Lecture 4.3 Function

January 31, 2022

[ ]:
```
#function is block of statements

# function call
# function definition
```

[ ]:
```
#Syntax of fun:

# function definition
def function_name(parameters):
    statements....

# function call
function_name(arguments)
```

[1]:
```
def greetings():
    print("Hello Students, Good Afternoon")

greetings()
```

```
Hello Students, Good Afternoon
```

[5]:
```
#default arguments
def greetings(name,msg=" Good Afternoon"):
    print("Hello " + name + msg)

greetings("Students")
```

```
Hello Students Good Afternoon
```

[7]:
```
#Arbitrary arguments-->non-keyword arguments
#If we don't know how many arguments that has to be passed into the function
#In function definition, we will add * before the parameter name
#it will receive a tuple of arguments
def greetings(*names):
    for name in names:
        print("Hello", name)

greetings("Amit","Dhruba","Ananya")
```

```
Hello Amit
Hello Dhruba
Hello Ananya
```

[ ]:
```
#keyword arguments
#we can send arguments with the key +value
#If we don't know how many arguments that has to be passed into the function
#In function definition, we will add ** before the parameter name
#it will receive a dictionary of arguments
```

[8]:
```
def fun(**kwargs):
    print("my name is " + kwargs['fname'])

fun(fname="Amit",lname="Kumar")
```

```
my name is Amit
```

[13]:
```
def add(a,b):
    return a+b

add(5,6)
```

[13]:
```
11
```

[14]:
```
help(add)
```

```
Help on function add in module __main__:

add(a, b)
```

[10]:
```
dir(tuple)
```

[10]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__gt__',
 '__hash__',
 '__init__',
```

```
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'count',
 'index']
```

[11]:
```
help(tuple.count)
```

```
Help on method_descriptor:

count(self, value, /)
    Return number of occurrences of value.
```

[15]:
```
#docstrings

def add(a,b):
    """
    Return addition of two numbers
    """
    return a+b

add(5,6)
```

[15]: 11

[16]:
```
help(add)
```

```
Help on function add in module __main__:

add(a, b)
    Return addition of two numbers
```

[18]:
```
#Version 1
#input->list of numbers
#output->list of square of numbers
list1=[1,2,3,4,5]
list2=[]
for item in list1:
    list2.append(item**2)
print(list2)
```

```
[1, 4, 9, 16, 25]
```

[19]:
```
#Version 2- list comprehension
#input->list of numbers
#output->list of square of numbers
list1=[1,2,3,4,5]
list2=[item**2 for item in list1]
list2
```

[19]:
```
[1, 4, 9, 16, 25]
```

[20]:
```
# map()
list1=[1,2,3,4,5]

def square(n):
    return n*n

map(square,list1)
```

[20]:
```
<map at 0x1b98fe849a0>
```

[22]:
```
# map()
list1=[1,2,3,4,5]

def square(n):
    return n*n

list(map(square,list1))
```

[22]:
```
[1, 4, 9, 16, 25]
```

[23]:
```
#input: list of cities
#output:list of length of cities

list2=['BBSR','Chennai','Mumbai','Delhi']

def length(str1):
    return len(str1)
```

```
list(map(length,list2))
```

[23]: [4, 7, 6, 5]

[24]:
```
#input:list of numbers
#output:check number is even or not

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

def isEven(item):
    return (item%2==0)

list(map(isEven,list1))
```

[24]: [False, True, False, True, False]

[25]:
```
#input:list of numbers
#output:list of even numbers

#filter()->returns the iterator whichever true values
list1=[1,2,3,4,5]

def isEven(item):
    return (item%2==0)

list(filter(isEven,list1))
```

[25]: [2, 4]

[26]:
```
#input: list of strings
#output: string which ends with 'ai'

list2=['Chennai','Mumbai','BBSR']
def is_ends_ai(item):
    if 'ai' in item:
        return True
    else:
        return False

list(map(is_ends_ai,list2))
```

[26]: [True, True, False]

[27]:
```
#input: list of strings
#output: string which ends with 'ai'
```

```
list2=['Chennai','Mumbai','BBSR']
def is_ends_ai(item):
    if 'ai' in item:
        return True
    else:
        return False

list(filter(is_ends_ai,list2))
```

[27]:
```
['Chennai', 'Mumbai']
```

[ ]:
```
# Lambda function
# anonymous function
# can take any no of arguments
#but can only have one expression
#syntax: lambda arguments:expression
```

[28]:
```
list2=['Chennai','Mumbai','BBSR']
list(filter(lambda item:'ai' in item ,list2))
```

[28]:
```
['Chennai', 'Mumbai']
```

[29]:
```
x= lambda a: a+5
print(x(6))
```

```
11
```

[30]:
```
x= lambda a,b: a+b
print(x(5,6))
```

```
11
```

[31]:
```
list1=[1,2,3,4,5,6]
list(filter(lambda item: item%2==0,list1))
```

[31]:
```
[2, 4, 6]
```

[32]:
```
#input->list of numbers
#output->sum of all numbers in the list
list1=[1,2,3,4,5]

def fun(x,y):
    return x+y
reduce(fun,list1)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
```

```
<ipython-input-32-09d352413ef5> in <module>
      5 def fun(x,y):
      6     return x+y
----> 7 reduce(fun,list1)

NameError: name 'reduce' is not defined
```

[34]:
```
import functools
list1=[1,2,3,4,5]

def fun(x,y):
    return x+y
functools.reduce(fun,list1)
"""
Execution:
1,2->3
3,3->6
6,4->10
10,5->15
"""
```

[34]: 15

[35]:
```
#use 'alias' name
import functools as f
list1=[1,2,3,4,5]

def fun(x,y):
    return x+y
f.reduce(fun,list1)
```

[35]: 15

[37]:
```
#use 'from' keyword
from functools import reduce
list1=[1,2,3,4,5]

def fun(x,y):
    return x+y

reduce(fun,list1)
```

[37]: 15