Autori: Domenico Lembo, Antonella Poggi, Giuseppe Santucci e Marco Schaerf

Dipartimento di Ingegneria informatica, automatica e gestionale



# Introduzione a Python

1. Usare la Shell dei comandi
2. Primi esempi di operazioni su numeri
3. Importazione di funzioni (o moduli) esterne: `math` e `random`
4. Primi esempi di operazioni su stringhe

## Avviare Python

Cominciamo a vedere Python usando la shell dei comandi. Avviando Spyder verrà aperta la seguente finestra (ambiente windows)

In basso a destra c'è la Shell di Python, dove è possibile digitare ed eseguire singole istruzioni Python.

A sinistra c'è l'editor standard di script Python, dove è possibile digitare, salvare, ed eseguire insiemi di istruzioni Python.

In questo corso useremo 2 ambienti di programmazione Python:

- **Spyder**: In laboratorio e all'esame programmeremo usando Spyder (vedi immagini sopra)
- **JupyterLab**: Ambiente interattivo (quello che stiamo usando ora) che permette di integrare parti descrittive con celle di codice eseguibili e modificabili. Useremo questo ambiente a lezione per la sua facilità di integrare spiegazioni e codice

Un'importante differenza tra i due ambienti è che in Spyder si tende a scrivere il codice in un unico blocco nell'editor e a eseguirlo tutto contemporaneamente, mentre nei file JupyterLab si tende a spezzare il codice su più blocchi e a eseguirne una parte alla volta (oltre che a inframezzare il codice con parti esplicative).

Questa differenza, come vedremo più avanti, ha importanti ripercussioni.

Proviamo ad inserire il comando `7*(5+2)`

In [ ]:
```
7*(5+2)
```

Ora proviamo ad inserire il comando `21**31`. Notate che il simbolo `**` denota l'elevazione a potenza

In [ ]:
```
21**31
```

## Espressioni e operazioni su numeri

Una espressione è composta da operandi e operatori e, una volta valutata, restituisce un valore.

La più semplice espressione aritmetica è un singolo numero, e.g, 7 (valore restituito 7).

Python distingue tra due tipi di dati numerici:

- numeri interi (int); esempi: 12, -9, 1
- numeri frazionari (float); esempi: 3.14, -45.2, 1.0, -42.3e-4 (notazione esponenziale che rappresenta il numero $-42,3\times10^{-4}$)

Nota bene: Python, come tutti i linguaggi di programmazione, usa lo standard americano, ovvero il punto, per separare la parte intera di un reale da quella frazionaria: 3.14 e non 3,14.

Espressione aritmetiche più complesse si ottengono combinando numeri attraverso operatori (addizione, divisione, ecc.), e usando le parentesi tonde per definire la precedenza tra gli operatori. E.g., 3*(5+2) (valore restituito 21).

Gli operatori disponibili nel linguaggio Python sono i seguenti:

Simbolo | Descrizione Operatore :---: |:-------: + | somma - | sottrazione * | moltiplicazione / | divisione // | divisione intera % | modulo (resto di una divisione) ** | elevamento a potenza

Se entrambi gli operandi di +, - e * sono interi, il risultato è rappresentato come intero, altrimenti è rappresentato come numero frazionario.

In [ ]:
```
1 + 5
```

In [ ]:
```
1.0 + 5
```

Indipendentemente dal fatto che gli operandi di / siano interi o frazionari, il quoziente restituito è sempre frazionario

In [ ]:
```
2/5
```

In [ ]:
```
6/2
```

L'operatore // produce sempre *il più grande intero non maggiore del quoziente restituito dalla divisione /*, rappresentato come intero se entrambi gli operandi sono interi, altrimenti come numero frazionario.

In [ ]:
```
6//3
```

In [ ]:
```
6//4
```

In [ ]:
```
-2//3
```

L'operatore modulo % restituisce il resto della divisione intera. Utile, ad esempio, per vedere se un intero è pari o dispari: n%2 vale 0 se n è pari, 1 se n è dispari.

In [ ]:
```
15%12
```

Ma anche per contare in modo circolare. Ad esempio se vogliamo contare da 0 a 9 in modo circolare, questo è esattamente il calcolo del resto delle divisione per 10

In [ ]:
```
(5+7) % 10
```

### Funzioni matematiche predefinite

Una funzione è una sequenza di istruzioni a cui è associato un nome e degli argomenti di input e che restituisce un valore (o più valori).

E' possibile invocare l'esecuzione della sequenza di comandi definiti dalla funzione usando il suo nome e definendo i valori in input. L'invocazione di una funzione può comparire ovunque è ammissibile un valore del tipo restituito dalla funzione.

Python ha molte funzioni predefinite e (come vedremo più avanti) permette la definizione di nuove funzioni in un qualsiasi programma.

Alcune funzioni matematiche predefinite sono:

Funzione | Restituisce :------: | :-----: abs(x) | valore assoluto di x round(x) | valore di x arrotondato all'intero più vicino ad x round(x,n) | valore di x arrotondato ad n cifre decimali min(x1,..,xn) | valore minore fra quelli presenti come argomenti max(x1,..,xn) | valore maggiore fra quelli presenti come argomenti

Molte più funzioni matematiche sono definite in alcuni moduli, come il modulo *math*. Vedremo nel seguito come si possano utilizzare anche queste altre funzioni.

## Importazione di funzioni (o moduli) esterne

Python dispone di una grandissima libreria di funzioni (o metodi) predefiniti. Molti sono immediatamente utilizzabili, per molti altri invece bisogna prima renderli esplicitamente disponibili al programma. La maggior parte delle funzioni sono definite all'interno di *moduli*, per poterle utilizzare bisogna sapere a quale modulo appartengono. Vediamo oggi per esempio 2 moduli:

1. math: che contiene le funzioni matematiche più importanti
2. random: che contiene funzioni utili per generare e gestire numeri casuali (random)

Ci sono vari modi per rendere disponibile una funzione appartenente ad un modulo, almeno 3 diversi. Vediamo un esempio con il modulo `math` e la funzione seno (`sin()`), che in Python prende in input un angolo espresso in *radianti*):

1. `import math` : *Tutte le funzioni* del modulo math sono ora disponibili. Per utilizzarle bisogna che usiate la notazione `math.sin(3.14152)` che in questo caso restituisce il valore `7.265358972945281e-05` .
2. `from math import *` : *Tutte le funzioni* del modulo math sono ora disponibili. Per utilizzarle non serve più specificare il modulo a cui appartengono, ad esempio basta scrivere `sin(3.14152/2)` che in questo caso restituisce il valore `0.999999999340182` .
3. `from math import sin` : *Solo la funzione* `sin()` del modulo `math` è ora disponibile. Può essere usata come nel caso precedente senza bisogno di specificare il modulo.

In [ ]:
```
# importo solo la funzione sin()
from math import sin
sin(3.14152/2)
```

In [ ]:
```
cos(30) # invoco la funzione coseno, che però non è stata importata
```

In [ ]:
```
# importo tutto il modulo, ma le funzioni devono essere
# invocate anteponendo il nome del modulo
import math
math.cos(3.14152)
```

In [ ]:
```
cos(3.14152) # invoco coseno senza anteporre il nome del modulo (errore)
```

In [ ]:
```
# importo tutte le funzioni del modulo
# in modo da poterle usare senza anteporre il nome del modulo
from math import *
cos(3.14152)
```

In [ ]:
```
#Esempi importazione funzione randint() dal modulo random
#esempio di uso di randint

# si importa solo la funzione randint che può essere chiamata direttamente
from random import randint
randint(0,100) # restituisce un intero a caso compreso fra 0 e 100 (inclusi)
```

In [ ]:
```
randint(0,100) # restituisce un intero a caso compreso fra 0 e 100 (inclusi)
```

In [ ]:
```
# si importa tutto il modulo ma occorre anteporne il nome
import random

random.randint(0,100)
```

In [ ]:
```
#si importano TUTTE le funzioni del modulo random che possono essere chiamat
from random import *
randint(0,100)
```

### Principali funzioni della libreria math

funzione | descrizione :---: | :---: cos(x) | coseno (x deve essere espresso in radianti) sin(x) | seno (come sopra) tan(x) | tangente (come sopra) acos(x) | arco-coseno (x deve essere nell'intervallo [–1, 1])
asin(x) | arco-seno (come sopra) atan(x) | arco-tangente radians(x) | converte in radianti

un angolo espresso in gradi degrees(x) | converte in gradi un angolo espresso in radianti exp(x) | e$^x$ log(x) | ln(x) log(x ,b) | log$_b$(x) log10(x) | log$_{10}$(x) pow(x,y) | $x^y$ sqrt(x) | radice quadrata di x

Tutte le funzioni di questa libreria restituiscono un numero frazionario.

## Stringhe

Un altro tipo di dato fondamentale in Python sono le stringhe. Le stringhe sono sequenze ordinate di caratteri. Esempi di stringhe Python corrette:

- 'Ciao'
- "Ciao a tutti"
- "Sono andato all'interno del negozio"
- 'Il mio amico ha detto "Salve a tutti!" con voce sicura'

In [ ]:
```
'Il mio amico ha detto "Salve a tutti!" con voce sicura'
```

In [ ]:
```

```

## Stringhe non corrette

Esempi di stringhe non corrette sono:

1. 'Ciao
2. Ciao a tutti"
3. "Sono andato all'interno del negozio'
4. 'Il mio amico ha detto "Salve a tutti!' con voce sicura"
5. 'Ho aperto l'acqua'

La 1 è sbagliata perché manca l'apice alla fine. Nella 2 manca all'inizio. Nella 3 abbiamo usato apici diversi all'inizio ed alla fine. Nella 4 abbiamo invertito gli apici a destra in modo che Python pensa che la stringa finisca dopo il '!' e non capisce cosa siano i caratteri successivi. Nella 5 abbiamo usato ' all'interno di una stringa delimitata da apici e così, come nel caso 4, Python pensa che la stringa sia terminata dopo la 'l' e non capisce cosa siano i caratteri successivi.

In [ ]:
```
'Il mio amico ha detto "Salve a tutti!' con voce sicura"
```

## Operazioni su stringhe

In Python è possibile comporre facilmente le stringhe per ottenere stringhe più complesse. Tra le operazioni ammesse tra stringhe abbiamo il `+` ed il `*`. Vediamo come si comportano

In [ ]:
```
'prova'+' '+'del nove'
```

In [ ]:
```
'ciao '*5
```

## Esercizi

Completate questi esercizi a casa prima di cominciare il prossimo argomento

### Esercizio 1:

Scrivete l'istruzione Python che calcola l'espressione $(2^{10})^5$.

In [ ]:

### Esercizio 2:

Scrivete l'istruzione Python che calcola il resto della divisione di $5^5$ per $2^3$.

In [ ]:

### Esercizio 3:

Scrivete l'istruzione Python che calcola il $\sqrt{3^2-4\mathsf{x}2}$

In [ ]:

### Esercizio 4:

Scrivete l'istruzione Python che genera la stringa 'ciao!ciaociao!!' a partire dalle stringhe 'ciao' e '!' ed usando gli operatori `+` e `*`.

In [ ]:

### Esercizio 5:

Scrivete l'istruzione Python che genera la stringa 'ciao!!!ciaociao!!!ciaociaociao!!!' a partire dalle stringhe 'ciao' e '!' ed usando gli operatori `+` e `*`.

In [ ]:

### Esercizio 6:

Scrivete l'istruzione Python che genera la stringa 'ciao! ciao ciao!!ciao ciao ciao!!!' a partire dalle stringhe 'ciao', '!' e ' ' (spazio) ed usando gli operatori `+` e `*`.

In [ ]: