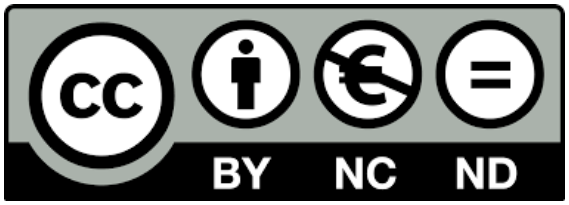
Introduzione alla programmazione

ITS - Umbria

A.S. 2022-23

MATERIA: fondamenti di informatica

Docente: prof. Paolo Bernardi



Liste 2: metodi

Nella lezione precedente sono state mostrate le operazioni possibili sulle sequenze, nello specifico stringhe e liste.

Queste operazioni erano l'indicizzazione, l' "affettatura" (slicing), la somma (concatenazione), la moltiplicazione (ripetizione) e il controllo dell'appartenenza. Inoltre, Python ha funzioni integrate per trovare la lunghezza di una sequenza e per trovare i suoi elementi più grandi e più piccoli.

Liste 2: operazioni e funzioni integrate

Operatore Funzione	Descrizione	Esempio
+	Concatenazione - Aggiunge valori su entrambi i lati dell'operatore	In : [1, 2, 3] + [4, 5, 6] Out : [1, 2, 3, 4, 5, 6]
*	Ripetizione: crea nuove liste, concatenando più copie della stessa lista	In : [1, 2] * 3 Out : [1, 2, 1, 2, 1, 2]
[]	Slice: fornisce l'elemento dall'indice specificato	In : a = [1, 2, 3, 4] a[2] Out : 3
[:]	Range Slice - Fornisce gli elementi dell'intervallo specificato	In : a = [1, 2, 3, 4] a[1:2] Out : [2, 3]
in	Appartenenza - Restituisce vero se un elemento esiste nella lista specificata	In : a = [1, 2, 3, 4] 4 in a Out : True
not in	Appartenenza - Restituisce vero se un elemento non esiste nella lista specificata	In : a = "Ciao" 4 not in a Out : True
len()	Lunghezza - Restituisce il numero di elementi nell'elenco	In : a = "ITS" len(a) Out : 3

Liste 2: funzioni integrate max, min e list

Sintassi

```
>>> max(sequence)  # restituisce l'elemento massimo di sequence
>>> min(sequence)  # restituisce l'elemento minimo di sequence
>>> list(sequence)  # converte la sequenza sequence in una lista
```

```
Python Console
>>> list1, list2 = ['C++','Java', 'Python'], [456, 700, 200]
>>> max(list1)
'Python'
>>> max(list2)
700
>>> min(list1)
'C++'
>>> min(list2)
200
>>> s = "Grande Python!"
>>> list(s)
['G', 'r', 'a', 'n', 'd', 'e', ' ', 'P', 'y', 't', 'h', 'o', 'n', '!']

>>>
```

Liste 2: metodo append

Sintassi

```
>>> a.append(b)  # 'appende' in coda alla lista a l'oggetto b
```

Il metodo append(*obj*) inserisce l'oggetto *obj* in ultima posizione alla lista esistente.

Questo metodo non restituisce alcun valore, ma aggiorna l'elenco esistente.

```
Python Console
>>> list1 = ['C++', 'Java', 'Python']
>>> list1.append('C#')
>>> list1
['C++', 'Java', 'Python', 'C#']

>>>
```

Liste 2: metodo count

Sintassi

```
>>> a.count(b)  # conta le occorrenze dell'oggetto b in a
```

Il metodo count(*obj*) restituisce il conteggio delle presenze dell'oggetto *obj* nella lista.

```
list1 = [5, 6, 1, 1, 6, 5, 6, 5, 4, 5, 5, 6, 6, 1, 2, 5, 6, 3, 5, 4]

list1.count(1) ->  3
list1.count(2) ->  1
list1.count(3) ->  1
list1.count(4) ->  2
list1.count(5) ->  7
list1.count(6) ->  6
```

Liste 2: metodo extend

Sintassi

```
>>> a.extend(b)  # 'appende' in coda alla lista a la sequenza b
```

Il metodo extend(*seq*) inserisce la sequenza *seq* in ultima posizione alla lista esistente.

Questo metodo non restituisce alcun valore, ma aggiorna l'elenco esistente.

```
Python Console
>>> list1 = ['C++', 'Java', 'Python']
>>> list2 = ['C#', "Perl", "Ruby"]
>>> list1.extend(list2)
>>> list1
['C++', 'Java', 'Python', 'C#', 'Perl', 'Ruby']

>>>
```

Liste 2: metodo insert

Sintassi

```
>>> a.insert(i, b)  # inserisce l'oggetto b in a alla posizione di indice i.
```

Il metodo insert(*index*, *obj*) inserisce l'oggetto *obj* nella posizione di indice *index*, gli altri elementi vengono spostati.

Questo metodo non restituisce alcun valore, ma inserisce l'elemento specificato nell'indice specificato.

```
Python Console
>>> list1 = ['C++', 'Java', 'Python', 'C#', "Perl", "Ruby"]
>>> list1
['C++', 'Java', 'Python', 'C#', 'Perl', 'Ruby']
>>> list1.insert(2, "Lua")
>>> list1
['C++', 'Java', 'Lua', 'Python', 'C#', 'Perl', 'Ruby']
>>> list1.insert(2, ["Grande", "ITS"])
>>> list1
['C++', 'Java', ['Grande', 'ITS'], 'Lua', 'Python', 'C#', 'Perl', 'Ruby']

>>>
```

Liste 2: metodo pop

Sintassi

```
>>> a.pop([i])  # elimina e restituisce l'elemento di indice i
```

Il metodo pop(*index*) rimuove e restituisce l'elemento di indice *index* dall'elenco.

Index è un parametro facoltativo: se non presente viene rimosso l'ultimo elemento della lista

```
Python Console
>>> list1 = ['C++', 'Java', 'Lua', 'Python', 'C#', 'Perl', 'Ruby']
>>> list1
['C++', 'Java', 'Lua', 'Python', 'C#', 'Perl', 'Ruby']
>>> list1.pop(2)
'Lua'
>>> list1
['C++', 'Java', 'Python', 'C#', 'Perl', 'Ruby']
>>> list1.pop()
'Ruby'
>>> list1
['C++', 'Java', 'Python', 'C#', 'Perl']

>>>
```

Liste 2: metodo remove

Sintassi

```
>>> a.remove(b)  # elimina l'elemento b dalla lista a
```

Il metodo remove(*obj*) rimuove l'elemento *obj* dalla lista.

Questo metodo non restituisce alcun valore, ma rimuove semplicemente l'oggetto specificato dalla lista

```
Python Console
>>> list1 = ['C++', 'Java', 'Lua', 'Python', 'C#', 'Perl', 'Ruby']
>>> list1
['C++', 'Java', 'Lua', 'Python', 'C#', 'Perl', 'Ruby']
>>> list1.remove("Perl")
>>> list1
['C++', 'Java', 'Lua', 'Python', 'C#', 'Ruby']

>>>
```

Liste 2: metodo reverse

Sintassi

```
>>> a.reverse()  # capovolge l'ordine degli elementi di a
```

Il metodo reverse() capovolge l'ordine degli elementi della lista.

Questo metodo non restituisce alcun valore.

```
Python Console
>>> list1 = ['C++', 'Java', 'Lua', 'Python', 'C#', 'Perl', 'Ruby']
>>> list1.reverse()
>>> list1
['Ruby', 'Perl', 'C#', 'Python', 'Lua', 'Java', 'C++']
>>> list2 = [12, "Gennaio", 2020]
>>> list2.reverse()
>>> list2
[2020, 'Gennaio', 12]

>>>
```

Liste 2: metodo sort

Sintassi

```
>>> a.sort([key,] [reverse])

# Questo metodo ordina l'elenco in uso,utilizzando solo confronti tra
# elementi; i parametri, se presenti, vanno passati attraverso keyword
# che permettono di cambiare i criteri di ordinamento della lista
```

```
Python Console
>>> list1 = ['C++', 'Java', 'Python', 'C#', "Perl", "Ruby"]
>>> list1.sort(reverse=True)
>>> list1
['Ruby', 'Python', 'Perl', 'Java', 'C++', 'C#']
>>> list1.sort()
>>> list1
['C#', 'C++', 'Java', 'Perl', 'Python', 'Ruby']

>>>
```

Liste 2: List Comprehension

La list comprehension fornisce un modo conciso per creare liste. Si ottiene racchiudendo il costrutto in parentesi quadre, contenenti un'espressione seguita da una clausola for, quindi zero o più istruzioni for o if.
Le espressioni possono essere qualsiasi cosa, nel senso che è possibile inserire tutti i tipi di oggetti nelle liste.
Il risultato sarà una nuova lista risultante dalla valutazione dell'espressione nel contesto delle istruzioni for e if che lo seguono. La list comprehension restituisce sempre come risultato una lista.

Sintassi

```
>>> [ expression for item in sequence if condition ]
```

Espressioni Equivalenti

```
>>> for item in sequence:
...   if condition:
...     expression
```

Liste 2: List Comprehension

Esempio: si voglia generare una lista di tutti i numeri minori o uguali a 20 divisibili per 3 oppure per 5

Soluzione classica

```
Python Console
>>> a = []
>>> for i in range(21):
...     if i % 3 == 0 or i % 5 == 0:
...         a.append(i)
...
>>> print(a)
[0, 3, 5, 6, 9, 10, 12, 15, 18, 20]

>>>
```

List Comprehension

```
Python Console
>>> a = [x for x in range(21) if x % 3 == 0 or x % 5 == 0]
>>> print(a)
[0, 3, 5, 6, 9, 10, 12, 15, 18, 20]

>>>
```

Liste 2: List Comprehension

Esempio: si voglia generare le prime 11 potenze di 2

Soluzione classica

```
Python Console
>>> a = []
>>> for i in range(11):
...     a.append(2 ** i)
...
>>> print(a)
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

>>>
```

List Comprehension

```
Python Console
>>> a = [2 ** i for i in range(11)]
>>> print(a)
[1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]

>>>
```

Liste 2: List Comprehension

Esempio

```
Python Console
>>> from random import randint
>>> a = [randint(-10, 10) for i in range(20)]
>>> a
[2, -5, -3, 4, 4, 9, 5, -9, -3, 1, 10, -5, 3, 1, 1, -10, -10, 7, 4, -3]
>>> b = [x for x in a if x > 0]
>>> b
[2, 4, 4, 9, 5, 1, 10, 3, 1, 1, 7, 4]

>>>
```

Liste 2: List Comprehension

Esempio

```
Python Console
>>> s = "Cristoforo Colombo scoprì l'America il 12 ottobre 1492"
>>> a = []
>>> for w in s.split():
...     if w.isdecimal():
...         a.append(w)
...
>>> print(a)
['12', '1492']
>>> b = [w for w in s.split() if w.isdecimal()]
>>> print(b)
['12', '1492']

>>>
```

NB: il metodo split

Tuple: introduzione

Le tuple sono sequenze simili alle liste, ma, a differenza delle liste, sono sequenze immutabili e non possono essere modificate nel corso del programma, esattamente come le stringhe. Contengono oggetti eterogenei, identificati da un indice numerico.

Le tuple sono immutabili, ma i loro elementi, se sono mutabili, possono essere cambiati, per cui se un elemento di una tupla è una lista questa può essere alterata e svuotata, ma non può essere tolta dalla tupla.

Le tuple sono rappresentate come insiemi di valori separati da virgole e racchiusi fra parentesi tonde; per il resto abbiamo operatori e notazioni analoghe a quelle delle liste e delle stringhe.

Tuple: introduzione

```
Python Console
>>> a = ("ITS", "Matematica", "Informatica", 23, 5, 2019)
>>> a
('ITS', 'Matematica', 'Informatica', 23, 5, 2019)
>>> type(a)
<class 'tuple'>

>>>
```

Attenzione: in questo esempio dobbiamo prestare attenzione ai caratteri "(" e ")". Sono proprio queste ultimi, le **parentesi tonde**, che indicano a Python che vogliamo creare una tupla.

Tuple: operazioni

Come per le stringhe, è possibile applicare una serie di operazioni sulle tuple, come dalla tabella sottoriportata:

Operatore	Descrizione
+	Concatenazione - Aggiunge valori su entrambi i lati dell'operatore
*	Ripetizione: crea nuove liste, concatenando più copie della stessa lista
[]	Slice: fornisce l'elemento dall'indice specificato
[:]	Range Slice - Fornisce gli elementi dell'intervallo specificato
in	Appartenenza - Restituisce vero se un elemento esiste nella lista specificata
not in	Appartenenza - Restituisce vero se un elemento non esiste nella lista specificata

Tuple: operazioni

```
Python Console
>>> a = (1, 3.5,'c')       # definisce una tupla di 3 elementi
>>> print(a)
(1, 3.5, 'c')
>>> b = ()                  # è una tupla vuota
>>> print(b)
()
>>> a[0] = 33               # da errore, la tupla non e' mutabile
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
>>> a[1]                    # il secondo elemento ( nel nostro 3.5)
3.5
>>> len(a)                  # numero di elementi
3
>>> a.index(3.5)
1

>>>
```

Tuple: concatenazione, ripetizione e slicing

```
Python Console
>>> a = (2, 3)
>>> b = (5, 5)
>>> c = a + b
>>> print(c)
(2, 3, 5, 5)
>>> hex(id(c))
'0x7f4683489908'
>>> c = c + a
>>> print(c)
(2, 3, 5, 5, 2, 3)
>>> hex(id(c))
'0x7f4683267948'
>>> c = a * 3
>>> print(c)
(2, 3, 2, 3, 2, 3)
>>> hex(id(c))
'0x7f468328d1c8'

>>>
```

```
Python Console
>>> a = ("ITS", "Matematica", "Informatica", 23, 5, 2019)
>>> a[3]
23
>>> a[:-2]
('ITS', 'Matematica', 'Informatica', 23)
>>> a[1:-1]
('Matematica', 'Informatica', 23, 5)
>>> a[1:-1:2]
('Matematica', 23)
>>> a[2:]
('Informatica', 23, 5, 2019)

>>>
```

Tuple: conversioni

Le tuple possono essere facilmente mutate in liste e viceversa

```
Python Console
>>> a = [1]
>>> type(a)
<class 'list'>
>>> b = tuple(a)
>>> b
(1,)
>>> type(b)
<class 'tuple'>

>>>
```

```
Python Console
>>> a = ("ITS", "Matematica", "Informatica", 23, 5, 2019)
>>> type(a)
<class 'tuple'>
>>> b = list(a)
>>> b
['ITS', 'Matematica', 'Informatica', 23, 5, 2019]
>>> type(b)
<class 'list'>

>>>
```

Tuple: utilizzo

NOTA: qualcuno si domanderà a cosa servono le tuple, visto che non offrono niente in più delle liste, ma presentano molte funzionalità in meno.

È una domanda legittima: il motivo principale sta proprio nella loro immutabilità che gli permette di fungere da indici per i dizionari (argomento che verrà trattato più avanti).

Alcuni esercizi

1) Calcolare la somma dei numeri naturali da 1 a 500 e stamparla a video.
2) Si scriva un programma che legga 5 numeri interi da tastiera. Il numero letto viene inserito in una lista se e soltanto se esso non è una copia di un numero già presente nella lista stessa. Il processo continua finché 5 numeri diversi tra loro sono inseriti da tastiera.
   A quel punto il programma calcola e stampa il valore medio dei 5 numeri inseriti.
3) Trovare gli indici di tutte le occorrenze della variabile *val1* nella lista mylist, entrambe inserite in input da tastiera. Il risultato, sotto forma di lista, va visualizzato sullo schermo.
   Ad esempio, se val1 = 1 e mylist = [1, 3, 6, 9, 1, 1, 2, 1, 2] il risultato deve essere [0, 4, 5, 7].
4) Riprendere l'esercizio n.3 con la seguente variante: mylist è popolata con numeri casuali generati dalla funzione randint.
5) Data una lista generata attraverso la funzione randint, se ne stampi la media aritmetica, la moda e la mediana.

Bibliografia

- Marco Beri, Programmare in Python, Apogeo
- Kenneth A. Lambert, Programmazione in Python, Apogeo
- Python Software Foundation, https://docs.python.org/3/
- Python - sito ufficiale della comunità italiana, www.python.it/doc/
- W3C programming language tutorials, www.w3schools.com/python/
- "The Hitchhiker's Guide to Python!", Kenneth Reitz, Tanya Schlusser, 2018,
  - https://media.readthedocs.org/pdf/python-guide/latest/python-guide.pdf
- www.tutorialspoint.com
  - www.tutorialspoint.com/python3/index.htm

Bibliografia

Software (free license)

- Interprete Python3: www.python.org/downloads/
- IDE: PyCharm (community): www.jetbrains.com/pycharm/download/

Immagini (free license)

- Fonte: www.duckduckgo.com