UCAM
UNIVERSIDAD CATÓLICA SAN ANTONIO

Escuela Universitaria Politécnica
Grado en Ingeniería Informática

UCAM
UNIVERSIDAD CATÓLICA DE MURCIA

f SéNeCa (+)
Agencia de Ciencia y Tecnología
Región de Murcia

Recursos de programación en Python

Olimpiadas Científicas

D. Juan Bonastre Egea

1. ¿Qué es programar? ¿Qué es Python?

Programar consiste en dar instrucciones a un ordenador para que realice una tarea de forma automática. Estas instrucciones se escriben en un **lenguaje de programación**.

Python es un lenguaje de programación:

- Muy fácil de leer y escribir
- Muy usado en ciencia, inteligencia artificial y educación
- Ideal para aprender a programar desde cero
- Podéis empezar a programar directamente buscando en internet "Online Python IDE"

Un programa en Python es simplemente una **secuencia de instrucciones** que se ejecutan de arriba a abajo.

```
print("Hola mundo")
```

2. Cómo se escribe en Python (sintaxis básica)

2.1. Python usa indentación (espacios)

En Python **los espacios importan**. Los bloques de código se indican con **sangría** (normalmente 4 espacios/ una tabulación).

Ejemplo correcto:

```
if 5 > 3:
    print("Es mayor")
    print("Esto está dentro del if")
```

Ejemplo incorrecto:

```
if 5 > 3:
print("Error")   # ❌ Falta indentación
```

2.2. Comentarios

Los comentarios sirven para explicar el código y **no se ejecutan**.

```
# Esto es un comentario
x = 5  # también aquí
```

3. Variables y tipos de datos

Una **variable** es una caja donde guardamos un valor.

```
edad = 17
nombre = "Ana"
altura = 1.65
```

Tipos básicos:

- int → números enteros (3, -5)
- float → números decimales (2.5, 1.0)
- str → texto ("Hola")
- bool → verdadero o falso (True, False)

Una variable debe ser de un tipo de dato en específico, pero Python lo detecta automáticamente.

4. Operaciones

4.1. Operaciones matemáticas

```
a = 10
b = 3

suma = a + b         # 13
resta = a - b        # 7
multiplicacion = a * b  # 30
division = a / b     # 3.333...
division_entera = a // b  # 3
resto = a % b        # 1
potencia = a ** b    # 1000
```

4.2. Operadores de comparación

Devuelven True o False.

```
a == b   # igual
a != b   # distinto
a > b
a < b
a >= b
a <= b
```

4.3. Operadores lógicos

```
and   # y
or    # o
not   # no
```

Ejemplo:

```
edad = 17
edad >= 16 and edad < 18   # True
```

5. Entrada y salida de datos

5.1. Mostrar información

```
print("Hola")
print(5)
print("Resultado:", 10 + 3)
```

5.2. Leer datos del usuario

```
nombre = input("Introduce tu nombre: ")
edad = int(input("Introduce tu edad: "))
```

⚠️ input() **siempre devuelve texto**, por eso a veces hay que convertir:

- int(...)
- float(...)

6. Condicionales (if)

Sirven para **tomar decisiones**. Si se cumple la condición expresada (que el resultado sea True) el código contenido será ejecutado.

Estructura básica:

```
if a==b:
    print("a y b son iguales")

if True:
    print("Siempre se imprimira este mensaje")

condicion = True
if condicion:
    print("Este mensaje se imprimira dependiendo del valor de
la variable 'condicion'")

if False:
    print("Nunca se imprimira este mensaje")
```

Con else:

```
if edad >= 18:
    print("Mayor de edad")
else:
    print("Menor de edad")
```

Con elif:

```
nota = 7

if nota >= 9:
    print("Sobresaliente")
elif nota >= 5:
    print("Aprobado")
else:
    print("Suspenso")
```

7. Bucles (repeticiones)

7.1. Bucle for

Se usa cuando sabemos cuántas veces repetir algo.

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

Salida:

```
0
1
2
3
4
```

Otro ejemplo:

```
for i in range(1, 6): # aquí especifica un range entre 1 y 6
    print(i)
```

- **for i in range(5)**: se podría traducir en “para cada i dentro del rango entre 0 y 5…”.
- Esto se traduce a que el valor de la i se va a incrementar de 1 en 1 empezando en 0 y terminando cuando el valor alcance o supere 5.
- Es por eso que esa única instrucción de “print(i)” se ejecuta 5 veces y cada vez con un valor distinto, porque la variable de i se está incrementando.

7.2. Bucle while

Se repite **mientras se cumpla una condición**.

```
contador = 0

while contador < 5:
    print(contador)
    contador += 1
```

- El while es más explicito que el for, el código contenido se ejecutará siempre que la condición sea True.
- Por eso es muy importante asegurarse de que la condición pueda alcanzar el estado False. Si no, entrearemos en un bucle infinito.

8. Funciones básicas en Python

8.1. ¿Qué es una función?

Una **función** es un bloque de código que:

- Tiene un **nombre**
- Realiza una tarea concreta
- Se puede **usar (llamar)** tantas veces como queramos

Sirve para:

- No repetir código
- Organizar mejor los programas
- Hacer el código más claro

Ejemplo sencillo:

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

print es una función que muestra información por pantalla.

8.2. Cómo usar una función (llamada a función)

Para usar una función:

1. Escribimos su nombre
2. Ponemos paréntesis ()
3. Dentro, si hace falta, ponemos datos (argumentos)

1. Las funciones suelen tener parámetros opcionales, esto quiere decir que si no ponemos nada tendrán un valor por defecto. Esto es el caso de “sep” y “end” en la función print.
2. Otros parámetros tienen nombre, tendremos que poner *nombre_parametro* como es el caso de “sep” o “end” de print (print(“Hola”, end=” “))

```
print("Hola mundo")
abs(-5)
```

8.3. Funciones que devuelven un valor

Algunas funciones **devuelven un resultado** que podemos guardar en una variable.

```
x = abs(-10)
print(x)   # 10
```

8.4. Funciones más importantes para empezar

print()

La función `print()` se usa para **mostrar texto, números y resultados** por pantalla. Es una de las funciones más importantes en Python.

```
print("Hola")
print(5)
print(3 + 2)
```

También se pueden mostrar **varias cosas a la vez**, separándolas por comas:

```
edad = 17
print("Edad:", edad)
```

Cuando usamos comas, Python:

- Añade espacios automáticamente
- Convierte los números a texto por nosotros

Concatenar

También podemos concatenar texto usando comas “,” o “+” en caso de que lo que queramos concatenar sea de tipo cadena (texto).

```
nombre = "Ana"
print("Hola " + nombre)

edad = 17

print("Tengo " + edad + " años")  # ❌ Error

print("Tengo", edad, "años")

print("Tengo " + str(edad) + " años")
```

Saltos de línea con \n

\n significa **salto de línea**.

```
print("Hola\nMundo")
```

Salida:

```
Hola
Mundo
```

También se puede usar para mostrar varias líneas en un solo print:

```
print("Nombre: Ana\nEdad: 17\nCurso: Bachiller")
```

Al igual que podemos imprimir salto de línea con \n, podemos imprimir una tabulación con \t.

Imprimir varias líneas

Opción 1: varios print

```
print("Línea 1")
print("Línea 2")
```

Opción 2: un solo print con \n

```
print("Línea 1\nLínea 2")
```

Cambiar el separador (sep)

Por defecto, print separa con espacios:

```
print(1, 2, 3)
```

Salida:

```
1 2 3
```

Podemos cambiarlo:

```
print(1, 2, 3, sep="-")
```

Salida:

```
1-2-3
```

Evitar salto de línea al final (end)

Por defecto, print termina con un salto de línea.

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

Salida:

```
Hola
Mundo
```

Si queremos que continúe en la misma línea:

```
print("Hola", end=" ")
print("Mundo")
```

Salida:

```
Hola Mundo
```

Imprimir resultados de cálculos

```
a = 5
b = 3
print("La suma es:", a + b)
```

input()

Lee un valor introducido por el usuario. Tiene como parámetro el texto a mostrar previo a la espera de datos.

```
nombre = input("Introduce tu nombre: ")

# Esto tendría el mísmo efecto, es otra forma de hacerlo
print("Introduce tu nombre: ", end="")
nombre = input()
```

int(), float(), str()

Sirven para convertir tipos de datos.

```
edad = int("18")
altura = float("1.75")
texto = str(123)
```

len()

Devuelve la longitud (número de elementos o caracteres).

```
len("Hola")    # 4
```

abs()

Devuelve el valor absoluto.

```
abs(-7)   # 7
```

round()

Redondea un número.

```
round(3.6)     # 4
round(3.14159, 2)  # 3.14
```

range()

Genera números para usar en bucles for.

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

max() y min()

Devuelven el mayor o el menor valor.

```
max(3, 7, 2)   # 7
min(3, 7, 2)   # 2
```

8.5. Crear funciones propias

También podemos **crear nuestras propias funciones**.

Sintaxis básica:

```
def nombre_funcion():
    instrucciones
```

Ejemplo:

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

Para usarla:

```
saludar()
```

Funciones con parámetros

Los **parámetros** son datos que recibe la función.

```
def saludar(nombre):
    print("Hola", nombre)

saludar("Ana")
saludar("Luis")
```

Funciones que devuelven valores (return)

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

resultado = suma(3, 4)
print(resultado)
```

Cuando se ejecuta return, la función termina.

8.7. Cuándo usar funciones en los ejercicios

Usa funciones cuando:

- El mismo cálculo se repite varias veces
- Quieres dividir un problema grande en partes pequeñas
- El enunciado pide explícitamente una función

9. Listas en Python

9.1. ¿Qué es una lista?

Una **lista** es una variable que puede guardar **varios valores a la vez**, en un cierto orden.

Ejemplo:

```
numeros = [3, 5, 7, 9]
nombres = ["Ana", "Luis", "María"]
```

Las listas pueden contener:

- Números
- Texto
- Mezclas de ambos

```
datos = [10, "Hola", 3.5]
```

9.2. Acceder a los elementos de una lista

Cada elemento tiene una **posición (índice)** que empieza en **0.**

```
numeros = [10, 20, 30]

print(numeros[0])  # 10
print(numeros[1])  # 20
print(numeros[2])  # 30
```

También se puede acceder desde el final:

```
print(numeros[-1])  # último elemento
```

9.3. Cambiar elementos de una lista

```
numeros = [1, 2, 3]
numeros[0] = 10
print(numeros)   # [10, 2, 3]
```

9.4. Longitud de una lista

```
numeros = [4, 8, 15]
print(len(numeros))  # 3
```

9.5. Recorrer listas con bucles

Con for (forma más habitual)

```
numeros = [1, 2, 3, 4]

for x in numeros:
    print(x)
```

- En este caso no usamos range() como antes.
- La variable x no se va a incrementar de uno en uno
- En este caso x va a ir cambiando su valor por cada uno de los elementos de la lista numeros

Con índices

```
numeros = [1, 2, 3, 4]

for i in range(len(numeros)):
    print(i, numeros[i])
```

- Ahora al usar range, sí que estamos incrementando el valor de la variable (ahora llamada i)
- Esta va a recorrer todas las posiciones de la lista, desde la 0 hasta la 3 (inclusive).

9.6. Operaciones básicas con listas

Sumar elementos

```
numeros = [1, 2, 3]
suma = 0

for x in numeros:
    suma += x

print(suma)
```

Buscar el mayor o el menor

```
numeros = [4, 9, 2]

mayor = numeros[0]
for x in numeros:
    if x > mayor:
        mayor = x

print(mayor)
```

9.7. Añadir y eliminar elementos

A continuación se mostrarán los métodos append(), remove() y pop(). Un método se puede entender como una función que esta contenida dentro de un objeto (una variable).

Añadir con append()

```
numeros = []
numeros.append(5)
numeros.append(10)
print(numeros)
```

- En este caso append(5) agrega 5 a la última posición de “numeros” debido a que es un método del propio objeto “numeros” (numeros.append(5)).
- Si pusiesemos “append(5)” solo saría error, lo cual tiene sentido porque en ningún lado le estamos diciendo a que lista agregar ese número 5.

Eliminar elementos

```
numeros = [10, 11, 12]

numeros.remove(11)  # elimina el elemento con valor 11
print(numeros)
```

O eliminar por posición:

```
numeros.pop(0)  # elimina el elemento en la posición 0
```

- Al igual que con append(), remove() y pop() eliminan los elementos de la lista numeros debido a que llamamos al metodo contenido dentro de esa misma variable (numeros).

9.8. Comprobar si un elemento está en la lista

```
numeros = [3, 6, 9]

if 6 in numeros:
    print("Está en la lista")
```

- Aquí la lo clave es “6 in numeros”, esa es la condición que devolverá True o False en caso de que el valor 6 esté dentro de la lista de numeros.

9.9. Errores típicos con listas

Acceder a una posición que no existe (IndexError)

```
numeros = [1, 2, 3]
print(numeros[3])  # ❌
✔️ Solución:
print(numeros[2])
```

Confundir índice con valor

```
for i in numeros:
    print(numeros[i])  # ❌
✔️ Correcto:
for i in numeros:
    print(i)
```

Olvidar que los índices empiezan en 0

```
print(numeros[1])  # no es el primer elemento
```

- Recordad que en programación, las listas siempre empiezan en la posición 0, no la 1.

9.10. Listas y funciones

Una lista se puede **pasar a una función**:

```
def suma_lista(lista):
    suma = 0
    for x in lista:
        suma += x
    return suma

print(suma_lista([1, 2, 3]))
```

10. Errores típicos y cómo solucionarlos

Error de sintaxis (SyntaxError)

Ejemplo:

```
if 5 > 3
    print("Hola")
```

❌ Falta “:”
✔️ Solución:

```
if 5 > 3:
    print("Hola")
```

Error de indentación (IndentationError)

```
if x > 0:
print(x)
```

❌ Falta sangría
✔️ Solución:

```
if x > 0:
    print(x)
```

Error de tipo (TypeError)

```
edad = "17"
print(edad + 1)
```

❌ No se puede sumar texto y número
✔️ Solución:

```
edad = int("17")
print(edad + 1)
```

Variable no definida (NameError)

```
print(x)
```

❌ x no existe
✔️ Solución:

```
x = 5
print(x)
```

Bucle infinito

```
i = 0
while i < 5:
    print(i)
```

❌ i nunca cambia
✔️ Solución:

```
i = 0
while i < 5:
    print(i)
    i += 1
```

Olvidar los paréntesis al llamar a una función

```
print
```

❌ No ejecuta la función
✔️ Correcto:

```
print()
```

Llamar a una función antes de definirla

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

❌ Error
✔️ Solución: definir la función **antes** de usarla.

Confundir print con return

```
def suma(a, b):
    print(a + b)
x = suma(2, 3)
```

❌ x vale None
✔️ Solución:

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

Número incorrecto de parámetros

```
def saludar(nombre):
    print("Hola", nombre)
saludar()
```

❌ Falta argumento
✔️ Solución:

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

No usar el valor devuelto

```
def doble(x):
    return 2 * x

doble(5)
```

❌ El resultado se pierde
✔️ Solución:

```
resultado = doble(5)
print(resultado)
```

Acceder a una posición que no existe (IndexError)

```
numeros = [1, 2, 3]
print(numeros[3])
```

❌ Fuera de rango
✔️ Solución:

```
print(numeros[2])
```

Olvidar que los índices empiezan en 0

```
print(numeros[1])  # no es el primer elemento
```

✔️ El primer elemento es numeros[0]

Confundir índice con valor en un bucle

```
for i in numeros:
    print(numeros[i])
```

❌ i ya es el valor
✔️ Solución:

```
for i in numeros:
    print(i)
```

Modificar una lista mientras se recorre

```
for x in numeros:
    numeros.remove(x)
```

❌ Comportamiento incorrecto
✔️ Solución:

- Recorrer una copia
- O usar índices
- O crear una lista nueva

Usar una lista vacía sin comprobar

```
numeros = []
mayor = numeros[0]
```

❌ Error
✔️ Solución:

```
if len(numeros) > 0:
    mayor = numeros[0]
```

Bucle infinito con while

```
i = 0
while i < len(numeros):
    print(numeros[i])
```

❌ Falta incrementar i
✔️ Solución:

```
i = 0
while i < len(numeros):
    print(numeros[i])
    i += 1
```

Usar mal range

```
for i in range(numeros):
```

❌ range necesita un número
✔️ Solución:

```
for i in range(len(numeros)):
```

12. Ejercicio guiado: Análisis de números

12.1. Enunciado

Vamos a crear un programa en Python que trabaje con una lista de números introducidos por el usuario.

El programa debe hacer lo siguiente:

1. Pedir al usuario cuántos números va a introducir.
2. Pedir esos números uno a uno y guardarlos en una lista.
3. Mostrar la lista completa.
4. Calcular y mostrar:
    - La suma de todos los números
    - El número mayor
    - El número menor
5. Indicar cuántos números son mayores o iguales que 10.

12.2. Pistas y pasos guiados

Paso 1: Pedir la cantidad de números

Recuerda que input() devuelve texto y hay que convertirlo a entero.

Pista:

```
n = int(input("¿Cuántos números vas a introducir? "))
```

Paso 2: Crear una lista vacía

```
numeros = []
```

Paso 3: Leer los números y guardarlos en la lista

Pista: usa un bucle for y append().

```
for i in range(n):
    x = int(input("Introduce un número: "))
    numeros.append(x)
```

Paso 4: Mostrar la lista

Usa print() para mostrar la lista completa.

Pista:

```
print("Lista de números:", numeros)
```

Tambień se puede imprimir de usando un bucle.

```
for i in range(n):
    print(numeros[i])
```

Paso 5: Calcular la suma

Pista: inicializa una variable y usa un bucle.

```
suma = 0
for x in numeros:
    suma += x
```

También se puede hacer usando la función sum().

```
suma = sum(numeros)
```

Paso 6: Encontrar el mayor y el menor

Pista: empieza suponiendo que el primero es el mayor y el menor.

```
mayor = numeros[0]
menor = numeros[0]
```

Luego recorre la lista y compara.

```
for i in range(n):
    if numeros[i] > mayor:
        mayor = numeros[i]
    if numeros[i] < menor:
        menor = numeros[i]
```

También se puede hacer usando las funciones max() y min()

```
mayor = max(numeros)
menor = min(numeros)
```

Paso 7: Contar números mayores o iguales que 10

Pista:

```
contador = 0
for x in numeros:
    if x >= 10:
        contador += 1
```

12.3. Salida esperada (ejemplo)

Si el usuario introduce:

```
¿Cuántos números vas a introducir? 5
2
10
7
15
4
```

El programa podría mostrar:

```
Lista de números: [2, 10, 7, 15, 4]
Suma: 38
Mayor: 15
Menor: 2
Números mayores o iguales que 10: 2
```