PRÁCTICA 5: Funciones en Python

1.- Funciones

Se puede ver una función como un conjunto de instrucciones que agrupamos y a las que damos un nombre para poder llamarla (ejecutar esas instrucciones) todas las veces que queramos, desde cualquier otro punto de nuestro programa, caracterizando esa ejecución mediante la utilización de parámetros.

Para definir una función se escribe primero la ***cabecera***, con el nombre de la función, y entre paréntesis los parámetros que recibe. Inmediatamente a continuación ponemos las instrucciones y finalmente el valor final resultante con un `return`:

```
def NombreFuncion(param1, param2...):
    // instrucciones
    instrucción 1
    instrucción 2
    ...
    return resultado
```

Por ejemplo:

```
def Potencia(base: float, exponente: int)-> float:

    resultado = 1

    for i in range(1, exponente + 1)
        resultado *= base

    return resultado
```

Los parámetros podemos considerarlos como un tipo especial de variables locales a la función. Lo que tienen de especial es que los valores de los parámetros se inician en el momento en que se llama (se *invoca*) a la función.

Nota: *Si necesitamos que la función devuelva más valores, podemos usar "return" para devolver más de un valor, pero eso lo veremos más adelante en esta práctica.*

`return` es una instrucción especial que indica el valor devuelto por la función (y que además termina la ejecución de la función, de manera que cualquier instrucción tras el **`return`** **no se ejecutará nunca**).

Si la función no devuelve nada, no hace falta la sentencia **`return`** o podemos ponerla sin ningún valor.

Por ejemplo, esta función acepta como parámetro único un entero y lo muestra por pantalla, pero no devuelve nada.

```
def VerNumero(x: int):
    print(x)
```

```
def VerNumero(x: int):
    print(x)

    return
```

Para llamar (invocar) a la función, es decir, para que se ejecuten sus instrucciones, ponemos su nombre en cualquier punto de nuestro programa, y en el lugar de los parámetros, por el mismo orden, ponemos el nombre de las variables o constantes o el valor constante que queremos asignar a los parámetros. Las variables o las constantes que pasemos, no hace falta que se llamen igual que el

parámetro, de hecho lo normal es que se llamen diferente, porque son variables diferentes. Durante la ejecución, al llamar a la función, el valor de la variable o la constante se asigna al parámetro correspondiente y comienza a ejecutarse cada una de las instrucciones de la función.

Si la función devuelve algo, podemos recuperar el valor que devuelve (lo que devuelve con **`return`**) asignándolo a una variable del mismo tipo que se devolvió con el **`return`**.

Por ejemplo, podemos invocar así a nuestra función potencia desde otra parte del programa:

```
# siendo a, b dos variables float

a = Potencia (b, 4)

# Por ejemplo, si b vale 2.0, tras la instrucción anterior se pondrá
# el valor 16.0 en la variable a
```

O:

```
# siendo base, y res dos variables float y expo una variable entera

res = Potencia (base, expo)

# Por ejemplo, si base vale 3.2 y expo vale 2, tras la instrucción anterior
# se pondrá el valor 10.24 en la variable res
```

Tampoco es obligatorio que las funciones acepten parámetros. Por ejemplo, esto es un caso extremo de una función que no tiene parámetros y no devuelve nada:

```
def MostrarPresentacion():
    print("Hola a todos")
```

```
def DecirHola():
    print("Hola a todos")

    return
```

2.- Funciones con más de un valor de retorno

Si necesitamos que una función devuelva más de un valor, se puede utilizar la sentencia return con los valores que deseemos retornar separados por comas.

```
# Función con más de un valor de retorno
#   def Funcion (param):
#       ...
#       return val1, val2, ...
```

Ejemplo:

```
def DivisionEntera(a: int, b: int)-> (int, int):
    """
    Realiza la división entera entre a y b, devolviendo cociente y
    resto

    Args:
        a (int) Dividendo
        b (int) Divisor

    Returns:
        c (int) Cociente de la división
        r (int) Resto de la división
    """

    ...
    return c, r
```

En el programa principal:

```
def main():
    print("Dame numeros a dividir: ")
    dividendo = int(input("Dame dividendo: ")
    divisor = int(input("Dame divisor: ")

    cociente, resto = DivisionEntera(dividendo, divisor)

    print("Resultado:", cociente)
    print(" (con resto:", resto, ")\n")

if __name__ == '__main__':
    main()
```

3.- Guía de estilo

Comentarios

Como documentación del programa, se deberá escribir para cada función su nombre, los parámetros de entrada y salida que posee y una breve descripción de qué hace. La descripción y los parámetros de entrada y salida se deberán poner como comentarios a la función:

```
def Potencia(base: float, exponente: int)-> float:
    """
    Calculo de la potencia de un numero (Producto que resulta de
    multiplicar la base por si misma un numero de veces igual al
    exponente)

    Args:
        base (float) Base de la potencia
        exponente (int) Exponente de la potencia

    Returns:
        potencia (float) Resultado de la potencia base elevado a exponente
    """

    ...
```

Anotaciones en funciones

Cuando se define una función, es interesante saber qué tipo de información espera y qué tipo de información devuelve.

Esta información debe estar presente en los comentarios a las funciones, tal y como se explica en la Guía de Estilo de Python.

Más allá de esa información, la complementaremos mediante la utilización de anotaciones.

Se añadirán anotaciones a los parámetros añadiendo dos puntos y el tipo de información del parámetro. También añadiremos anotaciones de los valores de retorno de la función, mediante '**->**' antes de los dos puntos de final de línea de la cabecera de la función. Si hay varios valores de retorno se pondrán paréntesis englobando los tipos devueltos.

```
def Sumar(a: int, b: int)-> int:
    ...

def CalcularMaximoMinimo(a: float, b: float, c: float)-> (float, float):
    ...
```

Cuando acabes la primera parte de la práctica tendrás un programa que hace uso de distintas funciones. Para poder probar las funciones mientras vas construyéndolas, sin esperar a acabar el programa, puedes crear en el programa principal (**`main`**) varias llamadas a tus funciones, con valores diferentes para ver qué tal funcionan.

Recuerda que para usar una función tienes que haberla definido ***previamente***.

Las funciones deben de ser lo más independientes posible del resto del programa, por lo que sólo se pueden comunicar con el resto del programa mediante los parámetros y valores de retorno.

NO hay que utilizar variables globales (prohibidas en esta práctica y en toda la asignatura).

En esta primera parte de la práctica irás construyendo paso a paso un programa, comenzando por las funciones que usarás más adelante. Muchos ejercicios dependen de los anteriores. Intenta realizar cada ejercicio siguiendo el orden de la práctica.

4.- Realiza los siguientes ejercicios de programación en Python

Tarea 1: Realiza una función llamada **`NumeroEnRango`** con tres parámetros de entrada que serán números enteros:

`num`: el número que se va a comprobar si está dentro de un rango
`lim_inf`: límite inferior
`lim_sup`: límite superior

La función debe devolver un valor lógico. El valor devuelto será:
`true` si el número es mayor o igual que el límite inferior y menor o igual que el límite superior (el número está en el rango)
`false` si el número está fuera del rango

Recuerda que **solo** puede haber una instrucción **`return`** en la función y esta instrucción **`return`** **debe ser la última** de la función.

Tarea 2: Realiza una función llamada **`PedirNumero`** que pida un número entero que esté entre dos límites (incluyéndolos) y lo devuelva. **Utiliza la función anterior** para hacer las comprobaciones pertinentes.

A esta función hay que pasarle los límites entre los que debe estar el número. La función tiene que:

- Pedir al usuario un número, indicando el rango en el que debe estar (ver ejemplo).
- Si está fuera del rango, dar un mensaje de error y repetir el mensaje anterior (pedir el número)
- Si el número introducido está en el rango, la función devolverá ese número

Cumple exactamente este ejemplo: Se llama a la función pasando [3, 7] como rango. El usuario intenta dar como valor el 2, el 8, el 1 (valores incorrectos) y al final teclea un valor correcto, el 5:

```
[entre 3 y 7 incluidos]: 2
---> ERROR: Fuera del rango [3, 7]
[entre 3 y 7 incluidos]: 8
---> ERROR: Fuera del rango [3, 7]
[entre 3 y 7 incluidos]: 1
---> ERROR: Fuera del rango [3, 7]
[entre 3 y 7 incluidos]: 5
```

(y la función devolverá un 5)

Tarea 3. Realiza otra función que devuelva la hora (hora, minuto y segundo) llamada **`PedirHora`**. Esta nueva función tiene que pedir al usuario una hora del día válida. Primero pide las horas, luego los minutos, y luego los segundos, y devuelve los tres valores.

Utiliza las funciones que ya tienes.

Ejemplo de ejecución:

```
Horas:
[entre 0 y 23 incluidos]: 22
Minutos:
[entre 0 y 59 incluidos]: 65
---> ERROR: fuera de rango [0, 59]
[entre 0 y 59 incluidos]: 2
Segundos:
[entre 0 y 59 incluidos]: 7
```

Tarea 4: Implementa una función **`VerHora`**, que acepte tres valores (horas, minutos y segundos) y los muestre en formato hh:mm:ss (con ceros a la izquierda si es necesario).

Nota: Para poner estos ceros, puedes consultar el método **`zfill`** aplicado a la cadena a representar. Puedes probar el siguiente ejemplo y adaptarlo a tus necesidades:

```
def EscribeNumero (num, tam):
    print(str(num).zfill(tam))

def main ():
    numero = int(input("Dame numero: "))
    tamaño = int(input("Dame tamaño: "))

    EscribeNumero(numero, tamaño)

if __name__ == '__main__':
    main()
```

Tarea 5: Crea una función **`ConvertirHoraNum`**, que permitirá convertir una hora en un número, según una determinada fórmula.

A la función se le pasarán tres números (que serán las tres partes de una hora: horas, minutos, y segundos) y devolverá como resultado este cálculo:

```
res = (hora * 10000) + (minutos * 100) + segundos
```

Esta función será útil en nuestro programa para guardar una hora (3 números) en una sola variable.

Además, si un número calculado de esta forma es mayor que otro, sus horas correspondientes guardarán esa misma relación, por lo que será útil más adelante para comparar horas.

Tarea 6: Crea una función **`VerHoraNum`** a la que se le pase un número calculado según la función anterior. La función se limitará a mostrar por pantalla la hora con la que se ha calculado ese número.

Por ejemplo, con la llamada **`VerHoraNum(172307)`** se mostrará por pantalla: **`17:23:07`**

Utiliza las funciones que ya tienes.

Tarea 7: Utilizando adecuadamente las funciones que debes tener hechas si has completado todos los apartados anteriores, realiza un programa **`Practica5_Tarea7_Jugar.py`** que consistirá en el siguiente juego:

Primero se leerá de un fichero, previamente creado y que contiene 3 enteros (horas, minutos y segundos y en ese orden), "**`hora.txt`**" **una hora válida**.

Después se pedirá al jugador que adivine la hora guardada en el fichero. Si el jugador introduce una hora anterior a la que tiene que adivinar, aparecerá "**`Has puesto hh:mm:ss, pero es más tarde`**". Si es posterior, aparecerá el mensaje "**`Has puesto hh:mm:ss, pero es antes`**". Si la adivina, aparecerá el mensaje "**`Has puesto hh:mm:ss ACERTASTE!!! (has tardado X jugadas)`**". Todas las horas introducidas por el jugador se guardarán en un fichero de texto con nombre "**`horas_jugadas.txt`**".

Tras terminar el juego, y antes de terminar el programa, se mostrarán por pantalla primero todas las horas menores que la leída del fichero y a continuación todas las mayores.

5.- Funciones Recursivas

Dentro del código de una función recursiva siempre hay una sentencia o expresión donde aparecerá la **llamada a la misma función**. Para no generar infinitas llamadas (dado que la función se llama a sí misma) necesitamos tener en cuenta lo siguiente:

- Una función recursiva resolverá un problema de talla N, considerando resuelto el problema de talla N – 1. Por tanto, **los parámetros de la función deben variar** de una llamada a otra.

ej:
```
Factorial(N) = N * Factorial(N - 1);
// problema_de_talla_N = N * problema_de_talla_N - 1
```

- En toda función recursiva **necesariamente** debe aparecer la (o las) **condición de parada** (caso base), que estará relacionada con alguno de los parámetros de la función. Al detectar este caso, no se producirán más llamadas recursivas y se devolverá el valor correspondiente al caso sencillo o base (ej: **`Factorial(0) = 1`**)

6.- Realiza los siguientes ejercicios de programación en Python

Tarea 8: Realiza un programa **`Practica5_Tarea8_Contar.py`** con una función recursiva **`Contador`** que muestre la cuenta atrás de un número positivo.

```
Dame numero inicial: 8
Contador descendente:
Contador a 8
Contador a 7
Contador a 6
Contador a 5
Contador a 4
Contador a 3
Contador a 2
Contador a 1
>>>
```

El programa principal es el siguiente:

```
def main ():
    num = int(input("Dame numero inicial: "))

    print("Contador descendente: ")
    Contador (num)
```

Tarea 9: En general, el N-ésimo término de la sucesión de Fibonacci, se define como:

$$F(n) = \begin{cases} 0 & si \quad n = 0 \\ 1 & si \quad n = 1 \\ F(n-1) + F(n-2) & si \quad n > 1 \end{cases}$$

Escribe una función recursiva que devuelva el término n-ésimo de la sucesión de Fibonacci.

Escribe un programa **`Practica5_Tarea9_Fibonacci.py`** con un programa principal que pedirá el término de fibonacci que se quiere determinar y lo mostrará por pantalla.

7.- Ejercicios de laboratorio y entrega a través de a AulaVirtual

Una vez terminadas las tareas del guión, en el aula, el profesor de prácticas te entregará el enunciado del o los ejercicios de laboratorio que debes resolver durante la sesión de prácticas.

Al acabar estos ejercicios debes subirlos a AulaVirtual.

- Coge los códigos de los programas que tengas que subir como solución, es decir los ficheros **`.py`**.
- Comprímelos en un solo fichero (por ejemplo un **`.zip`** o un **`.rar`**)
- Conéctate a **`aulavirtual.uv.es`** y en la tarea correspondiente a la práctica de laboratorio de tu subgrupo de laboratorio, cuelga la solución.

Se os recuerda que en la escritura de vuestros programas debéis seguir adecuadamente las normas de tabulación, comentarios y separaciones descritas en la '**Guía de Estilo de Python**'.