Python Collections — A Quick Comparison

Sumedh Waghmare Follow
Jun 28, 2020 · 3 min read

Collections are fundamental ways to store and organize data. There are four basic types of collections available in Python — `tuples`, `lists`, `dictionaries` and `sets`. To become proficient in Python programming, developers should know basic differences between these data structures so they can choose the appropriate type of collection for a given job on hand.

Below table shows side-by-side comparison of different Python collections:

	Tuple	List	Dictionary	Set
Eample	('Book 1', 12.99)	['apple', 'banana', 'orange']	{'name': 'Joe', 'age': 10}	{10, 20, 12}
Mutable?	Immutable	Mutable	Mutable	Mutable
Ordered?	Ordered	Ordered	Preserves order since Python 3.7	Unordered
Iterable?	Yes (takes linear time)	Yes (takes linear time)	Yes (constant time)	Yes (constant time)
Use case	Immutable data	Data that needs to change	Key/Value pairs	Unique items

Common Methods

It is also quite important to remember some of the common methods available for each collection. This will help in certain circumstances such as interviewing process.

Collection	Common Methods

Collection	Common Methods
Tuple	my_tuple[0], len(my_tuple), .count(), .index()
List	my_list[0], len(my_list), .count(), .index(), .append(), .insert(), .pop(), .remove(), .reverse(), .sort()
Dictionary	d['key'], .keys(), .values(), .items(), .pop(), del d['key']
Set	len(my_set), .add(), .update(), .remove(), .union(), .intersection() and other set operation methods

Common Methods — Tuple

```
t = ('Book 1', 12.99)

# Get the value element for the supplied index, returns IndexError
for invalid index
print(t[1])   # 12.99

# Get length of a tuple
print(len(t))  # 2

# Get index of a given value, if the supplied value is not found,
returns value error
print(t.index('Book 1')) # 0

# Count number of items a given value is present in the collection
print(t.count(12))  # 0
```

Common Methods — List

```
my_list = ['apple', 'banana', 'orange']

# get element present in the specified index
print(my_list[1])  # banana

# find length of the list
print(len(my_list))  # 3

# get index of specified value
print(my_list.index('orange')) # 2

# count number of items times a specified value is present in the
collection
print(my_list.count('banana'))  # 1
```

```
# adds item to the end of the list
my_list.append('pear')

# adds item in the specified index
my_list.insert(1, 'grapes')

# removes and returns item from the end of the list
print(my_list.pop())   # pear

# removes specified value from the list
my_list.remove('banana')

# reverses the list
my_list.reverse()
print(my_list)   # ['orange', 'grapes', 'apple']

# sorts the list
my_list.sort()
print(my_list)   # ['apple', 'grapes', 'orange']
```

Common Methods — Dictionary

```
d = {'name': 'Joe', 'age': 10}

# returns value of specified key
print(d['name'])  # Joe

# returns length of a dictionary
print(len(d))  # 2

# returns iterable list of keys
print(d.keys())  # dict_keys(['name', 'age'])

# returns iterable list of values
print(d.values()) # dict_values(['Joe', 10])

# returns iterable list of key / value pairs
print(d.items()) # dict_items([('name', 'Joe'), ('age', 10)])

# deletes given key from the dictionary
del d['age']
print(d)   # {'name': 'Joe'}
```

Common Methods — Set

```
s1 = {10, 20, 12}
s2 = {100, 21, 12, 35, 40}

# finds length of a given set
print(len(s2))  # 5

# adds an element to the set, ignored if the element already exists
s1.add(21)

# updates set with elements of another set
s1.update({33, 30})

# removes specified element from the set, throws KeyError if given
element is not in the set
s1.remove(33)

# returns a new set containing all elements from both sets
print(s1.union(s2))  # {33, 35, 100, 40, 10, 12, 20, 21, 30}

# returns a new set containing common elements from both sets
print(s1.intersection(s2))   # {12, 21}
```

Python3 Python Programming Python Python Collections Python Data Science