```
M3:C5 Full Stack Programme
Alumno: Iñigo Uribarri
'''
------------------------------------------------------------------------
------------------------
** 1_CONDICIONALES**
Los condicionales son estructuras de control de flujo que permiten tomar
decisiones basadas en condiciones.
Un condicional es una declaración que evalúa si la expresión utilizada es
verdadera o falsa (True o False).
Usando dichas condiciones, es posible hacer que el código tome diferentes
decisiones, como si de una estructura en árbol se tratara.
Basándose en el resultado de la condición es posible, ejecutar un bloque
de código u otro, o incluso no hacer nada.

***¿Para qué sirven exactamente los condicionales? ***
Los condicionales se utilizan para:
- Tomar decisiones en el código, ejecutando diferentes partes en función
del resultado del condicional
- Manejar diferentes casos y condiciones de antemano

*** Como se expresan los condicionales en Python? ***
El condicional "if" se utiliza para comprobar la veracidad o falsedad de
una expresión
El condicional "elif" se utiliza para comprobar la veracidad o falsedad
de una expresión, después de haber comprobado previamente otra expresión
con if
El condicional "else" se utiliza para ejecutar el bloque de código si
ninguna de las anteriores expresiones era verdadera.

Es posible anidar los condicionales explicados previamente para formar
arboles de decisión complejos.

A continuación, se explica brevemente la estructura de los condicionales
en el lenguaje de programación Python:
'''
edad = 64
if edad>=65 :
    print ("Estás en edad de jubilarte!") # Parte del código a ejecutar
si la edad a comprobar es mayor que 65, y por lo tanto la condición es
True
elif edad>18:
    print("Es usted mayor de edad!") # Parte del código a ejecutar si la
edad a comprobar es mayor que 18 (pero menor que 65), y por lo tanto la
condición es True
else:
    print("Aún no eres mayor de edad!")#Parte del código a ejecutar si
las anteriores dos condiciones son False, y por lo tanto la edad es menor
que 18
```

```
'''
*** ¿Qué son los comparadores lógicos? ***
Los comparadores lógicos en Python son operadores para crear expresiones
complejas que comprueben expresiones en condicionales.
En el anterior ejemplo se han utilizado dichos comparadores para
establecer relaciones entre las condiciones a comprobar-
Los operadores lógicos más utilizados son:
- `==` Igual a
- `!=` Diferente a
- `<` Menor que
- `>`Mayor que
- `<=`Menor o igual que
- `>=`Mayor o igual que
A continuación, se elabora un ejemplo usando algunos de los
condicionales.
En el ejemplo se usan condicionales para conocer el signo de un número
entero y si es par o impar
'''
numero= 68
if numero>0: #Uso del operador mayor que
    if numero %2 ==0: #Uso del operador lógico igual a
        print("El número es positivo y par")
    else:
        print("El número es positivo e impar")
elif numero<0: #Uso del operador menor que
    if numero %2 !=0: #Uso del operador lógico diferente a
        print("El número es negativo e impar")
    else:
        print("El número es negativo y par")
else:
    print ("Tu número es CERO!")

'''
------------------------------------------------------------------------
------------------------
** 2_TIPOS DE BUCLE EN PYTHON**
Python dispone de dos tipos principales de bucles:
- Bucle "for" : Itera sobre una secuencia, por ejemplo una lista, un
diccionario o un string
- Bucle "while" : Ejecuta un bloque de código mientras una condición dada
siga siendo verdadera

¿Por qué son útiles los bucles?
Los bucles son útiles puesto que:
- Permiten automatizar tareas repetitivas
- Pueden ayudar a reducir el código necesario para ejecutar tareas
- Aumentan la eficiencia del código y permiten entenderlo de un vistazo
```

```
A continuación, se presentan ejemplos de los principales tipos de bucles (for y while) en Python:
'''
#Bucle for aplicado en una lista, de forma que itera sobre los elementos de la lsita
lista_compra= ["lechuga","tomate","queso","leche"]
for item in lista_compra: # Este bucle for itera sobre cada uno de los elementos en la lista "lista_compra"
    print (item) # por cada uno de los elementos de la lista, imprime el elemento

#Bucle while aplicado a un contador, de forma que ejecuta el código mientras la condición a cumplir siga siendo verdadera
contador=0
while contador <= 5: #Establece una condición. Mientras el valor de contador sea igual o menor que 5, ejecuta el código.
    print(contador) #Código a ejecutar mientras la condición previa sea verdadera.
    contador += 1 # Sumar uno al número asignado a la variable contador, y vuelve a ejecutar el código.
'''
En el caso de los bucles while, es necesario explicar el uso de control de bucles para evitar bucles ejecutados de forma infinita.
En Python existen dos instrucciones principales para el control de bucles:
- `break` Fuerza el bucle a interrumpirse antes de que terminara de ejecutarse
- `continue` Fuerza el bucle a saltar a la siguiente iteración
A continuación, se presentan dos ejemplos de los usos de control de bucles mediante `break`y `continue`:
'''
#Ejemplo continue:
for num in range(11):#Itera sobre todos los números en un rango de 11, es decir, del 0 al 10
    if num%2==0 :# Establecemos la condición. En este caso para los números pares
        continue # Ejecuta el código en caso de que el anterior condicional sea verdadero. En este caso, salta a la siguiente iteración de la lista
    print(num)

# Ejemplo break:
numero_secreto= 4
while True: #Iterar sobre el código de manera infinita
    #Pedir al usuario que adivine el número secreto y que introduzca el que crea que es.
    adivinanza = int(input("Adivinas el número secreto del 1 al 5? Introduce aquí tu número: "))
    #Verificar si el número introducido coincide con el número secreo
```

```
    if adivinanza== numero_secreto: #Comprobación de la condición, es
decir, si el número introducido coincide con el número secreto
        print ("Felicidades! Has adivinado el número secreto")
        break #Si la condición resulta verdadera, interrumpir el bucle
mediante break
    else: #Bloque de código a ejecutar si ninguna de las anteriores
condiciones es verdadera
        print("Oh! lo siento no has acertado el número") #Siguiente
iteración del código
'''
------------------------------------------------------------------------
------------------------
** 3_LISTAS POR COMPRENSIÓN EN PYTHON**
Una lista por compresión en Python es una forma breve y eficiente de
crear listas.
Permite generar listas nuevas aplicando una expresión a cada elemento de
una lista, rango o tupla.

¿Por qué son útiles las listas por comprensión?
Las listas por comprensión permiten escribir código de una manera más
concisa en comparación con los métodos tradicionales usando bucles.
De esta forma, se reduce la cantidad de líneas necesarias para realizar
un mismo objetivo.
Sin embargo, debido a que pueden llegar a no ser entendibles a simple
vista, su uso debe estar limitado a entornos que estén familiarizados con
dichos métodos.

A continuación, se presentan ejemplos de la sintaxis de las listas por
compresión:
["expresión" for "elemento" in "secuencia" if "condición]
Un ejemplo práctico del mismo sería:
La creación de una lista de los cubos de los números pares del 0 al 9
'''
cubos_pares= [x**3 for x in range(10) if x%2==0]
# Siendo x el número, la lista por compresión lo eleva al cubo
# dicho x ha de estar contemplado entre los números en range(10), es
decir del 0 al 9
#Por último, dicha operación solamente se aplicará a aquellos números
cuyo módulo 2 sea 0. Es decir, a los números pares.
print(cubos_pares)
'''
Otro ejemplo de la creación de listas por comprensión:
La creación de una lista a partir de las letras que componen una palabra
escritas en mayuscula
'''
palabra= "programación"
mayusculas= [letra.upper() for letra in palabra]
print(mayusculas)
```

```
'''
------------------------------------------------------------------------
-------------------------
** 4_ARGUMENTOS EN PYTHON**
Los argumentos, son inputs que se usan en una función cuando esta es
llamada.
De esta forma, los argumentos permiten que las funciones acepten
información y realicen operaciones basadas en dicha información.
Estos, se definen como parte de la declaración de la función y se
proporcionan en la llamada de la función.

A continuación, se proporcionan varios ejemplos de los argumentos en
python aplicados a funciones
'''
def llamar_cuadrados (numero, nombre): # la función toma dos argumentos.
En este caso un número y un string
    cuadrado= numero**2 # El primero de los argumentos, es decir el
número, lo eleva al cudrado
    print (f"El número elegido por {nombre} es {numero} y su cuadrado es
{cuadrado}") #por último mediante imprime el número elegido, por la
persona y su cuadrado
llamar_cuadrados(5, "Iñigo")
'''
Sin embargo, es posible no acotar el número de argumentos a aceptar por
una función, abriendo la posibilidad de aceptar múltiples argumentos.
Esto puede ser muy útil cuando no se sabe que cantidad de argumentos va
aceptar la función.
Para poder utilizar este tipo de argumentos en Python, se utilizan los
siguientes valores:
- `*args` Recibe una tupla de argumentos
- `**kwargs` Recibe un diccionario de argumentos
A continuación, se presentan dos ejemplos demostrando el uso de *args y
*kwargs
'''
def imprimir_palabras(*args):#mediante *args, se deja abierto el número
de argumentos a procesar por la función
    for palabra in args:
        print(palabra)
imprimir_palabras("Hola","Mundo","Estos","Son","Valores","Múltiples")

nombre1="Xabier"
edad1=30
ciudad1="Bilbao"

def identificar_persona (**kwargs):#mediante *kwargs, también se deja
abierto el número de argumentos a procesar por la función
    for clave, valor in kwargs.items():# por cada uno de los argumentos
en *kwargs
        print(f"{clave}: {valor}") #imprime el argumento y su valor
```

```
identificar_persona(nombre=nombre1,edad=edad1,ciudad=ciudad1)
'''
Por último, también es posible utilizar argumentos por defecto.
De esta forma es posible asignar un valor por defecto a un argumento, de
forma que, si no se define, se ejecutará el valor por defecto.
A continuación, se ilustra mediante un ejemplo:
'''
actualidad= 2024
def calcular_edad (nacimiento,nombre="Usted",):# En este caso, si no se
inserta el argumento edad, se usara el argumento "Usted" por defecto
    edad_calculada= int(actualidad)-int(nacimiento)#Calcular la edad
restando el año actual al año de nacimiento
    print(f"{nombre} nació el año {nacimiento} por lo que tiene
{edad_calculada} años" )
calcular_edad(1992)
'''
Hay que tener en cuenta que es posible mezclar los tipos de argumento
utilizados en esta sección
Ello abre un potencial enorme a la hora de configurar funciones
'''
'''
------------------------------------------------------------------------
-------------------------
** 5_FUNCIONES LAMBDA EN PYTHON **
Las funciones Lambda en Python, pueden ser funciones anónimas, es decir
funciones sin un nombre asignado en una variable
Esto permite crear pequeñas funciones de una sola línea de manera rápida
y eficaz

Dichas funciones siguen la siguiente estructura
lambda argumentos: expresión
A continuación, se proveen ejemplos del uso de la función lambda
'''
#Este ejemplo devuelve el cubo de cualquier número utilizado como
argumento en la función lambda
cubos_pares_lambda= lambda x: x**3
print(cubos_pares_lambda(2))
# Este otro ejemplo ordena las palabras en una lsita según su longitud en
una nueva lista
words=
["pájaro","ave","chiquilicuatre","petirrojo","desoxirribonucleico"]
words_ordenadas= sorted(words, key=lambda palabra: len(palabra))
#la función lambda palabra, toma un argumento palabra, y mediante la
función len.palabra, retorna su longitud
#la función sorted ordena la palabra en la lista "words", dependiendo de
su longitud
print(words_ordenadas)
'''
```

```
Estos ejemplos muestran como las funciones lambda anónimas pueden ser
útiles cuando se quieren realizar funciones rápidas de una sola linea
'''
'''
------------------------------------------------------------------------
-------------------------
** 6_PAQUETES PIP EN PYTHON **
En Python, un paquete pip es una colección de módulos y archivos
generados por otros desarrolladores.
El uso de estos paquetes permite aprovechar el código existente para no
tener que escribir código desde cero.
Además, mediante el uso de los paquetes pip, es posible homogeneizar y
estandarizar el código aplicado a cuestiones compartidas por multitud de
desarrolladores.
Estos paquetes están compuestos por todo tipo de recursos, imágenes,
archivos de datos, código...

¿Como se utiliza pip para instalar paquetes?
Para instalar paquetes se ha de ejecutar el siguiente comando en la
terminal
    pip install "introduzca aquí el nombre del paquete deseado"

Los paquetes pip son similares al node package manager en el lenguaje
Javascript.
En conclusión, estos permiten aprovechar al máximo la amplia gama de
recursos disponibles en la comunidad de Python
'''
```