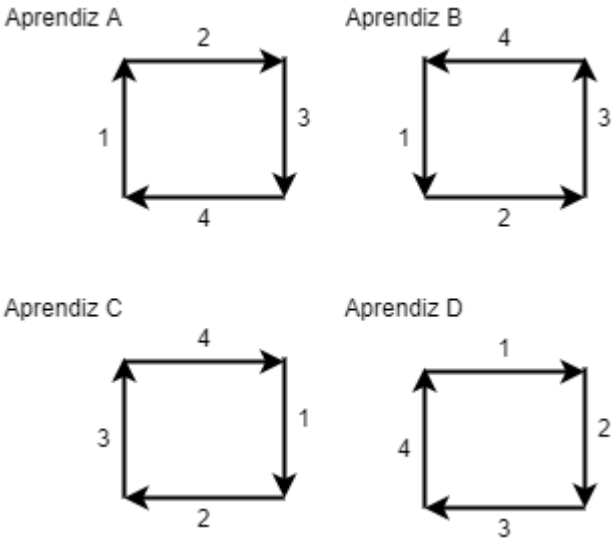
La solución de un problema usando computadoras fácilmente se puede convertir en una enorme cantidad de pequeñas operaciones matemáticas, tal vez muy abrumadoras desde el punto de vista humano. Es por eso que se requiere, la especialización de conjuntos de operaciones que se agrupan en pequeños procesos que componen la solución de un problema.

Para ampliar la definición puede visualizar el siguiente vídeo: https://www.youtube.com/watch?v=U3CGMyjzlvM

1.2. Pensamiento algorítmico

Un problema puede ser abordado de varias maneras y aportar la misma solución, pero lo que siempre tendrán en común es que para resolverlo usaron una secuencia de pasos ordenados.

Por ejemplo: si se pasa una hoja de papel cuadriculado a los aprendices de una clase y les pide que dibujen un cuadrado sin levantar el lápiz habrá quienes lo hagan de diferente manera y obtengan el mismo resultado cómo muestra la siguiente imagen:



Como se puede ver el aprendiz A, inicio haciendo una línea vertical hacia arriba mientras el aprendiz B una vertical hacia bajo o el estudiante D una horizontal hacia la derecha, y todos siguiente un proceso que les permitió cumplir con la solución al problema planteado, de forma global prácticamente tenemos una gran cantidad de soluciones al mismo problema.

Ilustración 1: Pensamiento lógico y procedimental

Imaginando que el problema es mucho más complejo que el descrito en el ejemplo se concluye que existen tantas soluciones como el enfoque o lógica empleada por cada aprendiz o persona.

Pero lo que sí es común en todos los casos es que se realizaron unas fases mentales secuencialmente ordenadas que son:

1. Entender el problema enunciado por el instructor. (Entender el problema)
2. Escoger una ruta a seguir del rectángulo (Hacer un plan)
3. Plasmar en el papel cuadriculado ese plan. (Ejecutar el plan)
4. Revisar el resultado y ver que está completado el objetivo. (Revisar)

Cuando el problema es muy complejo de resolver, muchas veces toca volver y analizar el problema o trazar un nuevo plan o plasmar ese nuevo plan una y otra vez revisando los resultados y volver a analizar hasta encontrar la solución al problema.

1.3. Solución de problemas y programación

En el ámbito de la computación, los algoritmos son una herramienta que permite describir claramente un conjunto finito de instrucciones, ordenadas secuencialmente y libres de ambigüedad, que debe llevar a cabo un computador para lograr un resultado previsible. Vale la pena recordar que un programa de computador consiste de una serie de instrucciones muy precisas y escritas en un lenguaje de programación que el computador entiende (Logo, Java, Pascal, etc). (López, 2019)

Ejemplo: se propone diseñar un algoritmo para apagar una computadora.

```
Algoritmo: apagar computadora
INICIO
1.-Dar clic en el icono INICIO
2.-Dar clic en el icono que dice APAGAR
3.-Esperar que se apague completamente
4.-Desconectar los cables del enchufe
FIN
```

Ejemplo: se propone diseñar un algoritmo para cargar un celular

```
Algoritmo: Cargar un celular
INICIO.
1. Buscar el cargador.
2. Agarrar el cargador.
3. Agarrar el celular.
4. Poner el enchufe en el celular
5. Conectarlo a tomacorriente.
6. Esperar a que se ponga a cargar.
7. Desconectarlo del tomacorriente.
8. Ponerlo en un lugar seguro.
FIN
```

Con base a los ejemplos anteriormente planteados se puede concluir, que un algoritmo debe ser:

- ✔ **Realizable**: el proceso algorítmico debe terminar después de una cantidad finita de pasos. Se dice que un algoritmo es inaplicable cuando se ejecuta con un conjunto de datos iniciales y el proceso resulta infinito o durante la ejecución se encuentra con un obstáculo insuperable sin arrojar un resultado (Lopez, 2009).
- ✔ **Comprensible**: debe ser claro lo que hace, de forma que quien ejecute los pasos (ser humano o máquina) sepa qué, cómo y cuándo hacerlo. Debe existir un procedimiento que determine el proceso de ejecución (Lopez, 2009).
- ✔ **Preciso**: el orden de ejecución de las instrucciones debe estar perfectamente indicado. Cuando se ejecuta varias veces, con los mismos datos iniciales, el resultado debe ser el mismo siempre. La precisión implica determinismo (Lopez, 2009).

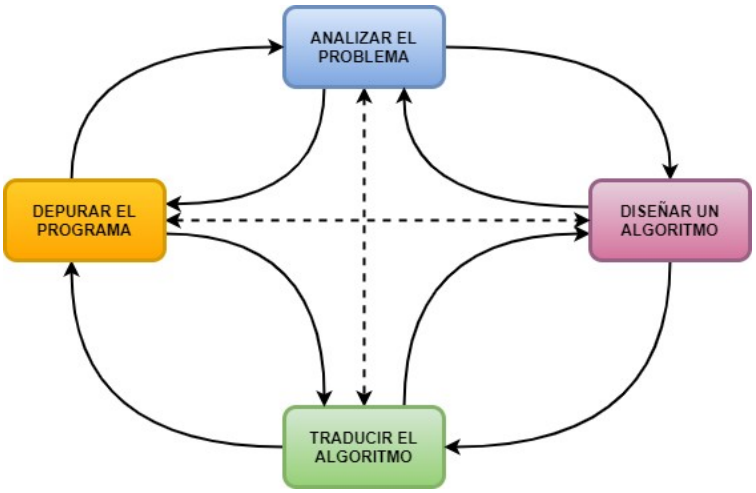


Ilustración 2: Fases del ciclo de programación

La ilustración 2 muestra las fases y sus relaciones en el proceso de diseño de algoritmos de programas informáticos, que están basados en un algoritmo que posteriormente es escrito en un lenguaje de programación.

1.4. Análisis del problema (entenderlo)

Cómo los programas de computador tienen como objetivo resolver problemas muy puntuales lo primero que se debe realizar para resolverlas es lograr una mejor comprensión posible de estas.

La forma de realizar esta actividad consiste en:

I. Formular claramente el problema
II. Especificar los resultados que se desean obtener con la solución.
III. Identificar la información disponible los datos.

1.4.1. Formular el problema

Algunas preguntas modeladoras antes de formular el problema son:

- ✔ ¿Puedo definir de forma más clara el problema?
- ✔ ¿Qué palabras no me son conocidas o desconozco el significado?
- ✔ ¿He resuelto antes algún problema similar?
- ✔ ¿Qué información es importante para resolver el problema?
- ✔ ¿Qué información NO es importante y puedo omitir?

Ejemplo

Don Juan necesita decidir cómo comprar un teléfono celular que cuesta $870.000 de contado o $980.000 a crédito, él tiene $630.000 pesos en efectivo.

Luego de revisar las preguntas modeladoras se puede concluir:

Como el efectivo que tiene don Juan no le alcanza para comprar el teléfono celular de contado, entonces él tiene dos opciones: comprarlo completamente a crédito o también pagar una parte al contado (como cuota inicial) y el resto diferirlo a crédito.

Para poder resolver este problema se debe conocer el número de cuotas en la que difiere si desea pagarlo totalmente a crédito o conocer el número de cuotas y el valor total del celular si se da una cuota inicial de $630.000 pesos.

1.4.2. Precisar elementos de entrada

Otro punto importante en la fase de análisis del problema es determinar cuál es la información disponible. Por lo tanto:

- ✔ ¿Qué información es importante y necesaria para resolver el problema?
- ✔ ¿Qué información no es importante y se puede prescindir?
- ✔ ¿Cuáles son los datos de entrada conocidos?
- ✔ ¿Cuál es la incógnita o que debemos calcular?
- ✔ ¿Los datos se pueden agrupar en categorías?
- ✔ ¿Qué información adicional hace falta para resolver el problema?

Otro elemento a resaltar es el nivel de conocimiento y las limitaciones que se tienen en el ámbito o el contexto del problema que está tratando de resolver. Por ejemplo, si el problema requiere conceptos financieros para entender la naturaleza de los datos, o si es necesario adquirir nueva información y/o poder consultarla, para estar seguros de que es posible tratarla conforme el contexto lo requiera.

1.4.3. Precisar resultados esperados

Determinar con claridad cuál es el resultado final esperado (o producto) que se quiere obtener, es necesario determinar o definir qué resultados se solicitan y en qué tipo, o formato deben tener esos (mostrados en pantalla, diagramación, orden, etc.). Para lograrlo es bueno preguntarse:

- ✔ ¿Qué información están solicitando?
- ✔ ¿En qué formato se debe entregar esta información?

1.5. Desarrollo de la creatividad elementos, modelos, fases y sus objetivos.

Resolver algoritmos es una actividad que requiere en la mayoría de los casos creatividad por parte del programador, esta es una habilidad que se va fortaleciendo en la medida que se desempeña más y más en procesos de desarrollo de software.

Para estimular el pensamiento creativo propuesto por (Resnick, 2007) el programador debe imagina que quiere hacer; crea un proyecto basado en sus ideas; juega y comparte sus ideas y creaciones con otros para reflexionar sobre sus experiencias y permitirse imaginar nuevas ideas y nuevos proyectos. La espiral genera un proceso indefinido de mejoramiento continuo (López, 2009).



Ilustración 3. Espiral del Pensamiento Creativo diseñada por el Dr. Mitchel Resnick

En conclusión, se puede decir que un algoritmo es la definición de una secuencia lógica de pasos que permiten solucionar un problema. Formular algoritmos es la base de los grandes adelantos en las ciencias matemáticas y tecnología, ya que se pueden aplicar para cualquier área del conocimiento.

2. Lógica matemática

La lógica matemática es la disciplina que trata de métodos de razonamiento. En un nivel elemental, la lógica proporciona reglas y técnicas para determinar si es o no válido un argumento dado. El razonamiento lógico se emplea en matemáticas para demostrar teoremas; en ciencias de la computación para verificar si son o no correctos los programas; en las ciencias físicas y naturales, para sacar conclusiones de experimentos; y en las ciencias sociales y en la vida cotidiana, para resolver una multitud de problemas. Ciertamente se usa en forma constante el razonamiento lógico para realizar cualquier actividad (Jiménez 2007).

Para ampliar los fundamentos de la lógica matemática puede revisar el vídeo: https://www.youtube.com/watch?v=9zYDaZhS7Ac

2.1. Componentes de la lógica matemática

Todo razonamiento lógico está compuesto por una serie de elementos, que se interrelacionan entre sí, la correcta interpretación de estos componentes y sus relaciones son lo que denominamos el análisis deductivo.

Estos componentes son las preposiciones y conectores lógicos descritos a continuación:

A. Proposiciones

Una proposición o enunciado es una oración que puede ser falsa o verdadera pero no ambas a la vez. La proposición es un elemento fundamental de la lógica matemática.

Las proposiciones pueden ser simples o compuesta, las primeras están compuestas por una sola proposición, por el contrario, en las proposiciones compuestas están conformadas por dos o más proposiciones unidas por un conector.

A continuación, se muestran una serie de ejemplos de proposiciones válidas y no válidas, con estos se explicará por qué algunos enunciados no son proposiciones.

p: La tierra es plana
q: -17 + 38 = 21
r: Hola ¿Cómo estás?
S: Lava el Carro por favor

Para los enunciados anteriores podemos determinar lo siguiente:

p y q son proposiciones simples válidas porque pueden tomar un valor verdadero o falso, sin embargo, los enunciados r y s son proposiciones no válidas, ya que no pueden tomar un valor de falso o verdadero, uno de ellos es un saludo y el otro es una orden.

B. Conectores lógicos y proposiciones compuestas.

Existen las proposiciones compuestas que son las que se componen de varias proposiciones las cuales utilizan para su conexión operadores o conectores lógicos que son:

- ✔ **Operador AND (y) Conjunción**

  Conecta dos proposiciones que se deben unir para que se pueda obtener un resultado verdadero, esta se representa por el siguiente símbolo ∧

  Ejemplos:

  - Nuestra bandera es de color amarillo, azul y rojo
  - El número 9 es mayor que 7 y menor que 12
  - El carro enciende cuando tiene corriente eléctrica y gasolina en el tanque.

- ✔ **Operador OR (o) Disyunción**

  Conecta dos proposiciones y se obtiene una proposición compuesta verdadera cuando por lo menos unas de las proposiciones son verdaderas.

  Ejemplos:

  - Puedo ir a Bogotá en avión o en bus
  - Puedo entrar al estadio si compro mi boleta o si tengo un pase de cortesía

- Voy a desayunar con huevos revueltos o huevos pericos

✔ **Operador NOT (no) Negación.**

Su propósito es negar una proposición es decir que si tenemos una proposición verdadera al aplicarle este tipo de operador se obtiene la negación es decir falso.

Ejemplos:

- No es cierto que Gabriel García Márquez escribió la Ilíada.
- No es cierto que el América de Cali es de la ciudad de Bogotá.
- Es falso que el agua es un hidrocarburo

2.2. **Tablas de verdad**

Es una estrategia de la lógica que tiene como objetivo determinar la validez de varias propuestas en alguna situación, es decir, determina las condiciones necesarias en las que un enunciado es verdadero, estas condiciones se representan con una tabla de verdad que determina el valor de verdad de todas las proposiciones que componen la situación. Así como lo explica el principio de bivalencia: **una proposición puede ser verdadera o falsa únicamente, nunca ambas**.

Algunas consideraciones:

- El valor de una proposición verdadera se representa por “V”, o un “1”
- El valor de una proposición falsa se representa por “F” o un “0”
- Existe una tabla de verdad para cada uno de los conectores lógicos, lo que se tiene que evaluar para responder si una proposición compuesta es verdadera o falsa es el estatus actual de las proposiciones simples. A continuación, se muestran las tablas de verdad.

P	Q	P ∧ Q
V	V	V
V	F	F
F	V	F
F	F	F

O puede ser

P	Q	P ∧ Q
1	1	1
1	0	0
0	1	0
0	0	0

Tabla 1. Tabla de verdad Conjunción(Y)

P	Q	P ∨ Q
V	V	V
V	F	V
F	V	V
F	F	F

O puede ser

P	Q	P ∨ Q
1	1	1
1	0	1
0	1	1
0	0	0

Tabla 2. Tabla de verdad Disyunción (O)

P	¬ P
V	F
F	V

O puede ser

P	¬ P
1	0
0	1

Tabla 3: Tabla de verdad Negación (NO)

Existe otro tipo de conectores que llamaremos proposiciones condicionales que son aquellas que están formadas por dos proposiciones simples o compuesta de p y q la cual se representa y lee de la siguiente manera:

p → q se lee “Si p entonces q”

Ejemplo 1:

Si no llueve se arruina la cosecha.
Este tipo de proposición se conoce como la condición “para que” de manera que para que no se arruine la cosecha se requiere que llueva.

p: Si no llueve
q: se arruina la cosecha

Al aplicar el operador p → q su tabla de verdad quedaría de la siguiente manera.

P	Q	P → Q
V	V	V
V	F	F
F	V	V
F	F	V

O puede ser

P	Q	P → Q
1	1	1
1	0	0
0	1	1
0	0	1

Tabla 4: Tabla de verdad Condicional (SI... ENTONCES)

De igual manera existe una proposición bicondicional que indica que una proposición es verdadera SI y SOLO SI las proposiciones simples o compuestas que forman la proposición son verdaderas o son falsas si y solo si ambas proposiciones son falsas, esta representa y se lee de la siguiente manera:

p ↔ q se lee "p si solo si q"

Ejemplo 2:

Es buen estudiante, si y solo sí; tiene promedio de cinco, donde se puede concluir lo siguiente:

p: es un buen estudiante

q: tiene promedio de 5

al aplicar el operador p ↔ q su tabla de verdad quedaría de la siguiente manera:

P	Q	P ↔ Q
V	V	V
V	F	F
F	V	F
F	F	V

O puede ser

P	Q	P ↔ Q
1	1	1
1	0	0
0	1	0
0	0	1

Cuando se aplican tablas de verdad se puede llegar a tres tipos de resultado que son:

- ✔ Tautología: es una expresión lógica que es verdadera para todos los posibles valores de verdad de sus componentes atómicos. Las proposiciones dan el resultado positivo (TOMi.digital, 2021).
- ✔ Contradicción: son aquellas fórmulas que son falsas para cualquier valoración de los símbolos proposicionales que contiene. Las proposiciones dan el resultado negativo (TOMi.digital, 2021).
- ✔ Contingencia: son aquellas fórmulas cuyo valor de verdad o falsedad depende de la valoración de los símbolos proposicionales que contiene. Las proposiciones dan un resultado negativo y positivo (TOMi.digital, 2021).

A continuación, se mostrará como aplicando lógica matemática podemos comprobar el valor de verdad de un enunciado, este tipo de práctica es muy útil en las computadoras, ya que permiten tener inferencia y deducir acciones o actividades a seguir.

Ejemplo 3:

P: todas las palomas vuelan
Q: todas las aves tienen plumas
R: luego todas las palomas son aves

Estas proposiciones se pueden escribir utilizando los operadores lógicos de la siguiente manera:

$(p \land q) \rightarrow r$

Para el anterior ejercicio se construye la tabla de verdad:

P	Q	R	P ∧ Q	→ R
V	V	V	V	V
V	V	F	V	F
V	F	V	F	V
V	F	F	F	V
F	V	V	F	V
F	V	F	F	V

F	F	V	F	V
F	F	F	F	V

2.3. Ejercicios de lógica proposicional

A continuación, se plantean a manera de ejemplo algunos ejercicios con su respectiva solución que permitirán identificar un problema, reconocer los datos de entrada y los procesos a ser aplicados como estrategia a seguir desde la utilización de la lógica proposicional en la identificación y solución de problemas:

2.3.1. Ejercicio 1:

Analizar el siguiente enunciado: “Se necesita personal femenino que sepa Microsoft Office o Gsuite.” Para dar solución al enunciado anterior vamos a realizar las siguientes actividades.

Pasos a seguir:

- ✔ Identificar el problema
- ✔ Descomponer el enunciado en proposiciones atómicas
- ✔ Evaluar los posibles casos derivados del enunciado anterior
- ✔ Determinar los casos donde se cumpla la condición y hacer la selección de las personas de acuerdo al requerimiento.

Solución:

- ✔ Identificar el problema: se trata de buscar o contratar personal femenino que sepa utilizar Microsoft Office o Google Suite.

- ✔ Descomponer en proposiciones atómicas del enunciado (cuando están separadas del contexto del problema se llaman proposiciones atómicas).
  - **o** p: es personal femenino
  - **o** q: sabe utilizar el Microsoft office
  - **o** r: sabe utilizar la suite de Google

- ✔ Determinar las relaciones lógicas de acuerdo a lo planteado en el enunciado utilizando las proposiciones atómicas definidas en el punto anterior
  - **o** $p \wedge (q \vee r)$

- ✔ Evaluar los posibles casos derivados del enunciado anterior

2.3.2. Ejercico 2:

Juan sabe utilizar Microsoft office y Gsuite, Martha sabe utilizar Gsuite, Natalia sabe utilizar Gsuite y Microsoft Office.

Crear la tabla de verdad y aplicar la lógica proposicional para elegir las personas que cumplan con los requisitos exigidos.

Para la construcción de la tabla de verdad vamos a evaluar cada una de las preposiciones atómicas e iremos validándolas para verificar si se cumple lo que la haría verdadera o si no se cumple lo cual haría que la proposición sea falsa.

Empezaremos a analizar cada posible caso por lo que evaluamos a Juan, para la primera proposición sería un valor falso, ya que p requiere que sea personal femenino el cual se pondrá en la columna 2, luego evaluamos si juan sabe utilizar Microsoft Office o la Suite de Google el valor ira en la columna tres y por último evaluamos la conjunción para determinar la solución.

Nombre	p	(q ∨ r)	p ∧ (q ∨ r)
Juan	F	V	F
Martha	V	V	V
Natalia	V	V	V

De acuerdo a la anterior tabla podemos concluir que Martha y Natalia son actas para el trabajo lo que permite seleccionarlas.

2.3.3. Ejemplo 3.

Enunciado: “Sandra dice Adelgazo o no hago dieta”

Para este enunciado además se proponen una seria de alternativas que podrían ser validas o no dependiendo de la solución propuesta utilizando la lógica matemática, las siguientes son las posibles respuestas de la cual se deberá elegir la que consideramos es la real al problema planteado.

- Si adelgazo, hago dieta
- Si no adelgazo, hago dieta
- Si hago dieta, no adelgazo
- Si hago dieta adelgazo
- Hago dieta o no adelgazo

Solución:

- ✔ Problema: identificar la respuesta correcta al problema presentado por Sandra
- ✔ Definir las proposiciones atómicas:

  p: Adelgazo
  q: Hago dieta

- ✔ Determinar la relación lógica

  p ∨ ¬ q

- ✔ Crear la tabla de verdad

p	(q)	(p ∨ ¬ q)
V	V	V
V	F	V
F	V	F
F	F	V

C. ACTIVIDADES DIDÁCTICAS (OPCIONALES SI SON SUGERIDAS)

DESCRIPCIÓN DE ACTIVIDAD DIDÁCTICA	
Nombre de la Actividad	Actividad didáctica 1. Organizar en el orden correcto las acciones a seguir para solucionar el problema
Objetivo de la actividad	Afianzar y aplicar los conceptos más importantes de la algoritmia a partir del desarrollo de un ejercicio práctico.
Tipo de actividad sugerida	Organice en el orden adecuado las actividades a desarrollar para dar solución al enunciado propuesto.
Archivo de la actividad (Anexo donde se describe la actividad propuesta)	Anexo documento en Word llamado Actividad didáctica 1

D. MATERIAL COMPLEMENTARIO:

Referencia APA del Material	**Tipo de material (Video, capítulo de libro, artículo, otro)**	**Enlace del Recurso o Archivo del documento o material**
Magic Markers. (21 Jul 2015). ¿Qué es un algoritmo?.	Vídeo	https://www.youtube.com/watch?v=U3CGMyjzlvM

https://www.youtube.com/watch?v=U3CGMyjzlvM		
Calderón, J. P. (2015, Apr. 30). Fundamentos de lógica matemática. Youtube. https://www.youtube.com/watch?v=9zYDaZhS7Ac	Vídeo	https://www.youtube.com/watch?v=9zYDaZhS7Ac

E. GLOSARIO:

TÉRMINO	SIGNIFICADO
ALGORITMO	Conjunto ordenado y finito de operaciones que permite hallar la solución de un problema.
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 (Asale, R.-, & Rae, 2021).
SOFTWARE	Conjunto de programas, instrucciones y reglas informáticas para ejecutar ciertas tareas en una computadora.
CONJUNCIÓN	Conecta dos proposiciones que se deben unir para que se pueda obtener un resultado verdadero
DISYUNCIÓN	Conecta dos proposiciones y se obtiene una proposición compuesta verdadera cuando por lo menos unas de las proposiciones son verdaderas
NEGACIÓN	Su propósito es negar una proposición es decir que si tenemos una proposición verdadera al aplicarle este tipo de operador se obtiene la negación es decir falso
PROPOSICIÓN BICONDICIONAL	Indica que una proposición es verdadera SI y SOLO SI las proposiciones simples o compuesta que forman la proposición son verdaderas o son falsas si y solo si ambas proposiciones son falsas
TAUTOLOGÍA	Es una expresión lógica que es verdadera para todos los posibles valores de verdad de sus componentes atómicos. Las proposiciones dan el resultado positivo (TOMi.digital, 2021).
CONTRADICCIÓN	Son aquellas fórmulas que son falsas para cualquier valoración de los símbolos proposicionales que contiene. Las proposiciones dan el resultado negativo (TOMi.digital, 2021).
CONTINGENCIA	Son aquellas fórmulas cuyo valor de verdad o falsedad depende de la valoración de los símbolos proposicionales que contiene. Las proposiciones dan un resultado negativo y positivo (TOMi.digital, 2021).

F. REFERENCIAS BIBLIOGRÁFICAS:

Asale, R.-, & Rae. (2021, Apr. 08). Algoritmo | Diccionario de la lengua española. https://dle.rae.es/algoritmo

Asale, R.-, & Rae. (2021, Apr. 08). Programa | Diccionario de la lengua española. https://dle.rae.es/programa
Hidalgo, M. I. M. (2018). Estrategias metodológicas para el desarrollo del pensamiento lógico-matemático. Didasc@ lía: Didáctica y Educación, 9(1), 125-132. https://dialnet.unirioja.es/descarga/articulo/6595073.pdf
López Juan Carlos, 2009, Algoritmos y programación (guía para docentes). https://libros.metabiblioteca.org/bitstream/001/169/8/AlgoritmosProgramacion.pdf
Resnick, Mitchel (2007): Sembrando las semillas para una sociedad más creativa. Laboratorio de medios de MIT, Massachussets [Consulta en línea: Eduteka, marzo 16, 2009, http://www.eduteka.org/ScratchResnickCreatividad.php .
Suppes, P., & Hill, S. (2021). Introducción a la lógica matemática. Reverté. https://www.reverte.com/libro/introduccion-a-la-logica-matematica_91496/
Tapia, L. (1995). Cómo desarrollar el razonamiento lógico matemático. Editorial Universitaria. https://www.worldcat.org/title/como-desarrollar-el-razonamiento-logico-y-matematico/oclc/50414752
TOMi.digital. (2021, Apr. 08). https://tomi.digital/es/59552/valores-de-las-tablas-de-verdad?utm_source=google&utm_medium=seo
TOMi.digital. (2021, Apr. 08). https://tomi.digital/es/59552/valores-de-las-tablas-de-verdad?utm_source=google&utm_medium=seo

G. CONTROL DEL DOCUMENTO

	Nombre	**Cargo**	**Dependencia** *(Para el SENA indicar Regional y Centro de Formación)*	**Fecha**
Autor (es)	Henry Eduardo Bastidas Paruma	Experto temático	Regional Cauca, Centro de teleinformática y producción industrial	Abril 2021
	Peter Emerson Pinchao Solis	Experto temático	Regional Cauca, Centro de teleinformática y producción industrial	Abril 2021
	Oscar Absalón Guevara	Diseñador instruccional	Regional Bogotá, Centro de Gestión Industrial	Abril 2021

H. CONTROL DE CAMBIOS

(Diligenciar únicamente si realiza ajustes a la Unidad Temática)

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

Taller de aplicación de conceptos y principios de programación orientada a objetos aplicados en lenguaje de programación

Realizar análisis y identificación correcta utilizando el paradigma de programación orientada a objetos requiere tanto del conocimiento de los conceptos de abstracción, encapsulamiento, modularidad, Jerarquía (Herencia), polimorfismo, clases y objetos, como de las características propias de este paradigma entre los cuales están métodos, atributos, asociaciones, composición y agregación entre otros.

Con el fin de fortalecer la aplicación de estos conceptos en situaciones que se pueden presentar en el contexto laboral, se plantean los siguientes problemas en los que se deberá identificar todas las características y conceptos de este paradigma aplicándolos en los casos pertinentes, se resaltaran las características del paradigma POO y diagramas sencillos de UML representando las clases, métodos, atributos… etc.

Se debe clasificarse leyendo uno a uno los problemas las clases, Métodos, atributos, identificado y explicando cada uno de los conceptos antes mencionados (abstracción, encapsulamiento, modularidad, Jerarquía (Herencia), polimorfismo).

1. Un lavadero de autos necesita registrar todos los autos que ingresan por el servicio, el lavadero también lava motocicletas, camionetas, automóviles y se debe registrar la placa, el color, el propietario, hora de llegada, hora de salida. Se deben registrar los servicios que se ofrecen en el lavadero, lavado, encerado, lavado de motor, lavado de cojineria. Si es un automóvil se ofrece lavado de motor, lavado de cojineria, lavado normal, encerado y lavado Normal. Si es una motocicleta solo se cobra lavado y encerada, el valor de estos servicios varia dependiendo del vehículo.

   Como podrías representar este problema y mapea en la herramienta UML seleccionada y argumenta por que.

2. Un zoológico quiere registrar sus animales, debe registrar los animales que vuelan, las serpientes y los felinos. Debe registrar fecha y hora de llegada, y la clasificación, si son africanos o si son andinos.

   Como podrías representar este problema y mapea en la herramienta UML seleccionada y argumenta por que.

3. Una empresa quiere registrar sus trabajadores para llevar el control del pago de nomina, la empresa es una empresa de desarrollo de software y contrata ingenieros por prestación de servicios y a termino indefinido, los cargos son: Jefe de desarrollo, desarrollador full stack, desarrolladro Junior, Desarrollador Master y tester y se paga la nomina de forma mensual y la hora pagada depende del cargo.

   Como podrías representar este problema y mapea en la herramienta UML seleccionada y argumenta por que.

4. Se necesita una aplicación que cree el árbol genealógico de una familia, debe registrar personas en donde se deben crear abuelos, padres e hijos, y debe calcular la edad de cada una de las personas a partir de la fecha de nacimiento, se debe registrar por persona nombres apellidos y la fecha de nacimiento.

   Como podrías representar este problema y mapea en la herramienta UML seleccionada y argumenta por que.

5. Una tienda que vende zapatos quiere registrar sus zapatos a la venta, los zapatos son de diferente clase, guayos, zapatillas para correr, zapatillas deportivas, zapato clásico, cada clase tiene diferentes características comunes, por ejemplo los guayos tiene taches, se debe registrar la entrada al almacén y la venta del zapato.

   Como podrías representar este problema y mapea en la herramienta UML seleccionada y argumenta por que.

TALLER DE APLICACIONES DE PROGRAMACIÓN ORIENTADA A OBJETOS (POO)

Realizar correctamente las aplicaciones en la programación orientada a objetos requiere tanto del conocimiento de los conceptos de abstracción, encapsulamiento, modularidad, Jerarquía (Herencia), polimorfismo, clases y objetos como de las características propias de este paradigma entre los cuales están métodos, atributos, asociaciones, composición y agregación entre otros.

Con el fin de fortalecer la aplicación de estos conceptos en situaciones que se pueden presentar en el contexto laboral, se plantean los siguientes problemas en los que se deberá completar la información solicitada y, en los casos pertinentes, se utilizará el IDE desarrollo, el lenguaje de programacion y la herramienta de modelado UML seleccionados.

Problema 1
Se necesita crear un método para calcular el número de personas que se encuentran viajando en un tren, en un momento determinado. Para ello, es necesario tener en cuenta que en POO se considera una *Persona* como un objeto que tiene propiedades tales como nombre, altura, peso, color de pelo, color de ojos, entre otros, y métodos determinados por la forma de hablar, mirar, andar, correr, parar, etcétera.
En este contexto, identifique y argumente:

1. Implementar la clase mediante el IDE de desarrollo y el lenguaje de programación seleccionado
2. ¿En qué parte de las clases se representa el concepto de abstracción? Argumente.

Problema 2
Crear una clase Persona con las siguientes características:

Atributos: nombre, fecha de nacimiento y edad, como
Métodos: registrar persona y calcular edad.

Aplicar los tres tipos de encapsulamiento en los atributos y en los métodos antes mencionados.

1. Implementar la clase utilizando el IDE de desarrollo y el lenguaje de programación seleccionado
2. Identificar en donde se cumple el concepto de encapsulamiento y explicar por qué.

Problema 3
Implementar una clase inmueble con los atributos que se presentan a continuación y, posteriormente, crear dos métodos que permitan arrendar y vender.
Atributos:

- Tipo de inmueble
- Ubicación
- Área de terreno
- Área construida

- Cantidad de habitaciones
- Cantidad de baños
- Años de antigüedad
- Precio

1. Implementar la clase utilizando el IDE de desarrollo y el lenguaje de programación seleccionado.
2. Identificar en este caso el concepto de abstracción y explicar por qué.

Problema 4
Se desea desarrollar una aplicación por capas Modelo-vista-Controlador, para lo cual:

1. Explicar cómo aplicar el concepto de Modularidad y por qué.

Problema 5
Teniendo en cuenta las características de la clase Persona del problema 4, generar el encapsulamiento con los métodos *get* y *set*.

1. Implementar la clase mediante el IDE de desarrollo y el lenguaje de programación seleccionado y agregar los métodos get y set, correspondientes.
2. Identificar en donde se cumple el concepto de encapsulamiento y argumentar.

Problema 6
Implementar una clase vehículo con los atributos que se mencionan a continuación y crear los métodos que permitan frenar, acelerar, apagar y encender.

Atributos:
- color
- Tipo Vehículo
- Placa
- Numero de Llantas

1**.** Implementar la clase utilizando el IDE de desarrollo seleccionado
2. Identificar el concepto de abstracción y explicar por qué.

Problema 7
Los automóviles presentan características y comportamientos particulares; dentro de las características se encuentran Marca, Modelo, Número de chasis, Peso llantas o cauchos, Puertas y Ventanas, y como comportamientos se mencionan acelerar, frenar y retroceder.
Con esta información:

1. Implemente la clase mediante el IDE de desarrollo y el lenguaje de programación seleccionado
2. ¿En qué parte de las clases se representa el concepto de abstracción? Argumente.

Problema 8
Se necesita una aplicación que registre animales domésticos con atributos comunes como el color, el peso, el tamaño y el sexo, relacionados con los siguientes métodos:

- Gatos, método cazar
- Perros, método ladrar
- Caballos, método cabalgar
- Gallinas, método colocar huevos
- Vacas, método pastar

Representar el concepto de jerarquía, utilizando el IDE de desarrollo y el lenguaje de programación seleccionado y explicar su implementación

Problema 9

Crear una clase llamada Cuenta que tendrá los siguientes atributos: titular (que es una persona) y cantidad (puede tener decimales). El titular será obligatorio y la cantidad es opcional. Construir los siguientes métodos para la clase y realizar el diagrama con sus relaciones respectivas:

- Un constructor, donde los datos pueden estar vacíos.
- Los setters y getters para cada uno de los atributos. El atributo no se puede modificar directamente, solo ingresando o retirando dinero.
- mostrar(): Muestra los datos de la cuenta.
- ingresar(cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.
- retirar(cantidad): se retira una cantidad a la cuenta. La cuenta puede estar en números rojos.

Implementar las clases utilizando el IDE de desarrollo y el lenguaje de programación seleccionado.

Ejercicio 10

Crear una clase llamada Persona. Sus atributos son: **nombre**, **edad** y **DNI**. Construir los siguientes métodos para la clase y realizar el diagrama con sus relaciones respectivas:

- Un constructor, donde los datos pueden estar vacíos.
- Los setters y getters para cada uno de los atributos. Hay que validar las entradas de datos.
- mostrar(): presenta los datos de la persona.
- esMayorDeEdad(): devuelve un valor lógico indicando si es mayor de edad.

Implementar las clases utilizando el IDE de desarrollo seleccionado.

TALLER DE APLICACIONES DE PROGRAMACIÓN ORIENTADA A OBJETOS POO

Ejercicio 1.

Vamos a desarrollar un sistema de gestión de coches que permita vender y alquilar coches a los clientes con las siguientes características: marca, modelo, color y matrícula.

1. Identificar los atributos, clases y métodos, subrayarlos y separar cada uno de ellos.

2. Utilizar una herramienta orientada a objetos y cree el diagrama de clases completo.

Ejercicio 2:

En POO, podemos considerar una *Persona* como un objeto que tiene propiedades (como nombre, altura, peso, color de pelo, color de ojos, etcétera) y métodos (como hablar, mirar, andar, correr, parar, etcétera).

Y una clase tren necesita tener el numero de personas que van en el por lo tanto se crea el método calcular la cantidad de personas que están en ese momento viajando, en que parte del problema identificamos el concepto de abstracción, implemente y explique.

1. Implementar la clase utilizando el IDE de desarrollo seleccionado e implementar las clases solicitadas en el problema anterior.

2. Identificar en donde se cumple con el concepto de abstracción y explica por que.

Ejercicio 3:

¿Qué características podemos abstraer de los automóviles? o ¿Qué características semejantes tienen todos los automóviles?

Características: Marca, Modelo, Número de chasis, Peso llantas o cauchos, Puertas, Ventanas... Comportamiento: Acelerar, Frenar, Retroceder…

1. Implementar la clase utilizando el IDE de desarrollo seleccionado e implementar las clases solicitadas en el problema anterior.

2. Identificar en donde se cumple con el concepto de abstracción y explica por que.

Ejercicio 4:

Implementar una clase inmueble con atributos como:

- Tipo de inmueble.
- Ubicación.

- Área de terreno.
- Área construida.
- Cantidad de habitaciones.
- Cantidad de baños.
- Años de antigüedad.
- Precio

crear métodos arrendar y vender.

1. Implementar la clase utilizando el IDE de desarrollo seleccionado e implementar las clases solicitadas en el problema anterior.

2. Identificar en donde se cumple con el concepto de abstracción y explica por que.

Ejercicio 5.

Implementar una clase vehículo con atributos como:

- color.
- Tipo Vehículo.
- Placa.
- Numero de Llantas.

Crear métodos que permita frenar, acelerar, apagar y encender.

1. Implementar la clase utilizando el IDE de desarrollo seleccionado e implementar las clases solicitadas en el problema anterior.

2. Identificar en donde se cumple con el concepto de abstracción y explica por que.

Ejercicio 6.

Crearemos una clase Persona.
Atributos: nombre, fecha de nacimiento y edad, como
Métodos: registrar persona y calcular edad.

Aplicar los tres tipos de encapsulamiento en los atributos y en los métodos antes mencionados.

1. Implementar la clase utilizando el IDE de desarrollo seleccionado e implementar las clases solicitadas en el problema anterior.

2. Identificar en donde se cumple con el concepto de encapsulamiento y explica por que.

Ejercicio 7.

Teniendo en cuenta la clase antes mencionada, generar encapsulamiento con los métodos get y set.

1. Implementar la clase utilizando el IDE de desarrollo seleccionado e implementar las clases solicitadas en el problema anterior utilizando métodos get y set.

2. Identificar en donde se cumple con el concepto de abstracción y explica por que.

Ejercicio 8.

Se desea desarrollar una aplicación Modelo vista Controlador, como aplicarías Modularidad y por que.

Ejercicio 9.

Se necesita una aplicación que registre animales domésticos como: gatos y sus métodos particular que es cazar, perros con su método ladrar, caballos con su método cabalgar, gallinas con su método colocar huevos y vacas con su método pastar, todos los animales tienen atributos comunes como: color, peso, tamaño y sexo, teniendo en cuenta este problema se debe representar la Jerarquía y su implementación explicando el por que de dicha implementación.

1. Implementar las clase utilizando el IDE de desarrollo seleccionado e implementar las clases solicitadas en el problema anterior.

Ejercicio 10.

Crea una clase llamada Cuenta que tendrá los siguientes atributos: titular (que es una persona) y cantidad (puede tener decimales). El titular será obligatorio y la cantidad es opcional. Construye los siguientes métodos para la clase y realizar el diagrama con sus relaciones respectivas:

- Un constructor, donde los datos pueden estar vacíos.
- Los setters y getters para cada uno de los atributos. El atributo no se puede modificar directamente, sólo ingresando o retirando dinero.
- mostrar(): Muestra los datos de la cuenta.
- ingresar(cantidad): se ingresa una cantidad a la cuenta, si la cantidad introducida es negativa, no se hará nada.
- retirar(cantidad): se retira una cantidad a la cuenta. La cuenta puede estar en números rojos.

1. Implementar las clase utilizando el IDE de desarrollo seleccionado e implementar las clases solicitadas en el problema anterior.

Ejercicio 11.

Vamos a crear una clase llamada Persona. Sus atributos son: **nombre**, **edad** y **DNI**. Construye los siguientes métodos para la clase y realizar el diagrama con sus relaciones respectivas:

- Un constructor, donde los datos pueden estar vacíos.
- Los setters y getters para cada uno de los atributos. Hay que validar las entradas de datos.
- mostrar(): Muestra los datos de la persona.
- esMayorDeEdad(): Devuelve un valor lógico indicando si es mayor de edad.

1. Implementar las clase utilizando el IDE de desarrollo seleccionado e implementar las clases solicitadas en el problema anterior.