Unidad 2

Flujo de Control

Al desarrollar programas, escribiremos una serie de sentencias en orden de ejecución, una por línea, respetando la sintaxis de cada una. La secuencia en la que estas instrucciones se ejecutan se llama **Flujo de Control de un Programa (FCP)**.

En un flujo de control estándar (secuencial), las instrucciones se ejecutan en el orden en el que fueron escritas, y sin omitir ninguna. Es decir:

```
- Esto se ejecuta primero
- Esto se ejecuta segundo
- Esto se ejecuta tercero
```

Los programas deben comunicarse con el mundo exterior, tanto para obtener los datos con los que trabajarán, como para entregar los resultados obtenidos.

La forma más básica para que el programa pueda comunicarse con el usuario, es realizando el ingreso de datos por **teclado** (dispositivo estándar de entrada), y cuando se necesiten mostrar resultados o mensajes al usuario, utilizar la **pantalla** (dispositivo estándar de salida).

Python dispone de herramientas tanto para poder leer datos desde el teclado, como para enviar resultados a la pantalla.

Para leer datos por teclado, utilizamos la función `input()`. Y para mostrar información por pantalla al usuario, utilizamos la función `print()`.

Variables y Constantes

Los programas, además de tener algoritmos, necesitarán trabajar con **datos**.

Todos los datos, sean ingresados por el usuario o producidos por el propio programa, y que necesiten ser recordados o utilizados más adelante, deben alojarse en la **memoria interna (MI)** de la computadora. Estos datos, pueden ser constantes o variables.

Si el dato permanecerá inalterable, decimos que ese dato es una **constante**. Por ejemplo:

- `2419` → es una constante numérica.
- `"Hola"` → es un texto (cadena de texto, *string*) constante.

Por el contrario, si un dato puede cambiar (*variar*) su valor, entonces será una **variable**.

Para manipular variables, necesitaremos un **nombre** que le pondremos a la misma para identificarla y utilizarla dentro de nuestro programa. Ese nombre puede ser cualquier combinación de letras y números, teniendo en cuenta que las letras mayúsculas y minúsculas son consideradas diferentes (es decir, `edad`, `Edad` y `EDAD` son diferentes variables).

Es una **buena práctica** que las variables lleven un nombre que sea representativo del dato que guardan. Por ejemplo, si lo que vamos a guardar es la edad de una persona, tendría sentido que la variable se llame `edad`.

En Python, por convención, se usa el estilo de escritura **snake_case**. Es decir, se escribe el nombre en minúscula, y para separar palabras se usa el guion bajo. Por ejemplo, para una variable que guarda la distancia recorrida por un vehículo, el nombre podría ser `distancia_recorrida`.

Además de ponerle un nombre a la variable, en la mayoría de los casos vamos a querer darle un valor, esto se llama **asignación** y se representa con el operador *igual* (=). Para asignar un valor a una variable en Python, se escribe de esta manera:

```
nombre_variable = valor

# Por ejemplo:
nombre = "Juan"
edad = 25
```

Tipos de Datos

Los programas trabajan con diversidad de datos, no todos de la misma naturaleza. A veces, necesitaremos transformar datos a números, a texto, o consultar si un dato es verdadero o falso. Es decir, se pueden realizar **operaciones** con los datos. Pero dependiendo de la naturaleza o el **tipo** de dato, cambiarán las operaciones que podemos realizar con él.

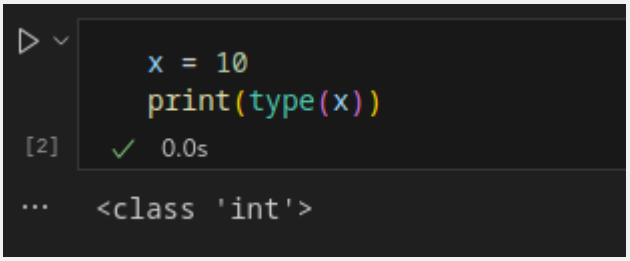
Todos los lenguajes disponen de un conjunto de tipos de dato predefinidos, es decir, que el lenguaje conoce, sabe cómo guardarlos en memoria, y qué operaciones se puede realizar con ellos.

En lo que a tipos de datos respecta, podemos clasificar a los lenguajes de programación en **fuertemente tipados** (el tipo de dato con el que se declara una variable no puede cambiar a lo largo de la ejecución del programa) y **débilmente tipados** (se puede cambiar el tipo de dato de una variable durante la ejecución). Python pertenece a los lenguajes de tipado débil.

Los tipos de dato predefinidos de Python, son:

Tipo de dato	Ejemplo
int (entero)	`2`, `5`, `23`
float (real)	`2.5`, `3.14159`
complex (complejo)	`(10, 9j)` (la componente con j es la parte imaginaria)
bool (booleano/lógico)	`True`, `False` (verdadero y falso, los únicos dos valores posibles)
str (cadena de texto)	`“Hola Mundo”`

Para conocer el tipo de dato de una variable, podemos utilizar la función `type()`:



Los **operadores aritméticos** (aplicables a datos numéricos) válidos en Python, son:

Símbolo	Definición	Ejemplo
+	Suma	10+cant
-	Resta	saldo-pago
*	Producto	a*20
/	División com Precisión Decimal	a/3.5
//	División Entera	a//12
%	Resto	num_par%2
+=	Suma abreviada, agrega	a=a+3 ≈ a+=3
-=	Resta abreviada, quita	a=a-3 ≈ a-=3
*=	Producto abreviado	a=a*3 ≈ a*=3
/=	División abreviada	a=a/3 ≈ a/=3
//=	División entera abreviada	a=a//3 ≈ a//=3
%=	Resto abreviado	a=a%3 ≈ a%=3

Se aplica el orden de las operaciones de la matemática, es decir: primero se resuelve lo que esté entre paréntesis, luego la multiplicación, luego la división y el módulo, y luego la suma y la resta. Para alterar este orden, se pueden usar los paréntesis, por ejemplo al escribir `(2 + 3) / 5` realizará primero la suma, y luego la división.

Los **operadores de texto** (aplicables a datos de tipo *str*) son:

Símbolo	Definición	Ejemplo
+	Concatenación	'hola'+' juan'-> hola Juan
*	Repetición de Texto	'ja'*3 -> jajaja
[k] o [-k]	Selección de caracter	a='hola' a[0] -> h a[2] -> l
[i : j : p]	Selección de una porción del texto. Siendo: i : inicio, j : final, p : pasos	a[0:2]-> ho
+=	Concatenación abreviada	a = hola a+='y chau' -> holay chau
*=	Repetición abreviada	a*=2 -> holahola

En Python, un string es internamente un conjunto (*set*) ordenado de caracteres, esto implica que podemos manipularlos utilizando la sintaxis aplicable a los conjuntos. Por ejemplo:

```
palabra = "Hola"
print(palabra[0]) # Imprime -> H
print(palabra[1:]) # Imprime -> ola
print(palabra[2:4]) # Imprime -> la
print(palabra[::2]) # Imprime -> Hl
```

Si necesitamos obtener la longitud de un string, en Python podemos usar la función `len()`:

```
nombre = "Juan"
print(len(nombre)) # Imprime -> 4
nombre2 = "Nicolás"
print(len(nombre2)) # Imprime -> 7
```

Ingreso de Datos

La mayoría de los programas van a necesitar que el usuario ingrese algún dato. Para poder trabajar con estos datos, los guardaremos en variables.

Para leer un dato desde el teclado, utilizamos la función `input()`.

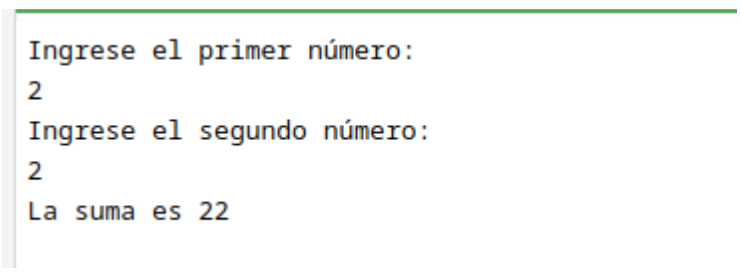
```
nombre = input("Ingrese su nombre: ")
print("Hola", nombre)
```

Al encontrar la instrucción `input`, el intérprete de Python mostrará el texto que le pasamos como parámetro (el cual sirve para indicar al usuario qué debe ingresar), y luego se quedará esperando hasta que ingresemos un valor y pulsemos *enter*.

Es importante notar que la función `input` siempre devolverá un string, independientemente del dato que haya ingresado el usuario. Por ejemplo:

```
num1 = input("Ingrese el primer número: ")
num2 = input("Ingrese el segundo número: ")
suma = num1 + num2
print("La suma es", suma)
```

Si ejecutamos el programa ocurre lo siguiente:



Esto se debe a que los datos son strings, así que al aplicar el operador de suma (+), lo que sucede en realidad es una concatenación. Para poder realizar la suma propiamente dicha necesitamos primero convertir (*castear*) los datos a números. La forma de transformar un string en un número, en este caso en un entero, es usando la función `int`:

```
num1 = input("Ingrese el primer número: ")
num2 = input("Ingrese el segundo número: ")
suma = int(num1) + int(num2)
print("La suma es", suma)
```

```
Ingrese el primer número:
2
Ingrese el segundo número:
2
La suma es 4
```

También se puede realizar la conversión en la misma línea donde se pide el dato, es decir:

```
num1 = int(input("Ingrese el primer número: "))
```

Así como existe la función `int()`, usando la función `float()` se puede convertir un string a un número real, y usando `str()` se puede convertir un número (entero o real) a string.

Funciones

En ocasiones, los programas van a ser largos, o realizar varias tareas. Es una buena práctica **modularizar** las diferentes partes del programa, empleando funciones que resuelvan un problema específico, para permitirnos resolver un problema más complejo.

Para escribir una función en Python, utilizamos la palabra clave `def` (de **define**, definir) seguido del nombre de la función. Al igual que con las variables, es buena práctica que el nombre de la función sea descriptivo y nos dé una idea de lo que hace.

```
def saludar(): # Firma de la función, nombre(parámetros):
    # Cuerpo de la función, desplazado hacia la derecha (inde
    print("Hola!")
```

Si ejecutamos el programa así como está, **no hará nada**. Esto debido a que definir la función por sí sola no hace nada, sino que necesitamos **llamar** o **invocar** a la función.

```
def saludar():
    print("Hola!")

saludar()
```

En este caso, sí imprimirá “Hola!” por pantalla.

Como la ejecución del programa es secuencial (línea a línea), primero necesitamos definir la función antes de invocarla. Hacerlo al revés nos dará error.

Parámetros

Las funciones son más útiles cuando pueden recibir valores y trabajar con ellos. A estos valores se los conoce como **parámetros** de la función. Por ejemplo, cuando usamos `print()`, el texto que le pasamos entre paréntesis es un parámetro.

Por ejemplo, podemos modificar la función `saludar()` para mostrar el nombre de la persona:

```
def saludar(nombre):
    print("¡Hola " + nombre + "!")

saludar("Juan")
saludar("Ana")
```

✓ 0.0s

```
¡Hola Juan!
¡Hola Ana!
```

Dentro del cuerpo de la función, podemos utilizar los parámetros refiriéndonos con su nombre, al igual que con las variables.

Es importante notar que cuando tenemos una función que recibe múltiples parámetros, al invocarla se debe respetar el orden establecido en la firma de la función. Por ejemplo:

```
def dividir(dividendo, divisor):
    cociente = dividendo / divisor
    print("El resultado de la división es:", cociente)

dividir(128, 4)
dividir(4, 128)
```

✓ 0.0s

```
El resultado de la división es: 32.0
El resultado de la división es: 0.03125
```

En el primer caso, la operación que estamos haciendo es 128/4, y en el segundo, 4/128.

Dentro de una función podemos crear variables, pero el **ámbito** (*scope*) de esas variables será la propia función. Es decir que no podemos usar una variable que se creó dentro de una función, fuera de esta.

```
def dividir(dividendo, divisor):
    cociente = dividendo / divisor

dividir(128, 4)
print("El resultado de la división es:", cociente)
```

⊗ 0.0s

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[13], line 5
      2   cociente = dividendo / divisor
      4 dividir(128, 4)
----> 5 print("El resultado de la división es:", cociente)

NameError: name 'cociente' is not defined
```

Si necesitamos utilizar ese valor fuera de la función, una forma de solucionarlo es hacer que la función **devuelva** el valor, utilizando la palabra clave `return`:

```
def dividir(dividendo, divisor):
  cociente = dividendo / divisor
  return cociente

cociente = dividir(128, 4)
print("El resultado de la división es:", cociente)
```

✓ 0.0s

```
El resultado de la división es: 32.0
```

Debido a que la función devuelve un valor, necesitaremos guardar ese valor en una variable.

La instrucción `return` le devuelve el valor que especifiquemos al código que invocó la función, y acto seguido sale de la misma. Si hay más instrucciones dentro del cuerpo de la función, debajo de la palabra clave `return`, no se ejecutarán.

```
def dividir(dividendo, divisor):
  cociente = dividendo / divisor
  return cociente
  print("Fin de la función dividir") # Esto nunca se ejecutar

cociente = dividir(128, 4)
print("El resultado de la división es:", cociente)
```

Las funciones pueden devolver múltiples valores, los cuales se separan con una coma:

```
def sumar_y_restar(num1, num2):
    suma = num1 + num2
    resta = num1 - num2
    return suma, resta

suma, resta = sumar_y_restar(10, 5)
print("La suma es:", suma) # Imprime -> La suma es: 15
print("La resta es:", resta) # Imprime -> La resta es: 5
```