01 - VARIABLES Y TIPOS DE DATOS

1. Variables

Una variable es un espacio en memoria que se utiliza para almacenar un valor. En Python, las variables se crean cuando se les asigna un valor. No es necesario declararlas previamente como en otros lenguajes de programación. El tipo de dato de la variable se determina en tiempo de ejecución, según el valor que se le asigne.

Para asignar un valor a una variable se utiliza el operador de asignación **=**. El valor de la derecha se asigna a la variable de la izquierda.

Ejemplos:

In [ ]:
```
a = 5
b = 3.14
c = "Hola"
```

En el ejemplo anterior, se crearon tres variables: a, b y c. La variable a es de tipo **entero**, la variable b es de tipo **flotante** y la variable c es de tipo **cadena de caracteres**.

Para mostrar el valor de una variable se utiliza la función **print()**.

Ejemplos:

In [ ]:
```
print(a)
print(b)
print(c)
```

```
5
3.14
Hola
```

Los nombres de las variables pueden contener letras, números y el carácter guión bajo (_). No pueden comenzar con un número. Python distingue entre mayúsculas y minúsculas, por lo que las variables **a** y **A** son diferentes.

En la funcion **print( ) podemos escribir texto utilizando comillas **" "**. Para imprimir el valor de una variable luego de un texto se separa con una coma (**,**) .*

Ejemplos:

In [ ]:
```
a = 5
A = 10
print("La variable a es:", a, "y la variable A es:", A)
```

```
La variable a es: 5 y la variable A es: 10
```

Los nombres de las variables deben ser descriptivos, para que el código sea más fácil de entender. Por ejemplo, si se desea almacenar el nombre de una persona, se puede utilizar la variable **nombre**. Si se desea almacenar la edad de una persona, se puede utilizar la variable **edad**.

Si utilizamos f antes de las comillas, podemos utilizar variables dentro de la cadena de texto, utilizando llaves **{ } para indicar que es una variable.***

In [ ]:
```
nombre = "Juan"
edad = 25
print(f"Mi nombre es: {nombre} y tengo {edad} años")
```

```
Mi nombre es: Juan y tengo 25 años
```

Convención de nombres

Para simplificar la tarea de leer el código, se utilizan algunas convenciones para nombrar las variables, funciones, clases, etc.

Convenciones para nombrar variables:

- Se utiliza minúscula para el primer caracter del nombre de la variable.
- Si el nombre de la variable está compuesto por más de una palabra, se separan las palabras con guión bajo (_).

Ejemplos:

```
variable = 10
numero_decimal = 5.78
apellido_persona = "Perez"
```

Convenciones para nombrar funciones:

- Se utiliza la misma convención que para las variables.

Ejemplos:

```
def calcular_area_circulo(radio):
    area = 3.14 * radio ** 2
    return area

def contar(n):
    for i in range(n):
        print(i)
```

Convenciones para nombrar clases:

- Se utiliza mayúscula para el primer caracter del nombre de la clase.
- Si el nombre de la clase está compuesto por más de una palabra, se utiliza mayúscula para el primer caracter de cada palabra.

Ejemplos:

```
class Persona:
    def __init__(self, nombre, apellido):
        self.nombre = nombre
        self.apellido = apellido
class UsarApiClima:
    def __init__(self, api_key):
        self.api_key = api_key
```

Convenciones para nombrar constantes:

- Se utiliza mayúscula para el nombre de la constante.
- Si el nombre de la constante está compuesto por más de una palabra, se separan las palabras con guión bajo (_).

Ejemplos:

```
PI = 3.14
VELOCIDAD_LUZ = 299792458
```

Convenciones para nombrar módulos:

- Se utiliza minúscula para el nombre del módulo.
- Si el nombre del módulo está compuesto por más de una palabra, se separan las palabras con guión bajo (_).

Ejemplos:

```
import math
import random
import mi_modulo
```

2. Tipos de datos

En Python existen varios tipos de datos. Para saber el tipo de dato de una variable se utiliza la función **type()**. Los más utilizados son:

Números

Los números se clasifican en:

- Enteros **(int)**: Los números enteros son aquellos que no tienen parte decimal. Por ejemplo: 1, 2, 3, 4, 5, etc.
- Flotantes **(float)**: Los números flotantes son aquellos que tienen parte decimal. Por ejemplo: 1.2, 3.14, 5.678, etc.

**Para escribir un número flotante se utiliza el carácter punto (.) como separador decimal.*

Ejemplos:

In [ ]:
```
a = 10 # Entero
b = 5.78 # Flotante
print("La variable a es de tipo:", type(a), "y la variable b es de tipo:", type(b))
```

```
La variable a es de tipo: <class 'int'> y la variable b es de tipo: <class 'float'>
```

Operaciones entre números

Para realizar operaciones entre números se utilizan los siguientes operadores:

- Suma **(+)**: Se utiliza para sumar dos números.
- Resta **(-)**: Se utiliza para restar dos números.
- Multiplicación **(*)**: Se utiliza para multiplicar dos números.
- División **(/)**: Se utiliza para dividir dos números.
- División entera **(//)**: Se utiliza para dividir dos números y obtener el cociente entero.
- Módulo **(%)**: Se utiliza para obtener el resto de la división entre dos números.
- Potencia (**): Se utiliza para elevar un número a otro número.

Cadenas de caracteres

Las cadenas de caracteres o strings se utilizan para almacenar texto. Para escribir una cadena de caracteres se utilizan comillas simples ('') o dobles (""). Por ejemplo: 'Hola', "Mundo", '123', "3.14", etc.

Las cadenas de caracteres se pueden concatenar utilizando el operador **+**.

Ejemplo:

In [ ]:
```
a = "Hola"
b = "Mundo"
print(a+b)
```

```
HolaMundo
o
```

Cuando se desea escribir una cadena de caracteres que ocupa más de una línea, se utiliza triple comilla simple (''') o triple comilla doble (""").

Ejemplo:

In [ ]:
```
menu="""
MENU
1) Sumar        2) Restar
3) Multiplicar  4) Dividir
5) Salir
"""
print(menu)
```

```
MENU
1) Sumar        2) Restar
3) Multiplicar  4) Dividir
5) Salir
```

Podemos convertir variables a diferentes tipos de datos utilizando las funciones **int(), **float()** y **str()**.***

Entrada de datos por teclado

Para ingresar datos por teclado se utiliza la función **input()**. Esta función muestra un mensaje en pantalla y espera a que el usuario ingrese un valor. Cuando el usuario presiona la tecla Enter, la función devuelve el valor ingresado como una **cadena de caracteres**.

In [ ]:
```
entrada = input("Ingrese una variable por teclado")
print("Ingreso",entrada,"y la variable es de tipo",type(entrada))
```

```
Ingreso  y la variable es de tipo <class 'str'>
```

Booleanos

Los booleanos son un tipo de dato que puede tener dos valores: **True** o **False**. Se utilizan para representar valores de verdad. Por ejemplo: 1 > 2 es falso, 2 == 2 es verdadero, etc. También se pueden utilizar para representar estados. Por ejemplo: si una luz está encendida, el valor es True, si está apagada, el valor es False, o para inicializar un bucle.

Tanto True como False deben escribirse con la primera letra en mayúscula ya que son palabras reservadas de Python.

Ejemplos:

In [ ]:
```
luz = False
if luz == True:
    print("La luz está prendida")
else:
    print("La luz está apagada")
```

In [ ]:
```
contador = 0
while True:
    contador += 1
    print(contador, end=" ")
    if contador == 5:
        break
print("\nFin del ciclo")
```

```
1 2 3 4 5
Fin del ciclo
```

Listas

Las listas son un tipo de dato que se utiliza para almacenar varios valores. Se pueden almacenar valores de diferentes tipos. Para escribir una lista se utilizan corchetes ([ ]) y se separan los valores con comas (,). Por ejemplo: [1, 2, 3, 4, 5], ["Hola", "Mundo"], [1, "Hola", 3.14, True], etc.

Ejemplos:

In [ ]:
```
lista_vacia = []
lista = list()
```

```
numeros = [1,2,3,4,5]
autos = ["Mazda","Toyota","Honda","Ford"]
mixto = [5.5,10,"Texto",False]

print(lista)
print(numeros)
print(autos)
print(mixto)
```

```
[]
[1, 2, 3, 4, 5]
['Mazda', 'Toyota', 'Honda', 'Ford']
[5.5, 10, 'Texto', False]
```

Para acceder a un elemento de la lista se utiliza el índice del elemento. Tanto en listas como en cadenas de caracteres, el primer elemento tiene índice 0. Podemos modificar el valor de un elemento de la lista utilizando el índice.

Por ejemplo: si se desea acceder al primer elemento de la lista, se utiliza el índice 0, si se desea acceder al segundo elemento de la lista, se utiliza el índice 1, etc.

Tambien podemos acceder a los elementos de la lista utilizando índices negativos.

Por ejemplo: si se desea acceder al último elemento de la lista, se utiliza el índice -1, si se desea acceder al penúltimo elemento de la lista, se utiliza el índice -2, etc.

In [ ]:
```
mi_lista = ["Hola", "Mundo", 5, True, 3.8]

print(mi_lista[1])
mi_lista[1] = "Chau"
print(mi_lista[1])
print(mi_lista[-1])
```

```
3.8
```

Las listas se pueden modificar. Se puede agregar un elemento al final de la lista utilizando el método **append()**. Tambien se puede agregar un elemento en una posición determinada utilizando el método **insert()**.

Es posible concatenar dos o mas listas utilizando el operador **+**.

Para eliminar un elemento de la lista podemos utilizar el método **remove()** o el método **del**. El método **pop()** elimina el elemento de la posición indicada y devuelve el valor eliminado. Si no se indica la posición, elimina el último elemento de la lista.

El método **clear()** elimina todos los elementos de la lista.

In [ ]:
```
numeros.append(6)
numeros.remove(3)
autos.insert(1,"Chevrolet")
mixto.pop()
```

```
print(numeros)
print(autos)
print(mixto)
```

Otra forma de anexar elementos a una lista es utilizando el operador **+=**. Tambien se puede utilizar el metodo **extend()**.

Ejemplos:

In [ ]:
```
lista = [1, 2, 3, 4, 5]
lista += [6, 7, 8, 9, 10]
print(lista)
lista.extend([11, 12, 13, 14, 15])
print(lista)
```

Si deseamos copiar una lista, podemos utilizar el método **copy()** o el método **list()**.

Es importante aclarar que si simplemente hacemos una lista igual a otra, al modificar la segunda estaremos modificando la original

Ejemplos:

In [ ]:
```
lista_1 = [1, 2, 3, 4, 5]
lista_2 = lista_1.copy()
lista_3 = list(lista_2)

lista_modificada = lista_1
lista_modificada[0] = 10    # También modifica lista_1

lista_2[0] = 20             # No modifica lista_1

print("Lista 1:",lista_1,"Lista modificada:",lista_modificada)

print("Lista 2:",lista_2,"Lista 3:",lista_3)
```

Otra operacion que podemos realizar con las listas es cortarlas. Para ello utilizamos el operador **[inicio : final : paso]**.

En caso de iniciar en 0, finalizar en el último elemento o utilizar paso 1, podemos omitirlos.

El final no es inclusivo, es decir, el elemento del índice final no se incluye en la lista resultante.

Ejemplos:

In [ ]:
```
mi_lista = [10,20,30,40,50]
mi_lista_cortada = mi_lista[1:4]
print(mi_lista[:3])
print(mi_lista[2:4])
print(mi_lista_cortada)
print("Mi lista al revés:",mi_lista[::-1])
```

Es posible crear matrices utilizando listas anidadas. Por ejemplo: [[1, 2, 3],[4, 5, 6],[7, 8, 9]].

Esta matriz tiene 3 filas y 3 columnas, y se puede representar de la siguiente manera:

1	2	3
4	5	6
7	8	9

Para acceder a un elemento de la matriz se utiliza el índice de la fila y el índice de la columna.

Ejemplos:

In [ ]:
```
matriz = [ [1,2,3], [4,5,6], [7,8,9] ]

print(matriz[2][2])

# print(matriz[0][0])
# print(matriz[2][2])
matriz[1][1] = 10
print(matriz[1])
print(matriz[1][1])
# print(matriz)
```

```
9
[4, 10, 6]
10
```

Tuplas

Las tuplas son un tipo de dato que se utiliza para almacenar varios valores. Se pueden almacenar valores de diferentes tipos. Para escribir una tupla se utilizan paréntesis (( )) y se separan los valores con comas (,). Por ejemplo: (1, 2, 3, 4, 5), ("Hola", "Mundo"), (1, "Hola", 3.14, True), etc.

A diferencia de las listas, las tuplas no se pueden modificar. Una vez creada la tupla, no se pueden agregar, eliminar o modificar los elementos.

In [ ]:
```
mi_tupla = tuple()
mi_tupla_2 = (("Lunes", "Martes", "Miercoles"), 2, 3, 4)

lista_interna = mi_tupla_2[0]
print(lista_interna)

# print(mi_tupla_2)
# print(type(mi_tupla_2))
```

```
('Lunes', 'Martes', 'Miercoles')
```

Diccionarios

Los diccionarios son un tipo de dato que se utiliza para almacenar varios valores. Se pueden almacenar valores de diferentes tipos. Para escribir un diccionario se utilizan llaves ({ }) y se separan las claves y los valores con dos puntos (:). Por ejemplo: {"nombre": "Juan", "apellido": "Perez"}, {"nombre": "Maria", "apellido": "Gomez"}, etc. Tambien podemos utilizar el método **dict()**.

Los diccionarios se pueden modificar. Se puede agregar un elemento utilizando una clave que no exista. Tambien se puede modificar el valor de un elemento utilizando la clave.

Para eliminar un elemento del diccionario se utiliza la palabra reservada **del**.

Para acceder a un elemento del diccionario se utiliza la clave del elemento como si fuera un índice. Tambien se puede utilizar el método **get()**.

Ejemplos:

In [ ]:
```
alumno_1 = {"nombre" : "Juan", "apellido" : "Perez", "edad" : 25, "notas" : [10,9,7

print(alumno_1["edad"])


# print("El alumno 1 se llama",alumno_1["nombre"],alumno_1["apellido"],"y tiene",al
# print("El alumno 2 se llama",alumno_2["nombre"],alumno_2["apellido"],"y tiene",al

nombre_alumno_1 = alumno_1.get("nombre")
# print(nombre_alumno_1)
```

```
25
25
```