Rudimenti di Python

senza rodimenti di Python

Contatti

_ Enrico Marino
_ Federico Spini

_ mail:
  _ `(marino|spini)@dia.uniroma3.it`

_ sito:
  _ `dia.uniroma3.it/~(marino|spini)/python`

Eventi

_ Possibili date per gli homework di Grafica Computazionale
  _ venerdì 18 marzo
  _ venerdì 8 Aprile
  _ venerdì 6 Maggio
  _ venerdì 27 Maggio
  _ venerdì 10 giugno

Riferimenti

_ sito ufficiale: http://python.org/

_ tutorial ufficiale: http://docs.python.org/tutorial/

_ documentazione: http://docs.python.org/index.html

Cos'è Python?

- Python è un linguaggio di programmazione ad alto livello
  - OpenSource
    - licenza PSF, GPL compatibile
  - multi paradigma
    - imperativo, funzionale, object-oriented

- Caratteristiche principali
  - sintassi minimale
  - libreria standard vasta e completa
  - gestione tipi dinamica e forte
  - gestione automatizzata della memoria

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
- Nel 1994 viene rilasciata la 1.0
- Nel 2000 viene rilasciata la 2.0
- Oggi convivono le versioni 2.7 e 3.2
  - la versione 3.x non è retrocompatibile

Utilizzi

_ Web
  _ Zope application server, YouTube, Google, Bittorrent

_ Grafica 3D
  _ Maya e Blender per gli script di automazione

_ Videogame
  _ giochi OpenSource e commerciali

Caratteristiche principali
Sintassi

_ Python ha una sintassi minimale
  _ non si usano simboli per delimitare i blocchi di codice
  _ non si usa il '**;**' per delimitare le righe di codice
  _ non si usano '{' e '}' per delimitare i blocchi di codice
  _ si usa il ’#’ per commentare
  _ si usano i '**:**' per indicare l'inizio di un blocco
    _ i blocchi devono essere indentati

```
# commento
if espressione:
    blocco_if_indentato
else:
    blocco_else_indentato
istruzione_flusso_principale
```

Caratteristiche principali
Gestione tipi di variabili

_ Gestione dinamica
  _ le variabili possono cambiare tipo durante l'esecuzione
  _ non viene effettuato alcun controllo statico
    sul tipo di valore assegnato ad una variabile

_ Tipizzazione forte
  _ Il casting dei tipi deve essere esplicito
  _ non vengono effettuate conversioni tra tipi

  _ non si deve dichiarare il tipo delle veriabili

```
# no
str stringa = “ciao”

# si
stringa = “ciao”
```

Python Standard Library

_ Libreria standard di Python
  _ contiene moduli delle più svariate funzionalità
    _ Matematica, RegExp, DataBase, Web, UI, ...

_ ogni modulo della libreria deve essere compatibile con tutte le piattaforme su cui Python viene eseguito

_ include altri moduli specifici per ogni piattaforma specificati nella documentazione

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
- E' possibile programmare parti dell'applicazione
  - in un linguaggio più performante come il C
  - per ottimizzare le prestazione del software
- E' possibile utilizzare Python in modalità Embedded
  - per integrare Python in altre applicazioni
  - per aggiungere un motore di scripting interno

"Hello world!"

```
# il mio primo programma
# questo è un commento
print "Hello world!"
```

_ Per eseguire uno script Python
  _ chiamare l'interprete seguito dal nome del file

```
>>> python helloworld.py
```

_ Per ogni file `.py` Python crea un file `.pyc`
  _ con il codice compilato (o meglio semicompilato)
  _ per velocizzare il caricamento e l'esecuzione del file
  _ per evitare la reinterpretazione del codice Python

Tipi di dati

- Built-in object types:
  - Numerici
    - int, float, long, complex
  - Stringhe
    - str, unicode
  - Booleano
    - bool
  - Array
    - list, tuple, dictionary, set
  - Classi
    - type

Tipi di dati
Numerici

- `type int`
  - numero intero
  - utilizza 4 byte e possiede il segno
  - gestisce numeri da -2 miliardi circa a +2 miliardi circa
  - corrisponde al long in C o in Java

- `type long`
  - numero intero con precisione illimitata
  - può gestire numeri di qualsiasi dimensione
  - l'unico limite (teorico) è la memoria della macchina
  - corrisponde al long in C o in Java

- `type float`
  - corrisponde al tipo double in C

Tipi di dati
Stringhe

- `type str`
  - rappresenta le stringhe come in qualsiasi altro linguaggio

- Operazioni comuni
  - Creazione:
    ```
    stringa = "stringa tra doppi apici”
    stringa = 'stringa tra singoli apici’
    ```
  - Lunghezza: `len(stringa)`
  - Upper case: `stringa.upper()`
  - Replace: `stringa.replace("old", "new")`

Tipi di dati
Booleani

- `type bool`
  - `True` e `False`
  - con le lettere iniziali maiuscole
  - Python è case-sensitive
    - **`t`**`rue` non è uguale a **`T`**`rue`

Gestire Input e Output

- `raw_input` è una funzione
  - attende un input da tastiera
  - restituisce l'input quando viene premuto invio

- `print` è una direttiva
  - stampa la tupla di argomenti che la segue
  - effettua il casting automatico degli elementi in stringhe

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

Controllo di flusso

If

```
x = 'killer rabbit'

if x == 'bunny':
    print 'hello little bunny'
elif x == 'bugs':
     print "what's up doc?”
else:
     print 'Run away! Run away!...'

>>> Run away! Run away!
```

Controllo di flusso
While

```
while True:
    print 'Type Ctrl-C to stop me!'
```

Controllo di flusso
For

```
for x in ["hello", "world"]:
    print x
```

Strutture dati
Gli array

_ Liste (`type list`)

```
lista = [ el_1, ..., el_N ]
```

_ Tuple (`type tuple`)

```
tupla = ( el_1, ..., el_N )
```

_ Dizionari (`type dict`)

```
dizionario = { key_1: el_1, ..., key_N: el_N }
```

_ Insiemi (`type set`)

```
insieme = set(lista)
```

Liste
Caratteristiche

_ Le liste sono semplici array
_ Ogni elemento è identificato da un indice numerico
  _ il primo indice è 0
_ Gli elementi possono non essere omogenei

Liste
Operazioni comuni

_ Creazione: `lista = [1, 5, 10, 20, 8, 5]`
_ Lettura: `valore = lista[2]`
_ Aggiunta: `lista.append(34)`
_ Eliminazione: `del lista[3]`
_ Modifica: `lista[0] = 7`
_ Lettura degli elementi partendo dalla fine: `lista[-1]`
_ Appartenenza di un elemento ad un lista: `5 in lista`
_ Concatenazione di liste: `[1,2,3] + [4,5,6]`
_ Ripetizione di liste: `[1,2,3] * 3`

_ Per saperne di più

```
>>> help(list)
```

Liste
Operazioni comuni

_ Slicing:

```
lista = [5, 10, 20, 1, 8, 9, 11]
lista[0: 2] = [5, 10]
lista[3:-2] = [1, 8]
lista[3:  ] = [1, 8, 9, 11]
lista[ : 2] = [5, 10]
```

_ I valori restituiti sono quelli
  _ dal primo indice (compreso)
  _ al secondo indice (escluso)
  _ separati da due punti

Tuple
Caratteristiche

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

Tuple
Caratteristiche

- Le tuple sono definite da una lista di valori
  - racchiusi tra parentesi tonde
  - separati da virgole

```
tupla = (1, 2, 3)
```

- Le parentesi in genere sono opzionali
  - aumentano la leggibilità del codice
  - sono necessarie quando la virgola può essere ambigua
    - 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

```
tupla = (1,)
```

Tuple
Operazioni comuni

_ Creazione: `tupla = (1, 2, 3, 4, 5)`
_ Lettura valore: `tupla[1]`
_ Lettura intervallo: `tupla[1:3]`
_ Concatenazione: `tupla + (6, 7)`
  _ crea una nuova tupla
_ Lunghezza: `len(tupla)`

_ Per saperne di più

```
>>> help(tuple)
```

Tuple
Esempi di applicazione

_ Ritornare più di un valore da una funzione tramite una tupla
  _ per risparmiare memoria
  _ per velocizzare l'esecuzione del codice

```
return (a, b, c)
```

_ Scambiare i valori di due variabili
  _ senza utilizzare una terza variabile
  _ tramite assegnazione multipla

`a,b=b,a` invece di `temp=a; a=b; b=a`

Dizionari
Caratteristiche

_ I dizionari sono array associativi
  _ mappe chiave-valore
  _ Si creano utilizzando le parentesi graffe

```
diz = {}
diz["nome"] = "Pluto"
diz["corso"] = "Python"
```

oppure

```
diz = {"nome": "Pluto", "corso": "Python"}
```

Dizionari
Indici

_ Gli indici possono essere di qualsiasi tipo
  _ tipi numerici, stringhe, tuple e istanze

```
diz[10] = "elemento con chiave int"
diz[21.5] = “elemento con chiave float"
diz["abc"] = ”elemento con chiave str"
diz["ok"] = 1
diz[(1,2,3)] = 0.5
```

Dizionari
Operazioni comuni

_ Creazione: `diz = { chiave : valore, … }`
_ Lista delle chiavi: `diz.keys()`
_ Lista dei valori: `diz.values()`
_ Lista di tuple (chiave, valore): `diz.items()`
_ Lettura elemento: `diz[chiave]`
_ Eliminazione elemento: `del diz[chiave]`
_ Lunghezza dizionario: `len(diz)`

_ Per saperne di più

```
>>> help(dict)
```

Insiemi
Caratteristiche

- Collezione non ordinata di elementi unici

- Supporta
  - membership testing
  - rimozione duplicati da una sequenza
  - operazioni insiemistiche standard
    - intersezione, unione, differenza e differenza simmetrica

- Non supporta
  - comportamenti tipici dei costrutti sequence-like
  - accesso posizionale, slicing, ...

Insiemi
Operazioni comuni

_ Creazione: `insieme = set([1, 2, 3, 3, 4, 5])`
_ Cardinalità: `len(insieme)`
_ Appartenenza: `x in insieme`
_ Non appartenenza: `x not in insieme`
_ Contenimento: `s.issubset(t)` oppure `s <= t`
`s.issuperset(t)` oppure `s >= t`
_ Unione: `s.union(t)` oppure `s | t`
_ Intersezione: `s.intersection(t)` oppure `s & t`
_ Differenza: `s.difference(t)` oppure `s - t`
_ Differenza simmetrica:
`s.symmetric_difference(t)` oppure `s ^ t`

_ Per saperne di più
`>>> help(set)`

Funzioni
Definire le funzioni

_ Le funzioni si definiscono con il comando **`def`**
  _ seguito dal nome della funzione
  _ seguito dall'elenco dei parametri (opzionale) tra parentesi tonde '**`()`**'
  _ seguito dai due punti '**`:`**' che indicano l'inizio del blocco

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

def saluto(nome):
    print "Ciao " + nome + "!"

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

>>> saluto("Pluto")
>>> Ciao Pluto!
```

Funzioni
Parametri opzionali

_ I parametri opzionali riportano il relativo valore di default

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

>>> potenza(3)
>>> 9

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

_ l'operatore `**` indica l'elevazione a potenza
_ la direttiva return ritorna il valore di output della funzione

Funzioni
Legame dei parametri

_ I parametri possono essere passati anche per “nome”

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

>>> utente("Pj", cognome="Junior", nome="Pluto")

>>> utente("secret_agent", id="007")
```

Funzioni
Parametri addizionali (“finezze”)

_ Il numero di parametri può essere dinamico

```
def sommatoria(nome, *elementi):
    print "Lista: ", nome
    print "Numero elementi: ", len(elementi)
    somma = sum(elementi)
    print "Somma: ", somma

>>> sommatoria("X", 1, 1, 2, 3, 5, 8, 13, 21)
```

_ L'asterisco indica che la variabile elementi
sarà una lista contenente tutti i parametri addizionali

Funzioni
Parametri addizionali (“finezze”)

_ 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("Titolo", "a", "b", "c",
        autore="Autore", anno="2011")
>>> Canzone: Titolo
>>> Tags: ["a", "b", "c"]
>>> Altro: {"autore": "Autore", "anno": "2011"}
```

_ Il doppio asterisco indica che proprieta conterrà tutti i parametri aggiuntivi passati per nome

Moduli
Importare i moduli

- **`from`** *`module`* **`import`** *`metodo`*$_X$*`, attributo`*$_Y$*`, ...`*
  - per importare selettivamente metodi e attributi del modulo

- **`import`** *`module`*
  - per importare tutti i metodi e attributi del modulo

- **`from`** *`module`* **`import *`**
  - per importare tutti i metodi e attributi del modulo
    - direttamente nel namespace locale
    - in modo che siano disponibili direttamente
    - senza qualificare il nome del modulo