



Unit 1

Basi di Python programmi, funzioni

Python!

- Creato nel 1991 da Guido van Rossum
 - Il nome deriva da Monty Python
- Utile come **linguaggio di scripting**
 - **script**: Piccolo programma per un unico utilizzo
 - Sviluppato per supportare progetti di tipo medio/piccolo
- Usato da:
 - Google, Yahoo!, Youtube
 - Molte distribuzioni Linux
 - Giochi and app
 - Recentemente diffuso in ambito scientifico



Installare Python

Windows:

- Download Python da <http://www.python.org>
- Installate Python.
- Aprite **Idle** dal menu' Start.

Mac OS X:

- Python e' gia' installato.
- Aprite un terminale ed eseguite `python` o lanciate Idle da Finder.

Linux:

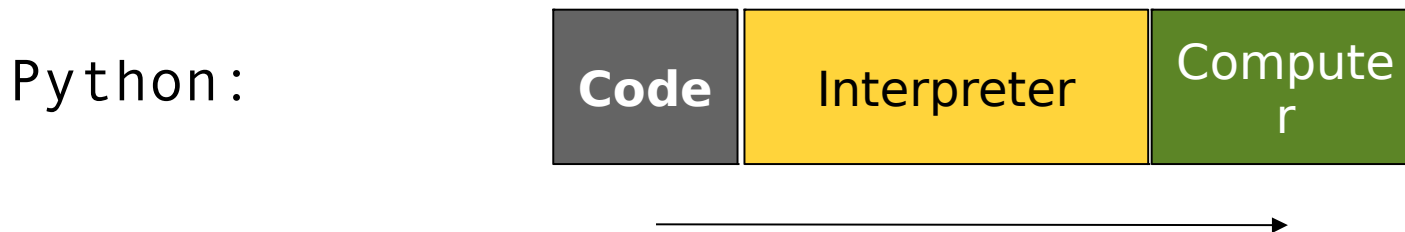
- E' probabile che Python sia gia' installato. Verificate scrivendo `python` nel terminale.
- Se non e' disponibile installatelo dal sistema di gestione dei pacchetti della vostra distribuzione.



Linguaggi interpretati

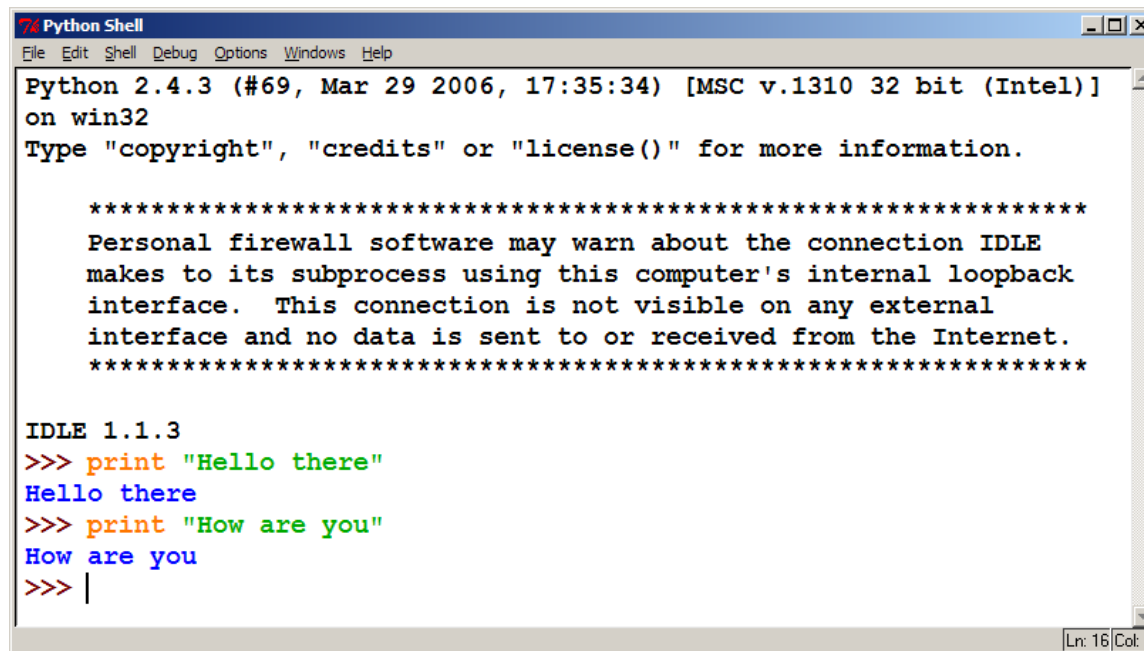
- **interpretato**

- Non compilato come Java, C, C++
- Codice scritto ed eseguito direttamente da un **interprete**
- E' possibile scrivere comandi direttamente nell'interprete e osservarne il risultato (come in R)



L'Interprete Python

- Permette di scrivere comandi uno alla volta ed osservarne il risultato
- Modo molto comodo di fare esperienza con la sintassi di Python



```
Python Shell
File Edit Shell Debug Options Windows Help
Python 2.4.3 (#69, Mar 29 2006, 17:35:34) [MSC v.1310 32 bit (Intel)]
on win32
Type "copyright", "credits" or "license()" for more information.

*****
Personal firewall software may warn about the connection IDLE
makes to its subprocess using this computer's internal loopback
interface. This connection is not visible on any external
interface and no data is sent to or received from the Internet.
*****

IDLE 1.1.3
>>> print "Hello there"
Hello there
>>> print "How are you"
How are you
>>> |
```



Primo programma in Python

- Python non ha un metodo `main` come Java
 - Il codice del programma e' semplicemente scritto nel file che verra' eseguito (file sorgente)
- In Python le righe non finiscono con `;` (come in PERL)

hello.py

```
1 print("Hello, world!")
```



La funzione print

```
print("text")
```

```
print()          (una linea vuota)
```

- Sequenze di escape come `\` sono le stesse di Java
- Le stringhe possono iniziare/finire con `'`

swallows.py

```
1 print("Hello, world!")
2 print()
3 print("Suppose two swallows \"carry\" it together.")
4 print('African or "European" swallows?')
```

Commenti

- Sintassi:
testo commentato (una riga)

swallows2.py

```
1 # Mario Rossi, MTD, autunno 19
2 # Messaggi importanti.
3 print("Hello, world!")
4 print()                # linea vuota
5 print("Suppose two swallows \"carry\" it together.")
6 print('African or "European" swallows?')
```


Funzioni

- **Funzioni:** Equivalenti di metodi statici in Java (e di funzioni in R).

- Sintassi

```
def name () :  
    statement  
    statement  
    ...  
    statement
```

hello2.py

```
1  # Prints a helpful message.  
2  def hello():  
3      print("Hello, world!")  
4  
5  # main (calls hello twice)  
6  hello()  
7  hello()
```

- DEVE essere dichiarata al di sopra del codice “principale” (main)
- Espressioni all’interno delle funzioni DEVONO essere INDENTATI (ad es. ogni riga inizia con una o piu’ tabulazioni)

Significato spazi bianchi

- Python usa le **indentazioni** per indicare i blocchi di codice invece di { }
- Questo rende il codice piu' leggibile
- In Java l'indentazione e' opzionale. In Python essa e' **obbligatoria!**

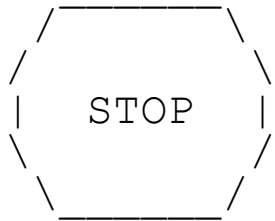
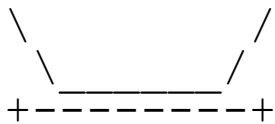
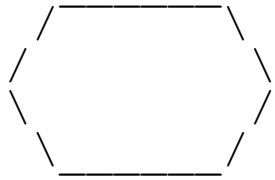
hello3.py

```
1 # Prints a helpful message.
2 def hello():
3     print("Hello, world!")
4     print("How are you?")
5
6 # main (calls hello twice)
7 hello()
8 hello()
```



Esercizio (funzioni)

- Scrivere un programma Python che ricrei questo output:



Soluzione esercizio

```
def egg():
    top()
    bottom()
    print()
```

```
def cup():
    bottom()
    line()
    print()
```

```
def stop():
    top()
    print("|   STOP   |")
    bottom()
    print()
```

```
def hat():
    top()
    line()
    print()
```

```
def top():
    print("           ")
    print("/       \\")
    print("/           \\")
```

```
def bottom():
    print("\\           /")
    print("\\       /")
```

```
def line():
    print("+-----+")
```

```
# main
egg()
cup()
stop()
hat()
```





Unit 2

Espressioni e variabili; ciclo `for`

Espressioni

- Aritmetica molto simile a Java e R
 - Operatori: + - * / %
 - Precedenza: () quindi ** quindi * / % quindi + -
 - Numeri interi e numeri reali

```
>>> 1 + 1
2
>>> 1 + 3 * 4 - 2
11
>>> 7 / 2
3
>>> 7.0 / 2
3.5
>>> 10 ** 6
1000000
```

Operatori aritmetici

Assumendo le variabili **a=5** e **b=3** :

+ Addizione Es: $a+b=8$

- Sottrazione Es: $a-b=2$

*** Moltiplicazione** Es: $a*b=15$

/ Divisione reale Es: $a/b= 1.6666666666666667$

// Divisione intera Es: $a//b=1$

% Resto della divisione Es: $a\%b=2$

**** Potenza di un numero** Es: $a**b=125$



Operatori di assegnazione 1

= assegna Es: $a=5$ assegna alla variabile a che sta a sinistra il valore 5 (ovvero ciò che è a destra dell'uguale.) Un altro esempio potrebbe essere $a=b+c$ dove ad a stavolta assegniamo la somma $b+c$.

+= somma e assegna Assegna all'operando di sinistra la somma tra esso e l'operando di destra. Es: $a+=2$ equivale a fare $a=a+2$, quindi assegna alla variabile a che sta a sinistra il valore di a (supponiamo sempre $a=5$) sommato a 2. Quindi otterremo $a=7$.

-= sottrae e assegna Assegna all'operando di sinistra la differenza tra esso e l'operando di destra. Es: $a-=2$ equivale a fare $a=a-2$, quindi assegna alla variabile a che sta a sinistra il valore di a (supponiamo sempre $a=5$) meno 2. Quindi otterremo $a=3$.

***= moltiplica e assegna** Assegna all'operando di sinistra il prodotto tra esso e l'operando di destra. Es: $a*=2$ equivale a fare $a=a*2$, quindi assegna alla variabile a che sta a sinistra il valore di a (supponiamo sempre $a=5$) moltiplicato per 2. Quindi otterremo $a=10$.

/= divide e assegna Assegna all'operando di sinistra la divisione reale tra esso e l'operando di destra. Es: $a/=2$ equivale a fare $a=a/2$, quindi assegna alla variabile a che sta a sinistra il valore di a (supponiamo sempre $a=5$) diviso 2. Quindi otterremo $a=2.5$.



Operatori di assegnazione 2

//= divide e assegna Assegna all'operando di sinistra la divisione arrotondata tra esso e l'operando di destra. Es: $a//=2$ equivale a fare $a=a//2$, quindi assegna alla variabile a che sta a sinistra il valore di a (supponiamo sempre $a=5$) diviso 2 arrotondato all'intero. Quindi otterremo $a=2$.

%= calcola il resto e assegna Assegna all'operando di sinistra il resto ottenuto dalla divisione tra esso e l'operando di destra. Es: $a%=2$ equivale a fare $a=a\%2$, quindi assegna alla variabile a che sta a sinistra il valore di a (supponiamo sempre $a=5$) diviso 2 arrotondato all'intero. Quindi otterremo $a=1$.

****= calcola la potenza e assegna** Assegna all'operando di sinistra il risultato dell'elevamento a potenza di esso come base ed esponente l'operando di destra. Es: $a**=2$ equivale a fare $a=a^2$, quindi assegna alla variabile a che sta a sinistra il valore di a (supponiamo sempre $a=5$) elevato a 2. Quindi otterremo $a=25$.



Variabili

- Dichiarazione
 - Non si scrive il tipo; stessa sintassi dell'assegnamento
- Operatori
 - Non esistono operatori ++ o -- operatori (si incrementa o decrementa di 1)

Java	Python
<pre>int x = 2; x++; System.out.println(x) ;</pre>	<pre>x = 2 x = x + 1 print(x)</pre>
<pre>x = x * 8; System.out.println(x) ;</pre>	<pre>x = x * 8 print(x)</pre>
<pre>double d = 3.2; d = d / 2; System.out.println(d) ;</pre>	<pre>d = 3.2 d = d / 2 print(d)</pre>



Variabili

- Regole di definizione dei **nomi di variabile**
 - Alle variabili viene sempre assegnato un nome
 - Il nome di una variabile non puo' essere un numero
 - Il nome di una variabile non puo' iniziare con un numero (es. 5area non e' ammesso, area5 si)
 - Gli spazi non sono ammessi nei nomi di variabile. Se servono usate `_`.
 - I nomi di varibile non possono contenere simboli (ad esempio \$,%,&,# ecc.)
 - Non possono essere nessuna delle parole chiave del linguaggio. Per ottenere una lista delle parole chiave di python scrivete **keywords** nel prompt di Python.



Tipi di variabile

- Python e' piu' permissivo di Java
 - Il tipo di variabile non va dichiarato
 - Le variabili possono cambiare di tipo durante l'esecuzione del programma

Value	Java type	Python type
42	int	int
3.14	double	float
"ni!"	String	str

Per conoscere il tipo (attuale) di una variabile Python mette a disposizione la funzione **type(nomevariabile)**



Moltiplicare di stringhe

- Le stringhe Python possono essere moltiplicate per un valore intero.
 - Il risultato sono diverse copie concatenate della stringa.

```
>>> "hello" * 3
"hellohellohello"
```

```
>>> print(10 * "yo ")
yo yo yo yo yo yo yo yo yo yo
```

```
>>> print(2 * 3 * "4")
444444
```

Concatenare stringhe

- Interi e stringhe non possono essere concatenate in Python.

- Possibile soluzione:

`str(value)`

- converte valore in stringa

`print(expr, expr)`
linea

- stampa due variabili sulla stessa

```
>>> x = 4
```

```
>>> print("Thou shalt not count to " + x + ".")
```

```
TypeError: cannot concatenate 'str' and 'int' objects
```

```
>>> print("Thou shalt not count to " + str(x) + ".")
```

```
Thou shalt not count to 4.
```

```
>>> print(x + 1, "is out of the question.")
```

```
5 is out of the question.
```

Il ciclo for

```
for name in range(max):  
    statements
```

- Ripete per valori tra 0 (incluso) e **max** (escluso)

```
>>> for i in range(5):  
...     print(i)  
0  
1  
2  
3  
4
```



for Variazioni ...

```
for name in range(min, max):  
    statements
```

```
for name in range(min, max, step):  
    statements
```

- Possibile specificare un minimo diverso da 0 ed un step diverso da 1

```
>>> for i in range(2, 6):  
...     print(i)  
2  
3  
4  
5  
>>> for i in range(15, 0, -5):  
...     print(i)  
15  
10  
5
```



Cicli annidati

- Cicli annidati sono spesso rimpiazzati da moltiplocazioni e addizioni tra stringhe

```
....1
...2
..3
.4
5
```

Java

```
1 for (int line = 1; line <= 5; line++) {
2     for (int j = 1; j <= (5 - line); j++) {
3         System.out.print(".");
4     }
5     System.out.println(line);
6 }
```

Python

```
1 for line in range(1, 6):
2     print((5 - line) * "." + str(line))
```



Esercizio

- Riscrivete il programma Mirror in Python. Il suo output e' il seguente:

```
#=====#  
|      <><>      |  
|     <>.....<>     |  
|    <>.....<>    |  
|   <>.....<>   |  
|  <>.....<>  |  
| <>.....<> |  
| <>.....<> |  
|  <>.....<>  |  
|   <>.....<>   |  
|    <>.....<>    |  
|     <>.....<>     |  
|      <><>      |  
#=====#
```

Soluzione esercizio


```
def bar():
    print "#" + 16 * "=" + "#"

def top():
    for line in range(1, 5):
        # Convenzione slide: linee troppo lunghe spezzate da \
        print "|" + (-2 * line + 8) * " " + \
            "<>" + (4 * line - 4) * "." + "<>" + \
            (-2 * line + 8) * " " + "|"

def bottom():
    for line in range(4, 0, -1):
        print "|" + (-2 * line + 8) * " " + \
            "<>" + (4 * line - 4) * "." + "<>" + \
            (-2 * line + 8) * " " + "|"

# main
bar()
top()
bottom()
bar()
```

NON INSERITE IL CARATTERE \ NEL SORGENTE!



Concatenazione di range

- I range possono essere concatenati con il +
 - E' possibile ciclare attraverso insiemi **disgiunti** di numeri

```
>>> range(1, 5) + range(10, 15)
[1, 2, 3, 4, 10, 11, 12, 13, 14]

>>> for i in range(4) + range(10, 7, -1):
...     print(i)
0
1
2
3
10
9
8
```

Soluzione esercizio

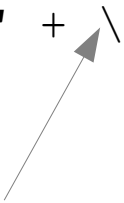
```
def bar():
    print "#" + 16 * "=" + "#"

def top():
    for line in range(1, 5):
        # Convenzione slide: linee troppo lunghe spezzate da \
        print "|" + (-2 * line + 8) * " " + \
              "<>" + (4 * line - 4) * "." + "<>" + \
              (-2 * line + 8) * " " + "|"

def bottom():
    for line in range(4, 0, -1):
        print "|" + (-2 * line + 8) * " " + \
              "<>" + (4 * line - 4) * "." + "<>" + \
              (-2 * line + 8) * " " + "|"

# main
bar()
top()
bottom()
bar()
```

NON INSERITE IL CARATTERE \ NEL SORGENTE!



Esercizio: soluzione 2

```
def bar():
    print "#" + 16 * "=" + "#"

def mirror():
    for line in range(1, 5) + range(4, 0, -1):
        print "|" + (-2 * line + 8) * " " + \
            "<>" + (4 * line - 4) * "." + "<>" + \
            (-2 * line + 8) * " " + "|"

# main
bar()
mirror()
bar()
```



Costanti

- Python non ha delle vere e proprie **costanti**.
 - Dichiariamo una variabile “globale” al di sopra del codice principale.
 - Tutti I metodi potranno usare questa variabile.

constant.py

```
1  MAX_VALUE = 3
2
3  def printTop():
4      for i in range(MAX_VALUE):
5          for j in range(i):
6              print(j)
7          print()
8
9  def printBottom():
10     for i in range(MAX_VALUE, 0, -1):
11         for j in range(i, 0, -1):
12             print(MAX_VALUE)
13     print()
```



Esercizio: soluzione 3

```
SIZE = 4

def bar():
    print "#" + 4 * SIZE * "=" + "#"

def mirror():
    for line in range(1, SIZE + 1) + range(SIZE, 0, -1):
        print "|" + (-2 * line + 2 * SIZE) * " " + \
            "<>" + (4 * line - 4) * "." + "<>" + \
            (-2 * line + 2 * SIZE) * " " + "|"

# main
bar()
mirror()
bar()
```





Unit 3

parameteri di funzione

Parametri di funzioni

```
def name (parameter, parameter, ...,  
         parameter) :  
    statements
```

- I parametri sono dichiarati scrivendo I loro nomi (e non il loro tipo)

```
>>> def print_many(message, n) :  
...     for i in range(n):  
...         print(message)  
  
>>> print_many("hello", 4)  
hello  
hello  
hello  
hello
```



Esercizio

- Ricreate questa serie di asterischi che disegnano linee e barre (rettangoli):

```
*****
```

```
*****
```

```
*****
```

```
*****
```

```
*           *
```

```
*****
```

```
*****
```

```
*           *
```

```
*           *
```

```
*****
```

Soluzione esercizio

stars.py

```
1  # Disegna un rettangolo di asterischi date larghezza e
2  altezza.
3  def box(width, height):
4      print(width * "*")
5      for i in range(height - 2):
6          print("*" + (width - 2) * " " + "*")
7      print(width * "*")
8
9  # main
10 print(13 * "*")
11 print( 7 * "*")
12 print(35 * "*")
13 box(10, 3)
   box(5, 4)
```



Valori di default

```
def name (parameter=value, ..., parameter=value) :  
    statements
```

- Potete rendere i parametri opzionali specificando un valore di default

```
>>> def print_many(message, n=1) :  
...     for i in range(n):  
...         print(message)  
  
>>> print_many("shrubbery")  
shrubbery  
>>> print_many("shrubbery", 3)  
shrubbery  
shrubbery  
shrubbery
```

- **Esercizio:** Modificare `stars.py` aggiungendo un parametro di default per il carattere da usare per il disegno dei rettangoli (default `"*"`).

Nomi dei parametri

name (parameter=value, ..., parameter=value)

- E' possibile specificare I nomi dei parametri durante la chiamata della funzione
- Questo permette di passare I parametri in qualsiasi ordine

```
>>> def print_many(str, n):  
...     for i in range(n):  
...         print(str)  
  
>>> print_many(str="shrubbery", n=4)  
shrubbery  
shrubbery  
shrubbery  
shrubbery  
>>> print_many(n=3, str="Ni!")  
Ni!  
Ni!  
Ni!
```





Unit 4

If/else, return, user input, stringhe

Funzioni matematiche

```
from math import *
```

Function name	Description
<code>abs (value)</code>	absolute value
<code>ceil (value)</code>	rounds up
<code>cos (value)</code>	cosine, in radians
<code>degrees (value)</code>	convert radians to degrees
<code>floor (value)</code>	rounds down
<code>log (value, base)</code>	logarithm in any base
<code>log10 (value)</code>	logarithm, base 10
<code>max (value1, value2, ...)</code>	larger of two (or more) values
<code>min (value1, value2, ...)</code>	smaller of two (or more) values
<code>radians (value)</code>	convert degrees to radians
<code>round (value)</code>	nearest whole number
<code>sin (value)</code>	sine, in radians
<code>sqrt (value)</code>	square root
<code>tan (value)</code>	tangent

Constant	Description
<code>e</code>	2.7182818...
<code>pi</code>	3.1415926...



Restituire valori

```
def name (parameters) :  
    statements  
    ...  
    return expression
```

- Python non richiede di specificare che una funzione ritorna un valore ... semplicemente restituite un valore come ultima istruzione della funzione.

```
>>> def ftoc(temp):  
...     tempc = 5.0 / 9.0 * (temp - 32)  
...     return tempc  
  
>>> ftoc(98.6)  
37.0
```



input

`input` : Legge una stringa dalla tastiera.

- Legge e restituisce un'intera riga di input *

```
>>> name = input("Howdy. What's yer name?")
Howdy. What's yer name? Mario Rossi

>>> name
'Mario Rossi'
```

** Queste slide si riferiscono alla versione 3.x di Python e successive. La vecchia versione 2.x di Python gestiva l'input in modo differente ...*



input

- Per leggere numeri, convertire il risultato di `input()` in un `int` o in un `float`
 - Se l'utente non inserisce un numero viene generato un errore.
 - Esempio:

```
age = int(input("How old are you? "))  
print("Your age is", age)  
print(65 - age, "years to retirement")
```

Output:

```
How old are you? 53  
Your age is 53  
12 years to retirement
```

if

if condition:
statements

- Esemplio:

```
gpa = float(input("What is your GPA? "))  
if gpa > 2.0:  
    print("Your application is accepted.")
```



if/else

```
if condition:  
    statements  
elif condition:  
    statements  
else:  
    statements
```

- Example:

```
gpa = float(input("What is your GPA? "))  
if gpa > 3.5:  
    print("You have qualified for the honor roll.")  
elif gpa > 2.0:  
    print("Welcome to Mars University!")  
else:  
    print("Your application is denied.")
```



if ... in

`if value in sequence:`
statements

- sequence puo' essere un range, una stringa, una tupla o una lista (trattate piu' avanti)
- Esempi:

```
x = 3
if x in range(0, 10):
    print("x is between 0 and 9")
```

```
letter = input("What is your favorite letter? ")
if letter in "aeiou":
    print("It is a vowel!")
```



Operatori logici

Operator	Meaning	Example	Result
==	equals	1 + 1 == 2	True
!=	does not equal	3.2 != 2.5	True
<	less than	10 < 5	False
>	greater than	10 > 5	True
<=	less than or equal to	126 <= 100	False
>=	greater than or equal to	5.0 >= 5.0	True

Operator	Example	Result
and	(2 == 3) and (-1 < 5)	False
or	(2 == 3) or (-1 < 5)	True
not	not (2 == 3)	True



Esercizio

- Scrivi un programma che legga le ore lavorate da due impiegati e scriva il totale e la media giornaliera (per impiegato) ed il totale di ore lavorate.
 - Riduci le ore inserite dall'utente ad un massimo di 8.

```
Employee 1: How many days? 3  
Hours? 6  
Hours? 12  
Hours? 5  
Employee 1's total hours = 19 (6.33 / day)
```

```
Employee 2: How many days? 2  
Hours? 11  
Hours? 6  
Employee 2's total hours = 14 (7.00 / day)
```

```
Total hours for both = 33
```



Stringhe

index	0	1	2	3	4	5	6	7
or	-8	-7	-6	-5	-4	-3	-2	-1
character	P	.		D	i	d	d	y

- Accesso ai caratteri:
 - variable** [**index**]
 - variable** [**index1:index2**]
 - **index2** exclusive
 - **index1** or **index2** can be omitted (goes to end of string)

```
>>> name = "P. Diddy"
>>> name[0]
'P'
>>> name[7]
'y'
>>> name[-1]
'y'
>>> name[3:6]
'Did'
>>> name[3:]
'Diddy'
>>> name[:-2]
'P. Did'
```

Metodi delle stringhe

Java	Python
length	len(str)
startsWith, endsWith	startswith, endswith
toLowerCase, toUpperCase	upper, lower, isupper, islower, capitalize, swapcase
indexOf	find
trim	strip

```
>>> name = "Martin Douglas Stepp"
>>> name.upper()
'MARTIN DOUGLAS STEPP'
>>> name.lower().startswith("martin")
True
>>> len(name)
20
```



Cicli `for` e stringhe

- Un ciclo `for` puo' accedere ad ogni carattere di una stringa nell'ordine in cui i caratteri compaiono nella stessa.

```
for name in string:  
    statements
```

```
>>> for c in "booyah":  
...     print c  
...  
b  
o  
o  
y  
a  
h
```

Formattazione del testo

"format string" % (parameter, parameter, ...)

- *Segnaposto* inseriscono **valori formattati** in una stringa:

- %d an integer
- %f a real number
- %s a string

- %8d an integer, 8 characters wide, right-aligned
- %08d an integer, 8 characters wide, padding with 0s
- %-8d an integer, 8 characters wide, left-aligned
- %12f a real number, 12 characters wide
- %.4f a real number, 4 characters after decimal
- %6.2f a real number, 6 total characters wide, 2 after decimal

```
>>> x = 3; y = 3.14159; z = "hello"
>>> print "%-8s, %04d is close to %.3f" % (z, x, y)
hello      , 0003 is close to 3.142
```



Stringhe e interi

- `ord(text)` - Converte una stringa in un numero.
 - `ord("a")` is 97
 - `ord("b")` is 98
 - Usa conversione standard (*ASCII e Unicode*).
- `chr(number)` - Converte un numero in una stringa.
 - `chr(97)` is "a"
 - `chr(99)` is "c"

Crittografia di base

- **Cifrario a rotazione** - scambia ogni lettera di un numero prefissato di posizioni
 - **Cifrario di Cesare** - scambia ogni lettera di 3 passi (in avanti)
"the cake is a lie" diventa
"wkh fdnh lv d olh"

abcdefghijklmnopqrstuvwxyz
↓
defghijklmnopqrstuvwxyzabc

- **Cifrario a sostituzione** - trasforma ognilettera in un'altra
 - Non adotta spostamenti costanti; deve esistere qualche forma di mapping (es. dizionario)
 - simile ai crittogrammi nelle riviste di enigmistica

Exercise

- Scrivere un programma che critti un messaggio segreto usando un cifrario di Cesare shiftando le lettere di 3 posizioni (in avanti):
 - es. "Attack" se rotato di 2 diventa "cwwcfn"
 - Se avete tempo rendete il programma capace di decrittare il testo.

abcdefghijklmnopqrstuvwxyz
↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓↓
defghijklmnopqrstuvwxyzabc

- Sapreste scrivere una funzione che implementa un cifrario a sostituzione?





Unit 5

Ciclo `while` ; flusso logico del programma; numeri casuali; tuple

Ciclo while

while **test**:
 statements

sentinel.py

```
1  # Somma interi inseriti dall'utente
2  # fino a quando un -1 viene inserito, usando un loop.
3  sum = 0
4  num = int(input("Type a number (-1 to quit)? "))
5
6  while n != -1:
7      sum += num
8      num = int(input("Type a number (-1 to quit)? "))
9
10 print("The total is", sum)
```



Random Numbers

```
from random import *
```

```
randint(min, max)
```

- Ritorna un intero casuale nel range [**min**, **max**] inclusi

```
choice(sequence)
```

- Restituisce un valore scelto a caso dalla sequenza data
- (la sequenza puo' essere un range, una stringa, un array, ...)

```
>>> from random import *
>>> randint(1, 5)
2
>>> randint(1, 5)
5
>>> choice(range(4, 20, 2))
16
>>> choice("hello")
'e'
```



while / else

```
while test:  
    statements
```

```
else:  
    statements
```

- Esegue la parte `else` se non si entra mai in loop
- Esosre uno statement `for / else` simile

```
>>> n = 91  
>>> while n % 2 == 1:  
...     n += 1  
... else:  
...     print(n, "was even; no loop.")  
...  
91 was even; no loop.
```



bool

- E' il tipo di dato logico di Python, equivalente ai boolean in Java
 - True e False iniziano con lettere maiuscole

```
>>> 5 < 10
True

>>> b = 5 < 10
>>> b
True

>>> if b:
...     print("The value is true")
...
The value is true

>>> b = not b
>>> b
False
```



Operatori logici

Operator	Meaning	Example	Result
==	equals	<code>1 + 1 == 2</code>	True
!=	does not equal	<code>3.2 != 2.5</code>	True
<	less than	<code>10 < 5</code>	False
>	greater than	<code>10 > 5</code>	True
<=	less than or equal to	<code>126 <= 100</code>	False
>=	greater than or equal to	<code>5.0 >= 5.0</code>	True

Operator	Example	Result
and	<code>2 == 3 and -1 < 5</code>	False
or	<code>2 == 3 or -1 < 5</code>	True
not	<code>not -1 < 5</code>	False

Esercizio

- Scrivere il programma `Dice` in Python. Il suo output e' il seguente (vittoria se due dadi restituiscono valori casuali che sommano a 7).

```
2 + 4 = 6
```

```
3 + 5 = 8
```

```
5 + 6 = 11
```

```
1 + 1 = 2
```

```
4 + 3 = 7
```

```
You won after 5 tries!
```

Tuple

tuple_name = (value, value, ..., value)

- Modo comodo per "impacchettare" **piu' valori** in un'unica variabile

```
>>> x = 3
>>> y = -5
>>> p = (x, y, 42)
>>> p
(3, -5, 42)
```

name, name, ..., name = tuple_name

- "estrarre" da tupla e assegnare a **piu' variabili**

```
>>> a, b, c = p
>>> a
3
>>> b
-5
>>> c
42
```



Using Tuples

- Utili per rappresentare dati **multidimensionali** (es. punti (x, y))

```
>>> p = (42, 79)
```

- Utili per ritornare **piu' di un valore**

```
>>> from random import *
>>> def roll2():
...     die1 = randint(1, 6)
...     die2 = randint(1, 6)
...     return (die1, die2)
...
>>> d1, d2 = roll2()
>>> d1
6
>>> d2
4
```



Tuple come parametri

```
def name ( (name, name, ..., name), ... ) :  
    statements
```

Dichiarare tuple come parametri assegnando nomi ad ognuna delle loro componenti

```
>>> def slope((x1, y1), (x2, y2)):  
...     return (y2 - y1) / (x2 - x1)  
...  
>>> p1 = (2, 5)  
>>> p2 = (4, 11)  
>>> slope(p1, p2)  
3
```

Tuple come valori di ritorno

```
def name (parameters) :  
    statements  
    return (name, name, ..., name)
```

```
>>> from random import *  
>>> def roll2():  
...     die1 = randint(1, 6)  
...     die2 = randint(1, 6)  
...     return (die1, die2)  
...  
>>> d1, d2 = roll2()  
>>> d1  
6  
>>> d2  
4
```



Unit 6

Processamento di file

Leggere File

name = open("filename")

- Apre il file di cui e' fornito il percorso in lettura e ritorna un oggetto di tipo file.

name.read() – Intero contenuto del file come stringa

name.readline() – Prossima linea del file come stringa

name.readlines() – Contenuto del file come lista di righe

- Le righe di un oggetto file possono anche essere lette usando un ciclo `for`

```
>>> f = open("hours.txt")
>>> f.read()
'123 Susan 12.5 8.1 7.6 3.2\n
456 Brad 4.0 11.6 6.5 2.7 12\n
789 Jenn 8.0 8.0 8.0 8.0 7.5\n'
```



Template file input

- Template generico per leggere file in Python:

```
name = open("filename")  
for line in name:  
    statements
```

```
>>> input = open("hours.txt")  
>>> for line in input:  
...     print(line.strip())    # strip() rimuove \n  
  
123 Susan 12.5 8.1 7.6 3.2  
456 Brad 4.0 11.6 6.5 2.7 12  
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

Esercizio

- Scrivere una funzione `input_stats` che accetta in input il nome di un file e ritorna la riga piu' lunga del file.

- Esempio di input file, `carroll.txt`:

```
Beware the Jabberwock, my son,  
the jaws that bite, the claws that catch,  
Beware the JubJub bird and shun  
the frumious bandersnatch.
```

- Output atteso:

```
>>> input_stats("carroll.txt")  
longest line = 42 characters  
the jaws that bite, the claws that catch,
```

Soluzione esercizio

```
def input_stats(filename):  
    input = open(filename)  
    longest = ""  
    for line in input:  
        if len(line) > len(longest):  
            longest = line  
  
    print("Longest line =", len(longest))  
    print(longest)
```



Ripasso: Metodi stringhe

Java	Python
length	len(str)
startsWith, endsWith	startswith, endswith
toLowerCase, toUpperCase	upper, lower, isupper, islower, capitalize, swapcase
indexOf	find
trim	strip
	ord, chr

```
>>> name = "Martin Douglas Stepp"  
>>> name.upper()  
'MARTIN DOUGLAS STEPP'  
>>> name.lower().startswith("martin")  
True  
>>> len(name)  
20
```



Spezzare stringhe

- `split` spezza una stringa in parti attraverso cui possiamo ciclare.

`name.split()`

#spezza agli spazi

`name.split(delimiter)`

#spezza al delimitatore

- `join` fa il contrario di `split` (ricuce)
`delimiter.join(list of tokens)`

```
>>> name = "Brave Sir Robin"
>>> for word in name.split():
...     print(word)
Brave
Sir
Robin
>>> "LL".join(name.split("r"))
'BLlave SiLL Robin'
```

Spezzare stringa in + variabili

- Se conosciamo il numero di parti in cui una stringa verra' spezzata possiamo assegnarle direttamente ad una sequenza di variabili.

```
var1, var2, ..., varN = string.split()
```

- E' possibile convertire il tipo di specifiche parti della stringa: **type (value)**

```
>>> s = "Jessica 31 647.28"  
>>> name, age, money = s.split()  
>>> name  
'Jessica'  
>>> int(age)  
31  
>>> float(money)  
647.28
```



Esercizio

- Supponiamo di avere i seguenti dati in `hours.txt` :

```
123 Suzy 9.5 8.1 7.6 3.1 3.2
456 Brad 7.0 9.6 6.5 4.9 8.8
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

- Calcolare per ogni lavoratore (uno per riga nel file) il totale di ore lavorate ed la media delle ore giornaliere.
 - Assumere che ogni lavoratore lavori esattamente 5 giorni.

```
Suzy ID 123 worked 31.4 hours: 6.3 / day
Brad ID 456 worked 36.8 hours: 7.36 / day
Jenn ID 789 worked 39.5 hours: 7.9 / day
```

Soluzione Esercizio

hours.py

```
1 input = open("hours.txt")
2 for line in input:
3     id, name, mon, tue, wed, thu, fri = line.split()
4
5     # Somma cumulativa delle ore di questo impiegato
6     hours = float(mon) + float(tue) + float(wed) + \
7             float(thu) + float(fri)
8
9     print(name, "ID", id, "worked", \
10           hours, "hours: ", hours/5, "/ day")
```



Scrivere su File

name = open("filename", "w")

name = open("filename", "a")

- Apre file in lettura (cancella contenuto), o
- Apre file per aggiungere righe (nuovi dati inseriti in coda al file)

name.write(**str**) - scrive la stringa str nel file

name.close() - salva il file quando la scrittura e' terminata

```
>>> out = open("output.txt", "w")
>>> out.write("Hello, world!\n")
>>> out.write("How are you?")
>>> out.close()
```

```
>>> open("output.txt").read()
'Hello, world!\nHow are you?'
```

Esercizio

- Scrivi del codice che legga il prezzo del gas negli USA ed in Belgio:

```
8.20    3.81    3/21/11
8.08    3.84    3/28/11
8.38    3.92    4/4/11
```

...

- Salva il prezzo medio del gas per ogni nazione in un file di output di nome `gasout.txt`.



Unit 7

Liste

Liste

- **liste**: equivalenti Python degli array Java (ma migliori)
 - Dichiarazione:
name = [**value**, **value**, ..., **value**] or,
name = [**value**] * **length**
 - Accesso/modifica elementi: (come in Java)
name[**index**] = **value**

```
>>> scores = [9, 14, 18, 19, 16]
[9, 14, 18, 19, 16]
>>> counts = [0] * 4
[0, 0, 0, 0]
>>> scores[0] + scores[4]
25
```


Indici

- Le liste si possono indicizzare utilizzando sia numeri positivi che negativi:

```
>>> scores = [9, 14, 12, 19, 16, 7, 24, 15]
[9, 14, 12, 19, 16, 7, 24, 15]
>>> scores[3]
19
>>> scores[-3]
7
```

index 0 1 2 3 4 5 6 7

<i>value</i>	9	14	12	19	16	7	24	15
--------------	---	----	----	----	----	---	----	----

index -8 -7 -6 -5 -4 -3 -2 -1

Richiamo: Stringhe

<i>index</i>	0	1	2	3	4	5	6	7
value	P	.		D	i	d	d	y
<i>-index</i>	-8	-7	-6	-5	-4	-3	-2	-1

- Accesso ai caratteri:
 - variable** [**index**]
 - variable** [**index1:index2**]
 - **index2** incluso
 - **index1** o **index2** si possono omettere(va fino a fine stringa)

```
>>> name = "P. Diddy"
>>> name[0]
'P'
>>> name[7]
'y'
>>> name[-1]
'y'
>>> name[3:6]
'Did'
>>> name[3:]
'Diddy'
>>> name[:-2]
'P. Did'
```

Slicing

- **slice**: Una sotto-lista creata specificando indice start e indice stop

name [**start**:**end**] # end e' escluso
name [**start**:] # fino a fine lista
name [:**end**] # da inizio lista
name [**start**:**end**:**step**] # ogni step-esimo valore

```
>>> scores = [9, 14, 12, 19, 16, 18, 24, 15]
>>> scores[2:5]
[12, 19, 16]
>>> scores[3:]
[19, 16, 18, 24, 15]
>>> scores[:3]
[9, 14, 12]
>>> scores[-3:]
[18, 24, 15]
```

<i>inde</i>	0	1	2	3	4	5	6	7
<i>x</i>								
<i>value</i>	9	14	12	19	16	18	24	15
<i>inde</i>	-8	-7	-6	-5	-4	-3	-2	-1
<i>x</i>								



Liste

- Le liste si possono stampare (o convertire in stringhe con `str()`).
- Lunghezza calcolata tramite funzione `len`.
- Ciclare attraverso una lista usando un loop `for ... in`.

```
>>> scores = [9, 14, 18, 19]
>>> print("My scores are", scores)
My scores are [9, 14, 18, 19]
>>> len(scores)
4
>>> total = 0
>>> for score in scores:
...     print("next score:", score)
...     total += score
next score: 9
next score: 14
next score: 18
next score: 19
>>> total
60
```



Range, Stringhe, e Liste

- La funzione `range` ritorna una lista.

```
>>> nums = range(5)
>>> nums
[0, 1, 2, 3, 4]
>>> nums[-2:]
[3, 4]
>>> len(nums)
5
```

- Le stringhe si comportano come liste di caratteri:
 - `len`
 - Indicizzazione e slicing
 - Loop `for ... in`

Splitting di stringhe

- `split` spezza una stringa in una lista di elementi.
`name.split()` # break by whitespace
`name.split(delimiter)` # break by delimiter
- `join` esegue l'operazione opposta di `split`
`delimiter.join(list)`

```
>>> name = "Brave Sir Robin"
>>> name[-5:]
'Robin'
>>> tokens = name.split()
['Brave', 'Sir', 'Robin']
>>> name.split("r")
['B', 'ave Si', ' Robin']
>>> "||".join(tokens)
'Brave||Sir||Robin'
```

Tokenizzazione File Input

- Use `split` per tokenizzare il contenuto delle righe lette da file.
 - Se e' necessaria una conversione: **type (value)**

```
>>> f = open("example.txt")
>>> line = f.readline()
>>> line
'hello world 42 3.14\n'

>>> tokens = line.split()
>>> tokens
['hello', 'world', '42', '3.14']

>>> word = tokens[0]
'hello'
>>> answer = int(tokens[2])
42
>>> pi = float(tokens[3])
3.14
```



Esercizio

- Riconsideriamo il file `hours.txt`. Supponiamo che # di giorni possa variare:

```
123 Susan 12.5 8.1 7.6 3.2
456 Brad 4.0 11.6 6.5 2.7 12
789 Jenn 8.0 8.0 8.0 8.0 7.5
```

- Calcolare il totale di ore lavorate da ogni lavoratore e la media di ore giornaliera.
 - Deve funzionare indipendentemente da quanti giorni ha lavorato un lavoratore

```
Suzy ID 123 worked 31.4 hours: 6.3 / day
Brad ID 456 worked 36.8 hours: 7.36 / day
Jenn ID 789 worked 39.5 hours: 7.9 / day
```



Soluzione Esercizio

hours.py

```
1 file = open("hours.txt")
2 for line in file:
3     tokens = line.split()
4     id = tokens[0]
5     name = tokens[1]
6
7     # somma cumulativa delle ore di questo impiegato
8     hours = 0.0
9     days = 0
10    for token in tokens[2:]:
11        hours += float(token)
12        days += 1
13
14    print(name, "ID", id, "worked", \
15          hours, "hours:", hours / days, "/ day")
```

Esercizio

- Supponiamo di avere un file contenente I punteggi ottenuti durante un parziale di meta' corso, `scores.txt`:

```
76
89
76
72
68
```

- Creare un istogramma degli score come segue:

```
75: *
76: *****
79: **
81: *****
82: *****
84: *****
```

Esercizio

- Supponiamo di avere dati scaricati da Internet Movie Database (IMDb):

```
1 9.1 196376 The Shawshank Redemption (1994)
2 9.0 139085 The Godfather: Part II (1974)
3 8.8 81507 Casablanca (1942)
```

- Scrivere un programma che cerchi in tutti I titoli di film a partire da una data parola:

Search word? part

```
Rank      Votes      Rating    Title
2         139085     9.0       The Godfather: Part II (1974)
40        129172     8.5       The Departed (2006)
95        20401      8.2       The Apartment (1960)
192       30587      8.0       Spartacus (1960)
4 matches.
```

Soluzione Esercizio

movies.py

```
1 search_word = input("Search word? ")
2 matches = 0
3 file = open("imdb.txt")
4 for line in file:
5     tokens = line.split()
6     rank = int(tokens[0])
7     rating = float(tokens[1])
8     votes = int(tokens[2])
9     title = " ".join(tokens[3:])
10
11     # il titolo contiene search_word?
12     if search_word.lower() in title.lower():
13         matches += 1
14         print(rank, "\t", votes, "\t", rating, "\t",
15 title)
16 print(matches, "matches.")
```



Unit 8

Classi e Oggetti; Ereditarieta'

OOP, Definire una Classe

- Python e' stato costruito come linguaggio procedurale ma
 - OOP (object oriented programming) esiste e funziona bene
- Dichiarazione di una classe:

```
class name:  
    statements
```



Campi (var. interne)

name = value

- Esempio:

```
class Point:  
    x = 0  
    y = 0
```

```
# main
```

```
p1 = Point()  
p1.x = 2  
p1.y = -5
```

point.py

```
1 class Point:  
2     x = 0  
3     y = 0
```

- Possono essere dichiarati direttamente all'interno della classe (come nell'esempio) o nei costruttori (piu' comune)
- Python non ha incapsulamento (campi privati)
 - Spera che il chiamante non cambi i valori dei campi interni ad una classe



Using a Class

import **class**

- Programmi client (utenti di classi esterne) DEVONO importare TUTTE le classi che usano.

point_main.py

```
1  from Point import *
2
3  # main
4  p1 = Point()
5  p1.x = 7
6  p1.y = -3
7  ...
8
9  # Oggetti Python sono dinamici (e' possibile aggiungere
10 # campi in qualsiasi momento)
    p1.name = "Tyler Durden"
```


Metodi degli oggetti

```
def name (self, parameter, ..., parameter) :  
    statements
```

- `self` DEVE essere il primo parametro di OGNI metodo di oggetto
 - Rappresenta il "parametro implicito" (`this` in Java)
- *L'ACCESSO ai campi di un oggetto passato come parametro avviene tramite un riferimento a `self`*

```
class Point:  
    def translate(self, dx, dy) :  
        self.x += dx  
        self.y += dy  
        ...
```



Parametro "Implicito" (self)

- Java: `this`, implicito

```
public void translate(int dx, int dy) {  
    x += dx;           // this.x += dx;  
    y += dy;           // this.y += dy;  
}
```

- Python: `self`, esplicito

```
def translate(self, dx, dy):  
    self.x += dx  
    self.y += dy
```

- **Esercizio:** Scrivete I metodi `distance`, `set_location`, e `distance_from_origin`.



Soluzione Esercizio

point.py

```
1  from math import *
2
3  class Point:
4      x = 0
5      y = 0
6
7      def set_location(self, x, y):
8          self.x = x
9          self.y = y
10
11     def distance_from_origin(self):
12         return sqrt(self.x * self.x + self.y * self.y)
13
14     def distance(self, other):
15         dx = self.x - other.x
16         dy = self.y - other.y
17         return sqrt(dx * dx + dy * dy)
```



Chiamate a Metodi

- Il programma client puo' chiamare I metodi di un oggetto in due modi:
 - (il valore di `self` puo' essere un parametro esplicito o implicito)

1) **object.method(parameters)**

oppure

2) **Class.method(object, parameters)**

- Esempio:

```
p = Point(3, -4)
```

```
p.translate(1, 5)
```

```
Point.translate(p, 1, 5)
```



Costruttori di classe

```
def __init__(self, parameter, ..., parameter):  
    statements
```

- un costruttore e' un metodo speciale di nome `__init__`
- Esempio:

```
class Point:  
    def __init__(self, x, y):  
        self.x = x  
        self.y = y  
    ...
```

- Come potremmo rendere possibile la costruzione di un oggetto `Point()` senza parametri ma ottenendo il punto (0, 0)?

toString and `__str__`

```
def __str__(self):  
    return string
```

- Equivalente del metodo `toString` di Java (converte oggetto in stringa)
- invocato automaticamente quando `str` o `print` vengono usate sull'oggetto

Esercizio: Scrivere un metodo `__str__` per oggetti di tipo `Point` che restituisca stringhe come questa: `"(3, -14)"`

```
def __str__(self):  
    return "(" + str(self.x) + ", " + str(self.y) + ")"
```



Classe Point completa

point.py

```
1  from math import *
2
3  class Point:
4      def __init__(self, x, y):
5          self.x = x
6          self.y = y
7
8      def distance_from_origin(self):
9          return sqrt(self.x * self.x + self.y * self.y)
10
11     def distance(self, other):
12         dx = self.x - other.x
13         dy = self.y - other.y
14         return sqrt(dx * dx + dy * dy)
15
16     def translate(self, dx, dy):
17         self.x += dx
18         self.y += dy
19
20     def __str__(self):
21         return "(" + str(self.x) + ", " + str(self.y) + ")"
```



Operator Overloading

- **operator overloading:** You can define functions so that Python's built-in operators can be used with your class.

- See also: <http://docs.python.org/ref/customization.html>

Operator	Class Method
-	<code>__neg__(self, other)</code>
+	<code>__pos__(self, other)</code>
*	<code>__mul__(self, other)</code>
/	<code>truediv__(self, other)</code>
Unary Operators	
-	<code>__neg__(self)</code>
+	<code>__pos__(self)</code>

Operator	Class Method
==	<code>__eq__(self, other)</code>
!=	<code>__ne__(self, other)</code>
<	<code>__lt__(self, other)</code>
>	<code>__gt__(self, other)</code>
<=	<code>__le__(self, other)</code>
>=	<code>__ge__(self, other)</code>

Esercizio

- Esercizio: Scrivere una classe `Fraction` per rappresentare numeri razionali come $1/2$ e $-3/8$.
- Le frazioni dovrebbero sempre essere salvate in forma ridotta; ad esempio, salvare $4/12$ come $1/3$ e $6/-9$ come $-2/3$.
 - S suggerimento: una funzione MCD (massimo comun divisore) puo' essere di aiuto.
- Definire metodi `add` e `multiply` che accettano come parametro un'altra variabile `Fraction` e modifichi la variabile corrente `Fraction` aggiungendo ad essa/moltiplicandola per il parametro passato alla funzione.
- Definire gli operatori `+`, `*`, `==`, e `<`.



Generare Eccezioni

```
raise ExceptionType ("message")
```

- Utile quando le classi che abbiamo scritto vengono usate in modo inappropriato
- **Tipi di eccezione:** `ArithmeticError`, `AssertionError`, `IndexError`, `NameError`, `SyntaxError`, `TypeError`, `ValueError`

- **Esempio:**

```
class BankAccount:  
    ...  
    def deposit(self, amount):  
        if amount < 0:  
            raise ValueError("negative amount")  
        ...
```

Ereditarieta'

```
class name (superclass) :  
    statements
```

- Esempio:

```
class Point3D(Point) :      # Point3D estende Point  
    z = 0  
    ...
```

- Python supporta *ereditarieta' multipla*

```
class name (superclass, ..., superclass) :  
    statements
```

(NB: se piu' superclassi hanno gli stessi campi/metodi, I conflitti sono risolti da sinistra a destra seguendo l'ordine nella definizione della classe)



Chiamare metodi delle Superclassi

- metodi: **class.method(object, parameters)**
- costruttori: **class.__init__(parameters)**

```
class Point3D(Point):  
    z = 0  
    def __init__(self, x, y, z):  
        Point.__init__(self, x, y)  
        self.z = z  
  
    def translate(self, dx, dy, dz):  
        Point.translate(self, dx, dy)  
        self.z += dz
```

