Lezioni di Python
Day 2

Dr. Giuliano Taffoni – Dr. Giuseppe Murante

Controlli e decisioni

- Le strutture di controllo sono delle particolari istruzioni, tipiche dei linguaggi imperativi, che permettono di eseguire delle istruzioni secondo determinate condizioni.

```
>>> a = 10

>>> if a == 1:
...     print(1)
... elif a == 2:
...     print(2)
... else:
...     print('A lot')
A lot
```

!Notate la indentazione del testo!

Iterazioni

- Come in altri linguaggi esistono cicli di vario tipo: **for, while**

```
>>> for i in range(4):
...     print(i)
0
1
2
3
```

```
>>> for word in ('cool', 'powerful', 'readable'):
...     print('Python is %s' % word)
Python is cool
Python is powerful
Python is readable
```

Iterazioni

```
>>> a = [1, 0, 2, 4]
>>> for element in a:
...     if element == 0:
...         continue
...     print(1. / element)
```

"continua" alla iterazione successiva

```
>>> z = 1 + 1j
>>> while abs(z) < 100:
...     z = z**2 + 1
>>> z
(-134+352j)
```

Interrompe il ciclo

```
>>> z = 1 + 1j

>>> while abs(z) < 100:
...     if z.imag == 0:
...         break
...     z = z**2 + 1
```

Iterazioni

```
>>> d = {'a': 1, 'b':1.2, 'c':1j}

>>> for key, val in sorted(d.items()):
...     print('Key: %s has value: %s' % (key, val))
Key: a has value: 1
Key: b has value: 1.2
Key: c has value: 1j
```

Nota la nuova espressione per **`print`** che fa uso degli *indicatori di formato* ricorda printf in C

Identificati di formato

- Identificano il formato con leggere o stampare un output
- Sono preceduti da '%'

Tipo	Espressione	A video
%c	char	singolo carattere
%d (%i)	int	intero con segno
%e (%E)	float or double	formato esponenziale
%f	float or double	reale con segno
%g (%G)	float or double	utilizza %f o %e in base alle esigenze
%o	int	valore base 8 senza segno
%p	pointer	valore di una variabile puntatore
%s	array of char	stringa (sequenza) di caratteri
%u	int	intero senza segno
%x (%X)	int	valore base 16 senza segno

I files in Python

- Oggetti di tipo file

```
>>> F = open('test.txt','w')
>>> print F
```

- '*r*' – Read mode
- '*w*' – Write mode
- '*a*' – Appending mode
- '*r+* – Special read and write mode

```
>>> F.name
>>> F.close()
```

```
>>> F.readline()
>>> F.readlines()
```

```
>>> M = F.read()
>>> F.seek(0)
```

Position in a file in bytes **0 == Inizio**

```
>>> A='pippo'
>>> F.write(A)
```

How to write your first program

- Begin the code with a description of your program
- Comment the functions and the algorithms

- Use always self-explaining variables (better use longer name but clearer)

```
Magnitude = 1.4
```

~~`M = 1.4`~~

- Structure the source code in functions or modules

```python
#!/usr/bin/env python
#-----------------------------------------------------------------------------
# Copyright (c) 2013-2017, PyInstaller Development Team.
#
# Distributed under the terms of the GNU General Public License with exception
# for distributing bootloader.
#
# The full license is in the file COPYING.txt, distributed with this software.
#-----------------------------------------------------------------------------
"""
Prints a list of (maybe) empty hooks.

A hook is listed here if it does not define any of the meaningful
names (eg. hiddenimports). So beside empty hooks, this will print
hooks which import these name from a shared hook (like PIL.Image) or
are calling functions in hookutils.

Proposed usage::

        develutils/find-empty-hooks.py | sort | xargs emacs
        # then in emacs, remove all content in hooks which are realy empty
        # Now delete all hook-files less then 2 bytes in size:
        find PyInstaller/hooks/ -size -2c -print —delete
"""
import glob
import os
```

```
# Wrap os.path.basename()
if sys.platform.startswith('win'):
        # Implementation from ntpath.py module
        # from standard Python 2.7 Library.
        def os_path_basename(pth):
                ## Implementation of os.path.splitdrive()
                if pth[1:2] == ':':
                        d = pth[0:2]
                        p = pth[2:]
                else:
                        d = ''
                        p = pth
                ## Implementation of os.path.split()
                # set i to index beyond p's last slash
                i = len(p)
                while i and p[i - 1] not in '/\\':
                        i = i - 1
                head, tail = p[:i], p[i:] # now tail has no slashes
                # Windows implementation is based on split(). We need
                # to return only tail.
                return tail
```

Excercise

Wallis's equation for π. In 1655, the English mathematician John Wallis (1616–1703) devised this wonderful-looking infinite product involving only rational numbers to calculate pi:

$$\frac{\pi}{2} = \prod_{n=1}^{\infty}\left[\frac{(2n)^2}{(2n-1)(2n+1)}\right] = \frac{2\cdot 2}{1\cdot 3}\,\frac{4\cdot 4}{3\cdot 5}\,\frac{6\cdot 6}{5\cdot 7}\cdots$$

Scope of variables: global vs local

- A variable which is defined in the main body of a file is called a *global* variable.
- It will be visible throughout the file, and also inside any file which imports that file.
- Global variables can have unintended consequences because of their wide-ranging effects: we should almost never use them

- A variable which is defined inside a function is *local* to that function.
- It is accessible from the point at which it is defined until the end of the function, and exists for as long as the function is executing.
- When we use the assignment operator (=) inside a function, its default behavior is to create a new local variable – unless a variable with the same name is already defined in the local scope.

Exercise

- What is the lifetime of these variables? When will they be created and destroyed?
- Can you guess what would happen if we were to assign c a value of 1 instead?
- Why would this be a problem? Can you think of a way to avoid it?

```
def my_function(a):
        b = a - 2
        return b

c = 3
if c > 2:
        d = my_function(5)
        print(d)
```

Python for science: numpy

- extension package to Python for multi–dimensional arrays
- closer to hardware (efficiency)
- designed for scientific computation (convenience)
- Also known as *array oriented computing*

```
>>> import numpy as np
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
```

- In scientific computing arrays contain your data.

Numpy performance example

```
In [1]: L = range(1000)
In [2]: %timeit [i**2 for i in L]
The slowest run took 19.21 times longer than the fastest. This could mean
that an intermediate result is being cached10000 loops,
best of 3: 58 µs per loop

In [3]: import numpy as np
In [4]: a = np.arange(1000)
In [5]: %timeit a**2
The slowest run took 17.85 times longer than the fastest. This could mean
that an intermediate result is being cached 1000000 loops,
best of 3: 1.51 µs per loop
```

Memory-efficient container that provides fast numerical operations

Howto create arrays

- a = np.ones((3, 3))
- b = np.zeros((2, 2))
- c = np.eye(3)
- d = np.diag(np.array([1, 2, 3, 4]))
- e = np.random.rand(4)
- f = np.random.randn(4)

- g = np.empty()

```
>>> a = np.array([0, 1, 2, 3])
>>> a
array([0, 1, 2, 3])
>>> a.ndim
1
>>> a.shape
(4,)
>>> len(a)
4
```

```
>>> b = np.array([[0, 1, 2], [3, 4, 5]])    # 2 x 3 array
>>> b
array([[0, 1, 2],
       [3, 4, 5]])
```

```
>>> c = np.array([[[1], [2]], [[3], [4]]])
```

Data Type and indexing

- Also Arrays have a data type

```
>>> a.dtype
```

- And an index as lists

```
>>> a = np.arange(10)
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> a[0], a[2], a[-1]
(0, 2, 9)
```

```
>>> a[::-1]
array([9, 8, 7, 6, 5, 4, 3, 2, 1, 0])
```

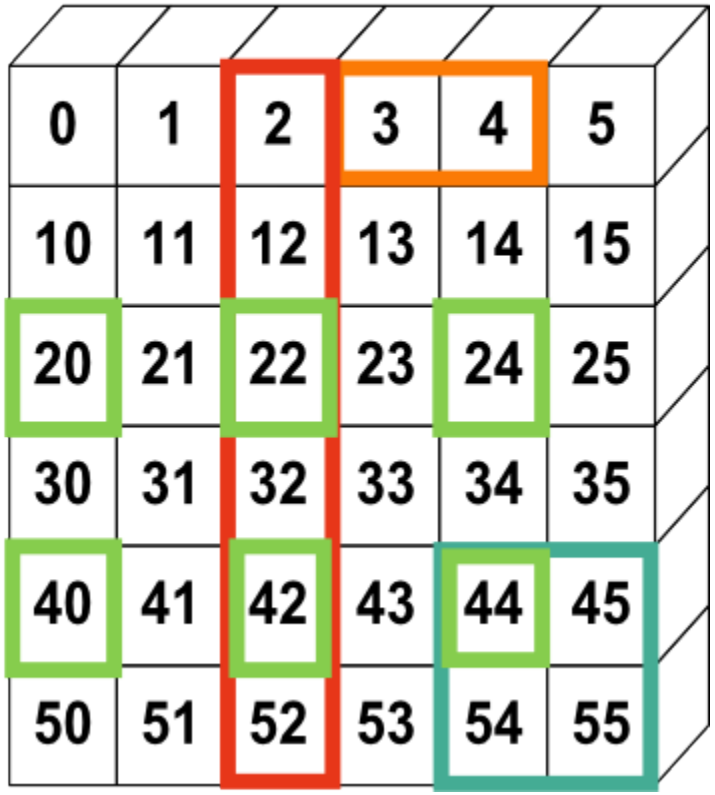
Slicing Arrays

```
>>> a[0,3:5]
array([3,4])

>>> a[4:,4:]
array([[44, 45],
       [54, 55]])

>>> a[:,2]
array([2,12,22,32,42,52])

>>> a[2::2,::2]
array([[20,22,24]
       [40,42,44]])
```



Copy an array

- Rember that in python a=b does not make a copy

```
>>> a = np.arange(10)
>>> c = a[::2].copy() # force a copy
>>> c[0] = 12
>>> a
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.may_share_memory(a, c)
False
```

Fancy Indexing

- NumPy arrays can be indexed with slices, but also with boolean or integer arrays (**masks**). This method is called *fancy indexing*. It creates **copies not views**.

```
>>> np.random.seed(3)
>>> a = np.random.random_integers(0, 20, 15)
>>> a
array([10,  3,  8,  0, 19, 10, 11,  9, 10,  6,  0, 20, 12,  7, 14])
>>> (a % 3 == 0)
array([False,  True, False,  True, False, False, False,  True, False,
        True,  True, False,  True, False, False], dtype=bool)
>>> mask = (a % 3 == 0)
>>> extract_from_a = a[mask] # or,  a[a%3==0]
>>> extract_from_a           # extract a sub-array with the mask
array([ 3,  0,  9,  6,  0, 12])
```

Polynomial and fitting

```
>>> p = np.poly1d([3, 2, -1])
>>> p(0)
>>> p.roots
>>> p.order
```

```
>>> x = np.linspace(0, 1, 20)
>>> y = np.cos(x) + 0.3*np.random.rand(20)
>>> p = np.poly1d(np.polyfit(x, y, 3))
```

Exercise

- Given the Right ascension (abbreviated RA) and Declination (abbreviation DEC) expressed in degrees convert them in radians.

Next Lesson

- Matplotlib and different kind of plots
- Advanced programming
- Error and exceptions
- Python for Astronomy

References and Credits

- http://python-textbok.readthedocs.io/en/1.0
- http://www.scipy-lectures.org/intro/index.html