

# Lezioni di Python

Dr. Giuliano Taffoni

# Cosa è Python

- Linguaggio di programmazione e scripting.
- “Object Oriented”
- Procedurale, funzionale → possiamo scegliere il nostro stile di programmazione
- Versioni: 2.x and 3.x
  - 3.x is **not** backwards compatible with 2.x

# Caratteristiche principali

- Linguaggio interpretato (o semi interpretato)
- Tipizzato e dinamico
- Gestione automatica della memoria
- Ampio set di librerie standard e prodotte da terzi (moduli)
  - E.g. numpy, matplotlib, astropy etc

# Caratteristiche principali

- Sintassi semplice e struttura molto leggibile
- Portabile (Unix/Linux, Mac OS X, Windows, Android)
- Interattivo
- Open source

# Tipi di Linguaggi di programmazione

- Un programma viene sviluppato scrivendo il codice sorgente in un opportuno linguaggio definito dalla sintassi del linguaggio stesso.
- Linguaggio compilato: codice sorgente viene trasformato (e ottimizzato) dal compilatore in linguaggio macchina che può essere eseguito dal processore;

```
void store(double *a, double *b, double *c)
{
    *c = *a + *b;
}
```



```
.text
.p2align 4,,15
.globl store
.type store, @function
store:
movsd (%rdi), %xmm0 # Load *a to
%xmm0
```

# Linguaggi interpretati

- **Linguaggio Interpretato:** il codice sorgente viene interpretato al volo e vengono eseguite le istruzioni così come descritte nel codice sorgente.
- **Linguaggio semi interpretato:** il codice sorgente viene compilato in un formato intermedio (chiamato bytecode), il quale a sua volta viene interpretato dall'interprete python, che ha il compito di interpretare “al volo” le istruzioni bytecode in istruzioni per il processore.

Myprogram.py  Myprogram.pyc

# Le variabili

- "la programmazione si basa sui dati (variabili) e le istruzioni (codice e funzioni). [...] Le istruzioni dicono al computer cosa fare con le variabili"
- "We store values in variables. Each variable is identified by a *variable name*. [...] Each variable has a *variable type*"
- "a variable must be defined in a declaration statement"
  - Serve a definire il nome e il tipo
- Variables are given a value through the use of an *assignment statement*.

```
la_mia_variabile = il_suo_valore
```

# Le variabili in Python

- Tutti gli elementi in python sono oggetti: variabili, funzioni, classi etc.
- Il linguaggio è tipizzato: oggetti tipizzati, identificatori non tipizzati.
- Il linguaggio è dinamico: gli oggetti non sono dichiarati all'inizio ma solo quando vengono usati (assignment statement)



# Python interattivo

- ipython

# Commenti e documentazione

```
'''
```

```
Commento su piu' righe  
Utile per documentare un programma  
'''
```

```
# commento su riga singola
```

# Alcuni esempi

a=3

p=[1, 2 , 3 , 4 ]

k=p

a="tre "

- Crea l'oggetto numero intero di valore 3 e gli assegna il nome a
- Crea l'oggetto lista di quattro elementi e gli assegna il nome p
- Assegna alla lista di nome p un secondo nome k. In altri termini, crea un nuovo riferimento all'oggetto
- Crea l'oggetto stringa e gli assegna il nome a. L'oggetto 3 di cui sopra viene distrutto

# Tipi numerici

`int, long, float, complex`

- `a=3`
- `b=3.1415926`
- `c=15000L`
- `d=a+b`
- `e=7/2`
- `f=float(333)`
- `g=complex(1.5,2)`
- `h=None`
- `i=True`
- `print f * g`
- `print e`

# Le operazioni

- $x + y$
- $x - y$
- $x * y$
- $x / y$
- $x // y$
- `long(x)`
- `float(x)`
- `complex(re,im)`
- `c.conjugate()`
- `divmod(x, y)`
- `pow(x, y)`  $x ** y$
- `abs(x)`
- `int(x)`

# Operazioni sui bit

$x \mid y$

$x \wedge y$

$x \& y$

$x \ll n$

$x \gg n$

$\sim x$

```
>>> x=3
```

```
>>> y=10
```

```
>>> print bin(x), bin(y)
```

```
>>> print bin(x|y)
```

```
>>> print bin(x^y)
```

```
>>> print bin(x&y)
```

```
>>> print bin(x<<3)
```

```
>>> print bin(x>>4)
```

```
>>> print bin(~x)
```

# Stringhe

- Sono oggetti immutabili che definiscono un insieme di caratteri.

A="Corso di Laboratorio di Astronomia"

B='Corso di Cosmologia Teorica'

- Le stringhe si possono concatenare

C = A + B

- Posso accedere ai singoli caratteri o a una "sotto-stringa" (**slice**), ma non posso modificarli

A[6]

A[3:7]

A[:5]

A[-1]

~~A[5]='k'~~

# Linguaggio è tipizzato: verifichiamolo

- Il linguaggio è tipizzato: distingue interi da array da stringhe

A="Corso di Astronomia"

C = A + 7

**TypeError:** cannot concatenate 'str' and 'int' objects

type(A) → str



# Gli oggetti di tipo sequenza

- Stringhe “...” , Liste [...] e Tuple (...)
- Liste e tuple sono collezioni **non** omogenee di oggetti (int, str, char, oggetti etc)
- Liste sono modificabili tuple sono immutabili

```
a='corso di Astronomia'  
b=['a',3,5,3.14]  
c=(3,'x',0,'I')
```

```
3 in b  
'h' not in a  
b + b  
b + c  
a * 3, 4 * c  
a[4]  
del(b[3])  
len(a) min(b) max(b)
```

# Operazioni su liste (ma non su tuple!)

- `lista.reverse()`
- `lista.sort([cmp[,key[,reverse]]])`
- `map(fun, lista)`
- `filter(fun, lista)`
- `reduce(fun, lista)`



Argomenti opzionali

`fun == Funzione generica`

# Esempio

```
In [80]: import operator as op
In [81]: op.mul?
Docstring: mul(a, b) -- Same as a * b.
Type:      builtin_function_or_method
In [82]:
```

Come sommo tutti gli elementi di una lista?

Come moltiplico tutti gli elementi di una lista?

# Liste: facciamoole a fette

`A[0:3]` (`A[:3]`)

`B=range(1,10)`

`B[2:8:2]`

`B[-3:]`

	0	1	2	3	4	5	
	-6	-5	-4	-3	-2	-1	
<b>a:</b>	12	14	22	27	31	44	
	0	1	2	3	4	5	6
	-6	-5	-4	-3	-2	-1	

# Copie di oggetti e referenze

- Quando faccio una assegnazione **a=b** non effettuo una copia ma solo duplico la referenza (puntatore)
- Esistono copie `shallow` e copie `deep`.

```
>>> M = range(10)
>>> C = M
>>> C[5]=56
>>> print M
>>> B = M[:]
>>> id(B)
```

- `copy.copy()` shallow copy
- `copy.deepcopy()` deepcopy

# List comprehension

- Approccio “matematico” alla costruzione di liste (evoluzione della funzione *map*)

Math style

```
S = {x2 : x in {0 ... 9}}  
V = (1, 2, 4, 8, ..., 212)  
M = {x | x in S and x even}
```

- In python si traduce

```
>>> S = [x**2 for x in range(10)]  
>>> V = [2**i for i in range(13)]  
>>> M = [x for x in S if x % 2 == 0]
```

# Esempi complessi di list comprehension

- Numeri primi

```
>>> N = [j for i in range(2, 8) for j in range(i*2, 50, i)]  
>>> M = [x for x in range(2, 50) if x not in N]  
>>> print M  
[2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47]
```

- Ma non solo numeri



```
>>> words = 'The quick brown fox jumps over the lazy dog'.split()  
>>> stuff = [[w.upper(), w.lower(), len(w)] for w in words]
```

# Dictionary: array associativi

- Liste chiave:valore separate dal ':'

```
>>> dict = {'Name': 'M31', 'RA': 121.1743, 'DEC': -21.5733, 'REDSHIFT': -0.001001}
>>> dict.keys()
>>> dict.values()
>>> dict['Name'] = 'NGC224'
```

- Gli oggetti dictionary hanno vari metodi

`dict.has_key('c')`

`dict.copy()` [*shallow copy*]



# 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)
```

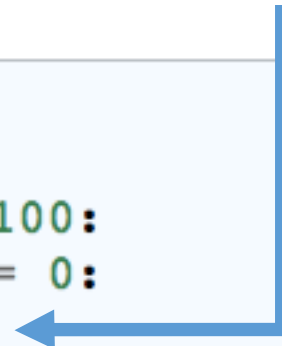
“continua” alla iterazione successiva



```
>>> 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

# Esercizio

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

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

# 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()
```

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



Position in a file in bytes 0 == Inizio

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

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

# Le prossime puntate

- Scrivere un programma in python
- Le funzioni
- Leggere argomenti dalla riga di comando
- Python 4 science: numpy e scipy
- Fare grafici scientifici in python
- Usare il python per data reduction



# Come scrivo un programma

- Leggibile
- `X=10 # cosa serve x`
- `def func:`
- `# cosa fa la funzione`
- `etcl`