GUIA 3. FUNCIONES Y MÓDULOS

Funciones

Una función es un bloque de código reutilizable usado para desarrollar o ejecutar una acción específica (sumar, restar, obtenerNombre, etc.). Las funciones proveen una mejor modularización de la aplicación y un alto grado de reutilización de código.

Python ofrece varias funciones en sus librerías como la función print(), para imprimir mensajes en consola. Sin embargo podemos crear nuestras propias funciones.

Creación de Funciones

Debemos considerar algunas reglas

- La definición de una función inicia con la palabra clave ***def*** seguido del nombre de la función y paréntesis ().
- Los parámetros de entrada de la función se definen dentro de los paréntesis.
- El primer parámetro de una función puede ser opcional.
- El bloque de código inicia con dos puntos (:) y es identado.
- La sentencia ***return VALOR*** finaliza la función y puede retornar un valor (return es lo mismo que return None)
- Los parámetros de la función tienen comportamiento posicional, así que debe tener consideración cuando envíe los datos en el orden requerido.

Sintaxis:

```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

El docstring de la línea de la sintaxis corresponde a la documentación de lo que hace la función.

Creemos una función que reciba un cadena de texto y la imprima, no se olvide que los archivos donde cree el código debe tener extensión ***.py***:

```
def imprimirtexto( str ):
   "Imprime el texto que llega como parámetro"
   print(str)
   return
```

Analice cada una de las partes de la función que cumple cada una de las reglas mencionadas anteriormente.

¿Cómo llamamos o usamos esta función?

Podemos llamarla desde otra función o directamente desde la consola de Python:

```
def imprimirtexto( str ):
   "Imprime el texto que llega como parámetro"
   print(str)
   return

imprimirtexto("hola")
```

Ejecute este código y compruebe su funcionamiento.

Paso por referencia.

En Python los parámetros enviados son pasados por referencia (¿Qué significa pasar por referencia o pasar por valor?). Pasar por referencia significa que si declaramos una variable y esta es usada para enviarse como parámetro cuando invocamos la función, si el parámetro es cambiado dentro de la misma, entonces el valor de la variable cambia.

Realice el siguiente ejemplo y observe si cambia o no la variable usada como parámetro en la función:

```
def changlist( mylist ):
   "Cambia una lista con nuevos valores"
   mylist.append([1,2,3,4]);
   print("Valores dentro de la función: ", mylist)
   return

mylist = [10,20,30];
changlist( mylist );
print("Valores fuera de la función: ", mylist)
```

Argumentos de Función

Existen varios tipos de argumentos en una función:

- Argumentos requeridos: Los que se pasan en un orden posicional y cuyo número son los mismos que los que define la función.
  Qué pasa si ejecuta el siguiente código:

```
def imprimirtexto( str ):
   "Imprime el texto que llega como parámetro"
   print(str)
   return

imprimirtexto()
```

- Argumentos de palabra clave: En este caso, el que invoca la función identifica el argumento por el nombre del parámetro. De esta forma, se puede ignorar el orden de los parámetros.
  Analice el siguiente ejemplo y pruébelo, observe el orden de los argumentos y como se hace para que se utilicen apropiadamente

```
def printinfo( name, age ):
   "Imprime variables de esta función"
   print("Name: ", name)
   print("Age ", age)
   return;

# Now you can call printinfo function
printinfo( age=10, name="michael" )
```

- Argumentos por defecto: Son argumentos que pueden tomar un valor por defecto cuando no se envía ningún valor. Analice el siguiente ejemplo y pruébelo:

```
def printinfo( name, age=100 ):
   "Imprime variables de esta función"
   print("Name: ", name)
   print("Age ", age)
   return;

# Now you can call printinfo function
printinfo( name="michael" )
```

- Argumentos de longitud variable: Útiles cuando no se saben cuántos argumentos se van a usar en la función. Se usa un * antes de la variable para indicar este tipo de argumentos, observe y pruebe el siguiente ejemplo:

```
def printinfo( arg1, *varios ):
   "Imprimir parámetros enviados y usar argumentos de longitud de variable"
   print("Resultado es : ")
   print(arg1)
   for var in varios:
      print("var: " ,var)
   return;

printinfo( 10 )
printinfo( 70, 60, 50)
```

Sentencia return

Esta sentencia finaliza una función y puede retornar un valor a quien invoca la función. Los ejemplos vistos hasta ahora no retornan valores. Miremos un ejemplo para retornar un valor:

```
def getnombrecompleto( name="James", lastname="Rodriguez" ):
   "Retorna nombre completo"
   nombre=name + " " + lastname
   return nombre;

print ( getnombrecompleto( name="michael",lastname="cruz" ) )
```

Variables locales y globales

Son variables locales las que se definen dentro de una función y pueden ser accedidas solo en la función, mientras que las globales se definen fuera de las funciones para que puedan ser accedidas desde cualquier punto.

Hagamos un cambio al ejemplo anterior definiendo la variable nombrecompleto global, analice lo que sucede:

```
nombre="nombre por defecto"
def getnombrecompleto( name="James", lastname="Rodriguez" ):
   "Retorna nombre completo"
   nombre=name + " " + lastname
   print("local variable ", nombre)
   return nombre;
getnombrecompleto( name="michael",lastname="cruz" )
print ("global", nombre )
```

Módulos

Un módulo permite organizar lógicamente el código en Python, podemos agrupar nuestro código para entenderlo mejor. Un módulo no es más que un archivo python que define funciones, clases (que veremos adelante) y variables, incluso puede incluir código ejecutable.

Podemos tomar los ejemplos creados anteriormente en guardarlos en un archivo ejemplo.py este sería nuestro módulo. Cómo hacemos para utilizarlo?:

Importar Módulos

Se puede importar los módulos con la sintaxis:

```
import module1[, module2[,... moduleN]
```

Hagamos un ejemplo:

Cree un script modulo1.py con la siguiente función:

moduloprincipal.py | modulo1.py

```
1 def print_func( par ):
2     print("Hello : ", par)
3     return
4
```

Cree un segundo módulo moduloprincipal.py **en el mismo directorio** e importe el modulo1.py, llame la función print_func:

moduloprincipal.py

```
import modulo1

modulo1.print_func("hola")

```

Ejecute el script moduloprincipal.py y analice los resultados.

from … import

Con from podemos importar atributos específicos del módulo sin necesidad de importar todo. La sintaxis es la siguiente:

```
from modname import name1[, name2[, ... nameN]]
```

Para el caso anterior, sería:

moduloprincipal.py

```
from modulo1 import print_func

print_func("hola")

```

Función dir()

Esta función obtiene una lista de los nombres de atributos definidos en un módulo (funciones y variable). Pruebe el siguiente ejemplo del módulo math:

```
import math

content = dir(math)
print("funciones y atributos del módulo math")
print(content)
```