Fondamenti di Python

Parte 1

- **In Python la tipizzazione delle variabili è dinamica, non è necessario dichiarare le variabili prima di utilizzarle ne dichiarare il loro tipo**

In [11]:

```
a = 1
type(a)
```

Out[11]:

```
int
```

In [9]:

```
a = 1.34567
type(a)
```

Out[9]:

```
float
```

- **Le stringhe si definiscono con ' ' 0 " " sono seguenze IMMUTABILI di caratteri sulle quali è possibile itarare (es. for, while)**

In [12]:

```
a = 'abc'
a(2) = c
```

```
  File "<ipython-input-12-dfcb56d73929>", line 2
    a(2)=c
    ^
SyntaxError: cannot assign to function call
```

In [15]:

```
a = 'abc'
for letter in a :
    print(letter)
```

```
a
b
c
```

In [17]:

```
for letter in 'abc' :
    print(letter)
```

```
a
b
c
```

- **La funzione print() permette di stampare oggetti in output, gli ogetti passati alla funzione vengono convertiti in stringhe**

In [18]:

```
print (2+8)
```

```
10
```

In [19]:

```
print ('fanti')
```

```
fanti
```

In [20]:

```
print('a')
print('b')
```

```
a
b
```

- **La formattazione di print(), di default la funzione va a capo da sola, mentre per mettere due output nell stessa riga separati da uno spazio si fa nel seguente modo:**

In [23]:

```
print('a', end='')
print('\tb')
```

```
a b
```

- **Slicing-> notazione che permette di accedere ad una serie di elementi di una sequenza ordinata attraverso il loro indice**

In [36]:

```
a = 'babbimorti'
a[:10]
```

Out[36]:

```
'babbimorti'
```

In [37]:

```
a = 'babbimorti'
a[2:8:2]
```

Out[37]:

```
'bio'
```

In [38]:

```
a = 'babbimorti'
a[6:]
```

Out[38]:

```
'orti'
```

In [39]:

```
a = 'babbimorti'
a[::-1]
```

Out[39]:

```
'itromibbab'
```

- **Gli operatori sono quelli di default(+ * - ecc...) tranne l'elevamento a potenza(**) e la floor division(//) che ritorna la parte intera della divisione**

- **Possiamo perfino concatenare gli operatori di confronto**

- **Possiamo perfino concatenare gli operatori di confronto**

In [48]:

```
a = 3
b = 4
c = 5
a < b < c
```

Out[48]:

```
True
```

- **Gli operatori logici sono quelli che conosciamo, più interessanti invece sono gli operatori di *identità* -> ulizzati per confrontare gli oggetti, non per verificare se i valori sono uguali ma se sono lo stesso oggetto(is ------- is not), operatori di *appartenenza*-> (in ------- not in)**

In [49]:

```
a = 'babbimorti'
'babbi' in a
```

Out[49]:

```
True
```

In [52]:

```
a = 'babbimorti'
b = a
b is a
```

Out[52]:

```
True
```

In [57]:

```
a = 'babbimorti'
'c' not in a
```

Out[57]:

```
True
```

- **Costrutti condizionali -> (if, elif,else)**

In [36]:

```
elementi = [2, 3, 4, 6, 7]


for elemento in elementi :

    if elemento%2==0:
        print('multiplo')

    else :
        print('non multiplo')
```

```
multiplo
non multiplo
multiplo
multiplo
non multiplo
```

- **Loop**
  - ***while loop* -> esegue un blocco di codice fintanto che la condizione è vera**
  - ***for loop* -> itera su una serie di elementi (utilizzabile su qualsiasi struttura iterabile come tuple, stringhe, liste ecc..)**

In [39]:

```
i = 1

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

```
1
2
3
4
5
6
7
8
9
10
```

In [37]:

```
stringa = 'TonyLota'
for lettera in stringa:
    print(lettera)
```

```
T
o
n
y
L
o
t
a
```

- **continue/break**
  - ***break*-> utilizzato per uscire da un for o un while anche se l'iterazione sugli elementi non è finita**
  - ***continue* -> utilizzato per saltare il blocco corrente di codice e tornare alla dichiarazione for o while**

In [25]:

```
i = 10

while i >= 1:
    print(i)
    if i == 5:
        break
    i -= 1
```

```
10
9
8
7
6
5
```

In [3]:

```
for lettera in 'casta':
    if lettera == 't':
        continue
    print(lettera)
```

```
c
a
s
a
```

In [ ]:

```
i = 10

while i >= 1:
    if i == 5:
        print(1)
        break
    print(i)
    i -= 1
```

Esercizi 1

PDF python-base-1

Scrivi uno script che accetta un input consistente in un numero intero rappresentante un raggio, calcola l'area del rispettivo cerchio e stampa il risultato.

In [23]:

```
import math

raggio = input('Inserisci raggio:')
r = int(raggio)

print('Area =', math.pi * r *r)
```

```
Inserisci raggio:2
Area = 12.566370614359172
```

Scrivi uno script che accetta tre numeri interi. Se sono tutti diversi, stampa la somma dei tre, se due sono uguali stampa il risultato della somma di quelli uguali divisi per il terzo; se tutti e tre sono uguali, stampa il risultato di (n+n)^n.

In [47]:

```
#soluzione poco pythonica
num1 = input('Inserisci il primo numero:')
num2 = input('Inserisci il secondo numero:')
num3 = input('Inserisci il terzo numero:')

n1 = int(num1)
n2 = int(num2)
n3 = int(num3)

if n1 != n2 != n3 :
    print('La loro somma è: ', n1 + n2 + n3)

elif n1 == n2 == n3 :
    print((n1+n1) ** n1)

if n1 == n2 :
    if n3 != n2 :
        print((n1 + n2) / n3)

elif n1 == n3 :
    if n2 != n3 :
        print((n1 + n3) / n2)

elif n2 == n3 :
    if n1 != n3 :
        print((n2 + n3) / n1)
```

```
Inserisci il primo numero:2
Inserisci il secondo numero:2
Inserisci il terzo numero:4
1.0
```

In [10]:

```
# soluzione pythonica by Antonio  Bocale

# creo una lista vuota l
l = []


# faccio un ciclo for con 3 iterazioni in cui inserisco gli input
for i in range (1,4):
    l.append(int(input("Valore {i}: ")))
#creo un set con gli elementi della lista l
s = set(l)

# controllo la lunghezza del mio set se questo ha 3 elementi significa che gli elementi de
lla lista sono tutti elementi
# diversi perchè nei set non esistono doppioni

if len(s) == 3:
    print(sum(s))


# se la lunghezza è due allora siamo nel caso con due numeri uguali

elif len(s) == 2:
    duplicato = [i for i in s if l.count(i) == 2][0]
    s.remove(duplicato)
    singolo = s.pop()
    print(duplicato*2/singolo)
else :
    n = s.pop()
    print((n+n)**n)
```

```
Valore {i}: 2
Valore {i}: 3
Valore {i}: 5
10
```

Parte 2

- **Context manager, *with*-> definisce un blocco di codice con metodi definiti dal context manager, quindi f dopo la chiususra del contesto non esiste più (la close del file è gestista in modo automatico)**

In [ ]:

```
with open('text.txt', 'w') as f
    f.write("Test text, frist line.")
```

- **Moduli-> ci permetto di importatre funzionalità a python una sorta di applicazioni aggiuntive, per importare un module si ricorre alla keyword import**

In [ ]:

```
i = 0
```

In [4]:

```
import math
```

In [17]:

```
a = 'x'
from string import ascii_lowercase
a in ascii_lowercase
```

Out[17]:

```
True
```

In [18]:

```
a = 'a'
from string import ascii_uppercase
a in ascii_uppercase
```

Out[18]:

```
False
```

- **Una funzione si dichiara con *def***

In [15]:

```
def prova():
    print('prova')

prova()

type(prova)
```

```
prova
```

Out[15]:

```
function
```

- **Un metodo si definisce sempre con *def* ma all'interno di una classe. Metodi interessanti:**
  - **pillow**
  - **matplotlib**
  - **numpy**
  - **opencv**
  - **requests**
  - **sqlalchemy**
  - **beautifulsoup**
  - **pandas**

Strutture dati base :

- **Lista -> struttura molto versatile, che si definisce come un elenco di elementi (anche DISOMOGENEI es: char,int) separati da virgola, tra parentesi quadre(o con costruttore list(), importante con la funzione list se voglio passare più di un elemento ricordarsi di mettere la doppia tonda sennò non funziona).**

In [49]:

```
a = 2

lista = [1 , 'ciao', [1, 2, 3], a]
print(lista)
```

```
[1, 'ciao', [1, 2, 3], 2]
```

equivalente a :

In [65]:

```
a = 2

lista = list((1 , 'ciao', [1, 2, 3], a))
print(lista)
```

```
[1, 'ciao', [1, 2, 3], 2]
```

- ***Accesso* ad un elemento della lista:**

In [68]:

```
l = [1, 2, 3]
l[0]
```

Out[68]:

```
1
```

- ***Sostituzione*** **di un elemento della lista:**

In [69]:

```
l = [1, 2, 3]
l[1]='ciao'
print(l)
```

```
[1, 'ciao', 3]
```

- ***Cancellazione*** **di un elemento della lista :**

In [70]:

```
l = [1, 2, 3]
del l[1]
print(l)
```

```
[1, 3]
```

- ***Slicing*** **di una lista:**

In [72]:

```
l = [1, 2, 3]
l[0:3:2]
```

Out[72]:

```
[1, 3]
```

- **Esistono anche una serie di metodi e funzioni integrati che si possono utilizzare tra cui:**
  - **l.append()-> inserisce un elemento in 'coda' alla lista**
  - **l.count()-> ritorna il numero di volte che l'elemento passato è presente nella lista**
  - **l.index()->ritorna la posizione dell'elemento nella lista**
  - **l.insert()-> inserisce un elemento in una posizione specifica della lista (la posizione si passa come parametro)**
  - **l.len()-> ritorna la lunghezza della lista**
  - **l.pop()-> estrae un elemento dalla lista e lo elimina**

- **Tupla -> sono sequenze proprio come le liste ma a differenza di quest'ultime non possono essere modificate**

In [75]:

```
t = (1, 2, 3, 4)
t
```

Out[75]:

```
(1, 2, 3, 4)
```

In [12]:

```
t = (1, 2, 3, 4)
t[3] = 1
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-12-07d43d1a2ccf> in <module>
      1 t = (1, 2, 3, 4)
----> 2 t[3] = 1

TypeError: 'tuple' object does not support item assignment
```

- **Set -> collezione di elementi non ordinata ed indicizzata di elementi unici e immutabili ovvero il set è modificabile ma i suoi elementi devono essere hashable(non modificabili)**
  - **esempio :**
    - **posso passare a un set una stringa od un int? Si!**
    - **posso passare ad un set una lista? No!**
  - **I set presentano un sacco di funzioni facilmente consultabili.**

In [81]:

```
s = {1, 2, 3}
s
```

Out[81]:

```
{1, 2, 3}
```

In [82]:

```
s = {1, [1, 2]}
s
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-82-e5c4e0146cbb> in <module>
----> 1 s = {1, [1, 2]}
      2 s

TypeError: unhashable type: 'list'
```

Metodo	Descrizione
.difference()	Ritorna un set contenente le differenze tra due o più set
.difference_update()	Rimuove gli elementi nel set che sono presenti in un altro set specificato
.intersection()	Ritorna un set che è l'intersezione di altri due set
.intersection_update()	Rimuove gli elementi nel set che non sono presenti nell'altro/negli altri set specificato/i
.isdisjoint()	Ritorna un booleano che indica se due set non hanno un'intersezione o meno

Metodo	Descrizione
.issubset()	Ritorna un booleano che indica se il set è compreso in un altro set specificato o meno
.issuperset()	Ritorna un booleano che indica se il set contiene un altro set specificato o meno
.symmetric_difference()	Ritorna un set con le differenze simmetriche di due set
.symmetric_difference_update()	Rimuove gli elementi comuni tra due set ed inserisce nel primo le differenze simmetriche
.union()	Ritorna un set contenente l'unione di uno o più set

.update()	Modifica il set con l'unione di uno o più set

- **Dizionari -> i dizionari non sono indicizzati da numeri ma da *key* che possono esssere di qualsiasi tipo IMMUTABILE accoppiati con *value***

In [86]:

```
d = {'a' : 1, 'b' : 2, 'c' : 3}

print(d['b'])

print(d.get('a'))
```

```
2
1
```

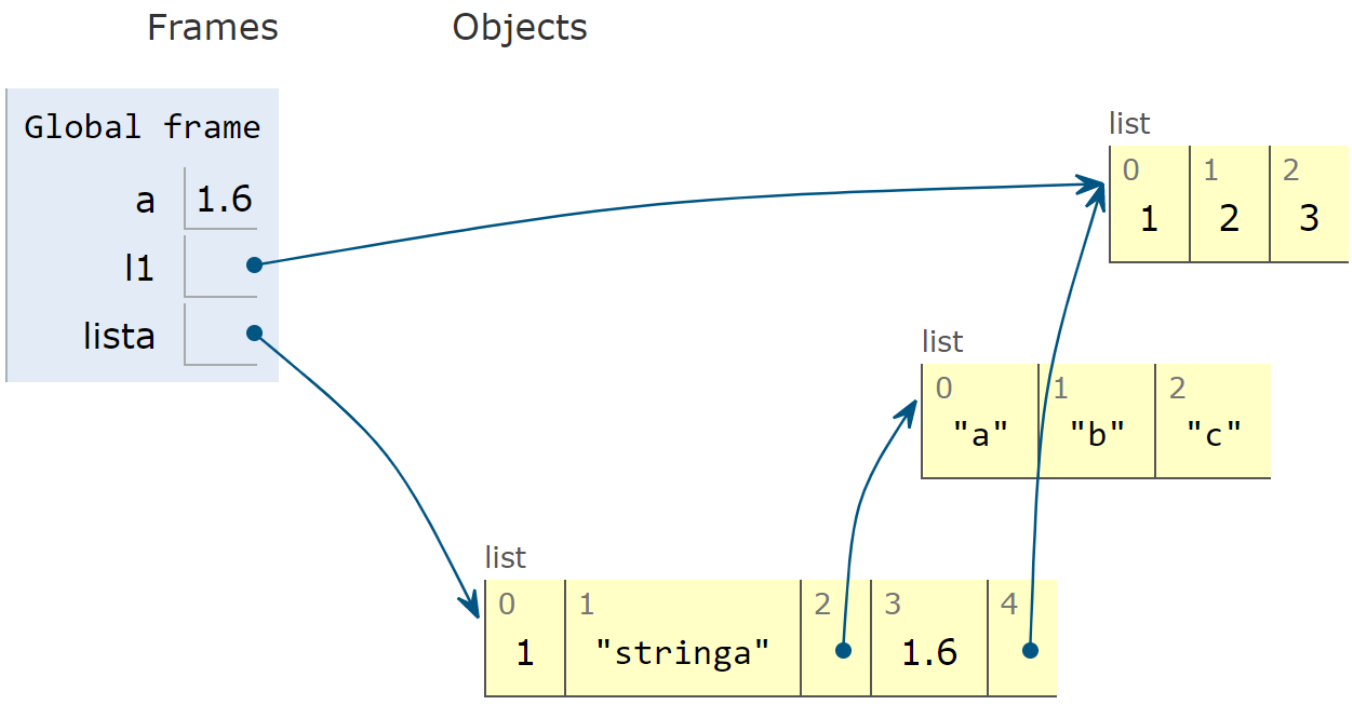
- **Esempio riassuntivo liste e spiegazione delle copy**

In [25]:

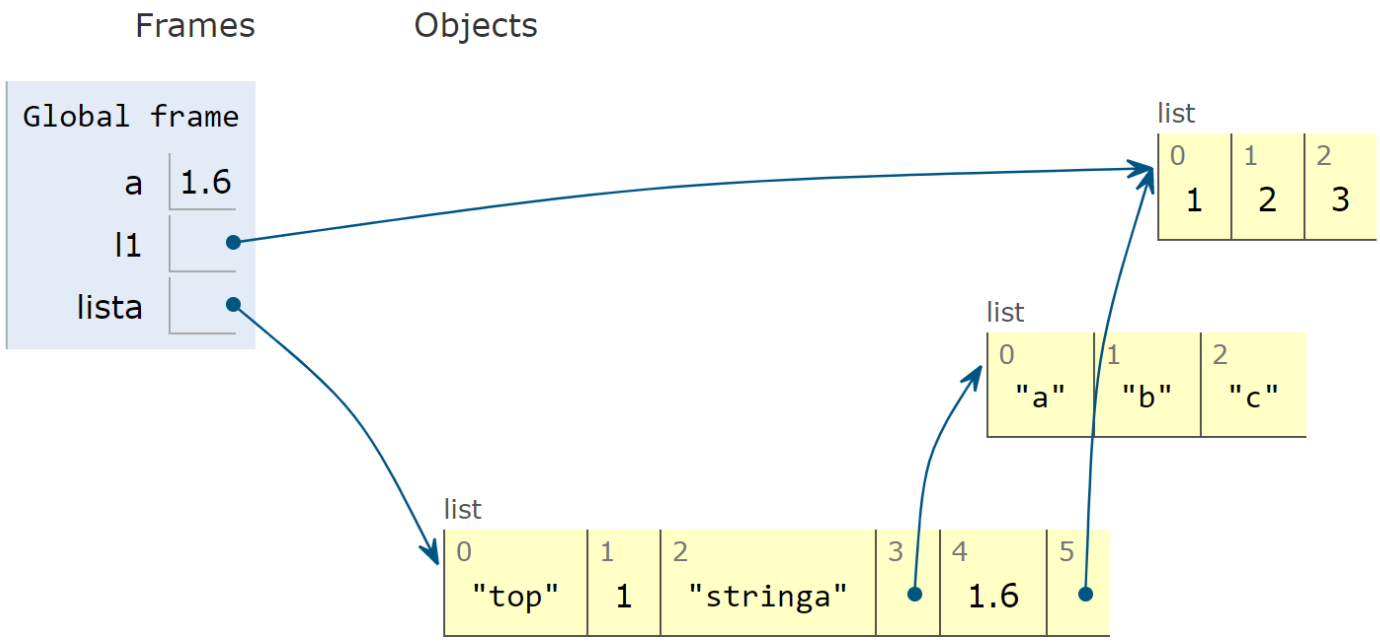
```
a = 1.6
l1 = [1,2,3]
lista = [1, 'stringa', ['a', 'b' , 'c'], a , l1]

lista.insert(0, 'top')
l1.append(4)

l2 = l1 #shallow copy
l3 = l1.copy() #copy
l4 = lista.copy
from copy import deepcopy
l5 = deepcopy(lista) #deep copy

print(lista [5][0])
print (lista)
```

```
1
['top', 1, 'stringa', ['a', 'b', 'c'], 1.6, [1, 2, 3, 4]]
```

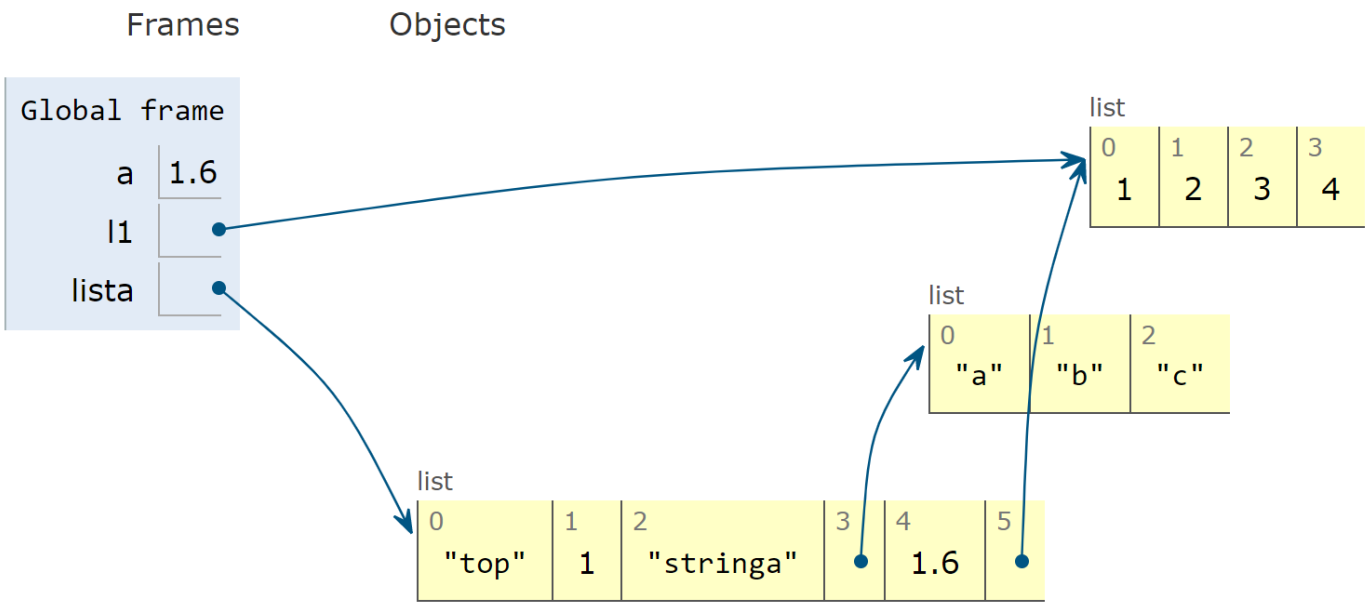
- **Nelle prime tre righe di codice abbiamo la dichiarazione delle variabili**



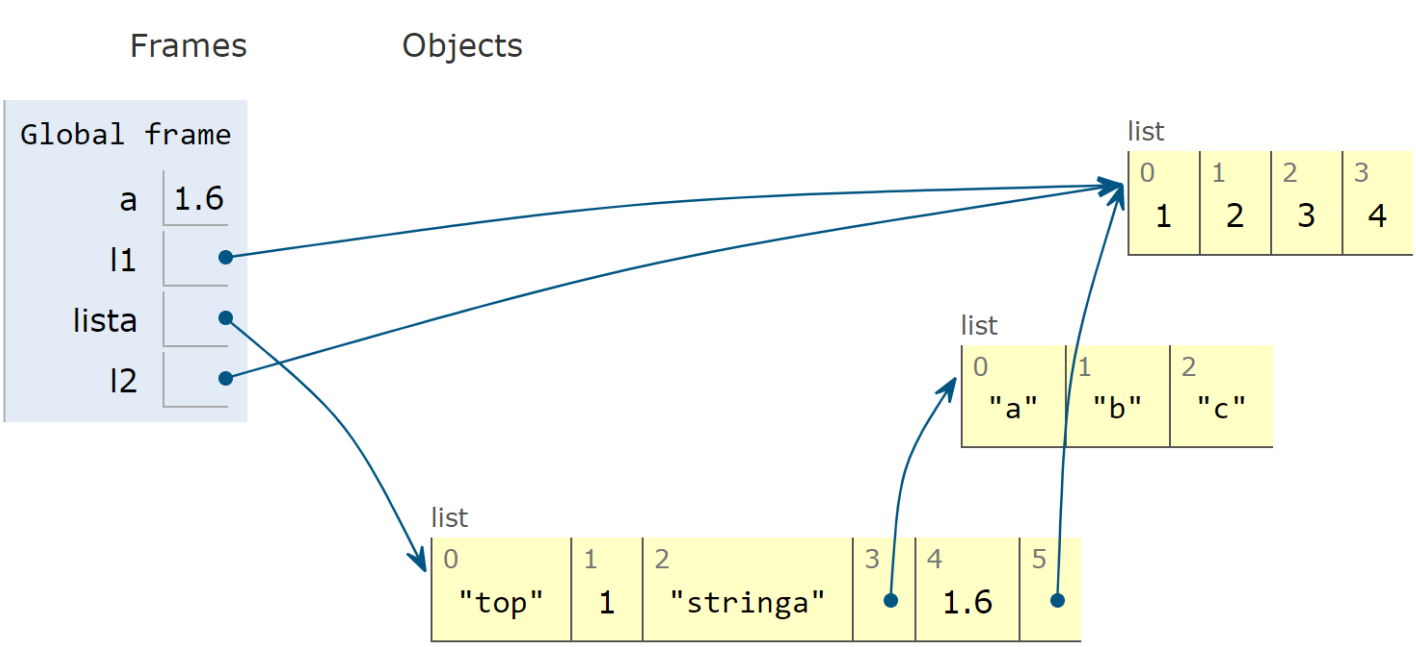
- **lista.insert(0, 'top')** -> inserisce la stringa 'top' nella posizione 0 della lista (prima posizione)



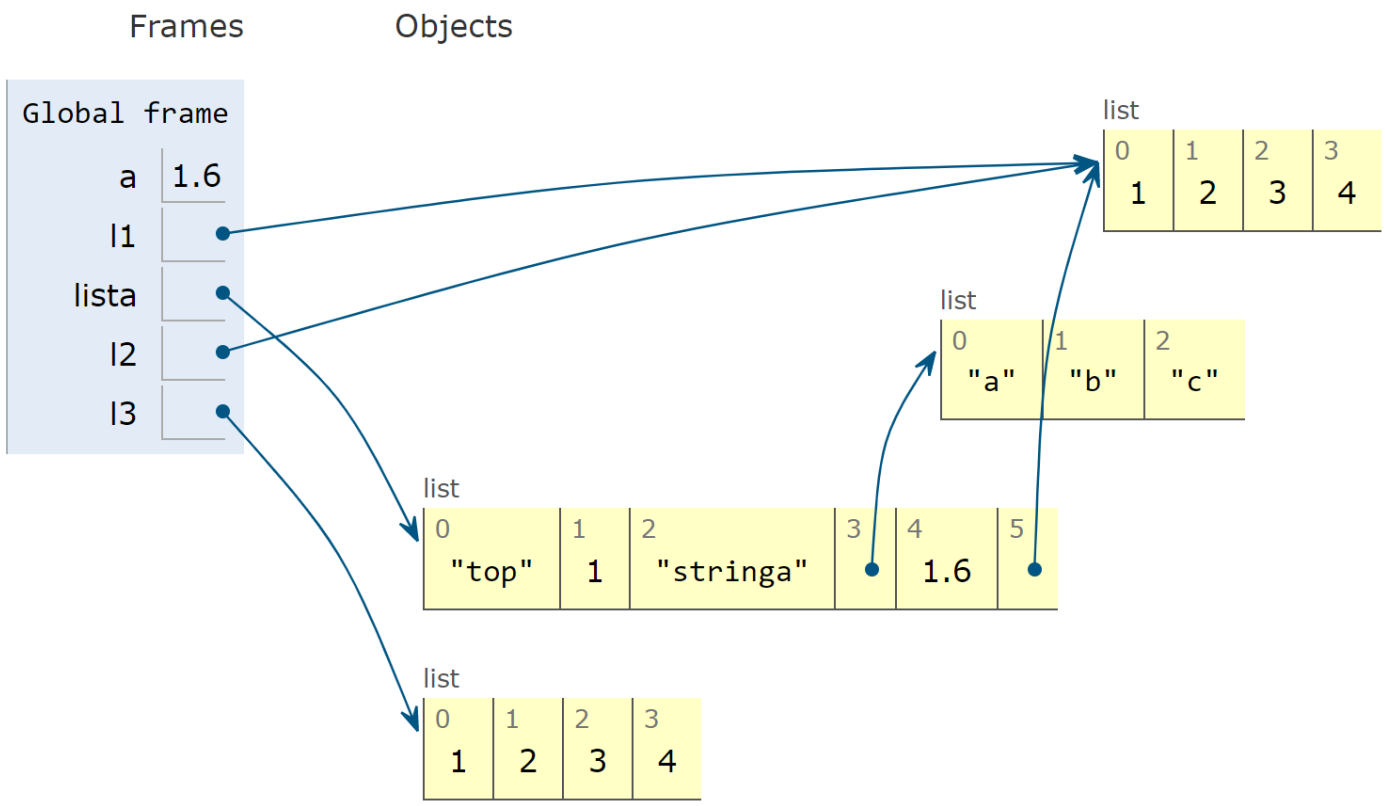
- **l1.append(4)** -> 'appende' il 4 alla coda della lista l1, che da 123 diventa 1234



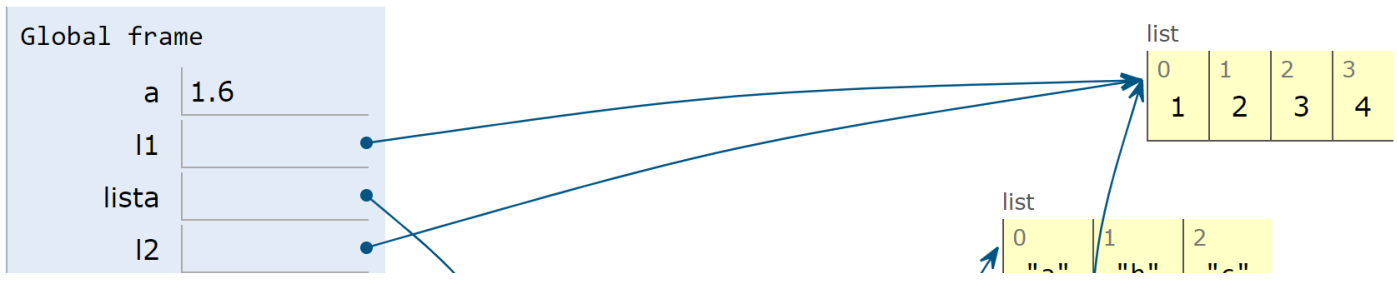
- **l2 = l1** -> è un esempio di shallow copy (copia superficiale)
  - l2 infatti non crea una copia effettiva di l1 ma va a puntare alla stessa area di memoria di l1

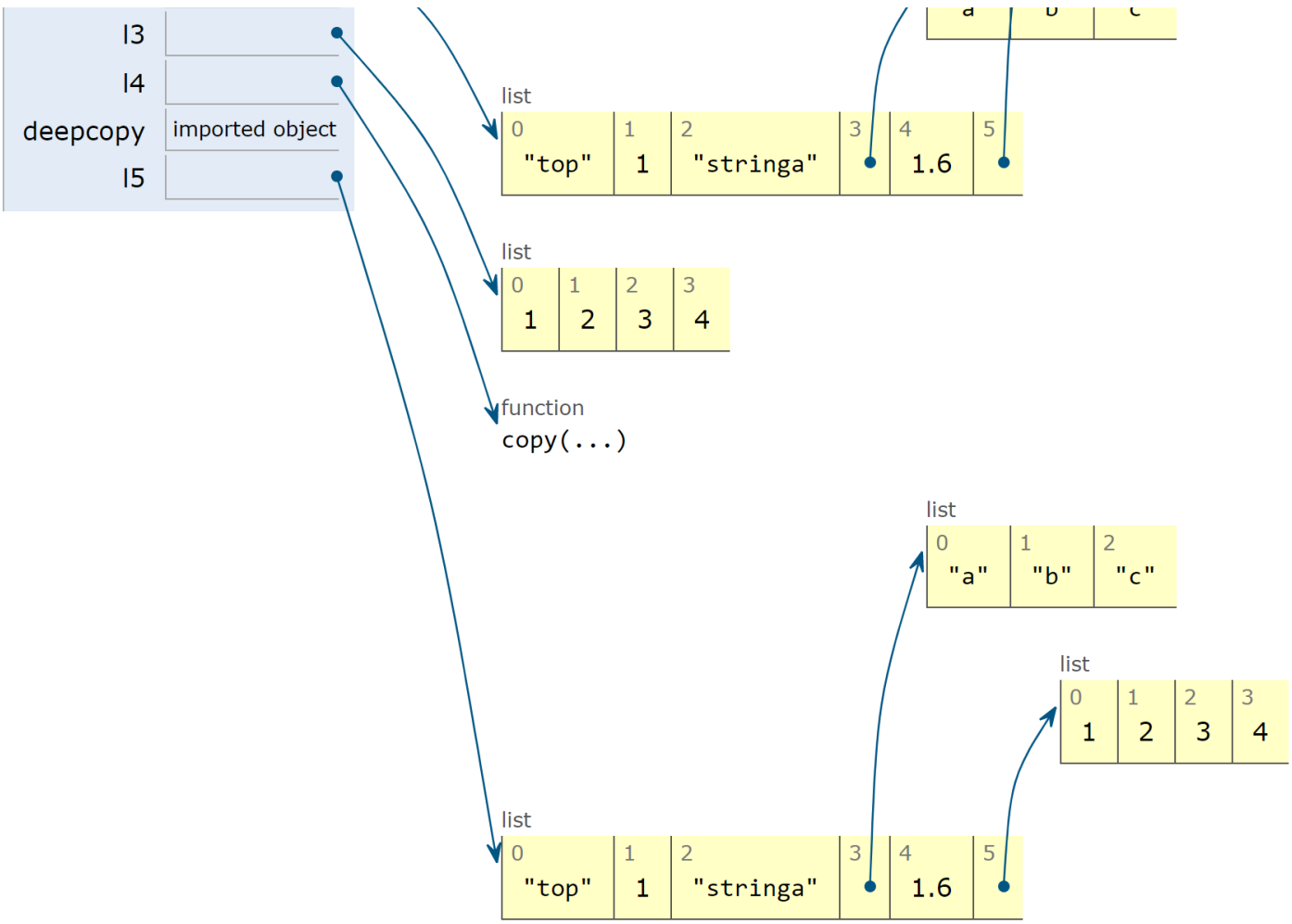


- **l3 = l1.copy() -> è un esempio di copy (copia effettiva)**
  - **l3 non va più a puntare alla stessa area di memoria puntata da l1 ma crea una nuova allocazione di memoria con la copia della lista l1**



- **l5 = deepcopy(lista) -> è un esempio di deep copy(copia profonda)**
  - **l5 copia in modo esatto lista con tutti i vari puntatori annidati**





- **print(lista [5][0]) -> stampa del 5° elemento della lista il 1° elemento (un pò confuso ma non so come altro scriverlo) in pratica va al 5° elemento della lista che in questo caso è l1 e stampa il 1° elemento di l1, quindi 1.**

- **Generatori -> sono strumenti per la creazione di iteratori (quegli oggetti che hanno implementati i metodi iter e next).**
  - **Possono essere definiti anche mediante la funzione *yield* ogni volta che vogliono restituire un dato**

In [2]:

```
def gen():
    for n in range(3):
        yield n

for n in gen():
    print(n)
```

```
0
1
2
```

- **Possono inoltre svolgere l'iterazione passo passo, bloccandosi dopo ogni iterazione e svolgendo la successiva grazie alla funzione next**

In [4]:

```
def gen():
    for n in range(3):
        yield n

iter_gen = iter(gen())
next(iter_gen)
next(iter_gen)
```

Out[4]:

```
1
```

- **Generator expression -> simili alle lambda function che creano funzioni anonime, le generator expression creano funzioni generatori anonime.**

In [5]:

```
sum(x for x in range(5))
```

Out[5]:

```
10
```

In questo esempio viene creata una variabile nella quale viene addizionato l'elemento successivo a ogni ciclo

- **Sintax comprehension -> fornisce un modo coinciso per creare un qualsiasi tipo di struttura dati.**

Forma tradizionale:

In [8]:

```
quadrati = []
for n in range(11):
    quadrati.append(n ** 2)
print(quadrati)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Sintax comprehension:

In [10]:

```
quadrati = [n ** 2 for n in range(11)]
print(quadrati)
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

Lista SC

In [12]:

```
l = [n for n in range (5)]
print(l)
```

```
[0, 1, 2, 3, 4]
```

Tupla SC

- **è necessario utilizzare la forma tuple(), perchè altrimenti si crea un generatore**

In [15]:

```
t = tuple(n for n in range(5))
print(t)
```

```
(0, 1, 2, 3, 4)
```

Dizionario SC

- **La funzione *zip()* associa gli elementi k,v ad ogni iterazione**

In [16]:

```
d = dict((k,v) for k,v in zip(range(5), 'abcde' ))
```

```
print(d)
```

```
{0: 'a', 1: 'b', 2: 'c', 3: 'd', 4: 'e'}
```

Set SC

In [17]:

```
s =  {n for n in range (5)}
print(s)
```

```
{0, 1, 2, 3, 4}
```

Generatore SC

In [18]:

```
g = (n for n in range(5))
print(g)
```

```
<generator object <genexpr> at 0x0000024E454C4200>
```

E' anche possibile stabilire delle condizioni

In [19]:

```
l = [(n, 'pari') if n % 2 == 0 else (n, 'dispari') for n in range(10)]
for el in l:
    print(el)
```

```
(0, 'pari')
(1, 'dispari')
(2, 'pari')
(3, 'dispari')
(4, 'pari')
(5, 'dispari')
(6, 'pari')
(7, 'dispari')
(8, 'pari')
(9, 'dispari')
```

Unicode -> è un sistema progettato per rappresentare ogni simbolo/carattere di ogni lingua.

- **Esistono due funzioni fondamentali per la codifica e decodifica di stringhe**
  - ***encode* -> restituisce una vesrione codificata della stringa come oggetto byte**
  - ***decode* -> restituisce una stringa decodificata da un oggetto byte**

In [26]:

```
'我'.encode('utf-8')
```

Out[26]:

```
b'\xe6\x88\x91'
```

In [27]:

```
b'\xe6\x88\x91'.decode()
```

Out[27]:

```
'我'
```