Introductory Python : Data Structures in Python

Assoc. Prof. Dr. Bora Canbula
Manisa Celal Bayar University

Lists in Python:

Ordered and mutable sequence of values indexed by integers

Initializing

```
a_list = [] ## empty
a_list = list() ## empty
a_list = [3, 4, 5, 6, 7] ## filled
```

Finding the index of an item

```
a_list.index(5) ## 2 (the first occurence)
```

Accessing the items

```
a_list[0] ## 3
a_list[1] ## 4
a_list[-1] ## 7
a_list[-2] ## 6
a_list[2:] ## [5, 6, 7]
a_list[:2] ## [3, 4]
a_list[1:4] ## [4, 5, 6]
a_list[0:4:2] ## [3, 5]
a_list[4:1:-1] ## [7, 6, 5]
```

Adding a new item

```
a_list.append(9) ## [3, 4, 5, 6, 7, 9]
a_list.insert(2, 8) ## [3, 4, 8, 5, 6, 7, 9]
```

Update an item

```
a_list[2] = 1 ## [3, 4, 1, 5, 6, 7, 9]
```

Remove the list or just an item

```
a_list.pop() ## last item
a_list.pop(2) ## with index
del a_list[2] ## with index
a_list.remove(5) ## first occurence of 5
a_list.clear() ## returns an empty list
del a_list ## removes the list completely
```

Extend a list with another list

```
list_1 = [4, 2]
list_2 = [1, 3]
list_1.extend(list_2) ## [4, 2, 1, 3]
```

Reversing and sorting

```
list_1.reverse() ## [3, 1, 2, 4]
list_1.sort() ## [1, 2, 3, 4]
```

Counting the items

```
list_1.count(4) ## 1
list_1.count(5) ## 0
```

Copying a list

```
list_1 = [3, 4, 5, 6, 7]
list_2 = list_1
list_3 = list_1.copy()
list_1.append(1)
list_2 ## [3, 4, 5, 6, 7, 1]
list_3 ## [3, 4, 5, 6, 7]
```

Sets in Python:

Unordered and mutable collection of values with no duplicate elements. They support mathematical operations like union, intersection, difference and symmetric difference

Initializing

```
a_set = set() ## empty
a_set = {3, 4, 5, 6, 7} ## filled
```

No duplicate values

```
a_set = {3, 3, 3, 4, 4} ## {3, 4}
```

Adding and updating the items

```
a_set.add(5) ## {3, 4, 5}
set_1 = {1, 3, 5}
set_2 = {5, 7, 9}
set_1.update(set_2) ## {1, 3, 5, 7, 9}
```

Removing the items

```
a_set.pop() ## removes an item and returns it
a_set.remove(3) ## removes the item
a_set.discard(3) ## removes the item
```

If item does not exist in set,
remove() raises an error, discard() does not

```
a_set.clear() ## returns an empty set
del a_set ## removes the set completely
```

Mathematical operations

```
set_1 = {1, 2, 3, 5}
set_2 = {1, 2, 4, 6}
```

Union of two sets

```
set_1.union(set_2) ## {1, 2, 3, 4, 5, 6}
set_1 | set_2 ## {1, 2, 3, 4, 5, 6}
```

Intersection of two sets

```
set_1.intersection(set_2) ## {1, 2}
set_1 & set_2 ## {1, 2}
```

Difference between two sets

```
set_1.difference(set_2) ## {3, 5}
set_2.difference(set_1) ## {4, 6}
set_1 - set_2 ## {3, 5}
set_2 - set_1 ## {4, 6}
```

Symmetric difference between two sets

```
set_1.symmetric_difference(set_2) ## {3,4,5,6}
set_1 ^ set_2 ## {3, 4, 5, 6}
```

Update sets with mathematical operations

```
set_1.intersection_update(set_2) ## {1, 2}
set_1.difference_update(set_2) ## {3, 5}
set_1.symmetric_difference_update(set_2)
## {3, 4, 5, 6}
```

Copying a set

Same as lists

Dictionaries in Python:

Unordered and mutable set of key-value pairs

Initializing

```
a_dict = {} ## empty
a_dict = dict() ## empty
a_dict = {"name":"Bora"} ## filled
```

Accessing the items

```
a_dict["name"] ## "Bora"
a_dict.get("name") ## "Bora"
```

If the key does not exist in dictionary,
index notation raises an error, get() method does not

Accessing the items with views

```
other_dict = {"a": 3, "b": 5, "c": 7}
other_dict.keys() ## ['a', 'b', 'c']
other_dict.values() ## [3, 5, 7]
other_dict.items()
## [('a', 3), ('b', 5), ('c', 7)]
```

Adding a new item

```
a_dict["city"] = "Manisa"
a_dict["age"] = 37
## {"name":"Bora", "city":"Manisa", "age":37}
```

Update an item

```
a_dict["age"] = 38
## {"name":"Bora", "city":"Manisa", "age":38}
other_dict = {"age":39}
a_dict.update(other_dict)
## {"name":"Bora", "city":"Manisa", "age":39}
```

Removing the items

```
a_dict.popitem() ## last inserted item
a_dict.pop("city") ## with a key
a_dict.clear() ## returns an empty dictionary
del a_dict ## removes the dict completely
```

Initialize a dictionary with fromkeys

```
a_list = ['a', 'b', 'c']
a_dict = dict.fromkeys(a_list)
## {'a': None, 'b': None, 'c': None}
a_dict = dict.fromkeys(a_list, 0)
## {'a': 0, 'b': 0, 'c': 0}
a_tuple = (3, 'name', 7)
a_dict = dict.fromkeys(a_tuple, True)
## {3: True, 'name': True, 7: True}
a_set = {0, 1, 2}
a_dict = dict.fromkeys(a_set, False)
## {0: False, 1: False, 2: False}
```

Tuples in Python:

Ordered and immutable sequence of values indexed by integers

Initializing

```
a_tuple = () ## empty
a_tuple = tuple() ## empty
a_tuple = (3, 4, 5, 6, 7) ## filled
```

Finding the index of an item

```
a_tuple.index(5) ## 2 (the first occurence)
```

Accessing the items

Same index and slicing notation as lists

Adding, updating, and removing the items

Not allowed because tuples are immutable

Sorting

Tuples have no sort() method since they are immutable

```
sorted(a_tuple) ## returns a sorted list
```

Counting the items

```
a_tuple.count(7) ## 1
a_tuple.count(9) ## 0
```

Some iteration examples:

```
a_list = [3, 5, 7]
a_tuple = (4, 6, 8)
a_set = {1, 4, 7}
a_dict = {"a":1, "b":2, "c":3}
```

For ordered sequences

```
for i in range(len(a_list)):
    print(a_list[i])
for i, x in enumerate(a_tuple):
    print(i, x)
```

For ordered or unordered sequences

```
for a in a_set:
    print(a)
```

Only for dictionaries

```
for k in a_dict.keys():
    print(k)
for v in a_dict.values():
    print(v)
for k,v in zip(a_dict.keys(),a_dict.values()):
    print(k, v)
for k, v in a_dict.items():
    print(k, v)
```