python

context manager

context manager

La dichiarazione with definisce un blocco di codice con metodi definiti dal context manager (oggetto che definisce il contesto di runtime da stabilire quando si esegue il blocco di codice with). Gli usi tipici dei context manager includono il salvataggio e il ripristino di vari tipi di stato globale, il blocco e lo sblocco delle risorse, la chiusura di file aperti, ecc.

Grazie a with il file si chiude automaticamente dopo l'esecuzione del blocco di codice.

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

with garantisce la chiusura del file a dispetto di possibili errori nel blocco di codice.

Se si presenta un'eccezione prima della fine del blocco, Python chiuderà il file per poi sollevare l'eccezione.

moduli

python - moduli -

I moduli sono librerie di codice che si possono importare nel proprio script o in un'istanza interattiva dell'interprete.
Per importare un modulo si ricorre alla keyword import.

```
>>> import math
>>> math.factorial(5)
120
```

→ importa l'intero modulo math

```
>>> from math import factorial
>>> factorial(5)
120
```

→ importa una funzione del modulo math

```
>>> from string import ascii_lowercase
>>> ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
```

→ Importa un attributo del modulo string

python - moduli -

Un metodo è una funzione che fa parte di una classe ed è utilizzabile su un oggetto della classe stessa. Una funzione è un oggetto istanza della classe function.

```
>>> def f():
...     pass
...
>>> type(f)
<class 'function'>
>>>
>>> type(print)
<class 'builtin_function_or_method'>
>>>
>>> type(list().append)
<class 'builtin_function_or_method'>
>>>
```

definisce una funzione

type class 'function'

funzione integrata print()

Metodo integrato della classe list (funzione legata all'oggetto)

SINTASSI

Funzione:
- funzione()

Metodo:
- oggetto.metodo()

python - funzioni/metodi -

Una brevissima lista dei moduli più diffusi in python.

pillow→ manipolazione immagini;

matplotlib → utilizzata per creare grafici matematici bi-dimensionali;

numpy → popolarissimo modulo per l'elaborazione di array, matrici, etc.;

opencv → open source computer vision, modulo python per computer vision;

requests → pane quotidiano di chi sviluppa orientato al web, permette di gestire richieste HTTP;

sqlalchemy → modulo per astrazione database;

beautifulsoup → modulo per parsing di documenti HTML/XML;

pandas → il pane quotidiano dei data scientist;

… e tanti altri: cirq, pytorch, delorean, theano, tensorflow, keras, etc.

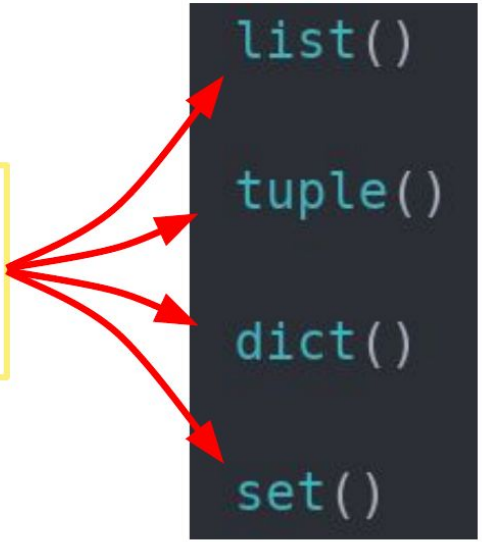
strutture dati

python - strutture dati -

Le principali strutture dati in Python sono:

- Liste
- Tuple
- Dizionari
- Set

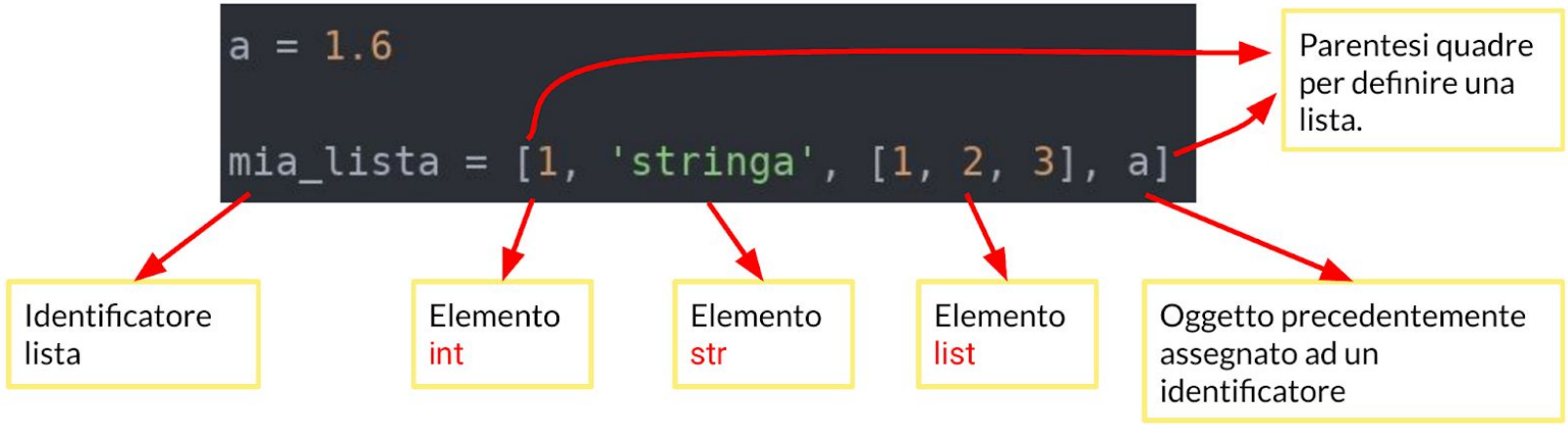
Costruttori di lista, tupla, dizionario e set



Esistono inoltre, importabili da moduli, molte altre strutture dati

python - liste -

La struttura dati più versatile è la lista, che può essere scritta come un elenco di oggetti (elementi),separati da virgola, tra parentesi quadre (o con costruttore list()) e può contenere oggetti di diverso tipo.



python - liste -

Le liste sono collezioni mutabili di oggetti ordinati selezionabili attraverso il loro "index". Il numero di index parte da 0.

accesso ad un elemento:

```
>>> l = ['a', 'b', 'c']
>>>
>>> l[0]
'a'
```

sostituzione di un elemento:

```
>>> l = ['a', 'b', 'c']
>>> l[1] = 'f'
>>> l
['a', 'f', 'c']
```

cancellazione di un elemento:

```
>>> l = ['a', 'b', 'c']
>>> del l[2]
>>> l
['a', 'b']
```

Sulle liste (e sulle stringhe) si può usare anche lo "slicing" (to slice = affettare) accedendo ad una "fetta" di elementi attraverso il loro index.

```
>>> l = ['a', 'b', 'c']
>>> l[0:2]
['a', 'b']
```

python - liste/funzioni -

Python ha una serie di funzioni e metodi integrati che si possono usare sulle liste. Tra i più usati ci sono:

```
>>> l = ['a', 'b', 'c']
```

.append()

```
>>> l.append('d')
>>> l
['a', 'b', 'c', 'd']
```

.count()

```
>>> l.count('a')
1
```

.index()

```
>>> l.index('c')
2
```

.insert()

```
>>> l.insert(3, 'd')
>>> l
['a', 'b', 'c', 'd', 'd']
```

.len()

```
>>> len(l)
5
```

.pop()

```
>>> l.pop(2)
'c'
>>> l
['a', 'b', 'd', 'd']
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

python - tuple -

Le tuple sono sequenze, proprio come le liste. La differenza principale tra le tuple e le liste è che le tuple non possono essere modificate. Per istanziare una tupla si usano parentesi tonde o il costruttore tuple().

#1 Istanza di tupla

#2 check tipo di oggetto

#3 accesso elemento con index 0

#4 Errore generato da Python (le tuple **NON** sono modificabili!)

```
>>> t = ('a', 'b', 'c')  # 1
>>>
>>> type(t)  # 2
<class 'tuple'>
>>>
>>> t[0]  # 3
'a'
>>>
>>> del t[0]  # 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

python - tuple/funzioni -

Un set è una collezione non ordinata e non indicizzata di elementi unici e immutabili. Per istanziare un set si usano parentesi graffe o il costruttore set(). L'idea del set è quella di ricreare le funzionalità tipiche degli insiemi matematici.
[N.B. anche i dizionari utilizzano le parentesi graffe. Per creare un dizionario vuoto sarà sufficiente utilizzare {}. Per creare un set vuoto si dovrà ricorrere invece al costruttore set()]

Utilizzo del costruttore set() per ottenere una collezione di oggetti unici.

```
>>> l = [1, 1, 2, 3, 4, 3, 3, 3, 4]
>>>
>>> setted_list = set(l)
>>> setted_list
{1, 2, 3, 4}
```

Istanza di un set con due elementi uguali → il secondo "4" non viene inserito.

```
>>> s = {1, 2, 3, 4, 4}
>>> s
{1, 2, 3, 4}
```

i set **NON** sono indicizzati!

```
>>> s = {1, 2, 3, 4}
>>> s[0]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'set' object is not subscriptable
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

python - set/funzioni metodi -

I set hanno a disposizione diverse funzioni (e metodi) finalizzate a controllare la struttura, modificarla o confrontare più strutture.

#1 istanza set

#2 tentativo di aggiungere un elemento modificabile con metodo .add() e relativo errore generato da Python

#3 Aggiunta di un elemento NON modificabile (str)

```
>>> s = {'a', 'b'}  # 1
>>> s
{'b', 'a'}
>>>
>>>
>>> l = [1, 2, 3]
>>>
>>> s.add(l)  # 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> s
{'b', 'a'}
>>> s.add('c')  # 3
>>>
>>> s
{'b', 'a', 'c'}
```

I set sono sequenze "**mutabili**" ma possono contenere **SOLO** elementi immutabili (**hashable**).

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

python - set/funzioni metodi -

I set hanno a disposizione le stesse funzioni usate anche sulle liste (min(), max(), len()) e molti metodi. Per la modifica strutturale di un set, ci sono:

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

.add()

```
>>> s.add(4)
>>> s
{1, 2, 3, 4}
```

.remove()

```
>>> s.remove(3)
>>> s
{1, 2, 4}
>>> s.remove(3)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 3
```

.discard()

```
>>> s.discard(2)
>>> s
{1, 3}
>>> s.discard(2)
>>> s
{1, 3}
```

.remove() e .discard() svolgono lo stesso compito ma .remove() solleva un errore se non trova l'elemento

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

python - set/funzioni metodi -

Altri metodi utili per confrontare due o più set:

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

python - set/funzioni metodi -

Altri metodi utili per confrontare due o più set:

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

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

python - dizionari -

I dizionari si trovano talvolta in altri linguaggi come "memorie associative" o "array associativi". A differenza delle sequenze (str, list, tuple), che sono indicizzate con numeri, i dizionari sono indicizzati da key, che possono essere di qualsiasi tipo immutabile, accoppiati con value; stringhe e numeri possono sempre essere chiavi. Si possono istanziare con parentesi graffe o con il costruttore dict().

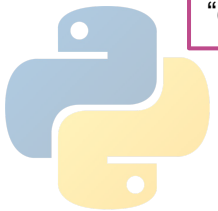
SINTASSI dictionary = {key: value, key: value[...]}

#1 istanza classe 'dict'

#2 accesso all'elemento con "chiava" 'a';

#3 accesso all'elemento con "chiave" 'b' con metodo .get()

```
>>> d = {'a': 1, 'b': 2, 'c': 3}  # 1
>>>
>>> d['a']  # 2
1
>>> d.get('b')  # 3
2
```



UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

python - dizionari/funzioni metodi -

I dizionari sono strutture dati mutabili e possono contenere QUALSIASI oggetto come value. Per modificare la struttura di un dizionario si può ricorrere a diversi strumenti.

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

Inserimento di un nuovo elemento:
- chiave → 'd'
- valore → lista [1, 2, 3]

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

Ottenibile anche con: d.update({ 'd': [1, 2, 3] })

Cancellazione di un elemento con keyword del (utilizzabile con qualsiasi elemento di sequenze mutabili o identificatore).

```
>>> del d['c']
>>> d
{'a': 1, 'b': 2, 'd': [1, 2, 3]}
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

python - dizionari/funzioni metodi -

Altri metodi:

.fromkeys(keys, [value]) → ritorna un dizionario con le chiavi e i valori (opzionale) specificati (se non viene specificato nessun valore, None viene assegnato di default)

```
>>> k = ['a', 'b', 'c']
>>>
>>> dict.fromkeys(k)
{'a': None, 'b': None, 'c': None}
```

.setdefault(keys, [value]) → ritorna il valore (secondo argument) inserito. Se la chiave è già esistente, non avvengono modifiche. Se non esiste, vengono inseriti chiave e valore (default=None)

```
>>> d = {'a': 1, 'b': 2, 'c': 3}
>>>
>>> d.setdefault('d', 4)
4
>>> d
{'a': 1, 'b': 2, 'c': 3, 'd': 4}
```

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

python - shallow copy / deep copy -

Esistono due tipi di copia: la copia **shallow** (“superficiale”) e la copia **deep** (“profonda”) [utilizzabile importando il modulo “copy”]. La differenza tra shallow copy e deep copy è rilevante solo per oggetti con altri oggetti annidati o istanze di classe. Ma prima, ecco come Python gestisce le “variabili”. [id() è una built-in function che ritorna l’indirizzo dell’oggetto nella memoria]

```
>>> # esempio 1
>>> lst_1 = [1, 2, 3]
>>> lst_2 = lst_1
>>> lst_1 == lst_2
True
>>> id(lst_1) == id(lst_2)
True
>>> id(lst_1); id(lst_2)
140080615844224
140080615844224
```

esempio 1
- lista assegnata a ’lst_1’
- ‘lst_1’ assegnata a ‘lst_2’
- ‘lst_1’ e ‘lst_2’ sono due identificatori che accedono allo stesso oggetto

esempio 2
- lista assegnata a ’lst_1’
- lista assegnata a ‘lst_2’
- ‘lst_1’ e ‘lst_2’ sono due identificatori che accedono a oggetti differenti

```
>>> # esempio 2
>>> lst_1 = [1, 2, 3]
>>> lst_2 = [1, 2, 3]
>>> lst_1 == lst_2
True
>>> id(lst_1) == id(lst_2)
False
>>> id(lst_1); id(lst_2)
140080615706944
140080615736320
```