HapPy Python
Corso di programmazione in Python
Lezione 1

a cura del
Neapolis Hacklab

Perchè un corso di Python ?

- Uno dei modi per contribuire alla diffusione del software libero è, ovviamente, quello di sviluppare, sia programmando nuovi software che migliorando software esistenti
- Programmare però può sembrare troppo complicato a chi non ha mai sviluppato software
- In effetti la programmazione richiede studio ed esercizio
- Molti linguaggi sono difficili da imparare e richiedono una conoscenza approfondita dell'architettura dei computer

Perchè un corso di Python ?

- Python facilità il compito del programmatore sollevandolo dalle operazioni di basso livello tipiche di linguaggi come C e Pascal
- Seguendo alcune semplici regole di sviluppo è possibile creare software multi piattaforma anche se non si conosce nel dettaglio l'architettura su cui sarà eseguito
- Facilitare lo sviluppo del software significa permettere ad un numero maggiore di persone di contribuire a progetti OpenSource

A chi è rivolto il corso ?

- A chiunque voglia sviluppare software open
- A chi vuole scoprire un nuovo linguaggio
- A chi vuole riscoprire il piacere di programmare
- A chi vuole un linguaggio che gli permetta di concentrarsi maggiormente sulle esigenze che il software deve soddisfare più che al modo si svilupparle

Nota importante !

A differenza di molti altri linguaggi Python unisce la facilità di sviluppo alla flessibilità, stabilità e velocità di esecuzione.

Python è utilizzato oggi in moltissimi campi dalle applicazioni desktop allo sviluppo di application server, dalle applicazioni multimediali alla gestione di server.

Python è un linguaggio detto **General-Purpose** ovvero adatto a svariati campi di applicazione

Cos'è Python ?

- Python è un linguaggio di programmazione ad alto livello
- E' **OpenSource**: rilasciato con licenza PSF (Python Software Foundation License), GPL compatibile
- Ha una **sintassi minimale**
- E' corredato da una **libreria standard vasta** e completa

Cos'è Python ?

- Gestione tipi **dinamica** e **forte**
- Gestione automatizzata della memoria (Garbage Collection)
- E' multi paradigma: permette la programmazione iterativa, object-oriented e funzionale.

Breve storia

- Concepito nel 1980 da Guido Van Rossum nel CWI (istituto di ricerca nazionale olandese) come successore del linguaggio ABC
- Nel 1991 viene rilasciato il codice della versione 0.9 con molte funzionalità ancora da sviluppare
- Nel 1994 viene rilasciata la 1.0 contenente la maggior parte delle funzionalità migliori di Python ancora presenti oggi

Breve storia

- Nel 2000 viene rilasciata la 2.0 che conferisce al linguaggio un aumento delle prestazioni e della stabilità
- L'ultima versione rilasciata è la 2.5 che aggiunge molte piccole novità alla sintassi del linguaggio
- Tutto il codice che sarà mostrato nel seguito del corso è pienamente compatibile con la versione precedente (2.4) di Python
- Saranno evidenziati utilizzi di funzionalità appartenenti dalla versione 2.5 in poi

Utilizzi

- Web: Zope application server, YouTube, Google, Bittorrent
- Sicurezza: molti software che eseguono test di sicurezza sono scritti in Python
- Grafica 3D: Maya e Blender integrano Python per gli script di automazione
- Video game: molti OpenSource e commerciali integrano Python per programmare il comportamento dei vari elementi del gioco

Utilizzi

- Python è un componente standard nell'installazioni di FreeBSD, OpenBSD, NetBSD, Mac OSX, Ubuntu, ecc.
- Ubuntu lo utilizza per creare le interfacce grafiche di gestione
- Il software di installazione di Redhat linux e Fedora linux (chiamato Anaconda) è scritto interamente in Python
- Il package manager di gentoo (emerge) è scritto in Python.
- Ecc. ecc. ecc. ecc.

Sintassi

- Python ha una **sintassi minimale**: non utilizza simboli o parole chiave per delimitare i blocchi di codice
- **L'identazione fa parte della sintassi del linguaggio**
- Il risultato è codice ad alta leggibilità

Gestione tipi di variabili

- **Gestione dinamica**: le variabili possono cambiare tipo nel corso dell'esecuzione del codice. Non viene quindi effettuato alcun controllo *statico* sul tipo di valore assegnato ad una variabile.
- **Tipizzazione forte**: le operazioni tra tipi di dati diversi devono essere specificate esplicitamente nel codice (casting). Non vengono effettuate conversioni automatiche dei tipi delle variabili.

Python Standard Library

- E' la libreria standard di Python che contiene moduli delle più svariate funzionalità: applicazioni web, interfacce grafiche, database, aritmetica, espressioni regolari, email, crittazione, xml, calendario, ecc.
- L'installazione di Python comprende sempre la libreria standard considerata elemento fondamentale: la cosidetta filosofia **Batteries Included**.

Python Standard Library

- Ogni modulo della libreria deve essere compatibile con tutte le piattaforme su cui Python può essere eseguito.
- Esistono però funzionalità specifiche di ogni sistema operativo non presenti su altri sistemi. Per queste funzionalità la Python Standard Library include altri moduli specifici per ogni piattaforma evidenziando nella documentazione l'utilizzo specifico per un sistema operativo.

Estensione di Python

- L'architettura di Python è stata pensata per permettere una semplice estensione del linguaggio. E' possibile programmare moduli di Python in linguaggio C, C++, Java. Ciò permette di programmare parti dell'applicazione in un linguaggio più performante come il C per ottimizzare le prestazione del software.
- Python può essere utilizzato anche in modalità **Embedded** ovvero intergrato in altre applicazioni alle quali aggiunge un motore di scripting interno.

- Dopo questa panoramica su Python vediamo ora una riga di codice... rigorosamente un *HelloWorld*

```
print "Hello world!"
```

- Ovviamente l'esecuzione di questa riga di codice mostra a schermo il testo **Hello world!**

Esecuzione di programmi in Python

- Per eseguire codice in Python basta chiamare l'interprete seguito dal nome del file:
  - python helloworld.py
- Per ogni file .py Python crea un file .pyc contenente il codice compilato (o meglio semicompilato)
- Ciò velocizza il caricamento e l'esecuzione dei file evitando la reinterpretazione del codice Python

Un altro semplice esempio

```
nome = raw_input("Ciao, come ti chiami? ")
if nome:
   print "Ciao", nome
else:
   print "Devi dirmi il tuo nome"
```

- *raw_input* attende un input da tastiera e lo restituisce quando viene premuto invio
- Se la variabile *nome* non è vuota viene visualizzato “Ciao *nome*”
- Altrimenti da un altro messaggio

Un altro semplice esempio

```
nome = raw_input("Ciao, come ti chiami? ")
if nome:
   print "Ciao", nome
else:
   print "Devi dirmi il tuo nome"
```

- Notare la sintassi: le istruzioni contenute nell'istruzione *if* e *else* sono associate al relativo blocco dagli spazi di identazione (in questo caso una tabulazione ma sono accettati anche spazi purchè coerenti)
- I due punti indicano l'inizio di un blocco di codice con una o più righe di istruzioni

Tipi di dati in Python

- Python ha pochi tipi di dati nativi semplici tipici di un linguaggio di programmazione:
  - Numerici: *int, float, long, complex*
  - Stringhe: *str, unicode*
  - Booleano: *bool*
  - Array: list, *tuple, dictionary, set*
  - Classi: *type*

Tipi di dati numerici - int

- Il tipo intero in python corrisponde al *long* del linguaggio C o Java
- Utilizza 4 byte e possiede il segno
- Può gestire numeri da -2 miliardi circa a +2 miliardi circa

```
a = 10

b = 1230

c = a + b
```

Tipi di dati numerici - long

- Il tipo di dati *long* è un numero intero con precisione illimitata
- Può contenere numeri di qualsiasi dimensione
- L'unico limite (teorico) è la memoria della macchina
- I tipi int sono automaticamente convertiti in long quando vengono superati i relativi limiti

```
a = 1500320l

b = 500l

c = a + b
```

Tipi di dati numerici - float

- Il tipo dati *float* di python corrisponde al tipo *double* del C
- Utilizza 8 byte di memoria
- Come noto per il C la sua precisione nei calcoli dipende dalla macchina su cui vengono eseguiti

```
a = 5.2

b = 102.0

c = a + b
```

Tipi di dati stringa - str

- Il tipo di dati *str* rappresentano semplici stringhe come per qualsiasi altro linguaggio
- La maggior parte delle funzioni di manipolazione delle stringhe in Python sono metodi del tipo *str*.

```
nome = "Linguaggio di programmazione Python"
len(nome)                          # => 35
nome.startswith("Ling")            # => True

nome.upper()   # => "LINGUAGGIO DI PROGRAMMAZIONE PYTHON"

nome.replace("Python", "OpenSource")  # => "Linguaggio di
programmazione OpenSource"
```

Tipi di dati boolean - bool

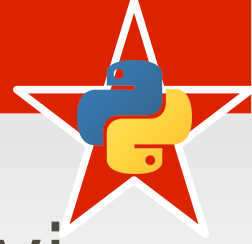
- I tipi di dati *bool* si definiscono assegnando alle variabili le parole chiave *True* o *False*
- Fare attenzione che le lettere iniziali siano maiuscole
- Python è *case-sensitive*, ovvero fa differenza tra maiuscole e minuscole, per cui la variabile *nome* è diversa da *NOME*.

Tipi di array in Python

- Liste: array dinamici corredati di tutti i metodi per effettuare le più comuni operazioni
- Tuple: array statici. Non possono essere modificati nell'esecuzione del software
- Dizionari: sono i comuni array associativi tipici di molti linguaggi
- Set: array dinamici contenenti solo elementi unici

Liste

- Le liste sono semplici array
- Ogni elemento è identificato da un indice numerico con base 0. Gli elementi possono non essere omogenei
- Su di essi è possibile effettuare tutte le operazioni base:
  - Creazione: **lista = [1, 5, 10, 20, 8, 5]**
  - Lettura: **valore = lista[2]** (ritorna 10)
  - Aggiunta: **lista.append(34)**
  - Eliminazione: **del lista[3]** (elimina il valore 20)
  - Modifica: **lista[0] = 7** (modifica il primo elem.)



- E' possibile utilizzare anche indici negativi che indicano la lettura degli elementi partendo dalla fine:
  - **lista = [1, 5, 10, 20, 8, 5]**
  - **lista[-1] = 5**
  - **lista[-3] = 20**
- Il controllo di appartenenza di un elemento ad un lista viene effettuato dall'istruzione **in**.
  - **5 in lista** # True
  - **11 in lista** # False

- Le liste supportano l'operazione di addizione:
  - **lista = [1, 5, 10, 20, 8, 5]**
  - **lista2 = [11, 12, 15]**
  - **lista + lista2 = [1, 5, 10, 20, 8, 5, 11, 12, 15]**
- e di moltiplicazione
  - **lista = [1, 5, 10]**
  - **lista * 3 = [1, 5, 10, 1, 5, 10, 1, 5, 10]**
- Per contare gli elementi contenuti si utilizza la funzione len: **len(lista)** # 9

- Partendo da una lista è possibile ricavare un'altra lista che è una porzione della prima:
  - **lista = [5, 10, 20, 1, 8, 9, 11]**
  - **lista[0:2] = [5, 10]**
  - **lista[3:4] = [1]**
  - **lista[3:-2] = [1, 8, 9]**
- I valori restituiti sono quelli che partono dal primo indice compreso al secondo indice escluso, separati da due punti

Tuple

- Le tuple sono array che non permettono la modifica durante il loro ciclo di vita
- Hanno vari campi di applicazione
- Le tuple hanno prestazioni nettamente superiori a quelle delle liste
- Permettono di proteggere il software da eventuali attacchi XSS (Cross Site Scripting)
- Parte delle operazioni delle liste sono applicabili anche alle tuple

- Creazione: **tupla = (4, 2, 50, 10, 30)**
  - Le parentesi sono opzionali, ma è consigliato utilizzarle per aumentare la leggibilità del codice
  - Sono necessarie quando la virgola può essere ambigua: ad esempio se la tupla è utilizzata come argomento di una funzione
  - Per creare una tupla con un singolo elemento è necessario che l'elemento sia seguito dalla virgola. Ad esempio: **tupla = (10,)**

- Parte delle operazioni delle liste sono applicabili anche per le tuple:
  - **tupla = (1, 5, 9, 2, 10, 51)**
  - **tupla[1]** # 5
  - **tupla[1:3]** # (5, 9)
  - **tupla + (43, 20)** # (1, 5, 9, 2, 10, 51, 43, 20)
  - **len(tupla)** # 6

- Esempi di applicazioni delle tuple sono:
  - Ritornare più di un valore da una funzione
    - **return (a, b, c)** # restituisce una tupla con tre valori contenuti nelle variabili a, b e c
    - Ritornando una tupla invece che una lista si risparmierà memoria e si velocizzerà l'esecuzione del codice
  - Si può scambiare i valori di due variabili senza utilizzare una terza variabile:
    - **(a, b) = (b, a)** anziché **temp = a; a = b; b = a**

Dizionari

- I dizionari sono array associativi
- Gli indici degli elementi possono essere di qualsiasi tipo (tranne qualche eccezione)
- Si creano utilizzando le parentesi graffe:
  - **diz = {}**
  - **diz[“nome”] = “neapolis”** # aggiunge un elemento o lo aggiorna se già esistente
  - **diz[“cognome”] = “hacklab”**
  - **diz[“nome”]** # “neapolis”
  - **diz = {“nome”: “neapolis”, “cognome”: “hacklab”}**

- Gli indici possono essere numerici, stringhe, tuple e istanze*:
  - **diz[10] = “numero 10”**
  - **diz[21.5] = “nuovo elemento con chiave float”**
  - **diz[“abc”] = “def”**
  - **diz[“ok”] = 99**
  - **diz[(1,2,4)] = 5702.31**
- *(Istanze di classe, seconda lezione)

- Metodi utili:
  - **diz.keys()**: restituisce una lista con tutte le chiavi
  - **diz.values()**: restituisce una lista con tutti i valori
  - **diz.items()**: restituisce una lista in cui ogni elemento è una tupla di due elementi, il primo è la chiave ed il secondo è il valore
- Eliminare un elemento: **del diz[chiave]**. Ad esempio **del diz[“nome”]**
- Contare gli elementi di un dizionario: **len(diz)**

Definire funzioni

- Le funzioni in Python si definiscono con il comando **def** seguito dal nome della funzione
- Segue poi l'elenco dei parametri tra parentesi (opzionale)
- Infine i due punti, che indicano l'inizio di un blocco e l'elenco delle istruzioni. Attenzione all'identazione che in Python è fondamentale

```
def funzione():
    print "La mia prima"
    print "funzione in Python"

funzione()
=> La mia prima
=> funzione in Python
```

Definire funzioni – i parametri

- I parametri vengono specificati tra le parentesi tonde accanto al nome della funzione
- Essendo Python un linguaggio dinamico non bisogna specificare il tipo di variabile. Esso cambiarà in base ai parametri che verranno passati

```
def ciao(nome):
   print "Ciao " + nome + "!"

ciao("Fabio")
=> Ciao Fabio!
```

Definire funzioni – i parametri

- I parametri opzionali vengono definiti assegnando il relativo valore di default

```
def potenza(x, y=2):
    return x**y

potenza(3)
=> 9

potenza(3, 3)
=> 27
```

- **x**y** è in Python l'elevazione a potenza x^y
- **return** è il classico comando che indica quale valore la funzione deve ritornare

Definire funzioni – i parametri

- I parametri di una funzione possono essere passati anche per “nome”:

```
def nuovoUtente(username, nome="", cognome="", id=None):
    print "Inserimento nuovo utente nel database..."
    print "Username:", username
    if nome: print "Nome:", nome
    if cognome: print "Cognome:", cognome
    if id: print "ID:", id

nuovoUtente(username="myuser", cognome="mylastname",
    nome="myname")

nuovoUtente("myuser2", id="0001")
```

- Entrambe le chiamate alla funzione **nuovoUtente** sono valide.

Definire funzioni – i parametri

- Il numero di parametri di una funzione può essere dinamico:

```
def sommatoria(nome, *elementi):
    print "Verra' calcolata la sommatoria per la variabile:",
        nome
    print "Numero di elementi:", len(elementi)
    somma = sum(elementi)
    print nome, "=", somma

sommatoria("X", 10, 30, 11, 5, 10, 74)
```

- L'asterisco significa che la variabile **elementi** sarà una lista contenente tutti i parametri addizionali
  - La funzione **sum** è una funzione standard di Python che restituisce la somma degli elementi di una lista

Definire funzioni – i parametri

- Anche il numero di parametri passati per nome può essere dinamico:

```
def canzone(nome, *tags, **proprieta):
    print "Canzone:", nome
    print "Tags:", tags
    print "Altro:", proprieta

canzone("Verguenza", "ska", "punk", "rock", autore="Ska-P",
anno="1998")

=> Canzone: Verguenza
=> Tags: [“ska”, “punk”, “rock”]
=> Altro: {“autore”: “Ska-P”, “anno”, “1998}
```

- Il doppio asterisco della variabile **proprieta** indica che essa conterrà tutti i parametri aggiuntivi passati per nome

Ciclo For

- La sintassi del ciclo For è semplice e intuitiva
- Necessita di una lista o una tupla su cui scorrere
- Assegna ad una variabile, uno per volta, ogni elemento della lista ed esegue il codice assegnato

```
nomi = ["Homer Simposon", "Peter Griffin", "Nerd Flanders",
"Monty Python"]

for nominativo in nomi:
   (nome, cognome) = nominativo.split(" ")
   print "Le iniziali di", nominativo, "sono", nome[0] +
      cognome[0]
```

Ciclo For

```
nomi = ["Homer Simposon", "Peter Griffin", "Nerd Flanders",
"Monty Python"]

for nominativo in nomi:
   (nome, cognome) = nominativo.split(" ")
   print "Le iniziali di", nominativo, "sono", nome[0] +
      cognome[0]
```

- Alla variabile **nominativo** viene assegnato ogni valore della lista nomi uno per volta
- Il comando **split** divide una stringa per il separatore specificato come parametro restituendo la lista delle sottostringhe:
  - **“Homer Simpson”.split(“ “)** => **[“Homer”, “Simpson”]**

Ciclo For

```
nomi = ["Homer Simposon", "Peter Griffin", "Nerd Flanders",
"Monty Python"]      # 1

for nominativo in nomi:    # 2
   (nome, cognome) = nominativo.split(" ")     # 3
   print "Le iniziali di", nominativo, "sono", nome[0] +
      cognome[0]     # 4
```

- La riga 3 è una forma *ridotta* per assegnare ogni elemento della lista, restituito dalla funzione split, ad una variabile
- In questo caso a **nome** sarà assegnato il primo elemento, a **cognome** il secondo
- Funzione solo se il numero degli elementi è uguale da entrambe le parti

Formattazione di stringhe

- La funzionalità di formattazione delle stringhe è tanto semplice quanto utile
- Ha lo scopo di costruire stringhe da concatenazioni di una o più variabili
- Il funzionamento è quello classico dei *template* in cui c'è un testo statico e dei *segnaposto* che vengono sostituiti dai valori della variabili passate
- In Python l'operatore di formattazione delle stringhe è **%**

Formattazione di stringhe

- Il tutto necessita di una stringa con gli appositi segnaposti e una tupla con i valori da sostituire nella stringa

```
nome = "Homer"
username = "homers"

saluto = "Ciao %s, il tuo username e' %s." % (nome,
   username)

# => "Ciao Homer, il tuo username e' homers"
```

- I segnaposti si indicano con **%** seguiti da un carattere che indica il tipo di variabile da sostituire
- Al primo segnaposto sarà sostituita la prima variabile della tupla, al secondo segnaposto la seconda variabile della tupla, e così via

Formattazione di stringhe

- %s indica che la variabile da sostituire è una stringa. Nel caso in cui la relativa variabile non fosse una stringa Python la sostituisce con la sua *rappresentazione* in stringa:
  - “ciao %s” % (“Peter”,) # => “ciao Peter”
  - “ciao %s” % (100,) # => “ciao 100”
  - “ciao %s” % ([1,2,3],) # => “ciao [1,2,3]”
  - “ciao %.5s” % (“HomerSimpson”,)
    # => “Homer”
    .5 indica la lunghezza massima della stringa
  - “ciao %5s” % (“no”,) # => “ciao no” (senza il punto indica la lunghezza minima

- **%d** e **%i** formattano numeri interi
- **%5d** indica la stringa risultante deve essere minimo di 5 caratteri. Se la conversione in stringa del numero è minore di 5 caratteri vengono aggiunti degli spazi all'inizio. Se il 5 viene preceduto da uno zero saranno aggiunti degli zeri anché spazi all'inizio del numero
  - “num %5d” % (99,) # => “   99” (tre spazi)
  - “num %05d” % (99,) # => “00099” (tre zeri)

Formattazione di stringhe

- **%f** formatta i numeri in virgola mobile (float)
- **%.3f** formatta il numeri visualizzando massimo tre cifre decimali
- **%5f** o **%05f** come per il numero intero. Non specificando le cifre decimali ne vengono visualizzte 6 di default
- **%010.3f** formatta il numero visualizzando tre cifre decimali e nel caso in cui tutto il numero sia inferiore a dieci caratteri aggiunge degli zeri all'inizio
  - “%010.3f” % (99.34,) # => “000099.340”

Formattazione di stringhe

- Esistono anche altri tipi di segnaposto come: **%x** per visualizzare numeri in esadecimale, **%o** per ottale, **%g** per il formato esponenziale. La sintassi è identica a quella dei segnaposti visti in precedenza. Per ulteriori dettagli riferirsi alla documentazione ufficiale
- La formattazione delle stringhe è utile perchè permette di creare stringhe inserendo valori di altro tipo senza effettuare il casting dei tipi