Hacettepe University

Computer Engineering Department

Programming in  python

BBM103 Introduction to Programming Lab 1

Week 4

Fall 2016

Functions

- **Good programming practice**: It is **functionality** that is important, not the amount of code!
- **The importance of functions**:
  - Break your code into separate, independent parts that will work together to solve the ultimate problem (**DECOMPOSITION**).
  - Hide the details of your computation as long as you know what it produces (**ABSTRACTION**)

Functions cont.

- **The advantages of functions**:
  - Break your code into **simpler independent modules**
  - These modules can be **reused** as many times as you like
  - And they need to be **debugged only once**
  - Keep your code **more organized and easier to understand**

Functions cont.

Defining functions:

Keyword

```
def function_name(arguments):
    function_body
    ...
```

0 or more parameters/arguments

Calling functions:

```
function_name(arguments)
```

- Example 1: **defining a <u>void</u> function (function that does not return a value)**

```
def greeting(name):
    print("Good afternoon, " + name + ".")

greeting("Emre")
```

Defining function

Calling function

  - **Output:** `Good afternoon, Emre.`

- Example 2: **defining a <u>fruitful</u> function (function that returns a value)**

```
def maximum(x, y):
    if x > y:
        return x
    else:
        return y
max_number = maximum(1, 5)
print("The maximum of two numbers is", max_number)
```

The function **returns** a value

- Example 3: **an example of a Boolean function (function that returns True or False)**

```
def is_even(number):
    return number%2 == 0
```

  - This function will return True if number is even, and False otherwise.

- An example use of this function:

Calling the function from within an **`if`** statement

```
if is_even(number):
    print (number," is an even number.")
else:
    print (number," is an odd number.")
```

- Example 4: **a function that calculates the factorial* of a number**

```
def factorial(number):
    product = 1
    for i in range(1, number+1):
        product = product * i
    return product
```

This line can be written more compactly as:

```
product *= i
```

- An example use of this function:

```
print("The factorial of 6: 6! = ", factorial(6))
```

* The **factorial** of a non-negative integer n, denoted by n!, is the product of all positive integers less than or equal to n. For example, 4! = 4 × 3 × 2 × 1 = 24

- Example 5: **Calculating area of plane shapes**

```
def triangle_area(b, h):
    return b*h/2
def square_area(a):
    return a*a
def rectangle_area(a, b):
    return a*b
user_choice = int(input("""Choose a shape you wish to calculate the area of:
(1) Triangle
(2) Square
(3) Rectangle\n1-3: """))
if user_choice == 1:
    base = int(input("Enter the length of the triangle base: "))
    height = int(input("Enter the height of the triangle: "))
    print("The area of the triangle is", triangle_area(base, height))
elif user_choice == 2:
    side = int(input("Enter the length of the square side: "))
    print("The area of the square is", square_area(side))
elif user_choice == 3:
    width = int(input("Enter the width of the rectangle: "))
    height = int(input("Enter the height of the rectangle: "))
    print("The area of the triangle is", rectangle_area(width, height))
else:
    print("Sorry, there is no such option.")
```

`\n` is the newline character

Collections

- **A Collection Groups Similar Things**

  List: ordered

  Set: unordered, no duplicates

  Tuple: unmodifiable list

  Dictionary: maps from values to values

Lists

Create list:

```
list1 = [1, 2, 3, 4, 5 ]
```

Example:

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

split()

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

Example:

```
t = ['pining', 'for', 'the', 'fjords']
delimiter = ' '
delimiter.join(t)
```

join()

Output: `'pining for the fjords'`

Accessing Values in Lists :

```
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
```

Delete 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:

Function with Description

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

Example:

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

Output:

```
length of the list 5
```

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

Example:

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

Output:

```
Max value element : 700
```

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

Example:

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

Output:

```
Min value element : 200
```

Method with Description

`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 count of how many times obj occurs in 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', 'abc', 123]
bList = [2009, 'manni']
aList.extend(bList)
print ("Extended List : ", aList )
```

Output:

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

list.index(obj) : Returns the lowest index in list that obj appears

Example:

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

Output:

```
Index for 700 :  1
```

list.insert(index, obj) : Inserts object obj into 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 last object or 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 objects of list in place

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, use 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]
```

Example:

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

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

Example:

```
numbers = [[0, 10], [6, 60],
…[12, 54], [67, 99]]
for i in numbers:
    print (*range(*i))
```

Example:

```
s = "İstanbul Büyükşehir Belediyesi"
s.split()
print(s)
```

Example:

```
list = []
while True:
    number = input("Please enter a number: (Enter q for quit) ")
    if number == "q":
        break
    number = int(number)
    if number not in list:
        list += [number]
        print(list)
    else:
        print("You entered this number before!")
```

Example:

```
numbers = 0
grades = []
for i in range(10):
    data = int(input("{}. not: ".format(i+1)))
    numbers += data
    grades += [data]
    print("Grades you entered ", *grades)
    print("Your GPA", numbers/10)
```

Sets

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
```

Example:

```
engineers = set(['John', 'Jane', 'Jack', 'Janice'])
programmers = set(['Jack', 'Sam', 'Susan', 'Janice'])
managers = set(['Jane', 'Jack', 'Susan', 'Zack'])
employees = engineers | programmers | managers           # union
engineering_management = engineers & managers            # intersection
fulltime_management = managers - engineers - programmers # difference
engineers.add('Marvin')                                  # add element
print (engineers)
employees.issuperset(engineers)
employees.update(engineers)
employees.issuperset(engineers)
for group in [engineers, programmers, managers, employees]:
    group.discard('Susan')          # unconditionally remove element
    print (group)
```

Tuples

Create 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

Accessing Values in 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 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 to learn his telephone number ")
if person in phone_book:
  answer = "{} adlı kişinin telefon numarası: {}"
  print(answer.format(person, phone_book [person]))
else:
      print("There is not this name in this telephone book!")
```

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

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

Example:

```
with open("fihrist.txt", "r+") as f:
    f.readline()
    f.seek(f.tell())
    f.truncate()
```

Example:

```
file = open("input.txt")
data=file. readline ()
list=data.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
```

readlines()
split()

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
```

Example:

```
def walk(dirname):
    for name in os.listdir(dirname):
        path = os.path.join(dirname, name)
        if os.path.isfile(path):
            print path
        else:
            walk(path)
```

Things to remember

- **Indentation is very important in Python**! To indicate a block of code in Python, you **must indent each line of the block by the same amount**.

- **Practice makes perfect**: the more you practice programming the easier it gets. It is easy to get stuck in the beginning, but don't get discouraged. Work with simple examples first. Move on to the harder examples when you have fully grasped the simple ones.

- **Try all examples on your own.**

- It is a lot of fun telling your computer what to do! **Stay motivated**.