



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)





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
- Efficienza di sviluppo

Contro:

- Velocita' di esecuzione

Compilato

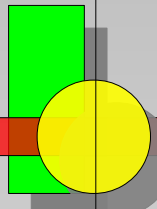
- Compilazione Differita
si deve creare un file poi si deve compilarlo
per ottenere il file eseguibile

Pro:

- Velocita' di esecuzione

Contro:

- Elaborazione dei risultati differita



Le Assegnazioni

Python le Basi



questo e' un commento (il commento viene preceduto 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

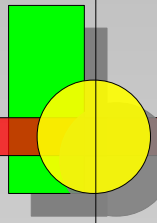
tua = essa = "nostra"

assegnazione multipla

mia, tua = 1, 2

assegnazione multipla con scambio dei contenuti

mia, tua = tua, mia



Le Condizioni

Python le Basi



```
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
```

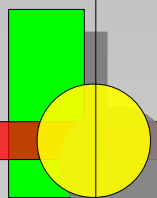
Condizioni

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

Boolean

<code>False</code>	0, "", None
<code>True</code>	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

Python le Basi



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

Funzioni Base

<code>x + y</code>	# somma
<code>x - y</code>	# sottrazione
<code>x * y</code>	# moltiplicazione
<code>x / y</code>	# divisione
<code>x % y</code>	# resto divisione
<code>x ** y</code>	# x alla potenza di y

Operazioni Unitarie

<code>+x</code>	# incremento
<code>-x</code>	# decremento

Funzioni sui Bit

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

Moduli matematici

<code>math</code>	#
<code>cmath</code>	#
<code>operator</code>	#



```
while mia > 20:
```

```
    mia -= 1
```

```
# condizione
```

```
# eseguita finche' la condizione e' VERA
```

```
for i in 1,2,4 :
```

```
    print i ** 2
```

```
# condizione (finche' esistono elementi)
```

```
# eseguita finche' la condizione e' VERA
```

```
for i in xrange(0,6,2)
```

```
    print i ** 2
```

```
# condizione (da 0 a 6 a step di 2)
```

```
# eseguita finche' la condizione e' VERA
```

```
a = ['Mary', 'had', 'a', 'little', 'lamb']
```

```
for i, item in enumerate(a)
```

```
    print i, item
```

```
# itera sugli indici della sequenza
```

```
# 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
```

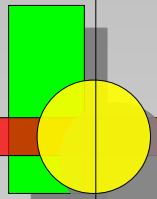
Nota:

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

Python le Basi



```
str = "questa e' una stringa"  
str = 'questa e\' una stringa'  
str = r"questa e' una stringa raw\n"  
str = u"questa e' una stringa unicode"
```

```
# assegnazione con i doppi apici  
# assegnazione con gli apici singoli  
# raw (non considera le sequenze di escape)  
# unicode (codice universale)
```

```
str = "questa e' una stringa \  
      con piu' lineee"  
str = """questa e' una stringa  
      con piu' lineee"""
```

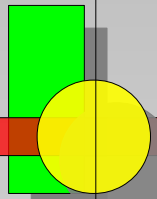
```
# multilinea  
# (si usa "\n" per indicare la nuova linea)  
# multilinea con i triplici apici  
# (usata molto dalle doc String)
```

Le stringhe sono sequenze
(**immutabili!**)

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

```
# alcune istruzioni possibili...
```

Modulo
string



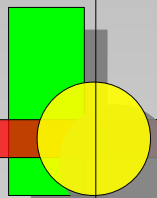
Metodi di Stringhe

Python le Basi



Alcuni Metodi:

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



Formattazione Stringhe

Python le Basi



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

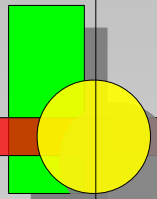
singolo carattere

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 impackettare e spackettare

```
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)
```

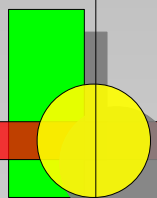
lunghezza di s

```
min(s)
```

il piu' piccolo elemento di s

```
max(s)
```

il piu' grande elemento di s



Operazioni sulle Liste

Python le Basi



Operazioni eseguibili su sequenze mutabili Liste

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

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`