Chapter Functions

Learning Outcomes

- Scope of function
- parameter passing to function
- mutable/immutable properties of data objects
- passing strings, lists, tuples, dictionaries to functions
- default parameters
- positional parameters
- return values
- Functions using libraries: mathematical and string functions.

By: Vijeta Vashisth

Introduction

- A function is a programming block of codes which is used to perform a single or related task.
- It only runs when it is called.
- We can pass data, known as parameters, into a function.
- A function can return data as a result.
- Example: print()- built-in function and in the same way we can create our own functions known as user-defined functions.

Advantages of Using functions

1. **Program development made easy and fast :** Work can be divided among project members thus implementation can be completed fast.
2. **Program testing becomes easy :** Easy to locate and isolate a faulty function for further investigation.
3. **Code sharing becomes possible :** A function may be used later by many other programs this means that a python programmer can use function written by others, instead of starting over from scratch.
4. **Code re-usability increases :** A function can be used to keep away from rewriting the same block of codes which we are going use two or more locations in a program. This is especially useful if the code involved is long or complicated.
5. **Increases program readability :** It makes possible top down modular programming. In this style of programming, the high level logic of the overall problem is solved first while the details of each lower level functions is addressed later. The length of the source program can be reduced by using functions at appropriate places.
6. **Function facilitates procedural abstraction :** Once a function is written, it serves as a black box. All that a programmer would have to know to invoke a function would be to know its name, and the parameters that it expects.
7. **Functions facilitate the factoring of code :** A function can be called in other function and so on…

Creating & calling a Function

- A function is defined using the def keyword in python.E.g. program is given below.
- def my_own_function():
  print("Hello from a function")

  Function Block Definition/ creation

  #program start here.program code
  print("hello before calling a function")
  my_own_function() **#function calling.now function codes will be executed**
  print("hello after calling a function")
  now save the above source code in python file and execute it.

Variable's Scope

There are three types of variables with the view of scope.

1. Local variable – accessible only inside the functional block where it is declared.
2. Global variable – variable which is accessible among whole program using global keyword.
3. Non local variable – accessible in nesting of functions, using nonlocal keyword.

Local variable program:

```
def fun():
    s = "I love India!" #local variable
    print(s)

s = "I love World!"
fun()
print(s)
```

Output:

I love India!
I love World!

Global variable program:

```
def fun():
    global s #accessing/making global variable for fun()
    print(s)
    s = "I love India!“ #changing global variable's value
    print(s)

s = "I love world!"
fun()
print(s)
```

Output:

I love world!
I love India!
I love India!

By: Vijeta Vashisth

Arguments

- Information can be passed into functions as arguments.
- Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.
- The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

```
In [1]: def my_function(fname):
            print(fname + " class XII")

        my_function("Adi")
        my_function("Louisa")
        my_function(" Ram")
```

```
Adi class XII
Louisa class XII
 Ram class XII
```

- *Arguments* are often shortened to *args* in Python documentations.

By: Vijeta Vashisth

Question:

- Find the output of below program

In [4]:
```
def fun(x, y): # argument /parameter x and y
    global a
    a = 10
    x,y = y,x
    b = 20
    b = 30
    c = 30
    print(a,b,x,y)

a, b, x, y = 1, 2, 3,4
fun(50, 100) #passing value 50 and 100 in parameter x and y of function fun()
print(a, b, x, y)
```

Answer:

```
10 30 100 50
10 2 3 4
```

By: Vijeta Vashisth

Question 2:

- **Non local variable**

```
def fun1():
x = 100
def fun2():
nonlocal x #change it to global or remove this declaration
x = 200
print("Before calling fun2: " + str(x))
print("Calling fun2 now:")
fun2()
print("After calling fun2: " + str(x))
x=50
fun1()
print("x in main: " + str(x))
```

Answer:

- Error as there is no nonlocal variable

- Before calling fun2: 100
  Calling fun2 now:
  After calling fun2: 200
  x in main: 50

The *return* Statement

- The statement return [expression] exits in a function, optionally passing back an expression to the caller.
- A return statement with no arguments is the same as return None.
- **For example:**
- # Function definition is here to add the parameters and return the sum.
- def sum(arg1, arg2):

  total = arg1 + arg2

  return total

Parameters / Arguments

- The terms *parameter* and *argument* can be used for the same thing: information that are passed into a function.
- **From a function's perspective:**
- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that is sent to the function when it is called.

```
In [2]: def sum(x,y):
            z=x+y
            return z
        x,y= 4,5
        r=sum(x,y)
        print(r)

        9

In [ ]:
```

By: Vijeta Vashisth

Question:

- Write a program using function to add two numbers given by user?

Function Arguments

- Functions can be called using following types of formal arguments -
  - **Required arguments** - arguments passed to a function in correct positional order.
  - **Keyword arguments** - the caller identifies the arguments by the parameter name.
  - **Default arguments** - that assumes a default value if a value is not provided to argu.
  - **Variable-length arguments** – pass multiple values with single argument name.

Required Arguments

- #Required arguments
- def square(x):
  z=x*x
  return z
  r=square()
  print(r)

- Q: Will it work?

By: Vijeta Vashisth

Keyword Arguments

- In this type we can send arguments with the *key = value* syntax.
- This way the order of the arguments does not matter.
- **Example:**

```
def my_function(child3, child2, child1):
  print("The youngest child is " + child3)

my_function(child1 = “Ram", child2 = “Shyam", child3 = “Tina")
```

Default Arguments

```
#Default arguments
def sum(x=3,y=4):
    z=x+y
    return z

r=sum()
print(r)
r=sum(x=4)
print(r)
r=sum(y=45)
print(r)


#default value of x and y is being used
when it is not passed
```

By: Vijeta Vashisth

Variable- length Arguments

- Also known as Arbitrary arguments.
- **Example:**

```
def my_function(*kids):
  print("The youngest child is " + kids[2])

my_function(“Ram", “Shyam", “Tina")
```

#output will be The youngest child is Shyam

Question:

```
#Variable length arguments
def sum( *vartuple ):
   s=0
   for var in vartuple:
      s=s+int(var)
   return s;

r=sum( 70, 60, 50 )
print(r)
r=sum(4,5)
print(r)


#now the above function sum() can sum
n number of values
```

By: Vijeta Vashisth

Arbitrary Keyword Arguments, **kwargs

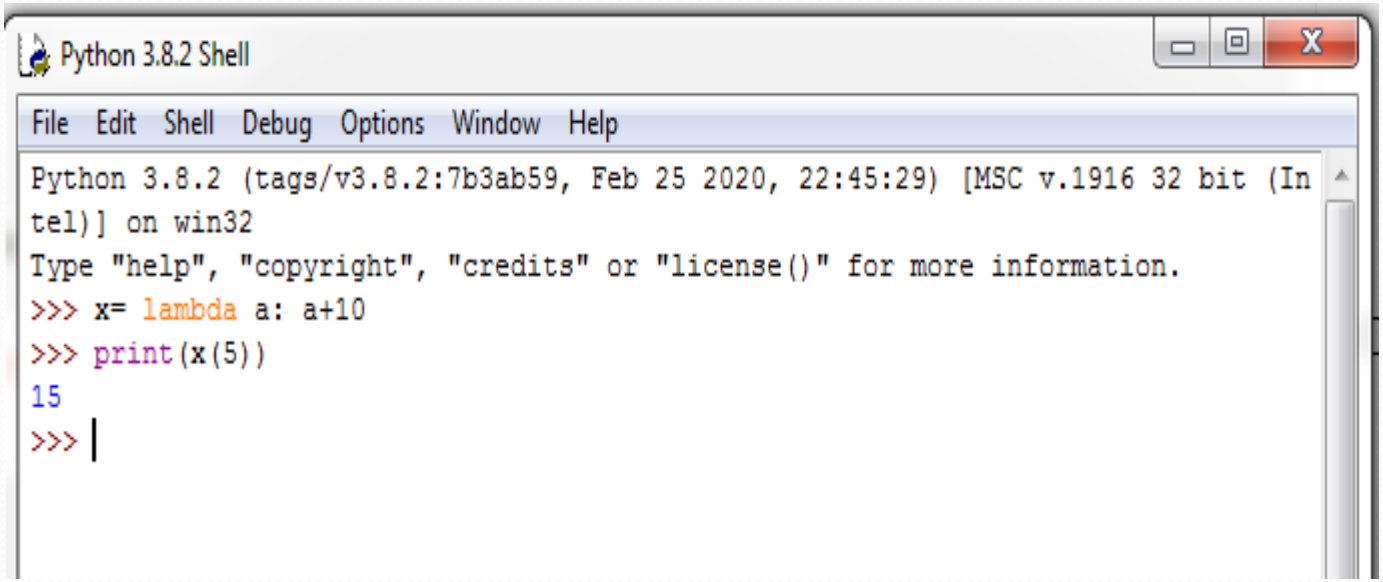
- If we do not know how many keyword arguments will be passed into a function, we add two asterisk: ** before the parameter name in the function definition.
- This way the function will receive a *dictionary* of arguments, and can access the items accordingly:
- **Example:**

```
def my_function(**kid):
  print("Her last name is " + kid["lname"])
my_function(fname = “Saga", lname = “Vashisth")

# Her last name is Vashisth
```

Python Lambda

- A lambda function is a small anonymous function.
- A lambda function can take any number of arguments, but can only have one expression.
- **Syntax: lambda *arguments* : *expression***
- The expression is executed and the result is returned:



Lambda (Contd)

- Lambda functions can take any number of arguments:
- **Example:** A lambda function that multiplies argument a with argument b and print the result:

```
x = lambda a, b : a * b
print(x(5, 6))
```

Q: Write a lambda function that sums arguments a, b and c and print the result?

Why Use Lambda Functions?

- The power of lambda is better shown when we use them as an anonymous function inside another function.
- Suppose, you have a function definition that takes one argument, and that argument will be multiplied with an unknown number:

```
def myfunc(n):
  return lambda a : a * n

mydoubler = myfunc(2)
mytripler = myfunc(3)

print(mydoubler(11))
print(mytripler(11))
```

Mutable/immutable properties of data objects w/r function

- Everything in Python is an object,and every objects in Python can be either **mutable** or **immutable**.
- Since everything in Python is an Object, every variable holds an object instance. When an object is initiated, it is assigned a unique object id. Its type is defined at runtime and once set can never change, however its state can be changed if it is mutable.
  Means a **mutable** object can be changed after it is created, and an **immutable** object can't.

- **Mutable objects: list, dict, set, byte array**
- **Immutable objects: int, float, complex, string, tuple, frozen set ,bytes**

How Objects are passed to functions

- Python uses a mechanism, which is known as "**Call-by-Object**", sometimes also called "**Call by Object Reference**" or "**Call by Sharing**".
- If we pass immutable arguments like integers, strings or tuples to a function, the passing acts like **Call-by-value**. It's different, if we pass mutable arguments.
- All **parameters (arguments**) in the Python language are **passed by reference**. It means that if we change what a parameter refers to within a function, that change also reflects back in the calling function.

```
#Pass by reference
def updateList(list1):
    print(id(list1))
    list1 += [10]
    print(id(list1))
n = [50, 60]
print(id(n))
updateList(n)
print(n)
print(id(n))
```

OUTPUT

```
34122928
34122928
34122928
[50, 60, 10]
34122928
```

#In above function list1 an object is being passed and its contents are changing because it is mutable that's why it is behaving like pass by reference

```
#Pass by value
def updateNumber(n):
    print(id(n))
    n += 10
    print(id(n))
b = 5
print(id(b))
updateNumber(b)
print(b)
print(id(b))
```

OUTPUT

```
1691040064
1691040064
1691040224
5
1691040064
```

#In above function value of variable b is not being changed because it is immutable that's why it is behaving like pass by value

Passing Arrays to the functions

- Arrays are popular in most programming languages like: Java, C/C++, JavaScript and so on. However, in Python, they are not that common. When people talk about Python arrays, more often than not, they are talking about Python lists. Array of numeric values are supported in Python by the array module.

- **Example:**

```
def dosomething( thelist ):
for element in thelist:
        print (element)

dosomething( ['1','2','3'] )
alist = ['red','green','blue']
dosomething( alist )

#OUTPUT:    1 2 3
             red
             green
             blue
```

Passing string to functions

- **# Python program to pass a string to the function**

- # function definition: it will accept a string parameter and print it
- def printMsg(str):

        # printing the parameter

        print str

printMsg("Hello world!")

printMsg("Hi! I am good.")

By: Vijeta Vashisth

Question:

- # function definition: it will accept a string parameter and return number of vowels

```
def countVowels(str):
        count = 0
        for ch in str:
                if ch in "aeiouAEIOU":
                        count +=1
        return count


str = "Hello world!"
Print( "No. of vowels are {0} in \"{1}\"".format(countVowels(str),str))

str = "Hi, I am good."
print ("No. of vowels are {0} in \"{1}\"".format(countVowels(str),str))
```

Function using libraries

- **Mathematical functions:**
  Mathematical functions are available under math module.To use mathematical functions under this module, we have to import the module using import math.
  For e.g.
  To use sqrt() function we have to write statements like given below.

```
import math
r=math.sqrt(4)
print(r)
```

OUTPUT :
2.0

Functions in Math Module

Function	Description	Example
ceil(n)	It returns the smallest integer greater than or equal to n.	math.ceil(4.2) returns 5
factorial(n)	It returns the factorial of value n	math.factorial(4) returns 24
floor(n)	It returns the largest integer less than or equal to n	math.floor(4.2) returns 4
fmod(x, y)	It returns the remainder when n is divided by y	math.fmod(10.5,2) returns 0.5
exp(n)	It returns e**n	math.exp(1) return 2.718281828459045
log2(n)	It returns the base-2 logarithm of n	math.log2(4) return 2.0
log10(n)	It returns the base-10 logarithm of n	math.log10(4) returns 0.6020599913279624
pow(n, y)	It returns n raised to the power y	math.pow(2,3) returns 8.0
sqrt(n)	It returns the square root of n	math.sqrt(100) returns 10.0
cos(n)	It returns the cosine of n	math.cos(100) returns 0.8623188722876839
sin(n)	It returns the sine of n	math.sin(100) returns -0.5063656411097588
tan(n)	It returns the tangent of n	math.tan(100) returns -0.5872139151569291
pi	It is pi value (3.14159...)	It is (3.14159...)
e	It is mathematical constant e (2.71828...)	It is (2.71828...)

Functions using libraries(string functions)

Method	Description
capitalize()	Converts the first character to upper case
casefold()	Converts string into lower case
center()	Returns a centered string
count()	Returns the number of times a specified value occurs in a string
encode()	Returns an encoded version of the string
endswith()	Returns true if the string ends with the specified value
find()	Searches the string for a specified value and returns the position of where it was found
format()	Formats specified values in a string
index()	Searches the string for a specified value and returns the position of where it was found

String function (contd)

Method	Description
isalnum()	Returns True if all characters in the string are alphanumeric
isalpha()	Returns True if all characters in the string are in the alphabet
isdecimal()	Returns True if all characters in the string are decimals
isdigit()	Returns True if all characters in the string are digits
isidentifier()	Returns True if the string is an identifier
islower()	Returns True if all characters in the string are lower case
isnumeric()	Returns True if all characters in the string are numeric
isprintable()	Returns True if all characters in the string are printable
isspace()	Returns True if all characters in the string are whitespaces
istitle()	Returns True if the string follows the rules of a title
isupper()	Returns True if all characters in the string are upper case
join()	Joins the elements of an iterable to the end of the string
ljust()	Returns a left justified version of the string
lower()	Converts a string into lower case
lstrip()	Returns a left trim version of the string
partition()	Returns a tuple where the string is parted into three parts

String function (contd)

Method	Description
replace()	Returns a string where a specified value is replaced with a specified value
split()	Splits the string at the specified separator, and returns a list
splitlines()	Splits the string at line breaks and returns a list
startswith()	Returns true if the string starts with the specified value
swapcase()	Swaps cases, lower case becomes upper case and vice versa
title()	Converts the first character of each word to upper case
translate()	Returns a translated string
upper()	Converts a string into upper case
zfill()	Fills the string with a specified number of 0 values at the beginning

Question Session