Python: una panoramica del linguaggio

Paolo Mossino
mox79@gmx.it

Cos'è Python

Python è un linguaggio di programmazione interpretato di alto livello (*VHLL*), con 13 anni di sviluppo alle spalle. Non è solo un linguaggio di scripting.

Python è molto potente e flessibile, ma nel contempo semplice da usare. Python è facile da imparare, leggere e usare.

Python è estensibile in C/C++, in Java (Jython) e molti altri linguaggi.

Python è Open Source (certificato OSI).

Python è disponibile su una grande varietà di piattaforme: Unix/Linux, Windows, Mac, PalmOS, WindowsCE, RiscOS, VxWorks, QNX, OS/2, OS/390, AS/400, PlayStation, Sharp Zaurus, BeOS, VMS...

Un po' di storia

Python è stato creato agli inizi del 1990 da Guido van Rossum al CWI (Stichting Mathematisch Centrum) in Olanda come successore di un linguaggio chiamato ABC. Guido, nominato BDFL (Benevol Dictator for Life) rimane il principale autore di Python, anche se include molti contributi esterni.

Il nome Python deriva dalla commedia "The monty Python's Flying Circus". Il logo è conseguente alla scelta del nome.

La versione attuale è la 2.3.3, rilasciata tra il 19 dicembre 2003.

Python viene è sia compilato che interpretato.

Compilatore: analizza il codice sorgente dello script e lo traduce in un linguaggio intermedio detto byte-code. La compilazione avviene in modo automatico e, se possibile, viene salvato un file (con estensione .pyc o .pyo) contenente solo il bytecode.

Interprete: implementazione software della macchina virtuale in grado di interpretare il byte-code prodotto dal compilatore.

Esistono varie versioni dell'interprete python, la più nota CPython (detta semplicemente Python). E' scritta in ISO-C ed è in grado di funzionare su un gran numero di piattaforme.
Esistono anche altre versioni, tra le quali possiamo citare Jython (scritto in Java, funziona ovunque sia disponibile una JVM) e Python.NET.

La syntassi di Python ricorda moltissimo lo pseudocodice e la lingua parlata (inglese). È stato definito “pseudocodice eseguibile”.

Alcune caratteristiche:

- tutto è un oggetto, ogni oggetto fa un tipo
- semantica per riferimento
- linguaggio multiparadigma (OO, procedurale, funzionale, ...)
- ereditarietà singola e multipla
- *overloading* degli operatori
- polimorfismo
- utilizza l'indentazione per la struttura dei blocchi
- tipizzazione dinamica, tipizzazione forte
- gestione delle eccezioni

Struttura di un sorgente Python

Un programma Python è composto da linee logiche, ciascuna composta da una o più linee fisiche.

L'indentazione non è semplicemente un modo per rendere il proprio codice leggibile e mantenibile, ma un requisito fondamentale del linguaggio, utilizzato per rendere i blocchi di codice.

> “Life is better without braces”.
> -- Bruce Eckel

In Python esiste il supporto per questi tipi:

- *int*: intero, dimensione: 32 bit
- *long*: intero in precisione arbitraria
- *float*: numero in virgola mobile, standard IEEE, dimensione: 64 bit
- *complex*: numero complesso, come 2 *float* per la parte reale e immaginaria
- *string*: sequenza di caratteri, ascii a 8 bit o unicode
- *None*: oggetto nullo
- *callable*: qualunche oggetto che possa essere chiamato (es. funzioni)
- *boolean*: tipo di dato booleano (introdotto in Python 2.3)

Nota: I valori sulla precisione dipendono dall'architettura.

I numeri e le stringhe sono immutabili: le operazioni su questi generano nuovi oggetti (salvo alcune eccezioni).

Per compatibilità con le versioni precedenti, *True* e *False*, quando necessario, vengono valutati come, rispettivamente, *1* e *0*.

In Python esistono questi tre tipi di contenitori:

- *tuple*: sequenza immutabile di oggetti arbitrari
- *liste*: sequenza mutabile di oggetti arbitrari
- *dizionari*: collezione di oggetti arbitrari non ordinata, indicizzata tramite chiave

Nota: Le liste sono implementate come array dinamici, non come liste linkate. I dizionari sono implementati come tabelle di hash.

Le sequenze sono ordinate, ed è possibile accedere ad un particolare elemento della sequenza in base alla sua posizione o ad un'intera sottosequenza: **lista[start:end:step]**.
Nota: il primo elemento ha indice 0; *end* è il primo elemento escluso, non l'ultimo considerato.

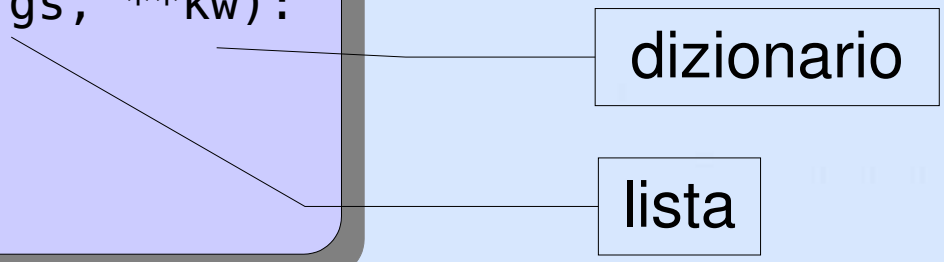
```
if condizione1:
    blocco1
elif condizione2:
    blocco2
else:
    blocco3
```

```
while condizione:
    blocco1
else:
    blocco2
```

```
for bersaglio in iterabile:
    blocco1
else:
    blocco2
```

```
break
continue
```

```
def funzione (a, d=val, *args, **kw):
    “docstring”
    codice
    return risultato
```




```
class classe (parent1, parent2):
    “docstring”
    variabili_di_classe
    def __init__(self):
        codice_inizializzazione
    def __add__(self, other):
        codice
    def metodo(self, a, d=val, *args
        return risultato
```

Esempio: classi

```
class Stack(object):
    def __init__(self):
        self.items = []

    def push(self, value):
        self.items.append(value)

    def pop(self):
        top = self.items[-1]
        del self.items[-1]
        return top

    def __add__ (self, other):
        self.push(other)

    def top(self):
        return self.items[-1]
    top = property(top)

    def empty (self):
        return len(self.items) == 0
```

```
>>>s = Stack()
>>>s.push(10)
>>>s+5
>>>s.empty()
False
>>>s.pop()
5
>>>s.top
10
>>>s.pop()
10
>>>s.empty()
True
```

```
try:
    codice
except (eccezioneX, eccezioneY), target:
    codice
else:
    codice
```

```
raise esp1, esp2
```

```
assert cond, esp
```

EAFP – it's **e**asy to **a**sk **f**orgive than **p**ermission

```
try:
    codice
finally:
    codice
```

- usare le espressioni dove rendono il programma più semplice e leggibile
- mantenere il blocco del *try* il più stretto possibile
- le eccezioni interrompono il flusso del programma e risalgono lo stack fino a trovare un gestore
- mai intercettare un'eccezione senza specificare il tipo di eccezione

Ciascun file Python corrisponde ad un modulo.

I moduli possono essere importati all'interno di altri moduli per utilizzarne le funzionalità.

- `import string; print string.join(L)`
- `from string import join; print join(L)`
- `import string as s; s.join(L)`

Un package è una collezione di moduli organizzati in una directory, che a sua volta può contenere altri package. Requisito: la presenza di un file *__init__.py*

La sintassi per l'importazione è analoga a quella dei moduli.

Batterie incluse

Python viene fornito con un'ampia libreria standard, composta da oltre 300 moduli di cui la maggiorparte disponibili ovunque.

Questi moduli coprono moltissime tematiche, come connessioni di rete, matematica, interfaccia verso il sistema operativo sottostante, gestione del testo, thread.

Python viene fornito con un modulo di interfaccia a Tk, che permette di scrivere semplici interfacce grafiche in modo portabile.

Oltre a questa ampia libreria standard, sono disponibili molte estensioni di terze parti.

Estensioni di terze parti

Esistono moltissimi moduli Python forniti da terze parti:

- Interfacce grafiche: PyGTK, wxPython, PyQT, MFC (Windows), ...
- DB: per i principali DB disponibili, esistono interfacce Python (tutte rispecchiano il *Python Database API Specification v2.0*), per esempio MySQLDb, psycopg, mxODBC, DB2, ...
- Gadfly
- Numeric/numarray, ScientificPython, ...
- Python Image Library (PIL)
- PyGame
- PyX, Reportlab, ...
- Twisted (framework asincrono ad eventi, inoltre supporta un numero notevole di protocolli di rete)
- XML: PyXML, 4Suite, PyRXP, ...
- Egenix-mx (in particolare mxDateTime)
- Zope, CherryPy, Quixote, Webaware, ...

Cos'è Python

Python è un linguaggio di programmazione interpretato, iterattivo e object oriented.

E' un linguaggio che unisce varie caratteristiche che possono apparire contradditorie: semplice e potente,

doctest – Verifica che le *docstring* rappresentino la realtà

unittest – Unit testing framework (chiamato anche *PyUnit*)
La versione Python di Smaltalk testing framework di Kent Beck (da cui deriva anche JUnit, dello stesso autore).

inspect – Ispeziona gli oggetti in esecuzione
Analizza moduli, classo, metodi, funzioni, traceback, oggetti di tipo frame (stack dell'interprete) e oggetti di tipo codice.

traceback – richiama e stampa i traceback dello stack

pdb – il debugger di Python
Un source code debugger iterattivo (è anche in grado di effettuare debug post mortem.

profile – profiler
pstat – analizza le informazioni del profiler

Quick Sort – algoritmo di ordinamento di una lista

```
def qsort (L):
    if len(L) <= 1: return L
    return qsort( [lt for lt in L[1:] if lt <  L[0]] ) +\
           L[0:1]                                      +\
           qsort( [gt for gt in L[1:] if gt >= L[0]] )
```

Semplice, elementare ed elegante implementazione del quick sort, nella sua versione più basilare.

WGet – semplice clone di wget

```
import sys, urllib

def reporthook (*a): print a

for url in sys.argv[1:]:
    i = url.rfind('/')
    file = url[i+1:]
    print url, '->', file
    urllib.urlretrieve(url, file, reporthook)
```

Ecco come implementare un semplice downloader in Python... piccolo sostituto di Wget dove non è disponibile. Nulla di paragonabile quanto a funzionalità, ma dimostra egregiamente cosa si può fare in una manciata di linee di codice Python.

Printf – implementazione di printf in Python

```
import sys

printf = lambda fmt, *args: sys.stdout.write(fmt % args)

if __name__ == '__main__':
    for i, arg in enumerate(sys.argv[1:]):
        printf("%d) %s\n", i, arg)
```

Emuliamo in Python la funzione C printf.

Le *list comprehension* (possiamo tradurlo come: “costruzione di lista”) sono una forma alternativa per indicare un ciclo su un oggetto iterabile che deve produrre come risultato una lista.

```
[esp for bersaglio in iterable clausola]
```

Nulla di difficile, vediamo un semplice esempio: vogliamo ottenere i quadrati di tutti i numeri divisibili per 7 nell'intervallo 12..472.

```
[x**2 for num in range(12,473) if x%7 == 0]
```

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Siti web di interesse (internazionali):

- *http://www.python.org*
- *http://www.zope.org*
- *http://www.oreilly.com/catalog/pythonian/chapter/ch04.pdf*
- *http://diveintopython.org*
- *http://www.onlamp.com/pub/a/python/2004/02/05/learn_python.html*
- *http://www.artima.com/intv/python.html*
- *http://www.artima.com/intv/aboutme.html*
- *http://www.artima.com/intv/prodperf.html*
- *http://www.pythonology.com*
- *http://aspn.activestate.com/ASPN/Python*
- *http://py.vaults.ca*
- *http://www.ferg.org/projects/python_java_side-by-side.html*
- *http://www.pyj.it*

Gruppi di discussione: *comp.lang.python*, *comp.lang.python.announce* e *it.comp.lang.python*