SINTAXIS BÁSICA Y OPERACIONES FUNDAMENTALES

GRUPO 2

SINTAXIS BÁSICA

- La sintaxis se define como el conjunto de reglas a seguir para la correcta interpretación del código por humanos y máquinas.
- Python se diseña como un lenguaje altamente legible, por lo que destaca su simplicidad.
- Uso principal de palabras clave en inglés, para facilitar su legibilidad.

"There should be one obvious way to do it"

- Python se reserva una serie de palabras clave para funciones internas, tales como: "while", "for", "True", "False"...
- También se reserva una serie de símbolos, que definirán diferentes funcionalidades en Python: "()", "[]", "{}", " ' ' ", ":", "."
- Para mantener una comunicación entre usuarios se hace uso de comentarios, este código no se ejecuta y no incluye limitaciones de palabras o símbolos.
- Mediante "#" delimitamos comentarios de una línea, y mediante tres comillas simples comentarios de documentación (Docstring).

```
# Esto es un comentario
''' Esto es un docstring'''
```

- Para delimitar el *control flow*, Python utiliza indentación.
- De esta forma se definen los bloques de código. Empleado para funciones, bucles, etc.

```
import pandas as pd

pd.read_csv("datafile.

variables = {"c1": [1

repeat = True
def renombrar():
    while repeat ==
        print("a")
```

ESTRUCTURA

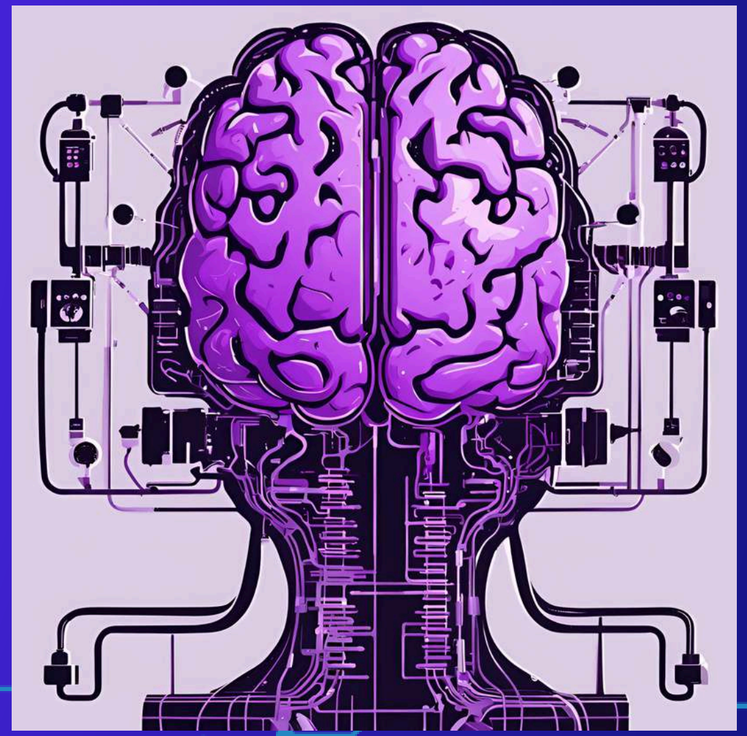
La estructura básica de un programa en Python parte de su ejecución *top-to-bottom*. Es decir, se ejecuta de forma secuencial.

Por ello han de definirse antes de su uso los objetos incorporados en un programa.

En este sentido encontraremos una estructura común en la que aparecen los siguientes elementos:

- Bibliotecas: Importadas al principio del programa para utilizar las funciones que contienen.
- Funciones: Permitirán realizar las operaciones básicas de Python.
- Variables: Guardan en la memoria de trabajo los datos que vayamos a utilizar a lo largo del programa.

CLASIFICACIÓN DE LOS TIPOS DE DATOS



NUMÉRICOS: (INT), (FLOAT), (COMPLEX),

- BOOLEANOS: (BOOL)

```
>>> TYPE(TRUE)
<CLASS 'BOOL'>
>>> ISINSTANCE(TRUE, INT)
TRUE
```

SECUENCIAS: (LIST), (TUPLE), (RANGE)

CADENAS DE CARACTERES: (STR)

SECUENCIAS BINARIAS: (BYTES), (BYTEARRAY), (MEMORYVIEW)

CONJUNTOS: (SET), (FROZENSET)

MAPAS: DICCIONARIOS

TIPOS DE DATOS:

MÁS REPRESENTATIVOS

BOOLEANOS

TIPO: BOOL

Representan valores de verdad, corresponden a los valores:

True
False

Nº ENTEROS

TIPO: INT

Son números enteros (o discretos) y son SIN decimales. Pueden ser positivos o negativos.

Ej. **3, 5, -8, 30_500**

Nº FLOTANTE

TIPO: FLOAT

Son números CON decimales (o continuos), siempre incluirán un punto (.) para separar el entero de los decimales.

Ej. **5.00**
3.45,
-5.67,
4.32e-2

COMPLEJOS

TIPO: COMPLEX

Son aquellos números con una parte real y parte imaginaria donde j = √-1 y gracias a ellos se pueden explicar sucesos naturales.

Ej. **2 + 3j,**
5j,

CADENA

TIPO: STR

Son cadenas de texto y utilizan caracteres UNICODE. Para que Pyhton los reconozca como caracteres han de estar entre comillas simples (') o dobles (").

Ej. **"56"**
"C/Ñu, nº89"
'País'
"teléfono de contacto"

TIPOS DE DATOS:

TIPO DE INFORMACIÓN QUE PUEDES INTRODUCIR

TUPLA

(TUPLE)

Conjuntos de datos ordenados e inmutables que pueden ser del mismo tipo o de otro.

```
>>>(2, 4.5, 'Project Euler')
```

LISTAS

(LIST)

Colecciones ordenadas y modificables. Pueden ser de igual o diferente tipo.

```
>>>['C', 'Java', 'SQL']
>>>['nombre', 57, 5,57]
```

CONJUNTO

(SET)

Colección desordenada de elementos únicos.
Permiten trabajar con operaciones de conjuntos (unión, intersección o diferencia)

```
>>> set('Carry', 'Carol', 'Ira', 'Leire', 'Naty')
>>>{'Carry', 'Carol', 'Ira', 'Leire', 'Naty'}
```

DICCIONARIO

(DICT)

Estructura de datos que almacena pares clave-valor. Las claves no pueden tener duplicados, aunque los valores si.

```
>>>{'Grupo 1': ['Nombre 1', 'Nombre2'], 'Año': [1984, 2001]}
>>>dict([('Nombre', 'Sara'), ('Edad', 27),('DNI', 1003882),])
>>>dict(Nombre='Sara',Edad=27,DNI=1003882)
```

ARITMÉTICA

UN OPERADOR ES UN SÍMBOLO DEL LENGUAJE DE PROGRAMACIÓN, EL CUAL ES CAPAZ DE REALIZAR OPERACIONES CON LOS VALORES.

PYTHON SOPORTA OPERACIONES ARITMÉTICAS BÁSICAS QUE SON FUNDAMENTALES PARA CUALQUIER TIPO DE PROGRAMACIÓN:

SUMA (+):

AÑADE DOS NÚMEROS.

RESTA (-):

RESTA UN NÚMERO DE OTRO.

MULTIPLICACIÓN (*):

MULTIPLICA DOS NÚMEROS.

POTENCIA (**):

ELEVA UN NÚMERO A LA POTENCIA DE OTRO.

DIVISIÓN (/):

DIVIDE UN NÚMERO POR OTRO, EL RESULTADO ES UN FLOTANTE.

DIVISIÓN ENTERA (//):

DIVIDE UN NÚMERO POR OTRO Y DEVUELVE SOLO LA PARTE ENTERA DEL RESULTADO.

MÓDULO (%):

DEVUELVE EL RESIDUO DE LA DIVISIÓN DE UN NÚMERO POR OTRO.

OPERADORES CADENA

Estos dos operadores aritméticos: + y * tienen una función secundaría: son capaces de manejar o manipular cadenas, aunque, en una manera muy específica

El signo de + , al ser aplicado a dos cadenas, se convierte en un operador de concatenación y simplemente concatena (junta) dos cadenas en una.

IMPORTANTE
NO ES UN SUMADOR, DEBES ASEGURATER QUE AMBOS ARGUMENTOS SEAN CADENAS.

El signo de * , cuando es aplicado a una cadena y a un número (o viceversa) se convierte en un operador de replicación.
Replica la cadena por el número de veces indicado.

RECUERDA
UN NÚMERO MENOR O IGUAL A CERO PRODUCE UNA CADENA VACÍA.

ENTRADA Y SALIDA DE DATOS

Ya conocemos la función print() que se usa para mostrar datos en la pantalla

La función:

1. toma sus argumentos
2. los convierte a una manera legible si es necesario
3. envía los datos resultantes al dispositivo de salida

Ahora se introducirá una nueva función, la cual pareciese ser un reflejo de la función print().

Mientras print() envía datos a la consola. input() obtiene datos de ella.

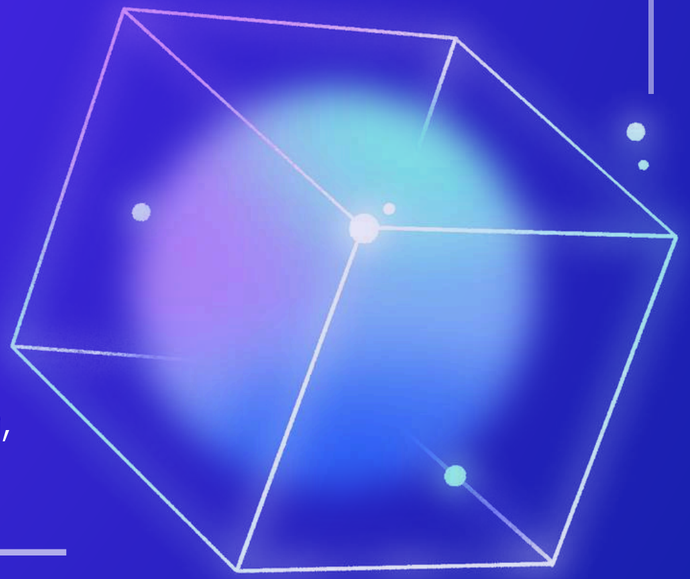
La función:

- solicita al usuario que inserte algún dato desde la consola
- puede ser invocada sin argumentos

RECUERDA

EL RESULTADO DEBE SER ASIGNADO A UNA VARIABLE

Después se utiliza la función print() para mostrar los datos que se obtuvieron, con algunas observaciones adicionales.



VARIABLES

¿QUÉ SON?

Las variables son contenedores de datos que se encuentran en la memoria del ordenador para poder operar con ellos.

Se crean en el momento en que se les asigna un valor

¿ CÓMO SE DECLARAN?

Se les asigna nombre con el operador de asignación “=” y no es necesario definir el tipo de variable.

nombre_variable = Valor de variable

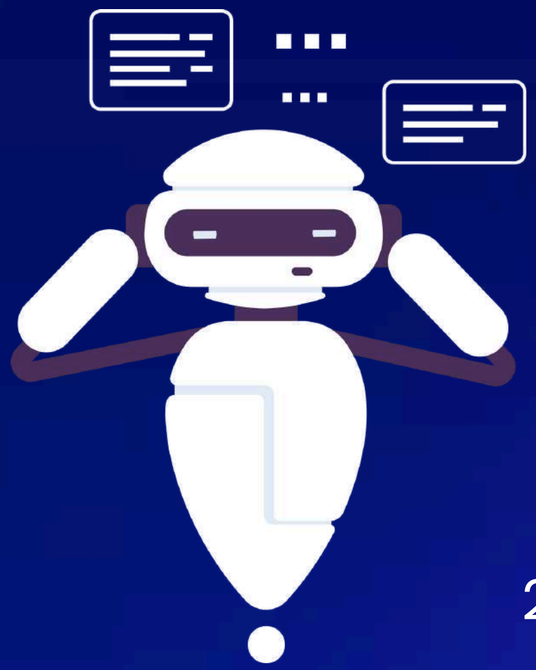
ASIGNACIÓN DE VALOR

Se puede asignar de forma individual

x = valor

o múltiples valores a múltiples variables en una sola instrucción

x, y, z = 10, "hola", {"leche", True}



VARIABLES

CONVENCIONES Y BUENAS PRÁCTICAS

1. Conviene asignar un nombre descriptivo que indiquen propósito y contenido. Preferiblemente en minúsculas. Python hace distinciones entre mayúsculas y minúsculas.
2. Se pueden usar letras, números y guiones bajos en el nombre de la variable. pero no se puede empezar con un número.
3. Para separar palabras, se utiliza la convención Snake Case: todas las palabras en minúsculas y separarlas con guiones bajos (_).
4. No utilizar palabrar reservadas como if, else, while, for.
5. Utilizar sustantivos singulares en lugar de plurales.
6. Ser consistente con las convenciones de tu propio código y del equipo y organización con quien estés trabajando.

TIPOS DE VARIABLES

SEGÚN TIPO DE DATOS

DATOS SIMPLES

CADENAS DE TEXTO (STRING)

string = ”Soy una variable de texto”

NUMERICAS

numerica = 10

BOOLEANAS

booleana = True

DATOS COMPUESTOS

LISTAS (LIST)

lista_mixta = [1, "hola", 3.14, True]

TUPLAS (TUPLE)

tupla_mixta = (1, "hola", 3.14, True)

CONJUNTOS (SET)

conjunto_mixto = {1, "hola", 3.14, True, }

DICCIONARIOS (DICT)

diccionario_mixto = { "id": 1, "nombre": "Luis", "altura": 1.75, "activo": True }

SEGÚN ÁMBITO

GLOBALES

```
global_variable = "Soy una variable global"
```

LOCALES

```
def funcion_con_local():
    local_variable = "Soy una variable local”
```

SEGÚN MUTABILIDAD

MUTABLES

INMUTABLES

TIPOS DE VARIABLES

SEGÚN USO

CONTROL O ITERACION

Usadas en bucles para controlar la cantidad de iteraciones.

i, j, k, index

ENTRADA (INPUT)

Almacenan datos que provienen de entradas del usuario o de otras fuentes externas.

input_data, user_input

SALIDA (OUTPUT)

Almacenan datos que serán enviados fuera del programa o funciones.

output_data, user_output

DE ESTADO

Mantienen el estado de un sistema o una parte del programa

is_running, current_state

TEMPORALES

Usadas para almacenar datos de forma temporal durante cálculos o transformaciones.

temp, buffer, intermediate_result

DE CONFIGURACION

Almacenan valores que configuran o parametrizan el comportamiento del programa.

config, settings, options.

CONTADORES

Usadas para llevar cuenta de eventos o elementos.

count, counter, num_items

FLAGS O BANDERAS

Variables booleanas que indican el estado de alguna condición.

is_valid, has_error, flag

ACUMULATIVAS

Almacenan el resultado acumulado de una operación iterativa.

sum, total, accumulator

AUXILIARES

Variables de apoyo utilizadas para simplificar operaciones o cálculos.

aux, temp_var

OPERADORES

COMPARACIÓN

Se utilizan para comparar dos valores y devuelven un resultado booleano (True o False). Algunos ejemplos son el operador de igualdad (==), el operador de desigualdad (!=), el operador de mayor que (>), el operador de menor que (<), entre otros.

LÓGICOS

Se utilizan para combinar expresiones condicionales y devuelven un resultado booleano. Algunos ejemplos son el operador AND (and), el operador OR (or) y el operador NOT (not).

DE IDENTIDAD

Se utilizan para verificar si dos variables hacen referencia al mismo objeto en memoria. Algunos ejemplos son el operador de igualdad de identidad (is) y el operador de desigualdad de identidad (is not).

DE PERTENENCIA

Se utilizan para verificar si un valor está presente en una secuencia. Algunos ejemplos son el operador de pertenencia (in) y el operador de no pertenencia (not in).

DE CONJUNTOS

Se utilizan para incluir, excluir o intersectar conjuntos de datos

OPERADORES

COMPARACIÓN

(==)

```
normal='pablo neruda'
title='Pablo Neruda'
normal==title
```

#Devuelve False

(!=)

```
print('9'!=9)
```

#Devuelve True

(>)

```
long1=len('roma')
long2=len('amor')
(long1>long2)
```

#Devuelve False

LÓGICOS

x = 5
y = 10

AND

```
(x > 2 and y < 15)
```

Devuelve True

OR

```
(x > 2 or y > 15)
```

Devuelve True

DE PERTENENCIA

l=[1,2,3]

in

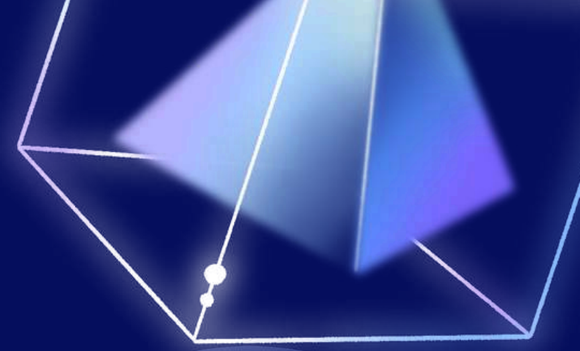
```
(f'El {9 in l}')
```

#Devuelve False

not in

```
(f'El{2 not in l}')
```

#Devuelve False



DE IDENTIDAD

num=9

is

```
num is 9
```

#Devuelve True

DE CONJUNTOS

l1={1,2,3}
l2={4}

```
c1 | c2
{1, 2, 3, 4}
```

THANK YOU!