



Jul. 2022

Unidad 2: Funciones



Introducción - las funciones

- Son una **parte integral** de muchos lenguajes
- **Agrupar** líneas de **código** que implementan alguna funcionalidad
- Si el código en cuestión será **utilizado varias veces** desde diferentes partes del programa
- Usar funciones para **abstraer** algún código complejo
- **Mini-programas** dentro de tu programa más grande
- Es mejor escribir funciones que sólo realicen una **tarea específica** en lugar de meter mucha funcionalidad en una sola programa
- **Facilitan la modularización** del código
facilidad para mantener y depurar

Objetivos de aprendizaje

Ser capaz de:

- Describir los distintos tipos de funciones en Python
- Definir las variables globales y locales
- Definir una función que tome un número variable de argumentos

Funciones incorporadas

- El intérprete de Python tiene una serie de funciones incorporadas que están siempre disponibles
- Se pueden utilizar en cualquier parte del código sin necesidad de ninguna importación
- Ejem:
 - `input([prompt])`: Lee una línea de la entrada
 - `print()`: Imprime objetos
 - `map()`: Devuelve un iterador

Funciones definidas por el usuario

- Para ayudar a lograr un objetivo específico
- El uso principal es organizar los programas en fragmentos lógicos que trabajan juntos
- Sintaxis:

```
def nombre_de_la_función( parámetro_uno, parámetro_dos, parámetro_n ):  
    # La lógica va aquí  
    return
```

Llamar a una función

- Significa ejecutar la lógica que se define dentro de ella
- A menudo son llamadas desde otras funciones
- Cualquier nombre para los argumentos que pasamos no tienen porqué coincidir con los nombres de los parámetros que la función espera

Variables globales y locales

- Variables locales
 - Se definen dentro del cuerpo de una función
 - Sólo son accesibles dentro de la función (ámbito local)
- Variables globales
 - Se definen fuera del cuerpo de una función
 - Son accesibles tanto fuera como dentro de las funciones (ámbito global)

E1: Definición de variables globales y locales

1. Define una variable global, `numero` inicializada a 5
2. Define una función llamada `suma` que toma dos parámetros de nombre: primero y segundo
3. Dentro de la función, suma los dos parámetros que se han pasado y la variable global `numero`, devuelve el total
5. Llama a la función de suma con dos parámetros (10 y 20)
6. Imprime el valor del número y el de la variable total

Salida de ejemplo:

El primer número que inicializamos fue 5
El total después de la suma es 35

Uso de `main()`

- Indica al sistema operativo qué código debe ejecutar cuando se invoca al programa
 - Requerido la mayoría de lenguajes de programación (Java y C++)
 - No es necesario en Python
- Estructurar un programa de forma lógica
- El programa se ejecutará por sí mismo, de forma autónoma
- Se define algunas variables especiales
 - `__name__` y se establecerá automáticamente a `__main__`

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

E2: Restructurar el código con main()

1. Restructurar el ejercicio anterior con la función main()

Ejemplo de la estructura:

```
def suma():  
  
def main():  
  
if __name__ == "__main__":  
    main()
```

Parámetros de la función

- Son la información que se necesita pasar a la función para que haga su trabajo
- Los argumentos son a las funciones como los ingredientes son a una receta.
- Tipos de argumentos:
 - Requeridos (tienen que estar presentes cuando se llama a una función)
 - Palabra clave (identificando los argumentos por sus nombres)
 - Por defecto (asignar un valor por defecto)
 - Número variable de argumentos (reciba cualquier número de variables, `*args`)

Tipos de argumentos en parámetros de la F

Argumentos requeridos

```
def divi(primerο, segundo):  
    return primerο/segundo  
  
cociente = di(10, 2)
```

Argumentos por defecto

```
def division(primerο, segundo=2):  
    return primerο/segundo  
  
cociente = division(10, 5)
```

Argumentos de palabras clave

```
def divi(primerο, segundo):  
    return primerο/segundo  
  
cociente = divi(segundo=2, primerο=10)
```

Número variable de argumentos

```
def suma(*args):  
    total = 0  
    for i in args:  
        total += i  
    return total  
  
respuesta = suma(20, 10, 5, 1)
```

Actividad 1: Argumentos de la función

- Escribe una función que reciba **n argumentos**
- Utiliza la sentencia *continue* para saltar los enteros e imprime todos los demás valores.

1. Definir una función llamada **imprimir_argumentos**

recibe un número variable de argumentos

```
imprimir_argumentos("a",1,"b",2,"c",3)
```

2. Utilice un bucle for para iterar sobre los argumentos
3. Compruebe si el valor del elemento es de tipo entero
Si lo es, utilice la sentencia *continue* para ignorarlo.
4. Imprime los argumentos que no son enteros

Salida de ejemplo:

```
a  
b  
c
```

Resumen

- Hemos aprendido
 - Sobre los distintos tipos de funciones, así como sus diferencias, sintaxis y casos de uso
 - Cómo y dónde aplicar los diferentes tipos de funciones
 - Cómo se pueden usar para ayudar a dividir tus programas en subprogramas que logren un propósito específico
 - El uso de funciones puede ayudar a reutilizar la funcionalidad del código
 - evitar repetir los mismos bloques de código