Programming Lab

Lezione 2

Intro a Python

Tipi dati, operatori, costrutti, funzioni.

Laura Nenzi

Cos’è Python?

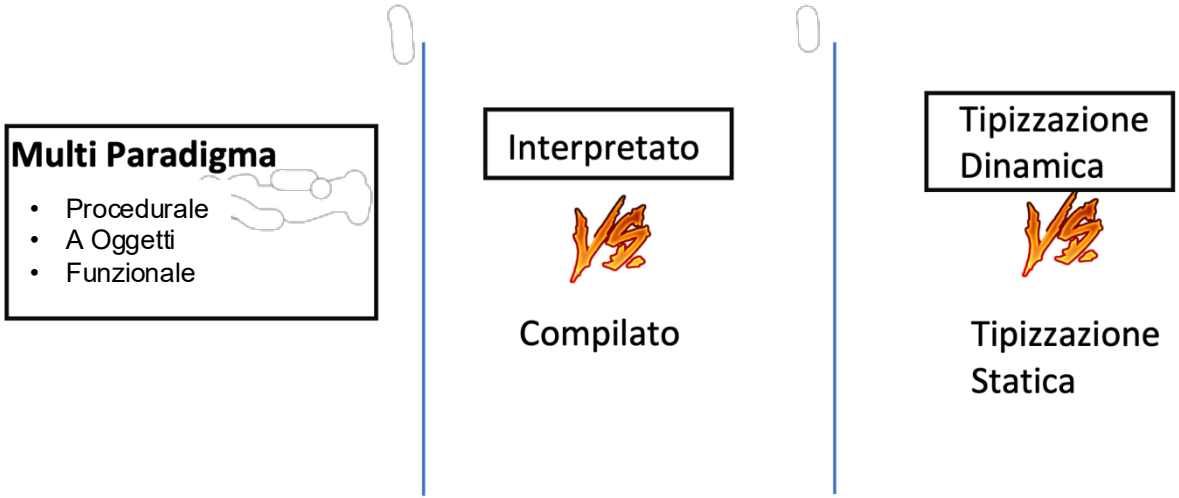
«Python è un linguaggio di programmazione ad alto livello, orientato agli oggetti, adatto, tra gli altri usi, a sviluppare applicazioni distribuite, scripting, computazione numerica e system testing» (wikipedia)

«È un linguaggio multi-paradigma che ha tra i principali obiettivi: dinamicità, semplicità e flessibilità. Supporta il paradigma object oriented, la programmazione strutturata e molte caratteristiche di programmazione funzionale ...» (wikipedia)

«Python è un linguaggio di programmazione interpretato, interattivo, orientato agli oggetti. Include moduli, eccezioni, tipizzazione dinamica...» (python.it)

Cos’è Python?

Linguaggio di programmazione ad alto livello



Basso livello → Alto Livello

```
00100111101111011111111111100000
10101111101111110000000000010100
10101111101001000000000000100000
10101111101001010000000000100100
10101111101000000000000000011000
10101111101000000000000000011100
10001111101011100000000000011100
10001111101110000000000000011000
00000001110011100000000000011001
00100101110010000000000000000001
00101001000000010000000001100101
10101111101010000000000000011100
00000000000000000111100000010010
00000011000011111100100000100001
00010100001000001111111111110111
10101111101110010000000000011000
00111100000001000001000000000000
10001111101001010000000000011000
00001100000100000000000001101100
00100100100001000000010000110000
10001111101111110000000000010100
00100111101111010000000000100000
00000011111000000000000000001000
00000000000000000001000000100001
```

Codice Macchina a 32 bit

```
        .text
        .align    2
        .globl    main
main:
        subu      $sp, $sp, 32
        sw        $ra, 20($sp)
        sd        $a0, 32($sp)
        sw        $0,  24($sp)
        sw        $0,  28($sp)
loop:
        lw        $t6, 28($sp)
        mul       $t7, $t6, $t6
        lw        $t8, 24($sp)
        addu      $t9, $t8, $t7
        sw        $t9, 24($sp)
        addu      $t0, $t6, 1
        sw        $t0, 28($sp)
        ble       $t0, 100, loop
        la        $a0, str
        lw        $a1, 24($sp)
        jal       printf
        move      $v0, $0
        lw        $ra, 20($sp)
        addu      $sp, $sp, 32
        jr        $ra

        .data
        .align    0
str:
        .asciiz   "The sum from 0 .. 100 is %d\n"
```

Codice Assembly

```
a = 5
b = 10
c= a+b
print("la somma è "+str(c))
```

Python

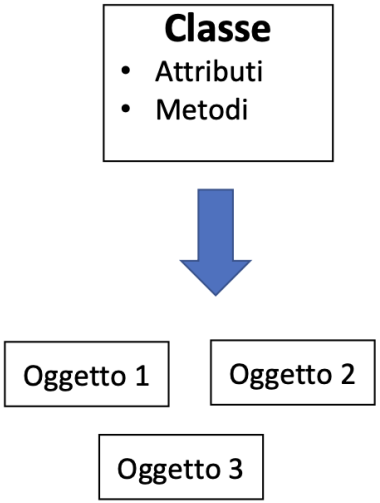
Paradigmi di programmazione

Linguaggio Procedurale

Paradigma di programmazione, dove il flusso di esecuzione è una sequenza di istruzioni e chiamate a procedure che modificano lo stato del programma.

- Output (Y) è determinato esclusivamente dall’ input (X)

Linguaggio orientato ad oggetti



Libro

Attributi (dati)
- Titolo
- Autore
- N pag.

Metodi
- printInfo()
- getNumeroPag()

Oggetto (23234)
Titolo: I Promessi Sposi
Autore: Alessandro Manzoni
N pag: 500

(23234).printInfo()
>>> Alessandro Manzoni, «I Promessi Sposi», numero di pagine 500.
(23234).getNumeroPag()
>>> 500

Linguaggio funzionale

Paradigma di programmazione, dove il flusso di esecuzione assomiglia a una serie di valutazioni di funzioni matematiche

- Output (Y) è determinato esclusivamente dall’ input (X)

Linguaggio multi-paradigma

Procedurale

```
def main():
    x = 5
    x = x * 2
    x = x + 3
    print(x)

main()
```

Ad Oggetti

```
class Calcolatore:
    def __init__(self, valore):
        self.valore = valore

    def raddoppia(self):
        self.valore *= 2

    def aggiungi_tre(self):
        self.valore += 3

    def stampa(self):
        print(self.valore)

c = Calcolatore(5)
c.raddoppia()
c.aggiungi_tre()
c.stampa()
```

Funzionale

```
def raddoppia(x):
    return x * 2

def aggiungi_tre(x):
    return x + 3

print(aggiungi_tre(raddoppia(5)))
```

- Nota bene: in un programma posso usare contemporaneamente diversi paradigmi

Interpretato

Compilato

Python: un linguaggio interpretato

Python è un linguaggio interpretato, non deve essere tradotto in linguaggio macchine come per il c (ovvero, compilato), ma viene eseguito “come sta”

Essendo un linguaggio interpretato, ha anche un interprete interattivo, che potete (e dovreste) usare ogni qualvolta vogliate testare delle cose in rapidità

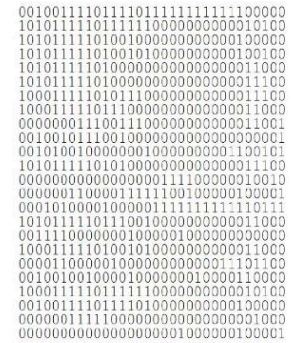
```
ste@Stes-MacAir:ProgrammingLab (master) $ python
Python 2.7.10 (default, Oct 23 2015, 19:19:21)
[GCC 4.2.1 Compatible Apple LLVM 7.0.0 (clang-700.0.59.5)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 2+2
4
>>> for item in [1,2,3]: print item
...
1
2
3
>>>
```

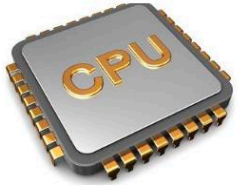
Linguaggio Compilato

```c
#include <stdio.h>

int main(void){
  printf("hello, world\n");
}
```

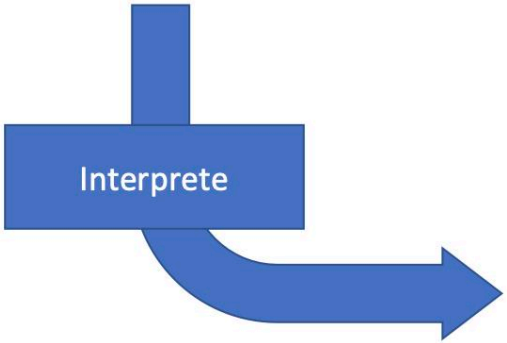
Compilatore

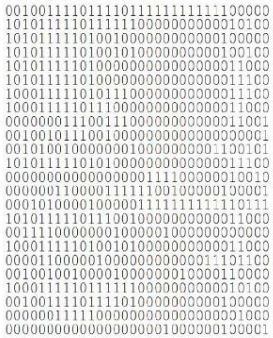


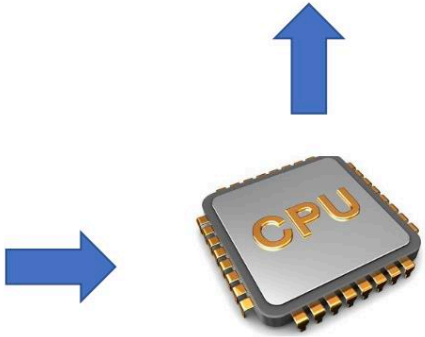


Linguaggio Interpretato

```
>>> print("Hello world!")
```

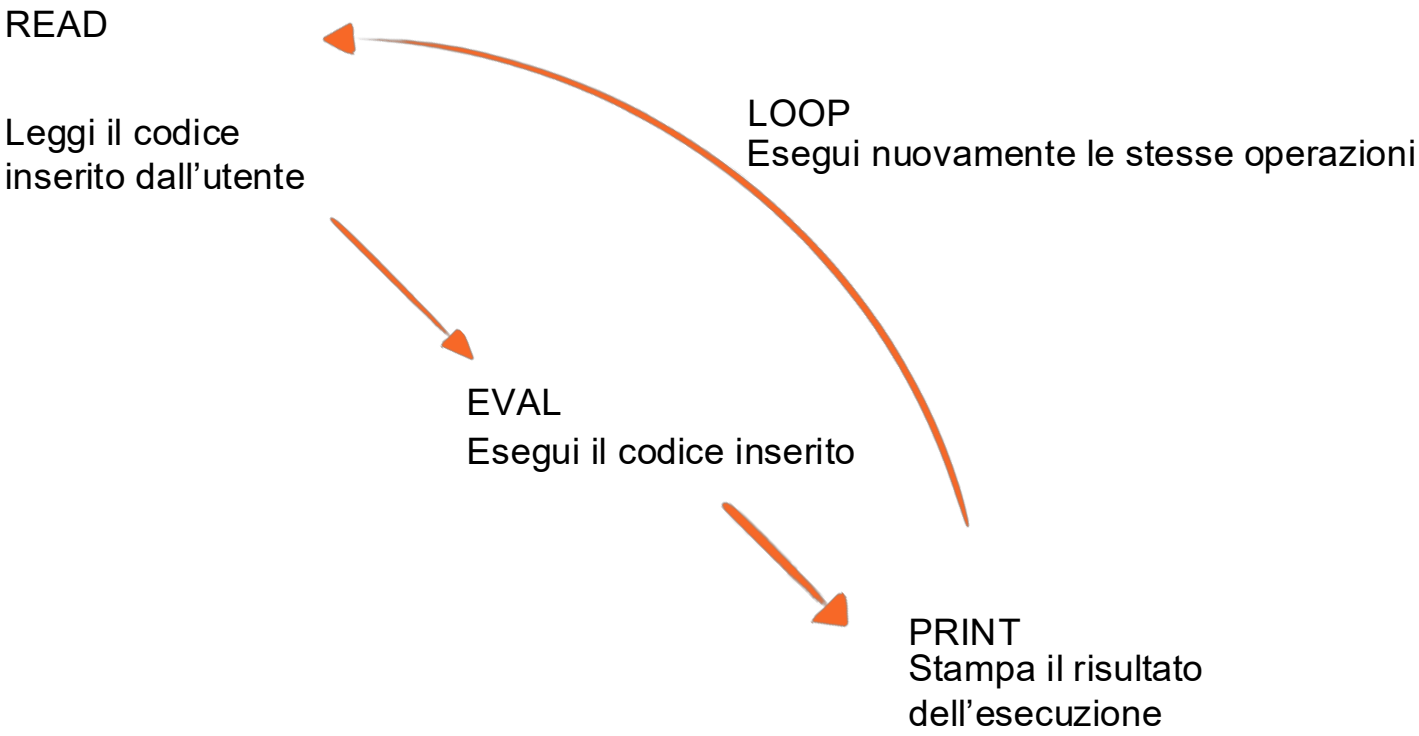







```
>>> print("Hello world!")
Hello world!
```

READ-EVAL-PRINT-LOOP (REPL)



Tipizzazione

Tipizzazione dinamica

Python non richiede di definire esplicitamente i tipi dati. Il tipo di una variabile viene determinato in fase di esecuzione. Aumenta la produttività del programmatore ma richiede cautela per prevenire errori di tipo durante l'esecuzione

```
my_var = 1        # Esempio di variabile tipo intero
my_var = 1.1      # Esempio di variabile tipo floating point
my_var = 'ciao'   # Esempio di variabile tipo stringa
my_var = “ciao”   # Esempio di variabile tipo stringa
my_var = True     # Esempio di variabile tipo booleano
my_var = None     # Il “niente” si rappresenta con il “None”
```

Python vs C

Altre importanti differenze da tenere presente. In Python:

- La memoria viene gestita automaticamente
- L’indentazione *è sintassi .* Usa l’indentazione significativa invece che grafe e punti i virgola
- Ha una libreria standard enorme

Benevolent dictator for «life»

Guido Van Rossum

- Olandese
- Crea Python nel 1989

Perché creare Python?

Per avere un linguaggio

- semplice, intuitivo e potente
- open source, aperto allo sviluppo condiviso
- facilmente comprensibile, come l’inglese parlato
- ottimo per risolvere problemi quotidiani (degli sviluppatori)

Voleva essere un linguaggio di rottura con il passato,
lievemente provocatorio

Perché chiamarlo Python?

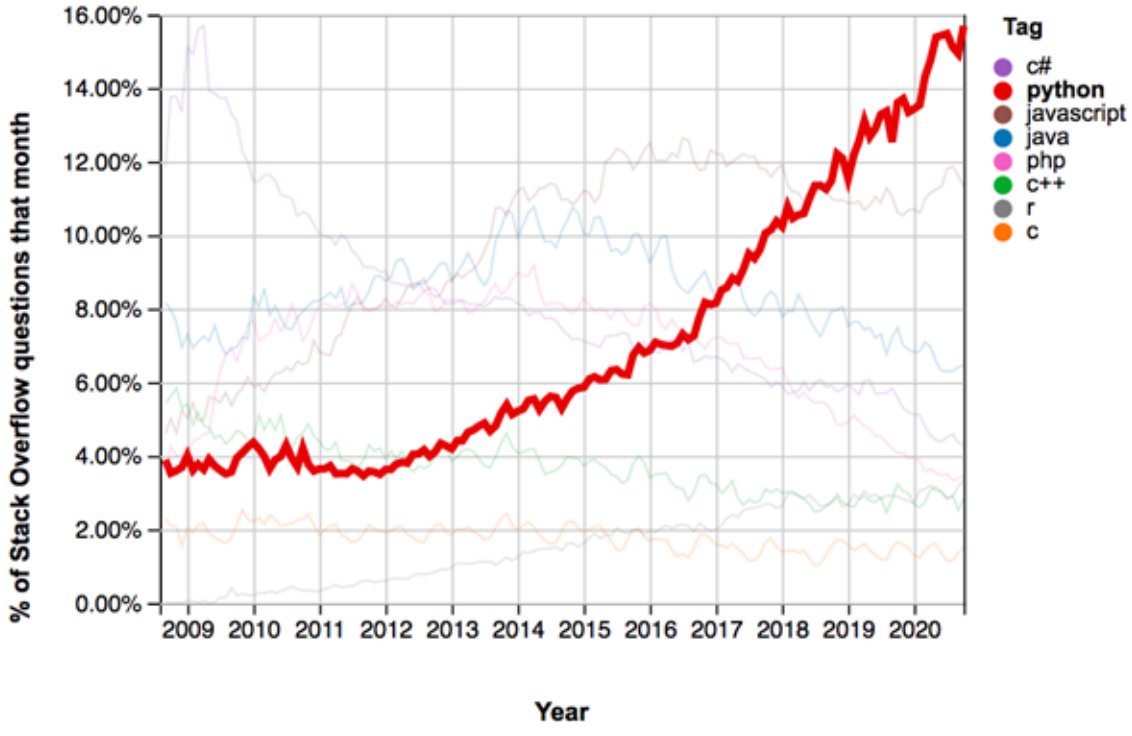
Guido V.R prende ispirazione da un gruppo comico inglese i Monty Python
«Il programma televisivo fu una rivoluzione — i Monty Python hanno rappresentato quello che i Beatles sono stati per la musica: un punto di partenza e di non ritorno. Il loro umorismo — anarchico, a tratti demenziale, sempre intellettuale, mai volgare, corrosivo, capace di toccare le vette rarefatte dell'assurdo — sfidò tutte le convenzioni comiche dell'epoca, sia in termini di stile che di contenuti»
Corriere della Sera, 22/10/20

Perché chiamarlo Python?

Guido V.R prende ispirazione da un gruppo comico inglese i Monty Python
«Il programma televisivo fu una rivoluzione — i Monty Python hanno rappresentato quello che i Beatles sono stati per la musica: un punto di partenza e di non ritorno. Il loro umorismo — anarchico, a tratti demenziale, sempre intellettuale, mai volgare, corrosivo, capace di toccare le vette rarefatte dell'assurdo — sfidò tutte le convenzioni comiche dell'epoca, sia in termini di stile che di contenuti» Corriere della Sera, 22/10/20

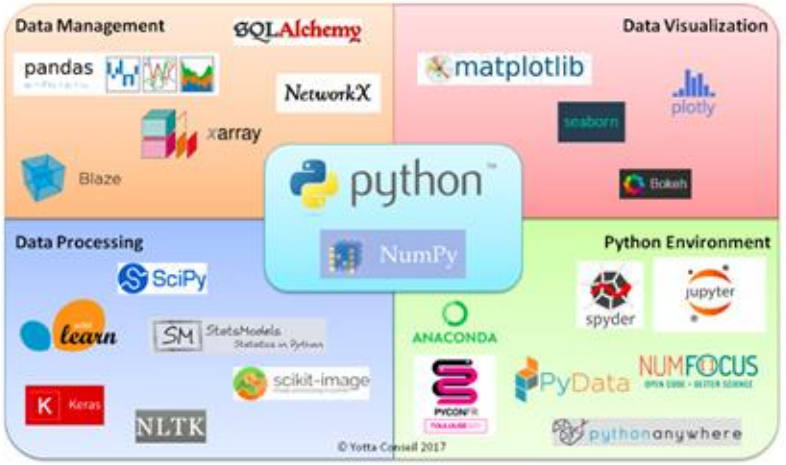
https://www.youtube.com/watch?v=ycKNt0MhTkk

Perchè Python?



Perchè Python?

- E’ un linguaggio semplice, intuitivo e potente
- Facilmente comprensibile, quasi pseudo-codice
- E’ il linguaggio della Data Science
- Ha un ecosistema di software per il calcolo scientifico / statistico invidiabile



The Python data science ecosystem (source: Yotta Conseil)

Una premessa

In questo corso di laboratorio viene assunto che siate già un minimo familiari con i costrutti e gli operatori di base di un linguaggio di programmazione. Per esempio:

- Assegnazione di variabili e stampa a schermo (`=`, `print`)
- Operatori artimetici (`+`, `-`, `*`, etc.)
- Operatori logici (`and`, `or`)
- Operatori di confronto (`==` , `<`, `>`, etc.)
- Operatori condizionali (`if`, `else`)
- Cicli (`for`, `while`, etc.)

Hello world!

Tipico esempio di “ciao mondo” che si usa come esempio in programmazione. Posso usare singole o doppie virgolette, più standard doppie(PEP 8, https://peps.python.org/pep-0008/.

```
print('Hello world!')
```

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

```
print('ciao','gigi', sep = ' [separatore] ', end =' [questo lo metto alla fine] ')" )
```

f-string

L’istruzione **print** può essere usata assieme alla formattazione delle stringhe per includere i valori delle variabili che vogliamo stampare a schermo:

```
print(f"Valore 1: {my_var_1}, valore 2: {my_var_2}")
```

..che vuol dire che Python formatterà la stringa da stampare andando a sostituire alle doppie graffe prima la variabile “`my_var_1`”, poi la variabile “`my_var_1`”.

Le doppie graffe rappresentano l’ancora.

Vecchio formato:

```
print('Valore 1: {}, valore 2: {}'.format(my_var_1, my_var_2))
```

Commenti

Con i cancelletti si inseriscono i commenti nel codice.
Commentate il più possibile quello che fate!

```
x = 4 # Commento su una singola linea
```

```
# Commento su linee multiple.
# Anche questo viene ignorato
# Fino a qui
```

```
“””
Commento su linee multiple.
Anche questo viene ignorato se usiamo
dei tre volte dei doppi apici
Fino a qui.
“””
```

Tipi di dato

Python non richiede di definire esplicitamente i tipi di dato, ovvero una variabile può cambiare contenuto e non deve esserne definito il tipo.
Tipi principali:

```
my_var = 1                    # tipo intero
my_var = 1.1                  # tipo floating point
my_var = True                 # tipo booleano
my_var = 'ciao'               # tipo stringa (sequenza di caratteri)
my_var = "ciao"               # tipo stringa (sequenza di caratteri)
my_var = [1, 2, 3]            # lista
my_var = (1, 2, 3)            # tupla
my_var = {1, 2, 3}            # insieme
my_var = {"a": 1, "b": 2}.    # dizionario
my_var = None                 # Il “niente” si rappresenta con il "None"
```

Tipi di dati numerici

- **int**: numeri interi (positivi e negativi)
- **float**: numeri a virgola mobile, *solitamente* a precisione doppia (64 bit)
- **complex**: numeri complessi
- Questi sono solo i tipi più comuni, ne esistono anche altri utilizzabili (e.g., per rappresentare frazioni o per usare una differente precisione)

- funzione type ritorna il tipo: print(type(2), type(42.0), type("Hello"))

Operatori aritmetici

Operatore	Nome	Esempio
+	Addizione	x+y
-	Sottrazione	x-y
*	Moltiplicazione	x*y
/	Divisione	x/y
%	Modulo [a]	x%y
**	Esponenziale	x**y

Operatori di confronto

Gli operatori di confronto ritornano True o False a seconda che la condizione indicata sia rispettata o no. Sono usati soprattutto nei costrutti if.

Operatore	Nome	Esempio
==	Uguale	x==y
!=	Diverso	x!=y
>	Maggiore	x>y
<	Minore	x<y
>=	Maggiore o uguale	x>=y
<=	Minore o uguale	x<=y

Booleani

- È un tipo di dato con solo due valori possibili:
- **True**, per indicare che qualcosa è vero
- **False**, per indicare che qualcosa è falso
- Vengono utilizzati per verificare condizioni
- Sono il risultato di comparazioni tra valori di altri tipi: gli operatori di confronto restituiscono dei Booleani

Esempio:

```
x = 5>6 # È una variabile booleana
```

Operatori Booleani

- Possiamo combinare i valori Booleani con le operazioni di **and**, **or** e **not**

A	not A
False	True
True	False

A	B	A and B	A or B
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

Operatore	Descrizione	Esempio
`and`	`True se entrambe True`	`x<5 and x<10`
`or`	`True se almeno una True`	`x<5 or x<4`
`not`	`Inverte il risultato`	`not(x<5 and x<10)`

Stringhe

- Una stringa, cioè una sequenza di caratteri, è un tipo in python
- Quando la scriviamo, una stringa è una sequenza di testo racchiusa da apici singoli ' ' o doppi " "
- Notare la differenza:
    - 3 è il numero 3.
    - "3" è la sequenza di caratteri che, stampata a schermo, fa apparire il carattere "3".

Operazioni sulle Stringhe

- **"ciao" + "mondo"** produce la stringa “ciaomondo”. Se volessimo uno spazio dovremmo inserirlo:
  "ciao" + " mondo" produce “ciao mondo”
- + tra stringhe è **concatenazione**
  + tra numeri è **addizione**.
  Non sono intercambiabili
- Alcuni simboli nelle stringhe sono interpretati in modo diverso: “\n” rappresenta il ritorno a capo. Provate a stampare la stringa "ciao\nmondo"

Accedere alle stringhe per posizione

Si può accedere all'elemento i-esimo di una stringa così:

```
mia_stringa[2]   # Terzo carattere della stringa
mia_stringa[-1]  # Ultimo carattere della stringa
```

Lo “slicing” delle stringhe

Si può tagliare una fetta (“to *slice*”) di una stringa così:

```
mia_stringa[0:50]   # Dal primo al cinquantesimo carattere
mia_stringa[30:50]  # Dal trentunesimo al cinquantesimo carattere
mia_stringa[0:-1]   # Dal primo al penultimo carattere
mia_stringa[1:3:2]) # Dal secondo al quarto con intervallo di 2
mia_stringa[-1::-1]) # Dall’ultimo carattere al primo con intervallo di -1
```

Conversione tra tipi (CASTING)

- Come facciamo a passare dalla stringa che contiene “3” al numero “3”? Ci sono funzione per passare tra tipi diversi
- **int(“3”)** restituisce il numero 3
- **float(“3.14”)** restituisce il numero 3.14
- **str(4)** restituisce la stringa “4”
- Se una conversione non è possibile viene generata una eccezione (simile ad un “errore”)

Input dell’utente

In Python esiste una funzione predefinita chiamata `input` che sospende il programma ed attende che l’utente scriva qualcosa

```
>>> testo = input()
Cosa stai aspettando
>>> testo
'Cosa stai aspettando'
```

Prima dell’inserimento dei dati, è buona norma visualizzare un messaggio, chiamato prompt, che informa l’utente di ciò che deve inserire.

```
>>> nome = input('Come...ti chiami?\n')
Come...ti chiami?
Artù, Re dei Bretoni!
>>> nome
'Artù, Re dei Bretoni!'
```

Istruzioni condizionali, blocchi, indentazione

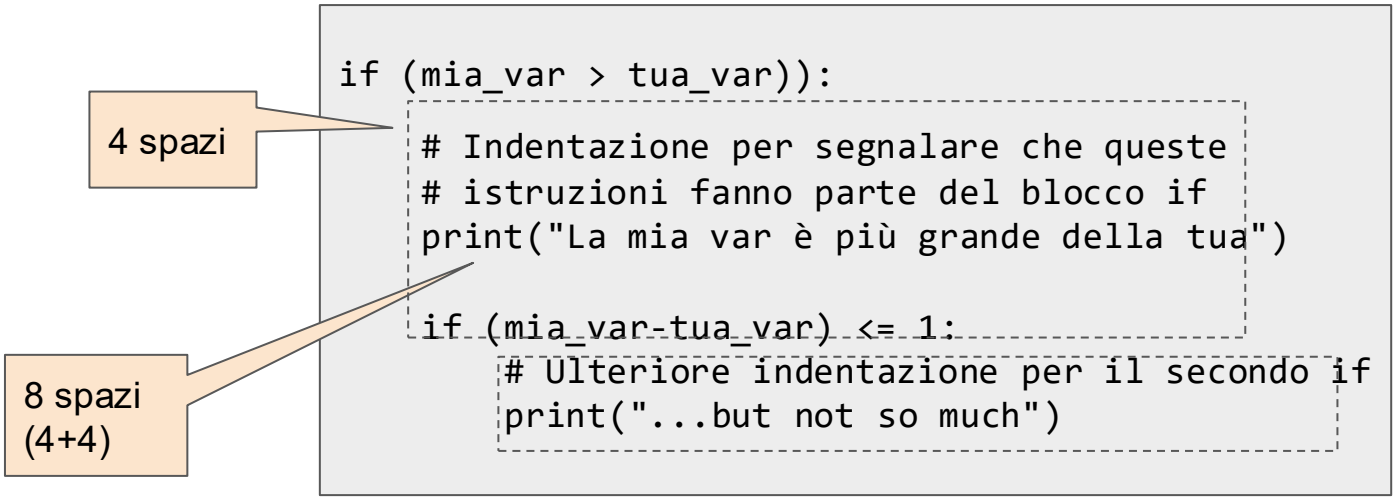
In Python valgono i soliti if-else, che ci introducono ai **blocchi** e l'**indentazione**:

```
if (my_var > your_var):
    print("My var is bigger than yours")

    if (my_var-your_var) <= 1:
        print("...but not so much")
```

Istruzioni condizionali, blocchi, indentazione

In Python valgono i soliti if-else, che ci introducono ai **blocchi** e l'**indentazione**:



Istruzioni condizionali, blocchi, indentazione

In Python condizioni aggiuntive si aggiungo con l’ “elif”

```
if (mia_var > tua_var):
    print("La mia var è più grande della tua")
    if (mia_var-tua_var) <= 1:
        print("...Non così tanto...")
    elif (mia_var-tua_var) <= 5:
        print("...Abbastanza")
    else:
        # L'ultima condizione ha bisogno solo dell'else, che
        # gestisce tutti i casi che non sono stati considerati
        print("...Di molto")
```

I cicli: while

In Python valgono i soliti cicli for e while ma come visto nell’esempio di prima, alcune cose come ciclare sugli elementi sono più facili.

```
i=0
while i<10:
    print(i)
    i = i+1
```

I cicli: for

Nel ciclo for, l'operatore in indica che si vuole iterare sugli elementi di un oggetto *iterabile* (lista, stringa, tupla, set, range, …).
Si legge come: "per ogni elemento contenuto in …"

```
for item in mylist:
    print(item)
```

```
for i in range(10):
    print(i)
```

Range(inizio,fine) ci permette di generare una sequenza di numeri nell'intervallo [inizio, fine) a distanza 1 l'uno dall'altro

I cicli: for

In Python valgono i soliti cicli for e while ma come visto nell'esempio di prima, alcune cose come ciclare sugli elementi sono più facili. Esempi:

```
for i, item in enumerate(mylist):
    print("Posizione {}: {}".format(i, item))
```

Le funzioni

Una funzione si definisce con:

```
def mia_funzione(argomento1, argomento2):
    print(f"Argomenti: {argomento1} e {argomento2}")
```

e si chiama con:

```
mia_funzione("Pippo","Pluto")
```

...che stamperà a schermo:

```
Argomenti: Pippo e Pluto
```

Le funzioni

Una funzione può anche (e in genere deve) tornare un valore:

```
def eleva_al_quadrato(numero):
    return numero*numero
```

e si chiama con:

```
eleva_al_quadrato(4)
```

...che tornera' il valore 16, che può essere poi assegnato a una variabile e stampato a schermo:

```
risultato = eleva_al_quadrato(4)
print(f"Risultato: {risultato}")
```

Le funzioni

Una funzione può anche avere dei valori di default per gli argomenti (sono quindi dei parametri)

```
def eleva_alla_n(numero, n=2):
    return numero**n
```

e si chiama con:

```
eleva_alla_n(4,n=3)
```

oppure:

```
eleva_alla_n(4,3)
```

Se nella chiamata non do un valore al secondo argomento lui prende quello di default:

```
eleva_alla_n(4)
```

La docstring delle funzioni

Una docstring è una stringa posizionata immediatamente sotto la definizione della funzione, racchiusa tra triple virgolette (""" ... """)

```
def somma(a, b):
    """
    Calcola la somma di due numeri.

    Args:
        a (int or float): Il primo numero.
        b (int or float): Il secondo numero.

    Returns:
        int or float: La somma dei due numeri.
    """
    return a + b
```

L'help

La funzione help mi descrive classi, funzioni, moduli

```
help('stringa'.isalpha)
```

```
Help on built-in function isalpha:

isalpha() method of builtins.str instance
    Return True if the string is an alphabetic string, False otherwise.

    A string is alphabetic if all characters in the string are alphabetic and there
    is at least one character in the string.
```

L'help

Descrive anche le funzioni che abbiamo definito noi riportando il docstring

```
help(somma)
```

```
Help on function somma in module __main__:

somma(a, b)
    Calcola la somma di due numeri.

    Args:
        a (int or float): Il primo numero.
        b (int or float): Il secondo numero.

    Returns:
        int or float: La somma dei due numeri.
```

Le funzioni built-in

Sono funzioni sempre disponibili, un paio le abbiamo già viste:

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Python: lo “scope”

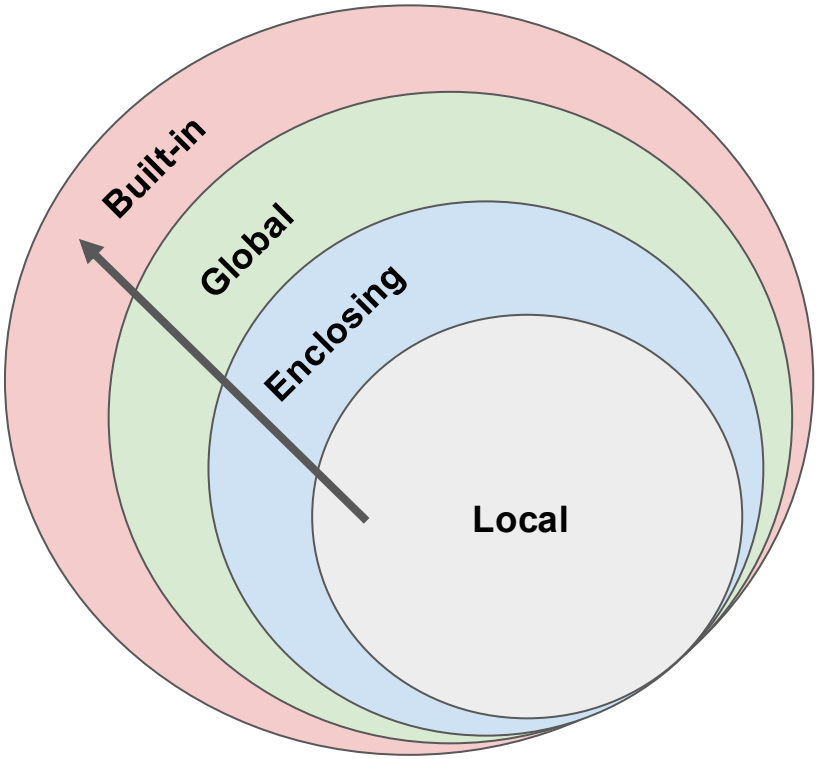
Come viene risolta (trovata) una variabile?

Python segue la regola LEGB:

- Local

  Le cose definite “dove sono”, ad esempio dentro una funzione
- Enclosing

  Ad esempio la funzione esterna se metto due funzioni una dentro l’altra, o il “corpo” esterno del programma
- Global
- Built-in



Come si scrive una buona funzione (1)

Una buona funzione agisce solo su variabili locali

Ovvero, qualsiasi cosa tratto deve “entrare” nella funzione con un argomento.

NO!

```
numero = 5

def eleva_al_quadrato():
    risultato = numero*numero
    return risultato
```

SI :)

```
def eleva_al_quadrato(numero):
    risultato = numero*numero
    return risultato
```

Come si scrive una buona funzione (2)

Una buona funzione torna sempre il suo risultato con un return()

Ovvero, non vado a modificare l'elemento che ho passato con gli argomenti!

NO!

```
risultati = []

def eleva_al_quadrato(numero, risultato):
    risultati.append(numero*numero)
```

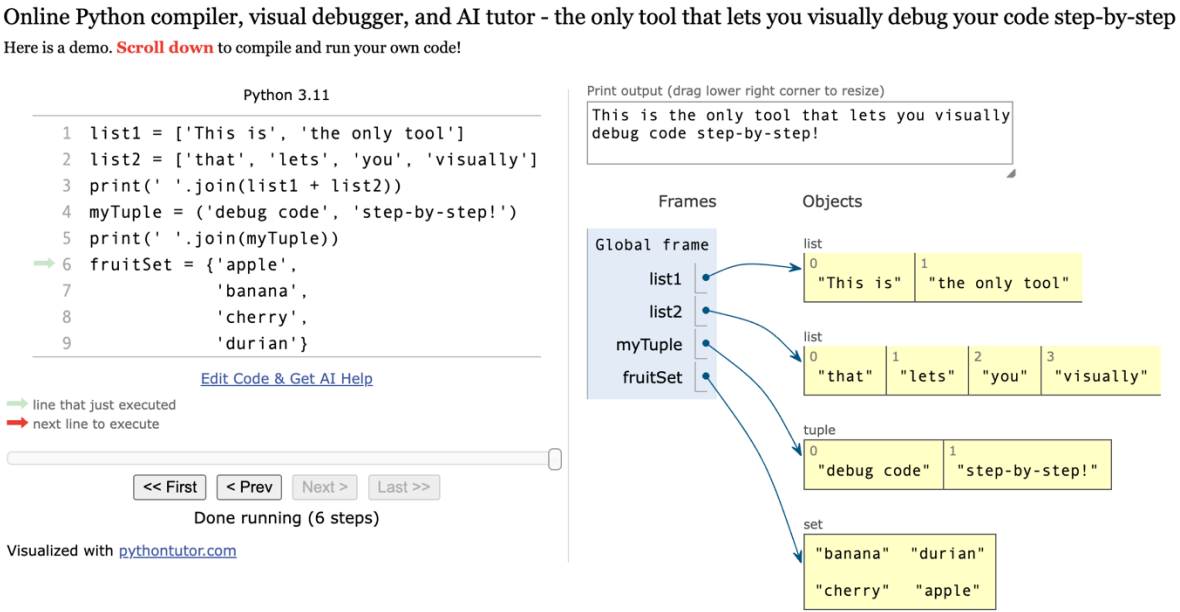
SI :)

```
risultati = []
def eleva_al_quadrato(numero):
    risultato = numero*numero
    return risultato

risultati.append(eleva_al_quadrato(...))
```

Online Python compiler

https://pythontutor.com/python-compiler.html - mode=edit



Lista Esercizi

1. Stampare l’equivalente di 538 minuti nel formato 8h:58min.
2. Scrivere un programma che chiede all’utente un numero intero e stampa il suo quadrato e il suo cubo.
3. Scrivere un programma che verifica se un numero inserito dall’utente è pari o dispari.
4. Definire una funzione che prende come argomento una parola e una lettera e ritorna quante volte quella lettera è contenuta nella parola.
5. Scrivere un programma che verifica se un numero inserito dall’utente è primo.
6. Scrivere un programma che fa la somma di n numeri inseriti dall’utente. Di all'utente di scrivere 0 per fermarsi.
7. Definire la funzione fattoriale (versione iterativa).
8. Definire una funzione che dati 3 numeri interi stabilisce se possono essere i valori dei lati di un triangolo e, se sì, di che tipo di triangolo.
9. Definire una funzione che conta quante vocali sono presenti in una stringa.