

Laboratorio 01

Programmazione - CdS Matematica

Mirko Polato

6 Novembre 2018

Prendiamo confidenza

Apriamo la console di Python:

```
python
Python 2.7.3 (default, Sep 26 2013, 20:03:06)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for
    more information.
>>>
```

- Informazioni sulla versione di Python
- Informazioni sulla nostra architettura
- Informazioni di servizio

Prendiamo confidenza

```
>>> 1 + 1
```

```
2
```

Prendiamo confidenza

```
>>> 1 + 1
2
>>> # io sono un commento
... 5 * 7
35
```

Prendiamo confidenza

```
>>> 1 + 1
```

```
2
```

```
>>> # io sono un commento
```

```
... 5 * 7
```

```
35
```

```
>>> _ - 5
```

```
30
```

Prendiamo confidenza

```
>>> 1 + 1
2
>>> # io sono un commento
... 5 * 7
35
>>> _ - 5
30
>>> "io sono una stringa"
'io sono una stringa'
```

Prendiamo confidenza

```
>>> 1 + 1
2
>>> # io sono un commento
... 5 * 7
35
>>> _ - 5
30
>>> "io sono una stringa"
'io sono una stringa'
>>> "io sono
    File "<stdin>", line 1
        "io sono
            ^
SyntaxError: EOL while scanning string literal
>>>
```

Prendiamo confidenza

- Ripulire lo schermo: Ctrl + L
- Accedere all'aiuto interattivo:

```
>>> help()
[...]
```

To quit this help utility and return to the interpreter, just type "quit".

To get a list of available modules, keywords, or topics, type "modules", "keywords", or "topics".

```
[...]
help>
```

- Uscire da Python:

```
>>> quit()
terminale:
```

Definire un intero x e verificarne il tipo

Definire un intero `x` e verificarne il tipo

```
>>> x = 42  
>>> type(x)  
<type 'int'>
```

Definire un intero x e verificarne il tipo

```
>>> x = 42  
>>> type(x)  
<type 'int'>
```

Definire un numero in virgola mobile y e verificarne il tipo

Definire un intero x e verificarne il tipo

```
>>> x = 42  
>>> type(x)  
<type 'int'>
```

Definire un numero in virgola mobile y e verificarne il tipo

```
>>> y = 6.28  
>>> type(y)  
<type 'float'>
```

Definire un numero complesso z , verificarne il tipo e stampare parte reale e immaginaria

Definire un numero complesso z , verificarne il tipo e stampare parte reale e immaginaria

```
>>> z = 3.12 + 3j  
>>> type(z)  
<type 'complex'>
```

Tipi numerici

Definire un numero complesso z , verificarne il tipo e stampare parte reale e immaginaria

```
>>> z = 3.12 + 3j
>>> type(z)
<type 'complex'>
>>> z.real
3.12
>>> z.imag
3.0
```

Tipi Booleani

I valori Booleani...

```
>>> True
True
>>> type(True)
<type 'bool'>
>>> not True
False
>>>
```

Tipi Booleani

I valori Booleani...

```
>>> True
True
>>> type(True)
<type 'bool'>
>>> not True
False
>>>
>>> True and False
False
>>> True and True
True
```

Tipi Booleani

I valori Booleani...

```
>>> True
True
>>> type(True)
<type 'bool'>
>>> not True
False
>>>
>>> True and False
False
>>> True and True
True
>>> False or False
False
>>> True or False
True
```

Tipi Booleani

...che, in effetti, sono numeri

```
>>> True == 1
True
>>> False == 0
True
>>>
```

Tipi Booleani

...che, in effetti, sono numeri

```
>>> True == 1
True
>>> False == 0
True
>>>
>>> 7 + True
```

Tipi Booleani

...che, in effetti, sono numeri

```
>>> True == 1
True
>>> False == 0
True
>>>
>>> 7 + True
8
>>> True + True + True
```

Tipi Booleani

...che, in effetti, sono numeri

```
>>> True == 1
True
>>> False == 0
True
>>>
>>> 7 + True
8
>>> True + True + True
3
>>> 5 * False
```

Tipi Booleani

...che, in effetti, sono numeri

```
>>> True == 1
True
>>> False == 0
True
>>>
>>> 7 + True
8
>>> True + True + True
3
>>> 5 * False
0
```

Gerarchia tipi numerici

`bool` $\subset$ `int` $\subset$ `float` $\subset$ `complex`

Operazioni che coinvolgono tipi diversi, trasformano i numeri nel loro tipo *più grande*

Gerarchia tipi numerici

$\text{bool} \subset \text{int} \subset \text{float} \subset \text{complex}$

Operazioni che coinvolgono tipi diversi, trasformano i numeri nel loro tipo *più grande*

```
>>> x = True
>>> type(x)
<type 'bool'>
>>> y = 1 + 2j
>>> type(y)
<type 'complex'>
>>> z = x + y
```

Gerarchia tipi numerici

$\text{bool} \subset \text{int} \subset \text{float} \subset \text{complex}$

Operazioni che coinvolgono tipi diversi, trasformano i numeri nel loro tipo *più grande*

```
>>> x = True
>>> type(x)
<type 'bool'>
>>> y = 1 + 2j
>>> type(y)
<type 'complex'>
>>> z = x + y
>>> type(z)
<type 'complex'>
>>> z
(2+2j)
```

Conversione tipi numerici

$\text{bool} \subset \text{int} \subset \text{float} \subset \text{complex}$

```
>>> x = 3.999
>>> type(x)
<type 'float'>
>>> y = int(x)
>>> type(y)
<type 'int'>
>>> y
```

Conversione tipi numerici

$\text{bool} \subset \text{int} \subset \text{float} \subset \text{complex}$

```
>>> x = 3.999
>>> type(x)
<type 'float'>
>>> y = int(x)
>>> type(y)
<type 'int'>
>>> y
3
```

Conversione tipi numerici

```
>>> x = True
>>> type(x)
<type 'bool'>
>>> f = float(x)
```

Conversione tipi numerici

```
>>> x = True
>>> type(x)
<type 'bool'>
>>> f = float(x)
>>> type(f)
<type 'float'>
>>> f
1.0
```

Alcuni operatori

Alcuni operatori e riassegnamenti

```
>>> x = 17 # x adesso riferisce l'oggetto 17
>>> x += 6 # equivalente a x = x + 6, ovvero x -> 23
>>> x *= 2 # equivalente a x = x * 2, ovvero x -> 46
>>> x -= 2 # equivalente a x = x - 2, ovvero x -> 44
>>> x /= 4 # equivalente a x = x / 4, ovvero x -> 11
>>> x %= 3 # equivalente a x = x % 3 (resto div intera),
           ovvero x -> 2
>>> x
2
>>> x ** 2 # elevamento a potenza
4
```

Priorità degli operatori

In generale: “parentesi” → elevamento a potenza → moltiplicazione e divisione → addizione e sottrazione.

```
>>> 3*(5-3) # prima valuta dentro la parentesi, poi il  
        prodotto
```

```
6
```

```
>>> (7-4)**(2+1) # prima valuta dentro le parentesi, poi  
        la potenza
```

```
27
```

```
>>> 3*2**4 # potenza, poi prodotto
```

```
48
```

Priorità degli operatori

```
>>> -6+3*2**4 # potenza, poi prodotto, poi somma
42
>>> 7+11-1 # da sx a dx
17
>>> 45/15*3 # da sx a dx
9
>>> 45/(15*3) # prima la parentesi
1
```

Esercizio

Dati $x = 17$, $y = 5$ e $z = 1$:

- Calcolare la differenza tra x e y e salvare il risultato in z
- Porre x uguale a y
- Incrementare y di 3

Domanda 1: Quanto vale x ?

Esercizio

Dati $x = 17$, $y = 5$ e $z = 1$:

- Calcolare la differenza tra x e y e salvare il risultato in z
- Porre x uguale a y
- Incrementare y di 3

Domanda 1: Quanto vale x ?

```
>>> x = 17
>>> y = 5
>>> z = 1
>>> z = x - y
>>> x = y
>>> y += 3
>>> x
```

Esercizio

Dati $x = 17$, $y = 5$ e $z = 1$:

- Calcolare la differenza tra x e y e salvare il risultato in z
- Porre x uguale a y
- Incrementare y di 3

Domanda 1: Quanto vale x ?

```
>>> x = 17
>>> y = 5
>>> z = 1
>>> z = x - y
>>> x = y
>>> y += 3
>>> x
```

Esercizio

- Calcolare il prodotto tra y , z e x e salvare il risultato in z

Domanda 2: Quanto vale z ?

Esercizio

- Calcolare il prodotto tra y , z e x e salvare il risultato in z

Domanda 2: Quanto vale z ?

```
>>> z *= x * y  
>>> z
```

Esercizio

- Calcolare il prodotto tra y , z e x e salvare il risultato in z

Domanda 2: Quanto vale z ?

```
>>> z *= x * y  
>>> z
```

480

Esercizio

- Calcolare il prodotto tra y , z e x e salvare il risultato in z

Domanda 2: Quanto vale z ?

```
>>> z *= x * y  
>>> z
```

480

- Decrementare y di 2

Domanda 3: Quanto vale z ?

Esercizio

- Calcolare il prodotto tra y , z e x e salvare il risultato in z

Domanda 2: Quanto vale z ?

```
>>> z *= x * y  
>>> z
```

480

- Decrementare y di 2

Domanda 3: Quanto vale z ?

```
>>> y -= 2  
>>> z
```

Esercizio

- Calcolare il prodotto tra y , z e x e salvare il risultato in z

Domanda 2: Quanto vale z ?

```
>>> z *= x * y  
>>> z
```

480

- Decrementare y di 2

Domanda 3: Quanto vale z ?

```
>>> y -= 2  
>>> z
```

480

- Convertire y in formato `float`
- Salvare in x il risultato di $z^{1/y}$

Domanda 4: Verificare che le prime tre cifre decimali di x valgono 798

- Convertire `y` in formato `float`
- Salvare in `x` il risultato di $z^{1/y}$

Domanda 4: Verificare che le prime tre cifre decimali di `x` valgono 798

```
>>> y = float(y)
>>> x = z**(1/y)
>>> x
```

- Convertire `y` in formato `float`
- Salvare in `x` il risultato di $z^{1/y}$

Domanda 4: Verificare che le prime tre cifre decimali di `x` valgono 798

```
>>> y = float(y)
>>> x = z**(1/y)
>>> x
```

2.7981664143395273

Esercizio sulle precedenze

Calcolare le seguenti espressioni per $x = 3.14$, $x = 23$

1. $3x^2 + 8\frac{4^2}{2}x - 6$ risultati attesi:

- con $x = 3.14$, 224.5388
- con $x = 23$, 3053

2. $(x + \frac{1}{2})^3 + x^{\frac{1}{3}} + \frac{x}{2} - 3.13$

- con $x = 3.14$, 48.13288835050312
- con $x = 23$, 12989.088866979853

Soluzione $3x^2 + 8\frac{4^2}{2}x - 6$

```
>>> x = 3.14
```

```
>>> 3*x**2 + 8*4**2/2*x - 6  
224.5388
```

```
>>> x = 23
```

```
>>> 2*x + 8 * 4**2 / 2  
3053
```

Soluzione $3x^2 + 8\frac{4^2}{2}x - 6$

```
>>> x = 3.14
>>> 3*x**2 + 8*4**2/2*x - 6
224.5388
>>> x = 23
>>> 2*x + 8 * 4**2 / 2
3053
```

Soluzione $(x + \frac{1}{2})^3 + x^{\frac{1}{3}} + \frac{x}{2} - 3.13$

```
>>> x = 3.14
>>> (x+1.0/2)**3 + x**(1.0/3) + x/2.0 - 3.13
48.13288835050312
>>> x = 23
>>> (x+1.0/2)**3 + x**(1.0/3) + x/2.0 - 3.13
12989.088866979853
```

Esercizio

Dato un oggetto di tipo numerico contenente una temperatura in gradi Celsius, determinare la temperatura equivalente in gradi Fahrenheit per i valori: -273.15, 0, 36, 100.

Nota: $t_F = \frac{9}{5}t_C + 32$

Esercizio

Dato un oggetto di tipo numerico contenente una temperatura in gradi Celsius, determinare la temperatura equivalente in gradi Fahrenheit per i valori: -273.15, 0, 36, 100.

Nota: $t_F = \frac{9}{5}t_C + 32$

```
>>> temp_c = -273.15
>>> temp_f = 9.0/5 * temp_c + 32
>>> temp_f
-459.66999999999996
>>> temp_c = 0
>>> temp_f = 9.0/5 * temp_c + 32
>>> temp_f
32.0
```

Le stringhe sono sequenze di caratteri.

Esempio:

```
>>> s1 = 'I tropici mamma! mi ci porti?'  
>>> s2 = "E l'idea era lecita. Li' ai lati c'e' l'area  
        edile."
```

Operazioni su stringhe (*overloading*):

```
>>> s1 = "ABE "  
>>> s2 = "C'E' DARIO?"  
>>> s1 + s2  
'ABE C'E' DARIO?'  
>>> s1 * 2  
'ABE ABE '  
>>> s1 * s2  
Traceback (most recent call last):  
  File "<stdin>", line 1, in <module>  
TypeError: can't multiply sequence by non-int of type '  
    str'
```

Date le assegnazioni

```
>>> a = "six"  
>>> b = a  
>>> c = " > "  
>>> d = "ty"  
>>> e = "1"
```

modificare le variabili usando **SOLO** i valori di *a*, *b*, *c*, *d*, *e* in modo che l'espressione *a+c+e+b* produca *sixty > 11 > six*, ovvero:

```
>>> a + c + e + b  
'sixty > 11 > six'
```

```
>>> a = "six"
>>> b = a
>>> c = " > "
>>> d = "ty"
>>> e = "1"
>>>
>>> a += d # a -> 'sixty'
>>> e *= 2 # equivalente a e = 2*e, e -> '11'
>>> e += c # e -> '11 > '
>>> a + c + e + b
'sixty > 11 > six'
```

Conversioni numeri e stringhe

```
>>> x = "23"  
>>> type(x)  
<type 'str'>  
>>> y = int(x)  
>>> type(y)  
<type 'int'>  
>>>
```

Conversioni numeri e stringhe

```
>>> x = "23"
>>> type(x)
<type 'str'>
>>> y = int(x)
>>> type(y)
<type 'int'>
>>>
>>> float("23")
23.0
>>> complex("12.34+5.6j")
(12.34+5.6j)
```

Conversioni numeri e stringhe

```
>>> float(True)
1.0
>>> float("True")
```

Conversioni numeri e stringhe

```
>>> float(True)
1.0
>>> float("True")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: True
>>>
```

Conversioni numeri e stringhe

```
>>> float(True)
1.0
>>> float("True")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: could not convert string to float: True
>>>
```

```
>>> str(3.14)
'3.14'
>>> type(str(3.14))
<type 'str'>
```

Operazioni su stringhe i

```
>>> "mirko".capitalize()
'Mirko'
>>> "mirko".center(11)
'  mirko  '
>>> "una stringa a caso".count('a')
4
>>> 'ab3de6'.isalnum()
True
>>> 'ab3 de6'.isalnum()
False
>>> 'mirko'.isalpha()
True
```

Operazioni su stringhe ii

```
>>> '314'.isdigit()
True
>>> '3.14'.isdigit()
False
>>> 'abracadabra'.islower()
True
>>> "ab;.bc;.cd".replace(';.',' -')
'ab-bc-cd'
>>> " mirko      ".strip()
'mirko'
>>> "mIrKo".swapcase()
'MiRkO'
>>> "mIrkO".upper()
'MIRKO'
```

Trasformare la stringa

```
>>> " pro-gram-ma-zio-ne " #1 spazio a sx, 3 a dx
```

nella stringa

```
>>> "  PROGRAMMAZIONE " #2 spazi a sx, 2 a dx
```

utilizzando gli operatori su stringhe.

Riuscite in un'unica riga di codice python?

Trasformare la stringa

```
>>> " pro-gram-ma-zio-ne      " #1 spazio a sx, 3 a dx
```

nella stringa

```
>>> "  PROGRAMMAZIONE  " #2 spazi a sx, 2 a dx
```

utilizzando gli operatori su stringhe.

Riuscite in un'unica riga di codice python?

Soluzione

```
>>> s = " pro-gram-ma-zio-ne      "  
>>> s.replace("-", "").strip().upper().center(18)  
'  PROGRAMMAZIONE  '
```

<http://en.wikipedia.org/wiki/ASCII>

ASCII in Python

```
>>> ord("a")
97
>>> ord("z")
122
>>> ord('z') - ord('a') + 1 # caratteri tra 'a' e 'z'
26
```

```
>>> chr(65)
'A'
>>> chr(90)
'Z'
>>> chr(ord('A') + 6)
'G'
```

Esercizio – Cifrario di Cesare

Il funzionamento del Cifrario di Cesare con chiave $k = 3$ é il seguente:

Caratteri originali: abcdefghijklmnopqrstuvwxyz

Caratteri cifrati: defghijklmnopqrstuvwxyzabc

- Data chiave k (es. $k = 3$)
- Dato un singolo carattere `chiaro` (es. `chiaro = 'a'`)
- Scrivere il codice python per codificare il carattere contenuto in `chiaro`

Attenzione: il codice Python deve essere una sola riga e deve funzionare per qualsiasi valore di k e qualsiasi singolo carattere da `chiaro = 'a'` a `chiaro = 'z'`

Esercizio – Cifrario di Cesare

Idea generale: $E_n(x) = (x + k) \bmod 26$

Completare il seguente codice:

```
>>> k = 3
>>> chiaro = "a"
>>> ...
'd'
>>>
>>> chiaro = "z"
>>> ...
'c'
```

Esercizio – Cifrario di Cesare

Idea generale: $E_n(x) = (x + k) \bmod 26$

Soluzione:

```
>>> k = 3
>>> chiaro = "a"
>>> chr(((ord(chiaro) - ord("a") + k) % 26) + ord("a"))
'd'
>>>
>>> chiaro = "z"
>>> chr(((ord(chiaro) - ord("a") + k) % 26) + ord("a"))
'c'
```

I moduli sono insiemi di funzionalità che assolvono a compiti particolari.

<http://docs.python.org/2/> → “Global Module Index”

Esempio d'uso moduli

Per poter utilizzare un modulo lo si deve importare

```
>>> import math
>>>
```

Una volta importato lo si può utilizzare

```
>>> math.pi
3.141592653589793
>>> math.cos(2 * math.pi)
1.0
>>>
>>> math.fabs(-1)
1.0
>>> math.ceil(3.6)
4.0
>>> int(3.6) == int(math.ceil(3.6))
False
```