Funciones: Pasaje de parámetros

Materia: Programación orientada a objetos – UPSO
Profesor: Carlos Caseres

Sintaxis

Una función en Python se define con la palabra def:

sintaxis

```
def  nombreFuncion ( .. ):
     #código de la función
```

ejemplo

```
def  saludar ( ):
     print(“hola mundo”)
```

Invocación

Una función en sólo se ejecuta cuando es invocada dentro de un programa. La invocación se realiza mediante el nombre de la función, seguido de un paréntesis.

#definición de la función

```
def    saludar ( ):
         print(“hola mundo”)
```

#invocación en un programa

```
saludar()
```

Parámetros

Cuando una función necesita cierta información para poder realizar las operaciones requeridas, esta información se le envía mediante sus parámetros.

Ejemplo

```
def    saludar (nombre):
         print(“hola”,nombre)
```

#invocación

```
saludar(“carlos”)
```

Parámetros

Algunos mecanismos de formateo de cadenas

Ejemplo

```
def    saludar (nombre):
         print("hola",nombre)
         print("hola " + nombre)
         print(f"hola {nombre}")
```

#invocación

```
saludar("carlos")
```

Parámetros

Si se necesita mas de un dato, puede definirse la cantidad deseada de parámetros, separados por comas.

Ejemplo

```
def    saludar (nombre, apellido, edad):
          …
```

#invocación

```
saludar(“carlos”,”caseres”,30)
```

¿Cómo debería ser el código para que esta función muestre “Hola carlos Caseres, su edad es de 30 años”?

Funciones que muestran y funciones que devuelven

Los ejemplos anteriores se caracterizan por ser funciones que muestran, dado que en su código interno incluyen (por ejemplo) sentencias print para mostrar algo en pantalla

En otros casos, una función puede ser creada para relizar una ciert operación o implementar un algoritmo, que debe generar un resultado en particular y no necesariamente ser visualizado en pantalla.

Ese resultado debe poder ser almacenado en una variable o combinado con otras operaciones

En otros lenguajes de programación son conocidos como, procedimientos y funciones, pero aquí en Python son funciones.

Funciones que muestran y funciones que devuelven

Para estos casos, en el código de la función se debe incluir la sentencia return.
Ejemplo

```
def  sumar(x,y,z):
        resultado = x+y+z
        return resultado
```

el resultado de la función se almacena en una variable

```
total = sumar(10,20,30)
```

el resultado de la función se utiliza como parte de otro cálculo

```
promedio = sumar(10,20,30)/3
```

Parámetros

Si definimos una función con un determinado numero de parámetros, debe ser utilizada con esa misma cantidad, de lo contrario se producirá un error.

```
def  sumar(x,y,z):
        resultado = x+y+z
        return resultado
```

error: falta un parámetro

```
total = sumar(10,20)
```

error: hay dos parámetros de mas

```
promedio = sumar(10,20,30,40,50)/3
```

Parámetros

Siguiendo el ejemplo, si creamos una función para sumar números, seguramente desearíamos que funcione para dos valores, para tres valores, o para cualquier cantidad.

¿Cómo podemos abordar esto en Python?

Parámetros

Siguiendo el ejemplo, si creamos una función para sumar números, seguramente desearíamos que funcione para dos valores, para tres valores, o para cualquier cantidad.

Esto se puede lograr en Python incorporando un asterisco en los parámetros, con lo cual indica que la cantidad no se conoce de antemano.

De esta manera la función recibirá una tupla como argumento, y cada valor se deberá acceder mediante un índice.

Importante: al usar índices, se generara un error si queremos acceder a una posición mayor al numero de argumentos utilizados en la invocación.

Parámetros

Ejemplo

```
def  sumar(*valores):
        resultado = valores[0]+ valores[1] valores[2]
        return resultado
```

```
# calcula la suma 10+20+30, ignora los demás valores.
total = sumar(10,20,30,40,50)

# error: no se podrá acceder al índice 2.
total = sumar(10,20)
```

Parámetros

Ejemplo

```
def  sumar(*valores):
        resultado = 0
        for valor in valores:
            resultado = resultado + valor
        return resultado

 # calcula la suma 10+20+30+40+50.
total = sumar(10,20,30,40,50)

 # calcula la suma 10+20.
total = sumar(10,20)
```

Parámetros

Cuando empleamos mas de un parámetro, la invocación debe respetar el orden en el que se definen.

```
def  saludar(nombre, edad):
        print(f“hola {nombre}. Su edad es {edad}”)
```

```
# imprime “hola Juan. Su edad es 25
saludar( “Juan”, 25)

# imprime “hola 25. Su edad es Juan”
saludar(25,”juan”)
```

Aunque en estos casos no genere un error en el programa, el funcionamiento no es el esperado. Enornces ¿Cómo nos independizamos del orden de los parámetros?

Parámetros

Para independizarnos del orden de los parámetros, podemos incluir claves para cada uno de ellos.

```
def  saludar(nombre, edad):
        print(f“hola {nombre}. Su edad es {edad}”)
```

En la invocación, indicamos a que parámetro corresponde cada valor

```
# imprime “hola juan. Su edad es 25
saludar(nombre = “juan”, edad = 25)

# imprime “hola juan. Su edad es 25
saludar(edad = 23, nombre = “juan”)
```

Aunque cambiemos el orden, en ambos casos indicamos correctamente cual es el valor que corresponde al nombre y cual corresponde al de edad.

Parámetros

Si no se conoce a cantidad de claves que serán utilizadas en la invocación, pueden agregarse dos asteriscos al nombre del parámetro.

```
def  saludar(**datos):
        print("hola { datos.get("nombre")}. Su edad es {datos.get("edad")}")
```

En la invocación, indicamos a que parámetro corresponde cada valor

```
# imprime "hola juan. Su edad es 25"
saludar( nombre = "juan", edad = 25)

# imprime "hola juan. Su edad es 25", e ignora el valor del apellido.
saludar( edad = 25, edad = "juan", apellido = "garcia")

# error: no contiene el valor de la edad. Aunque usando el get solo retornaría None al consultar por la edad.
saludar( nombre = "juan", apellido = "garcia")
```

Parámetros

En algunas funciones, ciertos parámetros se pueden tomar un valor habitual, por lo que es útil la opción de definir valores por defecto.

Para hacer esto, colocamos el nombre del parámetro y le asignamos el valor por defecto correspondiente.
Si un parámetro tiene un valor por defecto no es obligatorio asignarlo en la invocación.

De esta manera, el orden de los parámetros debe ser respetado, pero es posible realizar una invocación con menos parámetros (la cantidad total menos los que tienen valores por defecto)

Implementar una función que amplíe un valor recibido como parámetro, en un factor de escala con valor por defecto de 2.

Parámetros

Implementar una función que amplíe un valor recibido como parámetro, en un factor de escala con valor por defecto de 2.

```
def ampliar( valor, escala = 2):
        return valor*escala


x = ampliar(10)  # retorna 20
x = ampliar(3)   # retorna 6
x = ampliar(10,4) # retorna 40
x = ampliar(escala = 2, valor = 4)  # retorna 12

x = ampliar(escala = 5) # error, falta el argumento de valor
x = ampliar(10,4,6) #error, sobra un argumento
```

Retornos

Implementar una función que genere una lista con números del primero al ultimo.

```
def generarLista( primero, ultimo = 10):
        pass
```

```
x = generarLista(4)      # retorna [4,5,6,7,8,9,10]
x = generarLista(2,12)   # retorna [2,3,4,5,6,7,8,9,10,11,12]
x = generarLista(15)     # retorna [ ]
```

Retornos

Implementar una función que genere una lista con números del primero al ultimo.

```
def generarLista( primero, ultimo = 10):
        lista = []
        for i in range(primero, ultimo + 1):
             lista.append(i)
        return lista
```

```
x = generarLista(4)      # retorna [4,5,6,7,8,9,10]
x = generarLista(2,12)   # retorna [2,3,4,5,6,7,8,9,10,11,12]
x = generarLista(15)     # retorna [ ]
```

Retornos

¿Cómo podemos retornar mas de un resultado en la misma función?
Ejemplo, queremos implementar una función que extraiga el mayor y el menor de una lista.

```
def extraer(lista):
        pass
```

Retornos

Mediante el uso de tuplas, podemos lograr que una función retorne mas cantidad de resultados (empaquetados).

```
def extraer(lista):
        mayor = max(lista)
        menor = min(lista)
        return (menor, mayor)


x = extraer([5,10,15,20])   # retorna (5,20)
p, u = extraer([5,10,15,20])  # retorna p = 5, u = 20


x = extraer([])  # error: la lista esta vacía y no puede obtener un máximo o un mínimo.
```

Retornos

Incluso es posible generar resultados de distinto tipo.

```
def detallar(lista):
        cantidad = max(lista)
        primerElemento = lista[0]
        return (cantidad, primerElemento)


x = detallar([5,10,15,20])  # retorna (4, 5)
p, u = extraer([5,10,15,20])  # retorna p = 4, u = 5
p, u = extraer(["a","b","c"]) # retorna p = 3, u = "a"
```