Métodos de minería de datos en Python

Programación básica en Python

Contenido

1. Funciones
2. Expresiones lambda
3. Intro a Github

Funciones

Cómo programamos?

Hasta ahora:

1. Manejamos diferentes tipos de estructuras y operaciones
2. Sabemos escribir códigos para resolver problemas específicos
3. Cada archivo generado es una pieza del código
4. Cada código es una secuencia de instrucciones

Problemas con éste enfoque:

1. Fácil de usar en problemas de “mundo pequeño”
2. Desordenado para problemas de gran escala
3. Difícil realizar seguimiento a los detalles

¿Cómo se sabe si se suministra la información correcta en la parte correcta del script?

Buenas prácticas

Más código nos es necesariamente lo mejor

Los buenos programadores se miden por:

- La funcionalidad de sus códigos
- La introducción de funciones
- La utilización de mecanismos que permitan descomponer y abstraer la información

Estructuras con Descomposición

En programación el código se divide en módulos, los cuales:

- Son **autocontenidos**
- Se usan para **dividir el código**
- Se destinan a ser **reutilizados**
- Mantienen el código **organizado**
- Mantienen el código **coherente**

Una forma de descomoponer el código se realiza a través de funciones

Generar Abstracción

En programación, piense que cada pieza del código compone un juego de video:

- El usuario final **no puede ver** los detalles de programación
- El usuario final **no necesita ver** los detalles de programación
- El usuario final **no quiere ver** los detalles de programación

La abstracción del código se realiza por documentos de especificación de funciones

Funciones

- Una función es un bloque de código organizado y reutilizable que se utiliza para realizar una única acción relacionada.
- Las funciones se encuentran estáticas hasta que son **llamadas** o **invocadas** en un programa.
- Python ofrece muchas funciones integradas y la opción de crear funciones propias: funciones definidas por el usuario.

Funciones

Características:

1. Inicia con una **palabra clave**
2. Tienen un **nombre**
3. Tienen **parámetros** (0 o más)
4. Tienen **comentarios** (opcional pero deseable)
5. Tienen un **cuerpo de código**
6. **Retornan** algo, algún resultado.

Funciones

keyword

nombre

```
def par(n):
```

Parámetros/
Argumentos (Pueden ser obligatorios u opcionales)

```
## n: Un entero positivo
## Retorna True si n es par, False en caso contrario
```

Comentarios al código

```
print("El número es par?: ")
return n%2 == 0
```

Cuerpo del código

```
par(4)
```

Función invocada

Funciones: Cuerpo

```
def par(n):

    ## n: Un entero positivo
    ## Retorna True si n es par, False en caso contrario

    print("El número es par?: ")
    return n%2 == 0

par(4)
```

Corre algunas instrucciones

keyword

Expresión a evaluar y retornar

Funciones: Variable

- El **parámetro real** se vincula con el **parámetro formal** cuando se invoca la función
- Un nuevo **entorno** es creado cuando se invoca una nueva función
- El **alcance** es la asignación de nombres a objetos

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```

Parámetro formal

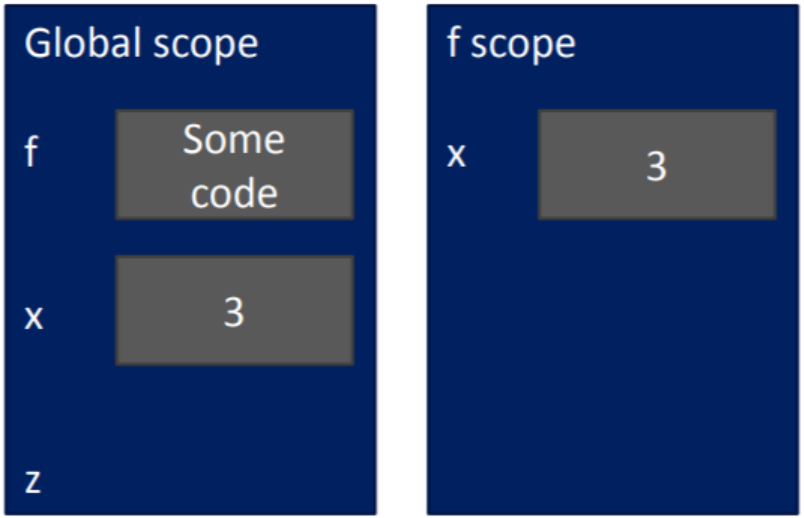
Parámetro actual

Definición de la función

Ejecuta la función:
- Inicializa la variable x
- Invoca la función f(x)
- Asigna el resultado de la función a la variable z

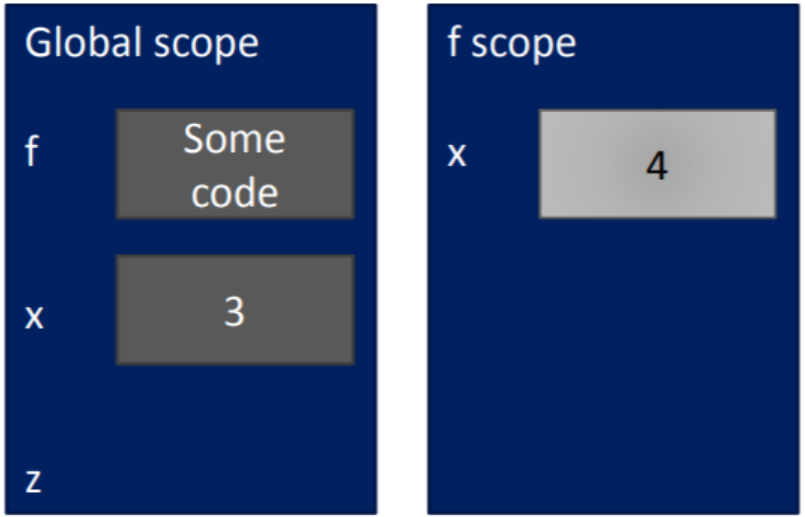
Funciones: Variable

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



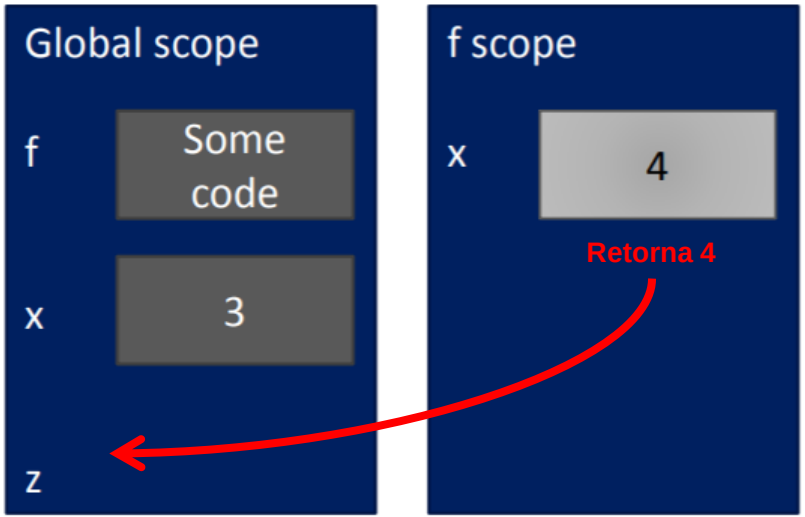
Funciones: Variable

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



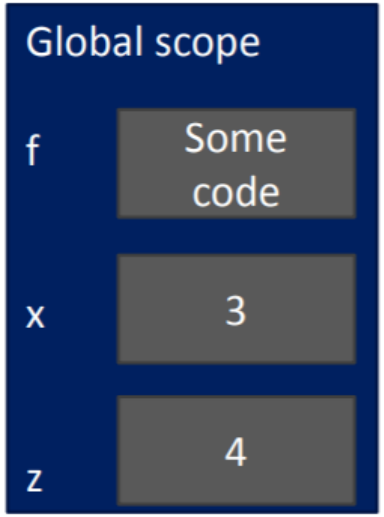
Funciones: Variable

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



Funciones: Variable

```
def f( x ):
    x = x + 1
    print('in f(x): x =', x)
    return x

x = 3
z = f( x )
```



Funciones: Variable

```
def conocer(nombre):
    """
    Esta función
    saluda a la persona
    que ingrese su nombre
    como parámetro
    """
    print("Hola, " + nombre + ". ¡Buenos días!")
```

```
conocer("Carlos")
```

```
Hola, Carlos. ¡Buenos días!
```

Funciones: Variable

```
def absolute_value(num):

    if num >= 0:
        return num
    else:
        return -num
```

```
print(absolute_value(2))
```

2

```
print(absolute_value(-4))
```

4

Funciones como argumentos

```
def conocer(nombre, mensaje = "Buenos días"):
    """
    Esta función
    saluda a la persona
    que ingrese su nombre
    como parámetro

    Si no se provee el mensaje,
    el valor por defecto será "Buenos días"
    """
    print("Hola", nombre + ', ' + mensaje)
```

```
conocer("María")
```

```
Hola María, Buenos días
```

```
conocer("Ana María", "¿Cómo estás?")
```

```
Hola Ana María, ¿Cómo estás?
```

Funciones como argumentos

```
def func_a():
    print('Función A')
def func_b(y):
    print('Función B')
    return y
def func_c(z):
    print('Función C')
    return z
print(func_a())
print(5 + func_b(2))
print(func_c(func_b(2)))
```

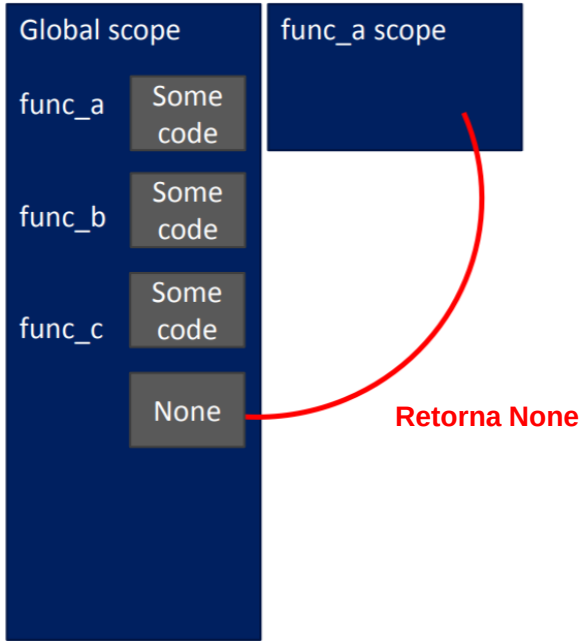
Llama a la función A: No toma argumentos

Llama a la función B: Toma un argumento

Llama a la función C: Toma un argumento, en este caso otra función

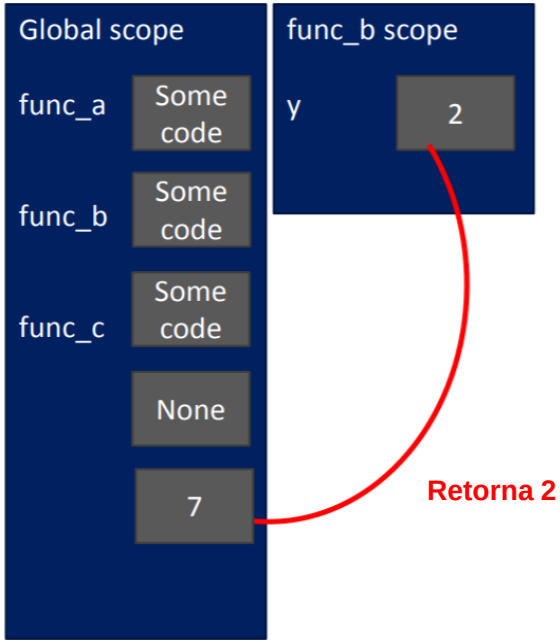
Funciones como argumentos

```
def func_a():
    print('Función A')
def func_b(y):
    print('Función B')
    return y
def func_c(z):
    print('Función C')
    return z
print(func_a())
print(5 + func_b(2))
print(func_c(func_b(2)))
```



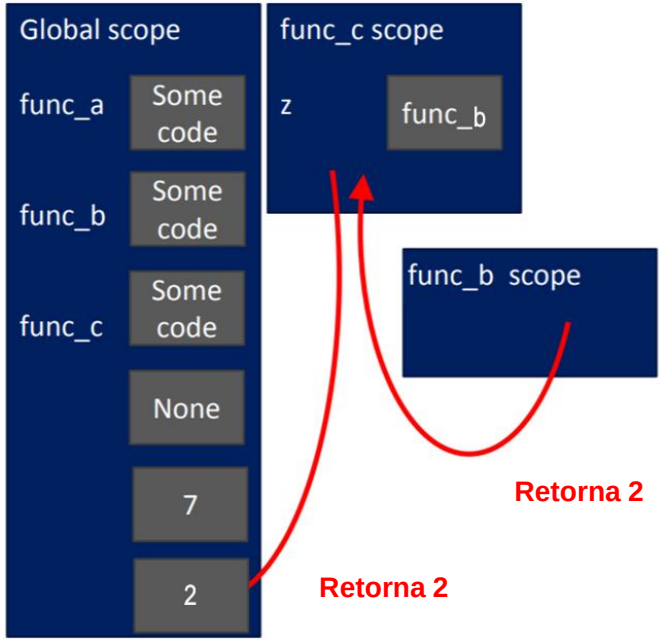
Funciones como argumentos

```
def func_a():
    print('Función A')
def func_b(y):
    print('Función B')
    return y
def func_c(z):
    print('Función C')
    return z
print(func_a())
print(5 + func_b(2))
print(func_c(func_b(2)))
```



Funciones como argumentos

```
def func_a():
    print('Función A')
def func_b(y):
    print('Función B')
    return y
def func_c(z):
    print('Función C')
    return z
print(func_a())
print(5 + func_b(2))
print(func_c(func_b(2)))
```



Funciones como elemento

```
def suma(a,b):
  return(a+b)
def elevar(c):
  return(c**2)
```

```
elevar(suma(2,3))
```

25

```
def suma(a,b=0):
  return(a+b)
def elevar(c):
  return(c**2)
```

```
elevar(suma(2))
```

4

```
def suma(a,b=0):
  return(a+b)
def elevar(c,d):
  return(c**d)
```

```
elevar(suma(2),3)
```

8

Expresiones lambda

Expresiones lambda

- Las expresiones lambda se usan idealmente cuando necesitamos hacer algo simple y estamos más interesados en hacer el trabajo rápidamente en lugar de nombrar formalmente la función.

- Las expresiones lambda también se conocen como funciones anónimas.

- Como sabes, una función en Python se define con la palabra reservada *def*. Sin embargo, una función anónima se define con la palabra reservada *lambda*.

Definir una función anónima en Python

La sintaxis para definir una *función lambda* es la siguiente:

```
lambda parámetros: expresión
```

A continuación, te detallo las características principales de una *función anónima*:

- Son funciones que pueden definir cualquier número de parámetros pero una única expresión. Esta expresión es evaluada y devuelta.
- Se pueden usar en cualquier lugar en el que una función sea requerida.
- Estas funciones están restringidas al uso de una sola expresión.
- Se suelen usar en combinación con otras funciones, generalmente como argumentos de otra función.

Ejemplo

```
# Función Lambda para calcular el cuadrado de un número
square = lambda x: x ** 2
print(square(5))

# Funcion tradicional para calcular el cuadrado de un numero
def square1(num):
  return num ** 2
print(square(5))
```

```
25
25
```

Ejemplos

```
lambda_func = lambda x: True if x**2 >= 10 else False
print(lambda_func(3))
print(lambda_func(4))
```

```
False
True
```

Ejemplos

```
my_dict = {"A": 1, "B": 2, "C": 3}
sorted(mi_dict, key=lambda x: my_dict[x]%3)
```

```
['C', 'A', 'B']
```

```
print(my_dict["A"]%3)
print(my_dict["B"]%3)
print(my_dict["C"]%3)
```

```
1
2
0
```

Reto

Crear una función que al insertar una lista de *números* arroje la siguiente salida:

```
Los números ingresados fueron [1, 2, 3, 4]
La suma de los números es: 10
El valor de la multiplicación de los números es: 24
El valor de los números elevados al cuadrado y sumados es: 331776
```

Condiciones:

- Tiene que servir con cualquier lista de números de cualquier tamaño la lista
- No puede usar librerías, sólo código escrito a mano (Usar for o while SÍ se puede, funciones o funciones lambda)

Intro a Github

¿Qué es Github?

GitHub es un sitio web y un servicio en la nube que ayuda a los desarrolladores a almacenar y administrar su código, al igual que llevar un registro y control de cualquier cambio sobre este código.

¿Qué es una versión de control?

Una Versión de Control ayuda a los desarrolladores llevar un registro y administrar cualquier cambio en el código del proyecto de software. A medida que crece este proyecto, la versión de control se vuelve esencial.

Con la *bifurcación*, un desarrollador duplica parte del código fuente (llamado *repositorio*). Este desarrollador, luego puede, de forma segura, hacer cambios a esa parte del código, sin afectar al resto del proyecto.

Luego, una vez que el desarrollador logre que su parte del código funcione de forma apropiada, esta persona podría *fusionar* este código al código fuente principal para hacerlo oficial.

¿Qué es Git?

Git es un **sistema de control específico de versión de fuente abierta** creada por Linus Torvalds en el 2005.

Específicamente, Git es **un sistema de control de versión distribuida**, lo que quiere decir que la base del código entero y su historial se encuentran disponibles en la computadora de todo desarrollador, lo cual permite un fácil acceso a las bifurcaciones y fusiones.

Fork and clone

La palabra fork se traduce al castellano, dentro del contexto que nos ocupa, como bifurcación. Cuando hacemos un fork de un repositorio, **se hace una copia exacta en crudo del repositorio original que podemos utilizar como un repositorio git cualquiera.**

Cuando creas un repositorio en GitHub, existe como un repositorio remoto. Puedes clonar tu repositorio para crear una **copia local en tu ordenador y sincronizarlo** entre las dos ubicaciones.

Commit, push and pull

Ahora que tienes una copia local y una copia en tu cuenta de GitHub, hay cuatro cosas que tendrás que saber hacer para colaborar.

- Commit (Confirmar) es el proceso que registra los cambios en el repositorio. Piensa en ello como una instantánea del estado actual del proyecto. Las confirmaciones se hacen localmente.
- Push (Empujar) envía el historial de confirmaciones recientes de tu repositorio local a GitHub. Si eres el único que trabaja en un repositorio, empujar es bastante simple. Si hay otras personas que acceden al repositorio, es posible que tengas que hacer un pull antes de poder hacer un push.
- Pull - un pull coge cualquier cambio del repositorio de GitHub y lo fusiona en tu repositorio local.
- Sincronización - la sincronización es como halar, pero en lugar de conectarse a su copia de GitHub del repositorio bifurcado, vuelve al repositorio original y trae cualquier cambio. Una vez que hayas sincronizado tu repositorio, tienes que empujar esos cambios de vuelta a tu cuenta de GitHub.

¡Gracias!

¿Preguntas?