Paradigmas de Programación

Sesión 8 Laboratorio

Cristian Tejedor García
Departamento de Informática
Universidad de Valladolid

Curso 2016-17

Grado en Ingeniería Informática
INDAT

Contenido

1. Objetivos
2. Programación funcional
3. Función anónima
4. Cálculo lambda
5. Lista por comprensión

1. Objetivos

- Introducir el paradigma funcional.
- Explicar las funciones anónimas en Python.
- Explicar el cálculo lambda en Python.
- Explicar las listas por comprensión en Python.
- Realizar ejercicios de repaso.

2. Programación funcional

- Utiliza ***funciones puras***: funciones que no tienen efectos secundarios, que no manejan datos mutables o de estado.
  - Lo contrario que la programación imperativa.
- Haskell es puro y Python no (es multiparadigma).
- Más **fácil** de **testear** y **depurar**: menos líneas, no bucles y condicionales en colecciones.
- Su **modularidad** facilita la computación concurrente y paralela: *n-cores* en multiprocesadores actuales.
- Algoritmos para **Big Data** (al estilo fórmulas Excel).

3. Función anónima

- Función sin nombre que no puede referenciarse más tarde.
- *Sintaxis*: unilínea: lambda <parámetros> : <función>

```
f = lambda x : x % 2 == 0
print f(5)  # False
print f(4)  # True
```

- Se utiliza en el cálculo *lambda* en Python.
- Si se quiere pasar como parámetro una función muy sencilla y definirla a parte sería innecesario.
- Si se quiere encapsular variables y que no queden globales.

4. Cálculo lambda (I)

- **map**: aplica una operación sobre cada uno de los ítems de un iterable (agregación de datos que se puede recorrer).
- *Sintaxis*: **map(lambda** <parámetros> **:** <función> **,** <iterable>**)**
- *Output*: lista con la función aplicada a todos los elementos.

```
print map(lambda x: x % 2 == 0, [1, 2, 3, 4, 5, 6])
# [False, True, False, True, False, True]
```

```
sonPares = [ ]
for x in [1, 2, 3, 4, 5, 6]:
   sonPares.append(x % 2 == 0)
print sonPares
# [False, True, False, True, False, True]
```

4. Cálculo lambda (II)

- **filter**: verifica que los elementos de un iterable cumplen una condición.
- *Sintaxis*: **filter(lambda** <parámetros> **:** <función> **,** <iterable>**)**
- *Output*: lista con los elementos que cumplen la condición.

```
print filter(lambda x : x % 2 == 0, [1, 2, 3, 4, 5, 6])
# [2, 4, 6]
```

```
doyPares = [ ]
for x in [1, 2, 3, 4, 5, 6]:
    if x % 2 == 0:
        doyPares.append(x)
print doyPares
# [2, 4, 6]
```

4. Cálculo lambda (III)

- **reduce**: reduce el iterable entero a un único elemento.
- *Sintaxis*: **reduce(**<par1,par2> **:** <función>**,**<iterable>**,** <defecto>**)**
- *Output*: valor de un iterable que se le ha aplicado una función por pares.

```
print reduce(lambda x, y : x + y, [2, 2, 2], 20)
# 26
```

```
resultado = 20
for i in [2, 2, 2]:
    resultado += i
print resultado
# 26
```

4. Cálculo lambda (IV)

- **zip**: reorganiza iterables. No hay que indicar *lambda*.
- *Sintaxis*: **zip(**<lista1> **,** <lista2>**)** o **zip(***<lista>**)**
- *Output 1*: toma el elemento *iésimo* de cada iterable y los une en una tupla. Después une todas las tuplas en una lista.
- *Output 2*: desune una lista de pares en dos listas.

```
lista = ['a', 'b', 'c', 'x', 'y', 'z']
lista_num = range(0, len(lista))
print zip(lista, lista_num)
# [('a', 0), ('b', 1), ('c', 2), ('x', 3),
('y', 4), ('z', 5)]
```

```
lista_nueva = [(1, 4), (2, 5), (3, 6)]
u, v = zip(*lista_nueva)
print u, v
# (1, 2, 3) (4, 5, 6)
```

5. Lista por comprensión (I)

- Al estilo matemático puro:
  - ***Comprensión***: {i \in Z | 1<i<6 } vs ***Extensión***: {1,2,3,4,5}
- **Elegante** y **rápido** ☺ ☺ ☺
- Se **ejecuta** más **rápidamente**: implementada directamente en lenguaje C.
- **No modifica** la **original**.
- *Sintaxis*: [<expresión> <(n) for> <condición opcional>]

5. Lista por comprensión (II)


```
# I. Potencias de los elementos de 0 a 9
print [x ** 2 for x in range(10)]
# [0, 1, 4, 9, 16, 25, 36, 49, 64, 81]

# II. Todo a mayúsculas
print [s.upper() for s in ("copos", "avena", "leche")]
# ['COPOS', 'AVENA', 'LECHE']
```

```
# III. Filtrado por pares
print [2 * i for i in range(1, 6) if i % 2 == 0]
# [4, 8]
```

```
ArrayList<Integer>pares =
            new ArrayList<Integer>();
for (int i = 1; i <= 5; i ++) {
   if (i % 2 == 0){
      pares.add(2 * i);   }  }
System.out.println(pares);
```

5. Lista por comprensión (III)


```
# IV. Filtrado en diccionario
notas = {'Rosa':10, 'Paco':7.5, 'Pedro':2, 'Natalia':5, 'Silvia':4, 'Jona':8}
print [nombre for nombre, nota in notas.items() if nota >= 5]
# ['Natalia', 'Rosa', 'Jona', 'Paco']
```

```
# Número primo: restos de sus divisores > 0
print [8 % d for d in range(2, 8)]
# [0, 2, 0, 3, 2, 1]
print [17 %d for d in range(2,17)]
# [1, 2, 1, 2, 5, 3, 1, 8, 7, 6, 5, 4, 3, 2, 1]
```

```
# lambda y list comprehension
def esprimo(x):
    return reduce(lambda x, y: x * y,
            [x % d for d in range(2, x)],
            1) > 0
print esprimo(8), esprimo(17)
# False, True
```

Para afianzar…

- Ejercicios de la asignatura (sesión 7) ~1hora:
  - https://www.infor.uva.es/~cvaca/asigs/docpar/sesion7.pdf
- “Python para todos”, Raúl González Duque
  - http://mundogeek.net/tutorial-python/
  - **Capítulo**: Programación funcional.