

Modularidad

Funciones y módulos

Jonatan Gómez Perdomo, Ph. D.

jgomezpe@unal.edu.co

Arles Rodríguez, Ph.D.

aerodriguezp@unal.edu.co

Camilo Cubides, Ph.D. (c)

eccubidesg@unal.edu.co

Carlos Andrés Sierra, M.Sc.

casierrav@unal.edu.co

Research Group on Artificial Life – Grupo de investigación en vida artificial – (Alife)

Computer and System Department

Engineering School

Universidad Nacional de Colombia

Agenda

- 1 Funciones
- 2 Paso de parámetros en python
- 3 Alcance de variables
- 4 Módulos en Python
- 5 Paquetes en python



Funciones

Son unidades pequeñas dentro de una construcción más grande que genera una salida esperada. Si se piensa en un televisor, cada botón en el control remoto cumple una función (subir volumen, bajar volumen, cambiar el canal, mutar, cambiar la fuente, ingresar al menú, etc.)

Las funciones tienen dos categorías:

- Funciones incluidas (len, append, pop, etc.).
- Funciones definidas por el usuario

Nosotros hemos visto diferentes formas de crear funciones:

- Funciones sin argumentos
- Funciones con argumentos y sin retorno
- Funciones con argumentos y retorno



Funciones sin argumentos

Ejemplo

```
# función sin argumentos

def saludar():
    nombre = input()
    print("Hola", nombre)

saludar()
```



La salida del programa anterior para una entrada Zeus es:

Hola Zeus

Funciones con argumentos y sin valor de retorno

Ejemplo

```
def saludar(nombre):  
    print('Hola', nombre)  
  
nombre = input()  
saludar(nombre)
```



La salida del programa anterior para una entrada 'Titán' es:

Hola Titán



Funciones con argumentos y con valor de retorno

Ejemplo

```
def saludar(nombre):  
    return 'Hola ' + nombre  
  
nombre = input()  
print(saludar(nombre))
```



La salida del programa anterior para una entrada 'Atenea' es:

```
Hola Atenea
```



Otros tipos de Función

Existen otras formas alternativas para definir funciones:

- Funciones con argumentos por defecto el argumento que la variable toma el valor dado en la definición
- Funciones con un número de argumentos variable que los tome como una tupla adicional
- Funciones con un número de argumentos variable que los tome como un diccionario



Funciones con argumentos por defecto

Ejemplo

La variable base toma el valor 2 por defecto

```
def log_entero(num, base=2):  
    cont = 0  
    while num >= base:  
        cont+=1  
        num /= base  
    return cont  
print(log_entero(1024))  
print(log_entero(1000,10))  
print(log_entero(9,3))
```



La salida del programa es:

```
10  
3  
2
```


Funciones con número de argumentos variable: Tupla

Se utiliza **nombre_var* quedando *nombre_var* como tupla.

Ejemplo

```
def variable_argument(var1, *vari):  
    print('salida:' + str(var1))  
    for var in vari:  
        print(var)  
  
variable_argument(60)  
variable_argument(100, 90, 67, 23, 10)
```



La salida del programa es:

```
salida:60  
salida:100  
90  
67  
23  
10
```

Funciones con número de argumentos variable: Diccionario

Es posible enviar parejas llave=valor como argumentos a una función con ****nombre_var**, la variable *nombre_var* queda definida como diccionario.

Ejemplo

```
from pprint import pprint
def informar(**var):
    pprint(var)
    for key, value in var.items():
        pprint("%s == %s" %(key,value))

informar(nombre="Poseidón",edad=6000,ciudad="Olimpo")
```



La salida del programa es:

```
{'ciudad': 'Olimpo', 'edad': 6000, 'nombre': 'Poseidón'}
'nombre == Poseidón'
'edad == 6000'
'ciudad == Olimpo'
```

Agenda

- 1 Funciones
- 2 Paso de parámetros en python**
- 3 Alcance de variables
- 4 Módulos en Python
- 5 Paquetes en python



Paso de parámetros en python (I)

El paso de parámetros a funciones se realiza por referencia: los objetos que lleguen pueden ser modificados dentro de la función, mientras que a la variable no se le asigne un nuevo valor ([click aquí para ver ejecución](#)).

Ejemplo

```
def unir_listas(lista1, lista2):  
    lista1.extend(lista2)  
  
avengers = ['Tony', 'Natalia', 'Steve']  
nuevos_avengers = ['Thor', 'Peter']  
  
unir_listas(avengers, nuevos_avengers)  
print(avengers)
```



La salida del programa es:

```
['Tony', 'Natalia', 'Steve', 'Thor', 'Peter']
```

Paso de parámetros en python (II)

En el siguiente ejemplo, el valor de la variable `a` del programa principal no se afecta, mientras que el valor de la variable `a` en la función si cambia: (click aquí para ver paso a paso):

Ejemplo

```
def func(a):  
    a *= 10  
    print('En la función a=',a)  
  
a = 45  
func(a)  
print('En el programa principal a=',a)
```



La salida del programa es:

```
En la función a=450  
En el programa principal a=45
```

Paso de parámetros en python (III)

En el siguiente ejemplo, el valor de avengers del programa principal no cambia debido a la asignación que se está realizando dentro de la función a la variable lista (click aquí para ver paso a paso):

Ejemplo

```
def no_limpia_lista(lista):  
    lista = []  
  
    avengers = ['Tony', 'Natalia', 'Steve']  
    no_limpia_lista(avengers)  
    print(avengers)
```



La salida del programa es:

```
['Tony', 'Natalia', 'Steve']
```

Agenda

- 1 Funciones
- 2 Paso de parámetros en python
- 3 Alcance de variables**
- 4 Módulos en Python
- 5 Paquetes en python



Alcance de variables

En python hay dos tipos básicos de alcance de las variables:

- Variables locales
- Variables globales

Para entender este concepto, se usarán los decretos expedidos por gobiernos locales y nacionales como símil. Un decreto expedido por la alcaldía de Bogotá tiene alcance sólo para los habitantes de la capital pero no para los del resto del país. Un decreto expedido por la presidencia de Colombia tiene alcance en el país (incluida Bogotá) pero no en otros países.



Alcance de variables (II)

En el siguiente programa, la variable `a` definida en la función `func` es local y solo tiene alcance en dicha función. No tiene nada que ver con la variable `a` que es definida después en el cuerpo principal del programa (click aquí para analizar el código):

Ejemplo

```
def func():  
    a=12  
    print('Variable local:', a)  
a=10  
func()  
print ('Variable del cuerpo principal:',a)
```



La salida del programa es:

```
Variable local:12  
Variable del cuerpo principal:10
```

Alcance de variables (III)

En el siguiente programa, `k` es una variable global (alcance en todo el programa), `lista` es una variable local (de `main`) con alcance en la función `main` y 'global' en `add`, mientras que la variable `x` es local pues solo tiene alcance en la función `add`. (click aquí para analizar el código):

Ejemplo

```
k = 4
def main():
    lista = []
    def add():
        for x in range(k):
            lista.append(x)
        print(lista)
    add()
main()
```



La palabra reservada global

Cuando se usa la palabra `global` antes de una variable, dentro de una función, se indica que dicha variable tiene alcance global y que cualquier operación que le hagamos dentro de la función puede modificar el valor de la variable global (click aquí para analizar el código):

Ejemplo

```
k=5

def func():
    global k
    k=k+7
    print("La variable k tiene alcance global:",k)
    k=10

func()
print ("Valor de la variable global k fuera de la función:",k)
```



La palabra reservada global (II)

Ejemplo (continuación)

La salida del código anterior es:

```
La variable k tiene alcance global: 12  
Valor de la variable global k fuera de la función: 10
```



La palabra reservada global (III)

De esta manera, con la palabra `global` se puede cambiar el valor de una variable global dentro de una función:

Ejemplo

```
x = "sorprendente"

def myfunc():
    global x
    x = "fantástico"

print("Antes Python era " + x)
myfunc()
print("Ahora Python es " + x)
```



La palabra reservada global (IV)

Ejemplo (continuación)

La salida de este programa es:

```
Antes Python era sorprendente  
Ahora Python es fantástico
```



Agenda

- 1 Funciones
- 2 Paso de parámetros en python
- 3 Alcance de variables
- 4 Módulos en Python
- 5 Paquetes en python

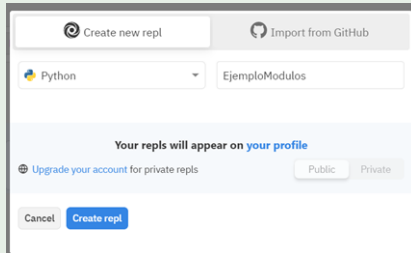


Módulos en Python (I)

Un módulo en Python es un archivo fuente con funciones y definiciones que pueden ser usadas en otro archivo mediante la palabra `import`.

Ejemplo

Este ejemplo se trabajará en <https://replit.com/>. Se debe crear un repositorio con el nombre deseado, en este ejemplo EjemploModulos:



Create new repl Import from GitHub

Python EjemploModulos

Your repls will appear on [your profile](#)

Upgrade your account for private repls Public Private

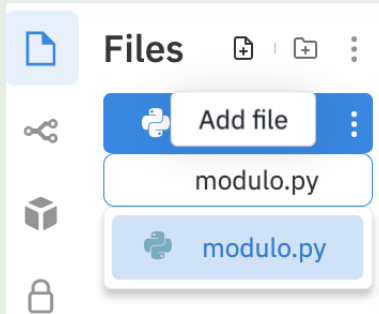
Cancel Create repl



Módulos en Python (II)

Ejemplo (continuación)

Al crear el repositorio se crea un archivo `main.py` donde se espera este el programa principal. El módulo será otro archivo con el nombre deseado, en este ejemplo `modulo.py`:



Módulos en Python (III)

Ejemplo (continuación)

En dicho archivo (`modulo.py`) se escriben las funciones y definiciones propias del módulo, en este caso se agregarán las siguientes:

```
def sum1(a,b):  
    c = a+b  
    return c  
  
def mul1(a,b):  
    c = a*b  
    return c
```



Módulos en Python (IV)

Para usar un módulo en otro archivo se utiliza `import modulo` donde `modulo` es el nombre del módulo a usar. Se llaman las funciones anteceditas por el nombre de módulo

Ejemplo (continuación)

Se modifica el archivo `main.py` para que use el módulo `modulo`.

```
import modulo

x = 12
y = 34
print ("La suma es ", modulo.sum1(x,y))
print ("La multiplicación es ", modulo.mul1(x,y))
```



Módulos en Python (V)

Es posible importar un módulo y usarlo con otro nombre

Ejemplo (continuación)

El archivo `main.py` se modifica para que use el módulo `modulo` con el nombre `md`:

```
import modulo as md

x = 12
y = 34
print ("La suma es ", md.sum1(x,y))
print ("La multiplicación es ", md.mul1(x,y))
```



Módulos en Python (VI)

Es posible importar únicamente una función específica de un módulo.

Ejemplo (continuación)

Se modifica el archivo `main.py` para que del módulo `modulo` importe la función `sum1`:

```
from modulo import sum1  
x = 12  
y = 34  
  
print ("La suma es ", sum1(x,y))
```



Módulos en Python (VII)

Para saber los métodos que tiene un módulo determinado se puede utilizar la función `dir`.

Ejemplo (continuación)

Se modifica el archivo `main.py` para usar la función `dir` con el módulo `modulo`:

```
import modulo  
print ("Las funciones de modulo son:", dir(modulo))
```



Localización de los módulos de Python

Cuando se utiliza import, python buscará el módulo en la siguiente forma:

- El directorio actual
- La variable de entorno PYTHONPATH

Para ver información sobre las rutas usadas por Python para import:

```
import sys
for p in sys.path:
    print(p)
```



La salida de este programa es:

```
/home/runner/EjemploModulos
/opt/virtualenvs/python3/lib/python3.8/site-packages
/usr/lib/python3.8.zip
/usr/lib/python3.8
/usr/lib/python3.8/lib-dynload
```



Agenda

- 1 Funciones
- 2 Paso de parámetros en python
- 3 Alcance de variables
- 4 Módulos en Python
- 5 Paquetes en python**

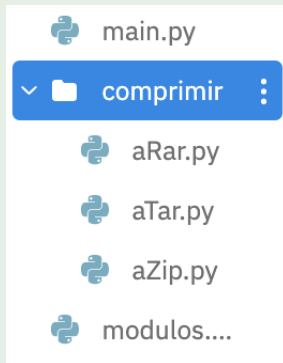


Definición de un paquete en Python (I)

Un paquete se define como un conjunto de módulos organizados, preferiblemente, en una sola carpeta, que se podrán importar en Python.

Ejemplo

Se creará una carpeta llamada `comprimir` con 3 archivos:



El contenido de cada archivo es el siguiente:

Definición de un paquete en Python (II)

Ejemplo (continuación)

① aZip.py

```
def crear_zip(archivo):  
    print('creando...' + archivo + '.zip')
```

② aRar.py

```
def crear_rar(archivo):  
    print('creando...' + archivo + '.rar')
```

③ aTar.py

```
def crear_tar(archivo):  
    print('creando...' + archivo + '.tar')
```

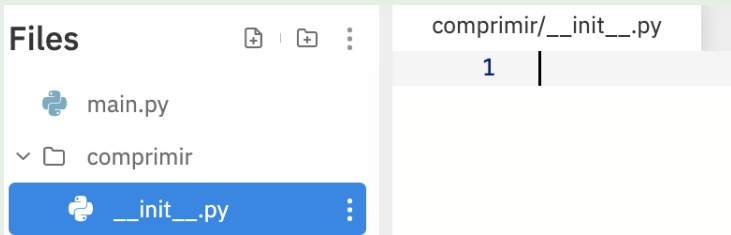


El archivo `__init__.py`

Para que una carpeta quede definida como un paquete, se le debe agregar el archivo vacío `__init__.py`:

Ejemplo (continuación)

Se agrega el archivo vacío `__init__.py` a la carpeta `comprimir`



Usando el paquete

Para usar un paquete se debe importar el módulo `sys` y agregar el directorio del paquete creado con `sys.path.append` e importar cada módulo con `from`.

Ejemplo (continuación)

Se edita `main.py` para incluir el paquete `comprimir` para usarlo.

```
from comprimir import aZip
from comprimir import aRar
from comprimir import aTar

aZip.crear_zip("archivo")
aRar.crear_rar("archivo")
aTar.crear_tar("archivo")
```



Referencias

- Das, B. N. (2017). Learn Python in 7 Days. Packt Publishing Ltd.
- <https://www.dummies.com/programming/python/how-to-find-path-information-in-python/>
- Rodríguez, A (2020). Curso de Programación en Python.
<https://github.com/arleserp/cursopython>

