Introducción a Big Data con Python

Programación en Python. Control de flujo, funciones y orientación a objetos

Jesús García López de Lacalle

6 de octubre de 2016

Control de flujo

- Sentencias condicionales:

```
if encontrado:
        print str(codigo)
```

```
if encontrado:
        print str(codigo)
else:
        print -1
```

```
if en_clase0:
        print 'clase0'
elif en_clase1:
        print 'clase1'
elif en_clase2:
        print 'clase2'
else:
        print 'error'
```

```
print str(codigo) if encontrado else print -1
```

Control de flujo

Bucles:

- edad = 0
  while edad < 18:
  edad += 1
  print 'Felicidades, tienes ' + str(edad) + ' años'

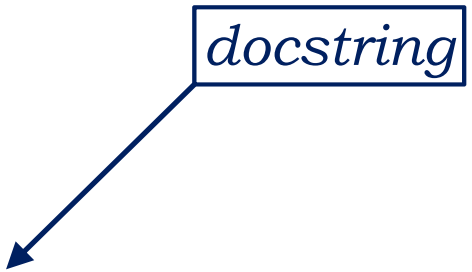
- secuencia = ['uno', 'dos', 'tres']
  for elemento in secuencia:
  print elemento

- range(3) → [0, 1, 2]
  range(2,5) → [2, 3, 4]
  range(3,8,2) → [3, 5, 7]
  range(8,3,-2) → [8, 6, 4]

  range(-2) → [ ]
  range(4,1) → [ ]
  range(8,3,2) → [ ]
  range(3,8,-2) → [ ]

Funciones

- Definición:
  - def mi_función(param1, param2):
    print param1
    print param2
  - def mi_función(param1, param2):
    """Esta función imprime los dos valores pasados como parametros"""
    print param1
    print param2
  - help(mi_función) → """Esta función imprime los dos valores pasados como parametros"""



Funciones

- Evaluación (ejecución):
  - mi_función('hola', 2) → 'hola'
    2
  - mi_función(param2 = 2, param1 = 'hola') → 'hola'
    2
  - mi_función('hola') → error
- Valores por defecto de los parámetros:
  - def imprimir(texto, veces = 1):
    print texto * veces
  - imprimir('uno', 2) → unouno
  - imprimir('uno') → uno

Funciones

- Número variable de parámetros:
  - def varios(param1, param2, *otros):
    ```
    for val in otros:
        print val
    ```
    tupla
  - varios(1, 2, 3, 4) → 3
    4
  - def varios(param1, param2, **otros):
    ```
    for i in otros.items():
        print i
    ```
    diccionario
  - varios(1, 2, tercero = 3, cuarto = 4) → 3
    4

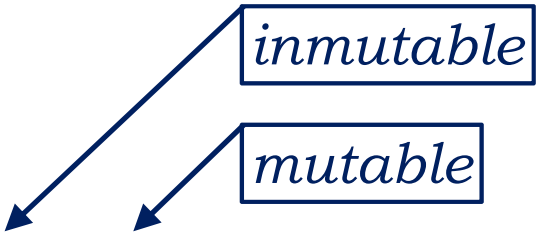
Funciones

- Paso de parámetros (valor/referencia):
  - En Python los objetos se pasan por referencia
  - En Python solo hay objetos: → todos los parámetros se pasan por referencia
  - def f(x, y):
        x += 3
        y.append(23)

```
x = 22
y = [22]
print x, y
f(x,y)
print x y
```

→

```
22 [22]
22 [22,23]
```



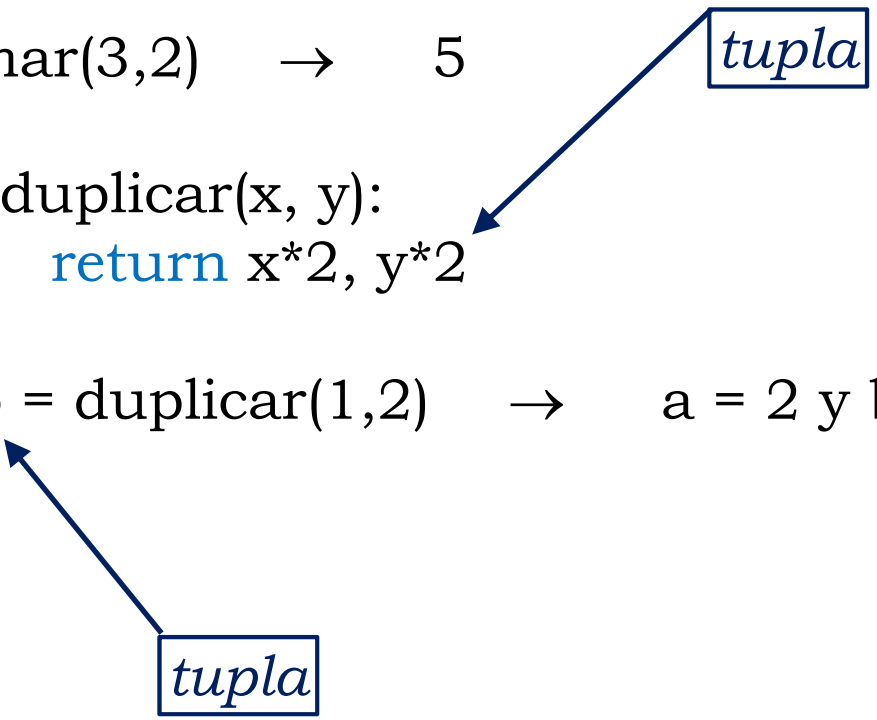
Funciones

- Devolver valores:
  - def sumar(x, y):
    return x + y

    sumar(3,2) → 5

  - def duplicar(x, y):
    return x*2, y*2

    a, b = duplicar(1,2) → a = 2 y b =4



Orientación a objetos

- Clases:
  - class Grafo:

```
"""Abstraccion de los objetos grafo."""

def __init__(self, n):
    self.num_ver = n
    self.Mady = [[0 for i in range(n)] for j in range(n)]

def conexo(self):
    <...>
```

instanciación

docstring

método

Atributos, declarados implícitamente

constructor

  - Primer argumento de todos los métodos: self
  - Crear objetos de una clase: mi_grafo = Grafo(5)
  - Llamar a un método de la clase: res = mi_grafo.conexo()

Orientación a objetos

- Encapsulación:
  - Atributos y métodos privados:
    su nombre debe empezar con dos guiones bajos (y no terminar también en dos guiones bajos)

```
class Ejemplo:
        def publico(self):
                print 'Publico'

        def __privado(self):
                print 'Privado'

ej = Ejemplo()
ej.publico()      →    'Publico'
ej.__privado()    →    <error>
```

Orientación a objetos

- Encapsulación:
  - Cómo acceder a métodos y atributos privados

```
class Ejemplo:
        def __init__(self,n)
                self.__n = n

        def __privado(self):
                print 'Privado‘

ej = Ejemplo(10)
ej._Ejemplo__privado()   →   'Privado'
ej.__n                   →   <error>
ej._Ejemplo__n           →   10
```

Orientación a objetos

- Encapsulación:
  - Propiedades (*getters* y *setters*)

```
class Fecha():
        def __init__(self)
                self.__dia = 1
        def getDia(self):
                return self.__dia
        def setDia(self, dia):
                self.__dia = dia
```

```
mi_fecha = Fecha()
mi_fecha.setDia(33)
print mi_fecha.getDia()
```

```
class Fecha(object):
        def __init__(self)
                self.__dia = 1
        def getDia(self):
                return self.__dia
        def setDia(self, dia):
                self.__dia = dia
        dia = property(getDia, setDia)
```

```
mi_fecha = Fecha()
mi_fecha.dia = 33
print mi_fecha.dia
```