Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 7

Fall 2017

Collections

- **A Collection Groups Similar Things**

  List: ordered

  Set: unordered, no duplicates

  Tuple: unmodifiable list

  Dictionary: maps from keys to values

Lists

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is **that items in a list need not be of the same type**.

Creating a list:

```
my_list = [1, 2, 3, 4, 5]
```

Splitting a string to create a list:

```
s = 'spam-spam-spam'
delimiter = '-'
s.split(delimiter)
```

split()

Output: `['spam', 'spam', 'spam']`

Lists

Making a list of chars in a string:

```
s = 'spam'
t = list(s)
print(t)
```

```
Output:
['s', 'p', 'a', 'm']
```

Joining elements of a list into a string:

```
t = ['programming', 'is', 'fun']
delimiter = ' '
delimiter.join(t)
```

join()

```
Output: 'programming is fun'
```

Accessing Values in Lists by Index:

```
list1 = [1, 2, 3, 4, 5]
print ("list1[0]: ", list1[0])
print ("list1[1:5]: ", list2[1:5])
```

```
Output:
list1[0]: 1
list1[1:5]:  [2, 3, 4, 5]
```

Updating Lists:

```
list = [1, 2, 3, 4, 5]
print ("Value available at index 2: ",list[2])
list[2] = 6
print ("New value available at index 2: ", list[2])
```

```
Output:
Value available at index 2:  3
New value available at index 2:  6
```

Deleting List Elements

```
list = [1, 2, 3, 4, 5]
print(list)
del list[2]
print ("After deleting value at index 2: ",list)
```

```
Output:
[1, 2, 3, 4, 5]
After deleting value at index 2:  [1, 2, 4, 5]
```

Basic List Operations

Python Expression	Results	Description
len([1, 2, 3])	3	Length
[1, 2, 3] + [4, 5, 6]	[1, 2, 3, 4, 5, 6]	Concatenation
['Hi!'] * 4	['Hi!', 'Hi!', 'Hi!', 'Hi!']	Repetition
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

List Functions & Methods:

len (list) : Gives the total length of the list.

Example:

```
list = [1, 2, 3, 4, 5]
print ('length of the list is',len(list))
```

```
Output:
length of the list is 5
```

max(list): Returns the item from the list with the maximum value.

Example:

```
list=[456, 700, 200]
print ("Max value element:", max(list))
```

```
Output:
Max value element: 700
```

min(list): Returns the item from the list with the minimum value.

Example:

```
list=[456, 700, 200]
print ("Min value element:", min(list))
```

```
Output:
Min value element: 200
```

list.append(obj): Appends object obj to list

Example:

```
list = [123, 'xyz', 'zara', 'abc']
list.append(2009)
print ("Updated List: ", list)
```

```
Output:
Updated List:  [123, 'xyz', 'zara', 'abc', 2009]
```

list.count(obj): Returns the count of how many times obj occurs in a list

Example:

```
aList = [123, 'xyz', 'zara', 'abc', 123]
print ("Count for 123: ", aList.count(123))
print ("Count for zara: ", aList.count('zara'))
```

```
Output:
Count for 123:  2
Count for zara:  1
```

list.extend(seq) : Appends the contents of seq to list

Example:

```
aList = [123, 'xyz', 'zara']
bList = [2009, 'manni']
aList.extend(bList)
print ("Extended List: ", aList )
```

```
Output:
Extended List:  [123, 'xyz', 'zara', 2009, 'manni']
```

list.index(obj): Returns the lowest index of obj in the list

Example:

```
list=[456, 700, 200]
print ("Index of 700: ", list.index(700) )
```

```
Output:
Index of 700:  1
```

list.insert(index, obj): Inserts object obj into the list at offset index

Example:

```
aList = [123, 'xyz', 'zara', 'abc']
aList.insert(3, 2009)
print ("Final List: ", aList)
```

```
Output:
Final List:  [123, 'xyz', 'zara', 2009, 'abc']
```

list.pop(obj=list[-1]): Removes and returns the last obj from list

Example:

```
aList = [123, 'xyz', 'zara', 'abc']
print ("A List: ", aList.pop())
print ("B List: ", aList.pop(2))
```

```
Output:
A List:  abc
B List:  zara
```

list.remove(obj): Removes object obj from list

Example:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.remove('xyz')
print ("List: ", aList)
aList.remove('abc');
print ("List: ", aList)
```

```
Output:
List :  [123, 'zara', 'abc', 'xyz']
List :  [123, 'zara', 'xyz']
```

list.reverse(): Reverses the objects of a list

Example:

```
aList = [123, 'xyz', 'zara', 'abc', 'xyz']
aList.reverse()
print ("List: ", aList)
```

```
Output:
List:  ['xyz', 'abc', 'zara', 'xyz', 123]
```

list.sort([func]): Sorts objects of list, uses compare func if given

Example:

```
aList = ['xyz', 'zara', 'abc', 'xyz']
aList.sort()
print ("List: ", aList)
```

```
Output:
List:  ['abc', 'xyz', 'xyz', 'zara']
```

List Comprehensions

```
liste = [i for i in range(1000)]
```

Method 1:

```
liste = [i for i in range(1000) if i % 2 == 0]
```

Method 2:

```
liste = []
for i in range(1000):
    if i % 2 == 0:
        liste += [i]
```

Sets

Sets are lists with no duplicate entries.

Operation	Equivalent	Result
s.update(t)	s |= t	return set s with elements added from t
s.intersection_update(t)	s &= t	return set s keeping only elements also found in t
s.difference_update(t)	s -= t	return set s after removing elements found in t
s.symmetric_difference_update(t)	s ^= t	return set s with elements from s or t but not both
s.add(x)		add element x to set s
s.remove(x)		remove x from set s; raises KeyError if not present
s.discard(x)		removes x from set s if present
s.pop()		remove and return an arbitrary element from s; raises KeyError if empty
s.clear()		remove all elements from set s

Example:

```
list = ["elma", "armut", "elma", "kebap", "şeker",
..."armut","çilek", "ağaç", "şeker", "kebap", "şeker"]
for i in set(list):
   print(i)
```

Output:
çilek
elma
kebap
armut
ağaç
şeker

Example:

```
list = ["elma", "armut", "elma", "kiraz",
... "çilek", "kiraz", "elma", "kebap"]
for i in set(list):
   print("{} count:  {}".format(i, list.count(i)))
```

Output:
armut count: 1
çilek count: 1
elma count: 3
kiraz count: 2
kebap count: 1

Tuples

- A tuple is a sequence of **immutable** Python objects. Tuples are sequences, just like lists.
- What are the differences between tuples and lists ?

Tuples

- A tuple is a sequence of **immutable** Python objects. Tuples are sequences, just like lists.
- The differences between tuples and lists are,
  - the **tuples cannot be changed** unlike lists,
  - tuples use parentheses, whereas lists use square brackets.

Creating a tuple:

```
tup = (1, 2, 3, 4, 5 )
```

Accessing Values in Tuples:

```
tup1 = ('physics', 'chemistry', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print ("tup1[0]: ", tup1[0])
print ("tup2[1:5]: ", tup2[1:5])
```

```
Output:
tup1[0]:  physics
tup2[1:5]:  (2, 3, 4, 5)
```

Updating Tuples

```
tup1 = (12, 34.56)
tup2 = ('abc', 'xyz')
tup3 = tup1 + tup2
print (tup3)
```

```
Output:
(12, 34.56, 'abc', 'xyz')
```

Dictionaries

- Dictionary as an unordered set of *key: value* pairs, with the requirement that the keys are **unique** (within one dictionary).

- Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by *keys*, which can be any immutable type.

Creating a dictionary:

```
dict = {'Name':'Zara','Age':7,'Class':'First'}
```

Accessing Values in a Dictionary:

```
dict ={'Name':'Zara','Age':7,'Class':'First'}
print ("dict['Name']: ",dict['Name'])
print ("dict['Age']: ",dict['Age'])
```

```
Output:
dict['Name']:  Zara
dict['Age']:  7
```

Updating a Dictionary

```
dict ={'Name':'Zara','Age':7,'Class':'First'}
dict['Age']=8
dict['School']="DPS School"
print ("dict['Age']: ",dict['Age'])
print ("dict['School']: ",dict['School'])
```

```
Output:
dict['Age']:  8
dict['School']:  DPS School
```

SN	Methods with Description
1	**dict.clear() :** Removes all elements of dictionary dict
2	**dict.copy() :** Returns a shallow copy of dictionary dict
3	**dict.fromkeys() :** Create a new dictionary with keys from seq and values set to value.
4	**dict.get(key, default=None) :** For key key, returns value or default if key not in dictionary
5	**dict.has_key(key) :** Returns true if key in dictionary dict, false otherwise
6	**dict.items() :** Returns a list of dict's (key, value) tuple pairs
7	**dict.keys() :** Returns list of dictionary dict's keys
8	**dict.setdefault(key, default=None) :** Similar to get(), but will set dict[key]=default if key is not already in dict
9	**dict.update(dict2) :** Adds dictionary dict2's key-values pairs to dict
10	**dict.values() :** Returns list of dictionary dict's values

Example:

```
phone_book = {"ahmet öz" : "0532 532 32 32",
"mehmet su": "0543 543 42 42",
"seda naz" : "0533 533 33 33",
"eda ala" : "0212 212 12 12"}
person = input("Please enter a name of a person: ")
if person in phone_book:
  answer = "{} adlı kişinin telefon numarası: {}"
  print(answer.format(person, phone_book [person]))
else:
      print("This name is not in this telephone book!")
```

Example:

```
names = ["ahmet", "mehmet", "fırat", "zeynep",
"selma", "abdullah", "cem"]
dict = {i: len(i) for i in names}
```

Create a dictionary from a list

File I/O

The open Function:

Example:

```
# Open a file
fo = open("foo.txt", "wb")
print ("Name of the file: ", fo.name)
print ("Closed or not : ", fo.closed)
print ("Opening mode : ", fo.mode)
```

Output:

Name of the file: foo.txt

Closed or not : False

Opening mode : wb

File I/O

grades.txt
98-86-100-54-63
54-89-78-90-85
0-95-70-69-87-55

Opening files to read:

```
my_file = open("grades.txt", "r")
first_line = my_file.readline()
grades = first_line.split('-')
print ("Grades from the first line: ", grades)
my_file.close()
```

Output:

```
Grades from the first line:  ['98', '86', '100', '54', '63\n']
```

File I/O

Opening modes

Sr.No.	Modes & Description
1	**r** Opens a file for reading only. The file pointer is placed at the beginning of the file. This is the default mode.
2	**rb** Opens a file for reading only in binary format. The file pointer is placed at the beginning of the file. This is the default mode.
3	**r+** Opens a file for both reading and writing. The file pointer placed at the beginning of the file.
4	**rb+** Opens a file for both reading and writing in binary format. The file pointer placed at the beginning of the file.
5	**w** Opens a file for writing only. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
6	**wb** Opens a file for writing only in binary format. Overwrites the file if the file exists. If the file does not exist, creates a new file for writing.
7	**w+** Opens a file for both writing and reading. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
8	**wb+** Opens a file for both writing and reading in binary format. Overwrites the existing file if the file exists. If the file does not exist, creates a new file for reading and writing.
9	**a** Opens a file for appending. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
10	**ab** Opens a file for appending in binary format. The file pointer is at the end of the file if the file exists. That is, the file is in the append mode. If the file does not exist, it creates a new file for writing.
11	**a+** Opens a file for both appending and reading. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.
12	**ab+** Opens a file for both appending and reading in binary format. The file pointer is at the end of the file if the file exists. The file opens in the append mode. If the file does not exist, it creates a new file for reading and writing.

Opening files and reading all lines:

```
my_file = open("expenses.txt", "r")
total_expense = 0
for line in my_file.readlines():
    expenses_list = line.split('-')
    for expense in expenses_list:
        total_expense += int(expense)
print("Total expense was:", total_expense)
my_file.close()
```

expenses.txt

100-54-63
78-90-85
70-69-87-55

Output:

```
Total expense was: 751
```

Example:

```
file = open("input.txt","r")
for aline in file.readlines():
    list = aline.split(':')
    print("name:",list[0],"phone number:",list[1])
file.close()
```

input.txt

Ahmet Özbudak : 0533 123 23 34
Mehmet Sülün : 0532 212 22 22
Sami Sam : 0542 333 34 34

Output:

```
name: Ahmet Özbudak  phone number:  0533 123 23 34
name: Mehmet Sülün  phone number:  0532 212 22 22
name: Sami Sam  phone number:  0542 333 34 34
```

Opening files to write (print output):

```
my_file = open("output.txt", "w")
my_file.write("I am writing this output to a file")
my_file.close()
```

Output: The sentence “I am writing this output to a file” will be written into a file named **`output.txt`**

New function:
`f.write(string)`

writes the contents of *string* to the file, returning the number of characters written.

Opening files to write (print output) cont.:

```
my_file = open("myage.txt", "w")
my_age = 20
my_file.write("I am " + str(my_age) + " years old.")
my_file.close()
```

Output: The sentence “I am 20 years old.” will be written into a file named **`myage.txt`**

`file.write(string)` takes only one argument, so you need to change any other types into strings and concatenate (+) all parts before passing them as an argument.

Exercise

- Write a program that reads an input file **`grades.txt`** which stores student names and their grades separated by a colon (:), prints out the name of the student with the highest grade, the name of the student with the lowest grade, and the average grade for this class. Your program should also write the same output to an output file named **`class_stats.txt`**
- **Note: use a dictionary to store the information from grades.txt**

grades.txt

```
Ahmet Özbudak:87
Mehmet Sülün:99
Sami Sam:45
Leyla Tan:93
Emre Göz:32
```