

Python  
Introduzione ad un  
Uso Cosciente  
Nello Rizzo

**QUANDO VEDI QUELLE RIGHE DI  
CODICE CHE NON SIGNIFICANO NIENTE**

**MA SE LE RIMUOVI  
L'APPLICAZIONE NON FUNZIONA**



# La Filosofia del Linguaggio

- Punti di forza
  - Semplice ma potente
  - Multipurpose
- Punti di debolezza
  - Lint integrato nella sintassi
  - Duck typing
  - Preferita l'immediatezza al formalismo

# La Filosofia del Linguaggio

- Una libreria molto “sviluppata”
  - Visione verso la retrocompatibilità
  - Ampio spettro di librerie verso tutti gli interessi
    - Audio
    - Grafica
    - Sviluppo giochi
    - GIS
  - Aperture verso tecnologie utilizzate in altri ambiti
    - PyQT
    - Librerie per Audio (es. MP3), Grafica

# Panoramica

- Linguaggio facile e potente
- Orientato agli Oggetti
- Sintassi orientata alla lettura
  - Sintassi descrittiva
- Modalità interattiva per il test di piccole parti di codice
- Interoperabilità con C o C++

# Panoramica

- Disponibile su tutte le piattaforme
  - Windows
  - \*nix
  - Mac OS X
  - Raspberry
  - Android (build non ufficiale)

# Panoramica

- Vasta scelta di tipi di dato di base
  - Numeri reali e complessi
  - Long con lunghezza qualsiasi
  - Stringhe ASCII e Unicode
  - Liste
  - Dizionari

# Panoramica

- Codice organizzato in moduli e packages
- Tipi di dato tipizzati in maniera stretta ma dinamici
  - Segnalazione di mix tra tipi incompatibili
- Presenza di algoritmi avanzati
  - Generatori
  - List comprehension
- Gestione automatica della memoria



# Generatori

```
# A generator function that yields 1 for first time,  
# 2 second time and 3 third time  
def simpleGeneratorFun():  
    yield 1  
    yield 2  
    yield 3  
  
# Driver code to check above generator function  
for value in simpleGeneratorFun():  
    print(value)
```

# Generatori

```
# A generator function
def simpleGeneratorFun():
    yield 1
    yield 2
    yield 3

# x is a generator object
x = simpleGeneratorFun()

# Iterating over the generator object using next
print(x.next()) # In Python 3, __next__()
print(x.next())
print(x.next())
```

# Pensare ad Oggetti

- Single responsibility principle
- Open closed principle
- Liskov substitution principle
- Interface segregation
- Dependency injection

# Pensare ad Oggetti

- Non è un linguaggio per OOP
- Non esiste il concetto di classe astratta o interfaccia in senso stretto
  - La libreria abc (Python 2.6) pone una soluzione (funzionale anche se farraginoso) tramite annotazioni (@)
- Duck typing!!!

# Programmazione ad Oggetti

- Classi
  - Il riferimento `self`
  - Hanno il compito di “classificare” un’informazione
    - Attributi
    - Comportamento
  - Public View e Private View
    - Double underscore
  - Esiste una vi(s)ta privata in Python?

# Programmazione ad Oggetti

- Special Methods!
  - `__init__(self[, ...])`
  - `__del__(self)`
  - `__repr__(self)`
  - `__str__(self)`
  - `__eq__(self, other)`
  - `__ne__(self, other)`
  - `__lt__(self, other)`
  - `__le__(self, other)`
  - `__gt__(self, other)`
  - `__ge__(self, other)`
  - `__nonzero__(self)`
  - `__call__(self[,args...])`

# Programmazione ad Oggetti

- Ereditarieta Multipla
  - Funzione `super()`
  - Overriding
    - Overriding in multiple inheritance

# Tutto è una Funzione?

- `__call__`
- Overloading di operatori
  - Aritmetici
  - Logici



# Tutto è una Funzione?

Addition	<code>p1 + p2</code>	<code>p1.__add__(p2)</code>
Subtraction	<code>p1 - p2</code>	<code>p1.__sub__(p2)</code>
Multiplication	<code>p1 * p2</code>	<code>p1.__mul__(p2)</code>
Power	<code>p1 ** p2</code>	<code>p1.__pow__(p2)</code>
Division	<code>p1 / p2</code>	<code>p1.__truediv__(p2)</code>
Floor Division	<code>p1 // p2</code>	<code>p1.__floordiv__(p2)</code>
Remainder (modulo)	<code>p1 % p2</code>	<code>p1.__mod__(p2)</code>
Bitwise Left Shift	<code>p1 &lt;&lt; p2</code>	<code>p1.__lshift__(p2)</code>
Bitwise Right Shift	<code>p1 &gt;&gt; p2</code>	<code>p1.__rshift__(p2)</code>
Bitwise AND	<code>p1 &amp; p2</code>	<code>p1.__and__(p2)</code>
Bitwise OR	<code>p1   p2</code>	<code>p1.__or__(p2)</code>
Bitwise XOR	<code>p1 ^ p2</code>	<code>p1.__xor__(p2)</code>
Bitwise NOT	<code>~p1</code>	<code>p1.__invert__()</code>

# Tutto è una Funzione?

Operator	Expression	Internally
Less than	<code>p1 &lt; p2</code>	<code>p1.__lt__(p2)</code>
Less than or equal to	<code>p1 &lt;= p2</code>	<code>p1.__le__(p2)</code>
Equal to	<code>p1 == p2</code>	<code>p1.__eq__(p2)</code>
Not equal to	<code>p1 != p2</code>	<code>p1.__ne__(p2)</code>
Greater than	<code>p1 &gt; p2</code>	<code>p1.__gt__(p2)</code>
Greater than or equal to	<code>p1 &gt;= p2</code>	<code>p1.__ge__(p2)</code>

# Gestione delle Informazioni

- Liste e tuple
- Dizionari
- Pandas

# Liste e Tuple

- Concetto di collezione
- Una lista è un array
  - Ricerca lineare, non utile per grandi quantità di dati
- Tupla: una lista immutabile
- Set
  - Non sono sequenze
  - Ricerca sub-lineare
- Dizionari
  - Mappatura chiave-valore

# Liste

```
1 bigList = [str(i) for i in range(10000000)]
2 "abc" in bigList
3 bigSet = set(bigList)
4 "abc" in bigSet           # 10000 volte più veloce
```

```
seq = ["alpha", "bravo", "charlie", "delta"]
dict(enumerate(seq))
```

```
{0: 'alpha', 1: 'bravo', 2: 'charlie', 3: 'delta'}
Press any key to continue . . .
```

# Pandas

- Modulo fondamentale nell'analisi dei dati
  - Series
    - Vettore monodimensionale etichettato
  - DataFrame
    - Tabella di righe e colonne dotate di etichette

# Pandas

- Supporto per
  - Indicizzazione monolivello e gerarchica
  - Gestione dei dati mancanti
  - Operazioni aritmetiche e booleane su intere colonne e tabelle
  - Unione e aggregazione
  - Tracciamento grafico
  - Lettura e scrittura dati su file

# Pandas

- Funzioni di trasformazione dei dati
  - Indicizzazione
    - Etichette associate alle righe del frame
  - Indicizzazione gerarchica
    - Tre liste
      - Nomi di livello
      - Etichette possibili per ogni livello
      - Liste dei valori per ogni elemento
  - Stack e pivot



# Pandas

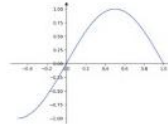
- Gestione dei dati mancanti
  - Un dato mancante è un dato non registrato
  - Cancellazione
  - Ricostruzione
  - Sostituzione
- Combinazione
  - Unione (join)
  - Concatenamento
- Ordinamento e classificazione

# Matplotlib

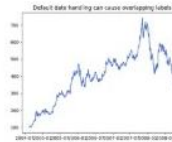
- Libreria per la creazione di grafici
  - Python + libreria matematica NumPy
- API orientate agli oggetti
  - Inserimento di grafici all'interno di applicativi usando toolkit GUI generici
    - WxPython, Qt o GTK
- Interfaccia "pylab"
  - procedurale
    - basata su una macchina degli stati progettata per assomigliare a quella di MATLAB
- Distribuita sotto licenza di tipo BSD.

# Matplotlib

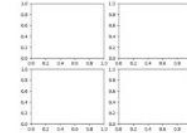
- Lines, bars and markers
- Images, contours and fields
- Subplots, axes and figures
- Statistics
- Pie and polar charts



Centered spines with  
arrows



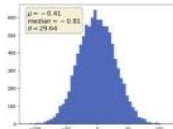
Fixing common date  
annoyances



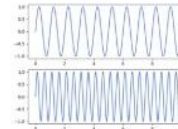
Easily creating  
subplots



Fill Between and  
Alpha



Placing text boxes



Sharing axis limits  
and views

# Matplotlib

- Text, labels and annotations
- Pyplot
- Color
- Shapes and collections
- Style sheets
- Axes Grid
- Axis Artist
- Showcase
- Animation
- Event handling
- Front Page
- Miscellaneous
- 3D plotting
- Our Favorite Recipes
- Scales
- Specialty Plots
- Ticks and spines
- Widgets

# Matplotlib

- Il report senza immagini è poco interessante
- Grafici
  - Sub-modulo pyplot
  - Gallery online
    - <https://matplotlib.org/gallery.html>
- Ideale per tracciare
  - Frames
  - Serie pandas

# Matplotlib

Thumbnail gallery — Matplotlib x +

← → ↺ 🏠 🔒 https://matplotlib.org/gallery.html

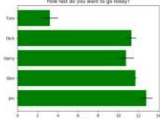
## matplotlib

[home](#) | [examples](#) | [gallery](#) | [pyplot](#) | [docs](#) »

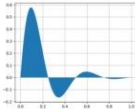
Click on any Image to see full size Image and source code

- [Gallery](#)
  - [Lines, bars, and markers](#)
  - [Shapes and collections](#)
  - [Statistical plots](#)
  - [Images, contours, and fields](#)
  - [Pie and polar charts](#)
  - [Color](#)
  - [Text, labels, and annotations](#)
  - [Ticks and spines](#)
  - [Axis scales](#)
  - [Subplots, axes, and figures](#)
  - [Style sheets](#)
  - [Specialty plots](#)
  - [Showcase](#)
  - [API](#)
  - [pylab examples](#)
  - [mplot3d toolkit](#)
  - [axes\\_grid toolkit](#)
  - [widgets](#)
  - [Miscellaneous examples](#)

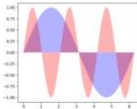
Lines, bars, and markers



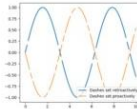
barh\_demo



fill\_demo



fill\_demo\_features



line\_demo\_dash\_control

# Scenari

- Flask
  - Sviluppo applicazioni web
- Django
  - “You can focus on writing your app without needing to reinvent the wheel”
- Cokiecutter
  - Distribuzione di progetti
    - Ottimizzazione
    - Automazione

# Data Science

- Estrarre conoscenze dai dati
- Tratta argomenti dissimili
  - Database
    - Memorizzazione e integrazione delle informazioni
  - Analisi del testo ed elaborazione del linguaggio naturale
    - I calcoli sulle parole
  - Analisi numerica dei dati e data mining
    - Ricerca di schemi coerenti e relazioni tra variabili



# Data Science

- Analisi delle reti complesse
  - Collezione di entità arbitrarie interconnesse
- Rappresentazione dei dati
  - Informare e convincere
- Machine Learning
  - Insegnare alla macchina i meccanismi decisionali
- Elaborazione di segnali
- Analisi di grandi quantità di dati