Python Workshop 2014

Politecnico Open unix Labs

Stefano Sanfilippo

3 giugno 2014

Contenuti del corso

1. Lo Zen di Python.
2. Pro, contro e critiche a Python.
3. Un po' di lessico.
4. *Dos and donts* – solo perché si può fare, non vuol dire che devi.
5. Idiomi e altre cose carine.
6. Interpreti, compilatori e macchine virtuali per Python.

Prerequisiti

Questo workshop non insegna
a programmare da zero.

Prerequisiti

Questo workshop non insegna
a programmare da zero.

Il testo ufficiale è il Python Tutorial:

- Online https://docs.python.org/3/tutorial/
- Scaricabile https://docs.python.org/3/download.html

Prerequisiti

Questo workshop non insegna
a programmare da zero.

Il testo ufficiale è il Python Tutorial:

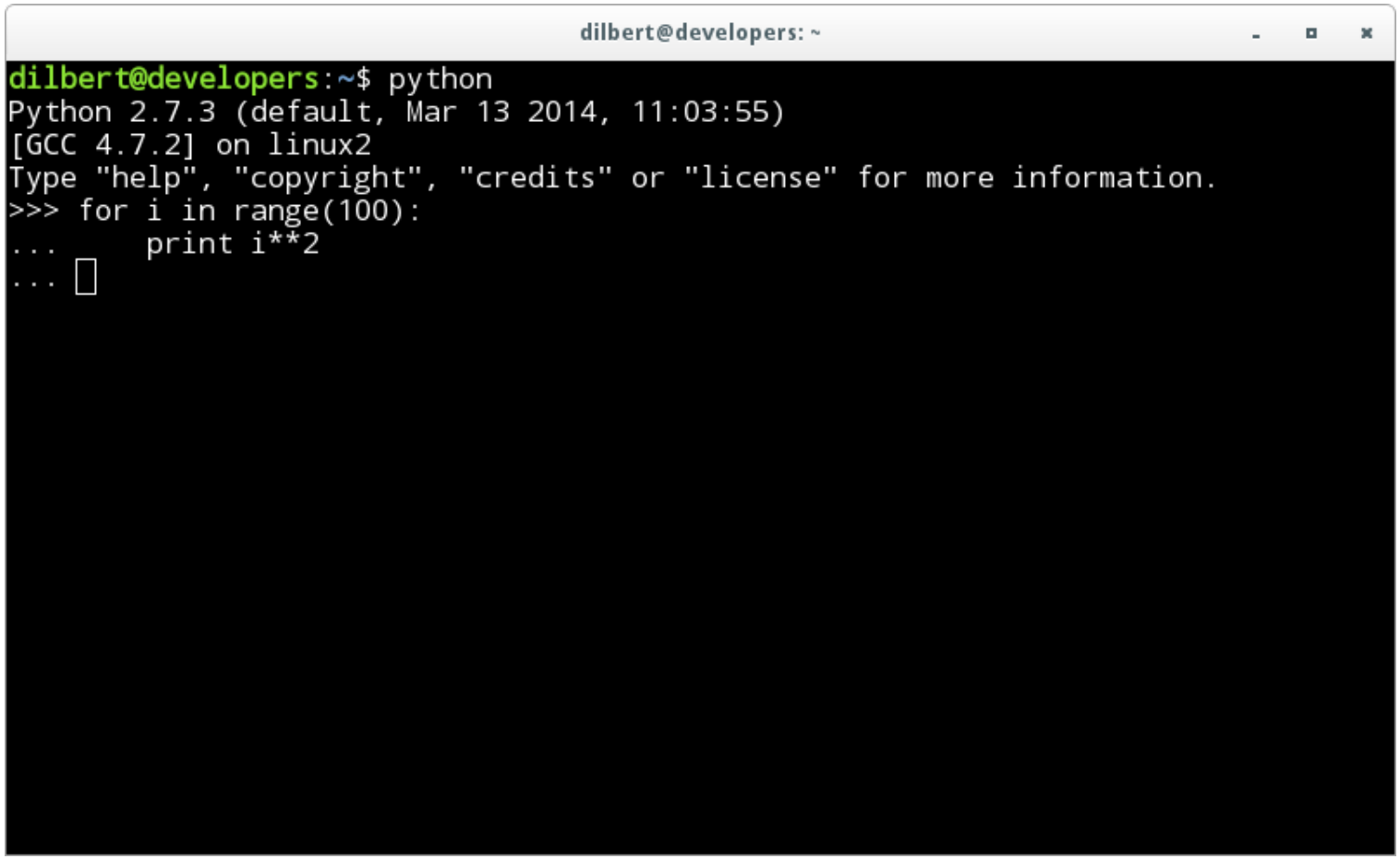
- Online https://docs.python.org/3/tutorial/
- Scaricabile https://docs.python.org/3/download.html

Un valido supporto interattivo è PythonMonk:

- https://pythonmonk.com/

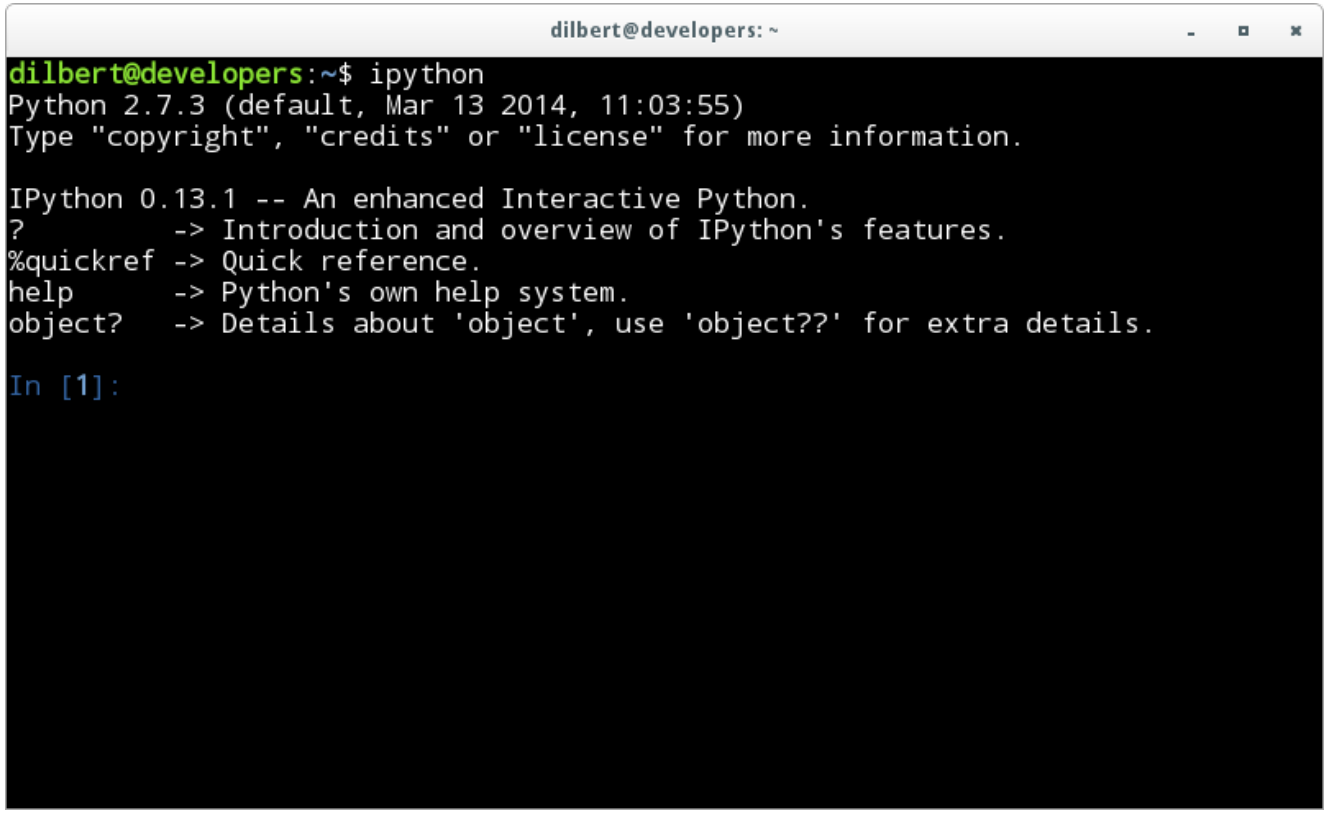
La shell

- **Shell interattiva**: facile sperimentare



(*detour*) IPython

IPython potenzia la shell di Python con *autocompletion*, informazioni contestuali, *history* e molto altro.



Con la ***I*** maiuscola. . .

PERL

```
0 and chr <<1;
'''#'
1
use strict;use warnings;
sub perl{
    map{m&${\uc(q,m,)}&&&($,.=$_,)=~s,[^A-z0-6],,xg}@_;($_=$,)
    =~tr$A-Za-z0-9+,$ -_$;for(unpack("u",join('',map(chr(32+
    length($_)*3/4).$_,m$(.{1,60})$gs)))){$.='$_'.'="';map
    {$..="\\$_"}unpack('(a3)*',$_.'012');$..='"';eval(${\$.}),
    s;\x50++\S{5}+;\u${\substr((caller(0e0))[0b11],-4)
};;print}}${{}={<<'1<<1'=>1}
#'''
l=('13713715115516015716216413713 7''0500471421411631450660640470 51',
   '16105614206606414414514315714 4''145050143051',
   'MTEyMTY1MTYzMTY0MDQwMTQxMTU2MTU3MTY0'
   'MTUwMTQ1MTYyMDQwMTIwMTcxMTY0MTUwMTU3'
   'MTU2MDQwMTUwMTQxMTQzMTUzMTQ1MTYyMDU0'
)
def l1(l):[(yield'\\'+l[i:i+(2|1)])for i in xrange(0,len(l),(2|1))]
i,I=lambda l:eval('"'+''.join(list(l1(l)))+'"'),lambda l:eval(i(l))
c,q=l[0o0+-0o1],I(l[0o1&0o2]);print(i(I(l[-(0o3^0o1)],)))
'''
1<<1
and 1;
perl(split/\n/,(keys(${{}}))[0])
#'''
```

(fonte: http://www.perlmonks.org/?node_id=1071660)

Lo Zen di Python

```
>>> import this
```

http://docs.python-guide.org/en/latest/writing/style/

- Ci deve essere un solo modo (evidente) di fare le cose
- La leggibilità conta
- Non reinventare la ruota

Il buono – la ruota è già stata inventata

- **Strutture dati** di alto livello
  - **Insieme**, **Lista**, **Dizionario**, **Tupla**
- **Batterie incluse**: una pletora di moduli pronti all'uso
  - HTTP, CSV, manipolazione XML, multiprocessing...
- **Aritmetica *done right***
  - **Multiprecisione** integrata, no *overflow*
  - **Divisione** senza troncamento
    - ⋆ Da Python 3, *backported* alla 2.7.5

Il buono – supporto

Esistono molte risorse e molte librerie esterne di ottima qualità.

- NumPy
- LXML
- SQLAlchemy
- Twisted
- Django
- Flask

Il buono – la sintassi I

Programs must be written for people to read, and only incidentally for machines to execute

—Abelson & Sussman
Structure and Interpretation of Computer Programs

Python privilegia la **velocità di sviluppo** e la **comprensibilità del codice** (a discapito della velocità di esecuzione).

Il buono – la sintassi II

- La sintassi punta ad essere **concisa ed espressiva**.
  - Le numerose funzioni incluse aiutano.
  - Meglio un *idioma* di un *design pattern*
- I blocchi del codice sono individuati **dall'indentazione**.
  - **Non c'è bisogno di parentesi**
  - L'indentazione **non può trarre in inganno**
- Esiste una guida allo stile ufficiale (PEP 8)
  - Ci ritorneremo più avanti

PEP: Python Enhancement Proposal

Le aggiunte o modifiche alle specifiche del linguaggio e dell'interprete vengono **pubblicamente discusse**, nella speranza di **attirare critiche costruttive**.

Il buono – il *type system*

- **Dynamic typing** una variabile può contenere valori di tipo diverso. Niente gerarchie di tipi complesse.
- **Automatic declaration** le variabili non vanno dichiarate prima dell'uso.
- **Duck typing** niente bisogno di interfacce.

(*detour*) Duck Typing I

It looks like a duck*, swims like a duck, and quacks like a duck.*

Then it probably is a duck.

(*detour*) Duck Typing II

```
class InfiniteA:
    def close():
        pass

    def read(self, size):
        return "A" * size
```

Ha un metodo `close()` *e un metodo* `read()` *come un file.*

Allora probabilmente è un file.

Il buono – docstring

- Funzioni, moduli e classi Python possono contenere **Docstring**, ovvero **descrizioni testuali del loro funzionamento**.
  - **Non sono semplici commenti**, sono accessibili dal codice
- Esistono ottimi strumenti per estrarre le docstring ed **impaginarle** in documentazione coerente.
  - Sphinx

Il buono – capacità dinamiche

- **Introspezione** permette di elencare e accedere a tutte le funzioni e le variabili definite nello scope.
- **Funzioni dinamiche** metodi e attributi possono essere aggiunti o modificati a runtime.
- **Funzioni come variabili** funzioni e *funtori* posso essere passati come variabili.
- **Annotazioni** (decoratori)

Python è il linguaggio **perfetto**!

- Leggibile
- Ricco di funzioni
- Semplice da imparare, semplice da leggere, veloce da scrivere

NOPE

Python (almost) everywhere!

- Python **non è il *silver bullet*** del software.
- Python **non è adatto a tutte le applicazioni**.
- È essenziale capirne i **limiti**, per non ritrovarsi con una **massa di codice ingestibile**.
  - Succede... *sadly*...

5 minuti di pausa

(. . . domande?)

Il brutto – type system I

- **Dynamic e duck typing rendono difficile determinare gli attributi** (metodi e proprietà) «caratterizzanti» dei tipi usati in un blocco di codice.

Perciò:

- **L'analisi statica** del codice è **impossibile**. O quasi. . .
- Il **refactoring** di progetti non triviali diventa un inferno
  - Auguri a scoprire quali attributi si aspetta una funzione da un certo argomento.

Il brutto – type system II

- Per tentar di **rimediare**, sono stati introdotti gli skeleton.
  - Per ciascuna funzione in una libreria, vengono delineati i tipi attesi e restituiti.
  - Qualcuno li deve scrivere
    - ⋆ Allora tanto valeva introdurre le dichiarazioni di tipo in Python

Il brutto – type system II

- Per tentar di **rimediare**, sono stati introdotti gli skeleton.
  - Per ciascuna funzione in una libreria, vengono delineati i tipi attesi e restituiti.
  - Qualcuno li deve scrivere
    - ⋆ Allora tanto valeva introdurre le dichiarazioni di tipo in Python
- Python 3 consente di **annotare** parametri e funzioni.
  - Per esempio, dando informazioni sul tipo atteso.
  - Da quando esiste Python 3, non ho mai visto nessuno farlo.

Il brutto – type system III

- In **assenza di documentazione**, non resta che **leggere** il codice sorgente.
  - La documentazione della libreria standard è di **ottima qualità**.
    - https://docs.python.org/3/
  - La documentazione di molte librerie esterne ***lascia a desiderare*** (per essere gentili)
    - M2Crypto ne è praticamente privo

Il brutto – . . .

Massì! Tanto due giorni e ho finito. . . [cit.]

- Spesso è più facile riscrivere il progetto da capo.
  - Ma tanto lo farà qualcun altro, vero? :P

Il cattivo – package management

- PyPI è un archivio online di *package* Python.
- È arrivato anni prima (2003) di RubyGems o NPM
- Nonostante ciò, conta meno *package.*
- pip (consigliato) e *setuptools* fungono da *package manager*
  - Non sono progetti ufficiali
  - Prima di Python 3.4, nessuno dei due era incluso nell'installazione standard
  - Installare pip su Windows è. . . divertente.

(detour) Egg?

- Nel gergo Python, `Egg == package`
- Prendono il nome da uno sketch dei Monty Python
  - da cui prende il nome il linguaggio
    - ★ No, i rettili non c'entrano

https://www.youtube.com/watch?v=anwy2MPT5RE

Il cattivo – bytecode

- **Non ci sono garanzie sulla retrocompatibilità** del bytecode
  - Ha cambiato versione (e struttura) con ogni major release.
  - È da considerare una **cache**, più che codice eseguibile
    - ★ Il software è distribuito come codice sorgente
    - ★ A meno di non distribuire interprete e *standard library*. . .
- Fino alla PEP 3147 (incorporata in Python 3.2), non erano supportate più versioni del bytecode.

2 vs. 3

Short version: Python 2.x is legacy, Python 3.x is the present and future
– Python 2 vs. 3 wiki page

Nel 2008 è stata introdotta la versione 3 di Python, che porta numerosi cambiamenti **incompatibili** con la versione 2:

- Stringhe **Unicode** di default
- **Separazione** netta tra stringhe e sequenze di byte
- Standard Library **ristrutturata**

2 vs. 3

Short version: Python 2.x is legacy, Python 3.x is the present and future
– Python 2 vs. 3 wiki page

Nel 2008 è stata introdotta la versione 3 di Python, che porta numerosi cambiamenti **incompatibili** con la versione 2:

- Stringhe **Unicode** di default
- **Separazione** netta tra stringhe e sequenze di byte
- Standard Library **ristrutturata**

- Divisione **senza troncamento** (`/` vs. `//`)
- Eliminate le «old style classes»
- *Exception chaining*
- **Annotazioni** per argomenti e tipi restituiti

__future__

```
from __future__ import with_statement
```

- Alcune feature introdotte in una certa versione, possono essere **backportate alle precedenti**.
- Per non rompere il codice esistente, vanno **esplicitamente richieste**.

https://docs.python.org/3/library/__future__.html

2 o 3?

- **Non tutte** le librerie sono state portate alla versione 3
- La macchina target potrebbe non avere Python 3
  - Di solito, è un problema sui server

2 o 3?

- **Non tutte** le librerie sono state portate alla versione 3
- La macchina target potrebbe non avere Python 3
  - Di solito, è un problema sui server

- Gli strumenti `2to3` e `3to2` possono guidare nella **parte noiosa del porting**.
- È più facile **backportare da Python 3 a Python 2** che farlo in avanti.
  - **Python 3 è più restrittivo**

Dos and Donts

Special cases aren't special enough to break the rules.
Although practicality beats purity.
–The Zen of Python

(vedi anche Idioms and Anti-Idioms in Python – Moshe Zadka)

Dos and Donts – import *

```
from module import *
```

- Rende difficile capire ad occhio da quale modulo arriva una certa funzione
- Inquina il namespace con definizioni non usate

Dos and Donts – import *

```
from module import *
```

- Rende difficile capire ad occhio da quale modulo arriva una certa funzione
- Inquina il namespace con definizioni non usate

MA

- È utile *nella shell interattiva* per avere a disposizione tutte le funzioni di certi moduli
  - `from numpy import *`
- Certi moduli sono *-safe

Dos and Donts – Pokémon exceptions

```
try:
    do_something()
except:
    pass
```

Dos and Donts – Pokémon exceptions

```
try:
    do_something()
except:
    pass
```

. . . Gotta catch'em all!

- In Python, **qualsiasi errore solleva un'eccezione**.
- Il blocco di codice sopra **silenzia tutto**, anche gli `ImportError`

Dos and Donts – esecuzione dinamica

Può essere utile,
ma è **estremamente pericolosa**.

Dos and Donts – esecuzione dinamica

Può essere utile,
ma è **estremamente pericolosa**.

- `r = eval("(1 + 2) - 3")`
  esegue espressioni e ne restituisce il risultato

Dos and Donts – esecuzione dinamica

Può essere utile,
ma è **estremamente pericolosa**.

- `r = eval("(1 + 2) - 3")`
  esegue espressioni e ne restituisce il risultato

- `exec "import sys; print(sys.argv)"`
  esegue la stringa passata come codice.

Dos and Donts – esecuzione dinamica

Può essere utile,
ma è **estremamente pericolosa**.

- `r = eval("(1 + 2) - 3")`
  esegue espressioni e ne restituisce il risultato

- `exec "import sys; print(sys.argv)"`
  esegue la stringa passata come codice.

Si può restringere lo scope, ma non ci mette al riparo.

Dos and Donts – `os.system()`

```
os.system("cat /etc/passwd")
```

Come la `system` del C.

Invoca una shell e le passa la stringa che ha come argomento.

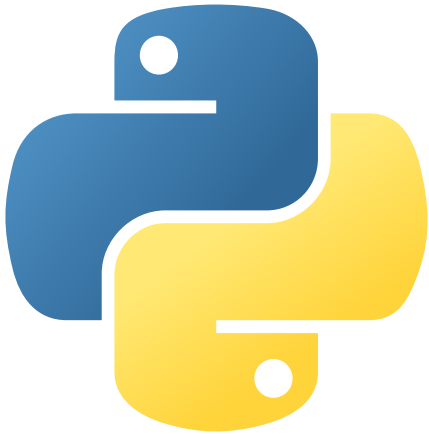
- *shell-injection* in 3... 2... 1...
- quando possibile, **usare** le funzioni in **`subprocess`**
  - non invocano una shell

L'interprete

. . . **gli** interpreti. . .

Una specifica
Più implementazioni

cPython



https://www.python.org/

- Scritto in C
- Implementazione di riferimento della Python Foundation
- Ultime versioni: **3.4.1** e 2.7.5
  - La prima a ricevere aggiornamenti

PyPy

http://pypy.org/

- Scritta in Python (*such recursion*)
- JIT
- Stackless
- Ultime versione: **3.2.3** e 2.7.6

Jython

http://www.jython.org/

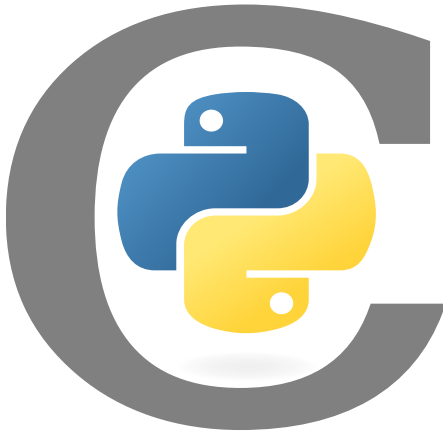
- Scritto in Java, esegue sulla JVM
- Ultima versione: 2.7.5

IronPython

http://ironpython.net/

- Scritto in C#, compila codice Python in assembly .NET
- Ultima versione: 2.7.4
- . . . sono in vacanza?

Cython



http://cython.org/

- Traduce codice Python in **estensioni C** per cPython
- Non è ufficialmente un'implementazione di Python
  - Arricchisce la sintassi con dichiarazioni di tipi statici che mappano direttamente su tipi C (`int`...)