

MARUDHAR KESARI JAIN COLLEGE FOR WOMEN, VANIYAMBADI

PG DEPARTMENT OF COMPUTER APPLICATIONS

Subject Name : PYTHON PROGRAMMING

CLASS : I-BCA

SUBJECT CODE :23UCA11

UNIT-III

Functions: Function Definition – Function Call – Variable Scope and its Lifetime-Return Statement.Function Arguments:Required Arguments,Keyword Arguments, Default Arguments and Variable Length Arguments-Recursion.Python Strings:String operations-Immutable Strings - Built-in String Methods and Functions - String Comparison.Modules: import statement- The Python module – dir() function – Modules and Namespace– Defining our own modules.

Function

The Python provides several functions and methods to write code very easily, and these functions and methods are called built-in function and methods. Python allows us to create our functions and methods, and these are called user-defined functions and methods.

2. 1.1 Built-In Functions

The Python interpreter has a number of functions that are built into it and are always available

Built-in Functions			
Input or Output	Datatype Conversion	Mathematical Functions	Other Functions
input() print()	bool() chr() dict() float() int() list() ord() set() str() tuple()	abs() divmod() max() min() pow() sum()	__import__() len() range() type()

Table 7.1 Commonly used built-in functions

Function Syntax	Arguments	Returns	Example Output
abs(x)	x may be an integer or floating point number	Absolute value of x	>>> abs(4) 4 >>> abs(-5.7) 5.7
divmod(x,y)	x and y are integers	A tuple: (quotient, remainder)	>>> divmod(7,2) (3, 1) >>> divmod(7.5,2) (3.0, 1.5) >>> divmod(-7,2) (-4, 1)
max(sequence) or max(x,y,z,...)	x,y,z,.. may be integer or floating point number	Largest number in the sequence/ largest of two or more arguments	>>> max([1,2,3,4]) 4 >>> max("Sincerity") 'y' #Based on ASCII value

			<pre>>>> max(23,4,56) 56</pre>
<pre>min(sequence) or min(x,y,z,...)</pre>	x, y, z,.. may be integer or floating point number	Smallest number in the sequence/ smallest of two or more arguments	<pre>>>> min([1,2,3,4]) 1 >>> min("Sincerity") 's' #Uppercase letters have lower ASCII values than lowercase letters. >>> min(23,4,56) 4</pre>
<pre>pow(x,y[,z])</pre>	x, y, z may be integer or floating point number	x^y (x raised to the power y) if z is provided, then: $(x^y) \% z$	<pre>>>> pow(5,2) 25.0 >>> pow(5.3,2.2) 39.2 >>> pow(5,2,4) 1</pre>
<pre>sum(x[,num])</pre>	x is a numeric sequence and num is an optional argument	Sum of all the elements in the sequence from left to right. if given parameter, num is added to the sum	<pre>>>> sum([2,4,7,3]) 16 >>> sum([2,4,7,3],3) 19 >>> sum((52,8,4,2)) 66</pre>
<pre>len(x)</pre>	x can be a sequence or a dictionary	Count of elements in x	<pre>>>> len("Patience") 8 >>> len([12,34,98]) 3 >>> len((9,45)) 2 >>> len({1:"Anuj",2:"Razia", 3:"Gurpreet",4:"Sandra"}) 4</pre>

Commonly Used Modules

- Python standard library also consists of a number of modules. While a function is a grouping of instructions, a module is a grouping of functions.
- A module is created as a python (.py) file containing a collection of function definitions.
- To use a module, we need to `import` the module. Once we import a module, we can directly use all the functions of that module.

The syntax of import statement is as follows:

```
import modulename1 [,modulename2, ...]
```

This gives us access to all the functions in the module(s). To call a function of a module, the function name should be preceded with the name of the module with a dot(.) as a separator.

The syntax is as shown below:

```
modulename.functionname()
```

Built-in Modules

Python library has many built-in modules that are really handy to programmers. Some commonly used modules and the frequently used functions that are found in those modules:

- math
- random
- statistics

math module:

It contains different types of mathematical functions. In order to use the `math` module we need to import it using the following statement:

```
import math
```

Commonly used functions in `math` module

Function Syntax	Returns	Example Output
<code>math.ceil(x)</code>	ceiling value of x	>>> <code>math.ceil(-9.7)</code> -9 >>> <code>math.ceil(9.7)</code> 10 >>> <code>math.ceil(9)</code> 9
<code>math.floor(x)</code>	floor value of x	>>> <code>math.floor(-4.5)</code> -5 >>> <code>math.floor(4.5)</code> 4 >>> <code>math.floor(4)</code> 4
<code>math.fabs(x)</code>	absolute value of x	>>> <code>math.fabs(6.7)</code> 6.7 >>> <code>math.fabs(-6.7)</code> 6.7 >>> <code>math.fabs(-4)</code> 4.0
<code>math.factorial(x)</code>	factorial of x	>>> <code>math.factorial(5)</code> 120
<code>math.fmod(x,y)</code>	$x \% y$ with sign of x	>>> <code>math.fmod(4,4.9)</code> 4.0 >>> <code>math.fmod(4.9,4.9)</code> 0.0 >>> <code>math.fmod(-4.9,2.5)</code> -2.4 >>> <code>math.fmod(4.9,-4.9)</code> 0.0
<code>math.gcd(x,y)</code>	gcd (greatest common divisor) of x and y	>>> <code>math.gcd(10,2)</code> 2
<code>math.pow(x,y)</code>	x^y (x raised to the power y)	>>> <code>math.pow(3,2)</code> 9.0 >>> <code>math.pow(4,2.5)</code>

		32.0 >>>math.pow(6.5,2) 42.25 >>>math.pow(5.5,3.2) 233.97
math.sqrt(x)	square root of x	>>> math.sqrt(144) 12.0 >>> math.sqrt(.64) 0.8
math.sin(x)	sine of x in radians	>>> math.sin(0) 0 >>> math.sin(6) -0.279

random module:

This module contains functions that are used for generating random numbers. In order to use the `random` module we need to import it using the following statement:

```
import random
```

Commonly used functions in `random` module

Function Syntax	Returns	Example Output
<code>random.random()</code>	Random Real Number (float) in the range 0.0 to 1.0	>>> random.random() 0.65333522
<code>random.randint(x,y)</code>	x, y are integers such that $x \leq y$ and returns Random integer between x and y	>>>random.randint(3,7) 4 >>> random.randint(-3,5) 1 >>> random.randint(-5,-3) -5.0
<code>random.randrange(y)</code>	Random integer between 0 and y	>>> random.randrange(5) 4
<code>random.randrange(x,y)</code>	Random integer between x and y	>>> random.randrange(2,7) 2

statistics module:

This module provides functions for calculating statistics of numeric (Real-valued) data. In order to use the `statistics` module we need to import it using the following statement:

```
import statistics
```

Commonly used functions in `statistics` module

Function Syntax	Returns	Example Output
<code>statistics.mean(x)</code>	arithmetic mean	<pre>>>> statistics. mean([11,24,32,45,51]) 32.6</pre>
<code>statistics.median(x)</code>	median (middle value) of x	<pre>>>>statistics. median([11,24,32,45,51]) 32</pre>
<code>statistics.mode(x)</code>	mode (the most repeated value)	<pre>>>> statistics. mode([11,24,11,45,11]) 11 >>> statistics. mode(("red","blue","red")) 'red'</pre>

- In order to get a list of modules available in Python, we can use the following statement:

```
>>> help("module")
```

- To view the content of a module say math, type the following:

```
>>> help("math")
```

From Statement

Instead of loading all the functions into memory by importing a module, from statement can be used to access only the required functions from a module. It loads only the specified function(s) instead of all the functions in a module.

Syntax :

```
>>> from modulename import functionname
```

Example :

```
>>> from random import random  
>>> random()  
0.04374770362702385  
>>> from math import ceil,sqrt  
>>> value = ceil(624.7)  
>>> sqrt(value)  
25.0
```

Function Definition and Calling the Function / User-defined Functions

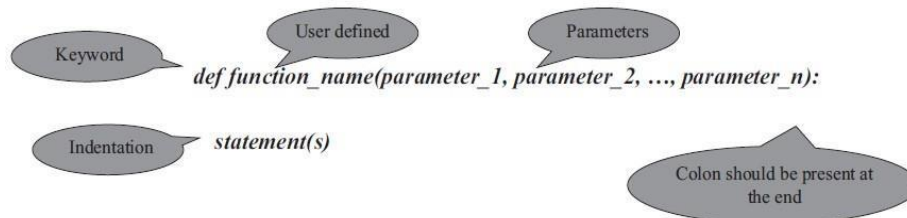
Defining a function in Python

The Python programming language provides the keyword `def` to create functions. The general syntax to create functions is as follows.

Syntax

```
def function_name(list_of_parameters):  
    statement_1  
    statement_2  
    statement_3  
    ...
```

The syntax for function definition is,



- ❖ The list of parameters in a function definition needs to be separated with a comma.
- ❖ In Python, every function returns a None value by default. However, we can return our value.
- ❖ Every function requires a function call to execute its code.

Calling a function in Python

In Python, we use the name of the function to make a function call. If the function requires any parameters, we need to pass them while calling it.

The general syntax for calling a function is as follows.

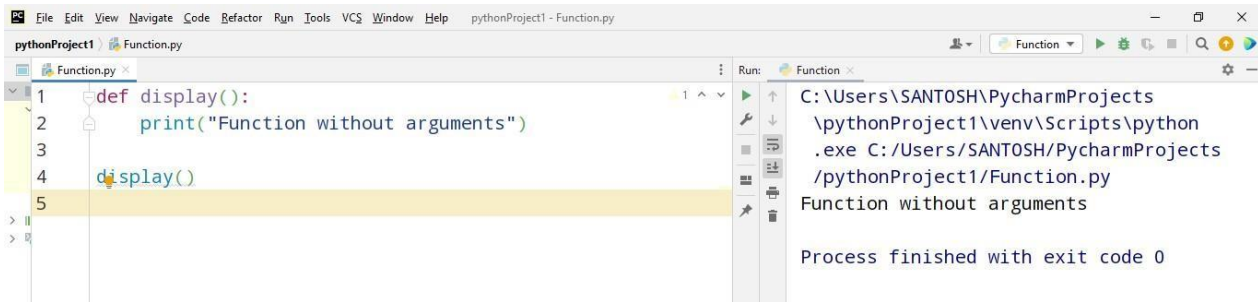
Syntax

```
function_name(parameter_1, parameter_2, ...)
```

Program to demonstrate a Function without arguments

```
def display():  
    print("Function without arguments")  
  
display()
```

When we run the above example code, it produces the following output.



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject1 - Function.py
pythonProject1 | Function.py
Function.py
1 def display():
2     print("Function without arguments")
3
4     display()
5
> ||
> |

Run: Function
C:\Users\SANTOSH\PycharmProjects
\pythonProject1\venv\Scripts\python
.exe C:/Users/SANTOSH/PycharmProjects
/pythonProject1/Function.py
Function without arguments

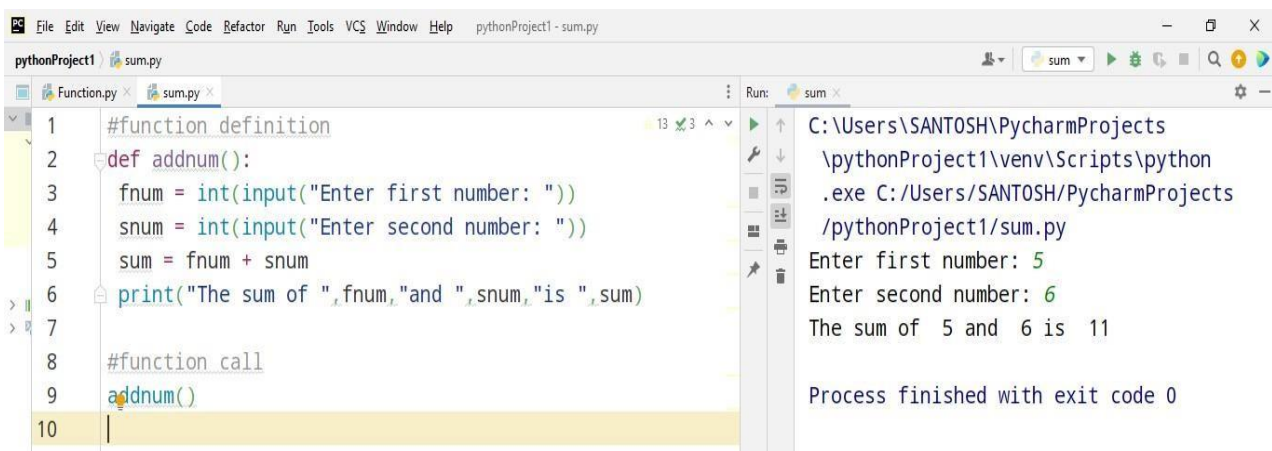
Process finished with exit code 0
```

Write a user defined function to add 2 numbers and display their sum.

```
#function definition
def addnum():
    fnum = int(input("Enter first number: "))
    snum = int(input("Enter second number: "))
    sum = fnum + snum
    print("The sum of ", fnum, "and ", snum, "is ", sum)

#function call
addnum()
```

When we run the above example code, it produces the following output.



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject1 - sum.py
pythonProject1 | sum.py
Function.py | sum.py
1 #function definition
2 def addnum():
3     fnum = int(input("Enter first number: "))
4     snum = int(input("Enter second number: "))
5     sum = fnum + snum
6     print("The sum of ", fnum, "and ", snum, "is ", sum)
7
8 #function call
9 addnum()
10

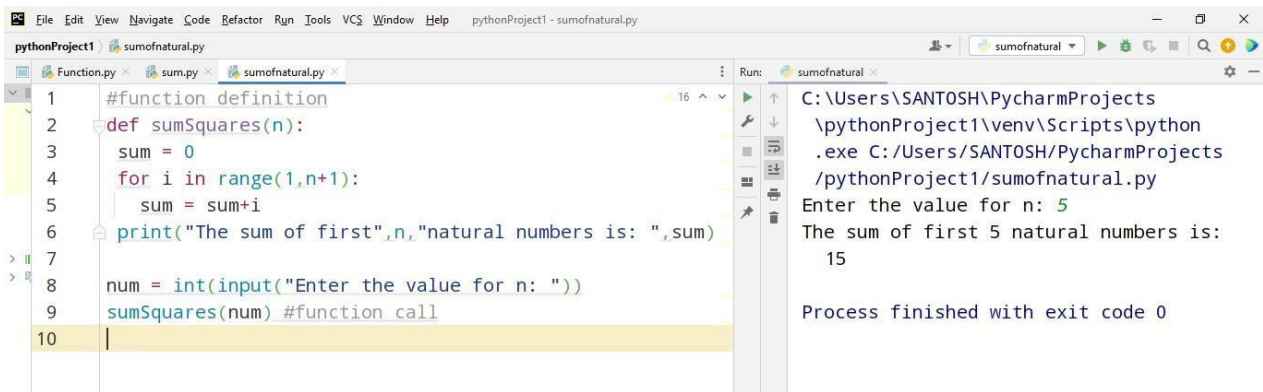
Run: sum
C:\Users\SANTOSH\PycharmProjects
\pythonProject1\venv\Scripts\python
.exe C:/Users/SANTOSH/PycharmProjects
/pythonProject1/sum.py
Enter first number: 5
Enter second number: 6
The sum of 5 and 6 is 11

Process finished with exit code 0
```

WAP using a user defined function that displays sum of first n natural numbers, where n is passed as an argument.

```
#function definition
def sumSquares(n):
    sum = 0
    for i in range(1, n+1):
        sum = sum+i
    print("The sum of first", n, "natural numbers is: ", sum)

num = int(input("Enter the value for n: "))
sumSquares(num) #function call
```

```
1 #function definition
2 def sumSquares(n):
3     sum = 0
4     for i in range(1,n+1):
5         sum = sum+i
6     print("The sum of first",n,"natural numbers is: ",sum)
7
8 num = int(input("Enter the value for n: "))
9 sumSquares(num) #function call
10
```

Run: sumofnatural

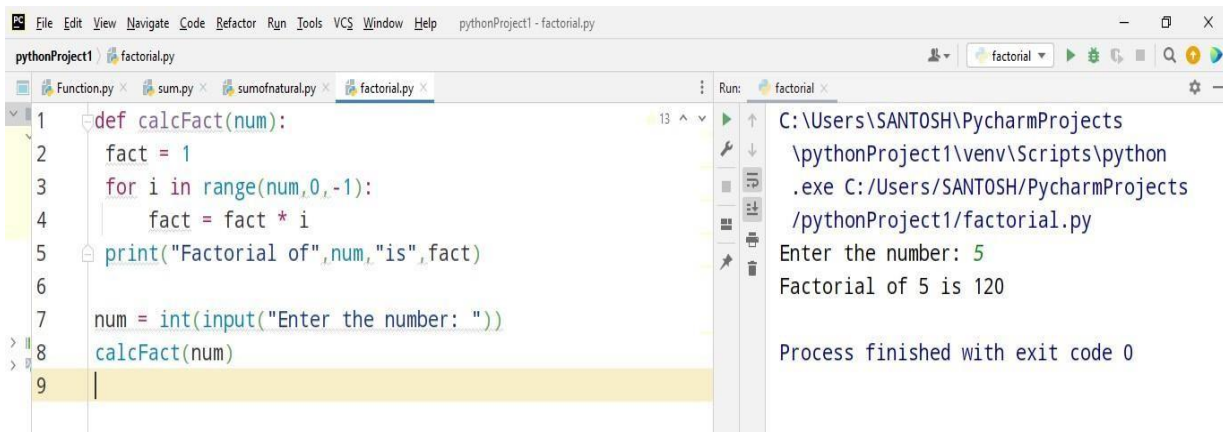
```
C:\Users\SANTOSH\PycharmProjects
\pythonProject1\venv\Scripts\python
.exe C:/Users/SANTOSH/PycharmProjects
/pythonProject1/sumofnatural.py
Enter the value for n: 5
The sum of first 5 natural numbers is:
15
Process finished with exit code 0
```

Write a program using a user defined function calcFact() to calculate and display the factorial of a number num passed as an argument

```
def calcFact(num):
    fact = 1
    for i in range(num,0,-1):
        fact = fact * i
    print("Factorial of",num,"is",fact)

num = int(input("Enter the number: "))
calcFact(num)
```

When we run the above example code, it produces the following output.



```
1 def calcFact(num):
2     fact = 1
3     for i in range(num,0,-1):
4         fact = fact * i
5     print("Factorial of",num,"is",fact)
6
7 num = int(input("Enter the number: "))
8 calcFact(num)
9
```

Run: factorial

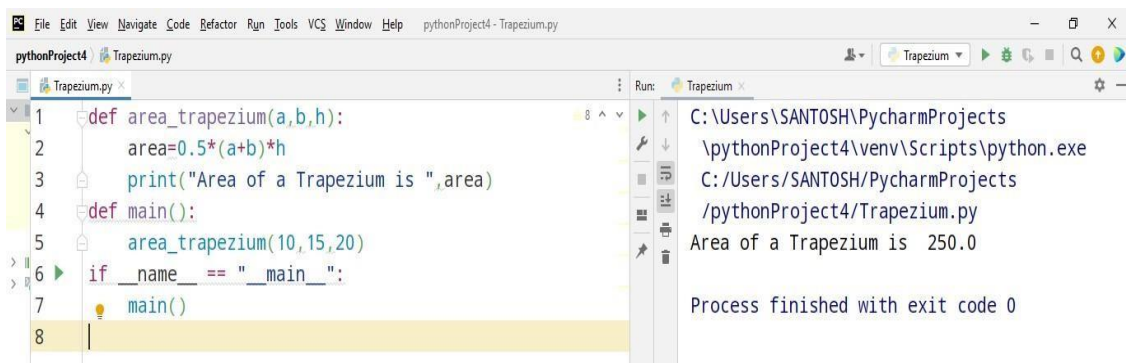
```
C:\Users\SANTOSH\PycharmProjects
\pythonProject1\venv\Scripts\python
.exe C:/Users/SANTOSH/PycharmProjects
/pythonProject1/factorial.py
Enter the number: 5
Factorial of 5 is 120
Process finished with exit code 0
```

Program to Find the Area of Trapezium Using the Formula

Area = $(1/2) * (a + b) * h$ Where a and b Are the 2 Bases of Trapezium and h Is the Height

```
def area_trapezium(a,b,h):
    area=0.5*(a+b)*h
    print("Area of a Trapezium is ",area)
def main():
    area_trapezium(10,15,20)
if __name__ == "__main__":
    main()
```

When we run the above example code, it produces the following output



```
pythonProject4 - Trapezium.py
pythonProject4 Trapezium.py
Trapezium.py x
1 def area_trapezium(a,b,h):
2     area=0.5*(a+b)*h
3     print("Area of a Trapezium is ",area)
4 def main():
5     area_trapezium(10,15,20)
6 if __name__ == "__main__":
7     main()
8

Run: Trapezium x
C:\Users\SANTOSH\PycharmProjects
\pythonProject4\venv\Scripts\python.exe
C:/Users/SANTOSH/PycharmProjects
/pythonProject4/Trapezium.py
Area of a Trapezium is 250.0
Process finished with exit code 0
```

The return statement and void Function

In Python, it is possible to compose a function without a return statement. Functions like this are called **void**, and they return `None`

A function may or may not return a value when called. The `return` statement returns the values from the function.

A `return` statement consists of the `return` keyword followed by an optional `return value`.

return [expression_list]

If an expression list is present, it is evaluated, else `None` is substituted

```
def add(a, b):
    result = a + b
    return result

print(add(2, 2))
```

Output : 4

Armstrong Number : The sum of cubes of each digit is equal to the number itself.

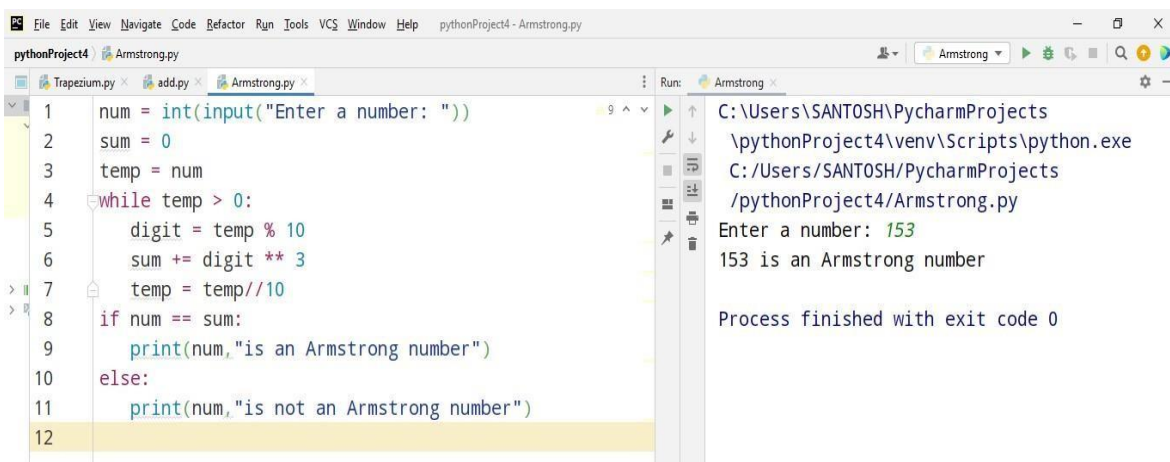
For example:

$153 = 1*1*1 + 5*5*5 + 3*3*3$ // 153 is an Armstrong number.

Program to Check If a 3 Digit Number Is Armstrong Number or Not

```
num = int(input("Enter a number: "))
sum = 0
temp = num
while temp > 0:
    digit = temp % 10
    sum += digit ** 3
    temp = temp//10
if num == sum:
    print(num,"is an Armstrong number")
else:
    print(num,"is not an Armstrong number")
```

When we run the above example code, it produces the following output



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject4 - Armstrong.py
pythonProject4 | Armstrong.py
Trapezium.py | add.py | Armstrong.py
1 num = int(input("Enter a number: "))
2 sum = 0
3 temp = num
4 while temp > 0:
5     digit = temp % 10
6     sum += digit ** 3
7     temp = temp//10
8 if num == sum:
9     print(num,"is an Armstrong number")
10 else:
11     print(num,"is not an Armstrong number")
12
Run: Armstrong
C:\Users\SANTOSH\PycharmProjects
\pythonProject4\venv\Scripts\python.exe
C:/Users/SANTOSH/PycharmProjects
/pythonProject4/Armstrong.py
Enter a number: 153
153 is an Armstrong number
Process finished with exit code 0
```

Enter a number: 121

121 is not an Armstrong number

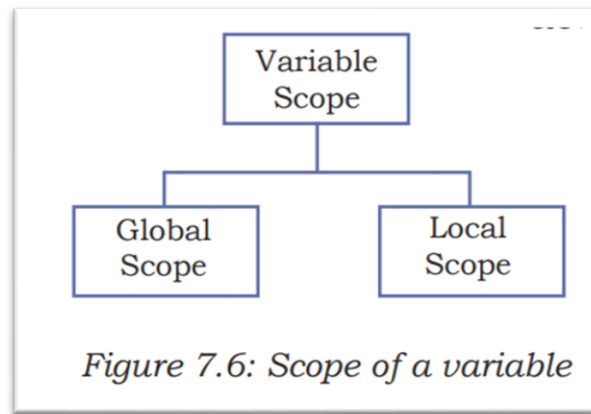
Enter a number: 407

407 is an Armstrong number

Scope and Lifetime of Variables

The part of the program where a variable is accessible can be defined as the scope of that variable. A variable can have one of the following two scopes:

A variable that has global scope is known as a global variable and a variable that has a local scope is known as a local variable.



Global Variable

A variable that is defined outside any function or any block is known as a global variable. It can be accessed in any functions defined onwards. Any change made to the global variable will impact all the functions in the program where that variable can be accessed.

Local Variable

A variable that is defined inside any function or a block is known as a local variable. It can be accessed only in the function or a block where it is defined. It exists only till the function executes.

Program to Demonstrate the Scope of Variables

```

num1 = 5 #globalvariable
def myfunc():
    # variable num1 outside the function
    print("Accessing num =", num1)
    num = 10 #localvariable
    print("num reassigned =", num)
myfunc()
print("Accessing num outside myfunc", num1)
  
```

When we run the above example code, it produces the following output

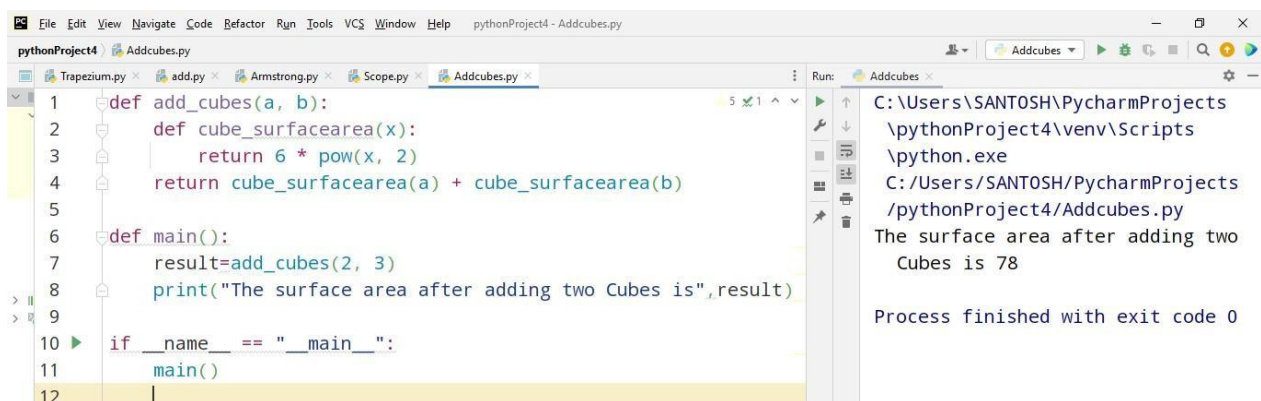
```

pythonProject4 - Scope.py
pythonProject4 Scope.py
Trapezium.py x add.py x Armstrong.py x Scope.py x
1 num1 = 5 #globalvariable
2 def myfunc():
3     # variable num1 outside the function
4     print("Accessing num =", num1)
5     num = 10 #localvariable
6     print("num reassigned =", num)
7 myfunc()
8 print("Accessing num outside myfunc", num1)
9
Run: Scope
C:\Users\SANTOSH\PycharmProjects
\pythonProject4\venv\Scripts\python.exe
C:/Users/SANTOSH/PycharmProjects
/pythonProject4/Scope.py
Accessing num = 5
num reassigned = 10
Accessing num outside myfunc 5
Process finished with exit code 0
  
```

Calculate and Add the Surface Area of Two Cubes. Use Nested Functions

```
def add_cubes(a, b):  
    def cube_surfacearea(x):  
        return 6 * pow(x, 2)  
    return cube_surfacearea(a) + cube_surfacearea(b)  
  
def main():  
    result=add_cubes(2, 3)  
    print("The surface area after adding two Cubes  
is",result)  
  
if __name__ == "__main__":  
    main()
```

When we run the above example code, it produces the following output



```
pythonProject4 - Addcubes.py  
pythonProject4 Addcubes.py  
1 def add_cubes(a, b):  
2     def cube_surfacearea(x):  
3         return 6 * pow(x, 2)  
4     return cube_surfacearea(a) + cube_surfacearea(b)  
5  
6 def main():  
7     result=add_cubes(2, 3)  
8     print("The surface area after adding two Cubes is",result)  
9  
10 if __name__ == "__main__":  
11     main()  
12  
Run: Addcubes x  
C:\Users\SANTOSH\PycharmProjects  
\pythonProject4\venv\Scripts  
\python.exe  
C:/Users/SANTOSH/PycharmProjects  
/pythonProject4/Addcubes.py  
The surface area after adding two  
Cubes is 78  
Process finished with exit code 0
```

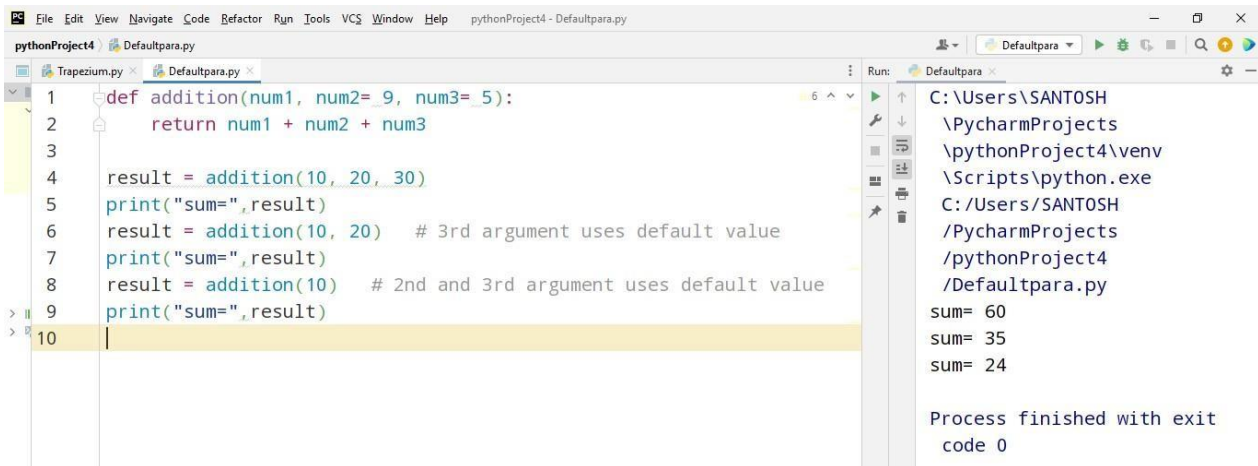
Default Parameters / Default Arguments

Python allows assigning a default value to the parameter. If the function is called with value then, the function executed with provided value, otherwise, it executed with the default value given in the function definition.

```
def addition(num1, num2= 9, num3= 5):  
    return num1 + num2 + num3
```

```
result = addition(10, 20, 30)  
print("sum=",result)  
result = addition(10, 20)    # 3rd argument uses default value  
print("sum=",result)  
result = addition(10)       #2nd and 3rd argument uses default value  
print("sum=",result)
```

When we run the above example code, it produces the following output



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject4 - Defaultpara.py
pythonProject4 Defaultpara.py
Trapezium.py Defaultpara.py
1 def addition(num1, num2= 9, num3= 5):
2     return num1 + num2 + num3
3
4     result = addition(10, 20, 30)
5     print("sum=", result)
6     result = addition(10, 20) # 3rd argument uses default value
7     print("sum=", result)
8     result = addition(10) # 2nd and 3rd argument uses default value
9     print("sum=", result)
10
Run: Defaultpara
C:\Users\SANTOSH
 \PycharmProjects
 \pythonProject4\venv
 \Scripts\python.exe
 C:/Users/SANTOSH
 /PycharmProjects
 /pythonProject4
 /Defaultpara.py
sum= 60
sum= 35
sum= 24
Process finished with exit
code 0
```

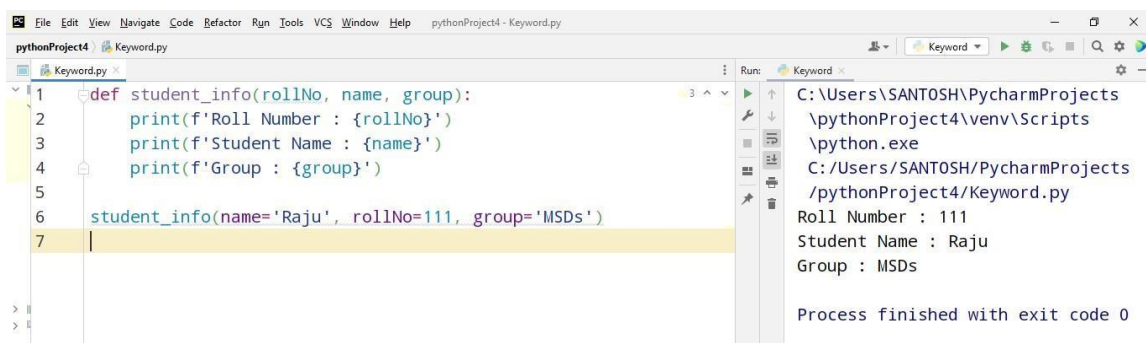
Keyword Arguments

The keyword argument is an argument passed as a value along with the parameter name (kwarg = value).

When keyword arguments are used, we may ignore the order of arguments. We may pass the arguments in any order because the Python interpreter uses the keyword provided to match with the respective parameter.

```
def student_info(rollNo, name, group):
    print(f'Roll Number : {rollNo}')
    print(f'Student Name : {name}')
    print(f'Group : {group}')
```

```
student_info(name='Raju', rollNo=111, group='MSDs')
```



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject4 - Keyword.py
pythonProject4 Keyword.py
Keyword.py
1 def student_info(rollNo, name, group):
2     print(f'Roll Number : {rollNo}')
3     print(f'Student Name : {name}')
4     print(f'Group : {group}')
5
6     student_info(name='Raju', rollNo=111, group='MSDs')
7
Run: Keyword
C:\Users\SANTOSH\PycharmProjects
 \pythonProject4\venv\Scripts
 \python.exe
 C:/Users/SANTOSH/PycharmProjects
 /pythonProject4/Keyword.py
Roll Number : 111
Student Name : Raju
Group : MSDs
Process finished with exit code 0
```


***args and **kwargs**

In Python, we can pass a variable number of arguments to a function using special symbols. There are two special symbols:

1. `*args` (Non Keyword Arguments)
2. `**kwargs` (Keyword Arguments)

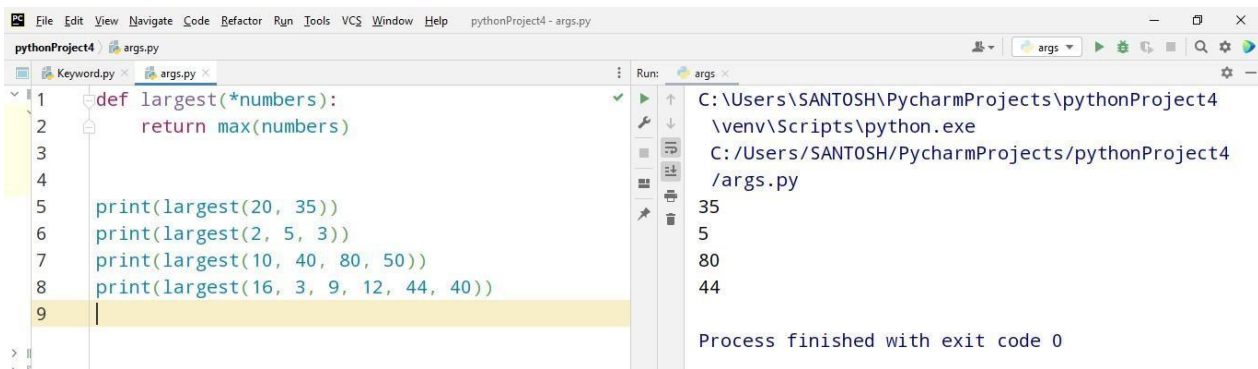
We use `*args` and `**kwargs` as an argument when we are unsure about the number of arguments to pass in the functions.

Python has `*args` which allow us to pass the variable number of non keyword arguments to function. In the function, we should use an asterisk (*) before the parameter name to pass a variable number of arguments.

Example program on *args

```
def largest(*numbers):  
    return max(numbers)  
  
print(largest(20, 35))  
print(largest(2, 5, 3))  
print(largest(10, 40, 80, 50))  
print(largest(16, 3, 9, 12, 44, 40))
```

When we run the above example code, it produces the following output



```
File Edit View Navigate Code Refactor Run Tools VCS Window Help pythonProject4 - args.py  
pythonProject4 args.py  
Keyword.py args.py  
1 def largest(*numbers):  
2     return max(numbers)  
3  
4  
5 print(largest(20, 35))  
6 print(largest(2, 5, 3))  
7 print(largest(10, 40, 80, 50))  
8 print(largest(16, 3, 9, 12, 44, 40))  
9  
Run: args  
C:\Users\SANTOSH\PycharmProjects\pythonProject4  
\venv\Scripts\python.exe  
C:/Users/SANTOSH/PycharmProjects/pythonProject4  
/args.py  
35  
5  
80  
44  
Process finished with exit code 0
```

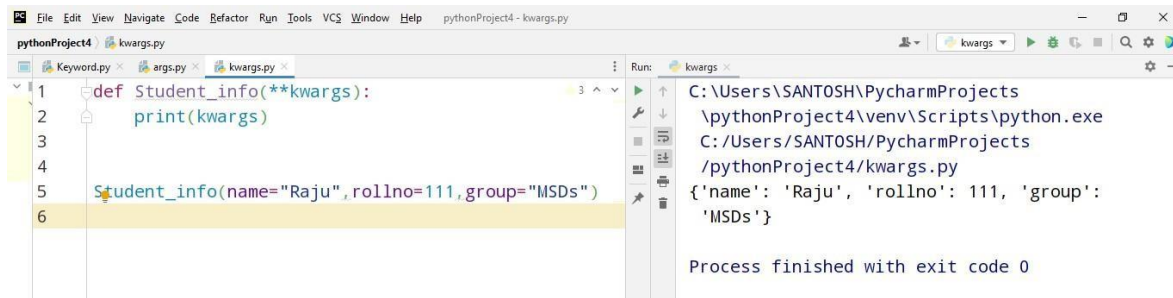
Python has `**kwargs` which allows us to pass the variable length of keyword arguments to the function. In the function, we use the double-asterisk (**) before the parameter name to denote this type of argument.

Example program on ****kwargs**

```
def Student_info(**kwargs):  
    print(kwargs)
```

```
Student_info(name="Raju", rollno=111, group="MSDs")
```

When we run the above example code, it produces the following output



The screenshot shows the PyCharm IDE interface. On the left, the code editor displays the following Python code:

```
1 def Student_info(**kwargs):  
2     print(kwargs)  
3  
4  
5 Student_info(name="Raju", rollno=111, group="MSDs")  
6
```

On the right, the Run console shows the output of the program:

```
C:\Users\SANTOSH\PycharmProjects  
\pythonProject4\venv\Scripts\python.exe  
C:/Users/SANTOSH/PycharmProjects  
/pythonProject4/kwarg.py  
{'name': 'Raju', 'rollno': 111, 'group':  
'MSDs'}
```

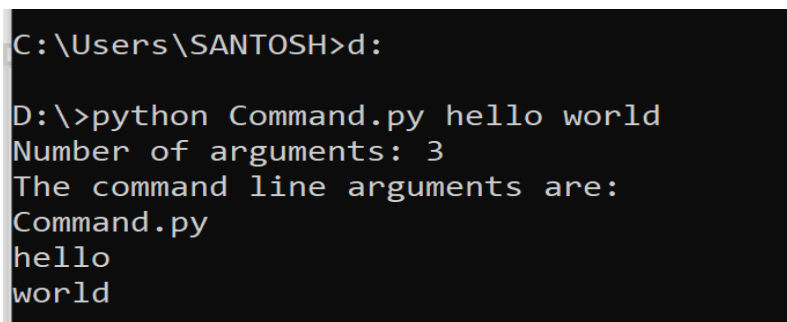
Below the output, it states: "Process finished with exit code 0".

Command Line Arguments

A Python program can accept any number of parameters from the command line. Python sys module stores the command line arguments into a list, we can access it using `sys.argv`

```
import sys  
print("Number of arguments:", len(sys.argv))  
print('The command line arguments are:')  
for i in sys.argv:  
    print(i)
```

Save above code Filename.py (Ex:- Command.py)



The terminal screenshot shows the following command line interaction:

```
C:\Users\SANTOSH>d:  
  
D:\>python Command.py hello world  
Number of arguments: 3  
The command line arguments are:  
Command.py  
hello  
world
```


3. Strings

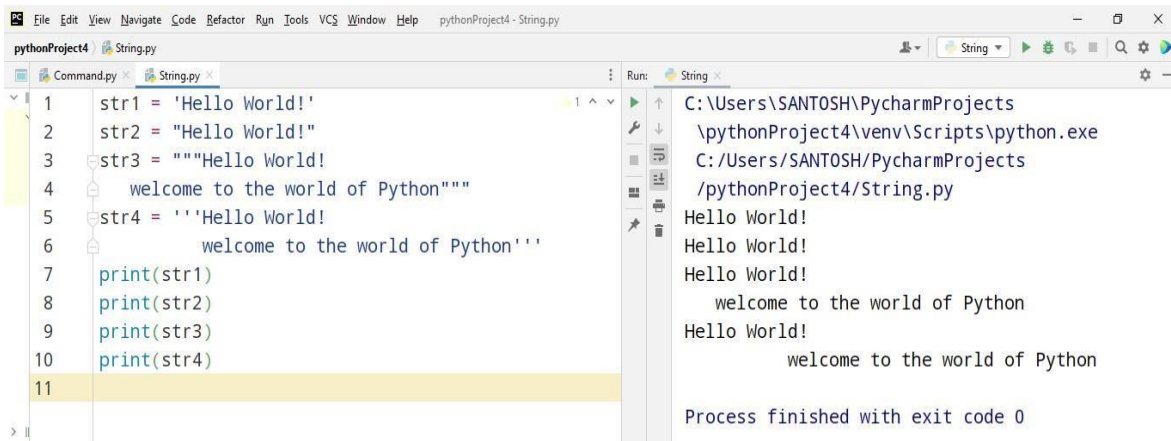
A string is a sequence of characters which is enclosed in quotes. In Python, A string can be created by enclosing one or more characters in single, double or triple quote.. The Python treats both single quote and double quote as same.

For example, the strings 'Hi Friend' and "Hi Friend" both are same.

Creating and Storing Strings

In Python, creating string variables is very simple as we need to assign a string value to a variable.

```
str1 = 'Hello World!'
str2 = "Hello World!"
str3 = """Hello World!
        welcome to the world of Python"""
str4 = '''Hello World!
        welcome to the world of Python'''
```



Accessing Characters in a String

- Each individual character in a string can be accessed using a technique called indexing.
- The index specifies the character to be accessed in the string and is written in square brackets ([]).
- The index of the first character (from left) in the string is 0 and the last character is n-1 where n is the length of the string.
- If we give index value out of this range then we get an IndexError. The index must be an integer (positive, zero or negative).

Table 8.1 Indexing of characters in string 'Hello World!'

Positive Indices	0	1	2	3	4	5	6	7	8	9	10	11
String	H	e	l	l	o		W	o	r	l	d	!
Negative Indices	-12	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1

```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MS
C v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>> str1='Hello World!'
>>> str1[0]
'H'
>>> str1[6]
'W'
>>> str1[11]
'!'
>>> str1[15]
Traceback (most recent call last):
  File "<pyshell#4>", line 1, in <module>
    str1[15]
IndexError: string index out of range
>>>
```

- The index can also be an expression including variables and operators but the expression must evaluate to an integer

```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MS
C v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>> str1='Hello World!'
>>> str1[2+4]
'W'
>>> str1[1.5]
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    str1[1.5]
TypeError: string indices must be integers
>>>
```

- Python allows an index value to be negative also. Negative indices are used when we want to access the characters of the string from right to left.
- Starting from right hand side, the first character has the index as -1 and the last character has the index -n where n is the length of the string.

```
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MS
C v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>> str1='Hello World!'
>>> str1[-1]
'!'
>>> str1[-12]
'H'
>>> n=len(str1)
>>> print(n)
12
>>> str1[n-1]
'!'
>>> str1[-n]
'H'
>>>
```

String is Immutable

A string is an immutable data type. It means that the contents of the string cannot be changed after it has been created. An attempt to do this would lead to an error.

```
IDLE Shell 3.10.4
File Edit Shell Debug Options Window Help
Python 3.10.4 (tags/v3.10.4:9d38120, Mar 23 2022, 23:13:41) [MS
C v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>> str1 = "Hello World!"
>>> #if we try to replace character 'e' with 'a'
>>> str1[1] = 'a'
Traceback (most recent call last):
  File "<pyshell#2>", line 1, in <module>
    str1[1] = 'a'
TypeError: 'str' object does not support item assignment
>>>
```

STRING OPERATIONS

Python allows certain operations on string data type, such as concatenation, repetition, membership and slicing.

Concatenation

To concatenate means to join. Python allows us to join two strings using concatenation operator plus which is denoted by symbol +.

```
>>> str1='Hello'
>>> str2='World!'
>>> str1+str2
'HelloWorld!'
```

Repetition

Python allows us to repeat the given string using repetition operator which is denoted by symbol *

```
>>> str1='Hello'
>>> str1*2
'HelloHello'
>>> str1*5
'HelloHelloHelloHelloHello'
>>>
```

Note: str1 still remains the same after the use of repetition operator

Membership

Python has two membership operators 'in' and 'not in'. The 'in' operator takes two strings and returns True if the first string appears as a substring in the second string, otherwise it returns False.

```
>>> str1='Hello World!'
>>> 'W' in str1
True
>>> 'Wor' in str1
True
>>> 'My' in str1
False
>>> |
```

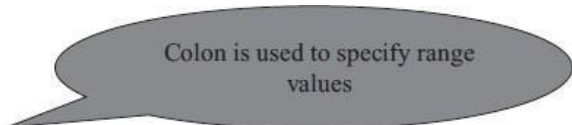
The 'not in' operator also takes two strings and returns True if the first string does not appear as a substring in the second string, otherwise returns False.

```
>>> str1='Hello World!'
>>> 'My' not in str1
True
>>> 'Hello' not in str1
False
>>> |
```

Slicing

In Python, to access some part of a string or substring, we use a method called slicing. This can be done by specifying an index range.

To access substring from a string, we use string variable name followed by square brackets with starting index, ending index and Step of the required substring.



Colon is used to specify range values

string_name[start:end[:step]]

```
>>> str1='Hello World!'
>>> 'My' not in str1
True
>>> 'Hello' not in str1
False
>>> str1 = 'Hello World!'
>>> str1[1:5]
'ello'
>>> str1[7:10]
'orl'
>>> str1[3:20]
'lo World!'
... ..
```

#first index > second index results in an #empty ' ' string

```
>>> str1='Hello World!'
>>> str1[7:2]
''
>>>
```

If the first index is not mentioned, the slice starts from index and If the second index is not mentioned, the slicing is done till the length of the string.

```
>>> str1='Hello World!'
>>> str1[:5]
'Hello'
>>> str1[6:]
'World!'
>>>
```

The slice operation can also take a third index that specifies the '**step size**'. For example, `str1[n:m:k]`, means every `k`th character has to be extracted from the string `str1` starting from `n` and ending at `m-1`. By default, the **step size** is **one**.

```
>>> str1='Hello World!'
>>> str1[0:10]
'Hello Worl'
>>> str1[0:10:2]
'HloWr'
>>> str1[0:10:3]
'HlWl'
>>>
```

Negative indexes can also be used for slicing. If we ignore both the indexes and give step size as `-1`, `str1` string is obtained in the reverse order.

```
>>> str1='Hello World!'
>>> str1[-6:-1]
'World'
>>> str1[::-1]
'!dlroW olleH'
>>>
```

Joining

The `join()` string method returns a string by joining all the elements of an iterable (list, string, tuple), separated by a string separator.

```
>>> text = ['Python', 'by', 'Santhosh', 'Sir']
>>> print(' '.join(text))
Python by Santhosh Sir
>>>
```

Traversing A String

We can access each character of a string or traverse a string using `for` loop and `while` loop

String Traversal Using for Loop:

```
str1 = 'Hello World!'
for ch in str1:
    print(ch)
```

String Traversal Using while Loop:

```
str1 = 'Hello World!'
index = 0
while index < len(str1):
    print(str1[index])
    index += 1
```

String Methods and Built-In Functions

Method	Description	Example
<code>len()</code>	Returns the length of the given string	<pre>>>> str1 = 'Hello World!' >>> len(str1) 12</pre>
<code>title()</code>	Returns the string with first letter of every word in the string in uppercase and rest in lowercase	<pre>>>> str1 = 'hello WORLD!' >>> str1.title() 'Hello World!'</pre>
<code>lower()</code>	Returns the string with all Uppercase letters converted to lowercase	<pre>>>> str1 = 'hello WORLD!' >>> str1.lower() 'hello world!'</pre>
<code>upper()</code>	Returns the string with all lowercase letters converted to uppercase	<pre>>>> str1 = 'hello WORLD!' >>> str1.upper() 'HELLO WORLD!'</pre>
<code>count(str, start, end)</code>	Returns number of times substring <code>str</code> occurs in the given string. If we do not give start index and end index then	<pre>>>> str1 = 'Hello World! Hello Hello' >>> str1.count('Hello', 12, 25) 2</pre>

	searching starts from index 0 and ends at length of the string	<pre>>>> str1.count('Hello') 3</pre>
find (str,start, end)	Returns the first occurrence of index of substring str occurring in the given string. If we do not give start and end then searching starts from index 0 and ends at length of the string. If the substring is not present in the given string, then the function returns -1	<pre>>>> str1 = 'Hello World! Hello Hello' >>> str1.find('Hello',10,20) 13 >>> str1.find('Hello',15,25) 19 >>> str1.find('Hello') 0> >> str1.find('Hee') -1</pre>
Index (str, start, end)	Same as find() but raises an exception if the substring is not present in the given string	<pre>>>> str1 = 'Hello World! Hello Hello' >>> str1.index('Hello') 0 >>> str1.index('Hee') ValueError: substring not found</pre>
endswith()	Returns True if the given string ends with the supplied substring otherwise returns False	<pre>>>> str1 = 'Hello World!' >>>str1.endswith('World!') True >>> str1.endswith('!!') True >>> str1.endswith('lde') False</pre>
startswith()	Returns True if the given string starts with the supplied substring otherwise returns False	<pre>>>> str1 = 'Hello World!' >>> str1.startswith('He') True >>> str1.startswith('Hee') False</pre>
isalnum()	Returns True if characters of the given string are either alphabets or numeric. If whitespace or special symbols are part of the given string or the string is empty it returns False	<pre>>>> str1 = 'HelloWorld' >>> str1.isalnum() True >>> str1 = 'HelloWorld2' >>> str1.isalnum() True >>> str1 = 'HelloWorld!!' >>> str1.isalnum() False</pre>
islower()	Returns True if the string is non-empty and has all lowercase alphabets, or has at least one character as lowercase alphabet and rest are non-alphabet characters	<pre>>>> str1 = 'hello world!' >>> str1.islower() True >>> str1 = 'hello 1234' >>> str1.islower() True >>> str1 = 'hello ??' >>> str1.islower() True</pre>

		<pre>>>> str1 = '1234' >>> str1.islower() False >>> str1 = 'Hello World!' >>> str1.islower() False</pre>
isupper()	Returns True if the string is non-empty and has all uppercase alphabets, or has at least one character as uppercase character and rest are non-alphabet characters	<pre>>>> str1 = 'HELLO WORLD!' >>> str1.isupper() True >>> str1 = 'HELLO 1234' >>> str1.isupper() True >>> str1 = 'HELLO ??' >>> str1.isupper() True >>> str1 = '1234' >>> str1.isupper() False >>> str1 = 'Hello World!' >>> str1.isupper() False</pre>
isspace()	Returns True if the string is non-empty and all characters are white spaces (blank, tab, newline, carriage return)	<pre>>>> str1 = ' \n \t \r' >>> str1.isspace() True >>> str1 = 'Hello \n' >>> str1.isspace() False</pre>
istitle()	Returns True if the string is non-empty and title case, i.e., the first letter of every word in the string in uppercase and rest in lowercase	<pre>>>> str1 = 'Hello World!' >>> str1.istitle() True >>> str1 = 'hello World!' >>> str1.istitle() False</pre>
lstrip()	Returns the string after removing the spaces only on the left of the string	<pre>>>> str1 = ' Hello World! ' >>> str1.lstrip() 'Hello World! '</pre>
rstrip()	Returns the string after removing the spaces only on the right of the string	<pre>>>> str1 = ' Hello World! ' >>> str1.rstrip() ' Hello World!'</pre>
strip()	Returns the string after removing the spaces both on the left and the right of the string	<pre>>>> str1 = ' Hello World! ' >>> str1.strip() 'Hello World!'</pre>
replace(olds tr, newstr)	Replaces all occurrences of old string with the new string	<pre>>>> str1 = 'Hello World!' >>> str1.replace('o', '*') 'Hell* W*rld!' >>> str1 = 'Hello World!' >>> str1.replace('World', 'Country') 'Hello Country!'</pre>

		<pre>>>> str1 = 'Hello World! Hello' >>> str1.replace('Hello','Bye') 'Bye World! Bye'</pre>
join()	Returns a string in which the characters in the string have been joined by a separator	<pre>>>> str1 = ('HelloWorld!') >>> str2 = '-' #separator >>> str2.join(str1) 'H-e-l-l-o-W-o-r-l-d-!'</pre>
partition()	Partitions the given string at the first occurrence of the substring (separator) and returns the string partitioned into three parts. 1. Substring before the separator 2. Separator 3. Substring after the separator If the separator is not found in the string, it returns the whole string itself and two empty strings	<pre>>>> str1 = 'India is a Great Country' >>> str1.partition('is') ('India ', 'is', ' a Great Country') >>> str1.partition('are') ('India is a Great Country', ' ', ' ')</pre>
capitalize()	Returns a copy of the string with its first character capitalized and the rest lowercased	<pre>>>>str='hello world!' >>>str.capitalize() 'Hello world!'</pre>
casefold()	returns a string where all the characters are in lower case. It is similar to the lower() method, but the casefold() method converts more characters into lower case.	<pre>>>>str="HELLO world" >>>str.casefold() 'hello world'</pre>

Write Python Program to Convert Uppercase Letters to Lowercase and Vice Versa

```
def case_conversion(string):
    str1=str()
    for ch in string:
        if ch.isupper():
            str1+=ch.lower()
        else:
            str1 += ch.upper()
    print("The modified string is ",str1)
def main():
    str2=input("Enter a String :")
    case_conversion(str2)
if __name__=="__main__":
    main()
```

Output :

```
Enter a String : Hello WORLD
The modified string is hELLO world
```

Example Program on String Methods

```
str1 = ' hello World! '  
print("String in Uppercase :", str1.upper())  
print("String in Lower case :", str1.lower())  
print("Capitalized String :", str1.capitalize())  
print("String with first letter :", str1.title())  
print("String alphanumeric :", str1.isalnum())  
print("String in lowercase :", str1.islower())  
print("String in uppercase :", str1.isupper())  
print("Swapcase :", str1.swapcase())  
print("Right Strip of String :", str1.rstrip())  
print("Left Strip of String :", str1.lstrip())  
print("Right & Left Strip of String :", str1.strip())
```

Output :

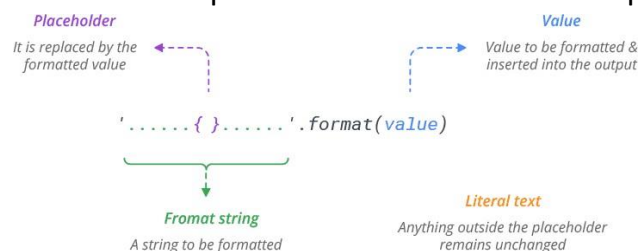
```
String in Uppercase : HELLO WORLD!  
String in Lower case : hello world!  
Capitalized String : hello world!  
String with first letter : Hello World!  
String alphanumeric : False  
String in lowercase : False  
String in uppercase : False  
Swapcase : HELLO wORLD!  
Right Strip of String : hello World!  
Left Strip of String : hello World!  
Right & Left Strip of String : hello World!
```

Formatting Strings

Python f-string is the newest Python syntax to do string formatting. It is available since Python 3.6. Python f-strings provide a faster, more readable, more concise, and less error prone way of formatting strings in Python.

The f-strings have the f prefix and use {} brackets to evaluate values.

The format strings will contain the curly braces {} and the format() method will use those curly braces {} as placeholders to replace with the content of the parameters.



```
name = 'Raju'
age = 23
print('%s is %d years old' % (name, age))
print('{} is {} years old'.format(name, age))
print(f'{name} is {age} years old')
```

Output :

```
Raju is 23 years old
Raju is 23 years old
Raju is 23 years old
```

Function Prototypes:

Based on the data flow between the calling function and called function, the functions are classified as follows...

- Function without Parameters and without Return value
- Function with Parameters and without Return value
- Function without Parameters and with Return value
- Function with Parameters and with Return value

Function without Parameters and without Return value

- In this type of functions there is no data transfer between calling function and called function.
- Simply the execution control jumps from calling-function to called function and executes called function, and finally comes back to the calling function.
- For example, consider the following program..

```
def add():
    a=int(input("enter a"))
    b=int(input("enter b"))
    c=a+b
    print(c)
add()
```

```
Output :
enter a 10
enter b 20
30
```

Function with Parameters and without Return value

- In this type of functions there is data transfer from calling-function to called function (parameters) but there is no data transfer from called function to calling-function (return value).
- The execution control jumps from calling-function to called function along with the parameters and executes called function, and finally comes back to the calling function.
- For example, consider the following program...

```
def add(a,b):
    c=a+b
    print(c)

a=int(input("enter a"))
b=int(input("enter b"))
add(a,b)
```

```
Output :
    enter a 10
    enter b 20
    30
```

Function without Parameters and with Return value

- In this type of functions there is no data transfer from calling-function to called-function (parameters) but there is data transfer from called function to calling-function (return value).
- The execution control jumps from calling-function to called function and executes called function, and finally comes back to the calling function along with a return value.
- For example, consider the following program...

```
def add():
    a = int(input("enter a"))
    b = int(input("enter b"))
    c = a + b
    return c

c = add()
print(c)
```

```
Output :
    enter a 10
    enter b 20
    30
```

Function with Parameters and with Return value

- In this type of functions there is data transfer from calling-function to called-function (parameters) and also from called function to calling-function (return value).
- The execution control jumps from calling-function to called function along with parameters and executes called function, and finally comes back to the calling function along with a return value

```
def add(a,b):
    c = a + b
    return c

a = int(input("enter a"))
b = int(input("enter b"))
c = add(a,b)
print(c)
```

```
Output :
    enter a 10
    enter b 20
    30
```

Unit-III Questions

1. Explain 4 different function prototypes with an example programs
2. Explain built in functions with suitable examples
3. Write a short note on return statement in function with an example program.
4. Explain Commonly used modules with an example programs
5. Write a short note on formatted strings
6. Explain built-in functions with examples
7. Differentiate between local and global variables with suitable examples
8. Define Function. Explain with syntax how to create a user-defined functions and how to call the user -defined functions / how to call user defined functions from the main function
9. Explain about default arguments , *args and **kwargs
10. Write a short note on Command line arguments with an example program
11. Explain strings in detail (Creating and accessing)
12. Explain string operations in detail.

Explain string methods with an