Python Basics

Course 4- Coursera Week 2

Objectives

- Understand Python List and Tuple
- Understand Python Dictionary
- Understand Python Sets

Python Lists

List type is another sequence type defined by the list class of python. List allows you add, delete or process elements in very simple ways. Lists are very similar to arrays.

Creating list in python

You can create lists using the following syntax.

```
>>> l = [1, 2, 3, 4]
```

Here, each element in the list is separated by comma and enclosed by a pair of square brackets ([]). Elements in the list can be of same type or different type. For e.g:

```
l2 = ["this is a string", 12]
```

Other ways of creating list.

TRY IT

1	list1 = list() # *Create an empty list*
2	list2 = list([22, 31, 61]) # *Create a list with elements 22, 31, 61*
3	list3 = list(["tom", "jerry", "spyke"]) # *Create a list with strings*
4	list5 = list("python") # *Create a list with characters p, y, t, h, o, n*

Lists are mutable.

Common List Operations

Method name	Description
x in s	True if element x is in sequence s, False otherwise
x not in s	True if element x is not in sequence s, False otherwise
s1 + s2	Concatenates two sequences s1 and s2
s * n , n * s	n copies of sequence s concatenated
s[i]	ith element in sequence s.
len(s)	Length of sequences, i.e. the number of elements ins`.
min(s)	Smallest element in sequence s.
max(s)	Largest element in sequence s.
sum(s)	Sum of all numbers in sequence s.

List slicing

Slice operator **([start:end])** allows to fetch sublist from the list. It works similar to string.

TRY IT

1	>>> list = [11,33,44,66,788,1]
2	>>> list[0:5] # this will return list starting from index 0 to index 4
3	[11,33,44,66,788]

Note:

If **start >= end**, **list[start : end]** will return an empty list. If end specifies a position which is beyond the **end** of the list, Python will use the length of the list for **end** instead.

+ AND * OPERATORS IN LIST

The + operator joins the two lists. TRY IT

```
1  >>> list1 = [11, 33]
2  >>> list2 = [1, 9]
3  >>> list3 = list1 + list2
4  >>> list3
5  [11, 33, 1, 9]
```

The * operator joins the two list.

```
1  >>> list4 = [1, 2, 3, 4]
2  >>> list5 = list4 * 3
3  >>> list5
4  [1, 2, 3, 4, 1, 2, 3, 4, 1, 2, 3, 4]
```

Commonly used list methods with return type

Method name	Description
append(x:object):None	Adds an element x to the end of the list and returns None.
count(x:object):int	Returns the number of times element x appears in the list.
extend(l:list):None	Appends all the elements in l to the list and returns None.
index(x: object):int	Returns the index of the first occurrence of element x in the list
insert(index: int, x: object):None	Inserts an element x at a given index. Note that the first element in the list has index 0 and returns None.
remove(x:object):None	Removes the first occurrence of element x from the list and returns None
reverse():None	Reverse the list and returns None
sort(): None	Sorts the elements in the list in ascending order and returns None.
pop(i): object	Removes the element at the given position and returns it. The parameter i is optional. If it is not specified, pop() removes and returns the last element in the list.

Python Tuples

In Python Tuples are very similar to lists, but once a tuple is created, you cannot add, delete, replace, or reorder elements.

Note: Tuples are immutable.

Creating a tuple

TRY IT

1	>>> t1 = () # creates an empty tuple with no data
2	>>> t2 = (11,22,33)
3	>>> t3 = tuple([1,2,3,4,4]) # tuple from array
4	>>> t4 = tuple("abc") # tuple from string

Tuples functions

Functions like **max(), min(), len()** and **sum()** can also be used with tuples.

TRY IT

1	>>> t1 = (1, 12, 55, 12, 81)
2	>>> min(t1)
3	1
4	>>> max(t1)
5	81
6	>>> sum(t1)
7	>>> t3 = tuple([1,2,3,4,4]) *# tuple from array*
8	>>> t4 = tuple("abc") # tuple from string

Slicing tuples

Slicing operators works same in tuples as in list and string.

TRY IT

1	**>>> t = (11,22,33,44,55)**
2	**>>> t[0:2]**
3	**(11,22)**

IN AND NOT IN OPERATOR

You can use **in** and **not in** operators to check existence of item in tuples as follows.

TRY IT

1	>>> t = (11,22,33,44,55)
2	>>> 22 in t
3	True
4	>>> 22 not in t
5	False

Python Dictionaries

Dictionary is a python data type that is used to store key-value pairs. It enables you to quickly retrieve, add, remove, modify, values using a key. Dictionary is very similar to what we call associative array or hash on other languages.

Note:

Dictionaries are mutable.

Creating a Dictionary

Dictionaries can be created using a pair of curly braces **(`{}`).** Each item in the dictionary consists of a key, followed by a colon, which is followed by a value. And each item is separated using commas **(`,`).** Let's take an example.

TRY IT

```
>>> dict_emp = {} # this will create an empty dictionary
```

Retrieving, modifying and adding elements in the dictionary

To get an item from the dictionary, use the following syntax:

```
>>> dictionary_name['key']
```

TRY IT

```
1  friends = {
2  'tom' : '111-222-333',
3  'jerry' : '666-33-111'
4  }
5  >>> friends['tom']
6  '111-222-333'
```

Deleting Items from the dictionary

If the key is found the item will be deleted, otherwise a **KeyError** exception will be thrown.

```
>>> del dictionary_name['key']
```

TRY IT

1	`>>> del friends['bob']`
2	`>>> friends`
3	`{'tom': '111-222-333', 'jerry': '666-33-111'}`

Looping items in the dictionary

You can use for loop to traverse elements in the dictionary.

TRY IT

```
>>> friends = {
... 'tom' : '111-222-333',
... 'jerry' : '666-33-111'
...}
>>> for key in friends:
...     print(key, ":", friends[key])
...
tom : 111-222-333
jerry : 666-33-111
>>>
```

Find the length of the dictionary

You can use the `len()` function to find the length of the dictionary.

```
>>> len(friends)
 2
```

IN OR NOT IN OPERATORS

in and **not in** operators to check whether key exists in the dictionary.

TRY IT

1	>>> 'tom' **in** friends
2	True
3	>>> 'tom' **not in** friends
4	False

Equality Tests in dictionary

The == and != operators tells whether dictionary contains the same items not.

TRY IT

1	>>> d1 = {"mike":41, "bob":3}
2	>>> d2 = {"bob":3, "mike":41}
3	>>> d1 == d2
4	True
5	>>> d1 != d2
6	False
7	>>>

Dictionary methods

Methods	Description
popitem()	
clear()	Delete everything from a dictionary
keys()	Return keys in the dictionary as tuples
values()	Return values in dictionary as tuples
get(key)	Return value of key, if key is not found it returns **None**, instead of throwing **KeyError** exception
pop(key)	Remove the item from the dictionary, if the key is not found **KeyError** will be thrown

Python sets

Sets are lists with no duplicate entries. Let's say you want to collect a list of words used in a paragraph:

Sets are a powerful tool in Python since they have the ability to calculate differences and intersections between other sets. For example, say you have a list of participants in events A and B:

TRY IT

```
1  a = set(["Jake", "John", "Eric"])
2  print(a)
3  b = set(["John", "Jill"])
4  print(b)
```

CONT.

To find out which members **attended both events**, you may use the "intersection" method:

TRY IT

```
1  a = set(["Jake", "John", "Eric"])
2  b = set(["John", "Jill"])
3  print(a.intersection(b))
4  print(b.intersection(a))
```

CONT.

To find out which members **attended only one of the events**, use the "symmetric_difference" method:

TRY IT

```
1  a = set(["Jake", "John", "Eric"])
2  b = set(["John", "Jill"])
3  print(a.symmetric_difference(b))
4  print(b.symmetric_difference(a))
```

CONT.

To find out which members **attended only one event and not the other**, use the "difference" method:

TRY IT

```
1  a = set(["Jake", "John", "Eric"])
2  b = set(["John", "Jill"])
3  print(a.difference(b))
4  print(b.difference(a))
```

CONT.

To receive a **list of all participants**, use the "union" method:

TRY IT

```
1  a = set(["Jake", "John", "Eric"])
2  b = set(["John", "Jill"])
3  print(a.union(b))
```