

**FUNDAMENTAL: REPASAR LOS EJERCICIOS HECHOS EN CLASE EN EL PRIMER TRIMESTRE;  
SOBRE TODO: Ejercicio notas, Clasificación por edades, Ejercicios de repaso (sobre todo el 12)**

## RESUMEN DE PROGRAMACIÓN EN PYTHON

Mostrar un mensaje por pantalla:

```
print(f"Este es el número: {numero}")  
print("Este es el número: " + str(numero))
```

Pedir datos al usuario:

```
resp = input("Dime tu nombre: ")  
print("Hola, usuario llamado...")  
print(resp)
```

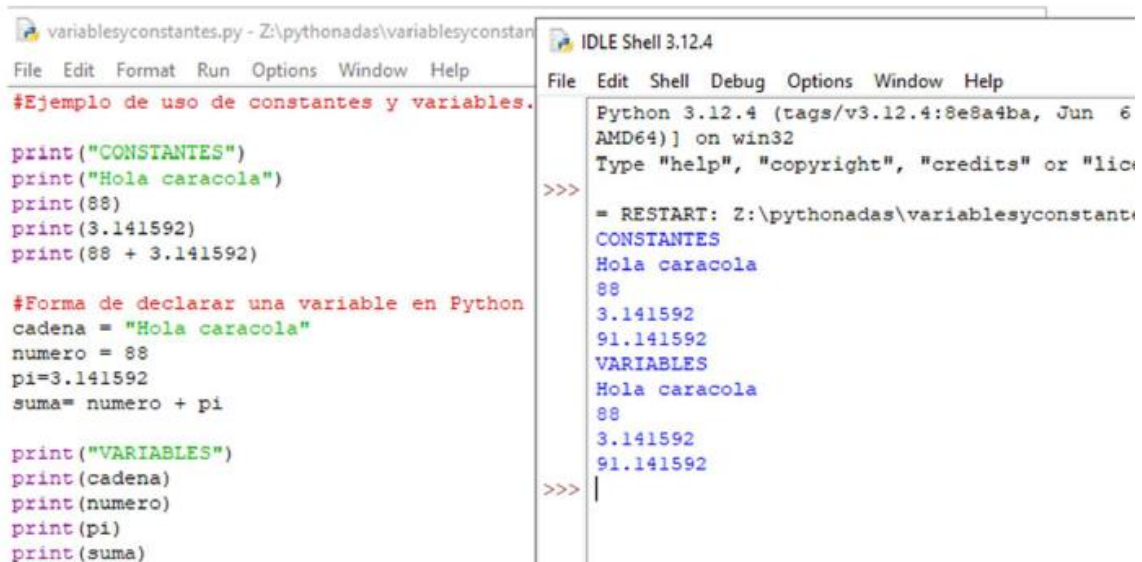
Constantes y variables:

Una constante es un valor que nunca cambia; ejemplo: cuando usamos un número directamente en el programa estamos usando una constante.

Normas para los nombres de variables y constantes:

- 1) No espacios.
- 2) Solo se admiten letras, números y el guion bajo “\_”.
- 3) No se puede comenzar el nombre de la variable con un número, sí con un guion bajo pero tiene un significado especial.
- 4) No usar palabras reservadas por el lenguaje de programación.

Ejemplo de uso de variables y constantes:



The screenshot shows a Python IDE with two windows. The left window, titled 'variablesyconstantes.py', contains the following code:

```
#Ejemplo de uso de constantes y variables.  
  
print("CONSTANTES")  
print("Hola caracola")  
print(88)  
print(3.141592)  
print(88 + 3.141592)  
  
#Forma de declarar una variable en Python  
cadena = "Hola caracola"  
numero = 88  
pi=3.141592  
suma= numero + pi  
  
print("VARIABLES")  
print(cadena)  
print(numero)  
print(pi)  
print(suma)
```

The right window, titled 'IDLE Shell 3.12.4', shows the output of the script after execution:

```
Python 3.12.4 (tags/v3.12.4:8e8a4ba, Jun 6  
AMD64) on win32  
Type "help", "copyright", "credits" or "lic  
>>> = RESTART: Z:\pythonadas\variablesyconstant  
CONSTANTES  
Hola caracola  
88  
3.141592  
91.141592  
VARIABLES  
Hola caracola  
88  
3.141592  
91.141592  
>>> |
```

### Operadores aritméticos

Para hacer operaciones de tipo aritmético (sumas, restas, divisiones,...) se necesitan los operandos y el símbolo que indica la operación que haremos; estos símbolos son los operadores aritméticos y en Python son los siguientes:

Operador	Significado	Ejemplo	Resultado
+	Suma	2 + 2	4
-	Resta	4 - 1	3
*	Multiplicación	3 * 5	15
/	División	13 / 8	1.625
//	División entera	13 // 8	1
%	Módulo o resto	13 % 8	5
**	Exponente	2 ** 4	16

### Operadores de comparación

Otra categoría de operadores que se usan muy a menudo son los operadores de comparación; en general sirven para comparar dos operandos (uno a cada lado del operador) y el resultado es siempre True o False (Verdadero o Falso).

Operador	Significado	Ejemplo	Resultado
>	Mayor que	100 > 10	True
<	Menor que	100 < 10	False
>=	Mayor o igual que	2 <= 2	True
<=	Menor o igual que	1 <= 4	True
==	Equal	6 == 9	False
!=	Distinto	3 != 2	True

### Operadores lógicos

Otra categoría de operadores que usan como operandos expresiones, variables o constantes que evalúan a True o False. El resultado será siempre True o False.

Operador	Significado	Ejemplo	Resultado
and	Y	True and True	True
or	O	True or False	True
not	no	Not True	False

## 2. ESTRUCTURAS DE CONTROL

### 2.1 Sentencias/Instrucciones Condicionales (Conditional statements. Statement = sentencia, instrucción)

Atención a los dos puntos después de la condición del if y después del else.

Ej.:

Pseudocódigo	Código
Si (X es divisible por 2) entonces mostrar (X es un número par) Si no mostrar (X es un número impar) Fin Si	if x % 2 == 0: print("El número es par") else: print("El número es impar")

Ejemplo: Cómo saber si un número X es par:

```
if x % 2 == 0:  
    print("El número es par")  
else:  
    print("El número es impar")
```

Suelen usarse if-else anidados: unos dentro de otros.

Ejemplo: clasificar a personas según su edad: preguntamos la edad y guardamos la respuesta en la variable edad; convertimos la respuesta del usuario en un número entero:

```
edad = int(input("Dame tu edad: "))
```

```
if edad <= 18:
    print("Eres menor de edad")
else:
    if edad == 18:
        print("Ya eres mayor de edad, aunque aún muy joven")
    else:
        if edad < 30:
            print("Eres un jovenazo")
        else:
            if edad >= 65:
                print("Estás jubilado")
            else:
                print("Tienes entre 30 y 64 años: te toca currar")
```

Otra forma de hacer lo anterior es mediante sentencias **elif** que son la suma de else con un if:

```
if edad < 18:
    print("Eres menor de edad")
elif edad == 18:
    print("Ya eres mayor de edad, aunque aún muy joven")
elif edad < 30:
    print("Eres un jovenazo")
elif edad >= 65:
    print("Estás jubilado")
else:
    print("Tienes entre 30 y 64 años: te toca currar")
```

En ocasiones tenemos que elegir una sola opción de entre muchas posibilidades; en este caso se puede usar una serie de if-else, como en el ejemplo: pedimos un número entre [1-7] al usuario e imprimimos el día de la semana correspondiente a ese número. El pseudocódigo sería:

Versión 1

```
Pedir número
si número == 1 entonces
    imprimir "lunes"
si número == 2 entonces
    imprimir "martes"
si número == 3 entonces
    imprimir "miércoles"
si número == 4 entonces
    imprimir "jueves"
si número == 5 entonces
    imprimir "viernes"
si número == 6 entonces
    imprimir "sábado"
si número == 7 entonces
    imprimir "domingo"
si día < 1 OR día > 7
    imprimir "Error"
```

Versión 2

```
Pedir número
si número == 1 entonces
    imprimir "lunes"
si no
    si número == 2 entonces
        imprimir "martes"
    si no
        si número == 3 entonces
            imprimir "miércoles"
        si no
            si número == 4 entonces
                imprimir "jueves"
            si no
                si número == 5 entonces
                    imprimir "viernes"
                si no
                    si número == 6 entonces
                        imprimir "sábado"
                    si no
                        si número == 7 entonces
                            imprimir "domingo"
                        si no
                            imprimir "Error"
```

## 2.2 Sentencias/Instrucciones Repetitivas o bucles (Loop statements).

Un bucle es una estructura de programación que sirve para repetir un conjunto de líneas (o sentencias) varias veces, mientras se cumpla una determinada condición.

En Python hay dos tipos de bucles: bucles for y bucles while.

### Bucles for:

En los bucles for sabemos desde el principio cuántas veces se va a repetir (sabemos cuántas iteraciones hará).

Hay dos formas de hacer bucles for:

1) Bucles for que recorren un conjunto de valores que se indica expresamente.

Tienen la forma siguiente:

```
for variable in conjunto_de_valores:
    líneas del interior del bucle
```

Ejemplo:

---

```
#Bucle for que recorre los días de la semana.
#Los días de la semana están definidos como un conjunto.

for dia in ["Lunes", "Martes", "Miércoles", "Jueves", "Viernes", "Sábado", "Domingo"]:
    print(dia)
```

Podemos definir el conjunto de valores creando, previamente, lo que se conoce como una lista; un par de ejemplos:

Definimos los siguientes conjuntos de valores:

```
Star_wars = ["Han Solo", "Luke Skywalker", "Chewbacca", "R2D2", "C3PO", "Darth Vader"]

Celtics = ["Larry Bird", "Kevin McHale", "Robert Parish", "Dennis Johnson", "Danny Ainge", "Cedric Maxwell", "Scott Wedman", "Ray Williams", "Greg Kite", "M.L. Carr", "Rick Carlisle", "Quinn Buckner"]

#Otro ejemplo de bucle for donde se definen previamente los valores mediante listas

Star_wars = ["Han Solo", "Luke Skywalker", "Chewbacca", "R2D2", "C3PO", "Darth Vader", "Maestro Yoda"]
Celtics = ["Larry Bird", "Kevin McHale", "Robert Parish", "Dennis Johnson", "Danny Ainge", "Cedric Maxwell"]

print("---- Personajes de Star Wars: ----")
for personaje in Star_wars:
    print(personaje)

print("$$$ Plantilla Boston Celtics 1984/85: $$$")
for personaje in Star_wars:
    print(personaje)
```

2) Bucles for que recorren un rango de valores, de los cuales se indican los valores inicial y final (y en ocasiones también el incremento).

Tienen la forma siguiente:

```
for variable in rango(valor_inicial,valor_final, incremento):
    líneas del interior del bucle
```

Donde pueden faltar algunos de elementos, lo que da lugar a tres posibilidades:

- a) for range(max): se recorren los valores desde 0 hasta *max*-1
- b) for range(min, max): se recorren los valores consecutivos desde *min* hasta *max*-1
- c) for range(min, max, step): se recorren los valores desde *min* hasta *max*-1 con incrementos de *step* en *step*

Hay que tener en cuenta la siguiente peculiaridad: la variable recorrerá los valores en el rango desde *valor\_inicial* hasta *valor\_final* - 1.

Ejemplo del tipo a:

```
#Ejemplo de bucle for que recorre un rango de valores: desde x=0 hasta x=5-1.  
#En los apuntes se ha llamado tipo a:
```

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

Resultado de la ejecución:

```
===== RESTART: G:\Curso24-25\INS\Python\pythonadas\for_rango_a.py =  
0  
1  
2  
3  
4
```

Ejemplo del tipo b:

---

```
#Ejemplo de bucle for que recorre un rango de valores: desde x=1 hasta x=10-1.  
#En los apuntes se ha llamado tipo b:
```

```
for x in range(1,10):  
    print(x)
```

Y lo que veríamos por pantalla

```
===== RESTART:  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

Ejemplo del tipo c:

---

```
#Ejemplo de bucle for que recorre un rango de valores: desde x=0 hasta x=11-1 con incrementos de 2 en 2.  
#En los apuntes se ha llamado tipo c:
```

```
for x in range(0,11,2):  
    print(x)
```

Al ejecutarlo obtenemos esto:

```
===== RESTART: G:/Curso24-25/INS/Python/pythonadas/for_rango_c.py  
0  
2  
4  
6  
8  
10
```



Bucles for anidados:

Es posible que entre los comandos que queramos ejecutar dentro de un bucle esté otro bucle; en este caso hay que tener en cuenta que para cada iteración del bucle externo se harán todas las iteraciones del bucle interno.

Con los bucles anidados recorreremos estructuras en dos niveles; ejemplo: filas y columnas.

Ejemplo:

**#Ejemplo de bucles for anidados: para cada elemento de los que recorre el bucle exterior se recorren todos los elementos del bucle interior.**

```
for i in range(1, 4): # Bucle para 1, 2, 3
    for j in range(1, 4): # Bucle para 1, 2, 3
        print(i, j)
```

Y el resultado sería:

```
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```

Otro ejemplo:

**#Ejemplo de bucles for anidados: para cada elemento de los que recorre el bucle exterior se recorren todos los elementos del bucle interior.**  
**#Suponemos un hotel con 5 plantas y cuatro habitaciones (A, B, C, D) en cada planta**

```
habitaciones = ["A", "B", "C", "D"]
for i in range(1,6):
    print(f"Piso: {i}")
    for j in habitaciones:
        print(f"Habitación: {j}", end="\t") # Mostrar la suma de índices
    print() # Salto de línea después de cada fila
```

Al ejecutarlo:

```
Piso: 1
Habitación: A Habitación: B Habitación: C Habitación: D
Piso: 2
Habitación: A Habitación: B Habitación: C Habitación: D
Piso: 3
Habitación: A Habitación: B Habitación: C Habitación: D
Piso: 4
Habitación: A Habitación: B Habitación: C Habitación: D
Piso: 5
Habitación: A Habitación: B Habitación: C Habitación: D
```

**Bucles while:**

En los bucles while no sabemos desde el principio cuántas veces se va a repetir: que se sigan ejecutando o no dependerá de una variable, que es la que controla el bucle, y que tiene que:

- 1) Inicializarse fuera del bucle.
- 2) Modificarse dentro del bucle.

Puede no llegar a ejecutarse ninguna vez o puede ejecutarse un número variable de veces (dependiendo del valor que la variable tome dentro del bucle) o puede ejecutarse indefinidamente (bucle infinito: se seguirá ejecutando hasta que, desde fuera del programa se cierre el proceso).

La forma genérica de este bucle es la siguiente:

while [expresión]: código a ejecutar.

Ejemplo:

```
#Bucle While que imprime los números del 1 a 10
x = 1
while x <= 10:
    print(f"{x}")
    x += 1

print("Hasta aquí la cuenta!")
```

El resultado de ejecutar el código anterior es el siguiente:

```
1
2
3
4
5
6
7
8
9
10
Hasta aquí la cuenta!
```

Otra forma de terminar un bucle de forma inmediata, sin esperar a que se evalúe la condición, es usando la palabra reservada **break** dentro del bucle; en cuanto se llegue a ella se dejará de ejecutar el bucle.

Ejemplo:

```
#Ejemplo de bucle que deja de ejecutar al encontrar un break
for i in range (1,100):
    print(i)
    break
```

Al ejecutar este bucle el resultado es el siguiente:

```
===== RESTART: C:/Python/Clase/ejercicioswhile/while_4break.py =====
1
```

Lo normal es que el **break** esté asociado a un **if** y que se ejecute solo en circunstancias especiales que tenemos que controlar nosotros.

Ejemplo:

El programa sigue preguntando hasta que respondamos "q":

```
pregunta = "¿Y tú de quién eres?"

while True:
    resp = input(pregunta)
    if resp == "q":
        break
    print("Así que eres de " + resp + " mira tú qué bien")

print("Ea, se acabó de preguntar, señora!")
exit()
```

### 3. FUNCIONES

Un principio fundamental a la hora de hacer programas es intentar que éstos no sean enormes cadenas de instrucciones sin una estructura. Es preferible dividir y encapsular las líneas de código en unos subprogramas llamados funciones: esto es un principio fundamental de la programación estructurada.

Una función consiste en una serie de líneas de código con un nombre que se definen en un programa y que posteriormente puede llamarse desde cualquier parte del programa.

#### Cómo se define una función en Python:

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

La función anterior se llama mi\_funcion, se le pasa un valor y en su interior la función calcula el doble del valor que se le pasa y lo devuelve usando return.

Ojo: en el nombre de una función no se ponen espacios en blanco ni tildes, paréntesis,...

**#Ejemplo de una función que recibe un número y lo multiplica por dos.**

**#Definición de funciones:**

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

```
num = int(input("Dame un número: "))  
doble_num = duplicadora(num)  
print(f"El doble de {num} es: {doble_num}")
```

```
Dame un número: 67  
El doble de 67 es: 134
```

Una función puede tener un parámetro, varios o ninguno:

**#Ejemplo de funciones que reciben distinto número de parámetros.**

**#Definición de funciones:**

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

```
def divisor(x,y):  
    if y != 0:  
        return x/y  
    else: #No se puede dividir por 0  
        return -1
```

```
def muestra_info():  
    print("Este es un programa que sirve para probar funciones en Python")  
    print("Esperamos que les guste!")
```

```
muestra_info()  
num = int(input("Dame un número: "))  
doble_num = duplicadora(num)  
print(f"El doble de {num} es: {doble_num}")  
numerador = int(input("Dame un numerador (el de arriba): "))  
denominador = int(input("Dame un denominador (el de abajo): "))  
resultado = divisor(numerador,denominador)  
  
if resultado == -1:  
    print("No se pudo hacer la división porque el denominador no puede ser 0")  
else:  
    print(f"{numerador} / {denominador} = {resultado}")
```