Juegos en Python

Grado 9° Guía 5

TIC



Estudiantes

Apoya:

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}

Juegos en Python

Grado 9° Guía 5

Estudiantes

Apoya:

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}

MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Yeimi Carina Murcia Yela
Ministra TIC

Giovanny Andrés López Cabezas
Viceministro de Transformación Digital

Gloria Patricia Perdomo Rangel
Viceministra de Conectividad

William Alexander Sánchez Martínez
Director (e) de Apropiación de TIC

Lorena Paola Fortich Tulena
Jefe de la Oficina Asesora de Prensa

Equipo Comunicaciones DATIC
Manuel Alejandro Guerrero Aponte
Carlos Andrés Méndez Silva

Equipo Técnico
Cristhiam Fernando Jácome Jiménez
Miller Jimmy Alarcón
Ricardo Cañón Moreno

BRITISH COUNCIL COLOMBIA

Felipe Villar Stein
Director de país

Laura Barragán Montaña
Directora de programas de Educación, Inglés y Artes

Marianella Ortiz Montes
Jefe de Colegios

Óscar Daniel Barrios Díaz
Jefe de Implementación Colombia Programa

Equipo operativo
Juanita Camila Ruiz Díaz
Jesica Lubrano Lavadera
Dayra Maritza Paz Calderón
Saúl Fabián Torres
César Augusto Herrera Lozano
Myriam Nathalia Salinas Tarazona
Paula Álvarez Peña
Elizabeth Marín Giraldo

Equipo técnico
Ana Lorena Molina Castro
Alejandro Espinal Duque
Vanesa Abad Rendón
Raisa Marcela Ortiz Cardona
Juan Camilo Londoño Estrada

Edición y revisión técnica de versiones finales
Alejandro Espinal Duque
Ana Lorena Molina Castro
Heidy Esperanza Gordillo Bogota
Juan Camilo Londoño Estrada

Apoyo en la edición y revisión académica y técnica (versión 1)
Vanesa Abad Rendón
Raisa Marcela Ortiz Cardona

Consultora experta
Heidy Esperanza Gordillo Bogota

Edición
Juanita Camila Ruiz Díaz
Alexandra Ruiz Correa

BRITISH COMPUTER SOCIETY (BCS) – Consultoría internacional

Julia Adamson
Directora General de Educación y Beneficio Público

Niel McLean
Jefe de Educación

Claire Williams
Coordinadora de Alianzas

ASOCIACIÓN COLOMBIANA DE FACULTADES DE INGENIERÍA - ACOFI

Edición general
Mauricio Duque Escobar

Coordinación pedagógica
Margarita Gómez Sarmiento
Mariana Arboleda Flórez
Rafael Amador Rodríguez

Coordinación de producción
Harry Luque Camargo

Asesoría estrategia equidad
Paola González Valcárcel

Asesoría primera infancia
Juana Carrizosa Umaña

Autoría
Alex González Jurado
Arlet Orozco Marbello
Harry Luque Camargo
Isabella Estrada Reyes
Lucio Chávez Mariño
Margarita Gómez Sarmiento
Mariana Arboleda Flórez
Mauricio Duque Escobar
Paola González Valcárcel
Rafael Amador Rodríguez
Rocío Cardona Gómez
Saray Piñerez Zambrano
Yimzay Molina Ramo

.Puntoaparte Editores

Diseño, diagramación, ilustración, y revisión de estilo

CC BY-NC 4.0

Esta obra se encuentra bajo una Licencia Creative Commons Atribución-No Comercial 4.0 Internacional. https://creativecommons.org/licenses/by-nc/4.0/

"La presente guía corresponde a una versión modificada del material originalmente diseñado por el equipo de autores de ACOFI, en tanto incorpora ajustes, actualizaciones y contribuciones posteriores realizadas por el equipo del British Council y el Ministerio de Tecnologías de la Información y las Comunicaciones, bajo la autorización correspondiente, para su publicación final".

ISBN: 978-628-7935-23-5

Prólogo

Ministerio de Tecnologías de la Información y las Comunicaciones

Estimadas y estimados educadores, estudiantes y comunidad educativa:

Colombia es un país de talento y de oportunidades que no siempre han estado al alcance de todas y todos. Entre ellas, la tecnología, que en las zonas más apartadas aún se percibe como un privilegio lejano.

Con la convicción de conectar territorios y cerrar brechas, emprendimos una transformación en el Ministerio TIC para acercarlo más a las regiones y a sus realidades.

Creemos en una tecnología que llegue con propósito a la vida de las personas, que permita cerrar brechas, abrir caminos y acompañar el desarrollo de las comunidades en todo el país.

Hoy, en ese camino, en el que ya hemos formado a cerca de un millón de colombianas y colombianos en habilidades digitales, presentamos las **Guías de Pensamiento Computacional de Colombia Programa**, una apuesta concreta para llevar estas oportunidades a las aulas del país.

Este material es el resultado de un trabajo colectivo en el que más de 2.000 docentes de 420 sedes educativas aportaron su experiencia y conocimiento para su construcción.

Además, contó con la asesoría de la *British Computer Society (BCS)*, entidad del Reino Unido con más de 20 años de experiencia en la incorporación del pensamiento computacional en el currículo educativo, gracias al convenio de cooperación internacional suscrito con el British Council Colombia.

Estas guías trazan una ruta pedagógica para que niñas, niños y jóvenes desarrollen nuevas formas de pensar, aprendan a resolver problemas y construyan soluciones. Son un recurso pensado para acompañar la labor de los y las docentes que enseñan pensamiento computacional.

Su uso requiere la mirada y la adaptación de quienes enseñan, teniendo en cuenta el contexto y las particularidades de cada estudiante.

Con ellas buscamos que las nuevas generaciones pasen de ser usuarias de la tecnología a protagonistas de su creación y transformación; que puedan comprenderla, dominarla y ponerla al servicio de sus propias vidas y de sus comunidades.

Queremos que, sin importar el lugar donde se estudie, en una zona rural, en la Amazonía o en una ciudad, existan las mismas oportunidades para desarrollar talento digital.

El gobierno del presidente Petro le cumplió a los colombianos y las colombianas y, por eso, buscamos que más niñas y jóvenes se reconozcan como creadoras, y que cada docente cuente con herramientas para transformar su aula.

Porque cuando conectamos territorios, abrimos oportunidades para que Colombia avance con equidad, desarrollo y justicia social.

Invitamos a estudiantes, docentes y comunidades a apropiarse de estas guías, a disfrutarlas y a llevarlas a su vida diaria. Porque cuando la educación digital llega a los territorios, no solo transforma el aprendizaje: transforma comunidades enteras y abre caminos para construir un mejor país.

Carina Murcia Yela

Carina Murcia Yela
Ministra de Tecnologías de la Información y las Comunicaciones
Gobierno de Colombia

Prólogo

British Council

Por más de 87 años, el *British Council* ha tenido presencia en el país, buscando construir lazos de confianza y cooperación entre el Reino Unido y Colombia. Uno de los mejores ejemplos de lo que podemos lograr a través de estas alianzas es *Colombia Programa*. Esta iniciativa, que desde hace tres años y de la mano con el Ministerio de Tecnologías de la Información y las Comunicaciones nos ha permitido alcanzar un objetivo ambicioso: ofrecer espacios de formación y recursos de alta calidad para que docentes y estudiantes de los colegios públicos del país desarrollen sus habilidades digitales y el pensamiento computacional con un enfoque de género.

Esta publicación marca un hito particularmente importante, pues con ella entregamos al país un material sin precedentes en América Latina y comparable con muy pocos a nivel mundial. Se trata de 79 guías pedagógicas que suman más de 4.000 páginas y contienen 30 sesiones por cada grado, desde transición hasta undécimo. Con estas, maestras y maestros podrán facilitar al menos una hora de clase semanal orientada al desarrollo del pensamiento computacional de sus estudiantes. Basta preguntarse qué otra área o asignatura en Colombia cuenta con un material semejante —en términos de amplitud, rigurosidad y acceso gratuito— para dimensionar la relevancia de este esfuerzo para el sector educativo y tecnológico del país.

Al abrir estas páginas impresas o navegar su versión digital, encontrarán una gran variedad de actividades que proponen a niños, niñas y jóvenes nuevas formas de acercarse a los temas más relevantes del presente, como el uso de la inteligencia artificial desde un enfoque ético y seguro. También verán cómo esta y otras temáticas se articulan con diversas áreas del conocimiento, promoviendo el trabajo colaborativo entre docentes y estudiantes y convirtiendo el pensamiento computacional en un puente entre las matemáticas, las ciencias, el arte y el lenguaje.

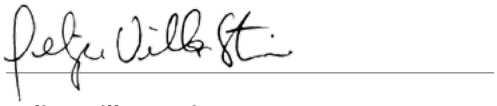
Encontrarán también en estas páginas orientaciones y sugerencias sobre cómo adaptar las actividades a diferentes contextos y necesidades. Por ejemplo, cómo desarrollar habilidades de pensamiento computacional en instituciones sin conectividad o incluso sin electricidad; cómo fomentar la equidad de género para que más niñas y jóvenes se interesen por la programación y las áreas STEM; o cómo ajustar ciertas actividades para que estudiantes en condición de discapacidad puedan participar plenamente.

Lo que quizá no sea visible a primera vista es el arduo proceso que hizo posible llegar a este punto. Un equipo de 25 personas se dedicó durante más de un año a diseñar y revisar cada una de las 79 guías. Contamos, además, con la asesoría de la *British Computer Society (BCS)*, a través de expertos que participaron y lideraron reformas curriculares en Inglaterra en el área de ciencias de la computación. Asimismo, realizamos 38 mesas de validación en cinco regiones del país, en las que 1.755 docentes revisaron y comentaron los primeros borradores. A esto se suman las innumerables horas que más de 2.100 docentes "pares expertos", pertenecientes a los 420 Nodos de Desarrollo de Pensamiento Computacional en todo el país, dedicaron en 2025 a pilotear las guías en sus aulas, asegurando así su pertinencia para los contextos educativos reales.

A cada uno de estos maestros y maestras, directivas docentes, mentores(as), tutores, expertos(as), académicos(as) y funcionarios(as) del *British Council* y de los Ministerios de las TIC y de Educación, así como a los y las estudiantes que nos acompañaron en este proceso, infinitas **gracias** por permitir que hoy Colombia cuente con esta herramienta tan poderosa.

Desde el *British Council*, sabemos que estas guías de pensamiento computacional abrirán nuevas posibilidades para que, a través del trabajo y esfuerzo de docentes en el aula, los y las estudiantes accedan a nuevas formas de pensar, de interactuar con la tecnología y de desarrollar capacidades alineadas con las demandas del mercado laboral presente y futuro. De lograrlo, conseguiremos el sueño que ha guiado esta labor y la alianza de *Colombia Programa*: que cada niña, niño y joven en Colombia pueda construir el futuro digital y material que desee.

Esperamos que las disfruten y que juntos podamos aprovechar al máximo su potencial.



Felipe Villar Stein
Director
British Council Colombia

El camino recorrido para llegar a estas guías

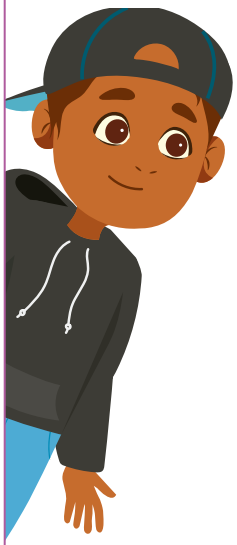
Creación

1 **8 de noviembre de 2023**
El Ministerio TIC y el *British Council* firman el convenio 1247 que da inicio a Colombia Programa.

Diciembre de 2023
La Asociación Colombiana de Facultades de Ingeniería (ACOFI) empieza a trabajar en la matriz de alcance y secuencia.

2 **Febrero de 2024**
Se aprueba la matriz de alcance y secuencia de las guías creadas por ACOFI, que incorpora retroalimentación de la *British Computer Society*. Con esta aprobación se inicia la creación de las guías de pensamiento computacional de Colombia Programa.

Validación y pilotaje



3 **Mayo a octubre 2024**
1.755 docentes, en 38 mesas de trabajo de las 5 regiones del país, validan la versión inicial de las guías y proponen ajustes para hacerlas más adaptables e inclusivas.

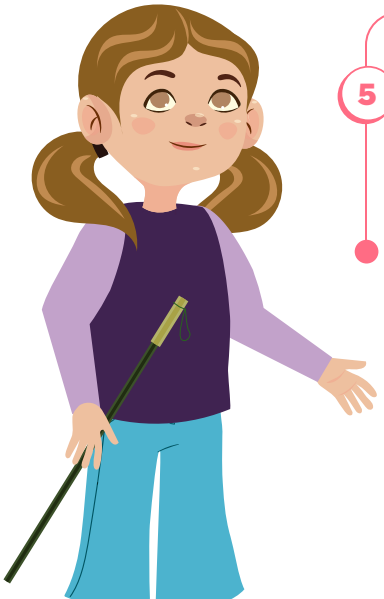
Mayo a diciembre 2024
Se incorporan a las guías los ajustes provenientes de las mesas de validación.

4 **Agosto 2024 a mayo 2025**
Diagramación e impresión de las guías para entrega a los nodos de pensamiento computacional.

Marzo a septiembre 2025
2.100 docentes de los 420 nodos prueban las guías en sus aulas. Se recopilan sugerencias de ajustes a partir de este ejercicio.

Octubre a noviembre 2025
Análisis de resultados y recomendaciones para ajustes de la versión final de las guías.

Ajustes finales



5 **Diciembre 2025 a abril 2026**
Implementación de ajustes finales.

Enero a junio 2026
Diagramación de la versión final de las guías de pensamiento computacional de Colombia Programa.



Grado 9º Guía 5

TIC

Identificación de los aprendizajes

Lógica, programación y depuración

Aprendizajes de la guía

Con las actividades de esta guía se espera que progresen en:

- Usar estructuras de condicionales múltiples (instrucciones si, si no si, si no), y diferentes tipos de ciclos para controlar el flujo de ejecución de un programa en *Python*.

Resumen de la guía

Esta guía está diseñada para presentar conceptos un poco más avanzados de programación en *Python*, a lo largo de 5 sesiones. Comienza con el aprendizaje del ciclo "para" y su implementación en *Python*, pasando luego a trabajar con variables tipo *string* y operaciones básicas sobre ellas. La guía también aborda la iteración sobre *strings* y el uso del operador "*in*".

En las últimas sesiones, aprendes sobre la función *randint* para generar números aleatorios y su aplicación en ejercicios prácticos. La guía también presenta las reglas de sintaxis en programación, enseñándoles a identificar y corregir errores comunes.

Resumen de las sesiones

Sesión 1

Introducción al ciclo "*para*" en *Python*. Aprenderás a crear y utilizar este ciclo, diferenciándolo del ciclo "*mientras que*". El enfoque está en comprender la estructura y aplicación del ciclo "*para*" para tareas repetitivas en programación.

Sesión 2

Se enfoca en variables tipo *string* y operaciones básicas en *Python*. Aprenderás a manejar y manipular texto en programación, incluyendo la concatenación y formateo de cadenas (*string*).

Aprendizajes de la guía

- Utilizar funciones e importar módulos para expandir las funcionalidades de un lenguaje basado en texto.
- Programar soluciones computacionales usando un lenguaje basado en texto.

Si se requiere

En la guía 4 de este grado se presenta una introducción a la programación en *Python*.

Sesión 3

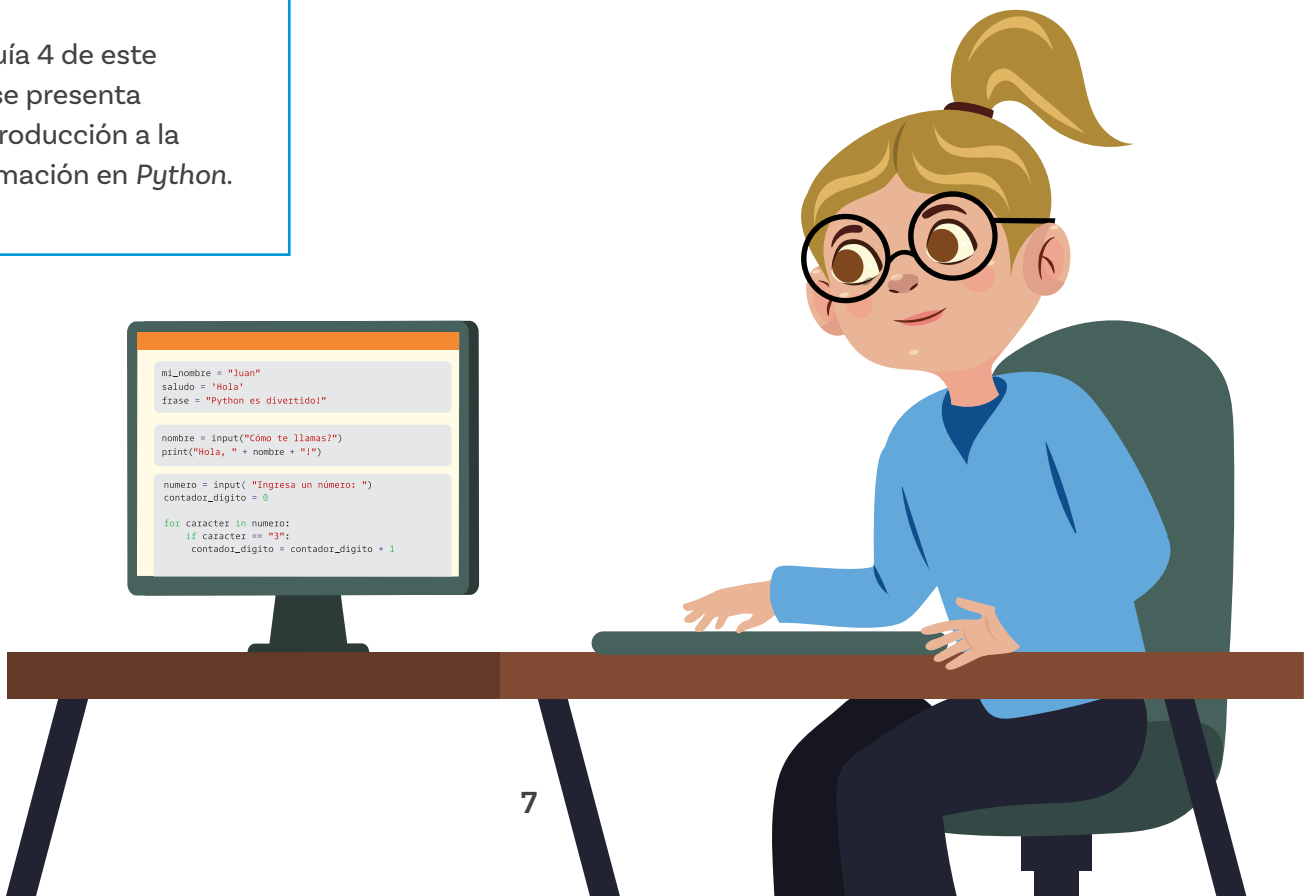
Aprendizaje sobre iteración de *string* y uso del operador "*in*". Esta sesión se centra en técnicas para recorrer y buscar dentro de *string*, así como verificar la pertenencia de elementos en secuencias.

Sesión 4

Enfoque en reglas de sintaxis y corrección de errores en programación. Esta sesión busca desarrollar habilidades para identificar, prevenir y corregir errores comunes de sintaxis, mejorando la calidad y robustez del código.

Sesión 5

Aprendizaje sobre la generación de números aleatorios a través de la función *randint*. Aprenderás a incorporar elementos de aleatoriedad en tus programas y solucionarás el reto integrando lo aprendido hasta el momento.

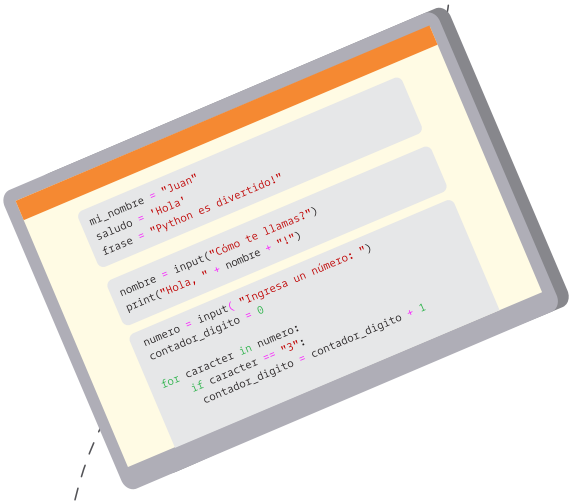


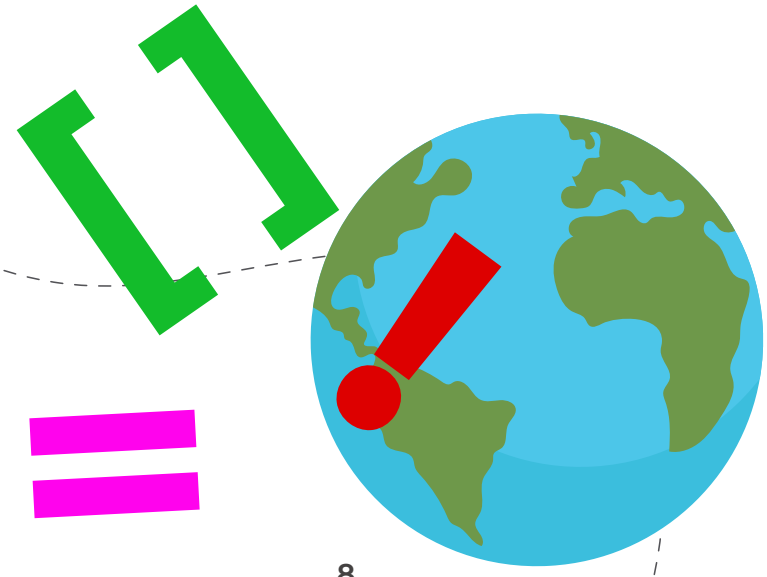
Conexión con otras áreas

Esta guía apoya el desarrollo de habilidades técnicas necesarias en la programación, a través de actividades prácticas que fomentan el desarrollo de la lógica computacional y la resolución de problemas. A continuación, se menciona la conexión de esta guía con otra área de aprendizaje:

Matemáticas

- Se desarrollan actividades que involucran los números, sus propiedades y operaciones matemáticas, así como el uso de números aleatorios.





Grado 9° Guía 5

TIC

Sesión 1

Aprendizajes esperados

Con las actividades de esta sesión se espera que puedas avanzar en:

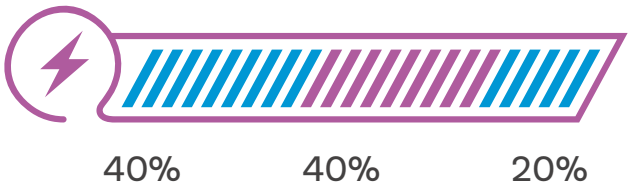
- Comprender la estructura del ciclo “para” en pseudocódigo.
- Diferenciar entre el ciclo “para” y el ciclo “mientras que”.
- Crear códigos simples utilizando el ciclo “para”.
- Implementar el ciclo “para” en *Python*.

Material para la sesión

Material por grupo:

- Anexo 1.1.
- Dispositivo con acceso a Internet.

Duración sugerida

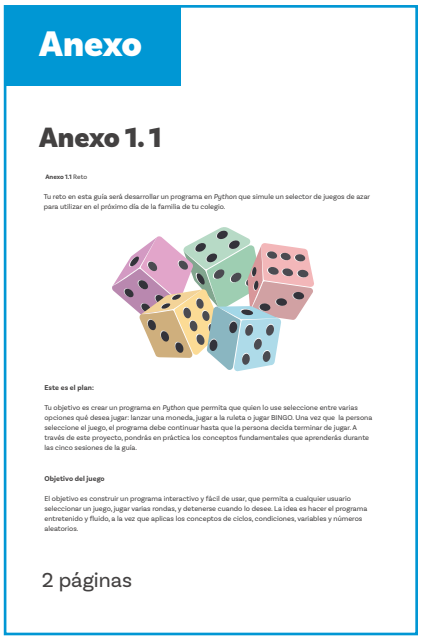


40% 40% 20%



Anexo

Anexo 1.1



Anexo 1.1 Reto

Tu reto en esta guía será desarrollar un programa en *Python* que simule un selector de juegos de azar para utilizar en el próximo día de la familia de tu colegio.

Este es el plan:

Tu objetivo es crear un programa en *Python* que permita que quien lo use seleccione entre varias opciones qué desea jugar: lanzar una moneda, jugar a la ruleta o jugar BINGO. Una vez que la persona seleccione el juego, el programa debe continuar hasta que la persona decida terminar de jugar. A través de este proyecto, pondrás en práctica los conceptos fundamentales que aprenderás durante las cinco sesiones de la guía.

Objetivo del juego

El objetivo es construir un programa interactivo y fácil de usar, que permita a cualquier usuario seleccionar un juego, jugar varias rondas, y detenerse cuando lo desee. La idea es hacer el programa entretenido y fluido, a la vez que aplicas los conceptos de ciclos, condiciones, variables y números aleatorios.

2 páginas

Lo que sabemos, lo que debemos saber

Esta sección corresponde al 40% de avance de la sesión

Empieza por leer el ejercicio que deberás resolver al final de esta guía, el cual se encuentra en el *Anexo 1.1*.

En la guía anterior aprendiste qué es un ciclo "mientras que" y para qué sirve. Además de este ciclo, también existe el ciclo para, que es igualmente fundamental en la programación. Ambos ciclos son herramientas poderosas que permiten ejecutar un bloque de código repetidamente, pero se utilizan en diferentes contextos y con distintas estructuras.

El ciclo para es una estructura de control repetitiva que se utiliza para ejecutar un bloque de código un número específico de veces. A diferencia del ciclo mientras que, donde la repetición depende de una condición, en el ciclo para se conoce de antemano cuántas veces se repetirá el bloque de código.

Veamos cómo es la estructura del ciclo para en pseudocódigo:




```
Para (contador) desde (inicio) hasta (final) con paso (incremento) hacer:
    Bloque de código a repetir
Fin Para
```

En esta estructura se utilizan una variable y 3 valores.

- **Contador**: es una variable que controla el ciclo. Cada vez que se repite el ciclo, esta variable se incrementa automáticamente al sumar el valor de paso.
- **Inicio**: es el valor inicial del contador. Es el número desde el cual comenzará la repetición.
- **Final**: es el valor final del contador. El ciclo se detendrá cuando el contador alcance este valor.
- **Incremento:** el valor que se suma (o resta, si es negativo) a la variable de contador en cada iteración.

Por ejemplo, si queremos contar de uno en uno, del 1 hasta el 4, podríamos utilizar el siguiente ciclo:

```
Para i desde 1 hasta 5 con paso 1 hacer
        Escribir i
Fin Para
```

Observa los siguientes fragmentos de código y responde a las preguntas.

Figura 1. Range (final)

```
for i in range(5):
    print(i)
```

¿Qué crees que imprimirá el programa cuando se ejecute?
Escribe el código en tu computador y ejecútalo
¿El resultado concuerda con tu predicción?

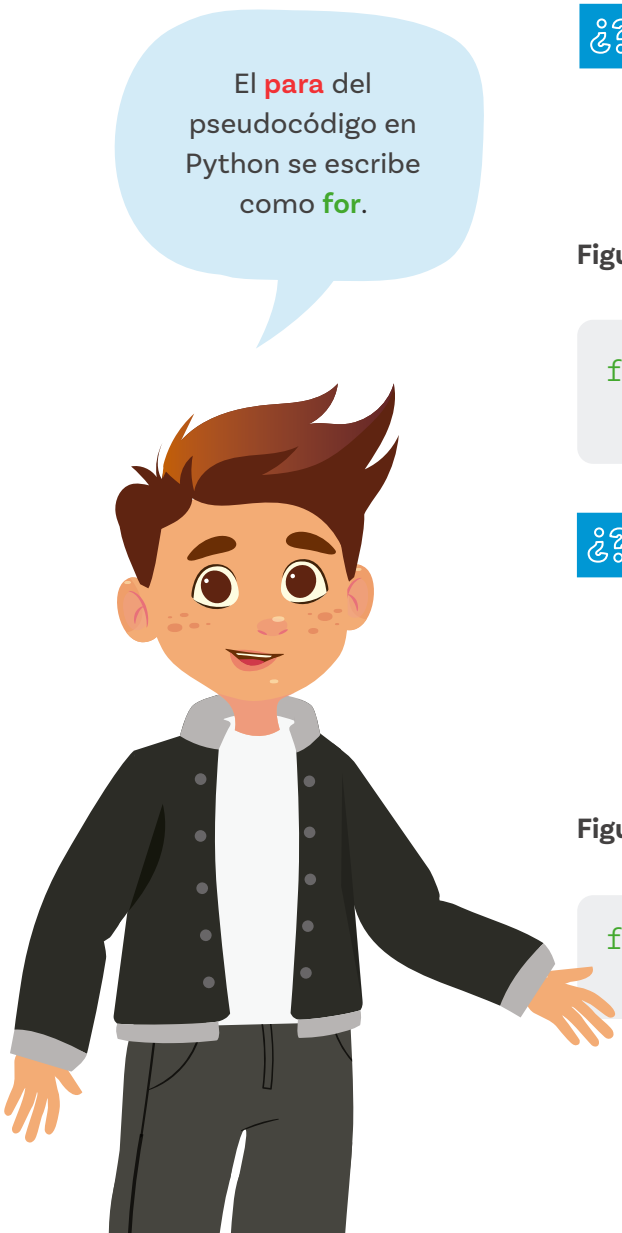


Figura 2. Range (inicio, final)

```
for i in range(1, 6):
    print(i)
```

Escribe el código en tu computador y ejecútalo.
¿El resultado es el mismo que en el programa anterior?
¿Por qué crees que pasa esto?
Cambia el 6 por otro número. ¿Qué ocurre con el resultado?

Figura 3. Range (inicio, final, paso)

```
for i in range(1, 10, 2):
    print(i)
```

Nota

La función range en *Python* no incluye el número final. Es decir, si imprimes todos los números del rango range (5) verás los números 0, 1, 2, 3, 4.

¿Qué diferencia tiene el programa con los anteriores?
¿Tiene similitudes?
¿Qué crees que ocurra si cambias el 2 por 3?

En este ejemplo:

- **i** es la variable de control del ciclo.
- El ciclo empieza con i igual a 1 y se incrementa en 2 después de cada iteración.
- El ciclo se ejecuta hasta que i alcanza el valor de 9.

En cada iteración se escribirá el valor que tenga i en ese momento: 1, 3, 5, 7, 9.

El bloque de código dentro del ciclo se ejecutará 5 veces.

Ahora vas a aprender cómo se usa el ciclo "para" en *Python*. En lugar de palabras clave como desde y hasta, *Python* utiliza la función range() para definir el rango de valores.

La función range() puede ser utilizada de tres formas diferentes:

- range(final): genera una secuencia de números desde 0 hasta 4. Ver *Figura 1*.

- range(inicio, final): genera una secuencia de números desde 1 hasta 5. Ver *Figura 2*.

- range(inicio, final, paso): genera una secuencia de números desde inicio (1) hasta final (10), incrementando los números por paso (2). Ver *Figura 3*.

Es crucial recordar que, en *Python*, la indentación y el uso de dos puntos (:) son fundamentales para definir el bloque de código que pertenece al ciclo. Cada línea de código dentro del ciclo debe tener sangría para que *Python* sepa que forma parte del ciclo para.

Glosario

Indentación: en programación, es el espacio en blanco al comienzo de una línea de código, que indica la estructura o jerarquía del programa. En *Python*, la indentación es esencial para definir bloques de código, como en ciclos o funciones. La correcta indentación permite que el código sea legible y funcione adecuadamente, mientras que la falta de ella puede causar errores.

Ejemplo:

```
Sin indentación
    Con indentación
```

Manos a la obra

Conectadas



Esta sección corresponde al 80% de avance de la sesión

Para resolver los siguientes ejercicios en *Python* te proponemos trabajar en grupos de 2 o máximo 3 personas, pide indicaciones a tu docente para la conformación del grupo.

En algunos ejercicios es posible que necesiten utilizar condicionales. Al finalizar, pueden comparar sus códigos con los de los demás grupos.

- Juan quiere ayudar a su hermano Martín a contar hasta 20. Escriban un programa que imprima los números del 1 al 20 para que Martín pueda practicar.

- Daniela quiere mostrarle a su sobrino Diego cómo sumar números. Creen un programa que calcule la suma del 1 al 50 para que puedan ver el resultado juntos.

- Emma quiere aprender la tabla de multiplicar de su número favorito. Ayuden a su prima Clara a hacer un programa que imprima la tabla de ese número del 1 al 10.

- Lucas quiere enseñarle a Camila los números pares. Hagan un programa que imprima todos los números pares del 1 al 50 para que ella pueda reconocerlos fácilmente.

Si les queda tiempo proponemos estos retos adicionales:

- **Factorial de un número:**
  Creen un código que calcule el factorial de un número dado por el usuario.
  Tip: recuerden que el factorial de un número positivo (n) se calcula multiplicando todos los números desde 1 hasta n. Por ejemplo, el factorial de 4 se calcula así: 1 × 2 × 3 × 4 = 24.

- **Números divisibles por 3 y 5:**
  creen un código que imprima los números del 1 al 100 que sean divisibles por 3 y por 5.
  Tip: Usen la condición n % 3 == 0 and n % 5 == 0 para saber si un número n es divisible por ambos.

Python se puede utilizar con herramientas de accesibilidad, como lectores de pantalla y software de reconocimiento de voz, en caso de ser necesario.



Antes de irnos

Esta sección corresponde al 100% de avance de la sesión

Revisa los aprendizajes de la sesión ¿Crees que lograste alcanzarlos?

1. ¿Puedes comprender la estructura del ciclo “para” en pseudocódigo?
   - Sí
   - Parcialmente
   - Aún no
2. ¿Puedes diferenciar entre el ciclo “para” y el ciclo “mientras que”?
   - Sí
   - Parcialmente
   - Aún no

3 ¿Puedes crear códigos simples utilizando el ciclo “para”?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

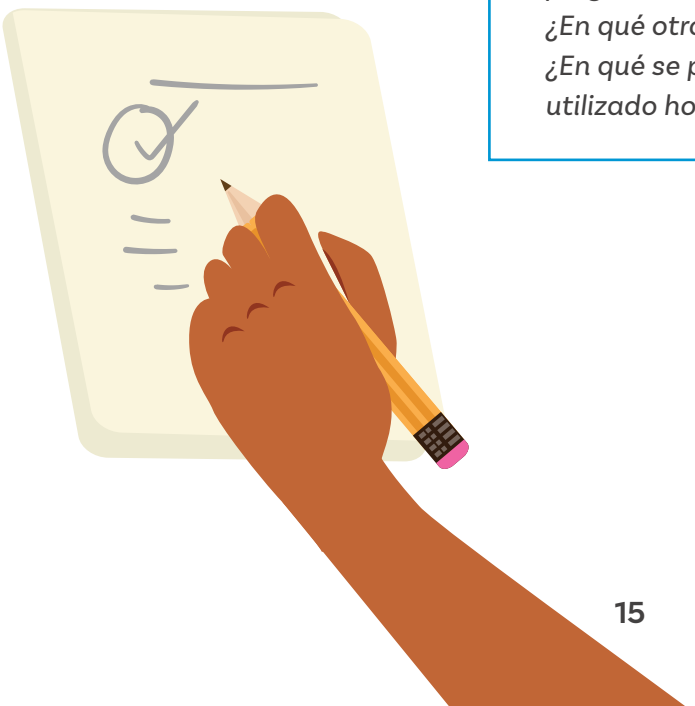
4 ¿Puedes implementar el ciclo “para” en *Python*?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

Si tus respuestas a las preguntas anteriores fueron “Parcialmente” o “Aún no”, realiza uno de los ejercicios adicionales que encuentras en la sección “ Manos a la obra ”, en compañía de tu docente o de un compañero o compañera que haya realizado todos los ejercicios. Pídele explicar en voz alta cómo están resolviendo cada paso de este.

Una vez resueltas estas dudas y antes de cerrar la sesión, responde las siguientes preguntas y comenta con otras compañeras y compañeros.

¿Qué diferencias y semejanzas encuentras entre el ciclo “para” y el ciclo “mientras que” en la programación?
¿En qué otros contextos has utilizado los rangos?
¿En qué se parecen y se diferencian de lo que has utilizado hoy?



Sesión 2

Aprendizajes esperados

Con las actividades de esta sesión se espera que puedas avanzar en:

- Reconocer qué son las variables tipo *string*.
- Comprender las operaciones básicas con cadenas de texto en *Python*.

Duración sugerida

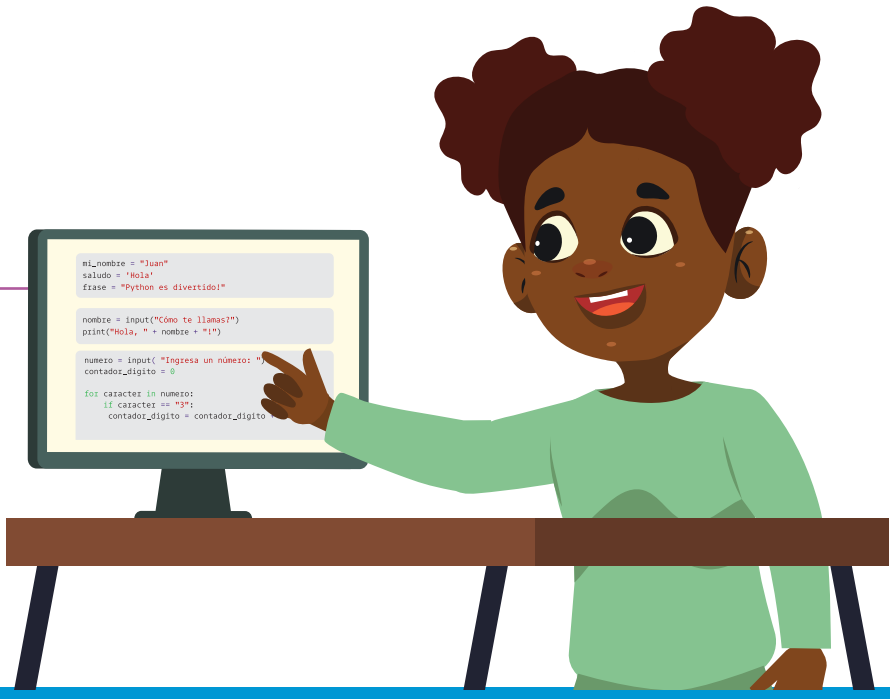


15% 70% 15%

Material para la sesión

Material por grupo:

- ○ Dispositivo con acceso a Internet.



Lo que sabemos, lo que debemos saber



Esta sección corresponde al 15% de avance de la sesión

En una guía anterior aprendiste sobre las variables numéricas y booleanas. Es hora de continuar con estos aprendizajes.

Recuerda que las variables numéricas almacenan valores como enteros y decimales, mientras las variables booleanas almacenan valores de verdadero (True) o falso (False).

También aprendiste a guardar texto en una variable, la cual se conoce como string (o **cadena**). Una variable tipo string es una secuencia de caracteres, como letras, números y símbolos.

En *Python*, las cadenas se crean poniendo el texto entre comillas simples (' ') o comillas dobles (" "). Algunos ejemplos son:

```
mi_nombre = "Juan"
saludo = 'Hola'
frase = "Python es divertido!"
```

¿Qué crees que ocurra si usas comillas simples y dobles a la vez? ¿Y si no usas comillas?

Podemos utilizar la función input() para leer información proporcionada por el usuario y almacenarla en una variable tipo *string*. Por ejemplo:

```
nombre = input("¿Cómo te llamas?")
print("Hola, " + nombre + "!")
```

¿Qué hace el signo más (+) usado en la línea 2? Quita las comillas antes y después de hola y nota qué sucede. Vuelve a agregarlas y compara la diferencia.

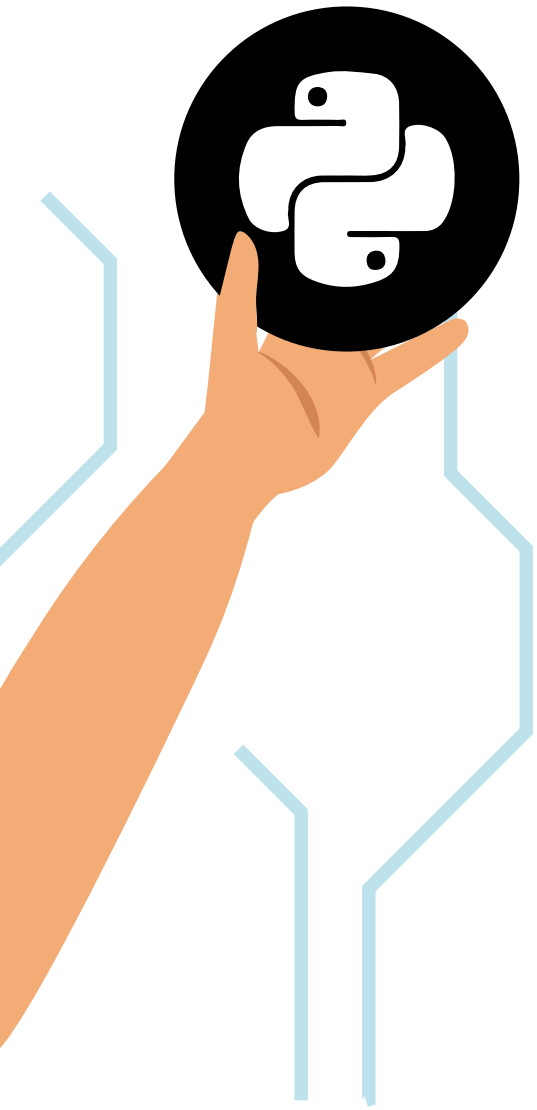
Anotación

El símbolo # se utiliza para colocar un comentario explicando algo en el código.

A diferencia de las variables numéricas, que se utilizan para realizar operaciones matemáticas, las variables tipo string se utilizan para manipular texto. Podemos realizar operaciones aritméticas como suma, resta, multiplicación y división con variables numéricas, pero con variables tipo string realizamos operaciones como **concatenación** y **repetición**.

La **concatenación** une dos o más variables de tipo string en una sola variable, para ello utilizamos el signo más (+), por ejemplo:

```
nombre = "Juan"
apellido = "Pérez"
nombre_completo = nombre + " " + apellido
print(nombre_completo) #salida: Juan Pérez
```



La **repetición** duplica un string un número específico de veces utilizando el asterisco (*), por ejemplo:

```
frase = "Hola!"
repeticion = frase * 3
print(repeticion) #salida: Hola! Hola! Hola!
```

También podemos saber la longitud que tiene una cadena, es decir, la cantidad de caracteres que contiene, para esto utilizamos la función len() :

```
mensaje = "Hola, Mundo!"
longitud = len(mensaje)
print(longitud) #salida: 12
```

Es importante que tengas en cuenta que los espacios en blanco también se consideran caracteres.

Anotación

Si una variable tiene varios caracteres, a cada uno de ellos se accede usando un índice. Ver este ejemplo:

```
nombre = "PRUEBA"
print(nombre[1])
```

R

Anotación

La función len() permite obtener el número de caracteres en una variable tipo string.

Además de estas operaciones, podemos acceder a caracteres individuales de un *string* utilizando **índices** (empezando desde 0):

```
mensaje = "Hola, Mundo!"
primer_caracter = mensaje[0]
print(primer_caracter)
```

¿Qué crees que imprimirá este código? ¿Y si el índice ahora es 2 en vez de 0?

Compara tus respuestas con tus compañeros.

Glosario

Cadena: tipo de variable donde se almacenan caracteres, que pueden ser letras, números y símbolos. En este caso, los números tienen un tratamiento como si fueran "letras", no se pueden hacer operaciones aritméticas directamente sobre números expresados como caracteres.

String: nombre que se les da a las variables tipo cadena en *Python*.

Concatenación: operación que consiste en unir o enlazar dos o más cadenas de texto *(strings)* para formar una nueva cadena.

Índice: posición de un caracter dentro de una cadena. En *Python* comienzan en 0 para el primer caracter, 1 para el segundo, y así sucesivamente.

Manos a la obra

Conectadas



Esta sección corresponde al 85% de avance de la sesión

Para realizar los siguientes ejercicios deberás trabajar en grupo, según la indicación de tu docente.

Modifiquen los programas que desarrollaron al inicio de la sesión para resolver al menos 3 de los siguientes ejercicios propuestos. En algunos ejercicios es necesario que utilicen condicionales y la función len(). Luego, comparen los códigos con los de otros grupos.

- Pidan a las y los usuarios que ingresen su nombre y apellido por separado. Combinen ambos en una sola variable y muestren el nombre completo.
- Pidan a las y los usuarios que ingresen una frase y un número. Impriman la frase repetida ese número de veces.
- Pidan a las y los usuarios que ingresen una palabra. Muestren la longitud de la palabra.
- Pidan a las y los usuarios que ingresen una palabra. Muestren el primer y el último caracter de la palabra.
- Pidan a las y los usuarios que ingresen una frase y verifiquen si la longitud de la frase es mayor a 10 caracteres.
- Pidan a las y los usuarios que ingresen dos palabras y luego comparen sus longitudes, mostrando cuál es más larga o si tienen la misma longitud.

Si aún tienen tiempo pueden abordar el siguiente reto:

Imaginen que trabajan en una tienda y necesitan desarrollar un sistema para registrar el pedido de productos de una clienta llamada Laura. Laura indicará cuántos productos desea ordenar (con un máximo de X productos diferentes) y, para cada producto, especificará el nombre y la cantidad que desea. Su tarea es concatenar esta información en un string que describa el pedido completo de Laura.

Antes de irnos



Esta sección corresponde al 100% de avance de la sesión

De manera individual revisa los aprendizajes de la sesión y reflexiona sobre el grado al que los alcanzaste.

1. ¿Puedes reconocer qué son las variables tipo *string*?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no
2. ¿Puedes comprender las operaciones básicas con cadenas de texto en *Python*?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

> **Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", regresa a los contenidos del inicio, busca los conceptos claves relacionados con los aprendizajes de la sesión y haz una bitácora de ellos. Utiliza palabras clave que te ayuden a recordar y comprender mejor e incluso asigna colores diferentes a cada uno para diferenciarlos.**

Ahora te proponemos un reto con tu grupo usando una rutina llamada Pensar, Presentar e Integrar (P-P-I):

- ◯ Primero, respondemos individualmente.
- ◯ Luego, cada persona en el grupo y en su turno le presenta al resto del equipo sus respuestas.
- ◯ Finalmente, el grupo integra una respuesta unificada.

Las preguntas que te proponemos:

¿?

Si regresamos al reto, lo que hemos aprendido en esta sesión: ¿Para qué serviría?
¿Puedes pensar en un problema que requiera que manipulemos texto y que implique usar la variable tipo string?

Aprovecha este espacio final para hacerte un esquema en que resumas algo de lo que aprendiste, por ejemplo, colocando las definiciones de palabras y algún ejemplo de uso de los tipos de variables empleadas.

Grado 9° **Guía 5**

TIC

Sesión 3

Aprendizajes esperados

Con las actividades de esta sesión se espera que puedas avanzar en:

- Realizar operaciones básicas con cadenas de texto en *Python*.
- Verificar la pertenencia de un carácter o secuencia de caracteres en una cadena de texto en *Python*.

Material para la sesión

Material por grupo:

- Dispositivo con acceso a Internet.

Duración sugerida



33% 34% 33%



Lo que sabemos, lo que debemos saber

Esta sección corresponde al 33% de avance de la sesión

Ahora que sabes cómo se utiliza el ciclo "para", que en *Python* corresponde a for, veamos cómo se puede implementar para trabajar con variables de tipo *string* en *Python*. Este es un concepto importante que te permitirá trabajar con cada caracter de una cadena de manera individual.

```
palabra = "manzana"
for i in palabra:
     print(i)
```

¿Qué crees que imprimirá el programa?
Prueba el código con diferentes palabras.
¿El resultado concuerda con tus predicciones?

Cuando iteramos sobre una cadena, el ciclo for recorre cada caracter uno por uno utilizando el operador **in**. Este operador nos permite acceder a cada elemento de la secuencia (en este caso, cada caracter de la cadena) y realizar operaciones sobre ellos. Veamos un ejemplo de esto en la *Figura 1*.

Figura 1. Código para imprimir la palabra Hola de a una letra por línea

```
mensaje = "Hola"
for caracter in mensaje:
    print(caracter)
```

```
H
o
l
a
```

En este ejemplo, el ciclo for recorre cada caracter de la cadena mensaje y lo imprime.

Al poder iterar sobre una cadena de texto tenemos acceso a cada caracter, esto nos permite hacer operaciones sobre este, por ejemplo, contar caracteres específicos, reemplazar o borrar uno o más carácteres y verificar ciertas condiciones.

```
palabra = input( "Ingresa una palabra: ")
vocales = "aeiouAEIOU"
nueva_palabra = ""

for caracter in palabra:
    if caracter in vocales:
       nueva_palabra = nueva_palabra + "$"
    else:
       nueva_palabra = nueva_palabra + caracter
print(nueva_palabra)
```

```
Ingresa una palabra: murcielago
m$rc$$l$g$
```

En este ejemplo, reemplazamos las vocales de una palabra por el signo $.

¿?

¿Por qué crees que se deben incluir las mayúsculas en vocales? ¿Qué sucede si no las incluyes?

Prueba el código con diferentes palabras y signos.

Veamos un ejemplo de cómo contar las vocales que hay en una frase:

```
palabra = input( "Ingresa una frase: ")
vocales = "aeiouAEIOU"
contador_vocales = 0

for caracter in frase:
    if caracter in vocales:
     contador_vocales = contador_vocales + 1

print("La frase contiene ", contador_vocales,
"vocales")
```

```
Ingresa una frase: El silencio es pacifico
La frase contiene 10 vocales
```

¿?

¿Cómo cambiarías el código para contar cuantas letras "m" hay en la frase? ¿Y para contar cuántos espacios en blanco?

De manera similar, podemos contar cuantas veces se encuentra un dígito dentro de un número:

```
numero = input( "Ingresa un número: ")
contador_digito = 0

for caracter in numero:
    if caracter == "3":
     contador_digito = contador_digito + 1

print("El número contiene ", contador_digito,
"veces el 3.")
```

```
Ingresa un número: 3642397
El número contiene 2 veces el 3.
```

Manos a la obra Conectadas

Esta sección corresponde al 67% de avance de la sesión



Para realizar los siguientes ejercicios, les proponemos trabajar en grupos, según las instrucciones de su docente.

Les proponemos resolver al menos 3 de los ejercicios propuestos, todos ellos requieren que modifiquen algunos de los códigos que ya han trabajado. Luego, comparen con otros grupos los programas.

- Pidan a las y los usuarios que ingresen una palabra. Cuenten cuántas consonantes hay en la palabra e impriman el resultado.
- Pidan a las y los usuarios que ingresen una frase. Eliminen todos los espacios en blanco de la frase e impriman el resultado.
- Pidan a las y los usuarios que ingresen una frase. Cuenten cuántas palabras contiene e impriman el resultado.

- Pidan a las y los usuarios que ingresen una frase. Reemplacen todos los espacios por guiones (-) e impriman la nueva frase.

- Pidan a las y los usuarios que ingresen una frase y un carácter específico. Cuenten cuántas veces aparece ese carácter en la frase e impriman el resultado.

Si aún tienen tiempo, les proponemos otra actividad que les permitirá practicar la iteración sobre variables tipo string y el uso del operador in para realizar búsquedas de caracteres, habilidades cruciales en el manejo de datos en programación.

Para ello, creen un código que cuente cuantas vocales tiene una frase. La o el usuario ingresa una frase, y el programa debe iterar sobre esta para contar cuántas veces se encuentra cada vocal en la frase.

Antes de irnos



Esta sección corresponde al 100% de avance de la sesión

Revisa de manera individual los aprendizajes de la sesión. ¿Crees que lograste alcanzarlos?

1. ¿Puedes realizar operaciones básicas con cadenas de texto en *Python*?

   - Sí
   - Parcialmente
   - Aún no

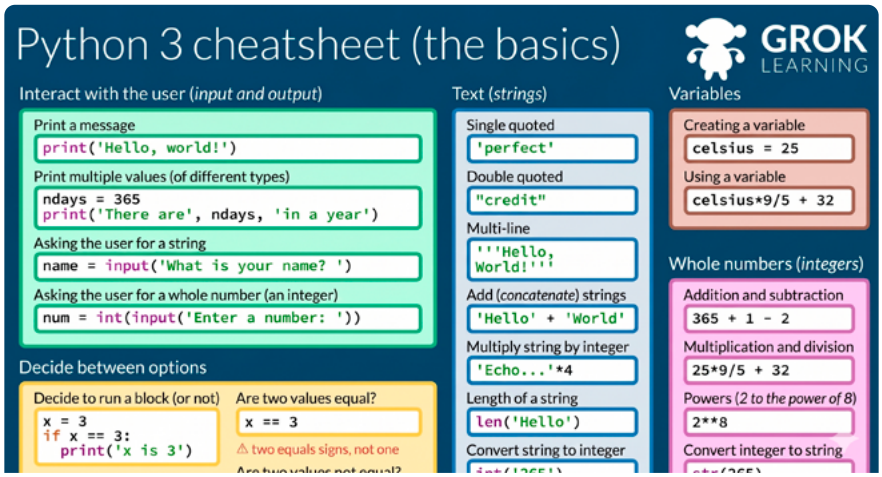
2. ¿Puedes verificar la pertenencia de un carácter o secuencia de caracteres en una cadena de texto en *Python*?

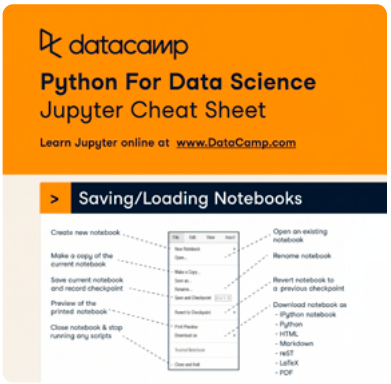
   - Sí
   - Parcialmente
   - Aún no

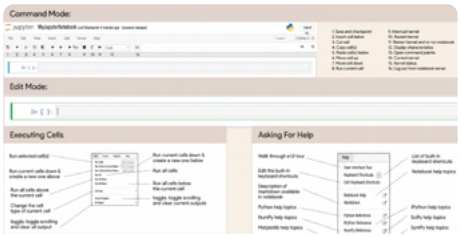
Aprender un nuevo lenguaje de programación requiere práctica y recordar múltiples conceptos importantes. Por eso, en la comunidad de personas que aprenden a programar en *Python* y otros lenguajes basados en texto, existen las **“hojas de referencia”**.

El problema es que casi todas se encuentran en inglés. Tómate unos minutos para empezar a diseñar tu propia hoja de referencia, donde incluyas lo que has aprendido hasta ahora. Esto te ayudará a mejorar aquellos aprendizajes que aún no se han logrado alcanzar o que están parcialmente alcanzados.

Las imágenes te muestran algunos ejemplos. Puedes inspirarte en ellos para crear tus propias hojas de referencia.







Glosario

Hojas de referencia (*Cheat Sheets*): documento que resume los conceptos claves, las reglas de sintaxis y las funciones básicas de un lenguaje de programación.

Grado 9° Guía 5

TIC

Sesión 4

Aprendizajes esperados

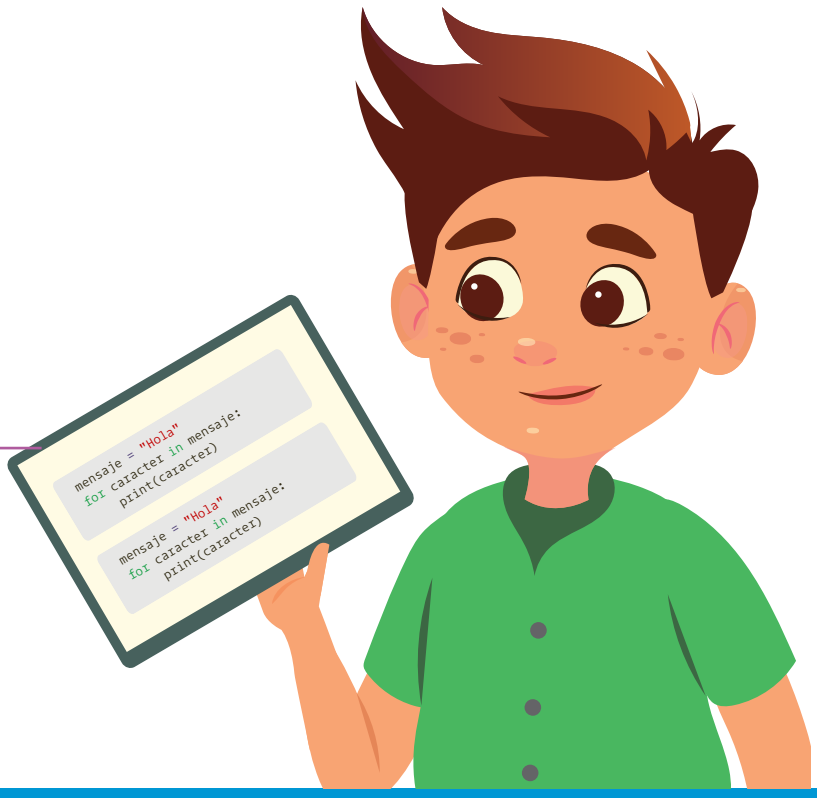
Con las actividades de esta sesión se espera que puedas avanzar en:

- Entender qué son las reglas de sintaxis en los lenguajes de programación.
- Identificar errores de sintaxis en un programa.
- Corregir errores de sintaxis en un programa.

Duración sugerida



40% 40% 20%



Material para la sesión

Material por grupo:

- Dispositivo con acceso a Internet.

Lo que sabemos, lo que debemos saber



Esta sección corresponde al 40% de avance de la sesión

Todos los lenguajes de programación tienen **reglas de sintaxis**, es decir, un conjunto de normas que determinan cómo se ensamblan las sentencias y estructuras de un programa. La sintaxis define el orden y la combinación de los símbolos, palabras y operadores que componen el código.

Los programas escritos en un lenguaje de programación deben seguir su sintaxis para ser traducidos a un lenguaje que tu computador pueda entender. Si hay errores de sintaxis, el procesador que interpreta las instrucciones no podrá traducir ni ejecutar el programa. La sintaxis asegura que el código sea claro y preciso, evitando confusiones que puedan llevar a malentendidos o errores en la ejecución.

Al igual que en la programación, todos los lenguajes humanos tienen reglas sintácticas que determinan cómo se ensamblan las frases. Al hablar o escribir en cualquier lenguaje debemos seguir las reglas de sintaxis para poder ser entendidos correctamente. Sin embargo, a diferencia de los computadores, los humanos podemos inferir significados incluso cuando no se cumplen las reglas sintácticas.

Considera la frase “esta noche nos vemos” en lugar de “nos vemos esta noche”. Aunque la estructura es diferente, probablemente se entenderá lo que se quiere decir debido a nuestra capacidad para inferir significados.

En programación, un pequeño error sintáctico puede causar que el código no se ejecute. Por ejemplo, en *Python*, olvidarse de cerrar un paréntesis o usar una sangría incorrecta resultará en un error de sintaxis que impedirá que el programa funcione. Verás un mensaje de error indicando el lugar donde está.

En *Python*, puedes cometer y cometerás errores de sintaxis. Los errores de sintaxis pueden ser frustrantes al comenzar a aprender un lenguaje de programación basado en texto. Para evitar esto, es necesario seguir las reglas de sintaxis.

Con el tiempo y la práctica, lograrás programar con muy pocos errores de sintaxis, ¡o hasta sin errores! Así que no te desanimes si cometes algunos errores; son parte del proceso de aprendizaje.

Veamos algunos errores de sintaxis comunes:

Sangría o indentación incorrecta:

```
if (x > 10):
print("Mayor que 10")
```

```
Cell In[3], line 2
  print ("Mayor que 10")
  ^
IndentationError: expected an indented block after "if" statement on line 1
```

Uso incorrecto de paréntesis:

```
print("Hola, mundo"
```

```
Cell In[5], line 1
  print ("Hola, mundo"
                      ^
SyntaxError: incomplete input
```

Usar comillas diferentes:

```
print("Hola, mundo’)
```

```
Cell In[7], line 1
  print ("Hola, mundo’)
         ^
SyntaxError: unterminated string literal (detected at line 1)
```

Olvidar el signo de comparación en condicionales:

```
if (x 10):
print("Mayor que 10")
```

```
Cell In[9], line 1
  if (x 10):
      ^
SyntaxError: invalid syntax. Perhaps you forgot a comma?
```

Olvidar importar los módulos necesarios:

```
numero = random.randint(1, 5)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[13], line 1
-----> 1 numero = random,randint(1, 5)
NameError: name "random" is not defined
```

Utilizar solo un signo igual (=) al comparar en condicionales:

```
if (x=10):
print("El número es 10")
```

```
Cell In[1], line 1
  if (x=10):
      ^
SyntaxError: invalid syntax. Maybe you meant "==" or ":=" instead of "="?
```

Uso incorrecto de mayúsculas:

```
Print("Hola, mundo")
```

```
---------------------------------------------------------------------------
NameError
Cell In[13], line 1
-----> 1 Print("Hola, mundo")
NameError: name ‘Print’ is not defined
```

No te dejes abrumar por estos errores. Todas las personas que programan comenten errores de este tipo. Lo importante es encontrarlos y corregirlos.

Practica un poco las reglas de sintaxis con esta actividad:

1. Escribe el siguiente código en *Python*:

```
usuario = "Claudia"
print("Hola", usuario)
```

2. ¿Cuál esperas que sea la salida del programa cuando lo ejecutes? Responde a esta pregunta en tu cuaderno antes de ejecutar el programa.

3. Ejecuta el programa. Si encuentras un mensaje de error, léelo y trata de arreglar el problema. Utiliza la lista debajo para verificar los errores más comunes y marca con una X si encuentras el tuyo:
   - ○ Has omitido una o las dos comillas alrededor de “Claudia”.
   - ○ Has omitido uno o ambos paréntesis en la función *print*.
   - ○ Has omitido una o las dos comillas alrededor de “Hola”.
   - ○ Faltó la coma entre “Hola” y usuario.

Si tu error no está incluido en la lista, escribe cómo lo solucionaste en tu cuaderno.

- ○ ¿Cuál fue la salida del programa una vez que conseguiste ejecutarlo con éxito?

- ○ Modifica la asignación de la línea 1, para que el programa muestre un nombre diferente al saludar al usuario. ¡Haz que muestre tu nombre!

Glosario

Reglas de sintaxis: conjunto de normas que determinan cómo se ensamblan las sentencias y estructuras de un programa.

Manos a la obra

Conectadas

Esta sección corresponde al 80% de avance de la sesión

A continuación, encontrarás algunos ejercicios con fragmentos de código que contienen errores de sintaxis. Tu tarea es escribir cada código en *Python*, luego identificar y corregir los errores para que funcione correctamente. Puedes utilizar la lista de errores comunes y la lista anterior para verificarlos. Al terminar, compara tus respuestas con tus compañeros y compañeras.



1	`if x > 10:`
	`print("Mayor que 10")`
	`else:`
	`print("Menor o igual al 10")`

2	`for i in range(5)` `    print(i)`
3	`print("Hola, mundo"`
4	`mensaje = 'Hola, mundo"` `print(mensaje)`

5	`x 10`
	`y = 5`
	`z = x + y`
	`print(z)`

6	`if x > 5`
	`    print("Mayor que 5")`
	`else:`
	`print("Menor o igual a 5")`

7	`nombre = Juan` `print("Hola, " + nombre)`
8	`for i in range(3):` `print ("Número: " i)`
9	`5numero = 5` `print(5numero)`
10	`for i in range 5:` `     print(i)`

Para ir más lejos

En esta actividad, aplicarás tu comprensión de las reglas de sintaxis en programación y practicarás la identificación y corrección de errores comunes. En el desarrollo de aplicaciones reales, es crucial validar la entrada del usuario para prevenir errores y garantizar el funcionamiento correcto del programa. Esta actividad te mostrará cómo implementar estas validaciones en el contexto de una aplicación de pedidos de productos.

Actividad: desarrolla un sistema de validación para la entrada de datos del usuario en el programa de pedidos que diseñaste en la sesión 2 de esta guía.





Las validaciones que debes tener en cuenta al modificar tu programa son las siguientes:

- El programa debe pedir al usuario que ingrese cuántos productos diferentes quiere ordenar. Asegúrate de que esta entrada sea un número entero positivo. Si el o la usuario ingresa un valor no válido (como un número negativo), muestra un mensaje de error y solicita el ingreso nuevamente hasta que la entrada sea correcta.
- **Nombre del producto:** asegúrate de que el o la usuario ingrese al menos un carácter y que sea diferente a un espacio en blanco.
- **Cantidad del producto:** asegúrate de que la cantidad sea un número entero positivo.



Antes de irnos



Esta sección corresponde al 100% de avance de la sesión

Revisa los aprendizajes de la sesión y reflexiona sobre el grado en que los alcanzaste.

1. ¿Puedes entender qué son las reglas de sintaxis en los lenguajes de programación?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

2. ¿Puedes identificar errores de sintaxis en un programa?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

3. ¿Puedes corregir errores de sintaxis en un programa?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", revisa nuevamente el listado de errores comunes y escribe el ejemplo presentado en cada error común en tu cuaderno. Revisa las notas de las sesiones anteriores y complementa tus apuntes con la sintaxis correcta. Es un buen momento para continuar alimentando la bitácora que iniciaste en la sesión anterior.

Recuerda, depurar es una habilidad muy valiosa en la programación, pero muchas personas que están aprendiendo suelen tenerle miedo a los errores en pantalla. Cierra esta sesión creando un “Muro del terror” junto al resto de tu salón.

Para hacerlo necesitarás dos hojas de papel.

- En la primera hoja, escribe un ejemplo de algún código con errores, puedes guiarte de los de la actividad “Manos a la obra”, o recordar algún error que hayas cometido al programar.

- En la segunda hoja, escribe el mensaje de error que arroja al ejecutarse.

Luego, pega tus hojas en diferentes lugares del “Muro del terror”. Cuando todos los ejemplos estén pegados, intenta encontrar las parejas que corresponden y hablar con tus compañeras y compañeros acerca de cómo corregirían los errores.



Sesión 5

Aprendizajes esperados

Con las actividades de esta sesión se espera que puedas avanzar en:

- Utilizar los aprendizajes logrados para resolver el reto planteado.
- Crear programas que incluyan generación de números aleatorios en *Python*.
- Crear y depurar programas que integren el uso de estructuras condicionales, diferentes tipos de ciclos e interacción con las y los usuarios para la solución de retos.

Duración sugerida



15% 70% 15%



Material para la sesión

Material por grupo:

- ○ Dispositivo con acceso a Internet.
- ○ Anexo 1.1

Anexo

Anexo 1.1



Anexo 1.1 Reto

Tu reto en esta guía será desarrollar un programa en *Python* que simule un selector de juegos de azar para utilizar en el próximo día de la familia de tu colegio.

Este es el plan:

Tu objetivo es crear un programa en *Python* que permita que quien lo use seleccione entre varias opciones qué desea jugar: lanzar una moneda, jugar a la ruleta o jugar BINGO. Una vez que la persona seleccione el juego, el programa debe continuar hasta que la persona decida terminar de jugar. A través de este proyecto, pondrás en práctica los conceptos fundamentales que aprenderás durante las cinco sesiones de la guía.

Objetivo del juego

El objetivo es construir un programa interactivo y fácil de usar, que permita a cualquier usuario seleccionar un juego, jugar varias rondas, y detenerse cuando lo desee. La idea es hacer el programa entretenido y fluido, a la vez que aplicas los conceptos de ciclos, condiciones, variables y números aleatorios.

2 páginas

Lo que sabemos, lo que debemos saber

Esta sección corresponde al 15% de avance de la sesión

Es el momento de recordar el reto que se encuentra en el *Anexo 1.1*, mencionado en la primera sesión de esta guía. En este punto ya tienes casi todo lo necesario para resolverlo.

En la guía 4 usaste un código de ejemplo con la función randint, la cual te sirvió para construir un juego de adivinanzas numéricas. Ahora vamos a aprender de qué se trata y cómo puedes usarla en diferentes situaciones. La función randint es una función del módulo aleatorio (random) en *Python* que genera un número entero aleatorio dentro de un rango especificado por la o el usuario. El rango incluye ambos extremos del intervalo, es decir, tanto el valor mínimo como el valor máximo.

Para utilizar la función randint, primero necesitas importar el módulo random. Aquí tienes un ejemplo de cómo usarla:

```
import random

numero_aleatorio = random.randint(1, 10)
print(numero_aleatorio)
```

En este ejemplo, la función random.randint(1, 10), parte del módulo random, generará un número entero aleatorio en el intervalo 1 y 10 (incluyendo los extremos).

Imagina que quieres simular el lanzamiento de un dado. Un dado tiene 6 caras, numeradas del 1 al 6. Puedes usar randint para generar un resultado aleatorio de un lanzamiento de dado.

```
import random

lanzamiento = random.randint(1, 6)
print("Has lanzado un dado y ha salido: ", lanzamiento)

Has lanzado un dado y ha salido: 6
```

Anotación

Cada vez que hagas un programa que requiera un módulo, debes empezar por cargarlo con la función import.

Veamos ahora un código para el juego de adivinanzas que construiste en la guía pasada. Para esto, puedes usar randint para generar el número que el usuario va a tratar de adivinar. En este caso, el número estará entre 1 y 100.

```
import random
numero_secreto = random.randint(1, 100)
print("Adivina el número entre 1 y 100")
ganar = False
while ganar == False :
    adivinanza = int(input("Introduce tu adivinanza:"))
    if adivinanza < numero_secreto:
       print("Muy bajo, intenta nuevamente.")
    elif adivinanza > numero_secreto:
       print("Muy alto, intenta nuevamente.")
    else:
       print("¡Felicidades! Adivinaste el número.")
       ganar = True
```

Analiza primero este código y trata de anticipar lo que hace. Luego, podrás probarlo.

Varias observaciones sobre este código:

1. La instrucción ganar = False crea una variable llamada ganar como variable lógica o booleana y le asigna el valor de *False* (falso).

2. adivinanza = int(input("Introduce tu adivinanza: ")) hace varias cosas:

   - Escribir en pantalla "Introduce tu adivinanza".
   - Leer un número (input).
   - Convertir el número que viene como caracter a un número entero usando int().
   - Asignar el valor a la variable adivinanza que, por contexto, queda definida como variable tipo entero.

3. El número es comparado para indicar si es muy bajo o muy alto o indicar que se ganó.

Ten en cuenta que la función randint es una herramienta útil en *Python* para generar números enteros aleatorios dentro de un rango específico. Su uso es esencial en muchas aplicaciones, desde juegos hasta simulaciones y análisis de datos.



Glosario

- **while:** en español, "mientras que", permite generar un bucle que se repetirá mientas se de una condición.
- **else / elif:** complementa un **if** (si), para el caso en que NO se cumpla la condición.
- **Variable booleana:** contiene solo dos posibles valores, False y True (falso y verdadero en español, respectivamente).
- **randint:** es una función del módulo aleatorio (random) en *Python* que genera un número entero aleatorio dentro de un rango especificado por el usuario.
- **import:** permite cargar un módulo con funciones existentes en *Python* para ser utilizadas.

Anexo

Anexo 5.1

B	I	N	G	O
8	28	44	53	73
5	19	37	49	64
7	27		52	65
14	30	39	57	61
13	29	38	58	62

B	I	N	G	O
10	18	37	46	74
11	20	41	55	61
1	30		50	72
6	26	35	47	64
3	22	31	59	71

B	I	N	G	O
12	17	41	51	72
11	24	38	46	74
5	27		56	75
8	29	39	60	70
2	26	36	57	68

B	I	N	G	O
2	28	45	49	71
5	24	44	46	64
15	18		59	73
14	29	41	50	70
13	30	34	56	75

B	I	N	G	O
11	28	34	50	68
8	21	39	47	72
1	27		54	75
9	29	32	51	61
13	19	35	53	63

B	I	N	G	O
15	16	39	49	73
12	23	40	52	66
7	20		60	61
4	28	36	56	65
1	24	34	58	70



Manos a la obra

Conectadas

Esta sección corresponde al 85% de avance de la sesión

Es hora de que trabajes en grupo con 2 o 3 personas más para resolver el reto propuesto, sigue las indicaciones de tu docente para la conformación del grupo. Lean de nuevo el reto y procedan a resolverlo.

Proponemos algunas indicaciones que les pueden servir. Tengan en cuenta que es posible que deban utilizar condicionales y ciclo for. Luego, comparen sus soluciones con otro grupo.

1. Escriban un programa que simule el lanzamiento de una moneda (cara o cruz) y compare con la elección de quien está jugando.

2. Escriban un código que simule la ruleta de un casino, generando números aleatorios entre 0 y 36, durante 10 giros. La o el usuario deberá apostar por un número antes de comenzar a jugar. Deberán verificar si el usuario ganó o perdió.

3. Escriban un programa que permita jugar BINGO. Para esto, debe elegir un número al azar del 1 al 5 que corresponda a las letras BINGO. Por ejemplo: 1 = B, 2 = I, 3 = N, etc. Luego debe generar un número aleatorio del 1 al 75 e imprimirlo. Después de cada turno, debe preguntar si el usuario quiere generar otro número o terminar el juego.

Es tiempo de completar la solución del reto. Cuando terminen, pueden jugar al BINGO utilizando los cartones disponibles en el *Anexo 5.1.*

Si quieren continuar programando, pueden agregar más opciones, como simular un lanzamiento de dados para jugar parqués.

La computación y la sociedad

Los computadores son esenciales en casi todas las actividades cotidianas y han transformado la manera en que resolvemos problemas en la sociedad. Sin embargo, el impacto de la computación no solo se debe a las máquinas, sino a las personas que crean los programas que nos ayudan a enfrentar estos desafíos.

Desarrollar programas es un reto creativo y técnico. Por ejemplo, imagina un programa diseñado para una farmacia: este sistema podría llevar el inventario de productos, alertar cuando un medicamento esté próximo a agotarse o cuando uno esté cerca de vencer. Este tipo de aplicación no solo facilita la administración del negocio, sino que ayuda a proteger la salud de los clientes, mostrando cómo la computación puede mejorar la calidad de vida en nuestra comunidad.

Antes de irnos



Esta sección corresponde al 100% de avance de la sesión

Revisa los aprendizajes de la sesión y reflexiona sobre el grado en que los alcanzaste.

1. ¿Puedes utilizar los aprendizajes logrados para resolver el reto planteado?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no



2. ¿Puedes crear programas que incluyan generación de números aleatorios en *Python*?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

3. ¿Puedes crear y depurar programas que integren el uso de estructuras condicionales, diferentes tipos de ciclos e interacción con los usuarios para la solución de retos?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no



Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", consulta con tu docente las dudas que aún tengas, identifica los aprendizajes que más te han costado trabajo y profundiza sobre ellos buscando más información en Internet.

Al terminar la sesión, continúa con la creación de tu hoja de referencia. Imagina que vas a presentar un examen sobre *Python* y solo te permiten sacar una hoja como ayuda. *¿Qué escribirías? ¿Qué te gustaría tener por si acaso?* Cuando la termines, guárdala en un lugar seguro, es posible que te sea útil en el proceso de aprendizaje.



Anexo 1.1 Reto

Tu reto en esta guía será desarrollar un programa en *Python* que simule un selector de juegos de azar para utilizar en el próximo día de la familia de tu colegio.



Este es el plan:

Tu objetivo es crear un programa en *Python* que permita a quien lo use seleccionar entre varias opciones qué desea jugar: lanzar una moneda, jugar a la ruleta o jugar BINGO. Una vez que la persona seleccione el juego, el programa debe continuar hasta que la persona decida terminar de jugar. A través de este proyecto, pondrás en práctica los conceptos fundamentales que aprenderás durante las cinco sesiones de la guía.

Objetivo del juego

El objetivo es construir un programa interactivo y fácil de usar, que permita a cualquier usuario seleccionar un juego, jugar varias rondas, y detenerse cuando lo desee. La idea es hacer el programa entretenido y fluido, a la vez que aplicas los conceptos de ciclos, condiciones, variables y números aleatorios.

Pasos para desarrollar el programa

1. Diseña el menú principal de selección de juegos:
   - Crea un menú de opciones donde el usuario pueda escoger entre “Lanzar una moneda”, “Ruleta,” o “BINGO.”
   - Usa un ciclo para que el menú se muestre de forma repetitiva hasta que el usuario decida terminar.
2. Implementa la función para lanzar una moneda:
   - Permite que la o el jugador lance la moneda varias veces hasta que decida salir al menú principal.
3. Desarrolla el juego de la ruleta:
   - Permite que la o el jugador siga jugando rondas o regrese al menú principal.
4. Implementa el juego de BINGO:
   - Después de cada ronda, dale la opción de continuar o regresar al menú principal.
5. Dale un toque final al programa:
   - Usa *strings* para personalizar los mensajes y mejorar la experiencia del usuario. Asegúrate de incluir instrucciones claras y utiliza condicionales para manejar las decisiones del usuario (como seguir jugando o salir del juego).
   - Agrega mensajes para avisar a los usuarios que su respuesta no es una opción válida. Por ejemplo, si eligen un juego que no está en el menú.

Ayudas

- Aprovecha el uso de ciclos for y while para repetir los juegos.
- Usa input() y print() para interactuar con la/el usuario y mostrar mensajes.
- randint() te ayudará a generar resultados aleatorios en cada juego.
- Utiliza variables de control para llevar un registro de los resultados de cada juego.

Anexo 5.1 Ejemplos

B	I	N	G	O
8	28	44	53	73
5	19	37	49	64
7	27		52	65
14	30	39	57	61
13	29	38	58	62

B	I	N	G	O
10	18	37	46	74
11	20	41	55	61
1	30		50	72
6	26	35	47	64
3	22	31	59	71

B	I	N	G	O
12	17	41	51	72
11	24	38	46	74
5	27		56	75
8	29	39	60	70
2	26	36	57	68

B	I	N	G	O
2	28	45	49	71
5	24	44	46	64
15	18		59	73
14	29	41	50	70
13	30	34	56	75

B	I	N	G	O
11	28	34	50	68
8	21	39	47	72
1	27		54	75
9	29	32	51	61
13	19	35	53	63

B	I	N	G	O
15	16	39	49	73
12	23	40	52	66
7	20		60	61
4	28	36	56	65
1	24	34	58	70



TIC

Apoya:

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}