Funciones en Python

Grado 10° Guía 5

TIC

Estudiantes

Apoya:

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}

Funciones en Python

Grado 10° Guía 5

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 10° Guía 5

TIC

Guía de íconos

 Lógica, programación y depuración

Aprendizajes de la guía

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

 Aplicar la descomposición, abstracción y pensamiento algorítmico para resolver retos y tareas.

 Diferenciar entre declarar, inicializar y asignar variables.

 Interpretar los mensajes de error y usarlos para identificar y corregir los errores en programas (lógica, sintaxis).

 Automatizar tareas simples relacionadas a otras áreas de conocimiento.

Resumen de la guía

Esta guía propone 5 sesiones de trabajo orientadas a continuar aprendiendo a programar en *Python*. De igual manera, permite afianzar los conceptos de manejo de listas, crear subrutinas e identificar errores. Se presentan una serie de actividades conectadas y desconectadas y la guía finaliza con un reto de programación utilizando arreglos de dos dimensiones.

Resumen de las sesiones

Sesión 1

Se retoman conceptos fundamentales de la programación en *Python* como la inicialización y asignación de variables. Además, se aprende a crear funciones para descomponer y mejorar la legibilidad de los programas.

Sesión 2

Se aprenden buenas prácticas de programación y sintaxis. Se reflexiona sobre la importancia de tener un código fácil de entender y reutilizar en el futuro. Se finaliza con una investigación y creación de un decálogo del aula para seguir las buenas prácticas de programación.

Sesión 3

Se profundiza en el concepto de funciones y se aprende a utilizar argumentos y retornar valores. Se desarrollan programas que utilizan funciones, diferentes tipos de variables y operaciones.

Nota

Parte de esta guía está inspirada en el currículo de Ciencias de la Computación creado por el Centro Nacional de Educación en Computación del Reino Unido (NCCE) para los niveles KS3 y KS4. https://teachcomputing.org/curriculum

Sesión 4

Se aprende acerca de las listas de dos dimensiones (matrices 2D) en *Python* y se realizan operaciones básicas de lectura, inserción y reemplazo.

Sesión 5

Se propone el desarrollo de un programa que permita jugar triqui (o tres en línea) en el computador, utilizando los conceptos aprendidos hasta el momento.

Conexión con otras áreas

A continuación, se brindan puntos de conexión de los temas abordados en las sesiones con otras áreas:

Matemáticas

- El manejo de arreglos bidimensionales en *Python* (Sesión 4) puede relacionarse con conceptos de álgebra lineal, como matrices y operaciones como la transposición o el cálculo de determinantes.

Ciencias Naturales

- Se pueden programar modelos simples que representen fenómenos naturales, como cambios en un ecosistema, reacciones químicas o simulaciones de movimiento en física usando fórmulas.

Ciencias Sociales

- Creación de programas para analizar datos como encuestas, poblaciones o estadísticas relacionadas con fenómenos sociales.

Grado 10° Guía 5

TIC

Sesión 1

Aprendizajes esperados

Al final de esta sesión se espera que puedas:

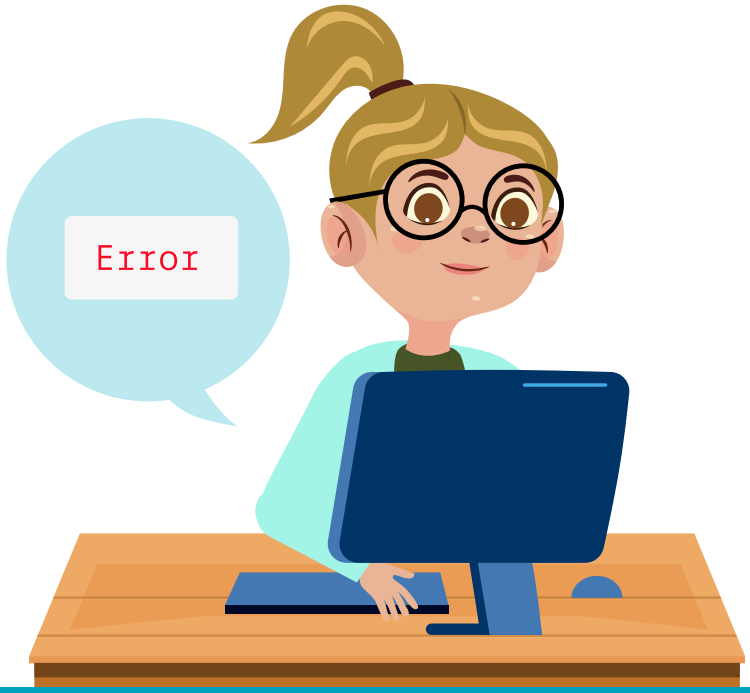
- Descomponer un programa en diferentes funciones.
- Predecir el resultado de un programa en *Python* y modificarlo.
- Interpretar mensajes de error de lógica y sintaxis e identificarlos en programas.

Material para la clase

- ◯ Computador con acceso a *Python*.
- ◯ Anexo 1.1

Duración sugerida

15% 70% 15%



Lo que sabemos, lo que debemos saber

Esta sección corresponde al 15% de avance de la sesión

En esta guía retomarás la programación en *Python*. Por eso, antes de comenzar, comenta con tu docente y tus compañeras y compañeros las siguientes preguntas:

> *¿Qué recuerdas de la programación en Python? ¿Qué te ha llamado la atención? ¿Qué se te ha dificultado? ¿En qué se parece y se diferencia a otros lenguajes de programación (como Scratch, MakeCode, y otros)?*

En las próximas dos sesiones vas a aprender más sobre las buenas prácticas de programación. Esto es, todos los consejos que debes aplicar para que tu código sea más fácil de leer y reutilizar en futuras ocasiones.

Lo primero que vas a aprender es a dividir tu código en funciones. Pero antes, piensa:

> *¿Has escuchado a tu docente hablar del pensamiento computacional? ¿Puedes nombrar las habilidades que se asocian al pensamiento computacional?*

A continuación, se presenta la definición de tres habilidades del pensamiento computacional. Intenta nombrarlas y revisa con tu docente su significado.

Opciones de respuestas:
Depuración – Descomposición – Iteración – Algoritmos – Abstracción – Recursión

_____: es la técnica de dividir un problema grande en partes más pequeñas y manejables. Al fragmentar un problema complejo, es más fácil concentrarse en resolver cada parte por separado, lo que simplifica el proceso de desarrollo.

_____: implica enfocarse en los detalles importantes de un problema y omitir lo innecesario, lo que permite simplificar el problema y concentrarse en lo que realmente importa.

_____: se refiere a la capacidad de diseñar un conjunto de instrucciones claras, ordenadas y precisas para resolver un problema.

En el mundo de la programación, las **funciones** son una forma de organizar el código para que sea más fácil de leer, reutilizar y mantener. Una función es un bloque de código que realiza una tarea específica. Para comenzar explorando las funciones, observa el siguiente fragmento de código y utiliza el recuadro para escribir detalladamente lo que crees que hace:

```
def saludar():
    print("Hola, soy una función")

saludar()
```

Escribe el código en *Python* y pruébalo.

¿Se comportó como esperabas?
¿Por qué el print() se encuentra algunos espacios hacia la derecha?
¿Qué pasa si borras la última línea?¿Por qué?

Como podrás notar, si eliminas la línea de código saludar(), no aparece nada en pantalla. Lo que pasa es que, es importante llamar las funciones para que se ejecuten.

Piénsalo así, tú tienes muchas funciones "programadas" en tu cuerpo. Puedes alimentarte, hacer tareas, divertirte o dormir. Sin embargo, tienes que esperar a que sea el momento adecuado para ejecutarlas. Esto también pasa con los programas.

Observa este otro ejemplo.

¿Puedes predecir lo que va a pasar?

```
def saludar():
    print("Hola, soy una función")

def preguntar_nombre():
    nombre = input("¿Cómo te llamas?")
    print("mucho gusto,", nombre)

saludar()
preguntar_nombre()
```

Discute con tu clase lo que se va a ver en pantalla.

Ahora analiza el siguiente ejemplo y escribe lo que va a suceder:

```
## Funciones
def saludar():
  print("Hola, soy una función")
def preguntar_nombre():
   nombre = input("¿Cómo te llamas?")
   print("mucho gusto,", nombre)
def despedirse():
   print("Nos vemos!")
```

```
## Llamados
despedirse()
saludar()
preguntar_nombre()
```

¿Puedes pensar en una mejora para el código? ¿Cuál?

Este ejemplo ilustra el concepto de secuencia. Una secuencia en programación es un conjunto de instrucciones que se ejecutan en el orden en que están escritas, una tras otra. Por ejemplo, la sección **Llamados** del ejemplo anterior. El orden es importante porque *Python* sigue las instrucciones en secuencia, de arriba hacia abajo.

Glosario

Pensamiento computacional: forma de abordar problemas y desafíos utilizando conceptos y técnicas inspirados en la informática y la programación.

Funciones: un bloque de código reutilizable diseñado para realizar una tarea específica.

Manos a la obra

Conectadas

Esta sección corresponde al 85% de avance de la sesión

¿Te gusta la música? ¿Puedes pensar en cuántas partes se descompone una canción?

En esta actividad vas a poner en práctica el uso de funciones, mientras programas un karaoke. Para hacerlo, reúnete en equipos de dos o tres personas, según las instrucciones de tu docente y completen las siguientes actividades.

1. Lean el código para predecir qué va a pasar cuando se ejecute.

```
from time import sleep

def estrofa_1():
  print("Colombia, tierra querida, himno de fe y armonía.")
  sleep(3)
  print("Cantemos, cantemos todos grito de paz y alegría.")
  sleep(3)
  print("Vivemos, siempre vivemos a nuestra patria querida.")
  sleep(3)

def estribillo():
  print("Su suelo es una oración y es un canto de la vida.")
  sleep(3)

def coro():
  print("Cantando, cantando, yo viviré.")
  sleep(3)
  print("Colombia, tierra querida.")
```

```
[ ] estrofa_1()
    estribillo()
```

Enlace

Archivos de esta sesión.

Discutan en grupo y usen el espacio para escribir su predicción de lo que va a suceder.

Ahora ejecuten el código en el editor. Pueden transcribirlo o acceder al enlace o QR de ***Archivos para esta sesión***, donde encontrarán los siguientes archivos que podrán usar para ejecutar el código:

- karaoke.ipynb – Si están utilizando cuadernos interactivos en Colab o Jupyter Notebook
- karaoke.py – Si están utilizando otro editor de python

¿Acertaron en su predicción? ¿El código se comportó como esperaban?

Escriban lo que observan.

Al ejecutar el código, ¿qué se muestra en la pantalla?

_____

_____

¿En qué línea está la llamada a la subrutina para estrofa_1() ?

_____

_____

Enlace

Video Colombia tierra querida (En lengua de señas)

¿Por qué el programa no muestra el resto de la canción?

Busquen la función **sleep(3)** y reemplácenla por **sleep(7).** ¿Qué pasa?

¿Qué creen que está haciendo la subrutina **sleep()**?

Quiten el código de la línea 1 (**from time import sleep**) y ejecuten el código. ¿Qué sucede?

Asegúrense de volver a colocar la línea de código antes de continuar.

2. Ahora modifiquen el código para que aparezcan todas las frases de la canción. Pueden buscar la letra si la necesitan o su docente puede permitir que escuchen la canción para verificar que su código de karaoke funciona. Para ir a la canción pueden utilizar el enlace o el QR titulado como Colombia tierra querida.

   En este punto, verifiquen las pausas. La letra debe coincidir, en lo posible, con la música.

3. Cuando logren el reto, intenten provocar errores en su código. Eliminen algún símbolo, cambien la forma de escribir las funciones, o modifiquen los espacios de la indentación. Esto es muy útil para que se familiaricen con los mensajes de error y sus causas.

   Experimenten con varios cambios y vuelvan siempre a corregir su código. Pueden guardar una copia para tener mayor tranquilidad de que podrán deshacer los cambios.

4. Ahora ustedes van a crear su propia canción en estilo karaoke: elijan una canción que les guste como grupo y busquen la letra.

Deben descomponer la canción en sus versos y coro, crear una subrutina para cada parte de la canción y utilizar la función sleep() para que la letra aparezca en el momento adecuado.

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 descomponer un programa en diferentes funciones?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

2. ¿Puedes predecir el resultado de un programa en *Python* y modificarlo?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

3. ¿Puedes interpretar mensajes de error de lógica y sintaxis e identificarlos en programas?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

Anexo

Anexo 1.1

Antes de cerrar la sesión, comenta brevemente con el resto de tu clase:

> *¿Identificaste formas en las que se pueden mejorar los códigos de karaoke? ¿Qué podrías hacer para que el código fuera menos repetitivo? ¿Qué estrategias has utilizado para corregir errores? ¿Por qué crees que se dice que las funciones ayudan a que el código sea más fácil de leer?*

Para ayudarte a afianzar lo aprendido y a consolidar los aprendizajes que aún no están claros, completa el *Anexo 1.1.*

Grado 10° | Guía 5

TIC

Sesión 2

Aprendizajes esperados

Al final de esta sesión verifica que puedas:

- Usar identificadores y nombres significativos en variables.
- Distinguir entre inicialización y asignación de variables en *Python*.
- Demostrar el uso adecuado de las convenciones de nomenclatura en la programación en *Python*.

Material para la clase

- Papel, marcadores para hacer un esquema.
- Computador con acceso a *Python*.

Duración sugerida

60% 30% 10%

Lo que sabemos, lo que debemos saber

Esta sección corresponde al 60% de avance de la sesión

En guías anteriores has interactuado con *Python* y has aprendido a programar operaciones, condicionales, y a utilizar diferentes tipos de variables como los números, cadenas de texto y las listas.

En esta sesión retomarás el uso de variables y aprenderás dos conceptos importantes: inicialización y asignación de variables.

Mira el siguiente ejemplo.

```
[ ]  def convertir_cm_pul():
       centimetros = float (input ("Ingresa la longitud en centímetros:"))
       if centimetros > 0:
         pulgadas = centimetros / 2.54

       print("el resultado es:", pulgadas)
```

```
convertir_cm_pul()
```

¿? *¿Qué hace el código?*

Ahora piensa y discute con la persona a tu lado.

¿?

¿Qué imprimirá el código si el/la usuario(a) ingresa el número 100?
¿Qué imprimirá el código si el/la usuario(a) ingresa el número 1?
¿Qué imprimirá el código si el/la usuario(a) ingresa el número 0?

La desarrolladora del programa notó que cuando ingresa un número negativo, sale el error que se presenta:

```
Ingresa la longitud en centímetros: -12
---------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-19-d625f9e6d4e9> in <cell line: 1>()
----> 1 convertir_cm_pul()

<ipython-input-17-8442c9c5c061> in convertir_cm_pul()
      4     pulgadas = centimetros / 2.54
      5
----> 6 print("el resultado es:", pulgadas)
UnboundLocalError: local variable “pulgadas” referenced before assigment
```

Esto sucede porque la variable “pulgadas” únicamente existe si se cumple la condición de centímetros > 0. Cuando el programa intenta usar la variable, no la encuentra porque nunca fue asignada.

Una forma de corregir este error es **inicializando** la variable. Revisa el código con detenimiento y encuentra las diferencias:

```
[20] def convertir_cm_pul():
       centimetros = float (input ("Ingresa la longitud en centímetros:"))

       #Inicializar la variable
       pulgadas  = 0

       if centimetros > 0:
         # asignar un nuevo valor a la variable
         pulgadas = centimetros / 2.54

       print("el resultado es:", pulgadas)
```

```
convertir_cm_pul()
```

```
Ingresa la longitud en centímetros: -12
el resultado es: 0
```

Ahora observa otros ejemplos e identifica el error.

```
def saludar():
    print(nombre)
    nombre = input("´¿Cómo te llamas?")

saludar()
```

¿Qué debe cambiar?

```
def jugar():
    jugador_1 = input("Jugador 1, elige piedra, papel o tijera:")
    jugador_2 = input("Jugador 2, elige piedra, papel o tijera:")
    if jugador_1 == "piedra" and jugador_2 == "tijera":
        puntaje = puntaje + 1
    elif jugador_1 == "tijera" and jugador_2 == "papel":
        puntaje = puntaje + 1
    elif jugador_1 == "papel" and jugador_2 == "piedra":
        puntaje = puntaje + 1
print("El puntaje es:",puntaje)

jugar()
```

¿Dónde está el error? ¿Qué debe cambiar?

Las **variables** son espacios en la memoria que usamos para almacenar datos en un programa. Es importante entender las diferencias entre **inicializar** y **asignar** variables:

- **Inicialización:** es cuando asignas un valor a una variable por primera vez. Por ejemplo: tiempo = 0.
- **Asignación:** es cuando cambias el valor de una variable ya existente en cualquier parte del programa. Por ejemplo: tiempo = tiempo + 1.

Darles un manejo correcto a las variables evita la acumulación de errores de lógica.

Ahora mira las *Figuras 1 y 2*.

¿Quién tiene un mejor espacio de trabajo? ¿Por qué?

Figura 1. Lista de archivos en el directorio de Python de Andrea

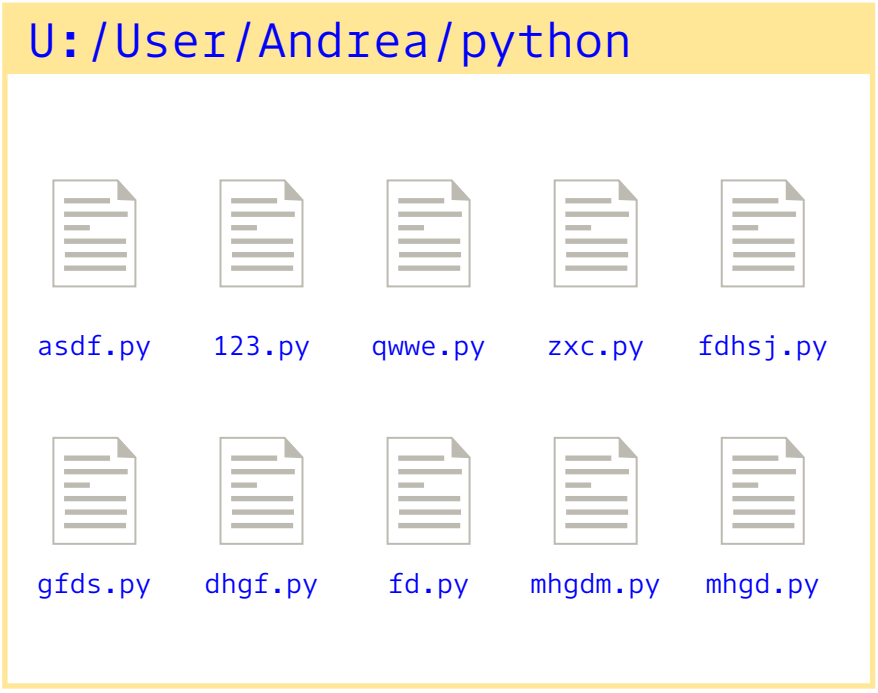
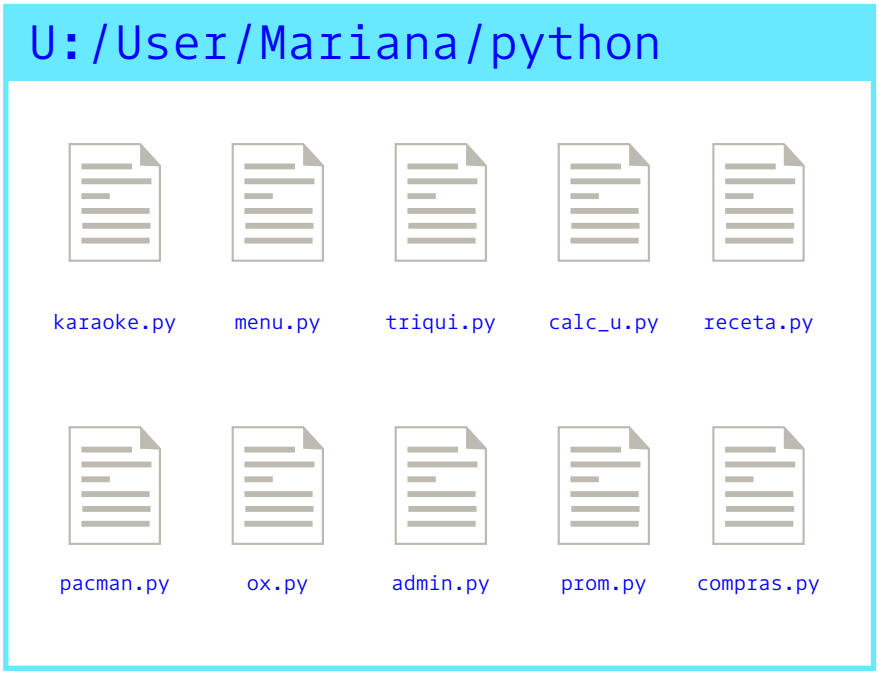


Figura 2. Lista de archivos en el directorio de Python de Mariana



¿Qué beneficios tiene el mejor espacio de trabajo sobre el otro? ¿Cómo sueles nombrar tus archivos?

Aunque cueste creerlo, una de las labores más complicadas para quienes desarrollan programas es nombrar las variables y funciones. Elegir nombres claros y descriptivos es crucial para que el código sea fácil de entender, pueda ser utilizado por otros y sea sencillo de modificar en el futuro.

Para ayudar con esta tarea, los lenguajes de programación han establecido convenciones y reglas que mejoran la claridad y la consistencia al nombrar variables. Estas normas actúan como guías, similares a las normas de la Asociación Estadounidense de Psicología (APA por su sigla en inglés) o al diccionario de la Real Academia de la lengua Española (RAE), pero aplicadas al mundo de la programación.

Glosario

- **Variable:** espacio en la memoria donde se almacena un valor que puede cambiar o usarse durante la ejecución del programa.
- **Inicializar:** dar un valor inicial a una variable antes de usarla.
- **Asignación:** darle un valor a una variable ya existente.

Manos a la obra

Conectadas

Esta sección corresponde al 90% de avance de la sesión

Seguir las convenciones de *Python* no es obligatorio para que tu código funcione. Sin embargo, entre más pronto te acostumbres a usar "la buena ortografía" de este lenguaje, más fácil te resultará en el futuro.

1. En equipo, y de acuerdo con las instrucciones de su docente, creen un programa para tomar los pedidos en un restaurante. Lee la siguiente situación:

   Un restaurante quiere implementar un nuevo chatbot para tomar los pedidos de sus clientes. El chatbot inicia saludando a sus clientes y crea una lista vacía de pedidos. Luego, pide al primer cliente que ingrese su pedido y lo guarda en la lista. Luego se despide del primer cliente y se prepara para saludar al siguiente cliente y así sucesivamente hasta alcanzar cinco pedidos.
   Cuando termina de tomar los pedidos, el chatbot imprime el pedido completo.

2. En grupo, investiguen acerca de las convenciones de nombres y el uso de espacios según la guía de estilos PEP-8.

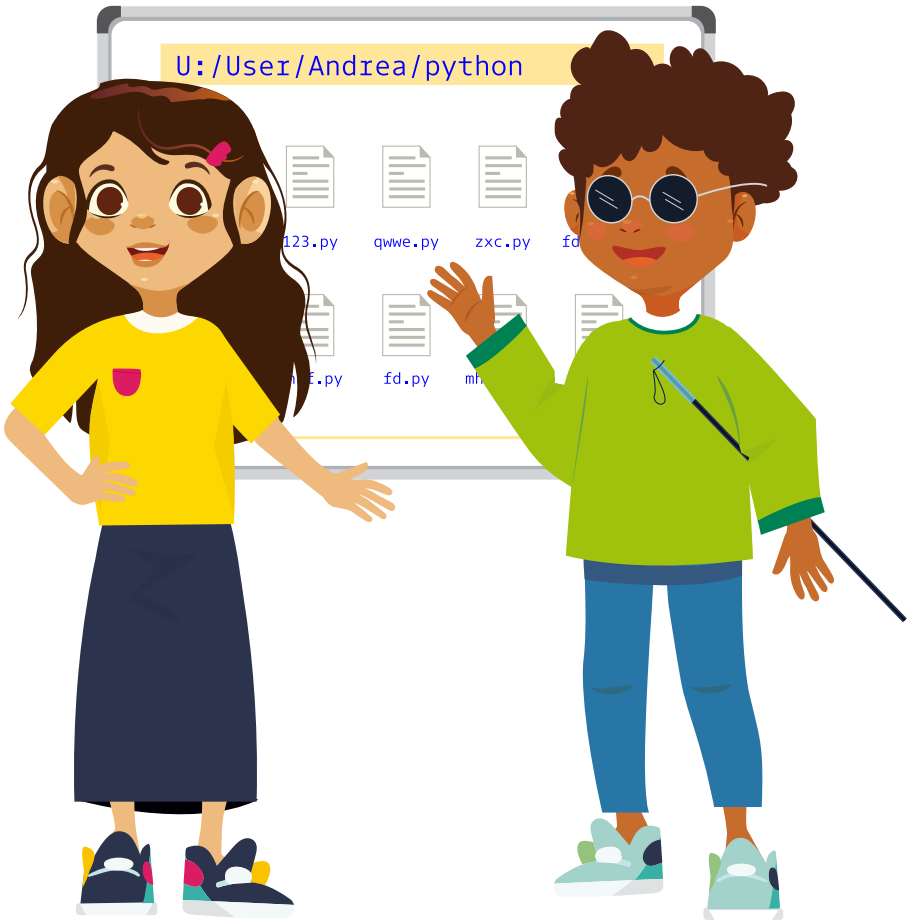
Modifiquen su código anterior para que refleje las buenas prácticas de nombres y espacios que se encuentran en esta guía de estilos.

Pueden utilizar las siguientes fuentes y buscar algunas adicionales:

https://ellibrodepython.com/python-pep8
https://elpythonista.com/pep-8

3. Creen un esquema, cartelera o resumen que les permita recordar fácilmente cómo deben nombrar las funciones, variables y qué símbolos pueden usar.

4. Al finalizar, compartan sus hallazgos y su programa al resto de la clase.



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 usar identificadores y nombres significativos en variables?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

2. ¿Puedes distinguir entre inicialización y asignación de variables en *Python*?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

3. ¿Puedes demostrar el uso adecuado de las convenciones de nomenclatura en la programación en *Python*?

   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

Para cerrar la sesión te proponemos un reto que te permitirá apropiar los contenidos, identificar dudas y consolidar aprendizajes que aún están en proceso. El siguiente fragmento de código, tiene varios errores y no cumple con las buenas prácticas de programación. Señala y anota todo lo que le mejorarías.

```
def funcion1 (Numero1, Numero2):
  return Numero1 + Numero2

def funcion2 (parametro.1):
  return len (parametro.1)

def funcion3 (Numero1, Numero2):
  return Numero1 * Numero2

print(funcion3(0,0))

resultado = funcion1 (1,2)
print("resultado")
```

Al terminar, comparte tus observaciones con el grupo con el que trabajaste previamente y respondan las siguientes preguntas:

¿Qué ventajas tiene utilizar buenas prácticas de programación? ¿Qué estrategias encontraron útiles? ¿Cuál ha sido el mensaje de error que más te ha aparecido? ¿Cómo lo solucionas?

Grado 10° Guía 5

TIC

Sesión 3

Aprendizajes esperados

Al final de esta sesión se espera que puedas:

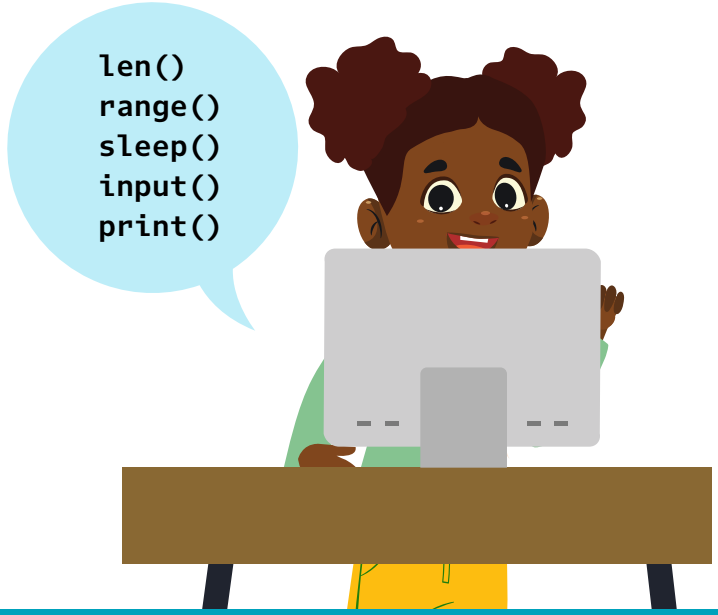
- Utilizar funciones para descomponer un código en *Python*.
- Describir el propósito de los parámetros en las funciones.
- Usar funciones que acepten argumentos.

Material para la clase

- ○ Computador con acceso a *Python*.

Duración sugerida

33% 34% 33%



Lo que sabemos, lo que debemos saber

Esta sección corresponde al 33% de avance de la sesión

En la primera sesión aprendiste cómo puedes descomponer tu programa en subrutinas. Además, has utilizado funciones nativas que vienen con *Python* como len() para calcular la longitud de una lista, range() para generar un rango de números, sleep() para programar una pausa, input() para interactuar con el usuario e incluso print() para mostrar un mensaje.

> *¿Recuerdas cómo se usan? ¿Cómo se escribiría el código para que haga una pausa durante cuatro segundos? ¿Cómo se escribe el código para obtener la longitud de una lista llamada compras?*

Como sabrás, las funciones reciben un número o una variable y retornan un valor o ejecutan una acción. A los valores esperados por una función, se les conoce como **parámetros.**

Observa el código que se presenta, especialmente la segunda línea.

> *¿Qué función utiliza? ¿Qué argumentos recibe? ¿Qué retorna?*

```
notas = [5,4.5,4.8,3]
cantidad_notas = len(notas)
print(cantidad_notas)
```

```
4
```

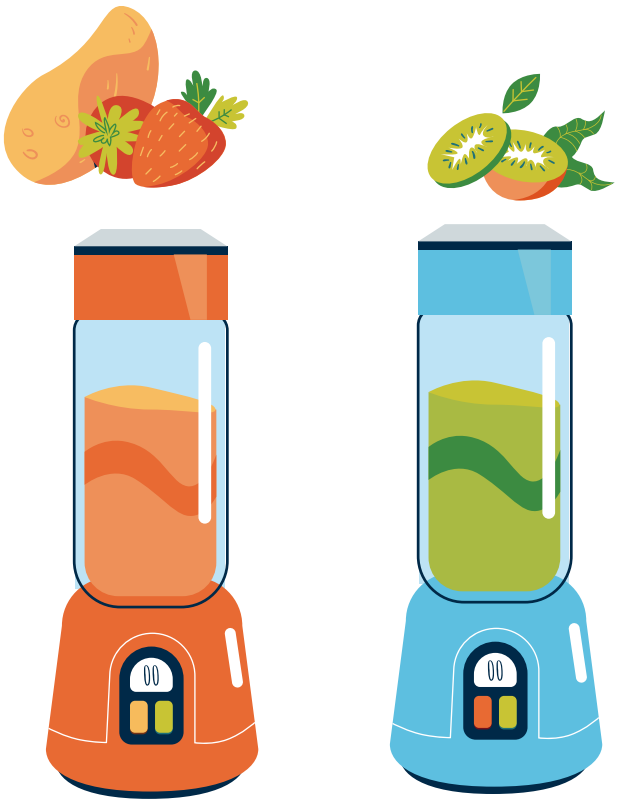
La función **len()** recibe un argumento, en este caso, la lista llamada notas. Y retorna un número que guardamos en la variable cantidad_notas, como se ve en la *Figura 1.*

Figura 1. Variable cantidad_notas

```
cantidad_notas = len(notas)
salida                Argumento
```

Las funciones son procesos que transforman una o más entradas, en una o más salidas. Como una licuadora que recibe frutas (argumentos) y retorna un jugo (salida).

Figura 2. Ejemplo de argumentos y salidas



Tus propias funciones también pueden seguir el mismo proceso a partir de esta estructura:

```
[ ]  def nombre_funcion(parametros):
         instrucciones
         return retorno

     nombre_funcion(argumentos)
```

Todas las funciones deben tener un nombre, unos parámetros de entrada, un código a ejecutar y una salida. Al igual que las funciones matemáticas, en programación nos permiten realizar diferentes operaciones con la entrada para entregar una determinada salida que dependerá del código que escribamos dentro. Por lo tanto, es análogo al clásico y=f(x) de las matemáticas.

¿? *¿Qué aparecerá en pantalla al ejecutar el siguiente código?*

```
[5]  def resta(a, b):
         return a-b

     print(resta(10, 5))
```

¿? *¿Y este código? ¿Qué pasa si una usuaria ingresa 10 y 5?*

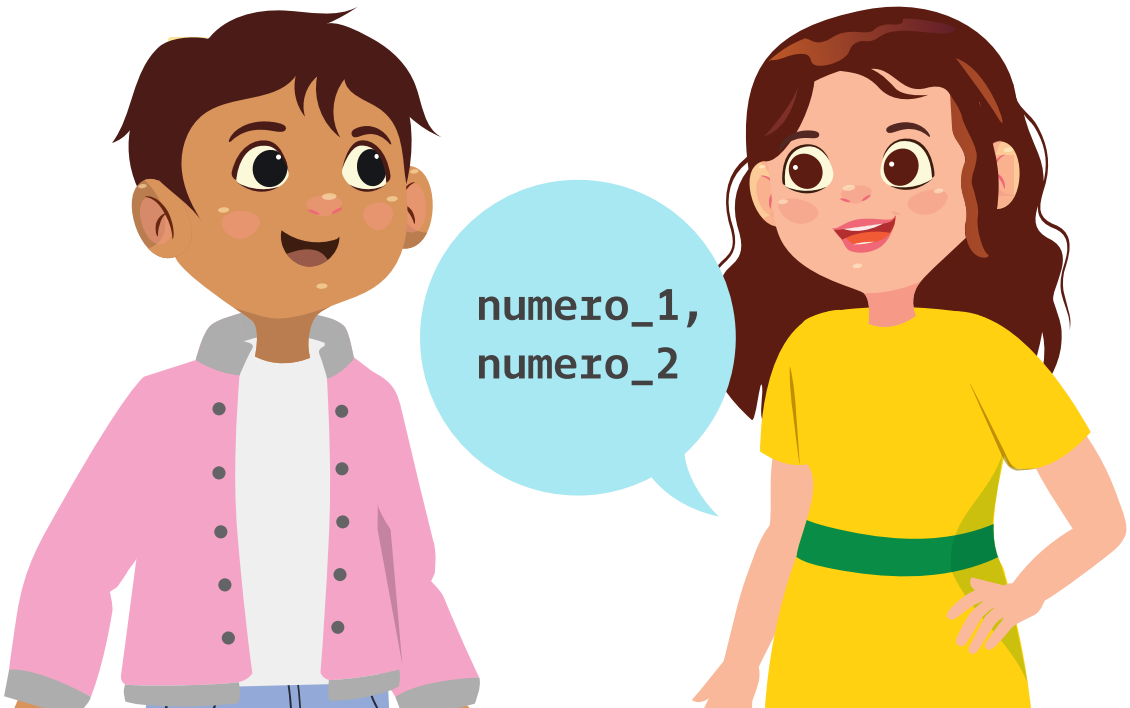
```
numero_1 = int(input("Ingreso el primer número: "))
numero_2 = int(input("Ingreso el segundo número: "))

print(resta(numero_1, numero_2))
print(suma(numero_1, numero_2))
```

Las funciones pueden esperar diferentes argumentos, recibir diferentes tipos de argumentos y retornar uno o varios valores, incluyendo listas. En las siguientes actividades vas a crear algunas funciones de práctica.

Glosario

- **Parámetros:** son los valores que espera una función.
- **Argumentos:** valores que se pasan a una función durante su llamado para que esta realice su tarea.
- **Relación entre parámetros y argumentos:** los parámetros son las "etiquetas" que definimos al crear una función para decir qué datos necesita. Los *argumentos* son los valores reales que enviamos a la función cuando la usamos.



Manos a la obra

Conectadas

Esta sección corresponde al 67% de avance de la sesión

Sigue las indicaciones de tu docente para formar equipos de dos o tres personas. Ingresen a *Python* y resuelvan las siguientes actividades propuestas. Primero se les presentará una actividad guiada y luego deberán desarrollar las actividades 2 y 3 de manera independiente.

Para iniciar, observen y repliquen el código.

```
def calcular(a, b):
    respuesta = a + b
    print(f"{a} + {b} = {respuesta}")

print("Ingresa un número")
num1= int(input())
print("Ingresa otro número")
num2 = int(input())

calcular(num1, num2)
```

¿Qué hace? ¿Pueden explicar todas las líneas? ¿Notan algo nuevo en la forma de imprimir?

Hagan pruebas con el código y describan lo que está pasando.

Nota

Recuerda que el promedio se calcula sumando todos los números y luego dividiendo el resultado entre la cantidad de números.

Actividad 1. El valor medio:

1. Define una función que se llame **valor_medio**.
   El **valor_medio** debe aceptar tres argumentos: a, b y c.

2. El objetivo de la función es que calcule el promedio de tres números y los imprima en la pantalla.
   Crea una variable **promedio** y asígnale el cálculo del promedio de a, b y c.

3. Después de calcular el promedio, el programa debería imprimir la frase:
   "El promedio de los números es:" seguido por el resultado.

4. Prueben que su código esté funcionando correctamente. Hagan los siguientes llamados a su función y verifiquen que el resultado sea el esperado.

Llamado	Resultado esperado
valor_medio(6,8,10)	8
valor_medio(2,2,2)	2
valor_medio(10, 15, 20)	15

5. Eliminen del código las líneas de prueba que crearon en el punto 4. Creen tres solicitudes al usuario para que ingrese tres números: num1, num2 y num3.

6. Usa los números como argumentos en tu función.
   Prueba el código.

Actividad 2: La calculadora

A una profesora le gustaría que crearas un programa para ayudar a sus estudiantes a comprobar sus respuestas a un cuestionario de matemáticas que les han dado. Cada pregunta involucra dos números. Los dos números se pueden usar para sumar, restar, multiplicar o dividir. Estas son algunas de las preguntas que se les hicieron a las y los estudiantes:

$$1 + 3 = ?$$
$$8 \times 2 = ?$$
$$3 - 1 = ?$$
$$8 \div 2 = ?$$

El programa deberá:

- Pedir dos números
- Preguntarles si desean sumar, restar, multiplicar o dividir
- Imprimir la respuesta correcta, por ejemplo, 1 + 3 = 4

En esta etapa, el programa solo necesita funcionar para un cálculo a la vez.

Usen el ejemplo previamente mostrado para ayudar a estructurar su solución.

Actividad 3: Modificando listas

Creen una función que modifique artículos de una lista. La o el usuario debe decir la posición en la que desea modificar el artículo y el nuevo valor de lo que desea guardar.

Pueden utilizar una lista de clases:

```
clases = [“Matemáticas”, “Física”, “Quimica”,
“Historia”,”Lengua”]
```

Cuando se ejecute, sus resultados deben verse similares a los que se presentan a continuación:

```
Así está tu horario
['Matemáticas', 'Física', 'Quimica', 'Historia','Lengua']
Qué clase deseas modificar?1
Ingresa el nuevo valor: Inglés
['Matemáticas', 'Inglés', 'Quimica', 'Historia','Lengua']
```

Antes de cerrar la actividad, inviten a otro grupo a revisar su código. Pídanles que busquen diferencias frente a la solución que propusieron. Luego, pueden revisar el código del otro equipo y hacer pruebas de su funcionamiento.

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 funciones para descomponer un código en *Python*?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

2. ¿Puedes describir el propósito de los parámetros en las funciones?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

3 ¿Puedes usar funciones que acepten argumentos?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

Si tus respuestas a las preguntas anteriores fueron "Parcialmente" o "Aún no", revisa nuevamente los contenidos y consulta las inquietudes que todavía tienes con tu docente.

Para cerrar la sesión, te proponemos el siguiente ejercicio:

Reúnete con la persona que tengas al lado y elijan uno de los dos roles:

Persona 1: Estuvo en la clase de hoy.

Persona 2: Faltó a clase y necesita entender el tema para llegar preparada a la siguiente sesión.

Algunas preguntas que pueden hacer son:

¿?

¿Qué son las funciones?

¿Por qué es importante utilizar funciones?

¿Qué son los números que están adentro de los paréntesis en este ejemplo? valor_medio(10, 15, 20)

¿Cómo puedo saber si la función hace lo que yo espero?

Aprovechen este espacio final para la explicación. Pueden escribir ejemplos, tomar notas y hacer un diagrama que explique los conceptos importantes.

Sesión 4

Aprendizajes esperados

Al final de esta sesión verifica que puedas:

- Acceder a listas dentro de listas.
- Acceder a elementos individuales dentro de listas de dos dimensiones.
- Modificar listas de dos dimensiones.

Material para la clase

- Anexo 4.1
- Computador con acceso a *Python*

Duración sugerida

40% 40% 20%

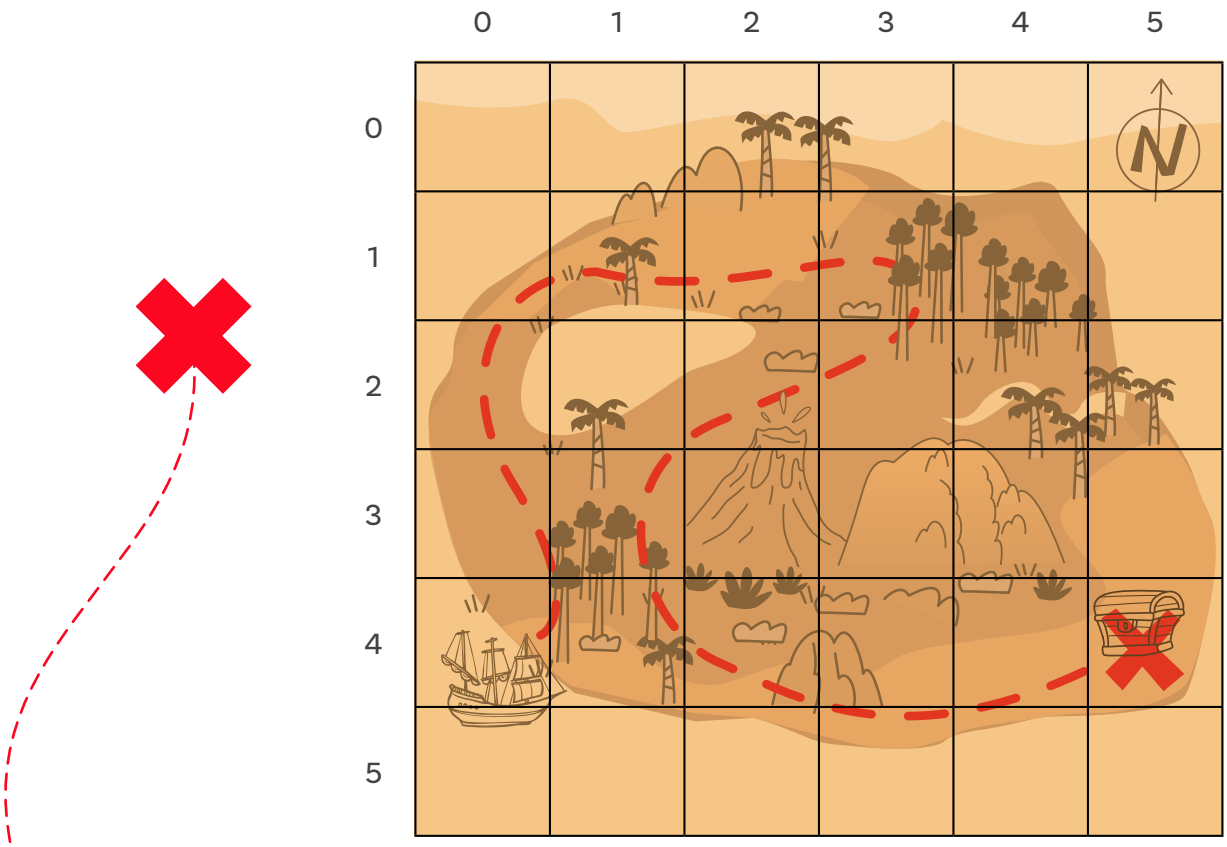
Lo que sabemos, lo que debemos saber

Esta sección corresponde al 40% de avance de la sesión

Vas a comenzar la sesión jugando. En el siguiente tablero hay un tesoro.

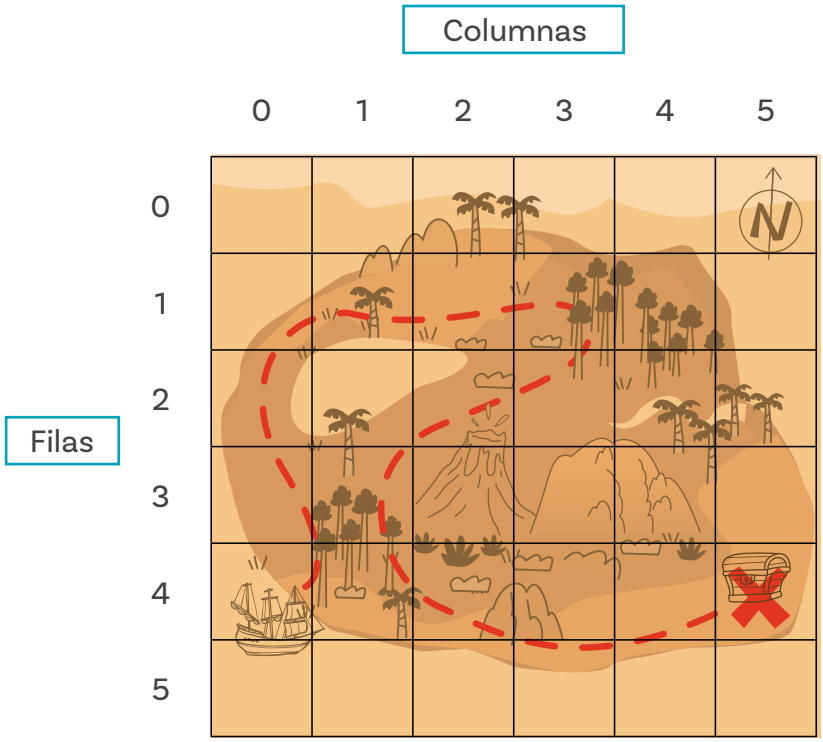
¿En qué posición está el tesoro?
¿En qué posición está el barco?

Figura 1. Mapa del tesoro



Puedes responder de la forma “Fila, columna”.

Figura 2. Filas y columnas del mapa del tesoro



El volcán, por ejemplo, se encuentra entre las posiciones:

- Fila 2, columna 2
- Fila 3, columna 2
- Fila 3, columna 3

Otra forma de decirlo sería:

- Tablero[2][2]
- Tablero[3][2]
- Tablero[3][3]

¿Notas alguna similitud con los conceptos que has aprendido hasta el momento?

Anexo

Anexo 4.1

En esta sesión aprenderás a usar las listas de dos dimensiones (matrices 2D). Como notarás, al igual que en las listas de una dimensión, se accede a los elementos utilizando corchetes [ ]. El primer corchete corresponde a las filas y el segundo a la columna en la que se encuentra el elemento.

Antes de pasar a *Python* tu docente dirigirá un juego de búsqueda del tesoro para practicar el uso de los índices. Este juego puede desarrollarse en el tablero o en parejas utilizando el *Anexo 4.1.*

Instrucciones:

En el siguiente tablero, una persona esconderá cuatro tesoros de diferentes tamaños. Estos escondites son secretos y no deben ser revelados.

Figura 3. Tablero para esconder tesoros

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

Figura 4. Lista de tesoros

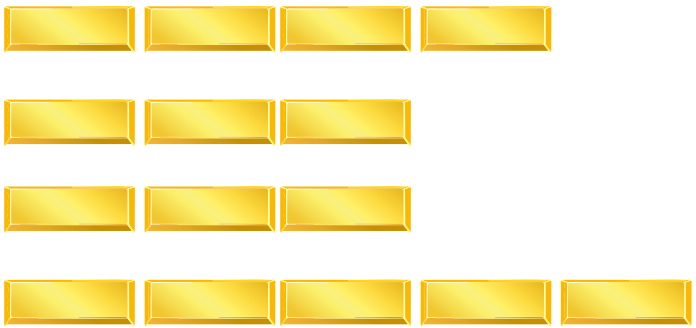
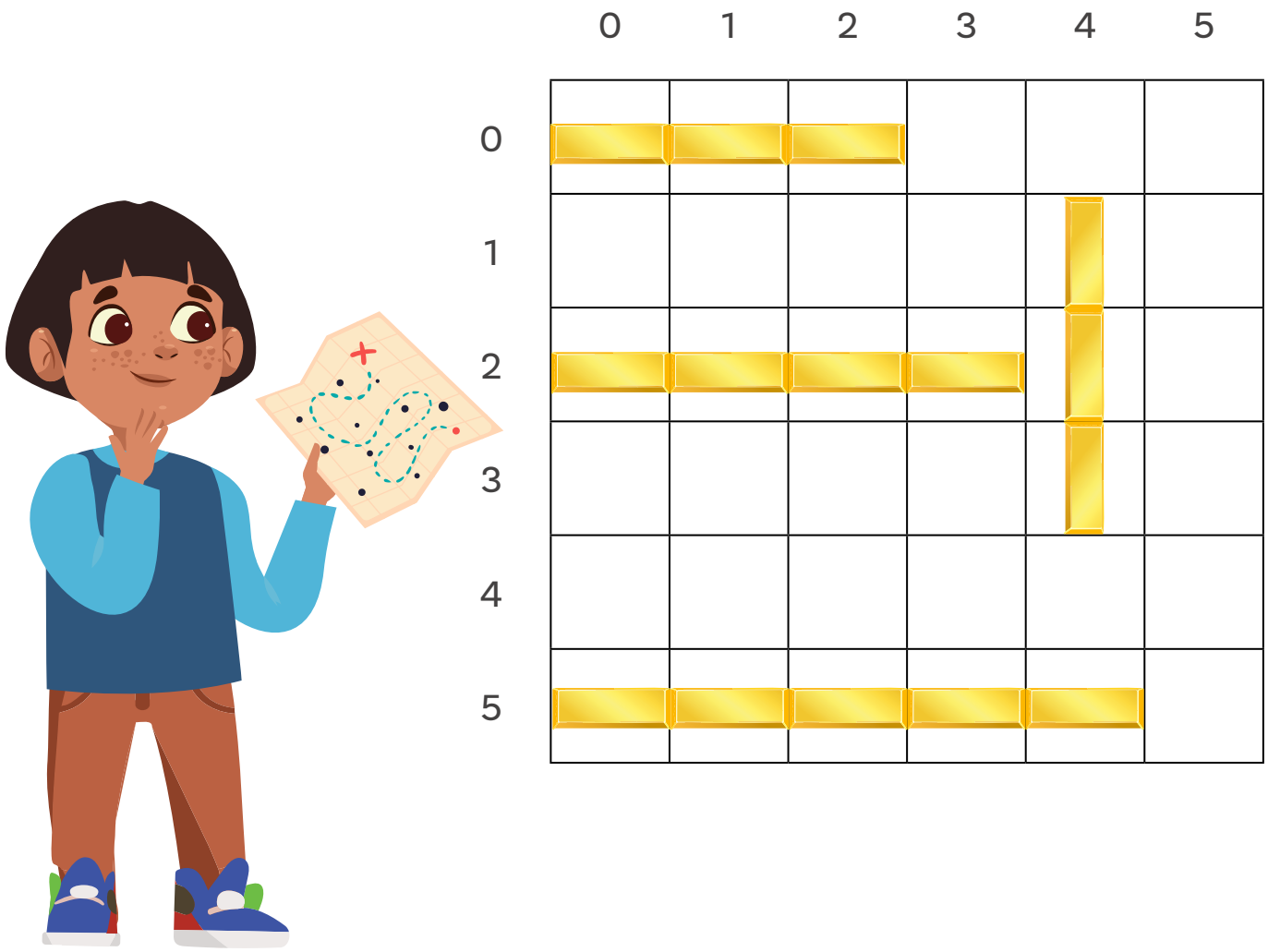


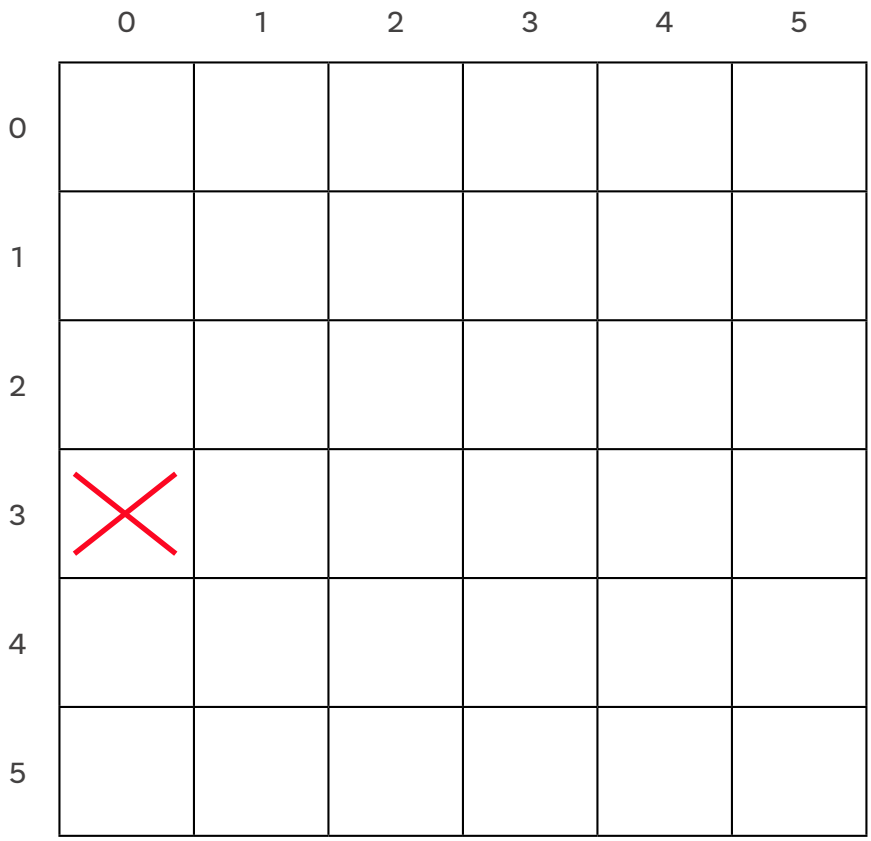
Figura 5. Tablero de quien esconde el tesoro



El resto del salón, o del equipo, deberá tomar turnos para elegir posiciones en donde se realizará la excavación. Si la excavación encuentra una parte de un tesoro, se marcará el mapa de rastreo con un signo positivo. Si no encuentra nada, se marcará el mapa con una X.

La primera persona en jugar elige la posición [3][0]. Pero él o la otra jugadora dice que ahí no hay un tesoro. Entonces marca su propio tablero como en la *Figura 6*.

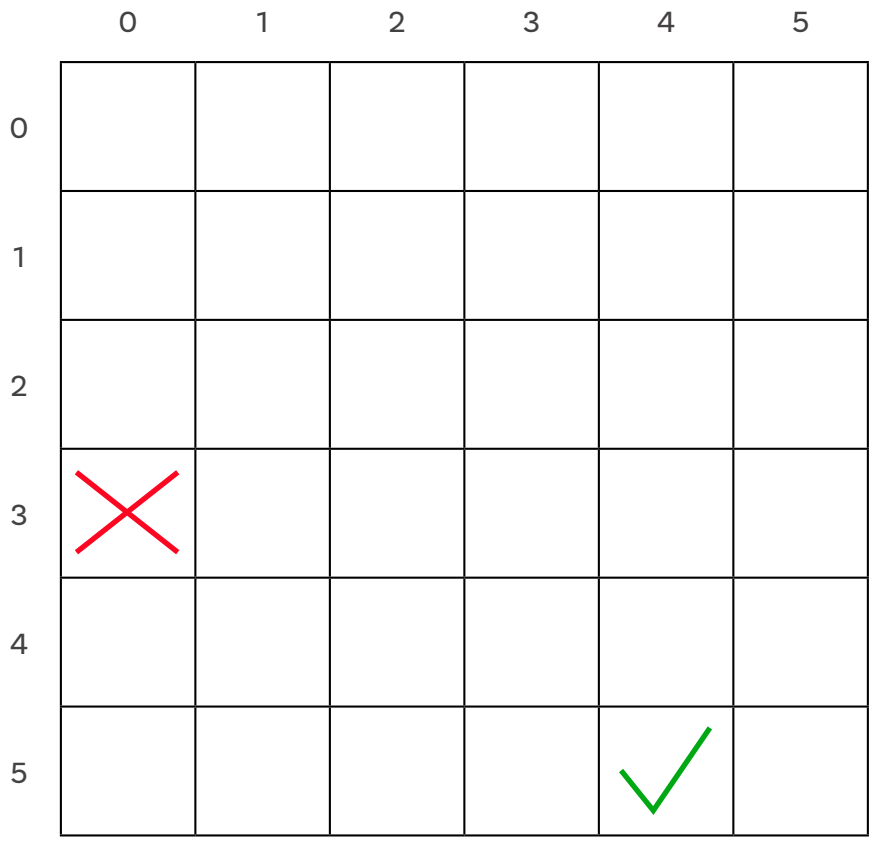
Figura 6. Posiciónal errada en el tablero



Luego intenta en la posición [5][4] y le confirman que allí sí hay un tesoro. Entonces marca su tablero como en la *Figura 7*.

Luego intenta en la posición [5][4] y le confirman que allí sí hay un tesoro. Entonces marca su tablero como en la *Figura 7.*

Figura 7. Posición errada en el tablero



El juego acaba pasados 10 minutos y se cuentan la cantidad de tesoros encontrados.

Al terminar, tu docente dirigirá una conversación para aclarar las dudas que hayan surgido y predecir cómo se puede ver esta nueva estructura de datos en *Python.*

Mira el siguiente ejemplo:

```
1  puntajes = [["Freddy", "Wendy", "Dilan"],
2              [4, 5, 6]]
3
4  print(puntajes[0])
```

Al ejecutar el programa, se imprime esto.
["Freddy", "Wendy", "Dilan"]

¿? *¿Entonces, cómo se accede al nombre Wendy?*

Manos a la obra
Conectadas

Esta sección corresponde al 80% de avance de la sesión

Ingresa a *Python* y completa las siguientes actividades.

Actividad 1:

Replica el ejemplo y ejecútalo. ¿Qué notas?

```
puntajes = [["Freddy", "Wendy", "Dilan"],
            [4, 5, 6]]

print(puntajes[0])
```

Modifica el código para que imprima el nombre de Wendy. Haz seguimiento a tus pruebas utilizando el recuadro:

Intenté	Obtuve
print(puntajes[0])	["Freddy", "Wendy", "Dilan"]
print(puntajes[1])	

Las listas de dos dimensiones, se consideran listas de listas. Es decir que existen como una lista, cuyos elementos son otras listas.

Para entenderlo, podemos ver el ejemplo de la siguiente manera.

```
puntajes = [["Freddy", "Wendy", "Dilan"] ,  [4, 5, 6]]

print(puntajes[0])
```

Como puedes ver, la lista de puntajes encierra sus elementos utilizando los corchetes exteriores (marcados en azul) **[ ]** y cada elemento es una lista más pequeña.

Sin embargo, es una mejor práctica visualizarlo en cuadrícula, pues facilita la comprensión.

Ahora responde:

> *¿Cuántas filas tiene la lista puntajes?*
> *¿Cuántas columnas tiene la lista puntajes?*

Replica el código en *Python* y verifica tus respuestas.

```
puntajes = [["Freddy", "Wendy", "Dilan"],
           [4, 5, 6]]

#filas
print(len(puntajes))

#columnas
print(len(puntajes[0]))
```

Actividad 2:

1. Observa y agrega debajo de cada print() el resultado esperado:

```
temperaturas = [
     ["Ciudad", "Lunes", "Martes", "Miercoles", "Jueves", "Viernes"],
     ["Bogotá",14, 15, 16, 17, 18],
     ["Medellín",22, 23, 24, 25, 26],
     ["Cali",28, 29, 30, 31, 32],
]
print(temperaturas[1])
print(temperaturas[1][3])
print(temperaturas[0][2])
print(temperaturas[3][4])
print(len(temperaturas))
print(len(temperaturas[0]))
```

2. Escribe el código para reemplazar el último valor de Bogotá, por 14.

3. Escribe el código para añadir una fila que contenga la información de una ciudad de tu preferencia

4. Utiliza la función Index para conocer la posición en la que se encuentra la palabra “Jueves”.

Actividad 3:

Ahora, crea una lista de dos dimensiones que cumpla con las siguientes condiciones:

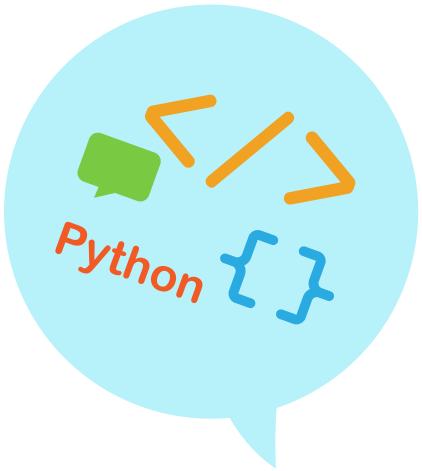
La lista “animales” debe contener tres filas y tres columnas. La primera fila contiene animales acuáticos, la segunda fila contiene animales terrestres y la tercera fila contiene aves.

Luego, crea un programa para preguntarle al usuaria/o qué tipo de animales quiere conocer e imprimir la lista de esos animales.

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 acceder a listas dentro de listas?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

2. ¿Puedes acceder a elementos individuales dentro de listas de dos dimensiones?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

3. ¿Puedes modificar listas de dos dimensiones?
   - ◯ Sí
   - ◯ Parcialmente
   - ◯ Aún no

Ahora te proponemos un reto con tu grupo usando una rutina llamada Pensar, presentar e integrar (P-P-I). Esta rutina te permitirá afianzar los conocimientos que ya tienes, identificar los que están parcialmente alcanzados o aún no están alcanzados y trabajar sobre las dudas que aún tengas:

- 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:

> *¿En qué situaciones de la vida puedes ver las estructuras de dos dimensiones? ¿Qué otras operaciones podrías hacer sobre los elementos? ¿Cuál fue tu mayor desafío en la clase de hoy? ¿Porqué? ¿Cuál ha sido tu mayor reto hasta el momento al programar en* Python?

Sesión 5

Aprendizajes esperados

Al final de esta sesión se espera que puedas:

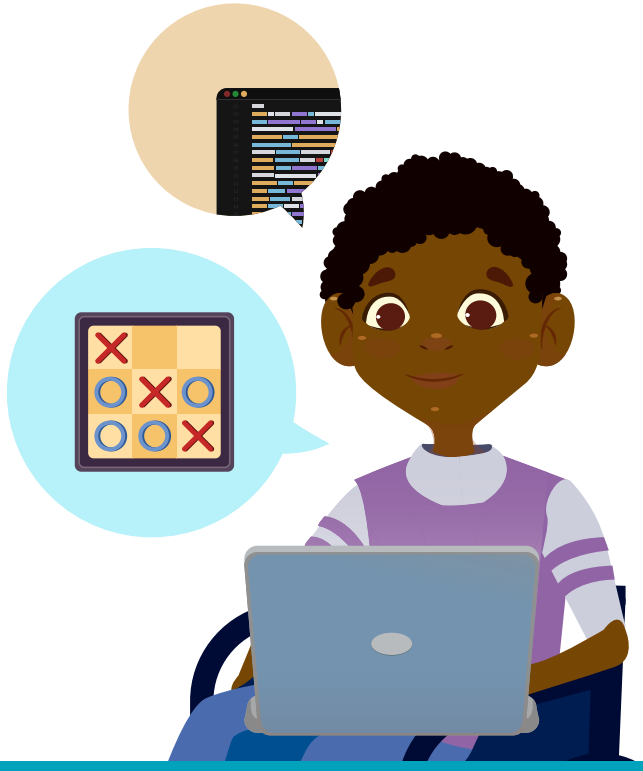
- Programar un juego interactivo en *Python*.
- Diferenciar entre declarar, inicializar y asignar variables.
- Interpretar los mensajes de error y usarlos para identificar y corregir los errores de lógica y sintaxis en programas creados en *Python*.

Material para la clase

- [ ] Anexos 5.1, 5.2 y, opcionalmente, el 5.3
- [ ] Computador con acceso a Python.

Duración sugerida

15% 70% 15%

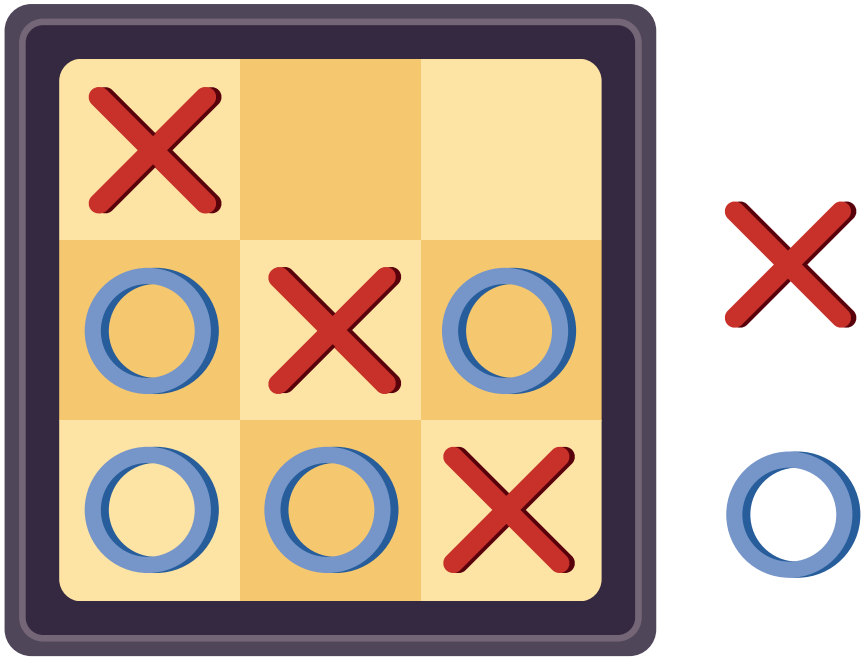


Lo que sabemos, lo que debemos saber

Esta sección corresponde al 15% de avance de la sesión

¿? *¿Cómo le llamas a este juego? ¿En qué consiste?*

Figura 1. Tablero de juego Triqui



Este juego recibe muchos nombres diferentes dependiendo de dónde estés. Por ejemplo: en Argentina le dicen ta-te-ti, en Chile le dicen gato, en Guatemala le dicen totito y en gran parte de Colombia le decimos triqui.

En esta sesión vas a aplicar tus conocimientos adquiridos para programar un juego de triqui en *Python*.

¿? *¿Cómo le darías indicaciones a tu docente de qué posición jugar, utilizando la notación de listas que aprendiste en la sesión pasada?*

Enlaces

Archivos de esta sesión:
Tu docente puede demostrar una partida de triqui en el tablero. O utilizar esta versión de Google.

Archivos de esta sesión.

Para desarrollar tu propio reto, debes tener muy claras las reglas del juego. Recuerda que el pensamiento algorítmico es la capacidad de resolver problemas siguiendo una serie de pasos claros, organizados y lógicos.

Antes de empezar, observa el video "IA jugando" que podrás descargar del enlace o QR *Archivos de esta sesión*. Responde:

¿Qué pasó? ¿Cómo evaluarías el programa de este robot? ¿Qué falta?

Tómate un momento para repasar los diagramas, tablas y resúmenes que has construido hasta el momento. Recordar las funciones que has utilizado y los métodos para modificar las listas te será muy útil para dar solución al reto.

Manos a la obra
Desconectadas

Esta sección corresponde al 85% de avance de la sesión

Reúnete en un equipo de tres personas para jugar tres partidas de triqui. Antes de empezar distribuyan los roles:

1. Computadora: será la única persona que podrá marcar las equis y los círculos en el papel. Sin embargo, no debe intervenir en el juego.
2. Jugadores(as): deberán tomar turnos e indicarle al computador cada una de sus jugadas.

Mientras juegan, deben reflexionar sobre todos los pasos que se llevan a cabo en un juego. ¿Qué decisiones toman? ¿Cuándo se acaba el juego?

Figura 2. Ejemplo de cómo puede verse el tablero

Anexos

Anexo 5.1

Anexo 5.2

Cambien de rol en cada partida para que cada integrante pueda hacer las veces de computador.

Ahora completen en equipo el *Anexo 5.1.* Deben descomponer el juego en todos sus pasos.

Compartan con otro grupo su lista y añadan los pasos que les hayan faltado.

Después de descomponer el problema, deben pensar en la estructura de su programa. En el *Anexo 5.2* podrán encontrar un ejemplo de la estructura final del juego. En esta sesión trabajarán dos de las funciones: Mostrar tablero y Validar ganador.

Estas dos funciones les permitirán:

1. Presentar el tablero de juego.
2. Validar si en el tablero hay alguna de las combinaciones ganadoras.

Manos a la obra

Conectadas

En los mismos equipos de trabajo ingresen a *Python* para desarrollar la parte conectada del proyecto.

Como pueden ver, la primera función que van a programar es la que les permitirá preparar y mostrar el tablero de juego.

Para crear el tablero de juego, pueden utilizar una lista de dos dimensiones así:

```
tablero = [[" ", " ", " "],
           [" ", " ", " "],
           [" ", " ", " "]]
```

Como recordarán, a cada posición de la lista se puede acceder de la forma
tablero[fila][columna]

Creen una función que reciba el tablero como argumento y lo imprima.

A continuación, pueden encontrar cómo empezar la función:

```
def mostrar_tablero(tablero):
    print(" ",tablero[0][0],"|", tablero[0][1], "|", tablero[0][2])
    print("———┼———┼——— ")
```

1. Modifiquen la función para que se vea el tablero completo.
2. Llamen la función para ver el tablero en su estado inicial.
3. Modifiquen la lista del tablero agregando O y X y verifiquen si el tablero de visualización todavía se visualiza correctamente.

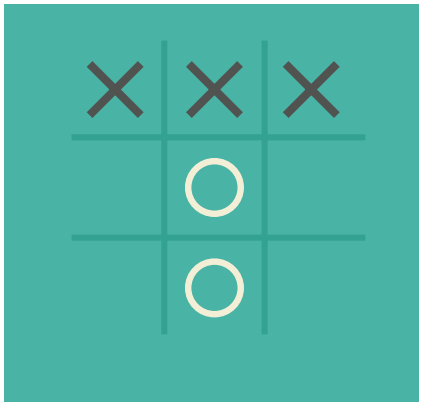
Ya tienen la primera función. Ahora van a evaluar si alguien ganó.

¿Cuántas formas existen para ganar el juego? Escribe el número total de opciones: _____
_____
_____
_____

1. Usando la cuadrícula a continuación como guía, anoten todas las combinaciones para ganar. Guíense del ejemplo para continuar como muestra la *Figura 3*.

2. Completen la función para validar todas las posibles condiciones y saber si una alguien ganó el juego.

Figura 3. Cuadrícula de ejemplo.



[0][0] [0][1] [0][2]

```
def validar_ganador(tablero,jugador):
    gana = False
    if (tablero[0][0]== jugador
        and tablero[0][1]== jugador
        and tablero[0][2]== jugador):
        gana = True
```

3 Comprueben su función cambiando el tablero para probar cada combinación posible. Por ejemplo, para probar la primera combinación su variable debe verse así:

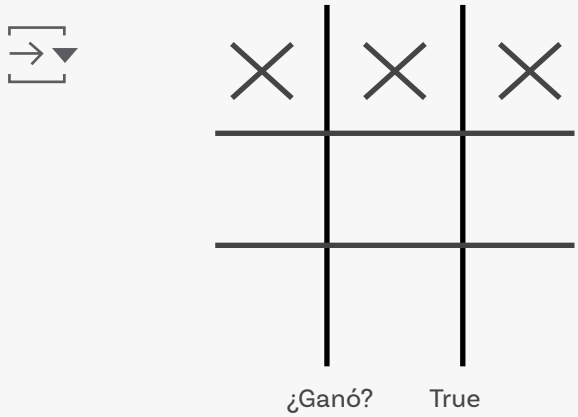
```
tablero = [["X", "X", "X"],
           [" ", " ", " "],
          [" ", " ", " "]]
```

Pista: deben crear una variable **jugador** para que puedan usar el argumento en la función **validar_ganador**. Asegúrense de que la jugadora o el jugador tenga el valor X o su programa no funcionará.

Pista: para asegurarse de que su código funciona, tendrán que imprimir la variable **gana** y ver si es **True** cuando se utiliza una combinación ganadora.

La salida podría verse como en la *Figura 4.*

Figura 4. Salida de variable gana



Realicen diferentes pruebas en su código. Si funciona correctamente, habrán terminado las dos funciones base del juego.

¿Cómo continuarían con la programación?

Antes de terminar la clase revisen su código y comenten en grupo los avances que han logrado. Luego, guarden su archivo y prepárense para hablar con el resto de la clase.

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 programar un juego interactivo en *Python*?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

2 ¿Puedes diferenciar entre declarar, inicializar y asignar variables?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

3 ¿Puedes interpretar los mensajes de error y usarlos para identificar y corregir los errores en programas (lógica, sintaxis)?

- ◯ Sí
- ◯ Parcialmente
- ◯ Aún no

Para terminar la sesión, tu docente dirigirá una mesa redonda en la que retomen todos los conceptos y retos trabajados durante esta guía. Antes de ello, toma un momento para reflexionar en los aprendizajes que aún no están alcanzados y pide apoyo a tu docente o plantea preguntas a la mesa de compañeras y compañeros que te permita resolver tus dudas.

Algunas preguntas para discutir son:

¿?

¿Por qué fue importante descomponer el juego antes de programar?

¿Qué descubriste en el proceso?

¿A qué dificultades te enfrentaste?

¿Cómo aplicarías tus conocimientos para finalizar el juego?

Para ir más lejos

Si cuentas con el tiempo, puedes continuar programando el juego.

1 Instrucciones del juego.

Es necesario crear una función que dé las instrucciones al usuario al comienzo del juego.

Durante el juego, el usuario verá el tablero con los números del 1 al 9 para mostrar las ubicaciones donde puede colocar su X u O. Luego debe ingresar el número de la casilla donde desea jugar y luego su X u O se colocarán en el tablero. El primer jugador(a) será X y el segundo jugador(a) será O.

Crea una función llamada **instrucciones** que proporcione estas instrucciones al usuario.

Luego, prueba tu programa llamando la función. Tú decides cuánta información debe aparecer. La *Figura 5* muestra un ejemplo de cómo pueden verse las instrucciones.

Figura 5. Ejecución de la función instrucciones

```
Te damos la bienvenida al juego
Instrucciones:
Este es un juego de dos jugadores
El primer jugador utilizará las X
El segundo jugador utilizará las O
El tablero de juego se presenta asi:

 1 | 2 | 3
___|___|__
 4 | 5 | 6
___|___|___
 7 | 8 | 9
   |   |

Para elegir una posición, entra el número de casilla que
quieres ocupar
```

2 Jugar un turno

Es necesario crear una función que mueva la pieza del jugador o jugadora al tablero. Crea una función llamada turno, que reciba dos parámetros: el tablero y el símbolo del jugador (X/O).

Todas las funciones deben definirse antes de que comience el programa principal.

Completa la función asegurándote de que:

- Pide al jugador o jugadora actual que ingrese la posición deseada.
- Utiliza la posición introducida para añadir una O o una X al tablero en la ubicación deseada.
- Retorna el tablero.

Para probar tu código, asegúrate de que la jugadora o el jugador tenga el valor X. Prueba tu programa ingresando cada número del 1 al 9 y asegurándote de que agregue una X a la posición correcta en el tablero.

En la *Figura 6* se muestra un ejemplo de salida.

Figura 6. Salida de la función turno

```
Turno del jugador X

 1 | 2 | 3
___|___|__
 4 | 5 | 6
___|___|___
 X | 8 | 9
   |   |

Elige una casilla 3

 1 | 2 | X
___|___|__
 4 | 5 | 6
___|___|___
 X | 8 | 9
   |   |

¿Ganó? False
```

Anexo 1.1 Antes de irnos

1 Creaste un programa para imitar el eco. ¿Qué va a pasar cuando ejecutes el código?

```
def eco_eco(palabra):
    print(palabra)
    print(palabra)
```

A Se va a imprimir "palabra"

B Se va a repetir una palabra dos veces

C Nada

D Va a salir un error de sintaxis

¿Por qué? _____

2 ¿Qué va a pasar cuando se ejecute el código?

```
def letra_b():
    print("b")

def letra_a():
    print("a")

letra_a()
letra_b()
```

A b y luego a

B a y luego b

C Nada

D letra_a y luego letra_b

¿Por qué? _____

3 ¿Qué va a pasar cuando se ejecute el siguiente código?

```
def menu_inicio():
    print("Bienvenido al menú de inicio")
    print("1. Registrarse")
    print("2. Iniciar sesión")

menu_inicio()
```

A Error de sintaxis

B Se va a imprimir un menú de inicio

C Nada

D Se va a imprimir menu_inicio

¿Por qué? _____

Anexo 1.1 Antes de irnos

4 ¿Te enfrentaste a errores en la actividad? ¿Cuáles?

5 ¿Cómo los solucionaste?

Anexo 4.1 Tableros de juego

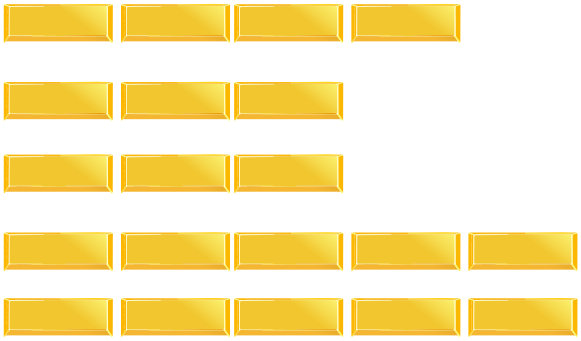
Tablero de quien esconde el tesoro

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

Lista de tesoros

	0	1	2	3	4	5
0						
1						
2						
3						
4						
5						

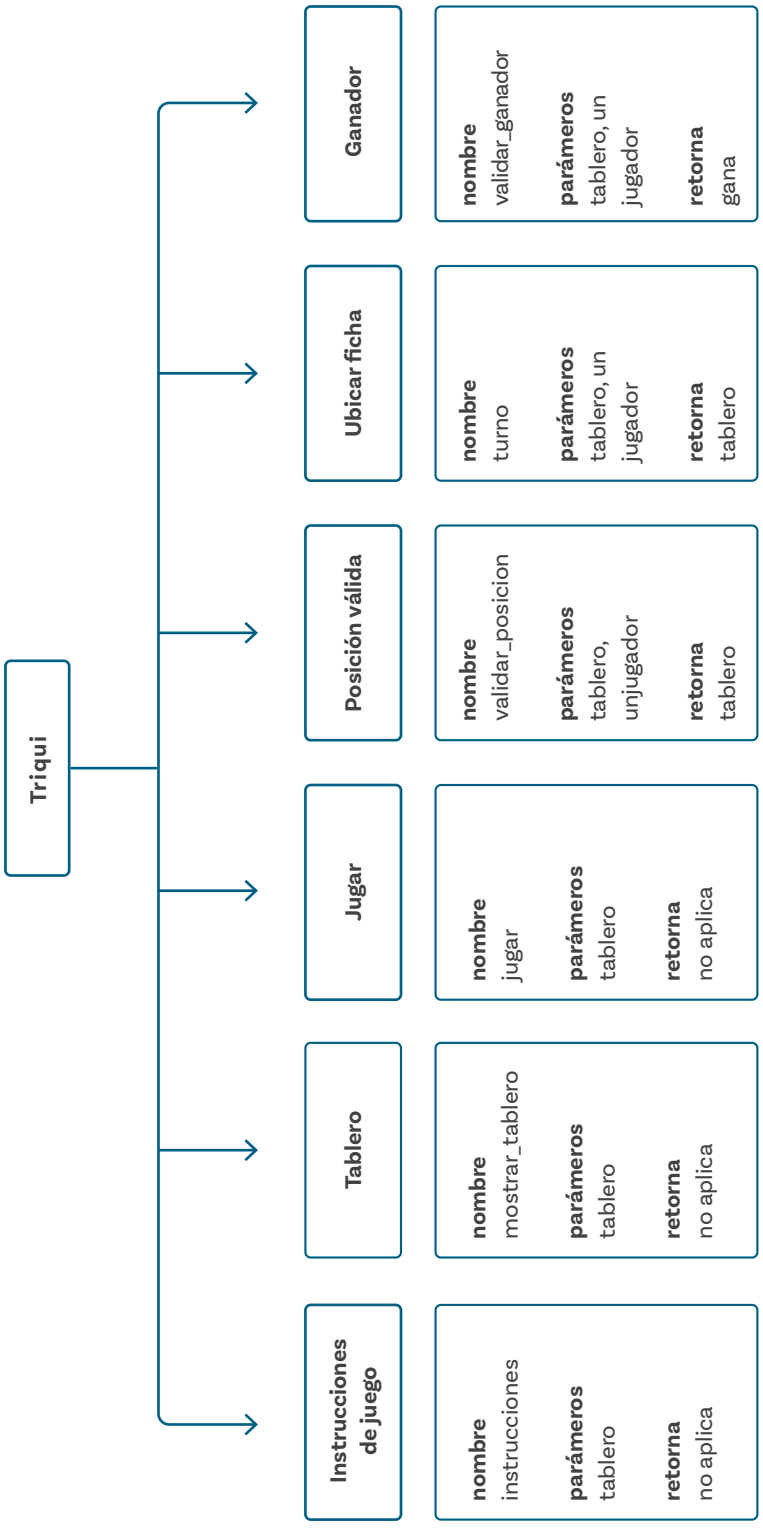
Lista de tesoros



Anexo 5.1 Descomposición

Piensa en todos los pasos que se requieren para que un(a) jugador(a) gane el juego. Anótalos todos en el cuadro provisto. No te preocupes demasiado por lograr un orden específico de los pasos.

Anexo 5.2 Estructura del juego



TIC

Apoya:

Educación

BRITISH COUNCIL

Colombia Programa
{EL CÓDIGO A TU FUTURO}