Python

Materiale tratto da presentazioni di
G. Folino e M. Barisione

Python

- Python è un linguaggio di programmazione dinamico orientato agli oggetti utilizzabile per molti tipi di sviluppo software
- È fornito di una buona libreria standard ed è facilmente estendibile
  - http://www.catswhocode.com/blog/python-50-modules-for-all-needs
- Python è distribuito con licenza Open-Source
  - È multipiattaforma

- www.python.it
- www.python.org

Python 2 o 3

- Oggi abbiamo sia 2.7 che 3.6
- Per un'introduzione preliminare, non ci sono differenze significative
  - Forse la principale è print che diventa una funzione – e non è più un'istruzione

Alcuni concetti introduttivi

- ogg.f() è un metodo
- I metodi possono inizialmente essere considerati come delle funzioni applicate sulla variabile prima del punto
- Il seguente codice Python

```
"CIAO".lower()
```

- può essere pensato equivalente al seguente codice C

```
stringa_lower("CIAO");
```

Interprete interattivo

- Python offre un prompt pronto a ricevere comandi
- Possiamo ora inserire qualsiasi costrutto del linguaggio e vedere immediatamente l'output:

```
>>> 3+5
8
>>> print "Hello world!"
Hello world!
```

Comandi

- I comandi si possono inserire direttamente dallo standard input
- I file sorgente Python sono file di testo, generalmente con estensione .py
  - python nomeProgramma.py

Comandi

- # inizia un commento che si estende fino a fine riga
- Le istruzioni sono separate dal fine riga e non da ;
  - Il ; può comunque essere usato per separare istruzioni sulla stessa riga ma è sconsigliato
- Per far continuare un'istruzione anche sulla linea successiva è necessario inserire un \ a fine riga
- Se le parentesi non sono state chiuse correttamente Python capisce che l'istruzione si estende anche sulla riga successiva

Input e Output

- “print” stampa il suo argomento come una stringa

```
>>> print 5
5
>>> print "Hello world"
Hello world
```

- Più argomenti separati da un virgola sono stampati separati da uno spazio

```
>>> print 1, 2, "XXX"
1 2 XXX
```

- Con una virgola alla fine non viene stampato il ritorno a capo

Input e Output

- Si può formattare l'output come in c

```
>>> x=18; y=15
>>> print "x=%d y=%d\n" % (x,y)
x=18 y=15
```

- Per leggere un numero si usa input()

```
>>> x=input('Scrivi un numero:')
```

  - Da errore se non si inserisce un numero

- Per leggere una stringa raw_input()

```
>>> x=raw_input('Scrivi il tuo nome:')
```

Operatori numerici

- Stessi operatori del C
  - Le parentesi possono essere usate per raggruppare:

```
>>> (5 + 3) * 2
16
>>> (6 & 3) / 2
1
```

- Esiste anche l'operatore elevamento **:

```
>>> 5 ** 2
25
```

- Non esistono ++ e —
  - Come in C, "3/2" è "1"
  - Altrimenti usare 3.0/2 (1.5)
  - 3.0//2 è la divisione intera

Variabili

- I loro nomi come in C
- Non devono essere dichiarate
  - Tipizzazione dinamica
- Non possono essere utilizzate prima che venga loro assegnato un valore
- Possono riferirsi ad oggetti di qualsiasi tipo

```
x=5
nome = “Marco”
```

- Sintetico

```
inizio, fine = 2,100
```

Assegnamento

- += -= *= /= //=%= **=
- Non è creata una copia dell'oggetto

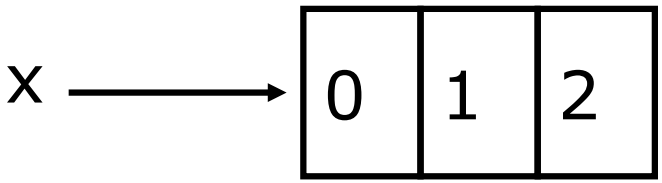
```
x = y # si riferiscono allo stesso oggetto
```

- Esempio:

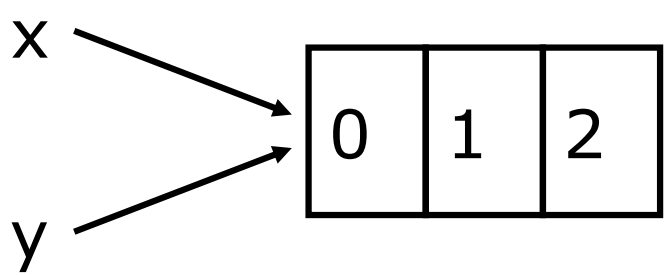
```
>>> x = [0, 1, 2]
>>> y = x
>>> x.append(3)
>>> print y
[0, 1, 2, 3]
```

Cosa succede

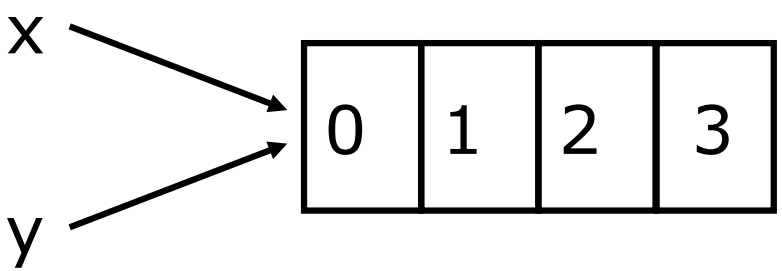
- `x = [0, 1, 2]`



- `y = x`

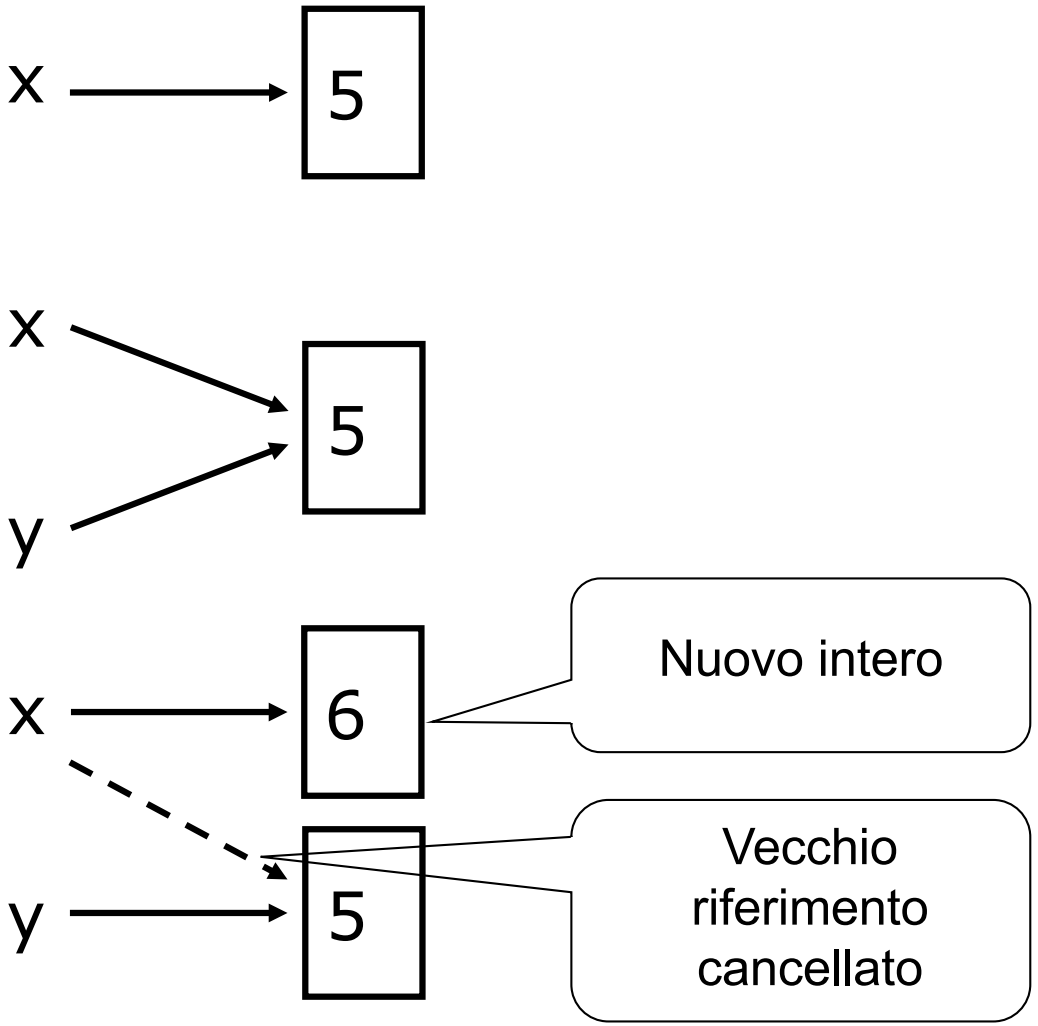


- `x.append(3)`



Assegnamento

- `x = 5`
- `y = x`
- `x = x + 1`



del (due usi differenti)

- del x cancella la variabile x, cioè non si potrà più usare x senza avergli prima assegnato un nuovo valore
  - Non distrugge ciò a cui la variabile si riferisce come free
- del seq[ind] cancella l'elemento con indice/chiave ind da seq

```
li = [1, 2, 3]
del li[1]                    → li = [1, 3]
d = {"a": 1, "b": 6}
del d["a"]                   → d = {'b': 6}
l2 = [1, 2, 3, 4]
del l2[1:3]                  → l2 = [1, 4]
```

Vero e falso

- In Python esistono due variabili di tipo bool
  - “True” uguale a 1 e “False” uguale a 0
- Ogni singolo tipo può definire quando il suo valore è vero o falso
  - Il numero “0” o “0.0”
  - Una stringa vuota (“”)
- Gli altri valori sono considerati veri

Confronto

- == != < >
- Se necessario vengono eseguite le necessarie conversioni intero → virgola mobile
  - `1 == 1.0  → Vero`
- Esiste anche l'operatore "<>" equivalente a "!=" ma obsoleto
- and or e not
- and e or utilizzano la logica del corto circuito
  - Il secondo argomento non viene valutato se il risultato dell'operazione è già noto in base al solo primo argomento

Ancora

- in: vero se il primo operando è contenuto nel secondo

```
5 in [1, 2, 3]              → Falso
2 in [1, 2, 3]              → Vero
"a" in {"x": 1, "a": 2}     → Vero
"a" in "ciao"               → Vero
```

- is: vero se il primo operando è il secondo
  - Non solo è uguale!
  - Attualmente implementato come confronto fra le posizioni in memoria degli operandi

if

```
if condizione:
    ...
elif condizione_alternativa:
    ...
else:
    ...
```

- Sia la parte elif sia la parte else sono facoltative
- Può esserci un numero qualsiasi di elif
- Non sono necessarie le parentesi intorno all'espresisone booleana
- Non sono possibili assegnamenti all'interno della condizione

Esempio

```
# Chiedi un numero.
# Stampa se è pari o dispari.

number = input("Tell me a number: ")
if number % 2 == 0:
    print number,"is even.”
elif number % 2 == 1:
    print number,"is odd."
else:
    print number,"is very strange."
```

Indentazione

- Il raggruppamento è definito dall'indentazione
  - Non si usano parentesi graffe, coppie "begin"/"end" e simili
  - Obbliga a scrivere codice ordinato
  - Più naturale, evita i tipici problemi del C

```
if (0)
    printf("Questo non viene eseguito");
    printf("Questo sì");
```

pass

- In Python dopo un “if” deve esserci un blocco indentato

```
if a:
b = 12 # Dovrebbe essere indentato!
```

- Si può usare l’istruzione pass che non ha nessun effetto

```
if a:
    pass      # Non fa nulla
b = 12
```

while

```
while condizione:
    ...
```

- break per uscire dal ciclo
- continue per passare all'iterazione successiva

```
while 1: # significa ciclo infinito
    if condizione1:
        continue
    if condizione2:
        break
print 42
```

Esempio

```
# Giocare ad indovinare il numero alto-basso

number = 78
guess = 0

while guess != number :
    guess = input ("Guess a number: ")

    if guess > number :
        print "Too high"

    elif guess < number :
        print "Too low"

print "Just right"
```

for

```
for variabile in iteratore:
    ...
```

- iteratore può essere una sequenza, lista, tupla, stringa, dizionario
- Si possono usare continue e break come per il while

Esempio

```
onetoten = range(1,11)
for count in onetoten:
    print count


list = [4, 5, 7, 8, 9, 1, 0, 7, 10]
list.sort()
prev = list[0]
del list[0]
for item in list:
    if prev == item:
        print "Duplicate of ", prev, " Found"
    prev = item
```

Stringhe

- Sono racchiuse fra apici singoli o doppi e si utilizzano le stesse sequenze di escape del C

```
>>> 'Python'
'Python'
>>> print "Ciao\nmondo"
'Ciao'
'mondo'
```

- Si possono usare alcuni operatori visti per i numeri

```
>>> "ciao " + "mondo"   # concatenazione
'ciao mondo'
>>> "ABC" * 3          # ripetizione
'ABCABCABC'
```

Sottostringhe

```
"ciao"[1]    # carattere 1              → “i”
"ciao"[1:3]  # dall’1 al 3 escluso      → “ia”
"ciao"[2:]   # dal 2 alla fine          → “ao”
"ciao"[:3]   # fino al 3 escluso        → “cia”
"ciao"[-1]   # l’ultimo carattere       → “o”
"ciao"[:-2]  # fino al penultimo        → “ci”
"ciao"[-1:1] # non esiste               → “”
```

- Le stringhe sono immutabili (come i numeri):

"ciao"[2] = "X" → Errore!

c	i	a	o
0	1	2	3
-4	-3	-2	-1

Ancora sulle stringhe

- Le stringhe possono estendersi su più righe, in questo caso sono delimitate da tre apici singoli o doppi

```
"""Questa è una
stringa su più righe"""
```

- For e stringhe

```
nome = ”Gianluigi”
for c in nome:
    print c
```

Stringhe

- split()
- join()
- lower()/upper()
- find()
- replace()
- strip()/lstrip()/rstrip()

Split

- str.split(del="", num=string.count(str))
  - str: stringa da dividere
  - del: delimitatore, lo spazio è il default
  - num: numero di linee
  - ritorna una lista di linee

```
str = "Line1-abcdef \nLine2-abc \nLine4-abcd"
print str.split()
print str.split(' ', 1)
```

Find

- str1.find(str2, beg=0, end=len(string))
  - str1: la stringa in cui cercare
  - str2: la stringa da cercare
  - beg: l'indice di partenza. Il default è zero
  - end: l'indice di fine. Il default è la lunghezza della stringa
  - ritorna il valore dell'indice o -1 se non trovato

```
str1 = "this is string example....wow!!!"
str2 = "exam"

print str1.find(str2)
print str1.find(str2, 10)
print str1.find(str2, 40)
```

Strip

- str.strip([chars])
  - str: la stringa da manipolare
  - chars: i caratteri da rimuove
  - ritorna una copia di str senza i caratteri
    - Strip: inizio e fine
    - Lstrip: inizio
    - Rstrip fine

```
str = "0000000this is string 000 example....wow!!!0000000";
print str.strip( '0' )
print str.lstrip( '0' )
print str.rstrip( '0' )
```

Altri esempi

```
Saluto = "Ciao!"
Saluto[0] = 'M'                 # ERRORE!
print Saluto


Saluto = "Ciao!"
NuovoSaluto = 'M' + Saluto[1:]
print NuovoSaluto


Prefissi = "JKLMNOPQ"
Suffisso = "ack"

for Lettera in Prefissi:
    print Lettera + Suffisso
```

None

- None è una variabile molto importante con lo stesso ruolo di NULL in C
- In C NULL è uguale a 0
- In Python None è di un tipo non numerico
  - Non vi è rischio di confusione con altri tipi

Esempio

```
items = {"cat" : 1, "dog" : 2, "piranha" : 3}

# Get an element that does not exist.
v = items.get("giraffe")

# Test for it.
if v == None:
    print("Not found")
```

Import

- Per importare il modulo delle funzioni matematiche

```
import math
```

- Per usarlo

```
math.sqrt(2), math.log(5), ecc.
```

Alcune funzioni built-in

- range([start,] stop[, step])
  - step è l'incremento, il valore di default è +1
  - Il valore di default di start è 0
  - Molto usato con i cicli for

```
for i in range(5): print i
```

- len(seq) ritorna il numero di elementi in seq

```
len("ciao")                    → 4
len("x\0x")                    → 3
len([1, 2, 3])                 → 3
len({"a": 1, "b": 5})          → 2
```

Nuove funzioni

- La sintassi è

```
def funzione(arg1, arg2, opz1=val1, opz2=val2):
...
```

- Non bisogna specificare il tipo ritornato
  - L'equivalente delle funzioni "void" del C sono funzioni che ritornano None
  - Una funzione può ritornare un oggetto di qualsiasi tipo
- Gli argomenti sono normali variabili e possono essere in qualsiasi numero
  - Se la funzione non accetta argomenti basta usare una lista di argomenti vuota, ad esempio def foo():

Nuove funzioni

- Gli argomenti opzionali possono non essere specificati dal chiamante, in questo caso assumono il valore di default
- Le variabili all'interno della funzione non sono visibili dall'esterno

```
def foo(x, y, z=42, k=12):
    print x, y, z, k

foo(5, 3, k=9)
```

return

- La sintassi è:

```
return valore_di_ritorno
```

- Se il valore di ritorno viene omesso viene ritornato None
- Se il flusso del programma esce dalla funzione senza aver trovato un'istruzione return viene ritornato None

```
def somma(a, b):
    return a + b
```

Esempio

```
def any_odd(xs):
    """ Return True if there is an odd number in xs, a
        list of integers. """
    for v in xs:
        if v % 2 == 1:
            return True
    return False


def mult(a,b):
    if b == 0:
        return 0
    rest = mult(a,b - 1)
    value = a + rest
    return value
```

Liste

- Contengono elementi anche eterogenei
- Sono implementate usando array e non liste

```
>>> [1, 2, "ciao"]
[1, 2, 'ciao']
```

- Stessi operatori delle stringhe ma sono mutabili

```
[1] + [3, 6]          → [1, 3, 6]
[1, 0] * 3            → [1, 0, 1, 0, 1, 0]
[2, 3, 7, 8][1:3]     → [3, 7]
[2, 3, 7, 8][:2]      → [2, 3]
[1, 2, 3][0] = 5      → [5, 2, 3]
```

Liste

- L.append(obj)/L.extend(list)
- L.insert(index, obj)
- L.pop([index])/L.remove(value)
- L.reverse()
- L.sort()
- Viene modificata la lista, non viene ritornata una nuova lista
  - + rispetto ad append crea una nuova lista

Tuple

- Delimitate da parentesi, gli elementi sono separati da virgole

```
(1, 2, 3)
(1,)      # un solo elemento
()    # nessun elemento
```

- Simili alle liste ma immutabili

```
(1, 2)[0] = 5 → Errore!
```

- Le parentesi possono essere omesse

```
variabile =  5, 2, 3
variabile = (5, 2, 3)
```

Dizionari

- Associano ad una chiave un valore
- Creati nella forma {chiave1: val1, chiave2: val2}

```
{"nome": "Mario", "cognome": "Rossi"}
```

- L'accesso e l'inserimento di elementi avviene come per le liste

```
{"a": 1, "b": 2}["a"]        → 1
{"a": 1, "b": 2}["X"]        → Errore!
{}["X"] = 2                  → {'X': 2}
```

- Le chiavi devono essere immutabili
- Le chiavi non vengono tenute ordinate!

Dizionari

- D.clear()
- D.copy()
- D.has_key(k)
- D.items()/D.keys()/D.values()
- D.update(D2)
- D.get(k, d)

Esempi

```
cLettere = {}
for Lettera in "Mississippi":
    cLettere [Lettera] = cLettere.get(Lettera, 0) + 1
cLettere = cLettere.items()
cLettere.sort()
print cLettere


def print_all_grades():
    print '\t',
        for i in range(len(assignments)):
            print assignments[i],'\t',
    print
        keys = students.keys()
        keys.sort()
        for x in keys:
            print x,'\t',
            grades = students[x]
            print_grades(grades)
```

Accesso ai file

- I file vengono gestiti in modo simile al C
- open(nomefile[, modo]) apre nomefile in modalità modo (r è il valore di default) e ritorna un oggetto di tipo file
- I metodi principali degli oggetti file sono
  - read([n]) ritorna n byte dal file
    - Se n è omesso legge tutto il file
  - readline() ritorna una riga
  - readlines() ritorna una lista con le righe rimanenti nel file
  - write(str) scrive str sul file

Accesso ai file

- writelines(list) scrive tutti gli elementi di list su file
- close() chiude il file (richiamato automaticamente dall'interprete)
- flush() scrive su disco i dati presenti in eventuali buffer
- seek(offset[, posiz]) muove di offset byte da posiz.
  - 0: dall'inizio del file (valore di default)
  - 1: dalla posizione corrente
  - 2: dalla fine del file (offset è normalmente negativo)
- tell() ritorna la posizione corrente
- truncate([n]) tronca il file a non più di n byte
  - Il valore di default è la posizione corrente

Esempio

```
import sys

fsillabe = open('sillabe.txt', 'r')
sillabe = fsillabe.read().split()
fsillabe.close()

fdizionario = open('italiano.txt', 'r')
paroleIta = fdizionario.read().split()
fdizionario.close()

fparole = open('parole.txt', 'w')

for s1 in sillabe:
    for s2 in sillabe:
        parola = s1+s2
      if parola.lower() in paroleIta:
            fparole.write(parola+'\n')

fparole.close()
```

global

- L'assegnamento all'interno di una funzione assegna il valore ad una variabile locale

```
x = 5
def f():
    x = 42 # x è nuova variabile locale!
```

- Con `global nome_var` si indica all'interprete che `nome_var` è globale e non locale

```
x = 5
def f():
    global x
    x = 42 # x è la variabile globale
```

Stampa maggiore n numeri letti

```
n = input('Inserisci un numero: ')
max = None

while n != 0:
    if not max or n > max:
        max = n
    n = input('Inserisci un numero: ')

print max
```

Versione II

```
n = raw_input('Inserisci un numero: ')
max = None

while n != '':
    if not max or int(n) > max:
        max = int(n)
    n = raw_input('Inserisci un numero: ')

print max
```

Mesi

```
which_one = input("What month (1-12)? ")
months = ['January', 'February', 'March', 'April',\
          'May', 'June', 'July', 'August', 'September',\
          'October', 'November', 'December']
if 1 <= which_one <= 12:
        print "The month is",months[which_one - 1]
```

Fattoriale

```
# Definisce una funzione che calcola il fattoriale.

def fattoriale(n):
    if n <= 1:
        return 1
    return n*fattoriale(n-1)

n = raw_input('Inserisci un numero: ')
v = int(n)
print "Il fattoriale di %d e' %d" % (v, fattoriale(v))
```

File

```
documento = open("pari.txt", "w")

for i in range(1,100):
    if i%2==0:
        riga = "%i\n" % (i)
        documento.write(riga)

documento.close()

pari = open("pari.txt", "r")
p = pari.readlines()

for i in range(0, len(p)):
    p[i] = int(p[i].strip("\n"))

print sum(p)
pari.close()
```

Python 3

- La divisione ritorna un float, anche se il risultato è un intero
- print -> print()
  - Python 2
    - print "The answer is", # comma suppresses newline
    - print 42
    - Output: The answer is 42
  - Python 3
    - print("The answer is", end=" ")
    - print(42)
    - Output: The answer is 42
  - print("01","12","1981",sep="-")
    - Output: 01-12-1981

Python 3

- <> non più ammesso
- Input
  - raw_input non più disponibile
  - Sempre input -> che restituisce una stringa
  - Si può fare cast: int(input(...))
- File
  - for line in file("my_file.txt"): print line
  - for line in open("alice.txt"): print(line)

Parametri dei main

```
#include<stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[]) {
     int v1 = atoi(argv[1]);
     int v2 = atoi(argv[2]);
     printf("La somma e' %d\n", v1+v2);

     return 1;
}
```

somma 13 38