2 Informática (Teoría)

Python: Esquemas Elementales

Joaquim Gabarro
gabarro@cs.upc.edu

Computer Science
Universitat Politècnica de Catalunya

Funciones que invocan a funciones

Iteración e iterable

Esquemas de búsqueda y recorrido

Ejemplos con el tipo `str`

Funciones que invocan a funciones

Ejemplo: ordenar 4 números

Queremos ordenar 4 números `x`, `y`, `z`, `t`

- Utilizar `sort3FS(x,y,z)` para ordenar `x`, `y`, `z` y almacenar en `first`, `second`, `third`.
- Mediante analisis por casos situar `t` en el lugar que le corresponde entre `first`, `second`, `third` y devolver el resultado.

Fichero `sort4-pelada.py`

```
def sort3FS(x,y,z):
    first=x
    if y <= first:
        #second = first
        first, second = y, first
    else:
        second =y
    if z <= first:
        return z, first, second
    elif first < z <= second:
        return first, z, second
    else:
        return first, second, z


def sort4(x,y,z,t):
    first, second, third = sort3FS(x,y,z)
    if t<= first: return t, first, second, third
    if first <= t < second: return first, t, second, third
    if second <= t < third: return first, second, t, third
    return first, second, third, t
```

Compilamos con `Run-> Run Module` obtenemos in `IDLE`

```
>>>
= RESTART: C:...\sort4-pelada.py =
>>>
```

Podemos llamar a `sort4` en `IDLE`.

```
>>> sort4(5,4,3,2)
(2, 3, 4, 5)
>>>
```

Tambien podemos añadir el `main` y el `doctest` como vemos seguidamente.

Versión Completa

fichero `sort4.py`

```
def sort3FS(x,y,z):
    '''
    >>> sort3FS(5,4,3)
    (3, 4, 5)
    '''
    first=x
    if y <= first: first, second = y, first
    else: second =y
    if z <= first: return z, first, second
    elif first < z <= second: return first, z, second
    else: return first, second, z

def sort4(x,y,z,t):
    '''
    >>> sort4(5,4,3,2)
    (2, 3, 4, 5)
    '''
    first, second, third = sort3FS(x,y,z)
    if t<= first: return t, first, second, third
    if first <= t < second: return first, t, second, third
    if second <= t < third: return first, second, t, third
    return first, second, third, t

if __name__ == "__main__":
    print(sort3FS(5,4,3))
    print(sort4(5,4,3,2))
    import doctest
    doctest.testmod()
```

Iteración e iterable

Iteration and Iterable

Iteration is a general term for taking each item of something, one after another.

In natural language:

- Iteration is the process of taking one element at a time in a row of elements.
- Iterable is an object that is, well, iterable, which simply put, means that it can be used in iteration, e.g. with a `for` loop.

Veremos seguidamente que, en el caso de los strings (palabras) podemos iterar:

- Sobre las letras de la palabra
- Sobre las posiciones que ocupan las letras en la palabra.

Ejemplo: vowels_count

Los strings son iterables.

```
>>> w="p" + "a" + "t" + "a" + "t" + "a"
>>> w
'patata'
```

Veamos que podemos iterar sobre las letras de la ”patata”.

vowels_count(w)

File `vowels_count.py`

```
def vowels_count(w):
    n=0
    for c in w:
        if c in "aeiouAEIOU": n=n+1
    return n
```

cuando ejecutamos:

```
>>> vowels_count("patata")
3
>>> vowels_count("cenicienta")
5
```

Ejemplo de ejecución de `vowels_count("patata")`:

- Inicialmente `n=0` y comenzamos a ejecutar el for. El `for` iterará, por orden, sobre las letras de la palabra.
- Inicialmente `c=''p''` como `c in ''aeiouAEIOU''` es `False`, la `n` no se incrementa.
- En el siguiente paso `c=''a''` como `c in ''aeiouAEIOU''` es `True`, la `n` se incrementa y pasa a valer `1`.
- En los pasos siguiente la `c` toma por valores `c=''t''`, `c=''a''`, `c=''t''`, `c=''a''`

Acceder por posición

```
>>> w="patata"
>>> w
'patata'
>>> len(w)
6
>>> list(range(len(w)))
[0, 1, 2, 3, 4, 5]
>>> w[0]
'p'
>>> w[1]
'a'
>>>
```

Veamos que podemos iterar sobre las posiciones las letras de la ”patata”.

Another `vowels_count(w)`

File `vowels_count_position.py`

```
def vowels_count(w):
    n=0
    l=len(w)
    for i in range(l):
        if w[i] in "aeiou": n=n+1
    return n
```

Class exercise: `consonant_count(w)`

Desing a function `consonant_count(w)` to count the number of consonants in a word `w`. Develop the following two approaches:

- The function `consonant_count(w)` calls `vowels_count(w)` and uses `len()` such that:

```
>>> w="patata"
>>> len(w)
6
```

- Remember that a letter is a consonant when is not a vowel.

Esquemas de búsqueda y recorrido

Esquema de Búsqueda

En `busqueda.py` tenemos un esquema transversal (también llamado genérico).

```
def busqueda(elemento, iterable):
    for elemento_actual in iterable:
        if elemento_actual==elemento: return True
    return False
```

Es directamente compilable y ejecutable:

```
>>>
=== RESTART: C:....\busqueda.py ===
>>> busqueda("a", "patata")
True
>>>  busqueda("i", "pera")
False
>>> busqueda("patata", ["pera", "hola", "patata"])
True
```

Atención `Python`, es de (muy) alto nivel (en relación a `C` o `Java`. A veces hay cosas muy fáciles de hacer en `Python`:

```
>>> "a" in "patata"
True
>>> "patata" in ["pera", "hola", "patata"]
True
>>>
```

Esquema de Búsqueda general

Atención NO es un esquema transversal (no compila).

```
inicializar_busqueda
for elemento_actual in iterable:
        if encontrado: return elemento_tratado
return no_encontrado
```

Atención hay que definir `inicializar_busqueda`, `elemento_tratado`, `no_encontrado`.

Un clásico: búsqueda de la primera vocal

Fichero `look_for_vowell.py`

```
def look_for_vowel(w):
    for c in w:
        if c in "aeiouAEIOU": return c
    return "no vowell"
```

Esquema de Conteo

En `conteo.py` tenemos un esquema transversal (o genérico).

```
def conteo(elemento, iterable):
    n=0
    for elemento_actual in iterable:
        if elemento_actual==elemento: n=n+1
    return n
```

Un clásico: contar letras “a” en una palabra

```
>>>
==== RESTART: C:\...\conteo.py ====
>>> conteo("a", "patata")
3
>>> conteo("a", "pera")
1
>>> conteo("a", "pero")
0
>>> conteo("a", "")
0
```

En este caso concreto, también podemos utilizar

```
>>> "patata".count("a")
3
>>> "".count("a")
0
```

Esquema de Recorrido General

Permite tratar (uno a uno) los elementos de una colección y obtener un resultado

```
inicializar_tratamiento
for elemento in iterable:
     tratar_elemento
 return resultado_tratamiento
```

Atención NO es un esquema transversal (no compila).

Hay que definir, en cada caso concreto hay que definir: `inicializar_tratamiento`, `tratar_elemento` y `resultado_tratamiento`.

Ejemplos con el tipo `str`

Ejemplo: Invertir (capgirar) una palabra

Por ejemplo

```
invertir("Eusebi Güell i Bacigalupi")
    = 'ipulagicaB i lleüG ibesuE'
```

Más simple

```
invertir("abcdefg")= 'gfedcba'
```

Soluciones aplicando un esquema de recorrido, varias posibilidades.

Primera solución: Iterar sobre (o recorrer) las posiciones de los caracteres dentro de la palabra.

```
>>> w="abcdefg"
>>> len(w)
7
>>> range(len(w))
range(0, 7)
>>> list(range(len(w)))
[0, 1, 2, 3, 4, 5, 6]
>>> w[0]
'a'
>>> w[6]
'g'
>>> w[6]+w[5]
'gf'
```

Hay que contruir `w[6]+w[5]+...+w[0] = 'gfedcba'`

Fichero `invertir-pos.py`

```
def invertir(w):
    w_invertit = ""
    for i in range(len(w)):
              w_invertit =  w[i]+w_invertit
    return w_invertit
```

```
>>>
= RESTART: C:....\invertir-pos.py =
>>> invertir("abcdefg")
'gfedcba'
>>>
```

Segunda solución: Iterar directamente sobre los caracteres en la palabra (fichero `invertir-char.py`):

```
def invertir(w):
    w_invertit = ""
    for c in w:
              w_invertit =  c+w_invertit
    return w_invertit
```

Finalmente: Python es un lenguaje de (muy) alto nivel. En la próxima clase explicaremos el `::`

```
>>> w="abcdefg"
>>> w_invertit=w[::-1]
>>> w_invertit
'gfedcba'
```

Ejemplo: palíndromo

Ejemplos: anilina, oso, ojo, radar, reconocer, salas, seres, somos, sometemos.
Tratemos un caso: `some-t-emos`

```
>>> w= "sometemos"
>>> list(range(len(w)))
[0, 1, 2, 3, 4, 5, 6, 7, 8]
>>> w[0]
's'
>>> w[-1]
's'
>>> w[1]==w[-2]
True
>>>
```

Notad que `w[0]` es el primer caracter y `w[-1]` el último. Así sucesivamente.

Primera solución: es una búsqueda; buscamos el primer caràcter del comienzo que sea distinto del correspondiente del final.

En el caso de `w= "sometemos"` se cumple:

```
w[0]==w[-1] and w[1]==w[-2] and ...  and
w[3]==w[-4]
```

Obtenemos el programa (fichero `palindrom-bus.py`):

```
def palindromo(w):
    for i in range(len(w)//2):
        if w[i]!=w[-(i+1)]: return False
    return True
```

Segunda solución: una palabra es un palindromo si es igual a su inversa (fichero `palindrom-inv.py`).

```
def invertir(w):
    w_invertit = ""
    for c in w:
                w_invertit =  c+w_invertit
    return w_invertit

def palindromo(w):
    return w==invertir(w)
```

Class exercise: `double_letter`

Design a function `double_letter` such that:

```
>>> double_letter("patata")
'ppaattaattaa'
```