Funciones

Una función es un bloque de código que realiza alguna operación. Una función puede definir opcionalmente parámetros de entrada que permiten a los llamadores pasar argumentos a la función. Una función también puede devolver un valor como .

Funciones

- La sentencia **def** es utilizada para definir funciones.
- Requieren de un nombre para su definición.
- Puede tener parámetros de entrada.
- Las sentencias de las funciones deben tener sangría.
- Todas las funciones retornan algo (Las anotaciones de retorno son definidas por el literal ->).
- La sentencia **return** retorna un valor en una función. **return** sin una expresión retorna **None**. Si se alcanza el final de una función, también se retorna **None.**

Sintaxis: **def** nombre_función([parámetros]) -> [tipo]:
código

Nota: Los parámetros y la anotación de retorno son opcionales. En Python todas las funciones retornan un valor.

Ing. Antonio Torres Sarmiento

Argumentos posicionales o clave valor

- Los parámetros pueden ser pasados a una función posicionalmente o por palabra clave.

Sintaxis: **def** nombre_función(arg_1 , arg_2):
código

Sintaxis: nombre_función(23 , “valor”)

nombre_función(arg_1 = 23 , arg_2 = “valor”)

Nota: Es la forma de definir funciones más utilizada.
Se pueden pasar argumentos de forma posicional o por palabra clave de forma combinada.

Argumentos únicamente posicionales

- Los parámetros pueden ser pasados únicamente por posición.
- Son ubicados antes de una barra (/).

Sintaxis:

```
def nombre_función( arg_1 , arg_2 , / ):
    código
```

Sintaxis:

```
nombre_función( 23 , “valor” )
```

Nota: Si no existe una / en la definición de la función, no existen parámetros únicamente posicionales.

Ing. Antonio Torres Sarmiento

Argumentos únicamente de palabras claves

- Los parámetros pueden ser pasados únicamente por palabra clave.
- Son ubicados después de una asterisco (*).

Sintaxis: **def** nombre_función(* , arg_1 , arg_2):
código

Sintaxis: nombre_función(arg_1 = 23 , arg_2 = “valor”)

Nota: Si no existe un * en la definición de la función, no existen parámetros únicamente posicionales.

Ing. Antonio Torres Sarmiento

Listas de argumentos arbitrarios

- Número arbitrario de argumentos.
- Los argumentos pueden ser organizados en una tupla y/o diccionario.
- Normalmente estos argumentos serán los últimos en la lista de parámetros formales.
- Cualquier parámetro que suceda luego del ***args y ****argskv** será “sólo de palabra clave”, o sea que sólo se pueden usar como argumentos nombrados y no como posicionales.

Sintaxis: **def** nombre_función(arg_1 , arg_2 , ***args , ****argskv**):
código

Sintaxis: nombre_función(23 , “valor” , 2 , 4 , 5 , 1 , 2 , 3 , 3 , 1 , **{ “a” : 2 , “b” : 3 })
nombre_función(23 , “valor” , * (2 , 4 , 5 , 1 , 2 , 3 , 3 , 1) , **{ “a” : 2 , “b” : 3 })

Desempaquetando una lista de argumentos

- Cuando una función que requiere argumentos posicionales separados y los argumentos están en una lista o tupla. se puede escribir la llamada a la función con el operador * para desempaquetar los argumentos de dicha lista o tupla.

- Del mismo modo, los diccionarios pueden entregar argumentos nombrados con el operador **.

Sintaxis:

```
def nombre_función( arg_1 , arg_2 , *args ,  **argskv):
    código

nombre_función( 23 , “valor”  , * ( 2 , 4 , 5 , 1 , 2 , 3 , 3 , 1 ) , **{ “a” : 2 , “b” : 3 } )
```