Functions

A function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.

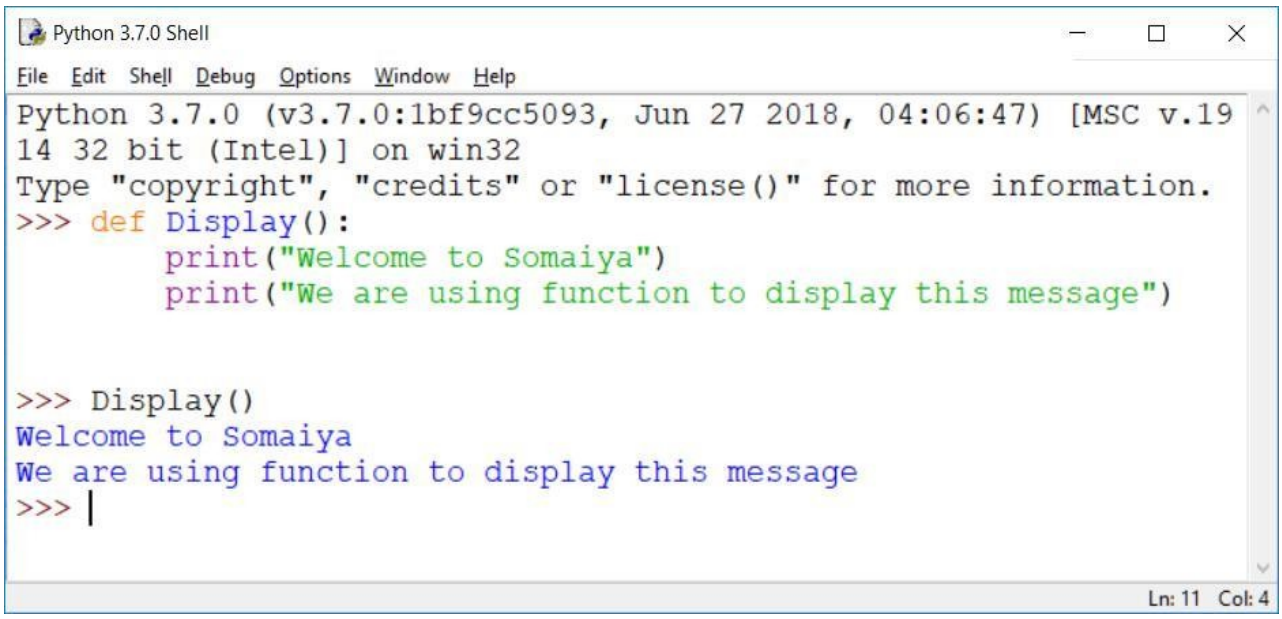
As you already know, Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called *user-defined functions.*

Defining a Function

You can define functions to provide the required functionality. Here are simple rules to define a function in Python.

- Function blocks begin with the keyword **def** followed by the function name and parentheses ( ).
- Any input parameters or arguments should be placed within these parentheses. You can also define parameters inside these parentheses.
- The first statement of a function can be an optional statement - the documentation string of the function or *docstring*.
- The code block within every function starts with a colon (:) and is indented.
- The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

User Defined Function Example:



```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
>>> def add(num1,num2):
        sum=num1+num2
        print("Sum of ",num1,"and ",num2," is", sum)


>>> add(5,7)
Sum of  5 and  7  is 12
>>>
Ln: 18  Col: 0
```

User defined Function with parameters All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

All parameters (arguments) in the Python language are passed by reference. It means if you change what a parameter refers to within a function, the change also reflects back in the calling function.

```
CallbyRef1.py - C:/Users/Rathod/Desktop/Python Codes/Callby...
File  Edit  Format  Run  Options  Window  Help
def square(mylist):
    mylist.append([16,25,36])
    print("Value after appending ",mylist)
    return
Ln: 5  Col: 0
```

```
Python 3.7.0 Shell
File  Edit  Shell  Debug  Options  Window  Help
======== RESTART: C:/Users/Rathod/Desktop/Python Codes/CallbyRef1.py ==
======
>>> mylist=[1,4,9]
>>> square(mylist)
Value after appending  [1, 4, 9, [16, 25, 36]]
>>> print('Value outside function ',mylist)
Value outside function  [1, 4, 9, [16, 25, 36]]
>>> |
Ln: 25  Col: 4
```

Function Arguments

You can call a function by using the following types of formal arguments −

- Required arguments

Example

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
======== RESTART: C:/Users/Rathod/Desktop/Python Codes/CallbyRef1.py ==
======
>>> mylist=[1,4,9]
>>> square(mylist)
Value after appending  [1, 4, 9, [16, 25, 36]]
>>> print('Value outside function ',mylist)
Value outside function  [1, 4, 9, [16, 25, 36]]
>>> square()
Traceback (most recent call last):
  File "<pyshell#13>", line 1, in <module>
    square()
TypeError: square() missing 1 required positional argument: 'mylist'
>>>
Ln: 30  Col: 4
```

- Keyword arguments

Keyword arguments are related to the function calls. When you use keyword arguments in a function call, the caller identifies the arguments by the parameter name.

Example

```
DisplayParam.py - C:/Users/Rathod/Desktop/Python Codes/DisplayParam....
File Edit Format Run Options Window Help
def Display(name,age,mobile):
    print("Name",name)
    print("Age",age)
    print("Mobile",mobile)
    return;
Ln: 4  Col: 26
```

Output

```
Python 3.7.0 Shell
File Edit Shell Debug Options Window Help
>>> Display("Roheet",12345,21)
Name Roheet
Age 12345
Mobile 21
>>> Display(age=21,mobile=12345,name="Roheet")
Name Roheet
Age 21
Mobile 12345
>>> |
Ln: 45  Col: 4
```

- Default arguments

A default argument is an argument that assumes a default value if a value is not provided in the function call for that argument.

Example

```
def Display(name,mobile,age=19):
    print("Name",name)
    print("Age",age)
    print("Mobile",mobile)
    return
```

Output

```
======= RESTART: C:/Users/Rathod/Desktop/Python Codes/DisplayParam.py =======
>>> Display(name="Akshay",age=25,mobile=54321)
Name Akshay
Age 25
Mobile 54321
>>> Display(name="Akshay",mobile=54321)
Name Akshay
Age 19
Mobile 54321
>>>
```

- Variable-length arguments

You may need to process a function for more arguments than you specified while defining the function. These arguments are called *variable-length* arguments and are not named in the function definition, unlike required and default arguments. An asterisk (*) is placed before the variable name that holds the values of all non-key word variable arguments. This tuple remains empty if no additional arguments are specified during the function call.

Example

DisplayParam.py - C:/Users/Rathod/Desktop/Python Codes/DisplayParam....

File Edit Format Run Options Window Help

```
def Display(name,mobile,*extra):
    print("Name",name)
    print("Mobile",mobile)
    print("Extra Info",extra)
    return;
```

Ln: 4 Col: 29

Output

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
======= RESTART: C:/Users/Rathod/Desktop/Python Codes/DisplayPara
m.py =======
>>> Display("Anu",98765,21)
Name Anu
Mobile 98765
Extra Info (21,)
>>> Display("Anu",98765,21,"Mumbai")
Name Anu
Mobile 98765
Extra Info (21, 'Mumbai')
>>>
```

Ln: 79 Col: 4

The *return* Statement

The statement return [expression] exits a function, optionally passing back an expression to the caller. A return statement with no arguments is the same as return None.

Example

ReturnExample.py - C:/Users/Rathod/Desktop/Python Codes/ReturnExam...

File Edit Format Run Options Window Help

```
def sum(num1,num2):
    total=num1+num2
    print("We are returning the sum value")
    return total
```

Ln: 3 Col: 41

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
>>> sum(5,7)
We are returning the sum value
12
>>>
```

Ln: 88 Col: 0

Scope of Variables

All variables in a program may not be accessible at all locations in that program. This depends on where you have declared a variable.

The scope of a variable determines the portion of the program where you can access a particular identifier. There are two basic scopes of variables in Python −

- Global variables
- Local variables

Global vs. Local variables

In general, variables that are defined inside a function body have a local scope, and those defined outside have a global scope. That means that local variables are defined within a function block and can only be accessed inside that function, while global variables can be accessed by all functions that might be in your script: **Example**

GlobalVarex.py - C:/Users/Rathod/Desktop/Python Codes/GlobalVarex.py (3.7.0)

File Edit Format Run Options Window Help

```
# Global variable
i = 1
# Define 'plus()' function to accept a variable number of arguments
def plus(*args):
  # Local variable 'sum()'
  total = 0
  for i in args:
    total += i
  return total

# Access the global variable
print("this is the initialized value " + str(i))

# (Try to) access the local variable
print("this is the sum " + str(total))
```

Ln: 16 Col: 0

You'll see that you'll get a NameError that says that the name 'total' is not defined when you try to print out the local variable total that was defined inside the function body. The init variable, on the other hand, can be printed out without any problems.

OUTPUT

Python 3.7.0 Shell

File Edit Shell Debug Options Window Help

```
>>>
======== RESTART: C:/Users/Rathod/Desktop/Python Codes/GlobalVarex.py =
=======
this is the initialized value 1
Traceback (most recent call last):
  File "C:/Users/Rathod/Desktop/Python Codes/GlobalVarex.py", line 15,
in <module>
    print("this is the sum " + str(total))
NameError: name 'total' is not defined
>>>
```

Ln: 100 Col: 4

Anonymous Functions in Python

Anonymous functions are also called lambda functions in Python because instead of declaring them with the standard defkeyword, you use the lambda keyword.

Lamda_Fun_Ex.py - C:/Users/Rathod/Desktop/Python Codes/Lamda_Fun_Ex.py (3.7.0)

File Edit Format Run Options Window Help

```
# "sum()" lambda function
sum = lambda x, y: x + y;

# Call the "sum()" anonymous function
sum(4,5)

# "Translates" to a User Defined Function
#def sum(x, y):
  #return x+y
```

Ln: 1 Col: 13

Note:
An anonymous function cannot be a direct call to print because lambda requires an expression
Lambda functions have their own local namespace and cannot access variables other than those in their parameter list and those in the global namespace.

Calculate Fibonacci Series using Functions

fibonacci.py - D:\Mousmi\Python\fibonacci.py (3.7.0)

File Edit Format Run Options Window Help

```
def fibonacci(n):
    #write Fibonacci series up to n
    a=0
    b=1
    while a < n:
        print(a,b,end=" ")
        a=b
        b=a+b
```

Ln: 9 Col: 0

OUTPUT

Python 3.7.0 Shell

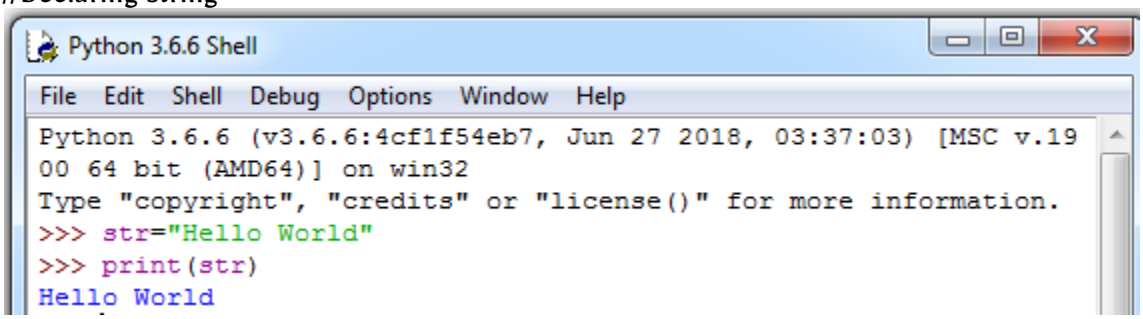
File Edit Shell Debug Options Window Help

```
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 27 2018, 04:06:47) [MSC v.1914 32
bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>>
=================== RESTART: D:\Mousmi\Python\fibonacci.py ============
=======
>>> fibonacci(10)
0 1 1 2 2 4 4 8 8 16
>>>
```

Ln: 7 Col: 5

➢ STRING

Consecutive sequence of characters is known as a string.
#Declaring string



STRING OPERATION

Operator	Description	Example
+ (Concatenation)	The + operator joins the text on both sides of the operator	>>> "Save "+"Earth" "Save Earth" To give a white space between the two words, insert a space before the closing single quote of the first literal.
* (Repetition)	The * operator repeats The string on the left hand side times the value on right hand side.	>>>3*"Save Earth " Save Earth Save Earth Save Earth
in (Membership)	The operator displays 1 if the string contains the given character or the sequence of characters	>>>A="Save Earth" >>> "S" in A True >>>"Save" in A

		True >>"SE" in A False
not in	The operator displays 1 if the string does not contain the given character or the sequence of characters. (working of this operator is the reverse of in operator discussed above	>>>"SE" not in "Save Earth" True >>>"Save " not in "Save Earth" False
Slice[n:m]	The Slice[n : m] operator extracts sub parts from the strings.	>>>A="Save Earth" >>> print A[1:3] av The print statement prints the substring starting from subscript 1 and ending at subscript 3 but not including subscript 3

➢ More on string Slicing

Consider the given figure

String A	S	A	V	E		E	A	R	T	H
Positive Index	0	1	2	3	4	5	6	7	8	9
Negative Index	-10	-9	-9	-7	-6	-5	-4	-3	-2	-1

Let"s understand Slicing in strings with the help of few examples.

Example

>>>A="Save Earth"

>>> print A[1:3]

av

The print statement prints the substring starting from subscript 1 and ending at subscript 3 .

Example

>>>print A[3:]

"e Earth"

Omitting the second index, directs the python interpreter to extract the substring till the end of the string

Example
>>>print A[:3]
Sav
Omitting the first index, directs the python interpreter to extract the substring before the second index starting from the beginning.

Example
>>>print A[:]
"Save Earth"
Omitting both the indices, directs the python interpreter to extract the entire string starting from 0 till the last index
Example
>>>print A[-2:]
"th"
For negative indices the python interpreter counts from the right side (also shown above). So the last two letters are printed.
Example
>>>Print A[:-2]
"Save Ear"

➢ String methods & built in functions

Syntax	Description	Example
len()	Returns the length of the string.	>>>A="Save Earth" >>> print len(A) >>>10
capitalize()	Returns the exact copy of the string with the first letter in upper case	>>>str="welcome" >>>print str.capitalize() Welcome
find(*sub*[, *start*[, *end*]])	The function is used to search the first occurrence of the substring in the given string. It returns the index at which the substring starts. It returns -1 if the substring does occur in the string.	>>>str='mammals' >>>str.find('ma') 0 **On omitting the start parameters, the function starts the search fromthe beginning.** >>>str.find('ma',2) 3 >>>str.find('ma',2,4) -1 **Displays -1 because the substringcould not be found between theindex 2 and 4-1** >>>str.find('ma',2,5) 3
isalnum()	Returns True if the string	>>>str='Save Earth'

	contains only letters and digit. It returns False ,If the string contains any special character like _ , @,#,* etc.	>>>str.isalnum() False The function returns False as **space**is an alphanumeric character. >>>'Save1Earth'.isalnum() True

isalpha()	Returns True if the string **contains only letters.**
	Otherwise return False.
		>>> 'Click123'.isalpha()
		False
		>>> 'python'.isalpha()
		True

isdigit()	Returns True if the string contains **only numbers**. Otherwise it returns False.	>>>print str.isdigit() False
lower()	Returns the exact copy of the string with all the letters in lowercase.	>>>print str.lower() “save earth“
islower()	Returns **True** if the string is in **lowercase.**	>>>print str.islower() True
isupper()	Returns **True** if the string is in **uppercase.**	>>>print str.isupper() False
upper()	Returns the exact copy of the string with all letters in **uppercase.**	>>>print str.upper() WELCOME

replace(old, new)	The function replaces all the occurrences of the old string with the new string	>>>str=“hello“
		>>> print str.replace('l','%')
		He%%o
		>>> print str.replace('l','%%')
		he%%%%o

join ()	Returns a string in which the string elements have been joined by a separator.	>>> str1=('jan', 'feb' ,'mar') >>>str=“&” >>>str.join(str1) 'jan&feb&mar'

Let’s discuss some interesting strings constants defined in string module:

string.ascii_uppercase

The command displays a string containing uppercase characters.

Example

```
>>>string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

string.ascii_lowercase

The command displays a string containing all lowercase characters.

Example

```
>>>string.ascii_lowercase
'abcdefghijklmnopqrstuvwxyz'
```

string.ascii_letters

The command displays a string containing both uppercase and lowercase characters.

```
>>>string.ascii_letters
'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
```

string.digits

The command displays a string containing digits.

```
>>>string.digits
'0123456789'
```

string.hexdigits

The command displays a string containing hexadecimal characters.

```
>>>string.hexdigits
'0123456789abcdefABCDEF'
```

string.octdigits

The command displays a string containing octal characters.

```
>>>string.octdigits
'01234567'
```

string.punctuations

The command displays a string containing all the punctuation characters.

```
>>>string.punctuations
'!"#$%&\'()*+,-./:;<=>?@[\\]^_`{|}~'
```

EXAMPLE:

To check string is palindrome or not

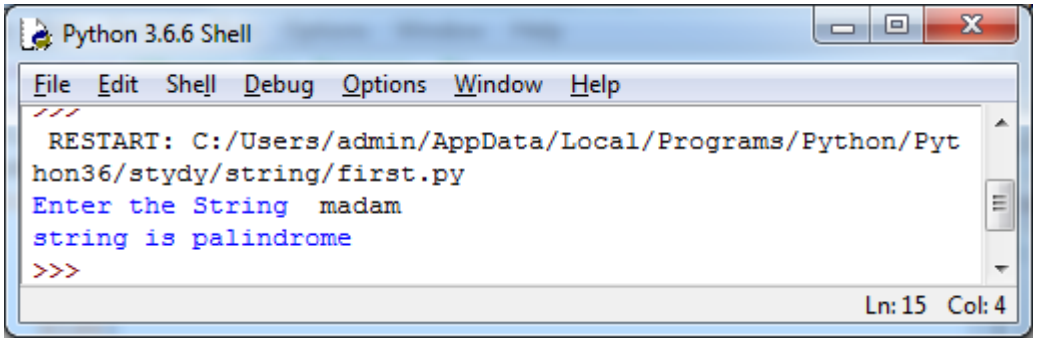
first.py - C:/Users/admin/AppData/Local/Programs/Python/Python36/sty...

File Edit Format Run Options Window Help

```
str=input("Enter the String")
l=len(str)
p=l-1
index=0
while (index<p):
    if(str[index]==str[p]):
        index=index+1
        p=p-1
    else:
            print("String is not a palidrome")
            break
else:
        print("string is palindrome")
```

Ln: 12 Col: 0

OUTPUT



String Formatting Operators in Python

Python Escape Characters

An Escape sequence starts with a backslash (\), which signals the compiler to treat it differently. Python subsystem automatically interprets an escape sequence irrespective of it is in a single-quoted or double-quoted Strings.

Let's discuss an example-One of an important Escape sequence is to escape a single-quote or a double-quote.

Suppose we have a string like – Python is a "widely" used language.

The double-quote around the word "widely" disguise python that the String ends up there.

We need a way to tell Python that the double-quotes inside the string are not the string markup quotes. Instead, they are the part of the String and should appear in the output.

To resolve this issue, we can escape the double-quotes and single-quotes as:

```
print ("Python is a "widely" used language")


# SyntaxError: invalid syntax


# After escaping with double-quotes


print ("Python is a \"widely\" used language")


# Output: Python is a "widely" used language
```

List of Escape Characters
Here is the complete list of escape characters that are represented using backslash notation.

Escape Char	Name
\\	Backslash (\)
\"	Double-quote (")
\r	Carriage Return (CR)
\t	Horizontal Tab (TAB)

Example

```
print ("Employee Name: %s,\nEmployee Age:%d" % ('Ashish',25))
```

```
# Employee Name: Ashish,
# Employee Age: 25
```

List of Format Symbols
Following is the table containing the complete list of symbols that you can use with the '%' operator.

Symbol	Conversion
%c	character
%s	string conversion via str() before formatting %s
	signed decimal integer %d
	octal integer %o
	hexadecimal integer (lowercase letters) %x
	hexadecimal integer (UPPER-case letters) %X
	floating-point real number %f

Way to use carriage return

1. Using only carriage return in Python

In this example, we will be using only the carriage return in the program in between the string.

```
string = 'My website is Latracal \rSolution'

print(string)
```

Output:

Solutionte is Latracal