Educa2 – Clases y cursos de matemáticas y Programación

Curso Básico de Programación en Python

Preparador: Alexanyer Naranjo

Guía Teórica N°6

Funciones

Una función es un bloque de código que tiene asociado un nombre, de manera que cada vez que se quiera ejecutar el bloque de código basta con invocar el nombre de la función.

Para declarar una función se utiliza la siguiente sintaxis:

```
def <nombre_funcion> (<parámetros>):
    Bloque de Código
    return <valor>
```

Parámetros de una función:

Una función puede recibir valores cuando se invoca a través de unas variables conocidas como parámetros que se definen entre paréntesis en la declaración de la función. En el cuerpo de la función se pueden usar estos parámetros como si fuesen variables.

```
def bienvenido(nombre):
    print('¡Bienvenido a Python',nombre + '!')
```

Argumentos de la llamada a una función:

Los valores que se pasan a la función en una llamada o invocación concreta de ella se conocen como argumentos y se asocian a los parámetros de la declaración de la función.

```
def bienvenido(nombre,apellido):
    print('¡Bievenido a Python',nombre,apellido +'!')

bienvenido('Alexanyer','Naranjo')
bienvenido(apellido = 'Naranjo', nombre = 'Alexanyer')
```

Los argumentos se pueden indicar de dos formas:

- **Argumentos posicionales**: Se asocian a los parámetros de la función en el mismo orden que aparecen en la definición de la función.
- **Argumentos por nombre**: Se indica explícitamente el nombre del parámetro al que se asocia un argumento de la forma **parámetro = argumento**.

Retorno de una función:

Una función puede devolver un objeto de cualquier tipo tras su invocación. Para ello el objeto a devolver debe escribirse detrás de la palabra reservada return. Si no se indica ningún objeto, la función no devolverá nada.

```
def area_triangulo(base,altura):
    return base * altura / 2
```

Argumentos por defecto:

En la definición de una función se puede asignar a cada parámetro un argumento por defecto, de manera que, si se invoca la función sin proporcionar ningún argumento para ese parámetro, se utiliza el argumento por defecto.

```
def bienvenido(nombre,lenguaje = "Python"):
    print("¡Bienvenido a",lenguaje,nombre + '!')

bienvenido('Alexanyer')
bienvenido('Alexanyer','Java')
```

Pasar un número indeterminado de argumentos:

Por último, es posible pasar un número variable de argumentos a un parámetro. Esto se puede hacer de dos formas:

- ***parámetro**: Se antepone un asterisco al nombre del parámetro y en la invocación de la función se pasa el número variable de argumentos separados por comas. Los argumentos se guardan en una lista que se asocia al parámetro.
- ****parámetro**: Se anteponen dos asteriscos al nombre del parámetro y en la invocación de la función se pasa el número variable de argumentos por pares **nombre = valor**, separados por comas. Los argumentos se guardan en un diccionario que se asocia al parámetro.

```
def menu(*platos):
   print("Hoy tenemos: ")
    for plato in platos:
        print(plato)

menu('pasta','pizza','ensalada')
```

Ámbito de los parámetros y variables de una función:

Los parámetros y las variables declaradas dentro de una función son de ámbito local, mientras que las definidas fuera de ella son de ámbito ámbito global.

Tanto los parámetros como las variables del ámbito local de una función sólo están accesibles durante la ejecución de la función, es decir, cuando termina la ejecución de la función estas variables desaparecen y no son accesibles desde fuera de la función.

```
def bienvenido(nombre):
    lenguaje = 'Python'
    print('¡Bienvenido a', lenguaje, nombre + '!')

bienvenido('Alexanyer')
print(lenguaje) #Sucederá un error al ejecutar esta instrucción

#Error al ejecutar el programa
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'lenguaje' is not defined
```

Ámbito de los parámetros y variables de una función:

Si en el ámbito local de una función existe una variable que también existe en el ámbito global, durante la ejecución de la función la variable global queda eclipsada por la variable local y no es accesible hasta que finaliza la ejecución de la función.

```
lenguaje = 'Java'

def bienvenido():
    lenguaje = 'Python'
    print('¡Bienvenido a', lenguaje + '!')

bienvenido()
print(lenguaje)
```