¿QUÉ ES UN PROGRAMA?

Un programa es una secuencia de instrucciones que especifican cómo ejecutar una computación. La computación puede ser algo matemático, como solucionar como solucionar un sistema de ecuaciones o determinar las raíces de un polinomio, pero también puede ser una computación simbólica, como buscar y reemplazar el texto de un documento o (aunque parezca raro) compilar un programa.

Las instrucciones (comandos, órdenes) tienen una apariencia diferente en lenguajes de programación diferentes, pero existen algunas funciones básicas que se presentan en casi todo lenguaje:

- **Entrada**: Recibir datos del teclado, o un archivo u otro aparato.
- **Salida**: Mostrar datos en el monitor o enviar datos a un archivo u otro aparato.
- **Matemáticas**: Ejecutar operaciones básicas de matemáticas como la adicción y la multiplicación.
- **Operación condicional:** Probar la veracidad de alguna condición y ejecutar una secuencia de instrucciones apropiada.
- **Repetición:** Ejecutar alguna acción repetidas veces, normalmente con alguna variación.

Lo crea o no, eso es todo. Todos los programas que existen, por complicados que sean, están formulados exclusivamente con tales instrucciones. Así, una manera de describir la programación es: El proceso de romper una tarea en tareas cada vez más pequeñas hasta que estas tareas sean suficientemente simples para ser ejecutadas con una de estas instrucciones simples.

Representa gráficamente que es un programa:

Hola mundo en Python

En cualquier introducción a un nuevo lenguaje de programación, no puede faltar el famoso *Hola Mundo*. Se trata del primer programa por el que se empieza, que consiste en programar una aplicación que muestra por pantalla ese texto. Si ejecutas el siguiente código, habrás cumplido el primer hito de la programación en Python.

```
print("Hola Mundo")
```

Las comillas señalan el comienzo y el final del valor; no aparecen en el resultado.

Por lo tanto ya te puedes imaginar que la función print() sirve para imprimir valores por pantalla. Imprimirá todo lo que haya dentro de los paréntesis. Fácil ¿verdad?

En Python, una **variable** es un nombre que se utiliza para almacenar un valor en la memoria del programa. Las variables son fundamentales en cualquier lenguaje de programación, ya que permiten almacenar y manipular datos.

¿Cómo se declara una variable en Python?

En Python, declarar una variable es muy sencillo. Solo necesitas asignarle un valor usando el operador de asignación =. No es necesario especificar el tipo de dato de la variable (como en otros lenguajes), ya que Python es un lenguaje de tipado dinámico. Esto significa que el tipo de dato se infiere automáticamente según el valor que le asignes.

Sintaxis básica: nombre_variable = valor

- **nombre_variable**: Es el nombre que le das a la variable. Debe seguir ciertas reglas:
  - Puede contener letras (a-z, A-Z), números (0-9) y guiones bajos (_).
  - No puede comenzar con un número.
  - No puede ser una palabra reservada de Python (como if, else, for, etc.).
  - Es sensible a mayúsculas y minúsculas (por ejemplo, edad y Edad son variables diferentes).
- **valor**: Es el dato que almacenará la variable. Puede ser un número, una cadena de texto, un booleano, una lista, etc.

Nombrando variables

Crear variables

Las variables en Python se pueden crear asignando un valor a un nombre.

```
x = 10
y = "Nombre"
z = 3.9
```

Nombres de variables

Podemos asignar el nombre que queramos, respetando no usar las palabras reservadas de Python ni espacios, guiones o números al principio.

# Válido	# No válido
_variable = 10	2variable = 10
vari_able = 20	var-iable = 10
variable10 = 30	var iable = 10
variable = 60	
variaBle = 10	

Asignar múltiples valores

Se pueden asignar múltiples variables en la misma línea.

```
x, y, z = 10, 20, 30
```

Imprimir variables

Una variable puede ser impresa por pantalla usando print()

```
x = 10
y = "Nombre"

print(x)
print(y)
```

Ejemplos de declaración de variables

1. **Variable con un número entero**:

edad = 25
Aquí, edad es una variable que almacena el valor 25. Python infiere que es un número entero (int).

2. **Variable con un número decimal**:

altura = 1.75
En este caso, altura es una variable que almacena un número decimal (float).

3. **Variable con una cadena de texto**:

nombre = "Juan"
nombre es una variable que almacena una cadena de texto (str). Las cadenas se escriben entre comillas simples (') o dobles (").

4. **Variable con un valor booleano**:

estudiando = True
estudiando es una variable que almacena un valor booleano (bool), que puede ser True (verdadero) o False (falso).

5. **Variable con una lista**:

numeros = [1, 2, 3, 4, 5]
numeros es una variable que almacena una lista (list) de números.

Reglas y buenas prácticas

1. **Nombres descriptivos**:
   Usa nombres que describan el propósito de la variable. Por ejemplo:
   - edad es mejor que e.
   - nombre_usuario es mejor que n.
2. **Mayúsculas y minúsculas**:
   Python distingue entre mayúsculas y minúsculas. Por ejemplo:
   - edad y Edad son variables diferentes.
3. **No usar palabras reservadas**:
   No uses palabras clave de Python como nombres de variables. Por ejemplo:
   - if, else, for, while, etc.
4. **Snake case**:
   Por convención, en Python se usa **snake_case** para nombrar variables. Esto significa separar palabras con guiones bajos (_). Por ejemplo:
   - nombre_usuario en lugar de nombreUsuario (camelCase).

Constantes

En Python, una **constante** es un tipo de variable cuyo valor no debe cambiar durante la ejecución del programa. Por convención, se utilizan nombres en **MAYÚSCULAS** para indicar que una variable debe ser tratada como una constante y no modificarse.

Ejemplo de constante en Python

PI = 3.14159
GRAVEDAD = 9.81

CONCATENACION

En Python, la **concatenación de variables** se refiere a la unión de dos o más valores, generalmente strings (cadenas de texto), aunque también se puede aplicar a otros tipos de datos como listas.

Para unir dos o más cadenas de texto, puedes usar el operador +.

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

OPERACIONES CON VARIABLES

Operador	Símbolo	Ejemplo	Resultado
Suma	`+`	`5 + 3`	`8`
Resta	`-`	`10 - 4`	`6`
Multiplicación	`*`	`6 * 7`	`42`
División	`/`	`20 / 4`	`5.0`
División entera	`//`	`20 // 3`	`6`
Módulo	`%`	`20 % 3`	`2`
Potencia	`**`	`2 ** 3`	`8`

Operaciones básicas con variables
Podemos hacer muchas cosas con las variables, como sumar, restar, multiplicar o dividir.
Ejemplos:

1. Sumar variables
Si tienes dos variables con números, puedes sumarlas.

```
num1 = 5
num2 = 3
resultado = num1 + num2
print(resultado)  # Esto imprimirá 8
```

2. Restar variables
También puedes restar una variable de otra.

```
num1 = 10
num2 = 4
resultado = num1 - num2
print(resultado)  # Esto imprimirá 6
```

- Una **variable** es como una cajita donde guardas información.
- Puedes **sumar**, **restar**, **multiplicar** y **dividir** variables con números.
- Puedes **cambiar** el valor de una variable cuando quieras.

Operaciones Aritméticos en Python		
Operador	**Jerarquía**	**Operación**
**	Mayor a Menor	Potencia
*, /, %, //		Multiplicación, división, modulo, división entera
+, -		Suma, resta

A continuación, algunos ejemplos demostrativos.

a) 6 * 5^3 / 5 div 3
6 * 125 / 5 div 3
750 / 5 div 3
150 / 3
50

b) 4*2*(160 mod 3^3) div 5*13-28
4*2*(160 mod 27) div 5*13- 28
4*2*25 div 5*13-28
8*25 div 5*13 – 28
200 div 5 *13 – 28
40 * 13 – 28
520 – 28
492

$$7+8/2$$
$$7+8/2$$
$$7+4$$
$$7+4=11$$

$$10+9/2$$
$$10+9/2$$
$$10+4.5$$
$$10+4.5=14.5$$

Los comentarios

En Python, los **comentarios** son líneas de texto que se ignoran al ejecutar el programa. Se utilizan para explicar el código, hacer anotaciones o desactivar temporalmente partes del programa. Los comentarios son muy útiles para que otros programadores (o tú mismo en el futuro) entiendan qué hace el código.

1. Comentarios de una línea. Se crean usando el símbolo #. Todo lo que esté después de # en la misma línea se considera un comentario y no se ejecuta.

```
# Esto es un comentario de una línea
print("Hola, mundo")  # Este comentario explica la línea de código
```

2. **Comentarios multilínea**: Usa varias líneas con # o docstrings (""").

```
# Este es un comentario
# que ocupa varias líneas
# para explicar algo más detallado.
print("Hola, mundo")
```

3. **Docstrings**: Úsalos para documentar funciones, clases y módulos.

```
def suma(a, b):
    """
    Esta función suma dos números.
    Parámetros:
    - a: primer número
    - b: segundo número
    Retorna:
    - La suma de a y b
    """
    return a + b
```

4. **Buenas prácticas**: Sé claro, evita comentarios obvios y mantén los comentarios actualizados.

Instrucciones básicas en Python

En Python, **leer** y **escribir** valores es una parte fundamental para interactuar con el usuario y manejar datos.

1. Escribir valores (salida). Para mostrar información en la pantalla, se utiliza la función print(). Esta función toma uno o más valores y los imprime en la consola.

Ejemplo básico:

```
print("Hola, mundo")  # Imprime: Hola, mundo
```

Ejemplo con variables:

```
nombre = "Juan"
edad = 25
print("Nombre:", nombre, "Edad:", edad)  # Imprime: Nombre: Juan Edad: 25
```

Formatear salida con f-strings (Python 3.6+):

```
nombre = "Ana"
edad = 30
print(f"{nombre} tiene {edad} años.")  # Imprime: Ana tiene 30 años.
```

2. Leer valores (entrada). Para obtener información del usuario, se utiliza la función input(). Esta función lee una línea de texto ingresada por el usuario y la devuelve como una cadena (str).

```
nombre = input("Ingresa tu nombre: ")
print("Hola,", nombre)  # Imprime: Hola, [lo que el usuario haya escrito]
```

Convertir la entrada a otro tipo de dato:
Como input() siempre devuelve un string, si necesitas un número (por ejemplo, un entero o un decimal), debes convertir el valor usando int() o float().

Ejemplo con números:

```
edad = int(input("Ingresa tu edad: "))  # Convierte la entrada a un entero
print("El próximo año tendrás", edad + 1, "años.")
```

3. Ejemplo completo: Leer y escribir valores

```
# Solicitar datos al usuario
nombre = input("Ingresa tu nombre: ")
edad = int(input("Ingresa tu edad: "))
altura = float(input("Ingresa tu altura en metros: "))

# Mostrar los datos
print(f"\nDatos ingresados:")
print(f"Nombre: {nombre}")
print(f"Edad: {edad} años")
print(f"Altura: {altura} metros")

# Realizar un cálculo
print(f"\nEl próximo año tendrás {edad + 1} años.")
```

- **Escribir valores**: Usa print() para mostrar información en la pantalla.
- **Leer valores**: Usa input() para obtener datos del usuario. Recuerda convertir el valor si necesitas un tipo de dato específico.
- **Formatear salida**: Usa f-strings para crear mensajes más claros y dinámicos.

Actividad 1. Ejercicios de declaración de variables en Python

Ejercicio 1: Declaración de Variables

Declara variables para almacenar la siguiente información:

- Tu nombre
- Tu edad
- Tu altura en metros
- Si estás estudiando o no (usando un booleano)

Ejercicio 2: Concatenación de Variables

Declara las siguientes variables y luego concaténalas para imprimir un mensaje:

- ciudad: Tu ciudad de residencia
- pais: Tu país de residencia

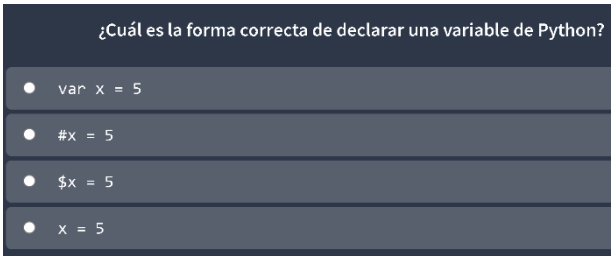
Ejercicio 3: Operaciones con Variables

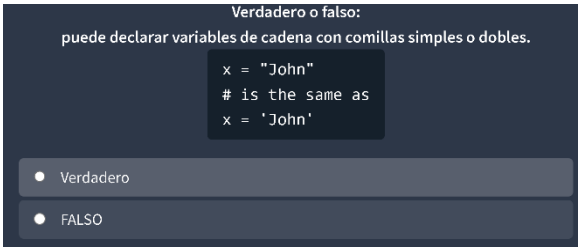
Declara dos variables a y b con valores numéricos y realiza las siguientes operaciones:

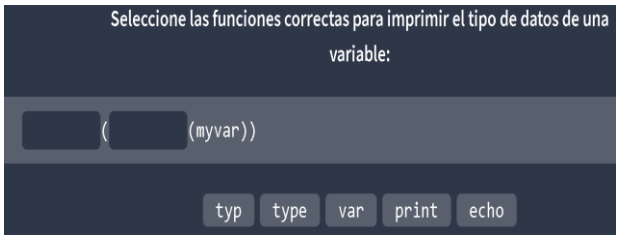
- Suma
- Resta
- Multiplicación
- División

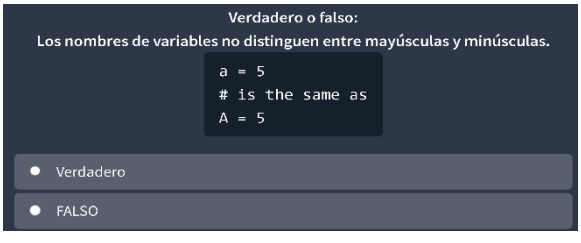
Ejercicio 4: Cambio de Valores

Declara dos variables x e y con valores iniciales, luego intercambia sus valores.









Ejercicio 5: Uso de input()
Pide al usuario que ingrese su nombre y edad, luego imprime un mensaje con esa información.

Ejercicio 6: Tipos de Datos
Declara variables con diferentes tipos de datos (entero, flotante, cadena, booleano) y luego imprime el tipo de cada variable usando type().

Ejercicio 7: Operaciones con Cadenas
Declara una variable texto con el valor "Python es genial" y realiza las siguientes operaciones:

- Imprime la longitud de la cadena.
- Imprime la cadena en mayúsculas.
- Imprime la cadena en minúsculas.

Ejercicio 8: Uso de f-strings
Declara variables para tu nombre y edad, y luego usa f-strings para imprimir un mensaje que incluya esas variables.

Ejercicio 9: Constantes
En Python, las constantes no existen como tal, pero por convención se usan mayúsculas para indicar que una variable no debe cambiar. Declara una "constante" para el valor de PI y úsala para calcular el área de un círculo.

Ejercicio 10: Múltiples Variables en una Línea
Declara tres variables a, b y c en una sola línea y asígnales valores. Luego, imprime sus valores.

Actividad 2. Completa la tabla de tipos de datos

Tabla de tipos de datos en Python

Tipo de dato	Descripción	Ejemplo
	Números enteros (positivos o negativos).	`edad = 25`
float	Números decimales.	`pi =`
	(strings).	`= "Ana"`
bool	Valores booleanos (o).	`es_mayor =`
	Listas (colecciones ordenadas y mutables).	`[1, 2, 3]`

Actividad 3. Tabla de operadores matemáticos

Operador	Símbolo	Descripción	Ejemplo	Resultado
Suma	+		5 + 3	
Resta	-		10 - 4	
Multiplicación	*		6 * 7	
División	/		20 / 4	
División entera	//		20 // 3	
Módulo	%		20 % 3	
Potencia	**		2 ** 3	
Negación	-		-5	

Actividad 4. Cuadro comparativo entre valores de entrada y salida en Python.

Función	Descripción	Ejemplo
Entrada		
Salida		

Identación

¿Qué es la identación?

- La **identación** es como usar sangrías para organizar el código.
- En Python, la identación es **obligatoria** y se usa para definir bloques de código.
- Usa **4 espacios** para identar y sé consistente.
- La identación se usa en condicionales, bucles y funciones.

Ejemplos

```
edad = 20  # Tu edad

if edad >= 18:  # Pregunta si la edad es mayor o igual a 18
    print("Hola")  # Esto está identado (4 espacios)
```

Ejemplo con un condicional `if-else`:

python

```python
edad = 15

if edad >= 18:
    print("Eres mayor de edad")  # Esto se ejecuta si la condición es verdadera
else:
    print("Eres menor de edad")  # Esto se ejecuta si la condición es falsa
```

Ejemplo con un bucle `for`:

python

```python
for i in range(3):  # Repite el bloque 3 veces
    print("Hola", i)  # Esto está identado
```

Ejemplo con una función:

python

```python
def saludar(nombre):  # Define una función
    print("Hola,", nombre)  # Esto está identado

saludar("Juan")  # Llama a la función
```

Operadores relacionales

Operaciones relacionales en Python			
Operador	**nombre**	**Ejemplo**	**Resultado**
==	Igualdad	"Cola" == "Sola"	False
!=	diferente	a' != 'x'	True
<	menor que	7 < 151	True
>	mayor que	22 > 10	True
<=	menor o igual que	14 <= 20	True
>=	mayor o igual que	55 >= 21	True

Operador	Ejemplo	Resultado
=	“cola” = “sola”	FALSO
<>	‘a’ <> ‘x’	VERDADERO
<	7 < 151	VERDADERO
>	22 > 10	VERDADERO
<=	14 <= 20	VERDADERO
>=	55 >= 21	VERDADERO

Condiciones y declaraciones If de Python

Python admite las condiciones lógicas habituales de las matemáticas:

- Es igual a: a == b
- No es igual a: a != b
- Menor que: a < b
- Menor o igual a: a <= b
- Mayor que: a > b
- Mayor o igual a: a >= b

Estas condiciones se pueden utilizar comúnmente en "declaraciones if" y bucles.
Una "declaración if" se escribe utilizando la palabra clave if

Ejemplo:

```
a = 200
b = 33
if b > a:
  print("b es más grande que a")
else:
  print("b es menor que a")
```

If anidado

Puedes tener ifdeclaraciones dentro de if declaraciones, esto se llama declaraciones *anidadas* if .

```
x = 41

if x > 10:
  print("Es más de 10,")
  if x > 20:
    print("y tambien arriba de0!")
  else:
    print(" pero no arriba de 20.")
```

Estructuras de control

1. Condicionales (if, elif, else)

Imagina que estás eligiendo qué hacer en un día soleado:

- **Si** hace sol, puedes ir al parque.
- **O si** está nublado, puedes ver una película.
- **Si no** (si está lloviendo), te quedas en casa.

En Python, esto se escribe así:

```
tiempo = "soleado"

if tiempo == "soleado":
    print("¡Vamos al parque!")
elif tiempo == "nublado":
    print("¡Veamos una película!")
else:
    print("Me quedo en casa.")
```

- **if**: "si pasa esto, haz esto".
- **elif**: "o si pasa esto otro, haz esto".
- **else**: "si no pasa nada de lo anterior, haz esto".

2. Bucles (for)

Imagina que tienes que contar cuántos dulces hay en una bolsa. En lugar de contarlos uno por uno, puedes usar un bucle para que la computadora lo haga por ti.

En Python, esto se escribe así:

```
dulces = ["chocolate", "caramelo", "gomitas", "chupetín"]

for dulce in dulces:
    print("¡Encontré un", dulce + "!")
```

- **for**: Es como decir "para cada cosa en esta lista, haz esto".
- En este caso, el bucle recorre la lista de dulces y los va mostrando uno por uno.

3. Bucles (while)

Imagina que estás jugando un juego donde tienes que seguir corriendo hasta que te canses. Mientras tengas energía, sigues corriendo.

En Python, esto se escribe así:

```
energia = 10

while energia > 0:
    print("¡Sigo corriendo!")
    energia = energia - 1

print("Me cansé, me detengo.")
```

- **while**: "mientras esto sea verdad, sigue haciendo esto".
- En este caso, el bucle se repite mientras la energía sea mayor que 0. Cuando la energía llega a 0, el bucle se detiene.

4. Break y Continue

Imagina que estás buscando un juguete en una caja llena de cosas:

- Si encuentras el juguete, dejas de buscar (**break**).
- Si encuentras algo que no es el juguete, lo ignoras y sigues buscando (**continue**).

En Python, esto se escribe así:

```
cosas = ["libro", "pelota", "muñeca", "juguete", "lápiz"]

for cosa in cosas:
    if cosa == "juguete":
        print("¡Encontré el juguete!")
        break  # Dejo de buscar
    else:
        print("Esto no es el juguete, sigo buscando...")
```

- **break**: Es como decir "detente, ya encontré lo que buscaba".
- **continue**: Es como decir "ignora esto y sigue adelante".

Las estructuras de control son como las reglas que le dicen a la computadora qué hacer en cada momento.

- **if, elif, else**: Para tomar decisiones.
- **for**: Para repetir algo con cada elemento de una lista.
- **while**: Para repetir algo mientras una condición sea verdadera.
- **break**: Para detener un bucle.
- **continue**: Para saltar al siguiente paso del bucle.

Explica los siguientes ejercicios, colocando la salida al evaluar la expresión

```
a = 4
b = 2
if b != 0:
    print(a/b)
```

```
if b != 0:
    c = a/b
    d = c + 1
    print(d)
```

```
a = 10
if a > 5 and a < 15:
    print("Mayor que 5 y menos que 15")
```

```
x = 5
if x == 5:
    print("Es 5")
else:
    print("No es 5")
```

```
x = 5
if x == 5:
    print("Es 5")
elif x == 6:
    print("Es 6")
elif x == 7:
    print("Es 7")
```

```
x = 5
if x == 5:
    print("Es 5")
elif x == 6:
    print("Es 6")
elif x == 7:
    print("Es 7")
else:
    print("Es otro")
```

```
# Verifica si un número es par o impar
x = 6
if x % 2 == 0:
    print("Es par")
else:
    print("Es impar")
```

```
for i in range(0, 5):
    print(i)
```

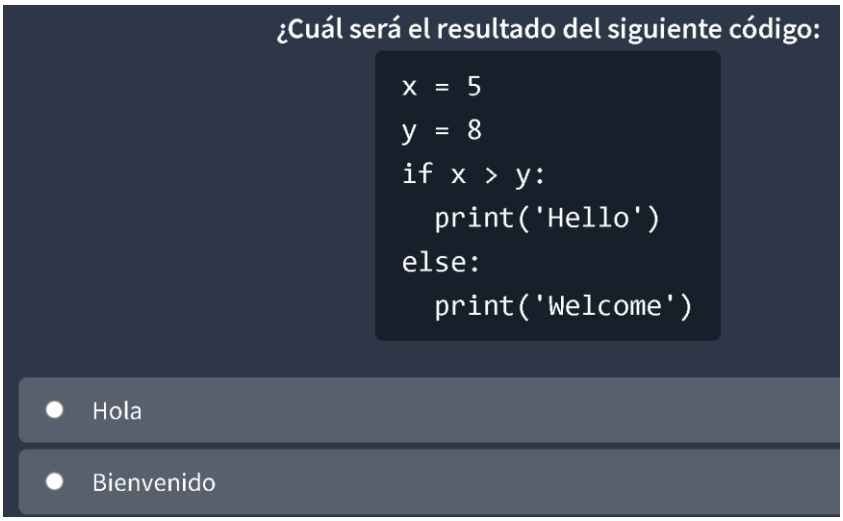
```
for i in "Python":
    print(i)
```

```
x = 5
while x > 0:
    x -=1
    print(x)
```
salida:

Comenta cada línea

```
x = 5
while x > 0:
    x -=1
    print(x) #4,3,2,1,0
else:
    print("El bucle ha finalizado")
```

```
z = 7
x = 1
while z > 0:
    print(' ' * z + '*' * x + ' ' * z)
    x+=2
    z-=1
```

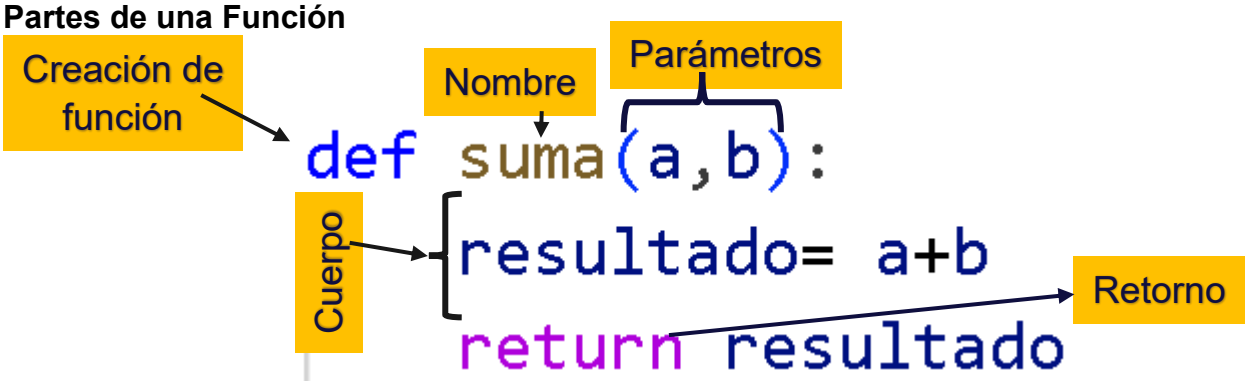


Actividad 5. Infografía de estructuras de control

Funciones

Una función es un bloque de código que realiza una tarea específica. En lugar de escribir el mismo código una y otra vez, puedes crear una función y usarla cuando la necesites.

Partes de una Función



1. **Nombre**: Es el nombre de la función. Te permite llamarla cuando la necesites.
2. **Parámetros**: Son los datos que le das a la función para que haga su trabajo.
3. **Cuerpo**: Es el código que realiza la tarea.
4. **Retorno**: Es el resultado que la función te devuelve.

Cómo Crear una Función

Para crear una función en Python, usas la palabra clave **def**, seguida del nombre de la función y paréntesis (). Aquí tienes un ejemplo:

```
def saludar(nombre):
    mensaje = "¡Hola, " + nombre + "!"
    return mensaje
```

- **def**: Indica que estás creando una función.
- **saludar**: Es el nombre de la función.
- **(nombre)**: Es el parámetro que recibe la función.
- **return**: Devuelve el resultado.

Ejemplos de Funciones

1. Función sin Parámetros

Una función que no necesita datos para funcionar:

```
def decir_hola():
    print("¡Hola a todos!")

decir_hola()  # Llama a la función
```

Salida: ¡Hola a todos!

2. Función con Parámetros

Una función que recibe datos para hacer su trabajo:

```
def sumar(a, b):
    resultado = a + b
    return resultado

total = sumar(5, 3)
print("La suma es:", total)
```

La suma es: 8

3. Función con Valor Predeterminado

Puedes darle a un parámetro un valor predeterminado:

```
def multiplicar(a, b=2):
    return a * b

print(multiplicar(5))      # Usa el valor predeterminado de b (2)
print(multiplicar(5, 3))   # Pasa un nuevo valor para b (3)
```

Salida: 10
15

4. Función sin Retorno

Una función puede no devolver nada, solo hacer algo:

```
def mostrar_menu():
    print("1. Jugar")
    print("2. Salir")

mostrar_menu()
```

Salida: 1. Jugar
2. Salir

¿Por qué Usar Funciones?

1. **Reutilización**: Puedes usar la misma función muchas veces.
2. **Organización**: Tu código es más fácil de leer y mantener.
3. **Ahorro de Tiempo**: No necesitas escribir el mismo código una y otra vez.
4. **Puedes llamar a una función cuantas veces necesites,**

¿Qué es un Módulo?

Un módulo es un archivo de Python que contiene código, como funciones, variables o clases, que puedes usar en otros programas. Python tiene muchos módulos ya creados (llamados módulos estándar), y también puedes crear tus propios módulos.

¿Para qué Sirven los Módulos?

- Reutilizar código: Puedes usar funciones de un módulo en muchos programas.
- Organizar el código: Puedes dividir tu programa en partes más pequeñas y manejables.
- Ahorrar tiempo: No necesitas escribir todo desde cero, puedes usar módulos ya existentes.

Cómo Usar un Módulo

Para usar un módulo, primero debes importarlo. Esto le dice a Python que quieres usar las herramientas de ese módulo. Aquí te explico cómo hacerlo:

1. Importar un Módulo Completo

Puedes importar todo el módulo y usar sus funciones con la sintaxis nombre_modulo.nombre_funcion.

```
import math  # Importa el módulo math

# Usar una función del módulo math
print(math.sqrt(16))  # Calcula la raíz cuadrada de 16
```

Salida: 4.0

Módulos Estándar de Python

Python viene con muchos módulos útiles. Aquí tienes algunos ejemplos:

1. **math**: Funciones matemáticas (raíz cuadrada, seno, coseno, etc.).
2. **random**: Generación de números aleatorios.
3. **datetime**: Trabajar con fechas y horas.
4. **os**: Interactuar con el sistema operativo (archivos, carpetas, etc.).
5. **sys**: Funciones relacionadas con el sistema.

Ejemplo con el módulo random:

```
import random

# Generar un número aleatorio entre 1 y 10
numero = random.randint(1, 10)
print("Número aleatorio:", numero)
```

Coloca la salida de ejemplo: _____

Crear tu Propio Módulo

También puedes crear tus propios módulos. Solo necesitas guardar tu código en un archivo .py y luego importarlo.

1. Crea un archivo llamado herramientas.py:

```
def saludar(nombre):
    return f"¡Hola, {nombre}!"

def sumar(a, b):
    return a + b
```

2. En otro archivo, importa y usa tu módulo:

```
import herramientas

print(herramientas.saludar("Ana"))  # Usa la función saludar
print(herramientas.sumar(5, 3))     # Usa la función sumar
```

Salida:

¿Dónde se Guardan los Módulos?

- Los módulos estándar de Python están en la carpeta de instalación de Python.
- Tus módulos deben estar en la misma carpeta que tu programa o en una ruta que Python conozca.

Resumen

- Un módulo es un archivo de Python con funciones, variables o clases.
- Puedes importar módulos estándar o crear los tuyos.
- Usa import para traer un módulo a tu programa.
- Los módulos te ayudan a reutilizar código, organizar tus programas y ahorrar tiempo.

Actividad 6. Infografía de funciones Vs Módulos

¿Qué es una Excepción?

Una excepción es un **error** que ocurre mientras un programa se está ejecutando. Por ejemplo:

- Intentar dividir un número entre cero.
- Intentar abrir un archivo que no existe.
- Intentar usar una variable que no ha sido definida.

Si no manejas estas excepciones, el programa se detendrá y mostrará un mensaje de error. Pero con el manejo de excepciones, puedes controlar estos errores y evitar que el programa se detenga.

```
try:
    numero = int(input("Ingresa un número: "))
    resultado = 10 / numero
except ValueError:
    print("¡Eso no es un número válido!")
except ZeroDivisionError:
    print("¡No puedes dividir entre cero!")
else:
    print("El resultado es:", resultado)
finally:
    print("Fin del programa.")
```

- **try**: Prueba un bloque de código que podría generar un error.
- **except**: Atrapa y maneja el error.
- **else**: Se ejecuta si no hay errores.
- **finally**: Se ejecuta siempre, haya o no errores.

Listas, arreglos y matrices

1. Listas

Una **lista** en Python es una colección ordenada y mutable (puedes cambiar sus elementos) que puede contener diferentes tipos de datos, como números, cadenas, otras listas, etc.

Características:

- **Flexible**: Puede contener cualquier tipo de dato.
- **Dinámica**: Puedes agregar, eliminar o modificar elementos.
- **Ordenada**: Los elementos tienen un orden definido (índices).

```
lista = [1, "Hola", 3.14, True]
print(lista)
```

Salida

```
[1, "Hola", 3.14, True]
```

Operaciones comunes:

- Agregar elementos: lista.append(5)
- Eliminar elementos: lista.remove("Hola")
- Acceder a un elemento: lista[0] (primer elemento).

2. Arreglos (Arrays)

Un **arreglo** es una colección de elementos del **mismo tipo**. En Python, los arreglos no son nativos, pero puedes usar el módulo array para crearlos.

Características:

- **Homogéneo**: Todos los elementos deben ser del mismo tipo.
- **Eficiente**: Ocupa menos memoria que una lista porque solo almacena un tipo de dato.
- **Menos flexible**: No puedes mezclar tipos de datos.

```
import array

arreglo = array.array('i', [1, 2, 3, 4])  # 'i' indica que es un arreglo de enteros
print(arreglo)
```

Salida:

```
array('i', [1, 2, 3, 4])
```

Operaciones comunes:

- Agregar elementos: arreglo.append(5)
- Acceder a un elemento: arreglo[0] (primer elemento).

3. Matrices

Una **matriz** es una estructura bidimensional (filas y columnas) que se utiliza para representar datos en forma de tabla. En Python, no hay un tipo de dato nativo para matrices, pero puedes usar **listas de listas** o bibliotecas como NumPy.

```
matriz = [
    [1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]
]
print(matriz)
```

Salida:

```
[[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Características:

- **Bidimensional**: Tiene filas y columnas.
- **Homogénea**: En bibliotecas como NumPy, todos los elementos son del mismo tipo.
- **Matemáticas**: Se usa mucho en operaciones matemáticas y científicas.

Operaciones comunes:

- Acceder a un elemento: matriz[0][0] (primer elemento).
- Sumar matrices: matriz1 + matriz2 (con NumPy).
- Multiplicar matrices: np.dot(matriz1, matriz2) (con NumPy).

¿Cuándo Usar Cada Uno?

1. **Listas**: Flexibles y dinámicas, pueden contener cualquier tipo de dato.
   - Cuando necesitas almacenar diferentes tipos de datos.
   - Cuando necesitas flexibilidad para agregar, eliminar o modificar elementos.
2. **Arreglos**: Eficientes para datos del mismo tipo, pero menos flexibles.
   - Cuando necesitas almacenar muchos elementos del mismo tipo y quieres ahorrar memoria.
   - Cuando trabajas con datos numéricos simples.
3. **Matrices**: Ideales para datos bidimensionales y operaciones matemáticas.
   - Cuando trabajas con datos bidimensionales, como tablas o operaciones matemáticas.
   - Cuando necesitas realizar cálculos complejos (usando NumPy).