TEMA 6

Funciones

Curso de Python

Alfredo Sánchez Sánchez
@alfsan04
www.educoteca.com

ÍNDICE

Funciones

Llegados a este punto vamos a hacernos una pregunta: si hay una instrucción que se repite muchas veces en un programa… ¿no podría escribirlo una única vez y que el programa sepa lo que tiene que hacer cada vez que toque? Algo así como lo que ocurre cuando usas una biblioteca (recordad que hemos usado la biblioteca *time* y la biblioteca *random*).

Pues sí, podemos utilizar una función de Python que nos permite almacenar el código que necesitemos ejecutar y aplicarlo a cualquier parte de nuestro programa. Esta función lo que hace es encapsular una fracción de código y, cuando nosotros le digamos, la utiliza en aquel punto del script que sea necesario.

Aprendamos pues a crear funciones, estructuras de código que se pueden reutilizar a lo largo de un programa las veces que necesitemos. Para crear una función hay que definirla:

```
def miPrimeraFuncion():
```

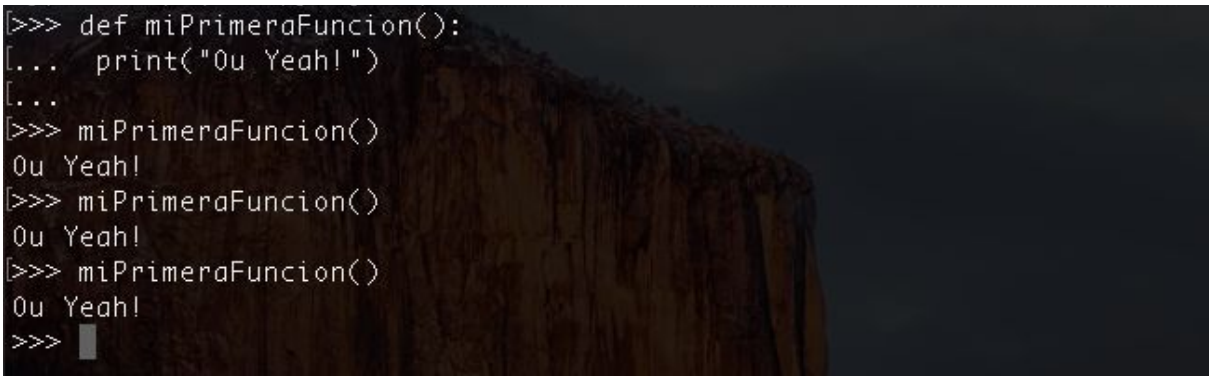
Una vez que hemos definido nuestra función sólo tenemos que crear el código que queremos almacenar en la función, por ejemplo:

```
def miPrimeraFuncion():
   print("Ou Yeah!")
```

Posteriormente podremos llamar a la función en cualquier punto de un programa simplemente mencionándola:

```
def miPrimeraFuncion():
   print("Ou Yeah!")
miPrimeraFuncion()
miPrimeraFuncion()
miPrimeraFuncion()
```

Aquí tienes una captura de cómo se utiliza por terminal:



¡Fantástico! Vamos a darle uso. Guarda el siguiente programa como script y ejecútalo por terminal o IDLE:

```
terminamos = 0
#Creo mi función
def funcionSuma():
    #Pregunto los números a sumar
    numero1 = input("Primer numero: ")
    numero2 = input("Segundo numero: ")
    #Los convierto en entero:
    numero1 = int(numero1)
    numero2 = int(numero2)
    #Sumo los números y los imprimo
    suma = numero1+numero2
    print(suma)
while terminamos == 0:
    print("Nuestras opciones:")
    print("1. Realizar una suma")
    print("2. Terminar")
    pregunta = input("¿Qué quieres hacer? (1 o 2): ")
    if pregunta == "1":
        #Llamo a la función
        funcionSuma()
    elif pregunta == "2":
        print ("Adiós")
        terminamos = 1
```

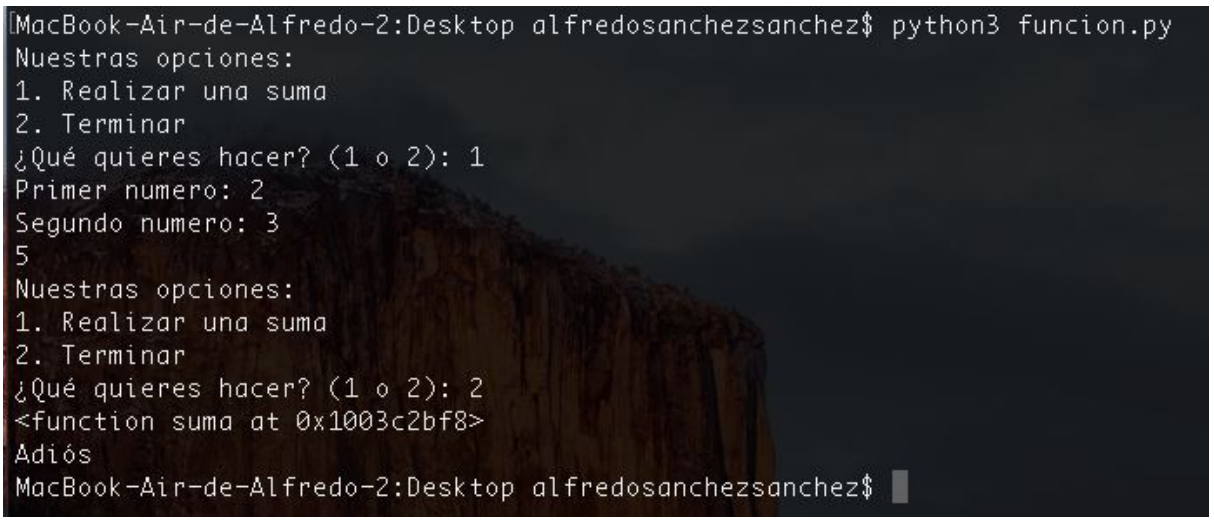
Si lo ejecutas obtendrás lo siguiente:

¡Perfecto! Hemos conseguido sumar dos números y hacerlo las veces que haga falta. Ahora bien, a veces querremos usar el resultado de la ejecución de la función fuera de la misma, además de imprimirlo. En nuestro ejemplo anterior, sin ir más lejos, puede que en otra parte posterior de nuestro script tengamos que utilizar la suma que hemos realizado.

Vamos a incluir una línea que imprima el valor que tiene la variable *suma* cuando queramos salir del programa:

```
terminamos = 0
def funcionSuma():
    numero1 = input("Primer numero: ")
    numero2 = input("Segundo numero: ")
    numero1 = int(numero1)
    numero2 = int(numero2)
    suma = numero1+numero2
    print(suma)
while terminamos == 0:
    print("Nuestras opciones:")
    print("1. Realizar una suma")
    print("2. Terminar")
    pregunta = input("¿Qué quieres hacer? (1 o 2): ")
    if pregunta == "1":
        funcionSuma()
    elif pregunta == "2":
        #Tratamos de imprimir la variable suma propia de la función
        print(suma)
        print("Adiós")
        terminamos = 1
```

Aquí tenéis el resultado de ejecutarlo:



Vaya, no está saliendo la cosa como esperábamos. Al presionar el 2 el script se finaliza, sí, pero no me muestra el valor de la variable suma que era 5, me dice que la función suma está en… un número algo complejo (*0x1003c2bf8*).

Este ejercicio que acabamos de hacer es muy importante y además nos ha llevado de lleno a uno de los problemas de Python para la gente que ha programado en otros lenguajes: el ámbito de las variables.

Las variables se crean en un ámbito. Si están creadas en el programa general (lo que hemos hecho hasta ahora) **se pueden usar en ese ámbito** y **se pueden leer desde otros ámbitos, pero no modificarlas.**

Un ámbito es un espacio de código, está delimitado. Para entenderlo podemos poner un ejemplo: piensa en un edificio de viviendas. Cada propietario tiene su casa, es su ámbito y sólo él puede usar todo lo que en ella se encuentra puede ser usado. Ahora bien, cada propietario tiene también acceso a las zonas comunes (pasillos, ascensor, parking…). Las cosas que allí se encuentran son accesibles por todos los propietarios ya que es un ámbito global, pero no pueden ser modificadas por un propietario ya que no son exclusivas del mismo.

Si una variable está creada en un ámbito concreto (como el ámbito de una función), sólo existe en ese ámbito. Se crea con el ámbito y se destruye al acabar este. Por ello, toda variable creada en una función no puede ser utilizada fuera de la función a no ser que indiquemos que esa variable será **global.** Las variables usadas en la función anterior son **locales**, es decir, se crean con la función y tienen vida en su ejecución, desapareciendo tras ello.

Podemos, a nuestra voluntad, indicar que en un ámbito vamos a usar una variable global. Para ello tenemos que indicar al principio de una función que vamos a crear una variable pero que queremos que permanezca, es decir, que sea global, lo cual se hace simplemente poniendo *global* seguido del nombre de la variable:

```
terminamos = 0
def funcionSuma():
    #Creo mi variable global
    global suma
    numero1 = input("Primer numero: ")
    numero2 = input("Segundo numero: ")
    numero1 = int(numero1)
    numero2 = int(numero2)
    suma = numero1+numero2
    print(suma)
while terminamos == 0:
    print("Nuestras opciones:")
    print("1. Realizar una suma")
    print("2. Terminar")
    pregunta = input("¿Qué quieres hacer? (1 o 2): ")
    if pregunta == "1":
        funcionSuma()
    elif pregunta == "2":
        print(suma)
        print("Adiós")
        terminamos = 1
```

Si pruebas el código anterior verás que, efectivamente, está devolviendo el valor de la variable suma antes de terminar.

Ahora bien, el problema de las variables globales es que muchas veces perdemos su rastro, permanecen creadas en programas muy extensos y no las usamos más o bien cambiamos su valor en determinados sitios del programa sin saber dónde se está cambiando y pueden llevarnos a errores. No es una práctica recomendable en programación en Python, por lo que existe una forma más sencilla de conseguir que el resultado de una función pueda ser utilizado fuera del ámbito de la misma: **las funciones con retorno.**

Funciones con retorno

Las funciones con retorno se ejecutan igual que las funciones que ya hemos visto y, al final de su ejecución, nos entregan todo aquello que le pidamos que nos entreguen, es decir, nos **retornan valores** que podemos almacenar y usar.

Para que una función se convierta en una función con retorno simplemente tengo que terminar con un *return* e indicar qué quiero que la función me devuelva. En nuestro caso necesitamos que nos devuelva la suma de *numero1* y *numero2*. Eso sí, es importante saber que **el valor que retorna la función debe ser almacenado en una variable** si queremos usarlo.

Para ello, a la hora de llamar a la función, la llamaremos dentro del valor de una variable (por ejemplo *suma = funcionSuma( )*):

```
terminamos = 0
def funcionSuma():
    numero1 = input("Primer numero: ")
    numero2 = input("Segundo numero: ")
    numero1 = int(numero1)
    numero2 = int(numero2)
    #Devuelvo la suma de los números
    return numero1+numero2
while terminamos == 0:
    print("Nuestras opciones:")
    print("1. Realizar una suma")
    print("2. Terminar")
    pregunta = input("¿Qué quieres hacer? (1 o 2): ")
    if pregunta == "1":
        """Llamamos la función suma y almacenamos el retorno en la
variable suma"""
        suma = funcionSuma()
    elif pregunta == "2":
        print(suma)
        print("Adiós")
        terminamos = 1
```

Guarda el código anterior y ejecútalo como script desde Terminal o IDLE.

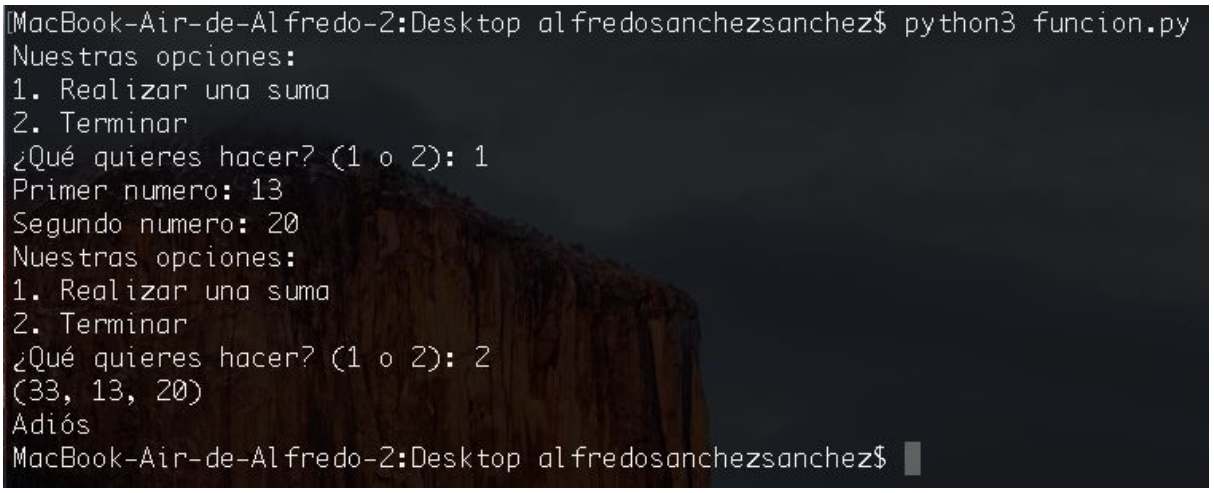
Como habrás podido comprobar, sale exactamente lo mismo que con la variable global, pero es una mejor práctica, ya que la variable se usa en su ámbito y nos devuelve un valor que nosotros podemos almacenar cómo y dónde queramos.

La realidad es que el programa anterior es poco útil, una suma es algo que es casi más sencillo hacer directamente que ponerlo en una función, pero podemos realizar funciones para diferentes opciones que sí requieren códigos más complejos y ahorramos un proceso tedioso, sobre todo si la función se repite muchas veces a lo largo del código.

Una función con retorno nos puede devolver los valores que queramos, pero sólo tenemos una opción de retornar un valor, una vez retornamos un valor se acaba la ejecución de la función. Si queremos retornar **varios valores** lo indicaremos todo dentro del *return* y Python nos **devolverá una lista**. Cambiemos el *return* de nuestra función y veamos qué pasa:

```
terminamos = 0
def funcionSuma():
    numero1 = input("Primer numero: ")
    numero2 = input("Segundo numero: ")
    numero1 = int(numero1)
    numero2 = int(numero2)
    #Devuelvo la suma de los números
    return numero1 + numero2, numero1, numero2
while terminamos == 0:
    print("Nuestras opciones:")
    print("1. Realizar una suma")
    print("2. Terminar")
    pregunta = input("¿Qué quieres hacer? (1 o 2): ")
    if pregunta == "1":
        suma = funcionSuma()
    elif pregunta == "2":
        print(suma)
        print("Adiós")
        terminamos = 1
```

Vamos a ejecutar el código anterior:



Como puedes ver, el script finaliza devolviendo una lista con valores *(33, 13, 20)* antes de despedirse. Como ya sabemos utilizar listas, podemos perfectamente decidir cómo usamos los valores que devuelve la función, pero es importante tener claro qué valores nos está devolviendo la función.

Ahora que sabemos utilizar funciones con retorno vamos a ver un paso más en las funciones: funciones a las que, al llamarlas, le indicamos ciertos parámetros o valores. Por ejemplo, al llamar a la función suma podríamos indicarle los números que tiene que sumar, algo así como *suma (2, 6)* para sumar un 2 y un 6. Dichas funciones se denominan funciones con parámetros.

Funciones con parámetros

Si os fijáis, al definir las funciones anteriores teníamos unos paréntesis vacíos detrás: *def suma ( )*

Entre esos paréntesis podemos indicar valores que deberán ser entregados a la hora de llamar a la función. En el caso de la suma podemos indicar que cuando queramos hacer la suma se darán los números a sumar.

En la función podemos determinar tantos parámetros como queramos recibir, si yo quiero realizar una suma de dos números declararé que van a llegar dos números:

```
def suma(numero1,numero2)
    suma = numero1+numero2
    return suma
```

La función anterior, al ser llamada, sumará los dos parámetros que le llegan entre paréntesis, a los que llamamos *numero1* y *numero2* en la definición de la función pero que realmente **tendrán el valor que declaremos a la hora de llamar a la función**:

```
def suma (numero1, numero2):
    suma = numero1 + numero2
    return suma
operacion = suma(2,7)
print(operacion)
```

El código anterior tiene definida una función que realiza una suma y posteriormente llamamos a la función, almacenando el resultado retornado en la variable *operacion*.

Por experiencia, este proceso es complicado de entender si no tenemos soltura en algún lenguaje de programación. El asunto es que, al llamar a una función que tiene parámetros, tenemos que indicar dichos parámetros para que la función se pueda ejecutar. Al definir la función *suma* declaramos que va a recibir dos números al ser llamada y que tiene que sumar dichos números.

En lenguaje humano es fácil de describir este proceso, le estamos diciendo a Python:

"Cuando el usuario lo necesite te va a llamar y te va a dar dos números, tu tienes que sumar dichos números y darle el resultado de la suma".

Ahora bien, el proceso se complica un poquito sí en vez de dar dos números al llamar a una función doy dos variables. Es decir, quiero sumar dos números, pero esos números están

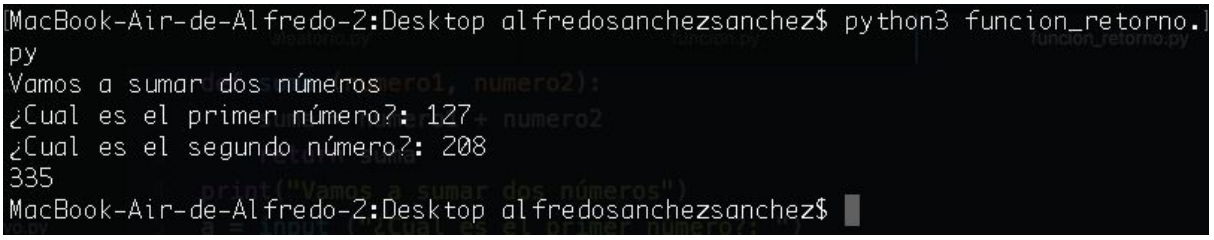
dentro de dos variables (imagina que queremos un programa en el cual el usuario pueda indicar qué números quiere sumar, como pasaba en el ejemplo de la función con retorno).

Veamos cómo quedaría el código:

```
def suma(numero1,numero2):
    suma = numero1+numero2
    return suma
print ("Vamos a sumar dos números")
a = int(input("¿Cual es el primer número?: "))
b = int(input("¿Cual es el segundo número?: "))
operacion = suma(a,b)
print(operacion)
```

Guarda el código anterior como script y ejecútalo antes de seguir.

Aquí tienes la captura del programa ejecutado:



Vamos a estructurar cómo sucede el script:

1. El script indica que vamos a sumar dos números.
2. Pregunta el primer número y guarda el valor contestado en la variable *a*.
3. Pregunta el segundo número y guarda el valor contestado en la variable *b*.
4. Llama a la función *suma* indicando como parámetros *a* y *b* e indica que el resultado debe almacenarse en la variable *operacion*.
5. Al ser llamada, la función *suma* debe recibir dos valores, que dentro de la función serán llamados *numero1* y *numero2*. Los números recibidos son dos variables (*a* y *b*), lo cual quiere decir que la función *suma* operará con los números que contienen dichas variables.
6. Imprimimos el valor de la variable *operacion*, que será el resultado de la suma de las variables *a* y *b*.

El proceso de las variables es el siguiente:

La función espera *numero1* → *numero1* recibe *a* → *a* vale 127 → *numero1* pasa a valer 127

Espero que haya quedado claro, no tengas problema en repasarlo con calma, es importante para el futuro asentar este proceso.

Vamos a ver algo un poco más "útil" que una simple suma. Veamos una función para determinar la media:

```
def media(numero1,numero2):
    suma = numero1+numero2
    media = suma/2
    return media
media = media(17,40)
print(media)
```

Puedes probar el script tras guardarlo.

El resultado del script será *28,5*, es decir, la media de los números 17 y 40. Ahora bien, las medias no son siempre de 2 números y tampoco tenemos por qué conocer los números de los que hay que hacer la media al crear el script… ¿qué tal si vemos cómo puedo hacer una función que calcule la media sin saber de cuántos números hay que hacerla?

Si, al declarar la función, situamos delante de uno de sus parámetros un asterisco, le estamos diciendo a Python que va a llegar un **arbitrario**, que no es más que un conjunto de números de los que desconocemos su longitud:

```
def media(*numeros):
```

A la hora de llamar a la función habrá que declarar los números de los que necesitamos hacer la media:

```
def media (*numeros):
    #Aquí se incluirá el código necesario para hallar la media
miMedia = media(6,7,8,1,5,1)
```

El resultado que me devuelva la función *media* será guardado en la variable *miMedia*.

Los parámetros arbitrarios se utilizan como si fuesen tuplas, se pueden recorrer para ver sus valores con un bucle *for* y se puede ver cuántos datos contiene con la función *len*.

```
def media(*numeros):
    a = 0
    for numero in numeros:
        #Recorro el arbitrario y voy almacenando el valor acumulado
        a = a+numero
    """Divido la suma de todos los números que contiene el arbitrario
entre su longitud"""
    media = a/len(numeros)
    return media
#Aplico la función a unos números concretos
miMedia = media(5,7,9,1,2,8)
miMedia = str(miMedia)
print("La media es "+miMedia)
```

Puedes probar el código anterior guardándolo como script y ejecutándolo:

```
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$ python3 arbitrarios.py
La media es 4.666666666666667
MacBook-Air-de-Alfredo-2:Desktop alfredosanchezsanchez$
```

Habría que utilizar la función de Python para redondear el resultado a dos o tres decimales, pero por lo demás, tenemos una media.

Para recoger todo lo tratado de las funciones veamos el siguiente esquema:

- Si quiero una función que repita un código que únicamente imprime un valor, puedo usar una **función**.
- Si quiero una función que me dé un valor para usar fuera de la misma, es decir, en cualquier punto del programa, usaré una **función con retorno**.
- Si quiero que la función, al ser llamada, ejecute su código con unos valores que variarán según cuando sea llamada utilizaré una **función con parámetros**. Dicha función puede ser con retorno o sin retorno, en función de lo que necesitamos.
- Por último, si la cantidad de valores variables que necesitará la función es también desconocida crearemos una **función con parámetros arbitrarios**.

Veamos una última posibilidad en las funciones con parámetros y retorno. Imagina que queremos hallar la media de una serie de números que debe decidir el usuario. Podríamos construir el siguiente programa:

1. El programa pregunta al usuario de cuántos números necesita calcular la media.

2. A continuación va preguntando cada número e introduciéndolo en una lista.

3. Cuando tenemos la lista completa llamaremos a una función que realice la media con los valores de la lista.

Vamos a ver primero el programa para guardar los números que quiera el usuario:

```
cantidad = input("¿De cuántos valores necesitas hacer la media?: ")
cantidad = int(cantidad)
valores = []
for i in range(cantidad):
    i = str(i+1)
    valor = input("Valor número "+i+": ")
    #Convertimos a float por si el usuario introduce un número decimal
    valor = float(valor)
    valores.append(valor)
print(valores)
```

Guarda el código anterior como script y ejecútalo para ver qué está ocurriendo, aunque deberías tener claro cómo va a funcionar llegados a este punto.

Una vez lo tengas claro vamos a crear la función que reciba esa lista y la recorra y haga la media. Es sencillo, de la misma forma que puedo enviar datos por parámetros puedo enviar listas. Una vez la recibo, tengo que recorrerla y realizar la media de todos sus elementos:

1. El programa recibe la lista

2. La recorre y va acumulando la suma de sus valores en una variable

3. Divide el valor acumulado en la variable entre el número de elementos de la lista

4. Devuelve el valor conseguido

¿Crees que eres capaz de hacerlo sin ver cómo se hace? Intenta crear esa función antes de seguir, si te genera problemas sigue leyendo para ver cómo se haría:

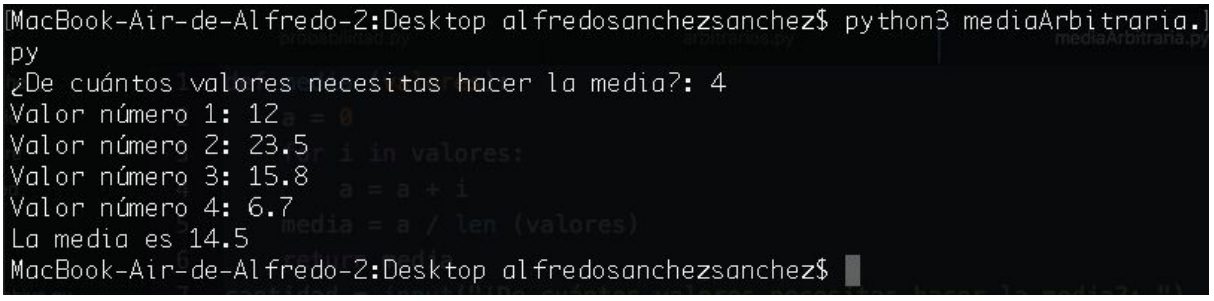
```
def media(valores):
    a = 0
    for i in valores:
        a = a+i
    media = a/len(valores)
    return media
```

La función recibe un parámetro llamado *valores* que, en el caso de nuestro programa, será una lista. A continuación creamos la variable *a* para ir guardando el valor de la suma de los valores que tiene nuestra lista. Una vez hemos recorrido la lista *valores* entera dividimos la suma de su contenido (*a*) entre el número de valores que tiene la lista (*len (valores)*) y guardamos el resultado en la variable *media*, devolviendo ese valor como retorno al programa.

Ahora sólo queda unir los ejemplos anteriores y llamar a la función *media* dando como parámetro la lista donde hayamos guardado los números e imprimir el resultado. El programa completo quedaría así:

```
def media(valores):
    a = 0
    for i in valores:
        a = a+i
    media = a/len(valores)
    return media
cantidad = input("¿De cuántos valores necesitas hacer la media?: ")
cantidad = int(cantidad)
valores = []
for i in range(cantidad):
    i = str(i+1)
    valor = input("Valor número "+i+": ")
    valor = float(valor)
    valores.append(valor)
miMedia = media(valores)
miMedia = str(miMedia)
print("La media es "+miMedia)
```

Aquí dejo una captura de pantalla de la ejecución del código con varios números para que se vea cómo funciona:



Y con ésto podemos dar por concluido el tema de funciones. Con el tiempo irás descubriendo muchas más opciones y utilidades de las funciones, pero de momento tienes una buena base para utilizarlas.

Instrucciones de código usadas

```
#Definir una función
   def funcion():
#Llamar a una función para que se ejecute
   funcion()
#Crear una variable global
   global variable
#Función con retorno
   def funcion():
      #Código
      return valor
#Función con parámetros
   def funcion(parametros):
#Función con parámetros arbitrarios
   def funcion(*arbitrarios)
```

Reto

Crea un script en el cual el usuario pueda pedir al programa combinaciones aleatorias para jugar a euromillones (un juego de azar europeo). El script debe cumplir con la siguiente serie de requisitos:

1. Cada combinación para euromillones debe tener cinco números entre el 1 y el 50 y dos estrellas (otros dos números) del 1 al 12.
2. El programa preguntará al inicio al usuario cuántas combinaciones de euromillones quiere crear y las imprimirá separadas y con alguna descripción del tipo *Combinación 1:*
3. El programa debe preguntar al usuario si quiere generar más combinaciones o terminar el script tras cada proceso de generación de combinaciones.
4. La generación de combinaciones debe estar en una función con parámetros y retorno.