

Corso di Python

Lezione 1

Introduzione al linguaggio Python

Editor: Davide Brunato

Scuola Internazionale Superiore di Studi Avanzati di Trieste



Come e perché Python

- Il corso illustrerà i vari aspetti di Python in 14 lezioni di circa un'ora e mezza ciascuna:
 - Prime 9 lezioni sulla sintassi del linguaggio e aspetti fondamentali
 - Le ultime 5 con esempi di utilizzo con librerie specifiche
- Perché Python per uso scientifico?
 - Disponibilità di librerie per il calcolo scientifico:
<https://wiki.python.org/moin/NumericAndScientific>
 - Semplicità nell'interfacciamento con altri linguaggi (C/C++)
- Perché Python per il sysadmin?
 - Scripting più sicuro e gestibile
 - Ha un'ampia community di supporto
 - Ricco di librerie aggiornate per dialogare con sistemi e database
 - Facilita l'approccio DevOps

Cos'è Python

- Linguaggio di programmazione ad alto livello, rilasciato per la prima volta nel 1991 da Guido Van Rossum
- Licenza open-source compatibile GPL
- Community di 80 sviluppatori
- Van Rossum decisore ultimo sulle modifiche al linguaggio (BDFL)
- Sito ufficiale: <http://www.python.org>

Caratteristiche di Python

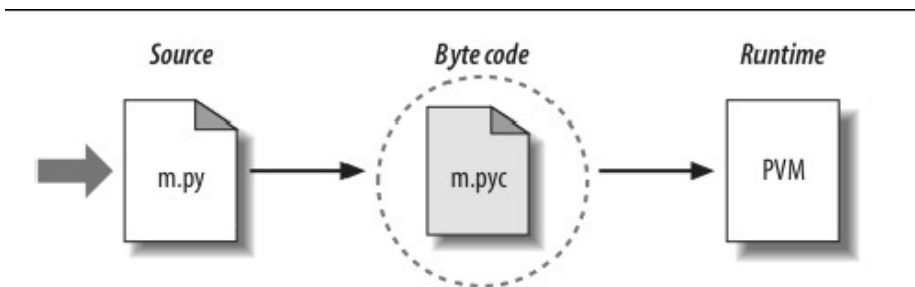
- Tipizzazione dinamica forte
- Linguaggio semplice, altamente leggibile
- Object oriented
- *Intepretato* (compilato in byte-code)
- Interattivo
- Portabile
- Modulare
- Interfacciabile con altri linguaggi
- Ampia disponibilità di librerie
- Documentazione

La filosofia di Python

- Ridurre le ambiguità ed eliminare il superfluo
- Per separare i blocchi utilizza l'indentazione
 - Le parentesi usate solo nell'ambito della singola istruzione
 - Miglioramento della leggibilità del codice
- Esplicito è meglio che implicito
 - Utilizzo di convenzioni sui nomi al posto di direttive ([PEP8](#))
- Loose coupling: *meno dipendenze possibili tra i componenti*
- Duck Typing: *“If it walks like a duck and talks like a duck, it’s a duck”*

```
# python -c 'import this' #Zen of Python
```

Runtime execution model



- Il sorgente viene tradotto in *byte code*
- Il byte code viene interpretato dalla Python Virtual Machine
- La velocità di esecuzione è comparabile a quella di Java
 - La dinamicità ha un costo sulle performance
 - Python è facilmente interfacciabile con moduli in C++
 - *BDFL*: “Non è molto veloce, ma è veloce abbastanza per fare quello che ci serve”

Da Python 2 a Python 3

- Revisione per rendere più coerente il linguaggio
- Le versioni 2.6 e 2.7 contengono costrutti di Python 3, per cui si può pensare ad passaggio dolce e prolungato nel tempo
- Principali differenze sintattiche tra v2 e v3:
 - In Python 3 tutte le stringhe sono per default Unicode (nella 2 sono ASCII)
 - != al posto di <> nei confronti
 - Revisione della sintassi di alcune istruzioni (evidenza di *print*)
 - Alcune librerie modificate nei path/nomi
 - Miglioramento di alcune funzioni base del linguaggio
 - Revisione rappresentazione testuale dei tipi predefiniti
- **2to3**: libreria per tradurre codice da python 2 a 3

Versioni di Python

- Binari e sorgenti disponibili su <https://www.python.org/downloads/>
 - Disponibile per piattaforme Linux/Unix based in vari formati
 - Disponibile per piattaforme Windows come installer a 32/64 bit
- Oltre alla versione base (**CPython**) esistono altre versioni speciali:
 - **Jython**: implementazione in Java di Python
 - **Python for .NET**: CPython + librerie specifiche per .NET
 - **IronPython**: sempre orientato a .NET ma implementato con un compilatore che genera codice oggetto per .NET
 - **PyPy**: Implementazione di Python scritta completamente in Python, che punta all'incremento delle performance con un compilatore Just In Time e esecuzione stackless
 - **Cython**: Superset di Python con un meccanismo di chiamata per funzioni esterne scritte in C/C++, che permette di generare moduli di estensione per CPython
 - **Skulpt**: Implementazione di Python in Javascript per la didattica interattiva con il browser (<http://www.skulpt.org/>)

Documentazione e libri

- Documentazione ufficiale: <https://docs.python.org/>
 - Disponibile per le versioni 2.6-3.x
 - Scaricabile in PDF, HTML, EPUB
- Alcuni libri free disponibili in PDF e/o EPUB:
 - “Dive into Python” e “Dive into Python 3” (Apress), disponibili anche in italiano
 - “A Byte of Python”: per principianti, con esempi ed esercitazioni
 - Il recente “Functional Programming in Python” (O'Reilly)
- Altre risorse free disponibili online:
 - “Python Cookbook” (O'Reilly)
 - “Python Practice Book”: per un approccio pratico al linguaggio
 - “Python Course” da Google (con esercizi)

Installazione su Linux

- La release binaria per Linux è solitamente suddivisa su più pacchetti:
 - **python** : interprete base
 - **python-libs** : librerie standard
 - **python-devel** : headers per i sorgenti in C
 - **python-tools** : utilities (**idle** IDE, 2to3, smtpd, ...)
 - **python-test** : librerie per il testing delle applicazioni
 - **python-debug** : versione dell'interprete abilitata per il debug

Strumenti di Python

- **python** in modalità interattiva per provare le funzionalità del linguaggio
- **pip** : gestore di pacchetti Python
 - Repository **PyPI** (59K pacchetti!)
- Ambienti virtuali separati per lo sviluppo:
 - Pacchetto **virtualenv**
 - Applicativo **pyenv**, incluso in Python 3.4+

Installazione multipla

- Si possono installare più versioni di Python
- Per Windows, che non lo utilizza di base sul sistema, basta installare i package scaricati
- Su Linux installare dai sorgenti con:

```
sudo make altinstall
```

con la possibilità poi di usare *virtualenv* per creare un ambiente virtuale basato sull'interprete alternativo:

```
virtualenv --python=/usr/bin/python2.6 myvirtualenv
```

L'interprete interattivo

- Per fare dei test di può usare l'interprete CLI in maniera interattiva:

```
$ python
```

```
Python 2.7.5 (default, Jun 24 2015, 00:41:19)
```

```
[GCC 4.8.3 20140911 (Red Hat 4.8.3-9)] on linux2
```

```
Type "help", "copyright", "credits" or "license" for more information.
```

```
>>> a = 10
```

```
>>> b = a + 1
```

```
>>> a, b
```

```
(10, 11)
```

```
>>> s = """Linea 1
```

```
... Linea 2"""
```

```
>>> s # Rappresentazione con codici di escape
```

```
'Linea 1\nLinea 2'
```

```
>>> print(s) # Come viene stampata
```

```
Linea 1
```

```
Linea 2
```

```
>>> exit
```

```
Use exit() or Ctrl-D (i.e. EOF) to exit
```

Gestione degli errori

- In Python gli errori sono gestiti in modo chiaro ed efficace

```
>>> while True print('Hello world')
File "<stdin>", line 1
    while True print('Hello world')
                    ^
```

SyntaxError: invalid syntax

```
>>> while True: pritrn('Hello world')
...
```

Traceback (most recent call last):

```
File "<stdin>", line 1, in <module>
NameError: name 'pritrn' is not defined
```

Bash

```
$ while true do echo "Hello world"; done
bash: syntax error near unexpected token `done'
$ while true; do eco "Hello world"; done
bash: eco: command not found...
...
bash: eco: command not found...
^C
```

La funzione help

- Visualizza informazioni su funzioni, istruzioni e moduli nella CLI dell'interprete interattivo:

```
>>> help('print')
```

```
The ``print`` statement
```

```
*****
```

```
print_stmt ::= "print" ([expression ("," expression)* [",,"]  
                        | ">>" expression [(",," expression)+ [",,"]])
```

``print`` evaluates each expression in turn and writes the resulting

...

form, the subsequent expressions are printed to this file object. If the first expression evaluates to ``None``, then ``sys.stdout`` is used as the file for output.

Variabili d'ambiente

- Variabili d'ambiente per customizzare il funzionamento dall'interprete:

PATH	La classica variabile PATH di sistema
PYTHONHOME	Directory di installazione di Python, generalmente una sottodirectory di <i>lib/</i> o <i>lib64/</i>
PYTHONPATH	Lista di directory separate da ':' su Linux e ';' su Windows, dove Python cerca i moduli da importare
PYTHONSTARTUP	Sorgente Python da eseguire ogni volta che viene lanciato l'interprete interattivo

- Non è necessario specificare queste variabili espressamente per far funzionare l'interprete

Struttura lessicale

- Un programma Python è suddiviso in **linee logiche**
- Ogni linea logica può essere definita mediante più **linee fisiche**
- Ogni linea fisica può terminare un **commento**, che inizia con il carattere '#'
- Una linea che contiene solo spazi e/o un commento è considerata una **linea bianca**
- La struttura dei blocchi è definita con l'**indentazione**

Come indentare i blocchi

- Usare spazi per indentare, non *tab*!!
- Lo stile PEP8 prevede indentazione fissa con **4 spazi**, ma si può scegliere un numero di spazi diverso
- Comunque si scelga è auspicabile mantenere un'indentazione fissa per tutto il file

```
# Esempio di indentazione corretta
```

```
if pwd == 'apple':
```

```
    print('Logging on ...')
```

```
else:
```

```
    print('Incorrect password.')
```

```
print('All done!')
```

```
# Esempio di indentazione errata!!
```

```
a = 1
```

```
    b = 2
```

```
c = 3
```

Linee logiche e fisiche

- La continuazione di una linea logica alla linea fisica successiva si verifica quando:
 - Si termina la linea fisica con un backslash ('\')
 - Se ci sono parentesi aperte
 - Se si sta definendo una stringa multilinea (3 apici)
- Diversamente la linea logica termina implicitamente
- Si può usare il ';' per definire più istruzioni non strutturare su una linea fisica (si chiamano *compound statements*, PEP8 ne scoraggia l'uso tranne in casi particolari)

```
with open('/path/to/some/file/you/want/to/read') as file_1, \  
    open('/path/to/some/file/being/written', 'w') as file_2:  
    file_2.write(file_1.read())
```

```
foo = long_function_name(var_one, var_two,  
                          var_three, var_four)
```

```
if foo == 'blah': do_blah_thing()    # Accettabile  
do_one(); do_two(); do_three()     # Meglio di no!
```

Keywords di Python

Identificativi riservati del linguaggio (33):

<code>False</code>	<code>def</code>	<code>if</code>	<code>raise</code>
<code>None</code>	<code>del</code>	<code>import</code>	<code>return</code>
<code>True</code>	<code>elif</code>	<code>in</code>	<code>try</code>
<code>and</code>	<code>else</code>	<code>is</code>	<code>while</code>
<code>as</code>	<code>except</code>	<code>lambda</code>	<code>with</code>
<code>assert</code>	<code>finally</code>	<code>nonlocal</code>	<code>yield</code>
<code>break</code>	<code>for</code>	<code>not</code>	
<code>class</code>	<code>from</code>	<code>or</code>	
<code>continue</code>	<code>global</code>	<code>pass</code>	

Numero di keyword

Linguaggio	Numero di keyword
Python	33
Java	50
Go	25
C++	~80
Ruby	39
PHP	58 (+ 11 con PHP 7)
Perl	40
JavaScript	68
Rust	52
Bash	16

N.B.: Il numero di keyword ha un valore meramente indicativo sulla complessità di un linguaggio, dato che poi poi le keyword sono coadiuvate da funzioni built-in ...

Funzioni built-in

Ci sono una serie di funzioni predefinite (76):

abs()	delattr()	globals()	locals()	property()	str()
all()	dict()	hasattr()	long()	range()	sum()
any()	dir()	hash()	map()	raw_input()	super()
basestring()	divmod()	help()	max()	reduce()	tuple()
bin()	enumerate()	hex()	memoryview()	reload()	type()
bool()	eval()	id()	min()	repr()	unichr()
bytearray()	execfile()	input()	next()	reversed()	unicode()
callable()	file()	int()	object()	round()	vars()
chr()	filter()	isinstance()	oct()	set()	xrange()
classmethod()	float()	issubclass()	open()	setattr()	zip()
cmp()	format()	iter()	ord()	slice()	__import__()
compile()	frozenset()	len()	pow()	sorted()	
complex()	getattr()	list()	print()	staticmethod()	

N.B.: I nomi delle funzioni predefinite non sono keyword riservate

Operatori e delimitatori

- Operatori per espressioni logiche e numeriche:

+	-	*	/	%	**	//	<<	>>	&
	^	~	<	<=	>	>=	<>	!=	==

- Delimitatori usati negli statement:

()	[]	{	}	,	:
.	`	=	;	+=	-=	*=	/=
//=	%=	&=	=	^=	>>=	<<=	**=

- Altri caratteri speciali:

'	"	#	\
---	---	---	---

Libreria standard di Python

- La libreria standard copre molti argomenti e permette di semplificare e velocizzare la scrittura di programmi Python
- Non vale la pena studiarla tutta, basta sapere che cosa c'è in generale, in modo da poter usare i moduli giusti quando serve
- Più di 250 moduli, suddivisi in più di 30 sezioni
- Ci sono moduli specifici per una sola piattaforma, ad esempio per Unix c'è una sezione intera con 16 librerie
- Docs: <https://docs.python.org/library/index.html>
 - Python 2: <https://docs.python.org/2/library/index.html>
 - Python 3: <https://docs.python.org/3/library/index.html>

Ambienti di sviluppo

- **IDLE** è l'IDE base fornito con la distribuzione
 - Incluso nel pacchetto python-tools
 - Leggero, usabile anche in remoto abilitando il forward X11
- **PyCharm** (free la versione Community)
- **Vim**
- Eclipse con PyDev
- Komodo Edit
- Emacs con pymacs
- Ninja-IDE
- Sublime Text

Sorgenti Python con VIM

- Per editare facilmente sorgenti Python con VIM (Visual editor iMproved) è meglio applicare alcune impostazioni che escludono l'inserimenti di caratteri TAB
- Per applicare le impostazioni solo a sorgenti Python creare il file `~/.vim/ftplugin/python.vim` con le seguenti configurazioni:

```
set tabstop=8
set expandtab
set shiftwidth=4
set softtabstop=4
```
- In alternativa le impostazioni possono essere fatte manualmente con un comando:

```
:set tabstop=8 expandtab shiftwidth=4 softtabstop=4
```
- Oppure inserendo nel file il seguente commento:

```
# vim: tabstop=8 expandtab shiftwidth=4 softtabstop=4
```
- Pagina Wiki: <https://wiki.python.org/moin/Vim>

Set di caratteri file sorgenti

- Normalmente in Python 2 un file sorgente è considerato **ASCII**, mentre in Python 3 il default è **UTF-8**
- Per far usare uno specifico charset mettere nella prima linea del file:

```
# -*- coding: <encoding name> -*-
```

oppure come seconda linea se si specifica la *shebang*:

```
#!/usr/bin/python
```

```
# -*- coding: <encoding name> -*-
```

- Per specificare un set UTF-8 si potrebbe in alternativa utilizzare la codifica BOM del file, ma è sconsigliato