Programming Lab

Parte 2

Python: tipi dati, costrutti,

funzioni, moduli, be pythonic.

Laura Nenzi

Many slides by Stefano Russo

Python: 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 condizionali (`if`, `else`)
- Operatori artimetici (`+`, `-`, `*`, etc.)
- Operatori logici (`and`, `or`)
- Operatori di confronto (`==` , `<`, `>`, etc.)
- Cicli (`for`, `while`, etc.)

Python: hello world!

Tipico esempio di “ciao mondo” che si usa come esempio in programmazione.

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

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

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

Python: .format()

Le doppie graffe rappresentano l’ancora.

Può essere comodo associare una chiave all'ancora { } scrivendo al suo interno una chiave univoca, {A} è un ancora con chiave A.

```
print('Valore 1: {A}, valore 2: {B}'.format(B=36, A=5))
```

Python: tipi dati

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

Ecco i tipi principali:

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

```
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: 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]  # Penultimo carattere della stringa
```

Python: 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
```

Python: operatori di confronto

Operator	Name	Example
==	Equal	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Python: operatori aritmetici

Operator	Name	Example
+	Addition	x + y
-	Subtraction	x - y
*	Multiplication	x * y
/	Division	x / y
%	Modulus	x % y
**	Exponentiation	x ** y

Python: operatori logici

Operator	Description	Example
and	Returns True if both statements are true	x < 5 and x < 10
or	Returns True if one of the statements is true	x < 5 or x < 4
not	Reverse the result, returns False if the result is true	not(x < 5 and x < 10)

Python: istruzioni condizionali, blocchi, indentazione

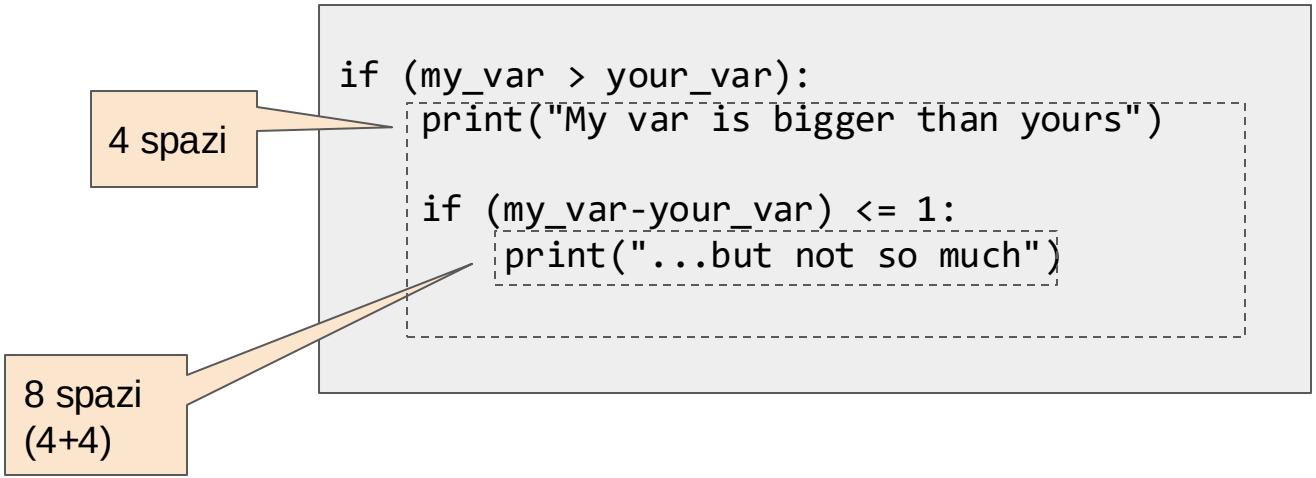
In Python valgono i soliti if-else, che ci introducono ai blocchi e l'indentazione:

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

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

Python: istruzioni condizionali, blocchi, indentazione

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



Python: istruzioni condizionali, blocchi, indentazione

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

```
if (my_var > your_var):

    print("My var is bigger than yours")

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

    elif (my_var-your_var) <= 5:
        print("...quite a bit")

    else:
        print("...a lot")
```

Python: i cicli

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 item in mylist:
    print(item)
```

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

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

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

Python: le funzioni

Una funzione si definisce con:

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

e si chiama con:

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

...che stamperà a schermo:

```
Argomenti: Pippo e Pluto
```

Python: 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('Risultato: {}'.format(risultato))
```

Python: 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)
```

Python: 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
```

Python: 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.
```

Python: 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.
```

Python: 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”

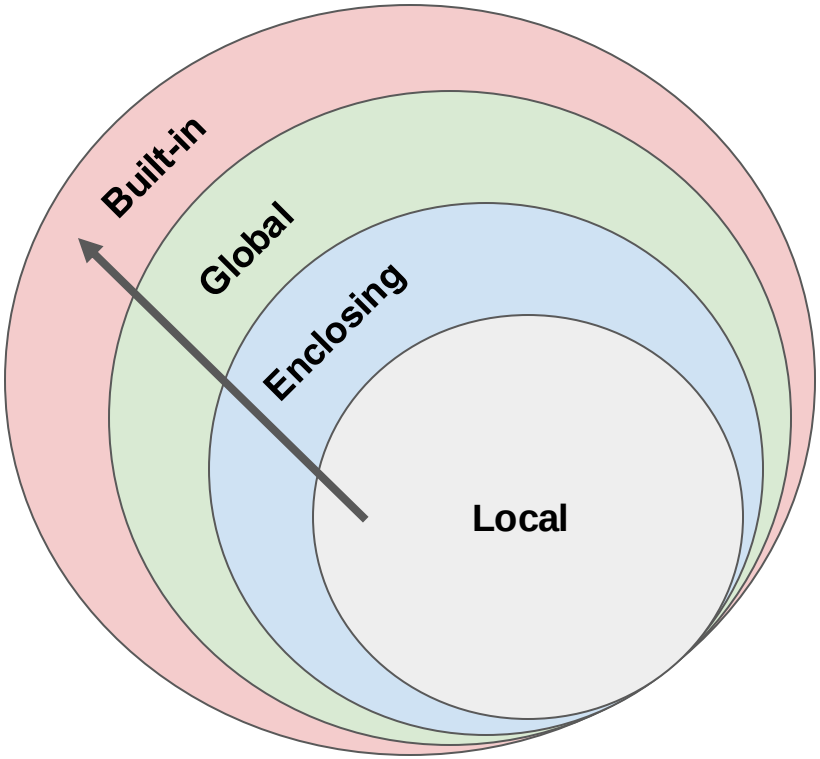
Come viene risolta (trovata) una variabile?

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(...))
```

Python: i moduli

In Python ci sono un sacco di funzionalità (funzioni ed oggetti) già disponibili con la cosiddetta “libreria standard”, ma che non sono “built-in”.

Questo vuol dire che non dovete installare niente per usarle, ma che dovete esplicitamente importare il modulo che le contiene.

Esempio con la radice quadrata:

```
>>> import math
>>> math.sqrt(600)
24.49489742783178
```

oppure

```
>>> from math import sqrt
>>> sqrt(600)
24.49489742783178
```

Python: i metodi di una classe

- I metodi di una classe si chiamano mettendoli alla fine della variabile dopo un punto

- Provate ad esempio con un tipo stringa (quindi della classe stringa), scrivendo una stringa ed un punto su VScode compaiono tutti i metodi di quella classe

```
‘mia_stringa’.nome_metodo()
```

Python: le liste

Uno dei tipi dati di Python un po’ più evoluti è la lista.

```
my_list = [1,2,3]                    # Lista di numeri
my_list = ['Marco', 'irene', 'paolo']  # Lista di stringhe
```

La lista ci introduce anche agli operatori di appartenenza:

Operator	Description	Example
in	Returns True if a sequence with the specified value is present in the object	x in y
not in	Returns True if a sequence with the specified value is not present in the object	x not in y

```
if 'Marco' in my_list ...
```

Python: le liste

Uno dei tipi dati di Python un po’ più evoluti è la lista.

```
my_list = [1,2,3]                    # Lista di numeri
my_list = ['Marco', 'irene', 'paolo']    # Lista di stringhe
my_list = ["ciao", 2.0, 5, [10, 20]] # Lista mista pericolosa!!!
```

Come per le stringhe accedo agli elementi con l’indice e posso creare sotto stringhe

```
x = [23,3,2,65, 6,7,8,9,10]
y = x[1:6:2] #dall'elemento di indice 1 all'elemento di indice 6 (escluso), con step 2
z = x[::2] #dall'inizio alla fine, con step 2.
```

Come per le stringhe posso concatenare e ripetere

Python: le liste sono mutabili

Le liste sono mutabili

```
x[1:3]=[1,2,3] #sostituisco gli elementi dall'indice 1 all'indice 3 (escluso)
x[1:3]=[]      #elimino gli elementi dall'indice 1 all'indice 3 (escluso)
```

Python: attenzione

Le stringhe hanno alcune cose in comuni con le liste (slices, molti metodi..)
Ma le stringhe sono immutabili

```
%%expect TypeError
saluti = 'Hello, world!’
saluti[0] = ‘B'
```

Devo creare una nuova stringa

```
nuovi_saluti = ‘B’ + saluti[1:]
print(nuovi_saluti )
```

I metodi delle liste modificano l'oggetto di partenza, i metodi delle stringhe non modificano la lista stessa

Python: attenzione

Tutte le modifiche che faccio su list2 le sto facendo anche su list1!!!!

```
list1 = ['carlo','magno']
print('prima: list1='+str(list1))
list2=list1 # list2 ed list1 puntano alla stesso oggetto in memoria.
list2[0]='alessandro’
print('dopo: list1='+str(list1))
```

```
prima: list1=['carlo', 'magno']
dopo: list1=['alessandro', 'magno']
```

Per puntare ad un oggetto diverso usare .copy() oppure list4=list3[:]

```
list3 = ['11',7,23]
list4 = list3.copy()
```

Python: le tuple

Liste di valori separati da virgole. Sono immutabili.

```
tupla1 = (2,)
tupla2 = 2,
```

Posso usare le tuple per assegnare più valori contemporaneamente

```
(a,b,c) = (1,2,3)
```

Python: i dizionari

Un altro tipo dati evoluto di Python sono i “dizionari”. Sono coppie chiave-valore: La chiave può essere un qualsisi oggetto immutabile, come tuple, stringe, interi etc.

```
my_dict = {'Trieste': 34100, 'Padova': 35100}   # stringa -> numero
```

Posso aggiungere elementi, cambiare valori e cancellare elementi:

```
my_dict[‘Venezia'] = 30121                  #aggiungere un elemento
my_dict[‘Venezia’]=30100                    #cambiare un valore
del my_dict[‘Padova']                       #cancellare un elemento
```

Python: i dizionari

Altri esempi di coppie chiave-valore:

```
my_dict = {'Trieste': 34100, 'Padova': 35100}   # stringa -> numero
my_dict = {34100: 'Trieste', 35100: 'Padova'}   # numero -> stringa
my_dict = {'Trieste': 'TS', 'Padova': 'PD'}     # stringa -> stringa
```

Si accede ai valori di un dizionario così:

```
my_dict['Trieste']
```

Valgono gli operatori di appartenenza della slide precedente, agiscono sulle chiavi

```
if 'Trieste' in my_dict ...
```

Python: i set

Collezione non ordinate di oggetti unici ed immutabili:

```
a={'a','a','a','c','b','s'} #a=set('s','5')
b = set('abracadabra')
print(type(a),type(b))
```

```
{'a', 'b', 's', 'c'} {'r', 'd', 'c', 'a', 'b'}
```

Ci sono le varie operazioni tra set

```
print ('a - b ', a - b)  # differenza
print ('a | b ', a | b)  # unione, or logico.
print ('a & b ', a & b)  #intersezione, and logico.
print('a^b      ' , a^b)           #simmetric
difference
print('a' in a)
```

Cosa vuol dire essere “pythonici”

Iterare sugli elementi di una lista

```
my_list = ["marco", "irene", "paolo"]

for i in range(len(my_list)):
    print(my_list[i])
```

vs.

```
my_list = ["marco", "irene", "paolo"]

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

Cosa vuol dire essere “pythonici”

Controllare se Marco è nella lista degli studenti:

```
my_list = ["marco", "irene", "paolo"]

for i in range(len(my_list)):
    if my_list[i] == "marco":
        print("Ho trovato marco!")
```

vs.

```
my_list = ["marco", "irene", "paolo"]

if "marco" in my_list:
    print("Ho trovato marco!")
```

Un primo esercizio

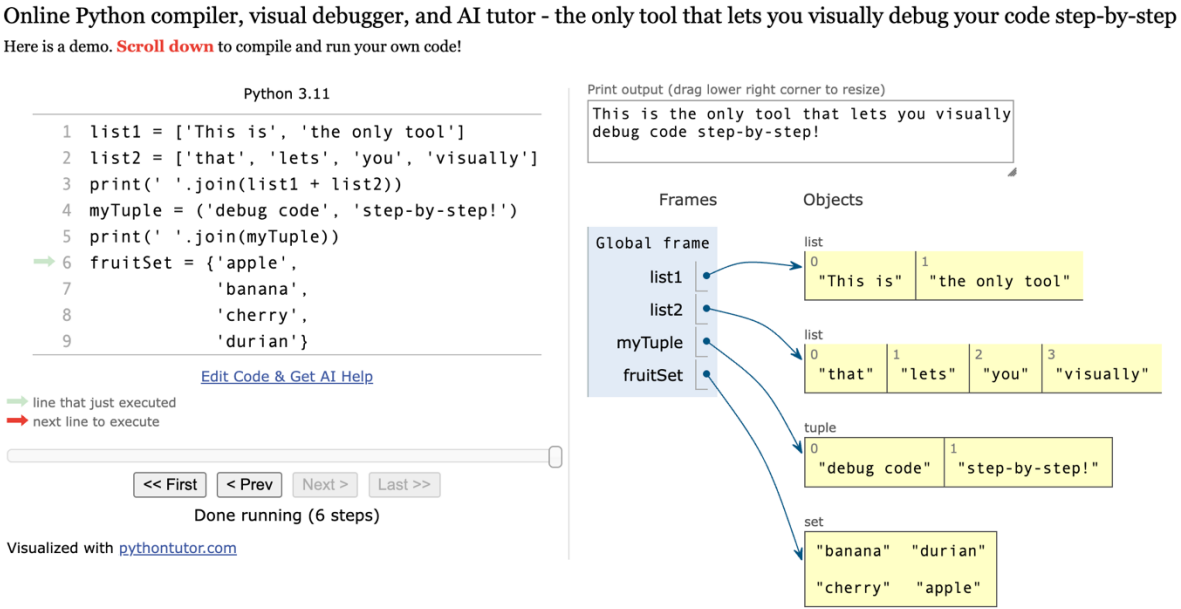
Scrivete una funzione `sum_list(my_list)`
che sommi tutti gli elementi di una lista.

Poi, scaricate il vostro script Python
e testatelo su Autograding

p.s. la lista passata da sommare è vuota, la funzione deve tornare `None` (la somma di una lista vuota non è definita!)

Online Python compiler

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



Lista Altri Esercizi

1. Stampare l'equivalente di 538 minuti nel formato 12h:32min
2. Definire una funzione che prende come argomento una parola e una lettera. Ritorna quante volte quella lettera è contenuta nella parola.
3. Scrivere una funzione che prende in input una stringa e ritorna True se è un palindromo, False altrimenti.
4. Definire una funzione che dati 3 numeri interi stabilisce se possono essere i valori dei lati di un triangolo e se si di che tipo di triangolo
5. Definire una funzione che prende in input una lista A, indici i, j, e scambia il valore di A[i] con A[j].
6. Definite la funzione fattoriale
7. Scrivere una funzione che prende in input due liste e ritorna `True` se le due liste hanno almeno un elemento in comune
8. Definire una funzione che prende in input una lista di numeri interi in [0, 9] e ritorna una lista di stringhe, corrispondenti ai numeri scritti in Italiano, es. [1,0,7,9,8] ->["uno","zero","sette","nove","otto"]