Python

In questa lezione:

- Python - introduzione
- Variabili e tipi
- Gli interi
- I reali
- Le stringhe
- Conversione tra tipi
- Input di una stringa

Il linguaggio Python

E' stato sviluppato agli inizi del 1990 da Guido van Rossum

Egli aveva necessità di scrivere piccoli programmi che non dovevano essere eseguiti a velocità elevata

L'autore voleva che fosse possibile creare velocemente un programma e che potesse essere facilmente aggiornabile

Il nome Python deriva dalla passione di van Rossum per il gruppo comico britannico Monty Python e per la loro serie televisiva Monty Python's Flying Circus.

Lo Zen di Python

Lo sviluppo di Python è condotto principalmente mediante Python Enhancement Proposal (PEP), documenti in cui si propongono, commentano e accettano proposte di miglioramento al linguaggio.

Lo "Zen of Python" (PEP 20) include diciannove aforismi di cui si spiega il senso e quindi la filosofia del linguaggio. Alcuni di questi sono

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Il PEP 8 (https://www.python.org/dev/peps/pep-0008/) è una guida su come scrivere codice in Python.

Versioni del Python

Il Python si è molto evoluto dalle prime versioni degli anni '90.
Una versione molto diffusa è la 2.7, ma questa è stata superata da versioni successive che hanno introdotto significative differenze nella sintassi del linguaggio stesso.
Ai fini del corso utilizzeremo le versioni 3.0 e successive.

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Link utili:

- `https://www.python.org`
- `https://docs.python.org/3/`

Un confronto tra Python e gli altri linguaggi

Il Python è un linguaggio in ascesa, in termini di utilizzo, rispetto a molti altri linguaggi.

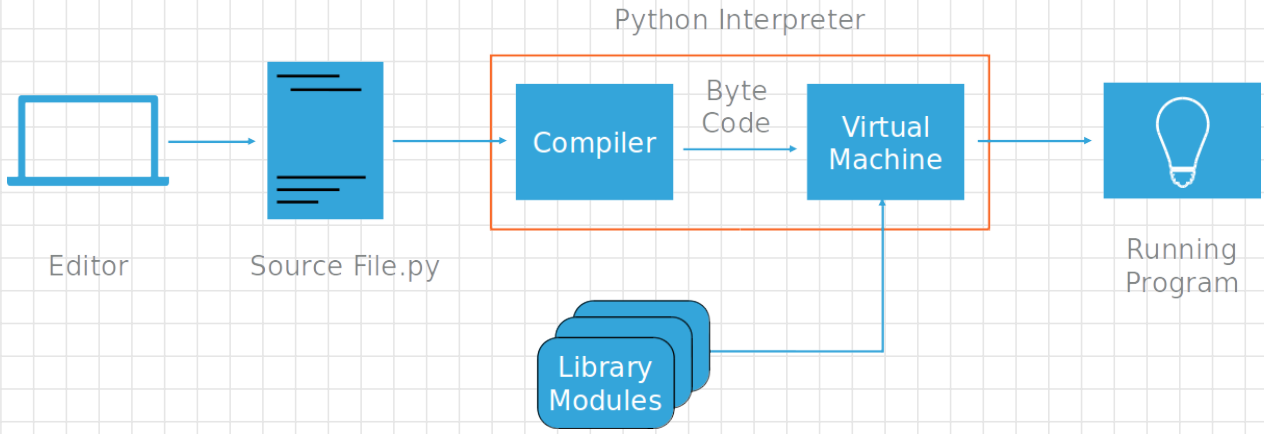
Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
C	1	2	2	1	1	2	1	1
Java	2	1	1	2	3	29	-	-
Python	3	6	6	6	21	15	-	-
C++	4	3	3	3	2	1	2	9
C#	5	4	5	7	9	-	-	-
JavaScript	6	8	8	10	7	-	-	-
PHP	7	7	4	5	19	-	-	-
SQL	8	-	-	-	-	-	-	-
Swift	9	16	-	-	-	-	-	-
R	10	13	49	-	-	-	-	-
Lisp	29	25	15	13	8	5	6	2
Fortran	31	24	24	15	15	17	3	5
Ada	33	27	22	17	17	4	7	3
Pascal	242	15	14	16	16	3	10	6

Interprete Python

Possiamo immaginare che l'interprete Python esegua una istruzione per volta, ma in realtà la fase di lettura e interpretazione di un programma viene eseguita una sola volta da un compilatore. Il codice compilato, (byte code), viene eseguito da un programma detto macchina virtuale che ha un comportamento analogo ad una CPU.



Un primo programma Python

Ecco un primo programma Python:

```
# Il mio primo programma in Python

print("Hello World!")
```

Il programma semplicemente visualizza ("stampa", `print`) una riga di testo: "Hello World!".

La prima linea del programma è un commento.
I commenti iniziano con # e non sono enunciati.

Confrontando questo programma con l'analogo in C++ presentato in una scorsa lezione, si nota subito la sua semplicità e chiarezza.

Variabili

Un programma ha spesso la necessità di memorizzare dei valori, prodotti intermedi di un calcolo, per poi riutilizzarli in seguito.

Per questo fine si usano locazioni di memoria a cui si dà un nome per facilitarne l'accesso.

In Python, questi valori vengono memorizzati all'interno di variabili.

In Python, per definire una variabile bisogna indicare:

- Qual è il nome da usare per riferirsi ad essa
- Qual è il valore iniziale da memorizzare nella variabile

Esempio

```
pezziVenduti = 0        # definizione e assegnazione di un valore
                        # il nome è pezziVenduti, il valore è 0

pezziVenduti = 6        # assegnazione di un nuovo valore


# il segno = è l'operatore di assegnazione
# da non confondere con il segno = usato in algebra

pezziVenduti = pezziVenduti + 2 # l'espressione a destra viene valutata
                                # il valore risultante (8) viene
                                # asseganto alla variabile


print(pezziVenduti)     #stampa 8
```

L'operatore = assegna il risultato dell'espressione (a destra di "=") alla variabile (a sinistra di "="). Quindi prima viene valutata l'espressione e poi il risultato viene memorizzato nella variabile.

Tipi di dato

I valori elaborati dai calcolatori possono essere di tipi diversi. In Python ciascun valore è di uno specifico tipo.

Il tipo di dato specifica:

- un insieme di valori
- definisce l'insieme di operazioni che possono essere compiute con i valori
- determina come un valore è rappresentato e memorizzato all'interno del computer

Un tipo di dato messo a disposizione di un linguaggio è detto tipo di dato primitivo. I programmatori possono però definire tipi di dati detti appunto tipi di dati definiti dall'utente.

Tipi di dato in Python - esempi

Ecco alcuni esempi di tipi di dato primitivi in Python:

Tipi di dato	**Nome in Python**	**Esempi**
Numeri interi	`int`	`-1, 0, 10, 1345, -217`
Numeri reali	`float`	`3.14, -0.55, .3333, 6.0`
Stringhe di caratteri	`str`	`"Ciao", "", 'A', "53"`
Booleani	`bool`	`True, False`

I tipi `int` e `float` si chiamano tipi di dato numerici, poiché rappresentano numeri.

Gli interi

I numeri interi comprendono lo 0, i numeri interi positivi e i numeri interi negativi.

In molti linguaggi gli interi possono essere rappresentati solo con un numero limitato di cifre e il tipo `int` permette di rappresentare solo i numeri interi tra $-2147483648 (-2^{31})$ e $-2147483647 (-2^{31} - 1)$.

In Python i numeri interi possono essere rappresentati con un numero di cifre qualsiasi, limitato solo dalla memoria disponibile.

Operazioni con gli interi

La tabella mostra alcuni operatori aritmetici in Python.

Operatore	Descrizione	Sintassi	Esempi
-	Cambio di segno	-a	-3 → -3
**	Elevamento a potenza	a ** b	3 ** 2 → 9
*	Moltiplicazione	a * b	3 * 2 → 6
/	Divisione	a / b	3 / 2 → 1.5
//	Quoziente	a // b	3 // 2 → 1
%	Resto	a % b	3 % 2 → 1
+	Addizione	a + b	3 + 2 → 5
-	Sottrazione	a - b	3 - 2 → -1

Esempio di espressione: `10 - 7 // 3`. L'espressione vale 8 poiché l'operatore // si applica prima di -.
Valgono le regole di precedenza dell'algebra. Per cambiare le precedenze usiamo le parentesi. Esempio: `(10 - 7) // 3` vale 1.

I reali

I numeri reali sono rappresentati in Python con il tipo `float`.
I valori sono codificati con 64-bit in *doppia precisione* (standard IEEE 754). Hanno una precisione di 16 cifre. Il valore reale assoluto più grande è circa 1.8×10^{308}. Valori più alti vengono rappresentati con `inf`. Valori inferiori a circa 5×10^{-324} sono equivalenti a 0.0.
I numerali sono rappresentati in notazione decimale o notazione scientifica.

```
#precisione limitata
print (0.123456789012345678901) # stampa 0.12345678901234568
print(4.35 * 100)               # stampa 434.99999999999994

#notazione decimale e scientifica
print(3.72e5)    #stampa 372000.0

print(3.72e18)   #stampa 3.72e+18

#valori troppo grandi o troppo piccoli
print(1.8e308)   #stampa inf
print(1e-325)    #stampa 0.0
```

Operazioni con i reali

Tutti gli operatori visti per gli interi (`int`) possono essere applicati ai reali (`float`). In caso di espressioni miste, cioè con valori interi e reali, i primi vengono convertiti in `float` prima di eseguire una operazione.

Esistono anche funzioni che operano sui numeri. Una funzione, in programmazione, è un insieme di istruzioni per svolgere uno specifico compito. Una funzione già vista è `print`.

Funzioni predefinite:

```
y = abs(-10)              # y vale 10

y = round(7.526)          # y è l'intero più vicino, cioè 8

y = round(7.526,2)        # il secondo argomento è il numero di cifre
                          # frazionarie desiderato. y vale 7.53

y = min(5, 2.1, 3, 7.4) # y vale 2.1

y = max(-1.3, 7, 6.9)   # y vale 7 (intero)
```


Funzioni reali

Esistono altre funzioni che possono essere utilizzate importando il modulo `math`. Il modulo include anche la definizione di costanti come π (`math.pi`)

Esempio con π e radice quadrata (`math.sqrt`)

```
from math import pi, sqrt

raggio = 3
circonferenza = 2 * pi * raggio # circonferenza per cerchio di raggio 3

lato = 4
diagonale = lato * sqrt(2)      # diagonale di un qudrato di lato 4
```

Oltre alla radice quadrata `sqrt`, il modulo include le funzioni trigonometriche `cos`, `sin`, `tan`, la funzione esponenziale `exp` e il logaritmo `log`.

Per importarle tutte:

```
from math import *
```

Le stringhe

In Python, una stringa (tipo `str`) è una sequenza di caratteri Unicode racchiusa tra virgolette, singole o doppie.

Il numero di caratteri di una stringa è la sua lunghezza data dalla funzione `len`. Una stringa di lunghezza 0 si dice stringa vuota e si rappresenta con `""` oppure `''`.

Esempio:

```
print('Ciao', "Mondo!" )
poeta = "D'Annunzio"    #notare l'apice interno alla stringa
lunghezza = len(poeta) #lunghezza vale 10
```


Operatori su stringhe

L'operazione più importante tra le stringhe è la concatenazione. Il risultato di una concatenazione è dato da una stringa costituita da tutti i caratteri della prima seguiti da quelli della seconda. In Python si usa l'operatore + per concatenare due stringhe.

```
nome = "Gabriele"
cognome = "D'Annunzio"
poeta = nome + " " + cognome
print(poeta)  # stampa: Gabriele D'Annunzio
```

Un'altra operazione è la ripetizione per concatenare più volte la stessa stringa. In Python si usa l'operatore * per la ripetizione.

```
trattino = "-"
linea = trattino * 20
```

Metodi per stringhe

In Python le stringhe sono oggetti, entità software che hanno un valore e un comportamento. Il comportamento di un oggetto è definito dai suoi metodi. I metodi, come le funzioni, sono sequenze di istruzioni, ma possono essere applicate solo ad un oggetto del tipo per cui è stato definito.

```
filosofo = "Bertrand Russell"

maiuscolo = filosofo.upper() # assegna "BERTRAND RUSSELL" a maiuscolo

print(filosofo.lower())      # stampa: bertrand russell

figlio = filosofo.replace("Bertrand", "Conrad") # la stringa figlio
                                                # vale "Conrad Russell"
```

Suggerimento: se si deve operare su una stringa, probabilmente l'operazione è già svolta da uno dei sui metodi!

Accesso ai caratteri di una stringa

È possibile accedere ai caratteri di una stringa mediante la loro posizione all'interno della stringa. Tale posizione è detta indice del carattere.

Per esempio, la stringa "SALVE" ha cinque caratteri di indice 0, 1, 2, 3 e 4.

S	A	L	V	E
0	1	2	3	4

Possiamo accedere ad un carattere di una stringa mediante l'operatore [].

```
saluto = "SALVE"

print(saluto[1] + saluto[3] + saluto[4]) #stampa: AVE

ultimo = len(saluto) - 1     #indice dell'ultimo carattere

print(saluto[ultimo])        #stampa: E
```

Conversioni tra tipi

È spesso necessario convertire un valore di tipo in un valore di un altro tipo. Es.:

```
targa = "DE" + 295 + "WK" # Errore: 295 non è una stringa
```

A questo fine esistono le funzioni di conversione, tra cui le seguenti.

Funzione	Esempio	Risultato
`int`	`int(3.14)`	`3`
	`int("12")`	`12`
`float`	`float(22)`	`22.0`
	`float("3.14")`	`3.14`
`str`	`str(3.14)`	`"3.14"`
	`str(12)`	`"12"`
`chr`	`chr(65)`	`'A'`
`ord`	`ord('A')`	`65`

Input di una stringa

Per chiedere dati all'utente di un programma è bene fornirgli prima un messaggio che spieghi quali dati sono attesi. Questo messaggio si chiama prompt (suggerimento).

In Python la visualizzazione del prompt e la lettura dei dati forniti mediante tastiera sono effettuati dalla funzione `input`.

La funzione `input restituisce una stringa` contenente la sequenza dei caratteri battuti dall'utente.

```
nome = input("Inserisci il tuo nome: ")

cognome = input("Inserisci il tuo cognome: ")

print("Benvenuto " + nome + " " + cognome)
```

Input di un valore

La funzione `input` restituisce sempre una stringa. Se i dati in ingresso sono di tipo diverso dalla stringa è necessaria una conversione di tipo.

```
anno_str = input("Inserisci il tuo anno di nascita: ")

anno = int(anno_str)    # anno_str è una stringa
                        # anno è un intero


#conversione nello stesso enunciato

altezza = float( input("Altezza in metri: ") )
```