MÓDULOS: ESP - ECP

EL LENGUAJE PYTHON

José L. Berenguel

Tabla de Contenidos

1. Introducción al lenguaje Python

- Solo se permite **una instrucción por línea**.
- No hay carácter de final de instrucción (el típico punto y coma)
- La **tabulación** o **indentación del código** determina a qué nivel de estructura de control se aplican las instrucciones de un determinado nivel (no hay caracteres de bloques, las típicas llaves {}). Esto se denomina **ámbito** o **scope**.
- **No se declaran las variables ni su tipo**. El tipo de la variable se decide en el momento de la asignación.

https://docs.python.org/3/library/index.html

1. Introducción al lenguaje Python

- El carácter # se utiliza para realizar comentarios de una sola línea (todo lo que haya a continuación se considera comentario.
- Se pueden crear **comentarios multilínea** con los caracterés comillas dobles y comillas simples por triplicado.

```
# Comentario de una línea
# Otro comentario de línea

"""
    Este es un comentario
    de varias líneas.
    No se ejecutará.
"""

'''
  Otro comentario de
  varias líneas
'''
```

2. Variables y tipos de datos

- Las variables se declaran asignándole un valor a su **identificador**.
- Pueden tener cualquier longitud y contener letras, números y el carácter _ (subrayado).
- Preferiblemente en minúscula (**snake_case**).
- No pueden comenzar con un número.

```
# Declarando variables
numero = 16
pi = 3.1416
nombre = "Jose L."
apellidos = 'Berenguel Gómez'
nombre_instituto = 'IES Zaidín-Vergeles'
```

2. Variables y tipos de datos

- No se puede utilizar como identificador una cadena que coincida con una palabra reservada del lenguaje.
- Los editores de código suelen resaltar las palabras reservadas.
- Principales palabras reservadas en Python:

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

2. Variables y tipos de datos

- Se pueden inicializar o asignar varias variables en una sola sentencia.
- Es posible escribir varias sentencias en una misma línea separándolas por **;** (punto y coma)

```
# Asignación de valores a variables
a = b = c = 7
x, y = 4, 2
a, b, c, = 1, 2, 3
```

```
# Varias sentencias en una línea
x = 5; y = 10
```

2. Variables y tipos de datos

- Las variables tienen un alcance (**scope**) o **ámbito** según el lugar donde se definen.
- Pueden existir variables distintas con el mismo nombre o identificador si su ámbito es diferente.

```
# Ámbito de las variables
x = 10

def funcion():
    x = 5 # ámbito local a la función

funcion()
print(x)  # ¿Qué valor se imprime?
```

2. Variables y tipos de datos

- Tipos de datos básicos o primitivos (**built-in**) en Python.
  - Tipos numéricos: **int**, **float** y **complex**.
  - Tipo lógico: **bool** (valores True o False).
  - Secuencias: **list**, **tuple**, **range**.
  - Cadenas: **str**.
  - Secuencias binarias: **bytes**, **bytearray**.
  - Conjuntos: **set**, **frozenset**.
  - Mapas: **dict**.

https://docs.python.org/3/library/stdtypes.html

2. Variables y tipos de datos

- Decoración de tipos (**type hint**) en Python.
  - Sirven para indicar el **tipo de dato esperado** para una variable, parámetro de una función, valor devuelto para una función, etc.
  - Facilitan la legibilidad del código y el análisis estático.
  - ¡Python no valida estos tipos de datos!

```
# Declaración de variables
nombre: str = "Ana"
edad: int = 30
# Parámetro y valor de retorno
def saludar(nombre: str) -> str:
  return f"Hola, {nombre}"
```

https://docs.python.org/3/library/typing.html

2. Variables y tipos de datos

- Decoración de tipos (**type hint**) en Python.
  - Para tipos complejos como listas, tuplas y diccionarios hay que importar el módulo **typing**.

```
from typing import List, Dict, Tuple

numeros: List[int] = [1, 2, 3]
usuario: Dict[str, str] = {"nombre": "Carlos", "email": "carlos@ieszaidinvergeles.org"}
coordenadas: Tuple[float, float] = (10.5, 20.3)
```

https://docs.python.org/3/library/typing.html

3. Lectura y escritura de datos

- **Entrada de datos por teclado**
  - Se utiliza la función **input(mensaje)**.
  - La función devuelve un tipo **str**. Si necesitamos operar con tipos numéricos debemos realizar una conversión de tipos con las funciones **int()** y **float()**.
- **Salida de datos por pantalla**
  - Se utiliza la función **print(mensaje1, mensaje2, ...)**.
  - La función **print** añade un espacio entre los valores recibidos por parámetro.

```
nombre = input("Introduce tu nombre")
apellidos = input("Introduce tu apellido")
print("Te llamas", nombre, apellidos)
```

4. Operadores

Operador	Descripción	Ejemplo
+, - ,	Suma, Resta	3 + 5, 3 - 2
* ,/, %	Multiplicación, División, Módulo	12 * 3, 5 / 6, 12 % 3
//	División truncada (resultado entero)	9 // 2 = 4
**	Exponente	5**2 = 25
+=, -=	Suma/Resta y asignación	x += 2 –> x = x + 2
*=, /=	Multiplicación/División y asignación	x *= 2 –> x = x * 2
**=, //=	Exponente/División truncada y asignación	x //= 2 –> x = x // 2
or, and, not	Operadores lógicos	if not terminado:
in	Está en la lista	if 2 in [1, 2, 3]:
<, ⇐, >, >=, !=, ==	Operadores de comparación	if x != y:

5. Sentencias de control

- **Sentencia condicional:**

```
if <condicion1>:
    <sentencias1>
elif <condicion2>:
    <sentencias2>
. . .
else:
    <sentencias_else>
```

```
x = int(input("Introduce n: "))
if x < 0:
    x = 0
    print('Puesto a cero')
elif x == 0:
    print('Cero')
elif x == 1:
    print('Simple')
else:
    print('Más')
```

¡LA INDENTACIÓN DE CÓDIGO DETERMINA LAS SENTENCIAS QUE SE EJECUTAN EN EL INTERIOR DE CADA BLOQUE!

5. Sentencias de control

- **Operador ternario:**

```
# SINTÁXIS DEL OPERADOR TERNARIO
# [código si se cumple] if [condición] else [código si no se cumple]

# Ejemplo número par o impar con operador ternario
n = 6
paridad_numero = "es par" if n%2==0 else "es impar"
print(paridad_numero)

# Ejemplo número par o impar con if
n = 6
if n % 2 == 0:
    print("Es par")
else:
    print("Es impar")
```

5. Sentencias de control

- **Bucle mientras:**

```
while <condicion>:
    <sentencias>
    # En cualquier momento podemos romper el bucle
    [break]
[else:
    # Sólo irá aqui si el bucle termina de forma natural
    <sentencias_finales>]
```

```
i = 1
while i < 6:
  print(i)
  i += 1
```

5. Sentencias de control

- **Bucle for (para o desde):**

```
for <variable> in <lista_valores>:
    <sentencias>
    # Opcionalmente podemos romper el bucle
    [break]
    # Opcionalmente podemos continuar con la siguiente iteración
    [continue]
[else:
    <sentencias_finales>]
```

```
# Midiendo cadenas de texto
palabras = ['gato', 'ventana', 'defenestrado']
for p in palabras:
    print(p, len(p))
```

5. Sentencias de control

- **Bucle for (para o desde):**
  - Función range devuelve una secuencia de números que se puede usar para iterar: **range([start,] stop [, step])**

```
# imprime 0 1 2
for i in range(3):
    print(i)

# imprime los numeros del 5 al 10
for i in range(5, 10):
    print(i)

# imprime los impares del 1 al 10
for i in range(1, 10, 2):
    print(i)
```

5. Sentencias de control

- **Bucle for (para o desde):**
  - El bucle for puede recorrer cualquier elemento de Python que sea **iterable**.
  - Aquellos elementos que pueden ser recorridos mediante índices o implementan un **iterador (interfaz iterable)**.
  - Cadenas, listas, diccionarios, tuplas y conjuntos son objetos iterable en Python.

```
# Imprime carácter a carácter
for i in "IES Zaidín-Vergeles":
    print(i)

lista = [[56, 34, 1],
         [12, 4, 5],
         [9, 4, 3]]
# Imprime las 3 sublistas
for i in lista:
    print(i)

# Imprime cada elemento individual
for i in lista:
    for j in i:
        print(j)
```

5. Sentencias de control

- **Uso de pass, break y continue:**
  - La palabra reservada **pass** se emplea cuando se desea crear una estructura de control vacía. El objetivo es posponer la implementación y que no se muestre ningún error en la ejecución.
  - Las sentencias **break** y **continue** permiten modificar el comportamiento normal de un bucle.
    - **Break** finaliza la ejecución del bucle (si es un bucle anidado se devuelve la ejecución al bucle exterior).
    - **Continue** finaliza la iteración actual del bucle comenzando una nueva iteración.
  - El uso de **break** y **continue** debe evitar que el código sea complejo y poco legible, si bien en algunos casos puede estar recomendado.

https://stackoverflow.com/questions/3922599/is-it-a-bad-practice-to-use-break-in-a-for-loop

https://fallenapples.medium.com/avoiding-break-and-continue-statements-in-python-8bf830f3ab12

6. Funciones y módulos built-in

- **Algunas funciones interesantes:**
  - **abs()**: devuelve el valor absoluto de un número.
  - **bin()**: devuelve la representación en binario de un número.
  - **bool()**: devuelve la representación lógica del objeto recibido.
  - **chr()**: devuelve el caracter que representa el valor Unicode especificado.
  - **complex()**: devuelve un número complejo.
  - **dict()**: devuelve un diccionario.
  - **divmod()**: devuelve el cociente y resto de los dos argumentos.
  - **enumerate()**: devuelve un objeto enumerado (añade un contador como clave a los elementos del mismo).
  - **float()**: devuelve un número en punto flotante.
  - **hex()**: convierte un entero en un número en hexadecimal.
  - **input()**: devuelve una cadena introducida por teclado.
  - **int()**: devuelve un valor int desde un número en punto flotante o una cadena.

https://docs.python.org/3/library/functions.html

6. Funciones y módulos built-in

- **Más funciones interesantes:**
  - **len()**: devuelve el número de elementos de una colección o el número de caracteres de una cadena.
  - **list()**: devuelve una lista.
  - **max()**: devuelve el mayor elemento de un iterable.
  - **min()**: devuelve el menor elemento de un iterable.
  - **oct()**: convierte un número en un octal.
  - **open()**: abre un fichero.
  - **ord()**: devuelve el valor Unicode de un carácter.
  - **pow()**: calcula la potencia (similar a a**b).
  - **set()**: devuelve un nuevo objeto set.
  - **slice()**: devuelve un objeto slice.
  - **str()**: devuelve un objeto cadena.
  - **type()**: devuelve el tipo del objeto del objeto recibido.

https://docs.python.org/3/library/functions.html

6. Funciones y módulos built-in

- Se puede extender el número de funciones disponibles a través de los módulos.
- Un módulo o librería contiene funcionalidad relacionada entre sí.
- Por ejemplo, el módulo **math** contiene funciones matemáticas avanzadas como **sqrt()** y constantes como el número **pi** o el número **e**.
- Se puede importar el módulo completo con **import**: **import math**.
- O elementos parciales con **from**: **from math import math, pi**.

```
import math

# Calculamos la raíz cuadrada de 3
math.sqrt(3)

# Error: no se encuentra la función.
# Es necesario usar modulo.funcion.
sqrt(3)
```

```
from math import sqrt

# Error: el módulo math no se ha definido
math.sqrt(3)

# Correcto
# Se llama directamente a la función
sqrt(3)
```

6. Funciones y módulos built-in

- Python ofrece un gran número de módulos. Algunos de ellos:
  - **asincio**. Entrada y salida asíncrona.
  - **base64**. Diferentes tipos de codificación de caracteres.
  - **binascii**. Conversion de binario a ascii.
  - **calendar**. Permite trabajar con diferentes calendarios.
  - **cmath**. Funciones para números complejos.
  - **datetime**. Fechas y tiempo.
  - **email**. Chequeo y manipulación de cadenas que representan direcciones de correo electrónico.
  - **html**. Manipulación de código HTML.
  - **http**. Manipulación y generación de código HTTP cliente y servidor.
  - **io**. Manipulación de flujos de entrada y salida.
  - **json**. Codificación y decodificación de formato JSON.

https://docs.python.org/3/py-modindex.html

6. Funciones y módulos built-in

- Python ofrece un gran número de módulos. Algunos de ellos:
  - **os**. Funciones relacionadas con el sistema operativo.
  - **pickle**. Serialización y deserialización de objetos.
  - **platform**. Funciones relacionadas con la arquitectura hardware.
  - **random**. Manejo de números pseudoaleatorios.
  - **re**. Operaciones de expresiones regulares.
  - **string**. Constantes y manejo de cadenas de caracteres.
  - **tkinter**. Librería para GUI mediante Tcl/Tk.
  - **turtle**. Módulo educativo para aplicaciones gráficas simples.
  - **typing**. Manejo de type hints.
  - **unittest**. Framework para crear test de unidad.
  - **xml**. Manejo y proceso de documentos en formato XML.

https://docs.python.org/3/py-modindex.html

7. Definición de funciones

- Las funciones permiten mejorar la encapsulación y reutilización de código.
- En Python se definen con la palabra reservada **def** seguida del nombre de la función y los parámetros que recibe entre paréntesis (puede no recibir ningún parámetro.
- La sentencia **return** se emplea para indicar el valor devuelto por la función (es opcional).

```
# Definición de función sin parámetros
def saludar():
  print("Bienvenido al curso de especialización en lenguaje Python")

# Llamada a la función definida anteriormente
saludar()
```

```
# Definición de función con parámetros
def saludar(nombre):
  print(f"Bienvenido/a {nombre} al curso de especialización en lenguaje Python")

# Llamada a la función definida anteriormente
saludar("Jose L.")
```

7. Definición de funciones

```python
# Función que devuelve el área de un círculo
# Recibe por parámetro el radio
def area_circulo(radio):
  area = math.pi * radio**2
  return area

# Llamada a la función anterior
area_circulo(5.3)
```

```python
# Función empleando type hint
def area_circulo(radio: float) -> float:
  area = math.pi * radio**2
  return area
```

El lenguaje Python

FIN