Python le Basi

Settembre 2006

Definizioni:

Le Istruzioni
- Sono dei comandi che si vuole impartire alla macchina

Il Terminale
- O comunemente chiamata shell
  e' il client che ci permette di comunicare i nostri comandi alla macchina

Lo script
- E' una sequenza di comandi salvati in un file che verra' poi mandato in esecuzione
  intese come gestione della macchina (piccolo programma che serve al sistemista)

Il Programma
- E' sempre una sequenza di istruzioni che l'elaboratore deve eseguire
  intese come elaborazione dati (applicativo che serve all'utente finale)

Il Linguaggio

Interpretato:
- Compilazione istantanea
- Interattivo
  quando siamo su una shell e impartendo dei comandi
  abbiamo i risultati in tempo reale

Pro:
- Velocita' di apprendimento
- Visualizzazione immediata dei risultati
- Efficenza di sviluppo

Contro:
- Velocita' di esecuzione

Compilato
- Compilazione Differita
  si deve creare un file poi si deve compilarlo
  per ottenere il file eseguibile

Pro:
- Velocita' di esecuzione

Contro:
- Elaborazione dei risultati differita

Le Assegnazioni

```
#  questo e' un commento (il commento viene precedeuto dalla chiave #)

# questa e' una assegnazione semplice
#  (la variabile mia conterra' l'indirizzo di dove e' definita la stringa)
mia = “ciao”

# una riassegnazione
# (la variabile mia conterra' l'indirizzo di dove e' definita l'altra stringa)
mia = “adesso contengo un'altra cosa”

# duplice assegnazione
tua = essa = “nostra”

# assegnazione multipla
mia, tua = 1, 2
#  assegnazione multipla con scambio dei contenuti
mia, tua = tua, mia
```

Le Condizioni

```
if mia > 20:              # condizione
   mia = 10               # eseguita se VERA
   pass                   # istruzione che non fa nulla
else:                     # questo e' facoltativo
   mia = 30               # eseguita se la precedente era FALSA

if mia > 20: mia = 10     # condizione e operazione sulla stessa riga
elif mia < 50: mia = 5    # testata solo se la precedente non e' soddisfatta
```

Condizioni	
== is	uguale
!= <> is not	non uguale
<	minore
>	maggiore
<=	minore o uguale
>=	maggiore o uguale
5<x<7	x compreso

Boolean	
False	0,"",None
True	tutti gli altri casi

- Una particolare attenzione si deve portare nelle operazioni di tipo boolean.
- In una espressione AND viene calcolata la 2a parte solo se, il valore della 1a parte e' VERO.
- In una espressione OR viene calcolata la 2a parte solo se, il valore della 1a parte e' FALSO.

I Numeri

```
int(num)          # intero      (normamente 32bit)
long(num)         # long        (illimitato)
float(num)        # floating    (normalmente 64bit, IEEE)
complex(rea,ima)  # complesso   (due float [parte reale, parte immaginaria])
```

Funzioni Base

```
x + y       # somma
x – y       # sottrazione
x * y       # moltiplicazione
x / y       # divisione
x % y       # resto divisione
x ** y      # x alla potenza di y
```

Operazioni Unitarie

```
+x          # incremento
-x          # decremento
```

Funzioni sui Bit

```
x | y       # x or y
x ^ y       # x xor y
x & y       # x and y
x << y      # shift a Sx di y bit
x >> y      # shift a Dx di y bit
~x          # negato
```

Moduli matematici

```
math        #
cmath       #
operator    #
```

I Cicli

```
while mia > 20:                    # condizione
   mia -= 1                        # eseguita finche' la condizione e' VERA

for i in 1,2,4          :          # condizione (finche' esistono elementi)
   print i ** 2                    # eseguita finche' la condizione e' VERA

for i in xrange (0,6,2)            # condizione (da 0 a 6 a step di 2)
   print i ** 2                    # eseguita finche' la condizione e' VERA

a =  ['Mary', 'had', 'a', 'little', 'lamb']
for i, item in enumerate (a)       # itera sugli indici della sequenza
   print i, item                   # stampa indice ed elemento
```

All'interno dei corpi possono esistere le istruzioni:

```
 break                             # interrompi il ciclo
 continue                          # cicla senza eseguire le istr. seguenti del corpo
 return                            # esce dalla funzione o dal metodo
```

Alla fine del corpo puo' esistere l'istruzione:

```
 else
```

Nota:

```
      xrange(0,6,2) crea una sequenza immutabile     # questo e' un oggetto iteratore
        range(0,6,2) crea una sequenza mutabile      # questa e' una lista
```

Le Stringhe

```
str = “questa e' una stringa”                # assegnazione con i doppi apici
str = 'questa e\' una stringa'               # assegnazione con gli apici singoli
str = r“questa e' una stringa raw\n”         # raw (non considera le sequenze di escape)
str = u“questa e' una stringa unicode”       # unicode (codice universale)

str = “questa e' una stringa \               # multilinea
        con piu' lineee”                     #  (si usa “\” per indicare la nuova linea)
str = “””questa e' una stringa               # multilinea con i triplici apici
        con piu' lineee”””                   #  (usata molto dalle doc String)
```

Le stringhe sono sequenze
(immutabili!)

```
for i in str :print i                        # alcune istruzioni possibili...
print len(str), str[:2], str[::-1], 'ci'+'ao'
```

Modulo
string

Loris Michielutti

Metodi di Stringhe

Alcuni Metodi:

```
s.capitalize()                  # ritorna una copia con il primo carattere Maiuscolo
s.center(width)                 # ritorna una copia di lunghezza passata e centrata
s.count(sub[,start[,end]])      # ritorna il numero di occorrenze della sottostringa
s.encode([encoding[,errors]])   # ritorna la codifica dell'errore
s.endswith(suffix[,start[,end]]) # ritorna vero se la stringa finisce col suffisso specificato
s.expandtabs([tabsize])         # ritorna una copia della con tutti i tab espansi usando ' '
s.find(sub[,start[,end]])       # ritorna l'indice piu' basso della sottostringa altrimenti -1
s.index(sub[,start[,end]])      # uguale al prec. ma solleva una ecc. se non la trova
s.isalnum()                     # ritorna vero se tutti i caratteri sono alfanumerici
s.isalpha()                     # ritorna vero se tutti i caratteri sono alfabetici
s.isdigit()                     # ritorna vero se tutti i caratteri sono numerici
s.isupper()                     # ritorna vero se tutti i caratteri sono maiuscoli
s.join(seq)                     # rit. una concat. della strin. nella seq. separata da uno ' '
s.ljust(width)                  # rit. una stringa di lungh. passata e giustificata a sinitra
s.lower()                       # ritorna una copia con tutti i caratteri minuscoli
s.lstrip()                      # ritorna una copia eliminando tutti gli spazi a inizio stringa
s.replace(old, new[, maxsplit]) # rit. una copia sostit. la sottostringa old con quella new
s.upper()                       # ritorna una copia con tutti i caratteri maiuscoli
```

Formattazione Stringhe

Usa il codice della libreria C printf (il prefisso % indica la variabile)

a = 12.5; b= “ciao”; c= 100

”%f e' un floating %03d e' un decimale formattato %s e' una stringa” % (a,c,b)

Flag

“%-5d”	# Indica di formattare allineando a sinistra un decimale a 5 cifre
“%5d”	# Indica di formattare allineando a destra un decimale a 5 cifre
“%+5d”	# come prima ma aggiunge il segno al decimale
“%05d”	# come prima ma se mancano cifre aggiunge degli 0
“%5.1f”	# Indica di formattare un frazionario con 5 caratteri in totale (compresa la virgola!!!) di cui 1 dopo la virgola

Codice del Formato

d,i	# intero decimale con segno
o	# ottale senza segno
u	# decimale senza segno
x,X	# esadecimale senza segno (lowercase, uppercase).
e,E	# formato esponenziale floating point (lowercase, uppercase).
f,F	# formato decimale floating point (lowercase, uppercase).
g,G	# come "e" ma meno preciso
c	# singolo carattere
r,s	# stringa (conversione ottenuta usando repr(), str()).
%%	# usato per ottenere “%”

Liste e Tuple

Liste (mutabili) e Tuple (immutabili)

sono vettori eterogenei: t = “vettore eterogeneo”, 1, 'ciao', 20

sono sequenze con indice e slice t[1] ; t [2:3]

si possono impachettate e spachettate

```
x = 1,3,6,9                                  # packing
a,b,c,d = x                                  # unpacking
ROSSO, GIALLO, VERDE = range(3)              # come enum in C

t = “questa e' una tupla”, 1, 'ciao', 20     # immutabile (tupla)
l = [“questa e' una lista”, 3, 'ciao', 15]   # mutabile (lista)
l [0] = “modifico il primo elemento”
```

Operazioni eseguibili su Liste, Tuple e Stringhe

```
x in s                  # vero  se l'elemento in s e' uguale a x
x not in s              # falso  se l'elemento in s e' uguale a x
s + t                   # concatenamento di s e t
s * n, n*s              # concatena n copie di s
s[i]                    # ennesimo elemento di s, origine = 0
s[i:j]                  # ritaglio di s da i (incluso) a j (escluso)
len(s)                  # lunghezza di s
min(s)                  # il piu' piccolo elemento di s
max(s)                  # il piu' grande elemento di s
```

Operazioni sulle Liste

Operazioni eseguibili su sequenze mutabili Liste

```
as[i] =x                                          # sostituzione di un elemento
s[i:j] = t                                        # sostituzione di una parte di elementi
del s[i:j]          equivale a s[i:j] = []        # eliminazione di una parte
s.append(x)         equivale a s[len(s) : len(s)] = [x]   # aggiunta di un elemento alla fine
s.extend(x)         equivale a s[len(s):len(s)]= x        # aggiunta di N elementi alla fine
s.count(x)                                        # ritorna il num. delle Occorrenze
s.index(x)                                        # ritorna la prima Occorrenza
s.insert(i, x)      equivale a s[i:i] = [x] if i >= 0     # aggiunge un elem. nella posizione (i)
s.remove(x)         equivale a del s[s.index(x)]          # rimuove il primo elemento
s.pop([i])          equivale a x = s[i]; del s[i]; return x   # estrae un elemento di indice (i)
s.reverse()                                       # inverte l'ordine della sequenza
s.sort()                                          # sistema l'ordine della sequenza
s.set(x)                                          # ritorna un insieme di elem. univoci
```

Attenzione se volete una copia non copiate il riferimento

copia = lista, y = x

ma richiedetela esplicitamente

y = list(x) o x[:] o x*1 o copy.copy(x)

Dizionari

I Dizionari sono mappe non sono sequenze

```
d = {1:2, 'mio':10, 2:[2,'i',5]}      # questo e' un dizionario (chiave, elemento)
```

la chiave di solito e' immutabile!!

```
d[ [1,2] ] = ...                      # una Lista non puo' essere una chiave
d[ {1,2} ] = ...                      # un   Dizionario non puo' essere una chiave
d[  1,2  ] = ...                      # una Tupla puo' essere una chiave
```

pero' un dizionario puo' essere iterato tramite la chiave

```
for k in d: print d[k]
```

Operazioni sui Dizionari

```
len(d)                                # numero di elementi in d
d[k], d.get(k)                        # elemento in d con chiave k
d[k] = x , d.setdefault(x)            # modifico l'elemento con chiave k
del d[k]                              # elimino l'elemento con chiave k
d.clear()                             # elimino tutti gli elementi di d
d.copy()                              # faccio una copia del dizionario
d.has_key(k)                          # ritorna 1 se esiste la chiave k
d.items()                             # creo una lista con le coppie (chiave,elemento)
d.keys()                              # creo una lista con le chiavi
d.values()                            # creo una lista con gli elementi
d1.update(d2)                         # aggiorna il dizionario d1 con d2
```

equivale a `for k, v in d2.items(): d1[k] = v`