Universidad del Rosario | Educación Continua

INTRODUCCIÓN A LA PROGRAMACIÓN EN PYTHON

- Miguel Orjuela

Personas con *propósito* que ayudan a transformar vidas

Universidad del Rosario | Educación Continua

Introducción a la programación en Python

2019

Sesión # 4

Funciones

Miguel Angel Orjuela Rocha

Ingeniero de Sistemas y Computación

Contenido

- ¿Qué es una función?
- Namespaces, Scope, Funciones locales
- Retornar múltiples valores
- Las funciones son objetos
- Funciones anónimas
- Curryin
- Generadores
- Errores y manejo de excepciones

¿Qué es una función?

Funciones

- Método principal y más importante de organizar código y poder reutilizarlo
- Si en ocasiones es necesario repetir más de una vez la ejecución de un mismo bloque de código o allguno muy similar, vale la pena hacer una función reutilizable
- Las funciones hacen el código más leíble
- Una función asigna un nombre a un grupo de declaraciones de Python

Funciones

Se declaran con la palabra def y retornan resultados con la palabra clave return

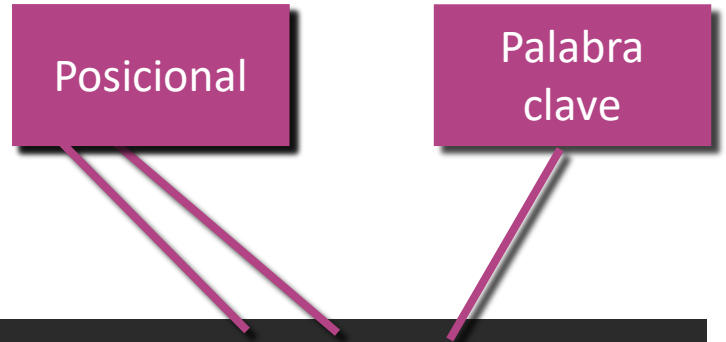
```
def my_function(x, y, z= 1.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

Si Python llega al final de una función y no encuentra un return, retorna None

Funciones

Cada función puede tener argumentos posicionales y de palabra clave

Los argumentos de palabra clave se usan para especificar valores por defecto o valores opcionales




```
def my_function(x, y, z= 1.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

```
my_function(5, 6, z = 0.7)
my_function(3.14, 7, 3.5)
my_function(10, 20)
```

Algunas formas de llamar a la función

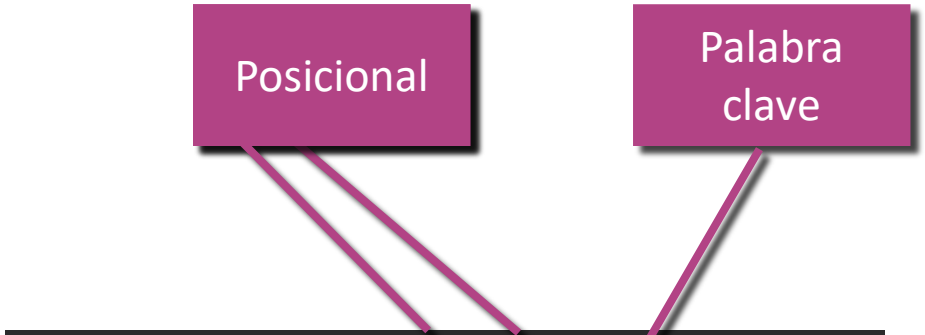
Funciones

Cuando se llama la función, los argumentos posicionales deben ir primero, y luego los de palabra clave

Los argumentos de palabra clave pueden ir en cualquier orden

```
my_function(x=5, y=6, z=7)
my_function(y=6, x=5, z=7)
```

```
def my_function(x, y, z= 1.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```



Algunas formas de llamar a la función

Namespaces, Scope, Funciones locales

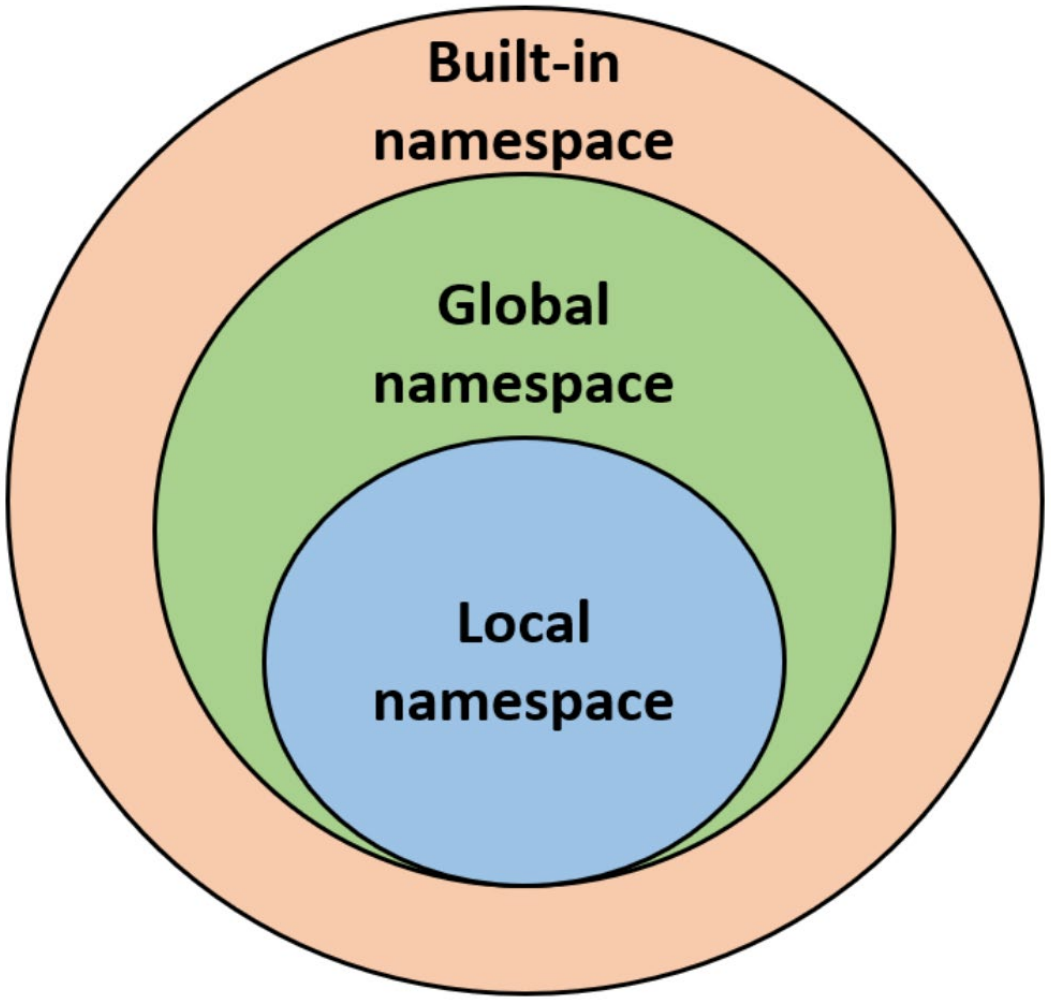
Namespaces, Scope, Funciones locales

Las funciones pueden acceder a las variables en dos diferentes alcances (scopes): **global** y **local**

Namespace: Espacio de nombres

Una variable asignada dentro de una función estará en el namespace local

Una vez se ejecuta la función, el namespace local es destruído



Namespaces, Scope, Funciones locales

```
def func():
    a = []
    for i in range(5):
        a.append(i)
```

Cuando se ejecuta la función func(), se crea una lista a y se le agregan cinco elementos. Cuando la función termina, el namespace local es destruído

```
a = []
def func():
    for i in range(5):
        a.append(i)
```

En este caso, a no toma los valores que se le agregan, pues la variable está fuera del alcance de la función

```
a = None
def cambiar_a():
    global a
    a = []
cambiar_a()
```

Asignar variables fuera del alcance de una función es posible, pero las variables deben ser declaradas con la palabra clave global

Retornar múltiples valores

Retornar múltiples valores

Característica especial de Python

```
def f():
    a = 5
    b = 6
    c = 7
    return a, b, c
```

```
x,y,z = f()
```

```
valor_retorno = f()
```

Muy usado en aplicaciones científicas

También puede retornar diccionarios

```
def f():
    a = 5
    b = 6
    c = 7
    return {'a': a, 'b': b, 'c': c}
```

Las funciones son objetos

Las funciones son objetos

Python permite expresar algunas operaciones que son difíciles de definir en otros lenguajes

Ejemplo: Limpieza de datos

```
ciudades = ['  Bogota','BucaramANGa', 'bogota','Bogota!',
            'MEDELLIN','santa marta##', 'bogota?']
```

```
import re

def limpiar_strings(strings):
    resultado = []
    for valor in strings:
        valor = valor.strip()
        valor = re.sub('[!#?]', '', valor)
        valor = valor.title()
        resultado.append(valor)
    return resultado
```

```
limpiar_strings(ciudades)
```

Las funciones son objetos

Forma alternativa, haciendo una lista de operaciones

```
def remove_punctuation(valor):
    return re.sub('[!#?]', '', valor)

operaciones= [str.strip, remove_punctuation, str.title]

def limpiar_strings_v2(strings, ops):
    resultado = []
    for valor in strings:
        for operacion in ops:
            valor = operacion(valor)
        resultado.append(valor)
    return resultado
```

```
limpiar_strings_v2(ciudades, operaciones)
```

Las funciones son objetos

Las funciones se pueden pasar como parámetro en otras funciones como map(), la cual aplica una función a una secuencia de algún tipo

```
for x in map(remove_punctuation, ciudades):
    print(x)
```

```
list(map(remove_punctuation, ciudades))
```

Funciones anónimas

Funciones anónimas

También conocidas como funciones lambda

Forma de escribir funciones en una sola declaración

El resultado de la declaración es el retorno de la función

Se definen con la palabra clave lambda

```
def duplicador(x):
    return x * 2

duplicador(3)
```

```
duplicador_anon = lambda x: x*2

duplicador_anon(3)
```

Se usan en análisis de datos para pasar funciones como argumentos

Funciones anónimas

Ejemplo: Crear una función que reciba como parámetros una lista y una función, y retorne unja nueva lista con los resultados de aplicar la función parámetro a la lista parámetro

```
enteros = [4, 0, 3, 6, 7]
```

```
def aplicar_a_lista(alguna_lista, f):
    return [f(x) for x in alguna_lista]
```

```
aplicar_a_lista(enteros, lambda x: x * 3)
```

Funciones anónimas

Ejemplo: Usar el parámetro key de la función sort

```
strings = ['papa', 'perro', 'paleta', 'pelado', 'papilla']
```

```
strings.sort(key = lambda x : len(set(list(x))))
```

Currying

Currying

Currying: Derivar nuevas funciones a partir de existentes usando aplicación de argumentos parciales

```
def add_numbers(x, y):
    return x + y
```

```
add_five = lambda y: add_numbers(5, y)
```

```
add_five(3)
```

```
from functools import partial
add_five_v2 = partial(add_numbers, 5)
```

Generadores

Generadores

Python tiene una forma consistente de iterar sobre secuencias (objetos en una lista, o líneas en un archivo)

Protocolo iterador: forma genérica de hacer los objetos iterables

Ejemplo: Diccionario

```
un_diccionario = {'a': 1, 'b': 2, 'c': 3}

for llave in un_diccionario:
    print(llave)
```

Cuado escribimos for llave in un_diccionario, el intérprete de Python crea un iterador sobre el diccionario

```
iter_diccionario = iter(un_diccionario)
```

```
iter_diccionario
```

```
list(iter_diccionario)
```

Generadores

Iterador: Cualquier objeto que entregará objetos al interprete de Python en contextos como ciclos for

- No se puede recorrer más de una vez
- Sus elementos no están almacenados en memoria, se generan “on the fly”

Un generador es una forma concisa de construir un objeto iterable. Para crearlos se usa la palabra clave yield

```
def cuadrados(n=10):
    print('Generando cuadrados de 1 hasta {0}'.format(n ** 2))
    for i in range(1, n + 1):
        yield i ** 2
```

```
generador = cuadrados()
```

```
for x in generador:
    print(x, end=' ')
```

Generadores

Expresión generadora: Es similar a una lista, diccionario o conjunto por comprensión (pero con paréntesis)

```
generador_v2 = (x ** 2 for x in range(100))
list(generador_v2)
```

```
def _hacer_generador():
    for x in range(100):
        yield x ** 2
generador_v1 = _hacer_generador()
list(generador_v1)
```

Pueden ser usadas como argumentos de funciones en vez de listas

```
sum(x ** 2 for x in range(100))
```

```
dict((i, i **2) for i in range(5))
```

Generadores

Módulo itertools `import itertools`

Colección de generadores para muchos algoritmos de datos

Ejemplo 1: groupby – función que toma una secuencia y una función, y agrupa elementos consecutivos de la secuencia de acuerdo al valor que retorna la función

```
primer_letra = lambda x: x[0]

nombres = ['Alan', 'Adriana', 'Bernardo', 'Carmen', 'Alberto', 'Brayan']

for letras, nombres in itertools.groupby(nombres, primer_letra):
    print(letras, list(nombres)) # nombres es un iterador
```

Generadores

Módulo itertools

```
import itertools
```

Colección de generadores para muchos algoritmos de datos

Ejemplo 2: cycle – ¿Qué hace la función cycle()?

```
ciclo = itertools.cycle('ABCD')

for i in ciclo:
    print(i)
    time.sleep(1)
```

Errores y manejo de excepciones

Errores y manejo de excepciones

Manejar errores da robustez a los programas

Problema: Los errores hacen terminar el programa

```
float('1.2345')
```

```
float('algo')
```

```
In[18]: float('algo')
Traceback (most recent call last):
  File "C:\Users\maorj\Anaconda3\lib\site-packages\IPython\core
 \interactiveshell.py", line 2910, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-18-037dd05fe7bb>", line 1, in <module>
    float('algo')
ValueError: could not convert string to float: 'algo'
```

El tipo de error

Errores y manejo de excepciones

Manejar errores da robustez a los programas

Solución: try + except en bloques que pueden arrojar error

```
float('algo')
```

```
In[18]: float('algo')
Traceback (most recent call last):
  File "C:\Users\maorj\Anaconda3\lib\site-packages\IPython\core
 \interactiveshell.py", line 2910, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-18-037dd05fe7bb>", line 1, in <module>
    float('algo')
ValueError: could not convert string to float: 'algo'
```

```
def intentar_float(x):
    try:
        return float(x)
    except:
        return x
```

```
intentar_float('4.20')
intentar_float('algo')
```

Errores y manejo de excepciones

Manejar errores da robustez a los programas

Solución: try + except en bloques que pueden arrojar error

```
float('algo')
```

```
intentar_float((1, 2))
```

```
In[18]: float('algo')
Traceback (most recent call last):
  File "C:\Users\maorj\Anaconda3\lib\site-packages\IPython\core
 \interactiveshell.py", line 2910, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-18-037dd05fe7bb>", line 1, in <module>
    float('algo')
ValueError: could not convert string to float: 'algo'
```

El tipo de error

```
def intentar_float(x):
    try:
        return float(x)
    except ValueError:
        return x
```

```
intentar_float('4.20')
intentar_float('algo')
```

Resumen

- Función: Método principal y más importante de organizar código y poder reutilizarlo. Reciben argumentos. Retornan información. Tienen acceso a un conjunto de variables de su alcance. Una función es un objeto
- Hay funciones anónimas/lambda. Con ellas se puede hacer currying
- Los generadores son eficientes con la memoria

A continuación

Archivos y el Sistema Operativo