# Programació per a *Data Science*

## Unitat 3: Conceptes avançats de Python

### Instruccions d'ús

Al mòdul anterior hem introduït conceptes bàsics sobre variables i el seu ús en Python. En aquest mòdul estudiarem conceptes més avançats com les instruccions de flux d'execució (*for*, *while*, *if*), com definir i utilitzar funcions, com llegir i escriure fitxers i com organitzar el codi.

## Iteració i operacions lògiques

En la majoria de casos haurem de manipular les nostres dades, i per fer-ho utilitzarem els conceptes d'iteració i les operacions lògiques. Les operacions lògiques ens permeten comparar valors entre variables (més gran, més petit, igualtat), i la iteració, anar visitant un a un els elements d'una llista, tupla, diccionari o qualsevol estructura de dades que sigui susceptible de seqüenciar-se.

In [1]:
```
# Les operacions lògiques tindran com a resultat un valor cert (True) o fals (False):

a = 5
b = 1

# El valor a és més gran que b?
print a > b
```

```
True
```

In [2]:
```
# El valor a és més petit que b?
print a < b
```

```
False
```

In [3]:
```
b = 5
# El valor de b és igual que el d'a?
print b == a
```

```
True
```

In [4]:
```
# Altres operadors lògics disponibles són més petit o igual '<=', més gran o igual '> ='
# o la negació 'not'
print a <= b
print a >= b

a = False
print not a
```

```
True
True
True
```

També podem alterar el flux d'execució del nostre programa utilitzant les estructures *if... else* o *if... elif... else*. Vegem-ne uns quants exemples:

In [5]:
```
a = 5
b = 6
if a > b:
    print 'a és major que b'
else:
    print 'a és menor o igual que b'
```

```
a és menor o igual que b
```

In [6]:
```
a = 5
b = 5
if a > b:
    print 'a és major que b'
elif a < b:
    print 'a és menor que b'
else:
    print 'a és igual a b'
```

```
a és igual a b
```

En Python hi només dues maneres d'iterar una seqüència: mitjançant ***for*** o mitjançant ***while***. La primera de les opcions, *for*, iterarà un per un els elements continguts en una llista. En el cas de *while*, iteraremos mentre la condició de permanència al bucle es compleixi. Vegem-ne uns exemples:

In [7]:
```
monsters = ['Kraken', 'Leviathan', 'Uroborus', 'Hydra']

# Primer mètode iterant mitjançant for:
for monster in monsters:
    print monster

print

# Segon mètode. La funció especial 'enumerate' ens retorna una tupla en què el primer element és un
# índex que comença per 0 i augmenta d'1 a 1 i el segon element és el valor de la posició a la llista:
for i, monster in enumerate(monsters):
    print i, monster
```

```
Kraken
Leviathan
Uroborus
Hydra

0 Kraken
1 Leviathan
2 Uroborus
3 Hydra
```

In [8]:
```
# També podríem iterar la llista mitjançant while, però és una manera molt menys idiomàtica en Python
# i preferirem sempre l'opció de for:
i = 0
# Mentre que l'índex 'i' sigui més petit que la longitud de la llista 'monsters':
while i < len(monsters):
    # Imprimeix el valor de la llista en la posició 'i'.
    print i, monsters[i]
    # No ens oblidem d'actualitzar el valor de 'i' sumant-li 1 o tindrem un bucle infinit.
    i += 1
```

```
0 Kraken
1 Leviathan
2 Uroborus
3 Hydra
```

En aquest moment seríem capaços de calcular la sèrie de Fibonacci fins a un determinat valor:

In [9]:
```
# Calculem el valor de la sèrie fins a un valor n = 100.
n = 100

a, b = 0, 1
while a < n:
    print a,
    a, b = b, a+b
```

```
0 1 1 2 3 5 8 13 21 34 55 89
```

A Python disposem d'una funció molt útil per generar una seqüència de nombres, que podem utilitzar de diferents maneres:

In [10]:
```
# La funció 'range' ens retorna una llista de nombres:
range(10)
```

Out[10]:
```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

In [11]:
```
# Podem utilitzar-la per iterar:
for i in range(10):
    print i,
print
```

```
0 1 2 3 4 5 6 7 8 9
```

In [12]:
```
# Podem definir el rang d'acció. Per exemple, calcula els nombres entre 5 i 20 de tres en tres:
range(5,20,3)
```

Out[12]:
```
[5, 8, 11, 14, 17]
```

In [13]:
```
# També és possible iterar en un diccionari:
country_codes = {34: 'Spain', 376: 'Andorra', 41: 'Switzerland', 424: None}

# Per clau:
for country_code in country_codes.keys():
    print country_code
print

# Per valor:
for country in country_codes.values():
    print country
print

# Per tots dos alhora:
for country_code, country in country_codes.iteritems():
    print country_code, country
```

```
376
41
34
424

Andorra
Switzerland
Spain
None

376 Andorra
41 Switzerland
34 Spain
424 None
```

## Funcions

Una altra manera molt important d'organitzar el flux d'execució és encapsulant una certa porció de codi en una funció reutilitzable. Una funció en Python utilitza el mateix concepte que una funció matemàtica. Per exemple, imaginem-nos la funció matemàtica:

*suma(x, y) = x + y*

A Python podem definir la mateixa funció de la manera següent:

In [14]:
```
# La funció 'suma' es defineix mitjançant la paraula especial 'def' i té dos arguments: 'x' i 'y':
def suma(x, y):
    # Tornem el valor de la suma
    return x + y

# En aquest moment, podem anomenar-la amb qualsevol valor:
print suma(2, 4)
print suma(5, -5)
print suma(3.5, 2.5)
```

```
6
0
6.0
```

In [15]:
```
# Podem definir una funció que no faci res utilitzant la paraula especial 'pass':

def dummy():
    pass

dummy()
```

In [16]:
```
# Podríem tornar a definir el tros de codi de la seqüència de Fibonacci com una funció:

def fibonacci(n=100):
    a, b = 0, 1
    while a < n:
        print a,
        a, b = b, a+b

fibonacci(10)
```

```
0 1 1 2 3 5 8
```

In [17]:
```
# A l'exemple anterior, hem definit que l'argument n tingui un valor per defecte. Això és molt útil
# en els casos en què utilitzem la funció sempre amb un mateix valor, vulguem deixar constància d'un
# cas d'exemple o per defecte. En aquest cas, podem executar la funció sense passar-li cap valor:
fibonacci()
```

```
0 1 1 2 3 5 8 13 21 34 55 89
```

In [18]:
```
# Podem definir part dels arguments amb valors per defecte i un altre sense.
# Els arguments sense valor per defecte sempre han d'estar més a l'esquerra a la definició de la funció:

def potencia(a, b=2):
    # Per defecte, elevarem al quadrat.
    return a**b

print potencia(3)
print potencia(2, 3)
```

```
9
8
```

**Recomanem la lectura de la documentació oficial (https://docs.python.org/2/tutorial/controlflow.html)** per acabar de fixar coneixements.

## Llegir i escriure des de fitxers

Una tasca habitual és llegir línies d'un fitxer o escriure línies en un fitxer. A Python, llegir i escriure fitxers es fa amb la llibreria *os*. Una llibreria és un conjunt de codi que té cert sentit agrupar per a ser utilitzat per altra gent. En el nostre cas, *os* és una llibreria que agrupa funcions relacionades amb el sistema operatiu (***o**perating **s**ystem*). Per carregar una llibreria, utilitzarem la paraula reservada ***import***. Més endavant explicarem algunes particularitats d'utilitzar i importar llibreries. A continuació us expliquem com escriure i llegir un fitxer:

In [1]:
```
# Primer de tot, escriurem en un fitxer:
import os

# Obrim un fitxer de nom 'a_file.txt' per a escriptura (d'aquí la 'w', 'writing').
# L'assignem a un objecte per gestionar el fitxer de nom 'out':
out = open('a_file.txt', 'w')
# Escriurem 10 línies, cadascuna amb un número del 0 al 9.
for i in range(10):
    # La línia següent escriu al fitxer tot el que posem dins 'out.write()'
    # En el nostre cas, és un string del tipus '0\n', '1\n', etc. Això ho aconseguim
    # utilitzant els wildcards %d i %s.
    # %s representa un string o una cadena de caràcters i %d un nombre enter.
    # Concatenem en aquest cas un nombre amb un string que es tracta del salt de
    # línia, os.linesep, que equival a Linux
    # a '\n':
    out.write("Línea %d%s" % (i, os.linesep))

out.close()
```

In [4]:
```
# Ara llegirem el fitxer que acabem d'escriure de tres maneres diferents:

# Primer mètode
f = open('a_file.txt')
for line in f:
    print line,
f.close()

print

# Segon mètode
f = open('a_file.txt')
lines = f.readlines()
for line in lines:
    print line,
f.close()

print

# Tercer mètode
with open('a_file.txt') as f:
    for line in f:
        print line,
```

```
Línea 0
Línea 1
Línea 2
Línea 3
Línea 4
Línea 5
Línea 6
Línea 7
Línea 8
Línea 9

Línea 0
Línea 1
Línea 2
Línea 3
Línea 4
Línea 5
Línea 6
Línea 7
Línea 8
Línea 9

Línea 0
Línea 1
Línea 2
Línea 3
Línea 4
Línea 5
Línea 6
Línea 7
Línea 8
Línea 9
```

## Organització del codi

Un mòdul de Python és qualsevol fitxer amb extensió *.py* que estigui sota la ruta del *path* de Python. El path de Python es pot consultar important la llibreria sys:

In [21]:
```
import sys
print sys.path
```

```
['', '/usr/local/lib/python2.7/dist-packages/python_louvain-0.4-py2.7.egg', '/usr/local/lib/python2.7/dist-packages/Ha
shmal-0.1.0a0-py2.7.egg', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-t
k', '/usr/lib/python2.7/lib-old', '/usr/lib/python2.7/lib-dynload', '/home/cris/.local/lib/python2.7/site-packages',
 '/usr/local/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages/PILco
mpat', '/usr/lib/python2.7/dist-packages/gtk-2.0', '/usr/lib/python2.7/dist-packages/IPython/extensions']
```

Per defecte, Python també mira les llibreries que s'hagin definit a la variable d'entorn $PYTHONPATH (això pot canviar lleugerament en un entorn Windows (https://docs.python.org/2/using/cmdline.html)).

Un paquet en Python és qualsevol directori que contingui un fitxer especial de nom *__init__.py* (aquest fitxer estarà buit la majoria de vegades).

Un mòdul pot contenir diferents funcions, variables o objectes. Per exemple, definim un mòdul de nom *prog_datasci.py* que contingui:

In [22]:
```
# prog_datasci.py

PI = 3.14159265

def suma(x, y):
    return x + y


def resta(x, y):
    return x - y
```

Per utilitzar des d'un altre mòdul o *script* aquestes funcions, hauríem d'escriure el següent:

In [23]:
```
from prog_datasci import PI, suma, resta

# I llavors podríem utilitzar-les normalment.
rset = suma(2,5)
```

A Python també podem utilitzar la directiva *from prog_datasci import* *, però el seu ús està **totalment desaconsellat**. La raó és que estaríem important gran quantitat de codi que no utilitzarem (amb el consegüent augment de l'ús de la memòria), però a més, podríem tenir col·lisions de noms (funcions que es diguin de la mateixa manera en diferents mòduls) sense el nostre coneixement. Si no és que és imprescindible, no utilitzarem aquesta directiva i importarem una a una les llibreries i les funcions que necessitarem.

Podeu aprendre més sobre importar llibreries i definir els vostres propis mòduls aquí (http://life.bsc.es/pid/brian/python/#/5).