Encuentro con Python

Grado 9° Guía 4

TIC

Estudiantes

Apoya:

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}

Encuentro con Python

Grado 9° Guía 4

TIC

Estudiantes

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}

MINISTERIO DE TECNOLOGÍAS DE LA INFORMACIÓN Y LAS COMUNICACIONES

Julián Molina Gómez
Ministro TIC

Luis Eduardo Aguiar Delgadillo
Viceministro (e) de Conectividad

Yeimi Carina Murcia Yela
Viceministra de Transformación Digital

Óscar Alexander Ballen Cifuentes
Director (e) de Apropiación de TIC

Alejandro Guzmán
Jefe de la Oficina Asesora de Prensa

Equipo Técnico
Lady Diana Mojica Bautista
Cristhiam Fernando Jácome Jiménez
Ricardo Cañón Moreno

Consultora experta
Heidy Esperanza Gordillo Bogota

BRITISH COUNCIL

Felipe Villar Stein
Director de país

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

Marianella Ortiz Montes
Jefe de Colegios

David Vallejo Acuña
Jefe de Implementación Colombia Programa

Equipo operativo
Juanita Camila Ruiz Díaz
Bárbara De Castro Nieto
Alexandra Ruiz Correa
Dayra Maritza Paz Calderón
Saúl F. Torres
Óscar Daniel Barrios Díaz
César Augusto Herrera Lozano
Paula Álvarez Peña

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

Edición y coautoría versiones finales
Alejandro Espinal Duque
Ana Lorena Molina Castro
Vanesa Abad Rendón
Raisa Marcela Ortiz Cardona

Edición
Juanita Camila Ruiz Díaz
Alexandra Ruiz Correa

British Computer Society – Consultoría internacional

Niel McLean
Jefe de Educación

Julia Adamson
Directora Ejecutiva de Educación

Claire Williams
Coordinadora de Alianzas

Asociación 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
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 Ramos

PUNTOAPARTE EDITORES

Diseño, diagramación, ilustración, y revisión de estilo

Impreso por Panamericana Formas e Impresos S.A., Colombia

Material producido para Colombia Programa, en el marco del convenio 1247 de 2023 entre el Ministerio de Tecnologías de la Información y las Comunicaciones y el British Council

“Esta guía corresponde a una versión preliminar en proceso de revisión y ajuste. La versión final actualizada estará disponible en formato digital y puede incluir modificaciones respecto a esta edición”

Prólogo

Estimados educadores, estudiantes y comunidad educativa:

En el Ministerio de Tecnologías de la Información y las Comunicaciones, creemos que la tecnología es una herramienta poderosa para incluir y transformar, mejorando la vida de todos los colombianos. Nos guia una visión de tecnología al servicio de la humanidad, ubicando siempre a las personas en el centro de la educación técnica.

Sabemos que no habrá progreso real si no garantizamos que los avances tecnológicos beneficien a todos, sin dejar a nadie atrás. Por eso, nos hemos propuesto una meta ambiciosa: formar a un millón de personas en habilidades que les permitan no solo adaptarse al futuro, sino construirlo con sus propias manos. Hoy damos un paso fundamental hacia este objetivo con la presentación de las guías de pensamiento computacional, un recurso diseñado para llevar a las aulas herramientas que fomenten la creatividad, el pensamiento crítico y la resolución de problemas.

Estas guías no son solo materiales educativos; son una invitación a imaginar, cuestionar y crear. En un mundo cada vez más impulsado por la inteligencia artificial, desarrollar habilidades como el pensamiento computacional se convierte en la base, en el primer acercamiento para que las y los ciudadanos aprendan a programar y solucionar problemas de forma lógica y estructurada.

Estas guías han sido diseñadas pensando en cada región del país, con actividades accesibles que se adaptan a diferentes contextos, incluyendo aquellos con limitaciones tecnológicas. Esta es una apuesta por la equidad, por cerrar las brechas y asegurar que nadie se quede atrás en la revolución digital. Quiero destacar, además, que son el resultado de un esfuerzo colectivo: más de 2.000 docentes colaboraron en su elaboración, compartiendo sus ideas y experiencias para que este material realmente se ajuste a las necesidades de nuestras aulas. Además, con el apoyo del British Council y su red de expertos internacionales, hemos integrado prácticas globales de excelencia adaptadas a nuestra realidad nacional.

Hoy presentamos un recurso innovador y de alta calidad, diseñado en línea con las orientaciones curriculares del Ministerio de Educación Nacional. Cada página de estas guias invita a transformar las aulas en espacios participativos, creativos y, sobre todo, en ambientes donde las y los estudiantes puedan desafiar estereotipos y explorar nuevas formas de pensar.

Trabajemos juntos para garantizar que cada estudiante, sin ·importar dónde se encuentre, tenga acceso a las herramientas necesarias para imaginar y construir un futuro en el que todos seamos protagonistas del cambio. Porque la tecnología debe ser un instrumento de justicia social, y estamos comprometidos a que las herramientas digitales ayuden a cerrar brechas sociales y económicas, garantizando oportunidades para todos.

Con estas guias, reafirmamos nuestro compromiso con la democratización de las tecnologías y el desarrollo rural, porque creemos en el potencial de cada región y en la capacidad de nuestras comunidades para liderar el cambio.

Julián Molina Gómez
Ministro de Tecnologías de la Información y las Comunicaciones
Gobierno de Colombia

Grado 9º Guía 4

TIC

Guía de íconos

Lógica, programación y depuración

Equidad en el acceso y la participación en el mundo digital

Aprendizajes de la guía

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

- Reconocer la sintaxis y el uso de un lenguaje de programación basado en texto.
- Usar estructuras de condicionales múltiples (instrucciones " si, sino si, sino ") para controlar el flujo de ejecución de un programa en un lenguaje basado en texto.

Resumen de la guía

En esta guía explorarás el lenguaje de programación *Python*, comenzando con una introducción a sus características y la creación de programas básicos en un editor interactivo. Aprenderás a manejar entradas y salidas mediante las funciones **input()** y **print()**, y profundizarás en el uso de condicionales para tomar decisiones lógicas en sus programas. Posteriormente, te centrarás en la sintaxis de condicionales en *Python* y cómo implementar ciclos "mientras que" (*while*) en el contexto del desarrollo de videojuegos, integrando todos estos conceptos para construir soluciones computacionales.

Resumen de las sesiones

Sesión 1

En esta sesión, se aprenderá qué es *Python* y cuáles son sus principales características. Crearás tus primeros programas en un editor interactivo utilizando variables y operaciones sencillas.

Sesión 2

La segunda sesión se centrará en el concepto de entradas y salidas en programación. Se utilizará la función **input()** de *Python* para recibir datos del usuario y la función **print()** para mostrar resultados y mensajes en pantalla, comprendiendo así cómo interactuar con el usuario en un programa.

Aprendizajes de la guía

- Reconocer la computación como un campo que se ha beneficiado de la diversidad y la colaboración.
- Mencionar algunas mujeres que han sido líderes en el desarrollo de la computación.
- Programar soluciones computacionales usando un lenguaje basado en texto.

Sesión 3

En esta sesión se retomará el concepto de condicionales, recordando cómo se representan mediante diagramas y utilizando pseudocódigo para diseñar soluciones a problemas cotidianos.

Sesión 4

La cuarta sesión profundizará en la sintaxis de condicionales en *Python*, enseñando a utilizar las estructuras " if, elif y else" correctamente. Se aplicarán estos conceptos para resolver problemas prácticos, fortaleciendo su comprensión de cómo tomar decisiones en el código.

Sesión 5

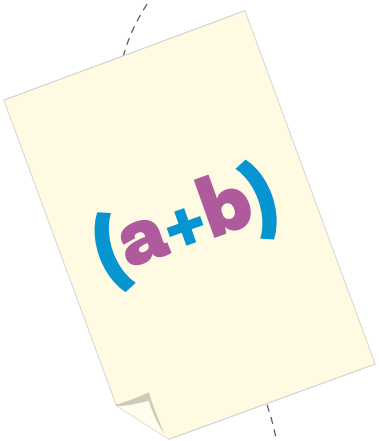
En la sesión final se introducirá el uso del ciclo “mientras que” (*while*) en el contexto del desarrollo de videojuegos. Se integran ciclos y condicionales para controlar el flujo del juego y aprenderán a escribir pseudocódigo para implementar la lógica del ciclo “mientras que”.

Conexión con otras áreas

En esta guía se recuerdan conceptos de la programación trabajados en otros grados y se fomenta el desarrollo de habilidades técnicas para aprender un nuevo lenguaje de programación desde sus aspectos básicos. A continuación, se menciona la conexión de esta guía con otra área del aprendizaje:

Matemáticas

- Se desarrollan actividades desconectadas y conectadas que implican el desarrollo de algoritmos y la aplicación de la lógica matemática en situaciones prácticas. De igual manera, se fortalecen conceptos como variables y se proponen ejercicios con operaciones básicas para la resolución de problemas.



Grado 9° Guía 4

TIC

Sesión 1

Aprendizajes esperados

Al final de esta sesión verifica que puedas:

- Explicar qué es *Python* y sus características principales.
- Ejecutar y modificar programas sencillos en *Python*.
- Emplear variables y realizar operaciones sobre ellas en un programa de *Python*.

Material para la clase

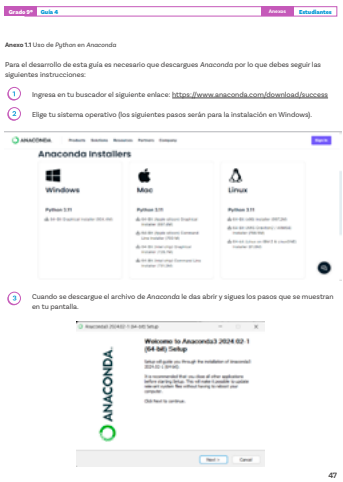
- Copia del Anexo 1.1 y 1.2 según el editor de Python utilizado.
- Acceso a un computador con internet.

Duración sugerida

33% 34% 33%

Anexos

Anexo 1.1



Anexo 1.2

Lo que sabemos, lo que debemos saber

Esta sección corresponde al 33% de avance de la sesión

En esta guía vas a aprender sobre el lenguaje de programación *Python*.

> *¿Lo conoces? ¿Lo has escuchado antes? ¿Qué otros lenguajes de programación conoces? ¿En qué se parecen? ¿En qué se diferencian?*

Python es un lenguaje de programación muy popular, utilizado en aplicaciones web, desarrollo de software, ciencia de datos y aprendizaje automático (*machine learning*). Los desarrolladores prefieren *Python* debido a su eficiencia y facilidad de aprendizaje, así como a su compatibilidad con diversas plataformas. Además, *Python* es de uso gratuito y se integra eficazmente con diferentes tipos de sistemas. Una de las características de *Python* es que se pueden leer y comprender fácilmente los programas escritos en este lenguaje debido a que su sintaxis básica es similar a la del inglés.

Por esta razón, vas a encontrar funciones como: **round(), sum(), replace(), sort() o print()**. Que, si las traduces al español, es fácil predecir lo que hacen. En este caso, redondear un número, sumar, reemplazar, ordenar o imprimir. En esta sesión vas a empezar por la más común y usada de todas: **print()**.

Para programar en *Python* vas a utilizar un **editor interactivo**. Sigue las instrucciones de tu docente o consulta el *Anexo 1.1*, que contiene tutoriales detallados para ingresar al editor que tu docente indique.

Una vez abierto tu editor de *Python*, el siguiente paso es aprender a utilizarlo correctamente. Esto te permitirá escribir y **ejecutar** código de manera interactiva, documentar tu proceso de pensamiento y visualizar datos de manera intuitiva.

Enlace

Archivos de esta sesión: Acceso a un editor de Python. Existen muchas otras plataformas que permiten programar en Python y probar estos programas. Por ejemplo, Google ofrece una herramienta en línea. Entrar a la opción de Notebook.

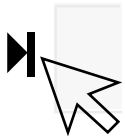
Para empezar, vas a probar algunas funciones. Observa el siguiente código:

```
print("¡Hola, Mundo!")
```

¿? *¿Qué crees que hará este código cuando lo ejecutes?*

Escribe tu predicción aquí:

Ahora escribe y **ejecuta** el código en tu cuaderno de *Python*. Haz clic en el botón de "ejecutar" que normalmente es un botón de forma triangular y que cambia dependiendo del la herramientas que estás utilizando. ¿Qué sucedió?



Como ves, hay varios elementos en una sola línea de código.

¿? *¿Qué crees que pasaría si olvidas escribir un paréntesis? ¿Qué pasaría si no utilizas las comillas "”?*

Haz pruebas y observa lo que pasa. Recuerda volver al código original antes de introducir otro cambio.

Es probable que en otros lenguajes de programación hayas aprendido sobre variables. En *Python*, las variables se definen escribiendo su nombre, seguido por un igual, y luego su valor, así:

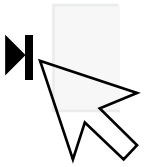
```
variable = valor
Ejemplo:
edad = 23
hora = 12
```

Pero, si son palabras largas, usamos guion bajo para separar las palabras:

```
numero_zapatos = 3
cantidad_estudiantes = 40
```

Y si queremos que la variable guarde texto, usamos comillas:

```
nombre = "Mariana"
```



El símbolo # se usa para colocar un comentario. Todo lo que esté a continuación, no será interpretado como programa.

Glosario

- **Editor interactivo:** entorno de programación que permite escribir y ejecutar código en el mismo lugar, viendo resultados al instante para facilitar el aprendizaje.
- **Ejecutar:** en programación, significa poner en marcha un programa o un código para que la computadora realice las instrucciones escritas y muestre el resultado esperado.

Manos a la obra

Esta sección corresponde al 67% de avance de la sesión

Organízate en grupos siguiendo las indicaciones de tu docente. Ahora vas a familiarizarte con las operaciones básicas en *Python*, como la suma, resta, multiplicación y división. Estas operaciones son fundamentales para realizar cálculos y desarrollar programas más complejos.

Observa el siguiente código y explica lo que hace. Escribe tu explicación en el recuadro.

¿? *¿Es fácil de entender? ¿Por qué?*

```
# Suma
a= 10
b= 5
suma = a + b
print( "Suma : " , suma)

# Resta
resta = a - b
print("Resta: ", resta)

# Multiplicación
multiplicacion = a * b
print("Multiplicación: ",
multiplicacion)

# División
division = a / b
print("División: ", division)
```

Ahora replícalos en tu cuaderno y pruébalos. No olvides hacer clic en ejecutar.

¿Qué sucede si quitas los # del código?
¿Cuáles crees que serán los valores resultantes de las operaciones?
¿Qué ocurre si cambias los valores de las variables a y b por otros números? ¿Qué pasa si escribes una variable en mayúsculas en el print()?

En este ejemplo, definimos dos variables, a y b, y luego realizamos operaciones básicas como suma (a + b), resta (a - b), multiplicación (a * b), y división (a / b). Cada operación produce un resultado que se almacena en una nueva variable y se muestra utilizando print().

Recuerda que las variables pueden tomar cualquier nombre.

Modifica las variables para que se llamen **base** y **altura**.

¿Cómo podrías calcular el área de un rectángulo de base 4 y altura 10?
¿Cómo podrías calcular el área de un triángulo de base 9 y altura 5?

Si no recuerdas las fórmulas del área puedes buscarlas en internet o preguntarle a tu docente.

Crea una nueva variable llamada **área** y guarda los resultados de las operaciones anteriores. Luego imprime la variable área.

¿Qué sucedió?

Veamos ahora un nuevo código.

```
palabra = "hola"
print(palabra * 3)
```

> *¿Qué crees que imprimirá el programa cuando se multiplica palabra por 3? ¿Y cuando se multiplica por 2?*

Ejecuta y observa si se cumple tu predicción.

Ahora suma dos palabras diferentes.

> *¿Qué pasaría si restamos dos palabras? Inténtalo y anota tus observaciones.*

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 explicar qué es *Python* y sus características principales?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no
2. ¿Puedes ejecutar y modificar programas sencillos en *Python*?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no
3. ¿Puedes emplear variables y realizar operaciones sobre ellas en un programa de *Python*?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", piensa: ¿puntualmente qué es lo que te causa mayor dificultad? Pide a tu docente apoyo para resolver esa duda que aún tienes sobre los contenidos de la sesión y compara tus notas sobre la clase y lo que aprendiste con los aprendizajes de tus compañeras y compañeros.

Ahora te proponemos un reto. Es probable que en grados anteriores hayas aprendido a programar en otros lenguajes de programación basados en bloques como *MakeCode* y *Scratch*.

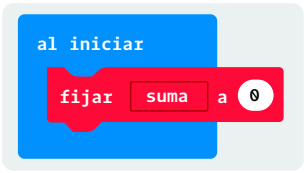
Vuelve a los aprendizajes de esta sesión y crea un gráfico comparativo sobre cómo programas en *Python* y cómo programarías algo similar en *MakeCode* o *Scratch*.

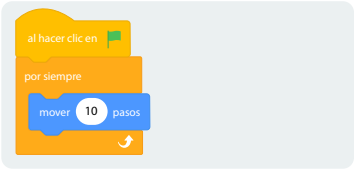
A continuación, te proponemos un ejemplo, pero puedes usar el formato que desees y tu creatividad.

python™ | Microsoft MakeCode | SCRATCH

Para asignar valores a variables...

```
suma = 0
```





...

Grado 9° Guía 4

TIC

Sesión 2

Aprendizajes esperados

Al final de esta sesión verifica que puedas:

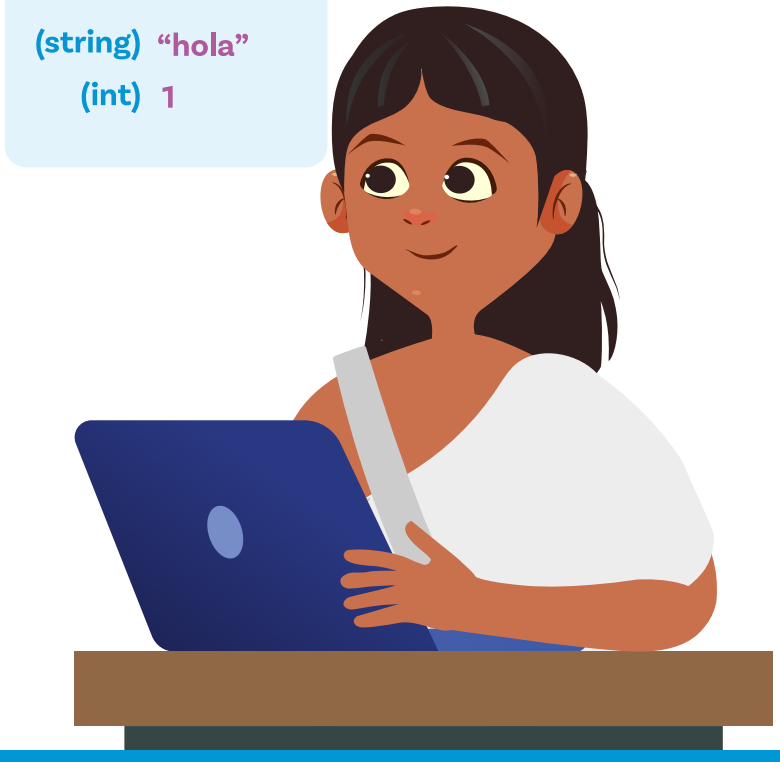
- Crear programas en *Python* que permitan interactuar con los usuarios.
- Comparar las entradas y salidas disponibles en *Python* respecto a las de un dispositivo como la micro:bit.
- Reconocer las diferencias entre las variables de tipo texto y numéricas.

Material para la clase

- Acceso a un computador con internet o con Anaconda instalado.

Duración sugerida

40% 40% 20%



Lo que sabemos, lo que debemos saber

Esta sección corresponde al 40% de avance de la sesión

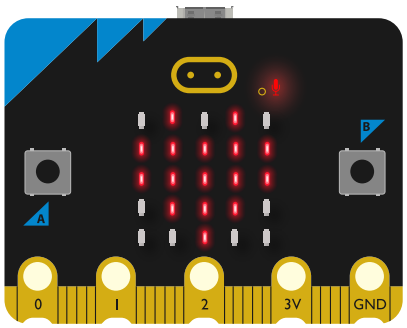
Python se puede utilizar con herramientas de accesibilidad, como lectores de pantalla y software de reconocimiento de voz, en caso de ser necesario.

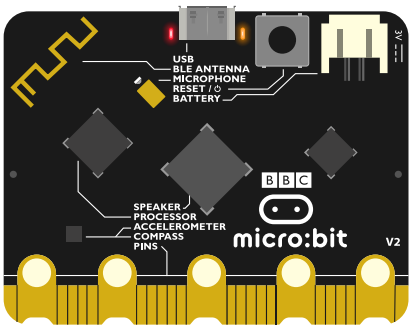
En tus proyectos de programación encontrarás dos conceptos clave: entradas y salidas. Si has trabajado con otros lenguajes como *MakeCode*, recordarás que las entradas se refieren a cualquier dato o información que se introduce en un sistema, dispositivo o programa para ser procesado. En el contexto de la programación, el *input* puede provenir de varias fuentes, como el teclado, el ratón, archivos de texto, sensores, o incluso otros programas.

Como podrás recordar, las salidas se refieren a cualquier dato o información que un sistema, dispositivo o programa envía o muestra al exterior. Esta información puede manifestarse de diversas formas, como texto en una pantalla, archivos guardados, señales de audio, gráficos, o datos enviados a otros sistemas.

¿?

¿Recuerdas qué salidas tiene una micro:bit? ¿Puedes nombrarlas? ¿Qué otros dispositivos conoces que tengan entradas y salidas?





Python es un lenguaje escrito, y sus funciones están basadas en palabras del inglés. Por eso, se puede usar la función input() para recibir datos del usuario. Y una de las salidas más comunes es la pantalla del computador, por eso la función print() es comúnmente utilizada para imprimir los resultados y mostrar mensajes a los usuarios.

En la mayoría de los programas, las entradas y las salidas están conectadas. El programa toma una entrada, la procesa de acuerdo con una serie de instrucciones, y luego produce la salida basada en ese procesamiento. Esta interacción es fundamental para la funcionalidad de aplicaciones y sistemas, permitiendo a los usuarios interactuar con ellos de manera efectiva.

En *Jupyter Notebook*, trabajar con entradas y salidas es muy sencillo. Puedes escribir código en una celda y ejecutarlo para ver inmediatamente el resultado. Por ejemplo, puedes solicitar al usuario que ingrese su nombre utilizando input() y luego mostrar un saludo personalizado con print(). Aquí tienes un ejemplo:

¿Qué crees que hace este código?
¿Qué pasará si no escribes el signo +?
¿Qué diferencia a las palabras en rojo de las que están en negro?

Comparte tus respuestas con alguien más. Luego, pruébalo.

Como notarás, cuando ejecutas esta celda en *Jupyter Notebook*, aparece una casilla donde puedes introducir tu nombre (entrada) y, al presionar la tecla *Enter*, el saludo personalizado se mostrará inmediatamente debajo (salida).

Si tu resultado no es el esperado, prueba revisar con mucho detalle la forma en la que escribes el código, por ejemplo haciéndote las siguientes preguntas:

¿Escribiste print en minúscula?
¿Abriste y cerraste todos los paréntesis?
¿Utilizaste el mismo tipo de comillas antes y después de las palabras?
¿Escribiste el nombre de tu variable exactamente igual, cada vez que la usaste?

Es importante utilizar comillas ("") cuando empleamos las funciones input() y print(), ya que, de lo contrario, el código no funcionará correctamente. Además, es importante usar el signo más (+) cuando queramos incluir una variable en el texto que estamos escribiendo, ya sea al usar input()o print().

En *Python*, es importante conocer exactamente el **tipo de variable** con el que estás trabajando. Por ejemplo, la función input() se utiliza para obtener datos del usuario como una cadena de texto (en inglés, se llama *string*). Sin embargo, en muchos casos necesitamos trabajar con números enteros (en inglés, se llama *integers* y se abrevian int) en lugar de cadenas de texto.

El siguiente programa de *Python* pide un número y muestra el resultado de sumarle 10.

```
numero = input("Por favor, introduce un número: ")
resulta = numero + 10
print("El resultado de sumer 10 a tu numero es:", resultado)
```

Al ejecutar el programa y escribir el número que quieres cuando se solicite, aparecerá un mensaje de **error** en la línea 2.

¿Cuál crees que es la razón para que no funcione el programa?

```
TypeError                    Traceback (most recent call last)
Cell In[6], line 2
      1 numero = input("Por favor, introduce un número: ")
      2 resulta = numero + 10
      3 print("El resultado de sumer 10 a tu número es:", resultado)
TypeError: can only concatenate str (not "int") to str
```

Cuando solicitas un número al usuario, la función input() devuelve la entrada del usuario como una cadena de texto. Para convertir esta a un número entero, usamos la función int(). Esto es especialmente útil cuando realizamos operaciones matemáticas. Aquí tienes un ejemplo de cómo hacerlo:

```
numero = int(input("por favor, introduce un número: "))
resultado = numero + 10
print("El resultado de sumar 10 a tu número es:", resultado)
```

- (int)
- (string)
- (float)
- (bool)

Glosario

Celda: Sección de un cuaderno interactivo donde se escribe y ejecuta código o texto. Cada celda permite ver el resultado de su ejecución de manera independiente.

Tipos de variables: Categorías de datos que una variable puede almacenar en programación. Los principales tipos son:

- **Entero (int):** Números enteros, como 1, -5 o 42.
- **Cadena (string)**: Texto, como “hola” o “Python”.
- **Flotante (float):** Números con decimales, como 3.14 o -0.001.
- **Booleano (bool):** Valores lógicos, *True* o *False*, que indican verdadero o falso.

Jupyter Notebook: Es una aplicación en la que puedes escribir y ejecutar código de programación, agregar imágenes, gráficos y explicaciones en un mismo lugar. Se usa mucho en ciencia de datos e inteligencia artificial para hacer pruebas y analizar información.

Manos a la obra

Conectadas

Esta sección corresponde al 80% de avance de la sesión

En la sesión anterior aprendiste sobre *Python* y algunas de sus características y creaste programas sencillos utilizando diferentes variables. En esta sesión aprendiste a pedir información al usuario y a mostrar mensajes en pantalla. Ahora vas a aplicarlo en la siguiente actividad, para ello, puedes reunirte con 2 compañeras o compañeros, siguiendo las instrucciones de tu docente.

Resuelvan los ejercicios propuestos, asegurándose de utilizar correctamente las funciones input() y print() en *Python*. Luego, comparen con otros grupos cómo abordaron cada problema.

1. Una niña quiere hacer un programa personalizado que les permita a docentes sumar las calificaciones obtenidas por las y los estudiantes en una asignatura.

2. Un niño quiere hacer un programa para ayudar a sus amigos y amigas a calcular la edad de sus abuelos. El programa debe pedir a cada persona que ingrese el año de nacimiento y, con base en eso, calcular cuántos años tienen actualmente los y las abuelas.

3. Un estudiante está desarrollando un programa para ayudar a sus amigas y amigos a convertir horas en minutos. El programa debe pedir a cada usuario que ingrese una cantidad de horas y, con base en eso, calcular el total de minutos correspondientes.

4. Una estudiante está creando un programa en *Python* para ayudar a sus amigas y amigos a personalizar saludos. El programa debe pedir a cada usuario que ingrese su nombre utilizando la función **input()**, y luego imprimir un mensaje de saludo personalizado usando la función **print()**.

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 crear programas en *Python* que permitan interactuar con los usuarios?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

2. ¿Puedes comparar las entradas y salidas disponibles en *Python* respecto a las de un dispositivo como la micro:bit?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

3. ¿Puedes reconocer las diferencias entre las variables de tipo texto y numéricas?

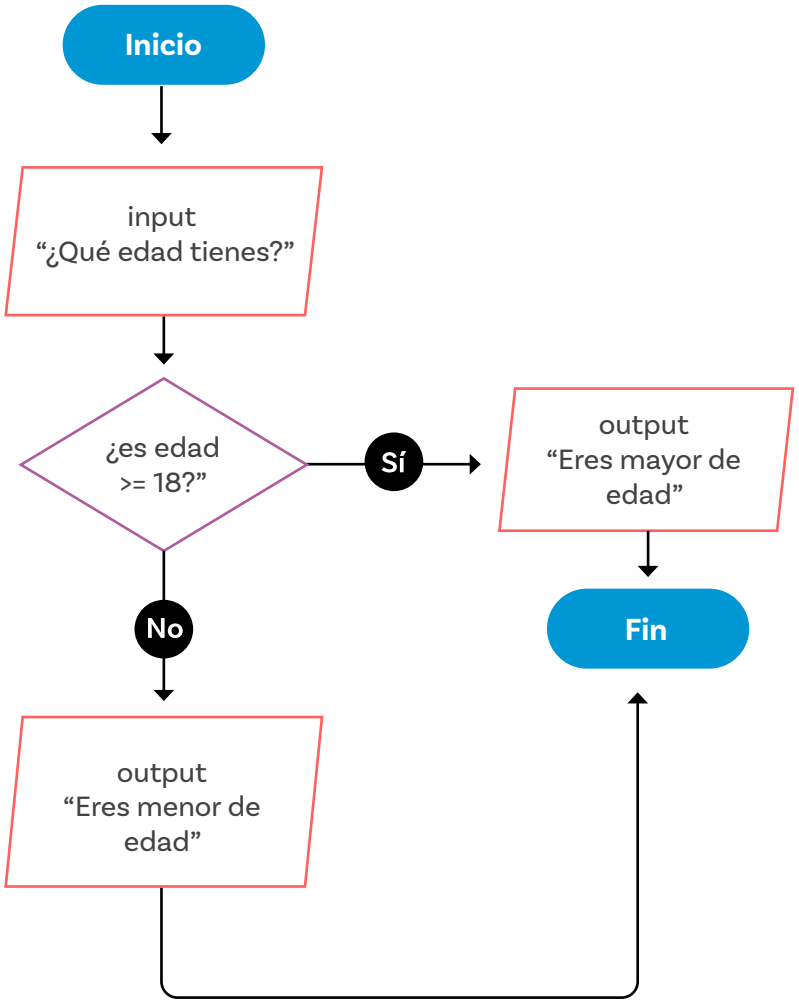
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

> **Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", regresa a los contenidos de la sesión, particularmente a Lo que sabemos, lo que debemos saber, lee cuidadosamente y pide apoyo a tu docente con las dudas que tengas.**

Ahora piensa por un momento en las siguientes preguntas:

> *¿Qué significa el término "entradas" en programación y por qué es importante?*
> *¿Qué preguntas te quedan de la sesión? ¿Sobre qué te gustaría aprender más?*

Aprovecha este espacio final para diseñar un diagrama de flujo que represente el proceso de un programa que utiliza **input()** para recibir datos del usuario y **print()** para mostrar resultados en pantalla. A continuación, encontrarás un ejemplo.



Grado 9° Guía 4

TIC

Sesión 3

Aprendizajes esperados

Al final de esta sesión verifica que puedas:

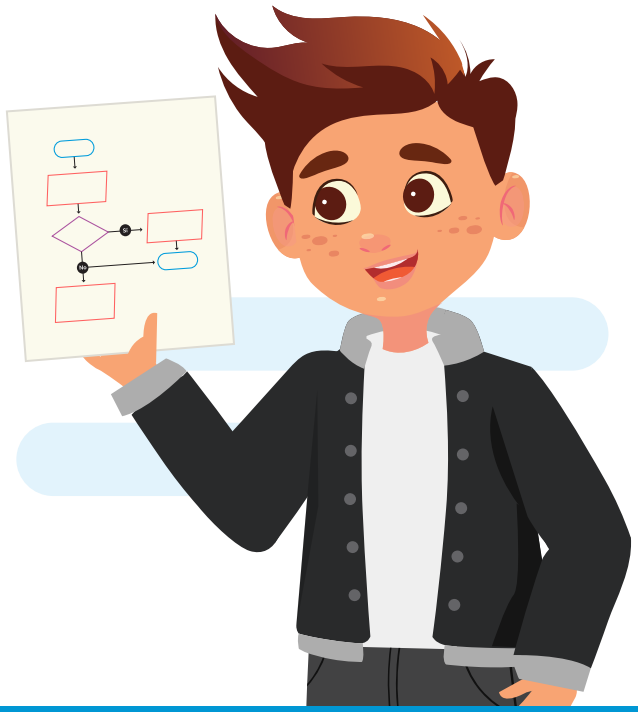
- Reconocer el uso de las estructuras condicionales en diferentes situaciones.
- Diseñar soluciones a situaciones problemas, utilizando correctamente los condicionales.

Material para la clase

- Tres copias del Anexo 3.1, dos sets de reacciones del Anexo 3.2

Duración sugerida

33% 34% 33%



Lo que sabemos, lo que debemos saber

Esta sección corresponde al 33% de avance de la sesión

Es posible que recuerdes que en grados anteriores aprendiste sobre los **condicionales**. Los condicionales son esas reglas que permiten que un código o programa se ejecute como queremos, tomando decisiones basadas en ciertas condiciones. En programación, los condicionales se usan para ejecutar diferentes bloques de código dependiendo de si una condición es verdadera o falsa.

Los condicionales los podemos ver en forma de **diagramas** que utilizan flechas para representar los pasos de un proceso y las decisiones que se toman en función de ciertas condiciones. Supongamos que queremos determinar si un número es positivo, para esto primero nos debemos preguntar: ¿cómo sabemos si un número es positivo? ¿Qué condición debe cumplir el número? Al tener claras las condiciones que se deben cumplir tenemos como resultado el siguiente diagrama:

¿El número es mayor que 0?

↓ **Sí**

El número es positivo

Además de diagramas, podemos escribir los condicionales en forma de **pseudocódigo**, siendo esta una forma de escribir las instrucciones de un programa en un lenguaje similar al humano.

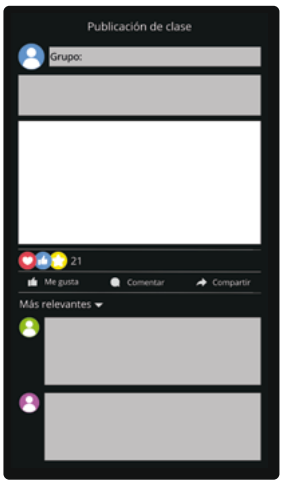
Por ejemplo, el diagrama anterior se puede escribir así:

```
Si (numero > 0) entonces
        Escribir "El número es positivo"
Fin_si
```

En este pseudocódigo, usamos "Si" para introducir una condición, "entonces" para el bloque de código que se ejecuta si la condición es verdadera. La palabra "Fin_si" se utiliza para indicar el final del programa o el bloque de código.

Anexos

Anexo 3.1



Anexo 3.2

Esto es importante porque señala claramente dónde termina la secuencia de instrucciones, facilitando la comprensión de dónde comienza y termina cada parte del algoritmo. Además de "Si", podemos usar otras formas para verificar nuestras condiciones:

- **Si no, si (elif):** Se utiliza para verificar condiciones adicionales si la condición anterior no se cumple.
- **Si no (else):** Opcionalmente, se utiliza al final para ejecutar un bloque de código si ninguna de las condiciones anteriores es verdadera.

Los condicionales son fundamentales en la programación porque permiten que los programas tomen decisiones basadas en condiciones específicas. Usar diagramas nos ayuda a visualizar el proceso, y el pseudocódigo nos permite planificar y entender la lógica antes de escribir el código en un lenguaje de programación específico.

Manos a la obra
Desconectadas

Esta sección corresponde al 67% de avance de la sesión

Para empezar, reúnete en equipos siguiendo las instrucciones de tu docente y realicen las siguientes actividades:

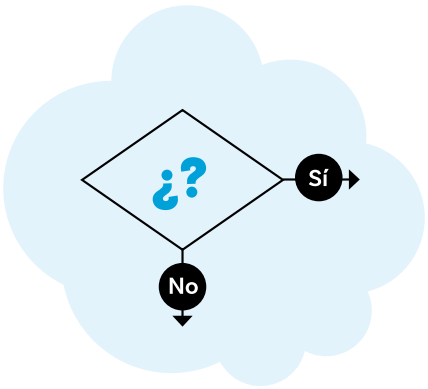
- **Actividad A:** Tu docente debe entregar tres copias del *Anexo 3.1* a cada grupo. Utilizando el formato del *Anexo 3.1*, creen los pseudocódigos para resolver los siguientes tres retos.

  1. Preguntar un número y determinar si es positivo, negativo o cero.
  2. Determinar si un número es o no es par.
  3. Identificar cuál es el número mayor entre dos números o si estos son iguales.

  Luego, péguenlo en la pared para que los demás grupos lo vean.

Dejen comentarios y reacciones en los pseudocódigos de los otros grupos. Al terminar, socialicen los comentarios y reacciones que obtuvieron en sus algoritmos.

- **Actividad B:** Ahora dibujen un diagrama de flujo que represente las siguientes lógicas condicionales. Tengan en cuenta que crear diagramas de flujo ayudará a visualizar cómo los condicionales controlan el flujo de un programa, reforzando su comprensión de la lógica condicional.



1. Si la temperatura es mayor a 30 grados, muestra el mensaje: “Hace calor”. De lo contrario, muestra “La temperatura es agradable”.
2. Si la edad es mayor o igual a 18, muestra “Puede votar”. De lo contrario, muestra “No puede votar”.
3. Si la calificación es mayor o igual a 60, muestra “Aprobado”. De lo contrario, muestra “Reprobado”.

Si al finalizar cuentan con tiempo, proponemos estos retos adicionales:

a. Creen el pseudocódigo para un algoritmo que lea un número y determine si es o no múltiplo de 3.

b. Creen el pseudocódigo de un algoritmo que determine si un(a) estudiante aprueba o reprueba un curso, sabiendo que aprobará si su promedio de tres calificaciones es mayor o igual a 70 y que reprueba en caso contrario.

Una vez hayan completado todos los retos, comenten con otros grupos sus ejercicios y encuentren similitudes y diferencias. Guarden sus respuestas, pues las van a necesitar en la próxima sesión.

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 reconocer el uso de las estructuras condicionales en diferentes situaciones?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

2. ¿Puedes diseñar soluciones a situaciones problemas, utilizando correctamente los condicionales?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", realiza en compañía de un compañero o compañera que haya completado todas las actividades, alguno de los ejercicios adicionales planteados en la sesión Manos a la obra. Pídele que explique paso a paso qué hacen y por qué. Además, consulta con tu docente las dudas que tengas.

Aprovecha este espacio final para hacer un esquema en el que resumas algo de lo que aprendiste. En este punto podrías realizar un comparativo sobre cómo resolverías los retos en un diagrama basado en bloques como *MakeCode*.

¿Podrías resolver todos los retos?
¿Con qué limitaciones te enfrentarías?

Grado 9° **Guía 4**

TIC

Sesión 4

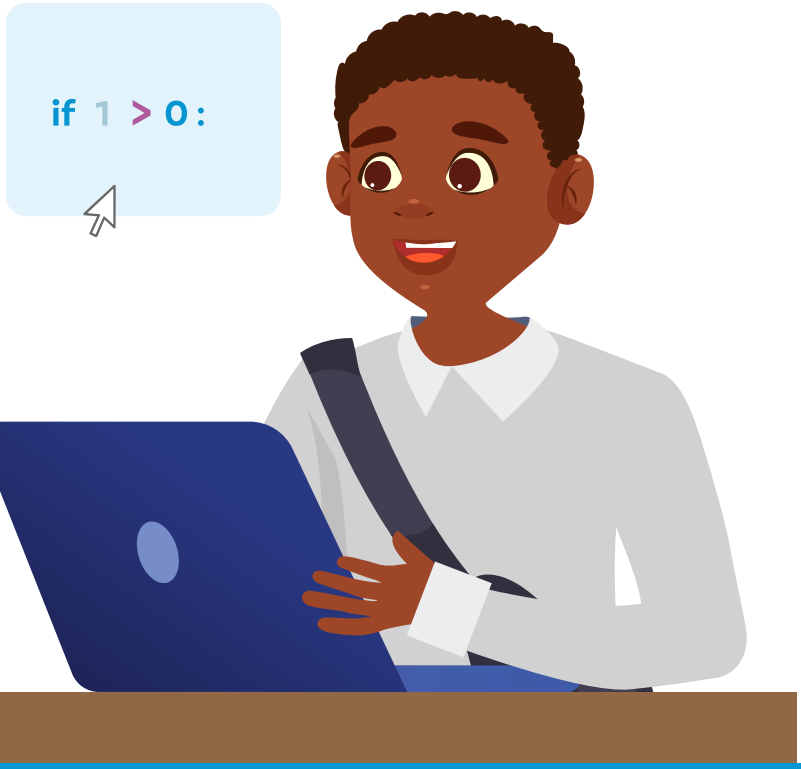
Aprendizajes esperados

Al final de esta sesión verifica que puedas:

- Comprender la sintaxis de condicionales en *Python*.
- Explicar cómo se programan condicionales en *Python*.
- Aplicar condicionales para resolver problemas prácticos en verifica que puedas.

Duración sugerida

60% 30% 10%



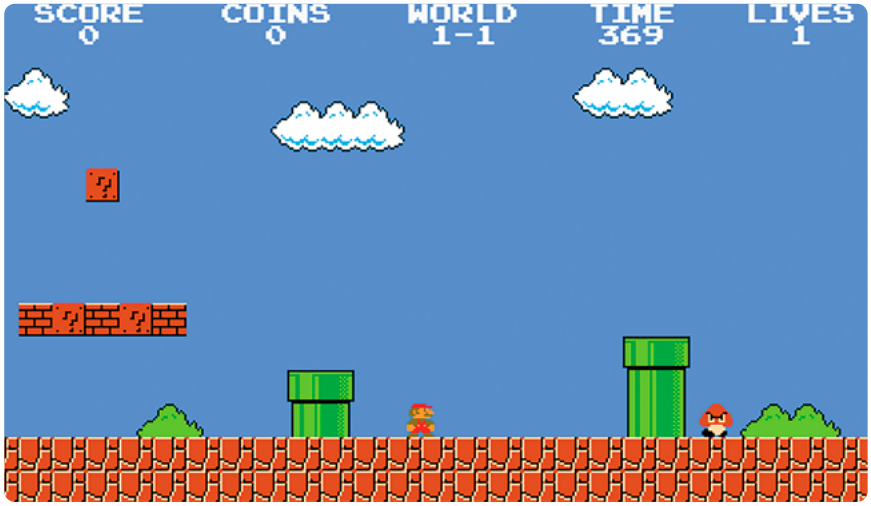
Material para la clase

- Computador con *Python*, respuestas a las actividades de la sesión anterior.

Lo que sabemos, lo que debemos saber

Esta sección corresponde al 60% de avance de la sesión

Casi todas las personas hemos jugado algún tipo de videojuego alguna vez, desde juegos simples en el celular hasta algunos más avanzados en consolas y computadoras. ¿Alguna vez te has preguntado cómo se crean estos juegos? Detrás de cada juego, hay un mundo de programación y lógica que permite que los personajes se muevan, los niveles cambien y la historia se desarrolle.



Ahora que sabemos construir algoritmos utilizando pseudocódigo, podemos utilizar esta habilidad para convertir nuestras ideas en código funcional que nuestra computadora pueda entender y ejecutar. El pseudocódigo nos permite planificar y organizar la lógica de nuestros programas de una manera que es fácil de leer y comprender. Al traducir esta lógica a un lenguaje de programación como *Python*, aprovechamos su sintaxis clara y directa para implementar las instrucciones de nuestro pseudocódigo.

```
if numero > 10:
    print("El número es mayor que 10.")
```

¿?

¿Qué crees que hace este código?

¿Qué crees que pasará si el número es 10? ¿Y si es 15?

Prueba diferentes números. ¿Los resultados coinciden con tus predicciones?

¿Qué función tiene la palabra if en el código?

Veamos cómo es la estructura de los condicionales en *Python*. La estructura de la condición *if* es la siguiente:

If (condición): Aquí van las órdenes que se ejecutan si la condición es cierta. Al verificar si la condición es cierta se tendrán dos resultados:

Si la condición se cumple se ejecutarán las órdenes dadas. En este caso diremos que el resultado es *True* (verdadero).

- Si la condición no se cumple entonces no se ejecutarán las órdenes dadas. En este caso diremos que el resultado es *False* (falso).

  Escribe el siguiente código en tu computador:

```
if temperatura > 30:
    print("Hace calor.")
```

¿?

¿Qué pasa si eliminas los dos puntos (:)?
¿Y si escribes if en mayúsculas?
¿Qué pasa si eliminas los espacios?

- Haz algunas pruebas y observa qué cambios afectan tu programa y cuáles no.

Ahora observa el siguiente código. ¿Qué crees que hace?

```
numero = int(input("Ingresa un número: "))
if numero > 0:
print("El número es positivo.")
```

Copia el código y ejecútalo en Python.
Observa qué sucede cuando intentas correr el código.
¿Te aparece algún error?
¿Por qué crees que ocurre?

Ahora veamos el mismo ejemplo, pero escrito de manera distinta:

```
numero = int(input("Ingresa un número: "))
if numero > 0:
    print("El número es positivo.")
```

Copia este código corregido y ejecútalo.
Compara los resultados con el código anterior. ¿Qué diferencias notas?

Algunas cosas que debes tener en cuenta a la hora de escribir en *Python* es que la primera línea de nuestro condicional siempre debe terminar con dos puntos (:), además de tener en cuenta la indentación.

La **indentación** es el pequeño espacio delante de las órdenes a ejecutar, es generalmente de cuatro espacios o un tabulador, y se usa para definir este bloque de código. Todo el código indentado o con sangría debajo de la condición pertenece al mismo bloque

y se ejecutará solo si la condición es verdadera. La correcta **indentación** no solo es crucial para la sintaxis de *Python*, sino que también mejora la legibilidad del código, haciendo claro qué partes del código dependen de cada condición.

Al conocer la estructura a seguir en *Python* para escribir los condicionales podemos escribir el ejemplo visto anteriormente de la siguiente forma:

```
if (numero > 0):
   print("El número es positivo")
```

Al copiar código desde la guía, verifica que los espacios (indentación) al inicio de cada línea estén bien. Si la indentación no se respeta, el programa puede mostrar errores.

De igual manera que en el pseudocódigo, a nuestro algoritmo en *Python* le podemos añadir más condiciones como *else* (si no) y *elif* (si no si). Incorporemos estas nuevas condiciones al algoritmo anterior de tal forma que nos dé más información sobre el número que tengamos:

```
if (numero > 0):
   print("El número es positivo")
elif (numero == 0):
   print("El número es positivo")
else:
  print("El número es positivo")
```

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.

Manos a la obra

Conectadas

Esta sección corresponde al 90% de avance de la sesión

Siguiendo las instrucciones de tu docente, reúnete con un grupo de compañeras y compañeros de clase y resuelvan los siguientes ejercicios que requieren el uso de *Python*. Luego pongan a prueba sus códigos con diferentes números y comparen sus resultados con otros grupos. Recuerden que para cada código necesitan solicitar un número de entrada para poder ejecutarlo. Estos son los retos:

- Creen un programa que pida al usuario que ingrese dos números y determine cuál de los dos es mayor. Si son iguales, el programa debe indicarlo.

- Un local de juegos cobra por el alquiler de consolas de videojuegos dependiendo de la cantidad de horas de uso.

Tabla 1. Precio por horas

Precio por hora	Cantidad de horas
$12.000	Menos de 3 horas
$10.800	Entre 3 y 6 horas
$9.600	Más de 6 horas

  Diseñen un algoritmo que determine cuánto debe pagar un(a) cliente por el alquiler de un videojuego, conociendo el tiempo de alquiler en horas.

- Diseñen un programa en *Python* que lea un número y determine si es o no múltiplo de 3.

- Programen un algoritmo que permita determinar si un(a) estudiante aprueba o reprueba un curso, sabiendo que aprobará si su promedio de tres calificaciones es mayor o igual a 70; reprueba en caso contrario.

- Creen un programa que le pida al usuario la temperatura (en grados Celsius). Si la temperatura es menor a 20 grados, imprime “Hace frío”.

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 sintaxis de condicionales en *Python*?

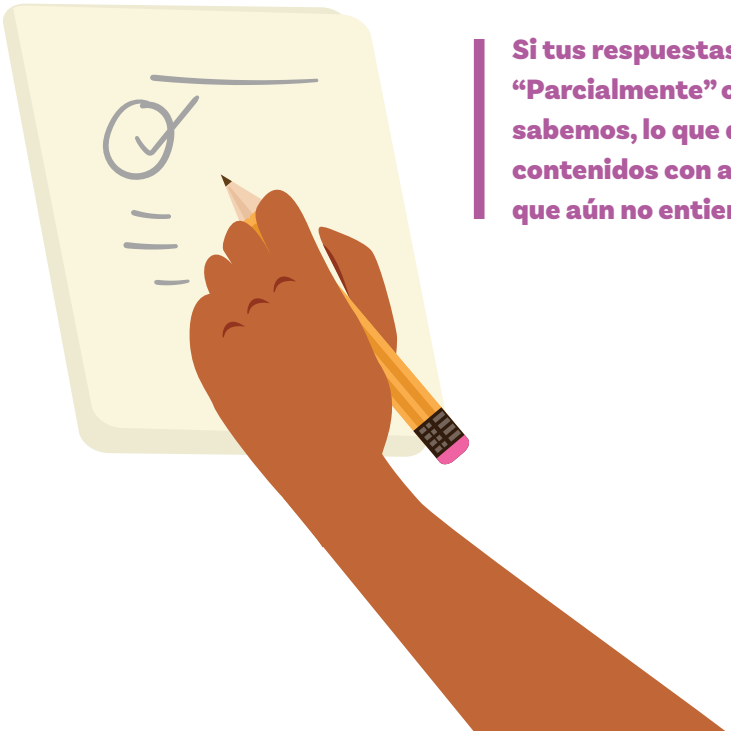
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

2. ¿Puedes explicar cómo se programan condicionales en *Python*?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

3. ¿Puedes aplicar condicionales para resolver problemas prácticos en *Python*?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no



Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", regresa a la sección Lo que sabemos, lo que debemos saber y lee cuidadosamente estos contenidos con ayuda de tu docente. Toma nota de las cosas que aún no entiendes y consulta en internet o a tu docente.

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:

¿Puedes pensar en otros ejemplos de programas que podrías hacer?
¿Qué se te dificulta en el proceso de programación?
¿Revisaste las soluciones que propusiste en la sesión 3 a los mismos retos? ¿Cómo se compara la programación con el pseudocódigo?

Aprovecha este espacio final para hacer un esquema en el que resumas algo de lo que aprendiste, por ejemplo, colocando las definiciones de palabras y ejemplos.

Sesión 5

Aprendizajes esperados

Al final de esta actividad verifica que puedas:

- Utilizar del ciclo “mientras que” en el desarrollo de videojuegos.
- Integrar ciclos y condicionales para controlar el flujo de un juego.
- Aplicar la sintaxis del ciclo “mientras que” en pseudocódigo.

Material para la clase

- Acceso a computador con acceso a Python.
- Un dado.
- Anexo 5.1

Duración sugerida

40% 40% 20%

Lo que sabemos, lo que debemos saber

Esta sección corresponde al 40% de avance de la sesión

En el desarrollo de videojuegos, el uso de ciclos es fundamental para crear bucles de juego que permitan acciones repetitivas, como actualizar la pantalla, detectar entradas del usuario y manejar la lógica del juego. Uno de los ciclos más importantes es el ciclo "mientras que" el cual repite un bloque de código siempre que una condición específica sea verdadera.

Imagina que estás desarrollando un videojuego en el que un personaje se mueve continuamente por la pantalla hasta que el jugador decide detenerlo. Aquí, los ciclos "mientras que" y los condicionales trabajan juntos para controlar el flujo del juego. Por ejemplo, el ciclo "mientras que" se puede utilizar para mantener el juego en ejecución, verificando continuamente si el jugador ha dado una instrucción para finalizar el juego. Dentro de este ciclo, los condicionales se utilizan para determinar acciones específicas basadas en las entradas del jugador, como mover el personaje hacia la izquierda, derecha, arriba o abajo.

Reúnete con un compañero o compañera y realicen la siguiente actividad donde simularán el siguiente bucle: Saltos de tijera si el dado muestra un número menor a 6.

Comienza haciendo saltos de tijera. Tu compañero(a) lanzará el dado. Mientras salga 1, 2, 3, 4 o 5, continúa haciendo saltos de tijera. Cuando salga un 6, deja de hacer saltos de tijera. Vamos a intentarlo de nuevo, esta vez simulando otro bucle: Flexiones de brazos tres veces.

Tu compañero(a) lanzará el dado. Haz una flexión por tres lanzamientos consecutivos del dado. No importa qué número salga en el dado, solo que dejes de hacer flexiones después de tres lanzamientos.

La sintaxis básica del ciclo "mientras que" en pseudocódigo es la siguiente:

```
Mientras que (condición) Bloque de código a repetir
mientras la condición sea verdadera
        Fin_Mientras
```

El ciclo comienza evaluando la condición especificada. Si la condición es verdadera, el bloque de código dentro del ciclo se ejecuta. Después de ejecutar el bloque de código, la condición se vuelve a evaluar. Si la condición todavía es verdadera, el ciclo se repite. Este proceso continúa hasta que la condición se vuelva falsa, momento en el cual la ejecución del ciclo "mientras que" se detiene y el programa continúa con la siguiente instrucción después del ciclo.

Por ejemplo, si queremos contar desde 1 hasta 5 usando un ciclo "mientras que", podemos escribir el siguiente pseudocódigo:

```
contador = 1
Mientras que (contador <= 5)
          Escribir contador
          contador = contador + 1
Fin_Mientras
```

El ciclo "mientras que" en *Python*, conocido como *while*, sigue la misma lógica que su contraparte en pseudocódigo. Veamos cómo se traduce el ejemplo anterior de pseudocódigo a *Python*:

```
contador = 1
while (contador <= 5):
    print(contador)
    contador = contador + 1
```

¿?

¿Qué sucede si cambias el contador por otro número?
¿Cuál será el input si sumamos 2 al contador en vez de sumar 1?
¿Qué ocurre si eliminas el símbolo =?

Al igual que en los condicionales, al usar *while* es necesario emplear la indentación para definir los bloques de código, junto con los dos puntos (:) después de especificar la condición.

Manos a la obra

Conectadas

Esta sección corresponde al 80% de avance de la sesión

Organízate en grupo, con 3 compañeras o compañeros, siguiendo las indicaciones de tu docente. En esta actividad crearán un juego de adivinanzas numéricas en el que competirán contra la computadora para adivinar un número.

Objetivos:

- Construir un juego de adivinanzas numéricas, en el que el usuario selecciona un rango.
- Digamos que el usuario seleccionó un rango, es decir, de A a B, donde A y B son números enteros (int).
- El computador seleccionará un número entero aleatorio y el usuario deberá adivinarlo en el menor número de intentos posible.

Instrucciones:

- El usuario introduce el límite inferior y el límite superior del intervalo.
- El computador genera un número entero aleatorio entre el rango y lo almacena en una variable para futuras referencias. Utiliza el siguiente código como ayuda:

```
import random as ran

rangoA = int(input("Ingresa el valor de A"))

rangoB = int(input("Ingresa el valor de B"))

numero_aleatorio = ran.randint(rangoA, rangoB)
```

Anexo

Anexo 5.1

Nombre: _____ Fecha: _____

¿En qué tipo de situaciones usarías un ciclo "mientras que" en un programa? *Escribe una o dos oraciones.*

¿Cuál fue el mayor reto que encontraste hoy al trabajar con ciclos "mientras que"? ¿Cómo lograste resolverlo? *Describe brevemente.*

Reflexión rápida:
¿Qué ocurre en un ciclo "mientras que" cuando la condición nunca se vuelve falsa? *(Marca la opción correcta)*

- El ciclo se detiene automáticamente.
- El ciclo se ejecuta indefinidamente.
- La condición cambia sola.

Nombre: _____ Fecha: _____

¿En qué tipo de situaciones usarías un ciclo "mientras que" en un programa? *Escribe una o dos oraciones.*

¿Cuál fue el mayor reto que encontraste hoy al trabajar con ciclos "mientras que"? ¿Cómo lograste resolverlo? *Describe brevemente.*

Reflexión rápida:
¿Qué ocurre en un ciclo "mientras que" cuando la condición nunca se vuelve falsa? *(Marca la opción correcta)*

- El ciclo se detiene automáticamente.
- El ciclo se ejecuta indefinidamente.
- La condición cambia sola.

- Para adivinar repetidamente se requiere utilizar un ciclo while.
- Si el o la usuario adivina un número mayor que un número seleccionado aleatoriamente, recibe el mensaje "¡Inténtalo de nuevo! Has ingresado un número demasiado alto".
- Si el o la usuario adivina un número que es menor que un número seleccionado al azar, el usuario obtiene el mensaje "¡Inténtalo de nuevo! Has ingresado un número demasiado pequeño".
- Y si el o la usuario ha acertado el número, recibe un mensaje de "¡Felicitaciones! Ha acertado el número".

La computación y la sociedad

Lenguajes de computador, como Python, permiten resolver muchos problemas de la sociedad usando la computación.

Estos problemas se presentan en muchos campos, en la educación, por ejemplo, desarrollando aplicativos para enseñar y en la salud, desarrollando aplicativos para guardar la información de los pacientes.

En la agricultura para, por ejemplo, controlar sistemas de riego o la dosificación de fertilizantes.

Cada vez hay más contextos en los que la computación ayuda a resolver problemas y cada vez se necesitan más personas que sepan de computación.

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 utilizar el ciclo “mientras que” en el desarrollo de videojuegos?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

2. ¿Puedes integrar ciclos y condicionales para controlar el flujo de un juego?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

3. ¿Puedes aplicar la sintaxis del ciclo “mientras que” en pseudocódigo?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

> **Si tus respuestas a las preguntas anteriores fueron “Parcialmente” o “Aún no”, retoma los aprendizajes de la sesión, regresa a los ejercicios y pide acompañamiento a tu docente.**

Antes de irte, completa el tiquete de salida del *Anexo 5.1.* Este te ayudará a reflexionar sobre lo aprendido antes de pasar a la siguiente guía.

Anexo 1.1 Uso de *Python* en *Anaconda*

Escanea el siguiente Qr para visualizar el anexo:

Enlace

Anexo 1.2 Uso de *Python* en línea

Escanea el siguiente Qr para visualizar el anexo:

Enlace

Anexo 1.2 Uso de *Google Colab*

Escanea el siguiente Qr para visualizar el anexo:

Enlace

Anexo 3.1 Actividad

Anexo 3.2 Reacciones

Anexo 5.1 Tiquete de salida

Nombre: _____ **Fecha:** _____

¿En qué tipo de situaciones usarías un ciclo "mientras que" en un programa? *Escribe una o dos oraciones.*

_____

_____

_____

¿Cuál fue el mayor reto que encontraste hoy al trabajar con ciclos "mientras que"? ¿Cómo lograste resolverlo? *Describe brevemente.*

_____

_____

_____

Reflexión rápida:

¿Qué ocurre en un ciclo "mientras que" cuando la condición nunca se vuelve falsa? *(Marca la opción correcta)*

- ◯ El ciclo se detiene automáticamente.
- ◯ El ciclo se ejecuta indefinidamente.
- ◯ La condición cambia sola.

Nombre: _____ **Fecha:** _____
¿En qué tipo de situaciones usarías un ciclo "mientras que" en un programa? *Escribe una o dos oraciones.*

_____

_____

_____

¿Cuál fue el mayor reto que encontraste hoy al trabajar con ciclos "mientras que"? ¿Cómo lograste resolverlo? *Describe brevemente.*

_____

_____

_____

Reflexión rápida:

¿Qué ocurre en un ciclo "mientras que" cuando la condición nunca se vuelve falsa? *(Marca la opción correcta)*

- ◯ El ciclo se detiene automáticamente.
- ◯ El ciclo se ejecuta indefinidamente.
- ◯ La condición cambia sola.

TIC

Apoya:

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}