

In Python tutto è un oggetto!

Oggetti di tipo intero, float, stringa, null, bool e tuple

Stringhe

Stampiamo l'oggetto "Hello World"

```
In [1]: print("Hello World")
```

Hello World

Stabiliamo il tipo dell'oggetto "Hello World" (per meglio dire la "classe")

```
In [2]: print(type("Hello World"))
```

<class 'str'>

Assegniamo all'oggetto "Hello World" il nome x

```
In [3]: x = "Hello World"
```

Stampiamo x (che è il nome che abbiamo assegnato all'oggetto "Hello World")

```
In [4]: print(x)
```

Hello World

Stampiamo il tipo di x (che è il nome che abbiamo assegnato all'oggetto "Hello World")

```
In [5]: print(type(x))
```

<class 'str'>

Assegniamo al nome y, lo stesso oggetto a cui ho assegnato il nome x

```
In [6]: y = x
```

Stampiamo y

```
In [7]: print(y)
```

Hello World

Posso accedere facilmente a un particolare carattere della stringa

```
In [8]: print(y[1])
```

e

```
In [9]: print(y[1:5])
```

ello

Analizziamo altri tipi. Partiamo dagli interi

Stampiamo l'oggetto 1

```
In [10]: print(1)
```

1

Stabiliamo il tipo dell'oggetto 1

```
In [11]: print(type(1))
```

<class 'int'>

Se assegnasi a l'oggetto 1 un determinato nome (scrivendo ad esempio x = 1) varrebbero tutte le considerazioni fatte in precedenza con le stringhe

Float

Stampiamo l'oggetto 1.2

```
In [12]: print(1.2)
```

1.2

Stabiliamo il tipo dell'oggetto 1.2

```
In [13]: print(type(1.2))  
<class 'float'>
```

Booleani

Stampiamo l'oggetto True

```
In [14]: print(True)  
True
```

Stabiliamo il tipo dell'oggetto "Nicola"

```
In [15]: print(type(True))  
<class 'bool'>
```

None

Stampiamo l'oggetto None

```
In [16]: print(None)  
None
```

Stabiliamo il tipo dell'oggetto "None"

```
In [17]: print(type(None))  
<class 'NoneType'>
```

Tuple

Stampiamo la tupla (1,"a",3,1)

```
In [18]: print((1,"a",3,1))  
(1, 'a', 3, 1)
```

```
In [19]: print(type((1,"a",3,1)))  
<class 'tuple'>
```

Alle tuple posso applicare anche altre funzioni e metodi

```
In [20]: print(len((1,"a",3,1)))  
4
```

```
In [21]: print((1,"a",3,1).count(3))  
1
```

```
In [22]: print((1,"a",3,1).index(3))  
2
```

Inoltre posso accedere ad un elemento specifico della tupla

```
In [23]: print((1,"a",3,1)[0])  
1
```

Tutti gli oggetti visti finora sono immutabili. Cioè non possono essere modificati

Nelle prossime due righe di codice, sto assegnando alla x prima l'oggetto 2 e poi l'oggetto 3. Al termine del codice il nome x sarà assegnato all'oggetto 3. Dire che sto modificando la x non è proprio corretto

```
In [24]: x = 2  
x = 3
```

```
In [25]: print(x)  
3
```

In questo codice assegno alla y lo stesso valore di x, poi alla y assegno un altro oggetto.

```
In [26]: y = x  
y = 4
```

Siccome "non sto modificando la y" la x resterà con il valore vecchio

```
In [27]: print(x)
```

3

```
In [28]: print(y)
```

4

Oggetti mutabili

Le liste sono oggetti mutabili (esistono metodi per modificarli)

Andiamo con ordine, partiamo con la stampa della lista [1,"a",3,1]

```
In [29]: print([1,"a",3,1])
```

[1, 'a', 3, 1]

Vediamo il tipo

```
In [30]: print(type([1,"a",3,1]))
```

<class 'list'>

Anche le liste hanno i metodi count e pos

```
In [31]: print([1,"a",3,1].count(3))
```

1

```
In [32]: print([1,"a",3,1].index(3))
```

2

Inoltre posso accedere ad un elemento specifico della tupla

```
In [33]: print([1,"a",3,1][0])
```

1

Esistono però anche metodi per modificare una lista. In questi casi conviene assegnare un nome alla lista. Partiamo assegnando alla lista [1,"a",3,1] il nome x

```
In [34]: x = [1,"a",3,1]
```

Modifichiamo la lista aggiungendo in fondo il valore 3 con il metodo append

```
In [35]: x.append(3)
```

Stampiamo x

```
In [36]: print(x)
```

[1, 'a', 3, 1, 3]

Attenzione: i metodi visti precedentemente count e index restituivano un intero. Append modifica direttamente la lista, non restituisce niente! Proviamo a lanciare questo codice

```
In [37]: y = x.append(5)
```

Cosa c'è in x e in y?

```
In [38]: print(x)
```

[1, 'a', 3, 1, 3, 5]

```
In [39]: print(y)
```

None

Vediamo il metodo sort per ordinare la lista

```
In [40]: z = [3,5,1]
```

```
In [41]: print(z)
```

[3, 5, 1]

```
In [42]: z.sort()
```

```
In [43]: print(z)
```

```
[1, 3, 5]
```

Se invece proviamo ad ordinare x otteniamo un errore

```
In [44]: x.sort()
```

```
-----
TypeError                                 Traceback (most recent call last)
Cell In[44], line 1
----> 1 x.sort()

TypeError: '<' not supported between instances of 'str' and 'int'
```

Risulta possibile convertire altri tipi in liste tramite la funzione list

```
In [ ]: x = "nicola"
```

```
In [ ]: print(list(x))
```

```
In [ ]: x = "nicola,giovanni,alberto"
```

```
In [ ]: print(x.split(","))
```

Un altro oggetto mutabile: l'insieme

Stampiamo l'insieme {1,"a",3,1}

```
In [45]: print({1,"a",3,1})
```

```
{'a', 3, 1}
```

Facciamo attenzione al fatto che nell'insieme non possono esserci duplicati! Inoltre non è presente un concetto di ordine

Vediamo il tipo

```
In [46]: type({1,"a",3,1})
```

```
Out[46]: set
```

Agli insiemi posso applicare funzioni e metodi, ma non quelli che riguardano la posizione! Attenzione al risultato

```
In [47]: print(len({1,"a",3,1}))
```

```
3
```

I metodi count e index danno errore, perché non sono applicabili a insiemi

```
In [48]: print({1,"a",3,1}.count(1))
```

```
-----
AttributeError                             Traceback (most recent call last)
Cell In[48], line 1
----> 1 print({1,"a",3,1}.count(1))

AttributeError: 'set' object has no attribute 'count'
```

```
In [49]: print({1,"a",3,1}.index(3))
```

```
-----
AttributeError                             Traceback (most recent call last)
Cell In[49], line 1
----> 1 print({1,"a",3,1}.index(3))

AttributeError: 'set' object has no attribute 'index'
```

Allo stesso modo non accedere ad un elemento specifico dell'insieme

```
In [50]: print({1,"a",3,1}[0])
```

```
<>:1: SyntaxWarning: 'set' object is not subscriptable; perhaps you missed a comma?
<>:1: SyntaxWarning: 'set' object is not subscriptable; perhaps you missed a comma?
C:\Users\Ianto\AppData\Local\Temp\ipykernel_20956\567347472.py:1: SyntaxWarning: 'set' object is not subscriptable; perhaps
you missed a comma?
  print({1,"a",3,1}[0])

-----
TypeError                                 Traceback (most recent call last)
Cell In[50], line 1
----> 1 print({1,"a",3,1}[0])

TypeError: 'set' object is not subscriptable
```

E nemmeno ordinare

```
In [51]: x = {1,3,5}
```

```
In [52]: x.sort()
```

```
-----
AttributeError                                Traceback (most recent call last)
Cell In[52], line 1
----> 1 x.sort()

AttributeError: 'set' object has no attribute 'sort'
```

Posso invece aggiungere un elemento all'insieme. Assegniamo anche in questo caso un nome all'insieme

```
In [53]: x = {1,"a",3,1}
```

```
In [54]: x.add(4)
```

```
In [55]: print(x)
```

```
{'a', 3, 4, 1}
```

Proviamo ad aggiungere un elemento già presente

```
In [56]: x.add(1)
```

```
In [57]: print(x)
```

```
{'a', 3, 4, 1}
```

Valgono le stesse considerazioni fatte per le liste

```
In [58]: x = {1,"a",3,1}
y = x
y.add(4)
print(x)
print(y)
```

```
{'a', 3, 4, 1}
```

```
{'a', 3, 4, 1}
```

```
In [59]: x = {1,"a",3,1}
y = x.add(4)
print(x)
print(y)
```

```
{'a', 3, 4, 1}
```

```
None
```

Gli insiemi hanno metodo interessanti non disponibili per le liste

```
In [60]: x = {1,"a",3}
y = {1,2}
```

```
In [61]: print(x.union(y))
```

```
{'a', 1, 3, 2}
```

```
In [62]: print(x.intersection(y))
```

```
{1}
```

```
In [63]: print(x.difference(y))
```

```
{'a', 3}
```

Un altro oggetto mutabile: il dizionario

Stampiamo il dizionario {"nome":"Nicola","eta":35}

```
In [64]: print({"nome":"Nicola","eta":35})
```

```
{'nome': 'Nicola', 'eta': 35}
```

I dizionari sono composti da coppie di chiave-valore. Nel dizionario non possono esserci due elementi con la stessa chiave. Se provo a creare un dizionario con la stessa chiave ripetuta due volte, sarà presa l'ultima coppia.

```
In [65]: print({"nome":"Nicola","eta":35, "nome":"Giovanni"})
```

```
{'nome': 'Giovanni', 'eta': 35}
```

Dalla versione Python 3.7 i dizionari sono ordinati, nel senso che le chiavi seguono l'ordine di inserimento

Vediamo il tipo

```
In [66]: type({"nome": "Nicola", "eta": 35})
```

```
Out[66]: dict
```

Agli insiemi posso applicare funzioni e metodi. Attenzione al risultato

```
In [67]: print(len({"nome": "Nicola", "eta": 35, "nome": "Giovanni"}))
```

```
2
```

Visualizziamo l'elenco di chiavi, valori e coppie chiave-valore

```
In [68]: print({"nome": "Nicola", "eta": 35}.keys())
```

```
dict_keys(['nome', 'eta'])
```

```
In [69]: print({"nome": "Nicola", "eta": 35}.values())
```

```
dict_values(['Nicola', 35])
```

```
In [70]: print({"nome": "Nicola", "eta": 35}.items())
```

```
dict_items([('nome', 'Nicola'), ('eta', 35)])
```

I metodo count e index danno errore, perché non sono applicabili a insiemi

```
In [71]: print({"nome": "Nicola", "eta": 35, "nome": "Giovanni"}.count("nome"))
```

```
-----
AttributeError                                Traceback (most recent call last)
Cell In[71], line 1
----> 1 print({"nome": "Nicola", "eta": 35, "nome": "Giovanni"}.count("nome"))

AttributeError: 'dict' object has no attribute 'count'
```

```
In [72]: print({"nome": "Nicola", "eta": 35, "nome": "Giovanni"}.index(3))
```

```
-----
AttributeError                                Traceback (most recent call last)
Cell In[72], line 1
----> 1 print({"nome": "Nicola", "eta": 35, "nome": "Giovanni"}.index(3))

AttributeError: 'dict' object has no attribute 'index'
```

Risulta facile accedere al valore di una determinata chiave

```
In [73]: x = {"nome": "Nicola", "eta": 35}
```

```
In [74]: print(x["nome"])
```

```
Nicola
```

Invece, analogamente agli insiemi, non posso accedere ad un elemento specifico del dizionario

```
In [75]: print({"nome": "Nicola", "eta": 35}[0])
```

```
-----
KeyError                                      Traceback (most recent call last)
Cell In[75], line 1
----> 1 print({"nome": "Nicola", "eta": 35}[0])

KeyError: 0
```

Tuttavia da Python 3.7 (in cui è mantenuto l'ordine di inserimento) potrei arrivare al risultato utilizzando le liste

```
In [76]: x = {"nome": "Nicola", "eta": 35}
```

```
In [77]: chiavi = list(x.keys())
print(chiavi)
```

```
['nome', 'eta']
```

```
In [78]: valori = list(x.values())
print(valori)
```

```
['Nicola', 35]
```

```
In [79]: print({chiavi[0]:valori[0]})
```

```
{'nome': 'Nicola'}
```

Posso invece aggiungere una coppia al dizionario.

```
In [80]: x = {"nome": "Nicola", "eta": 35}
```

```
In [81]: x["cognome"] = "Iantomasi"
```

```
In [82]: print(x)
```

```
{'nome': 'Nicola', 'eta': 35, 'cognome': 'Iantomasi'}
```

Proviamo ad aggiungere un elemento già presente

```
In [83]: x["cognome"] = "Rossi"
```

Sovrascriverò il valore

```
In [84]: print(x)
```

```
{'nome': 'Nicola', 'eta': 35, 'cognome': 'Rossi'}
```