Laboratorio 01

Programmazione - CdS Matematica

Michele Donini
27 Ottobre 2015

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:
```

Tipi numerici

Definire un intero `x` e verificarne il tipo

Tipi numerici

Definire un intero `x` e verificarne il tipo

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

Tipi numerici

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

Tipi numerici

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 = 3.14
>>> type(y)
<type 'float'>
```

Tipi numerici

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 = 3.14
>>> type(y)
<type 'float'>
```

Definire complesso `z`, verificare tipo e stampare parte reale e imm.

Tipi numerici

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 = 3.14
>>> type(y)
<type 'float'>
```

Definire complesso `z`, verificare tipo e stampare parte reale e imm.

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

Tipi numerici

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 = 3.14
>>> type(y)
<type 'float'>
```

Definire complesso `z`, verificare tipo e stampare parte reale e imm.

```
>>> 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

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

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

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

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

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

Conversione tipi numerici

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

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

Conversione tipi numerici

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

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

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

Conversione tipi numerici

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

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

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

Alcuni operatori

Alcuni operatori e riassegnamenti

```
>>> x = 23 # x adesso riferisce l'oggetto 23
>>> x += 3 # equivalente a x = x + 3, ovvero x -> 26
>>> x /= 2 # equivalente a x = x / 2, ovvero x -> 13
>>> x *= 3 # equivalente a x = x * 3, ovvero x -> 39
>>> x -= 4 # equivalente a x = x - 4, ovvero x -> 35
>>> x %= 4 # equivalente a x = x % 4 (resto div intera),
     ovvero x -> 3
>>> x
3
>>> x ** 2 # elevamento a potenza
9
```

Priorità degli operatori

In generale: "parentesi" $\rightarrow$ elevamento a potenza $\rightarrow$ moltiplicazione e divisione $\rightarrow$ addizione e sottrazione.

```
>>> 2 * (4-1) # prima valuta dentro la parentesi, poi il
    prodotto
6
>>> (3-1)**(4-1) # prima valuta dentro le parentesi, poi
    la potenza
8
>>> 2*2**3 # potenza, poi prodotto
16
>>> 1+2*2**3 # potenza, poi prodotto, poi somma
17
>>> 3+4-2 # da sx a dx
5
>>> 20/10*2 # da sx a dx
4
>>> 20/(10*2) # prima la parentesi
1
```

Esercizio

Dati `x = 1`, `y = 3` e `z = 0`:

- Calcolare la somma tra `x` e `y` e salvare il risultato in `z`
- Porre `x` uguale a `y`
- Incrementare `y` di 2

Domanda 1: Quanto vale `x`?

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

Domanda 2: Quanto vale `z`?

- Decrementare `y` di 1

Domanda 3: Quanto vale `z`?

- 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 783

Soluzione I

```
>>> x = 1
>>> y = 3
>>> z = 0
>>> z = x + y
>>> x = y
>>> y += 2
>>> x
```

Domanda 1: Quanto vale `x`? 3

```
>>> z *= x * y
>>> z
60
```

Domanda 2: Quanto vale `z`? 60

Soluzione II

```
>>> y -= 1
>>> z
60
```

Domanda 3: Quanto vale `z`? 60

```
>>> y = float(y)
>>> x = z**(1/y)
>>> x
2.7831576837137404
```

Domanda 4: Verificare che le prime tre cifre decimali di `x` valgono 783

Esercizio sulle precedenze I

Calcolare le seguenti espressioni per `x = 1`, `x = 5`

1. $2x + 8\frac{4^2}{2}$ risultati attesi:
   - con `x = 1`, 66
   - con `x = 5`, 74
2. $2x + 4^{1/2}$
   - con `x = 1`, 4.0
   - con `x = 5`, 12.0

Esercizio sulle precedenze II

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

```
>>> x = 1
>>> 2*x + 8 * 4**2 / 2
66
>>> x = 5
>>> 2*x + 8 * 4**2 / 2
74
```

Soluzione $2x + 4^{1/2}$

```
>>> x = 1
>>> 2*x + 4**(1.0/2)
4.0
>>> x = 5
>>> 2*x + 4**(1.0/2)
12.0
```

Esercizio

Calcolare le seguenti espressioni per `x = 1, y = 2`

$$2x + 4^{x/y}$$

Risultato atteso: 4.0

Esercizio

Calcolare le seguenti espressioni per `x = 1, y = 2`

$$2x + 4^{x/y}$$

Risultato atteso: 4.0

```
>>> x = 1
>>> y = 2
>>> 2*x + 4**(float(x)/y)
4.0
```

Esercizio

Dato un oggetto di tipo numerico contenente una temperatura in gradi Celsius, determinare la temperatura equivalente in gradi Farenheit 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 Farenheit 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
```

Stringhe

Le stringhe sono sequenze di caratteri.
Esempio:

```
>>> s1 = 'I topi non avevano nipoti'
>>> s2 = "Alle carte t'alleni nella tetra cella"
```

Stringhe

Le stringhe sono sequenze di caratteri.
Esempio:

```
>>> s1 = 'I topi non avevano nipoti'
>>> s2 = "Alle carte t'alleni nella tetra cella"
```

Operazioni su stringhe (*overloading*):

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

Esercizio

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'
```

Soluzione

```
>>> 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 = "3"
>>> type(x)
<type 'str'>
>>> y = int(x)
>>> type(y)
<type 'int'>
>>>
```

Conversioni numeri e stringhe

```
>>> x = "3"
>>> type(x)
<type 'str'>
>>> y = int(x)
>>> type(y)
<type 'int'>
>>>
>>> float("3")
3.0
>>> complex("4.3+2.1j")
(4.3+2.1j)
```

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

```
>>> "abcde".capitalize()
'Abcde'
>>> "abcde".center(10)
'  abcde   '
>>> "abcbcabc".count('bc')
3
>>> 'ab3 ab2'.isalnum()
False
>>> 'abab'.isalpha()
True
>>> '234'.isdigit()
True
```

Operazioni su stringhe II

```
>>> 'abracadabra'.islower()
True
>>> "ab;.bc;.cd".replace(';.','-')
'ab-bc-cd'
>>> "  abc        ".strip()
'abc'
>>> "aAbBcC".swapcase()
'AaBbCc'
>>> "aBBbbc".upper()
'ABBBBC'
```

American Std. Code for Inf. Interchange

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: $E_n(x) = (x + k) \mod 26$
Completare il seguente codice:

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

Esercizio – Cifrario di Cesare

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

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

Moduli

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
```