Python Tutorial (Basics)

CAP 4630 -Artificial Intelligence

Instructor : Sam Ganzfried
(sganzfri@cis.fiu.edu)
Presented by: Farzana Beente Yusuf
(fyusu003@fiu.edu)

FLORIDA
INTERNATIONAL
UNIVERSITY

Introduction

Developed by Guido van Rossum in the early 1990s.

Features:

- High-level powerful programming language
- Interpreted language
- Object-oriented
- Portable
- Easy to learn & use
- Open source

Uses:

- Internet Scripting
- Database Programming
- Image Processing
- Artificial Intelligence

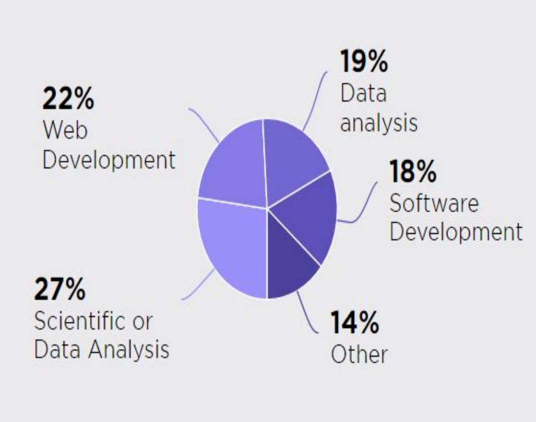


Fig: Percentage usage of Python [6]

Environment setup (Basic installation)

Major versions in use : Python 2.7.x and Python 3.x

Python 2.x is legacy, Python 3.x is the present and future of the language

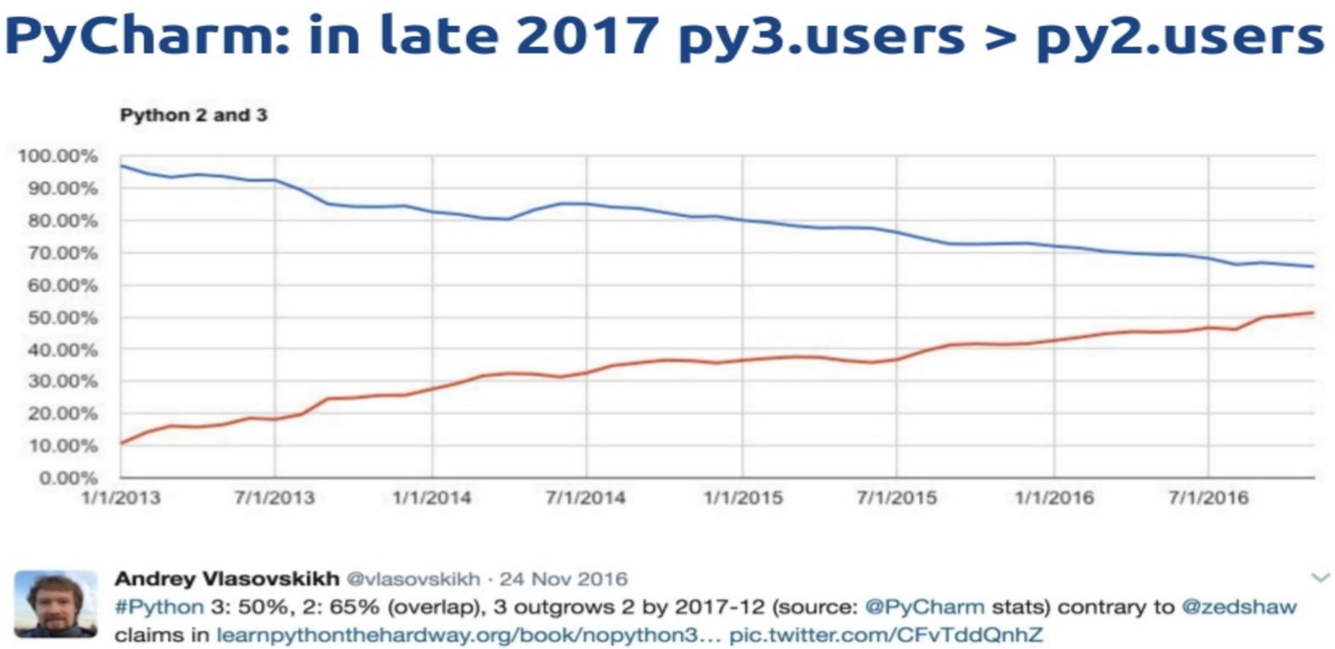
Windows:

Binaries of latest version of Python 3 (Python 3.5.1) are available on this download page [4] or install using **Anaconda** Package Manager (**Recommended**) [7]

Linux and Mac:

Follow the instructions from https://www.tutorialspoint.com/python3/python_environment.htm [2]

Python 2 vs 3 users in PyCharm [5]



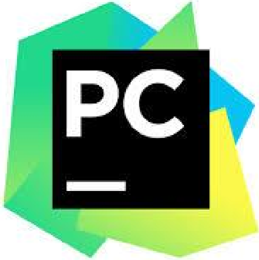
PyCharm: in late 2017 py3.users > py2.users

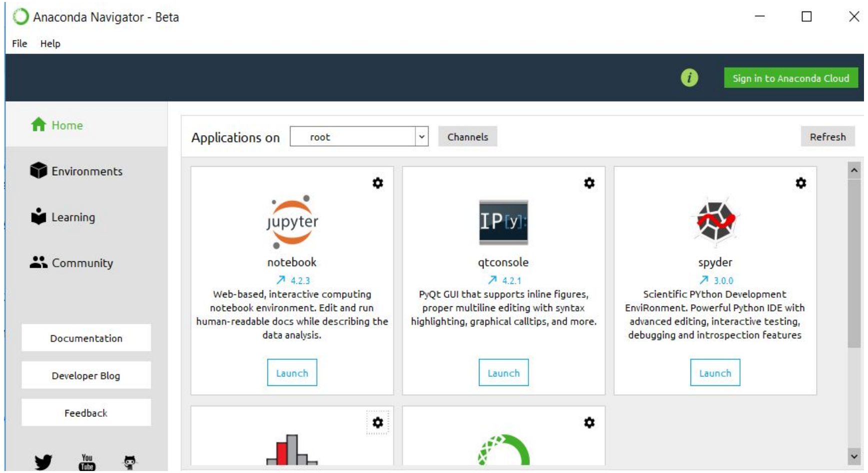
Andrey Vlasovskikh @vlasovskikh · 24 Nov 2016

#Python 3: 50%, 2: 65% (overlap), 3 outgrows 2 by 2017-12 (source: @PyCharm stats) contrary to @zedshaw claims in learnpythonthehardway.org/book/nopython3… pic.twitter.com/CFvTddQnhZ

Tools and IDE (Advanced installation)

- ➔ Download and install Pycharm Community edition - Free , No license required https://www.jetbrains.com/pycharm/download/#section=windows [6]
- ➔ Jupyter (Interactive python shell) [Anaconda] -http://jupyter.org/
- ➔ Spyder [Anaconda]
- ➔ Notepad++ - https://notepad-plus-plus.org/





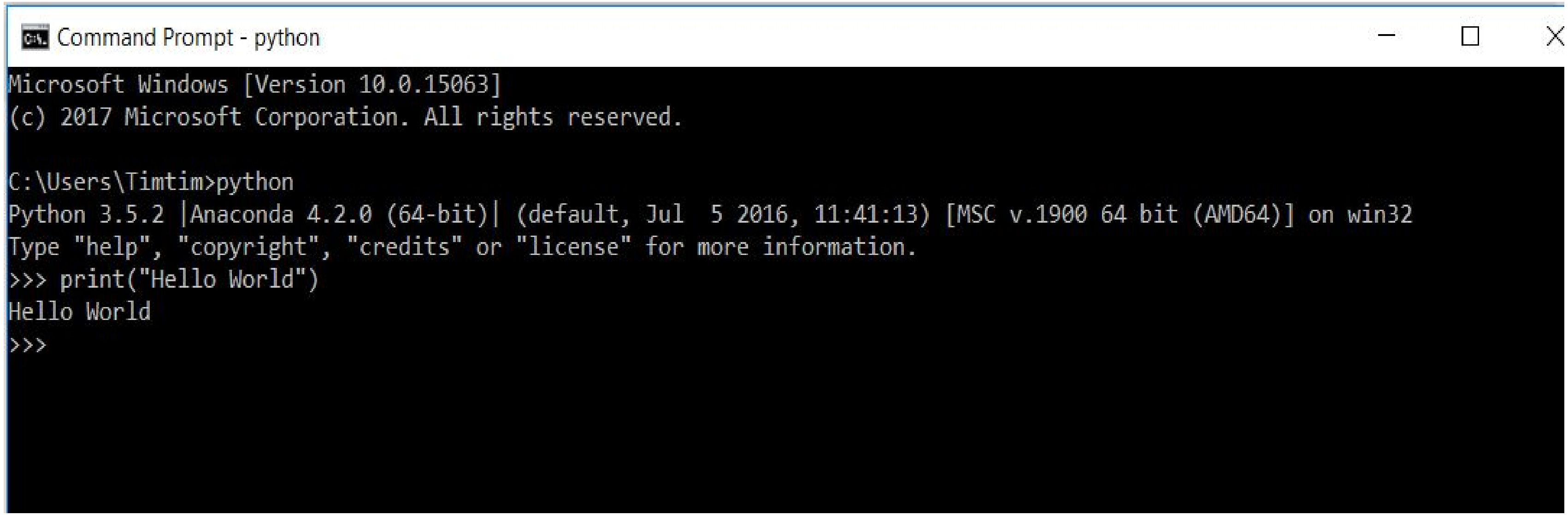
notebook
4.2.3
Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

qtconsole
4.2.1
PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

spyder
3.0.0
Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

First Program(Interactive mode)

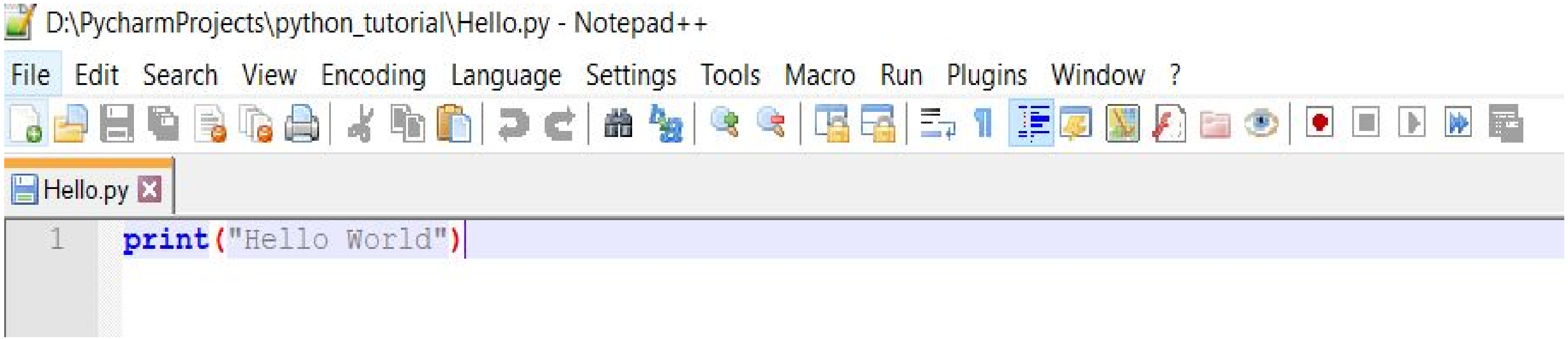
Enter **python** in the command line/terminal. Start coding right away in the interactive interpreter.



```
Microsoft Windows [Version 10.0.15063]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Users\Timtim>python
Python 3.5.2 |Anaconda 4.2.0 (64-bit)| (default, Jul  5 2016, 11:41:13) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

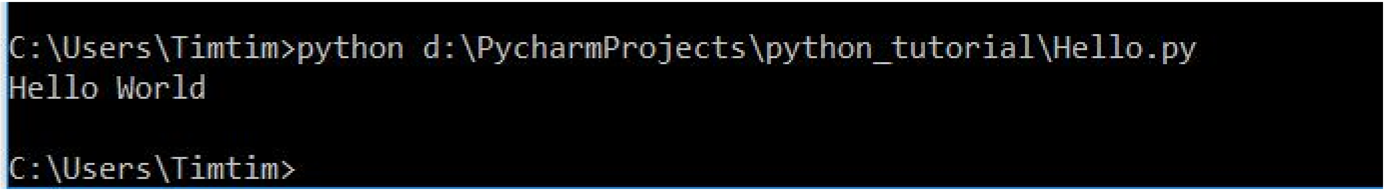
Scripting mode

Example (Hello.py):


D:\PycharmProjects\python_tutorial\Hello.py - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
Hello.py

```python
print("Hello World")
```

In Terminal:



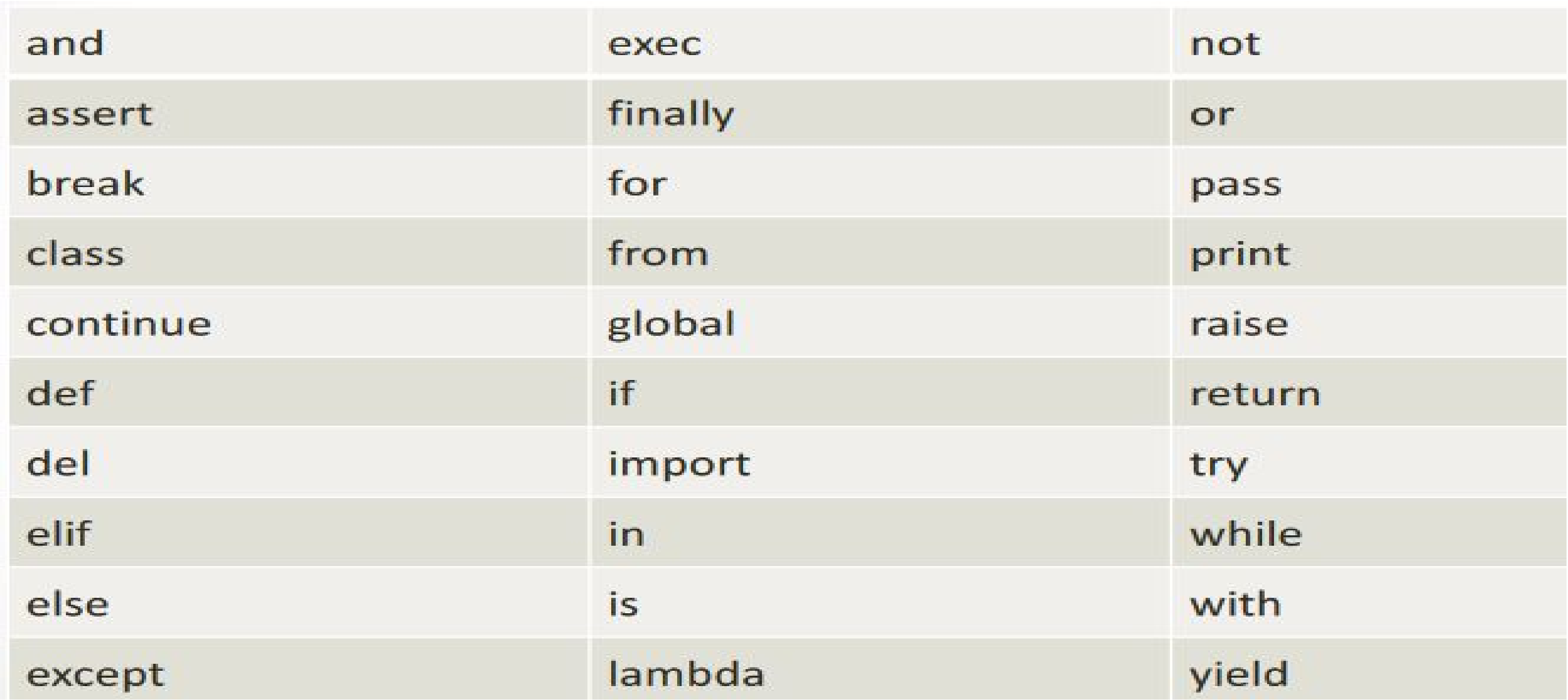
```
C:\Users\Timtim>python d:\PycharmProjects\python_tutorial\Hello.py
Hello World

C:\Users\Timtim>
```

Basic Syntax

Identifiers:

- Name used to identify a variable, function, class, module or other object
- Starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9)
- Punctuation characters such as @, $,and % within identifiers are not allowed
- Case sensitive programming language. Manpower and manpower are two different identifiers in Python

Reserved Keywords



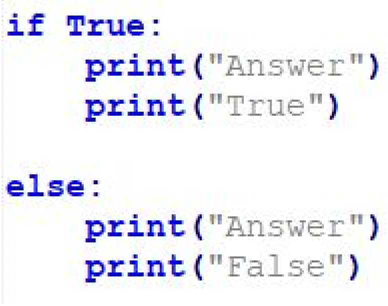
and	exec	not
assert	finally	or
break	for	pass
class	from	print
continue	global	raise
def	if	return
del	import	try
elif	in	while
else	is	with
except	lambda	yield

Lines and Indentation

Python does not use braces({}) to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation

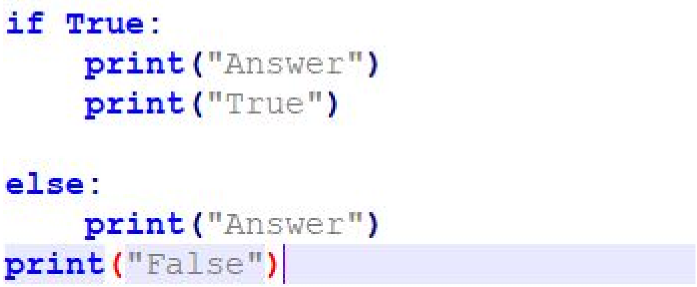
The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. Use of tab is recommended to be consistent and make the code more readable.

Correct implementation:



```
if True:
    print("Answer")
    print("True")

else:
    print("Answer")
    print("False")
```

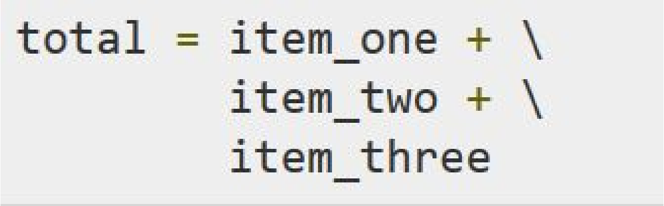
Erroneous implementation:



```
if True:
    print("Answer")
    print("True")

else:
    print("Answer")
print("False")
```

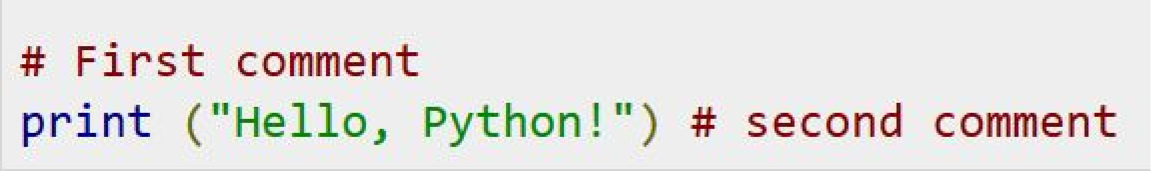
Misc.

Multi-Line Statements : Statements in Python typically end with a new line. Python, however, allows the use of the line continuation character (\) to denote that the line should continue.



```
total = item_one + \
        item_two + \
        item_three
```

Comments :A hash sign (#) that is not inside a string literal is the beginning of a comment.



```
# First comment
print ("Hello, Python!") # second comment
```

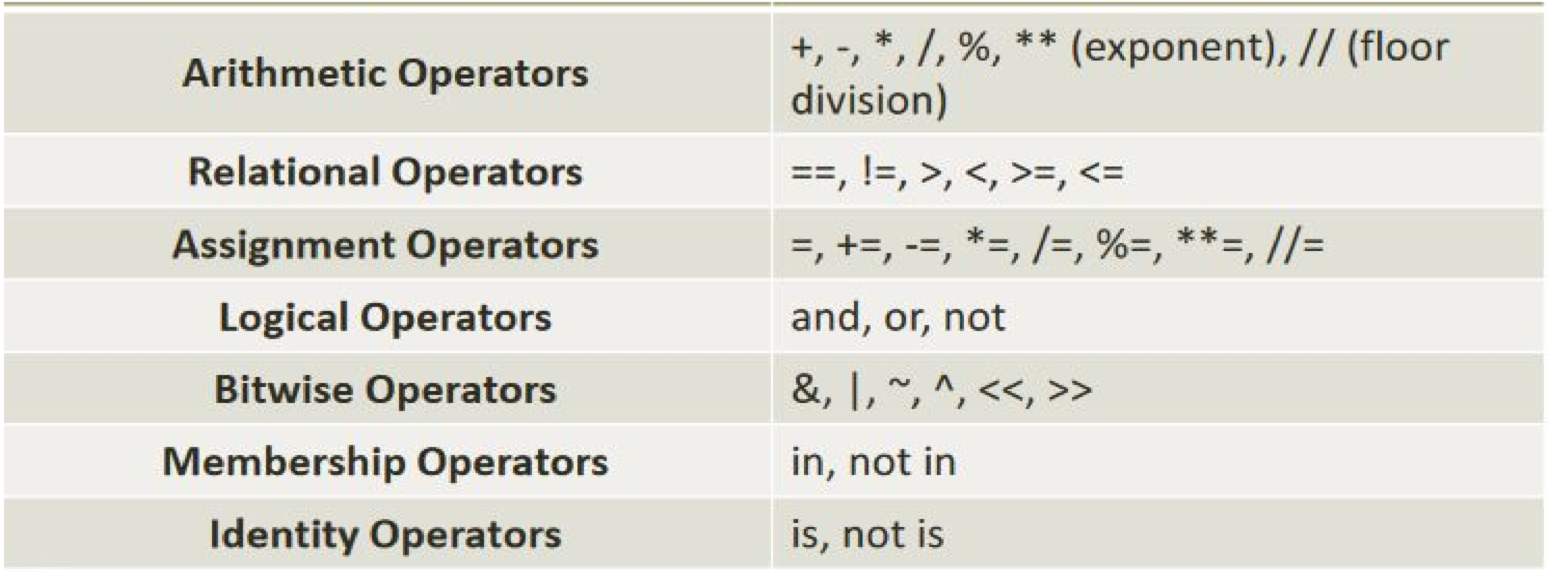
Variable types

Variables are nothing but reserved memory locations to store values. It means that when you create a variable, you reserve some space in the memory.

Standard Data Type	**Example**
Numbers	counter=100, miles=100.0
String	firstname= “john”
List (like Array in C,Java)	heights=[“John”, 157]]
Tuple (Read only lists)	heights = ("john', 157.2)
Dictionary (like hash-table)	Person={'name': 'john', 'dept': 'sales', 'code': 6734}

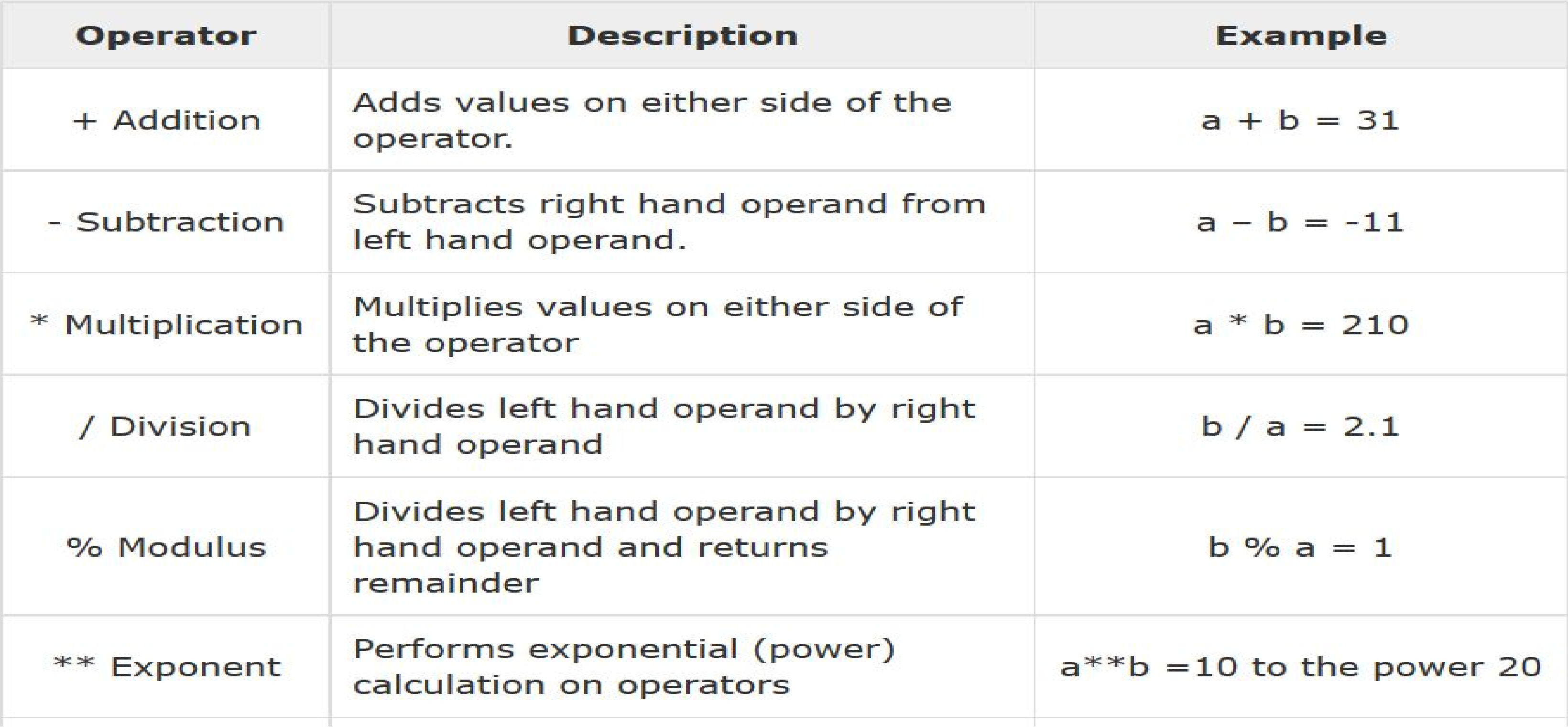
Type of Operators

Operators are the constructs, which can manipulate the value of operands. Python language supports the following types of operators –



Arithmetic Operators	+, -, *, /, %, ** (exponent), // (floor division)
Relational Operators	==, !=, >, <, >=, <=
Assignment Operators	=, +=, -=, *=, /=, %=, **=, //=
Logical Operators	and, or, not
Bitwise Operators	&, |, ~, ^, <<, >>
Membership Operators	in, not in
Identity Operators	is, not is

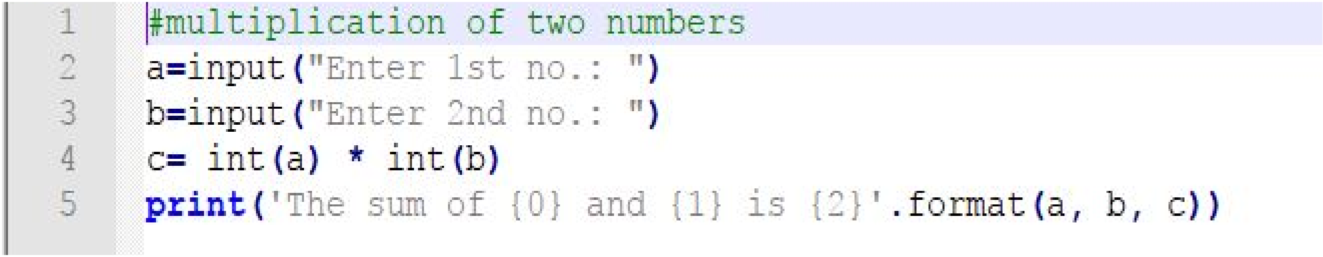
Basic Operators



Operator	Description	Example
+ Addition	Adds values on either side of the operator.	a + b = 31
- Subtraction	Subtracts right hand operand from left hand operand.	a – b = -11
* Multiplication	Multiplies values on either side of the operator	a * b = 210
/ Division	Divides left hand operand by right hand operand	b / a = 2.1
% Modulus	Divides left hand operand by right hand operand and returns remainder	b % a = 1
** Exponent	Performs exponential (power) calculation on operators	a**b =10 to the power 20

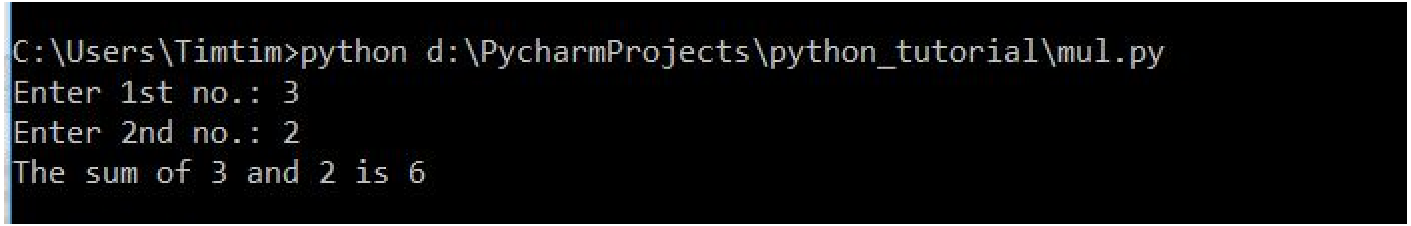
Using Multiplication operator

mul.py



```
#multiplication of two numbers
a=input("Enter 1st no.: ")
b=input("Enter 2nd no.: ")
c= int(a) * int(b)
print('The sum of {0} and {1} is {2}'.format(a, b, c))
```

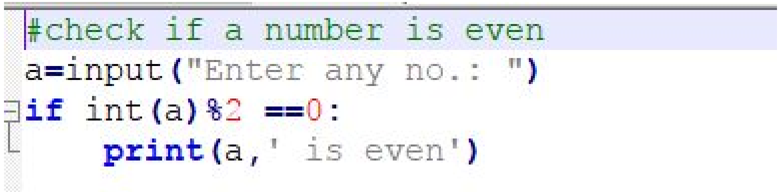
Console output:



```
C:\Users\Timtim>python d:\PycharmProjects\python_tutorial\mul.py
Enter 1st no.: 3
Enter 2nd no.: 2
The sum of 3 and 2 is 6
```

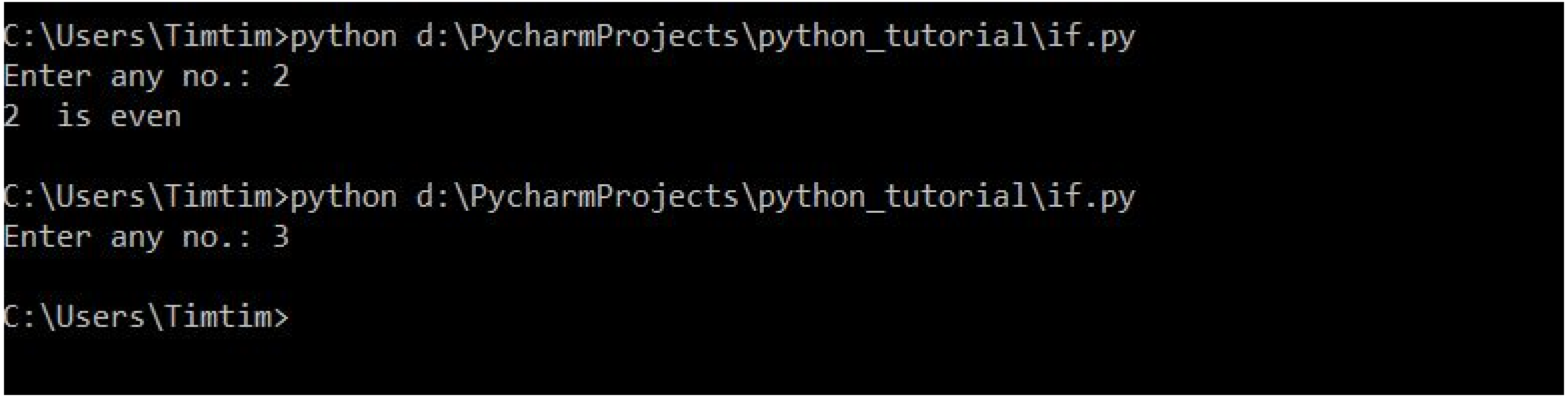
Decision Making (if statement)

If.py



```
#check if a number is even
a=input("Enter any no.: ")
if int(a)%2 ==0:
    print(a,' is even')
```

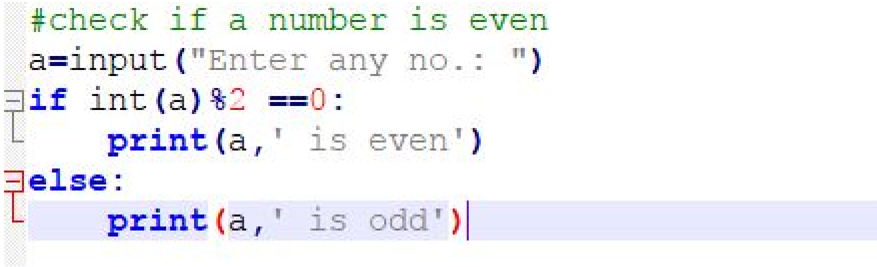
Console Output:



```
C:\Users\Timtim>python d:\PycharmProjects\python_tutorial\if.py
Enter any no.: 2
2  is even

C:\Users\Timtim>python d:\PycharmProjects\python_tutorial\if.py
Enter any no.: 3

C:\Users\Timtim>
```

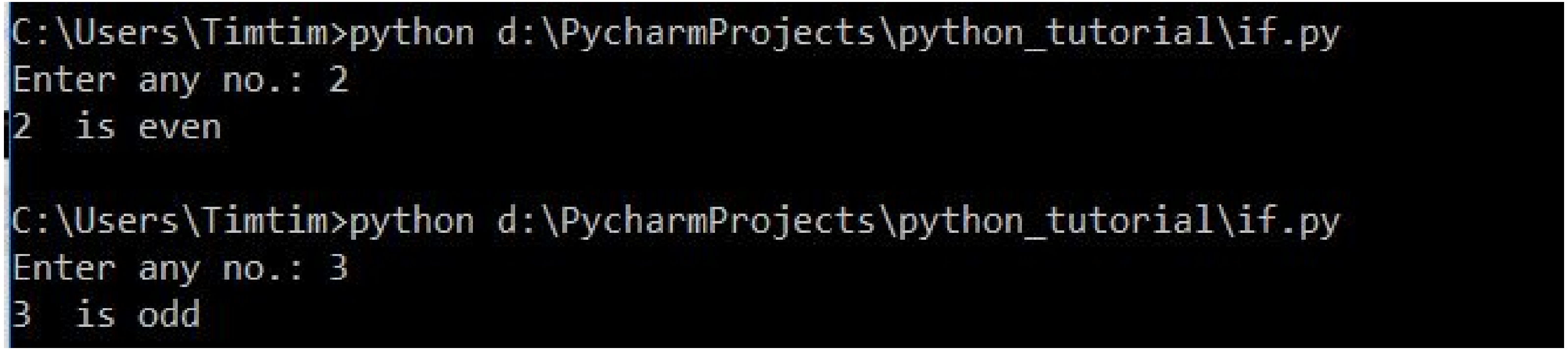
Decision Making (if -else statement)

IfElse.py



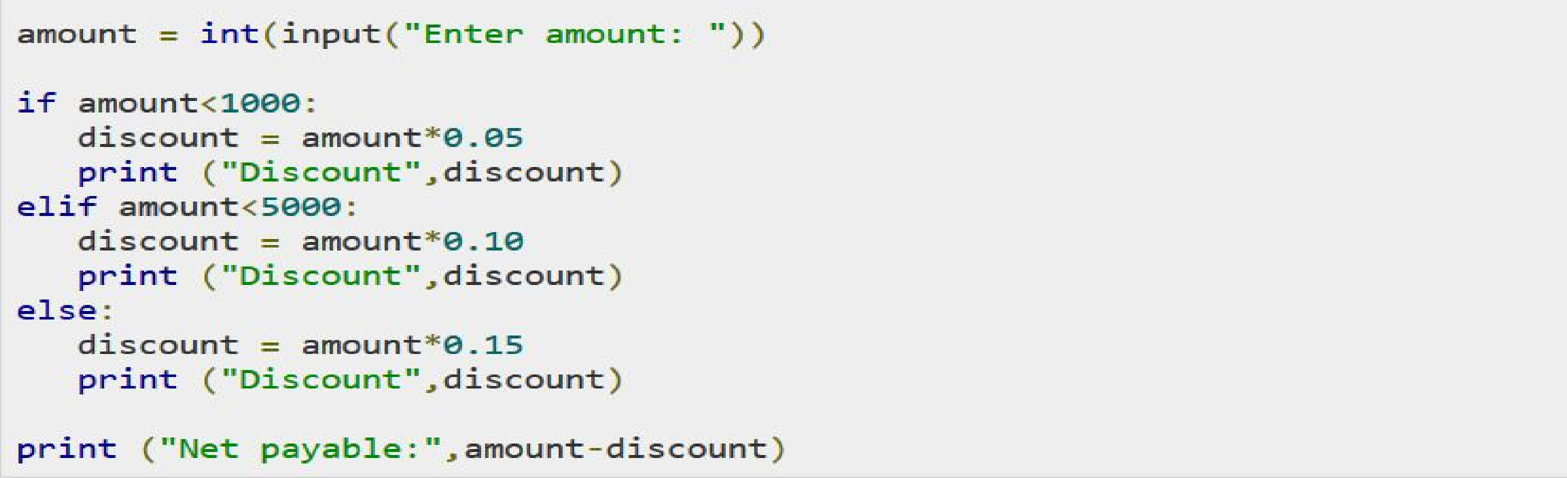
```
#check if a number is even
a=input("Enter any no.: ")
if int(a)%2 ==0:
    print(a,' is even')
else:
    print(a,' is odd')
```

Console Output:



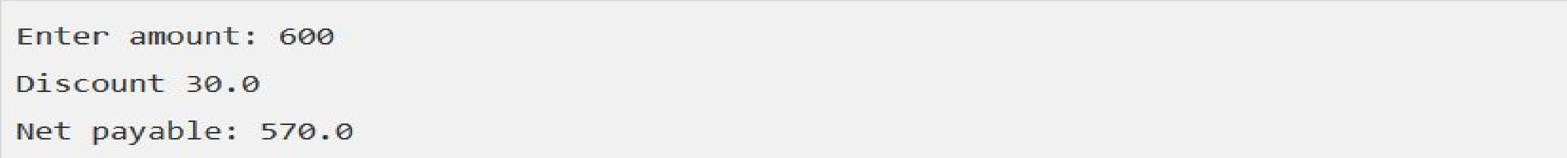
```
C:\Users\Timtim>python d:\PycharmProjects\python_tutorial\if.py
Enter any no.: 2
2  is even

C:\Users\Timtim>python d:\PycharmProjects\python_tutorial\if.py
Enter any no.: 3
3  is odd
```

Decision Making (elif)



```
amount = int(input("Enter amount: "))

if amount<1000:
   discount = amount*0.05
   print ("Discount",discount)
elif amount<5000:
   discount = amount*0.10
   print ("Discount",discount)
else:
   discount = amount*0.15
   print ("Discount",discount)

print ("Net payable:",amount-discount)
```

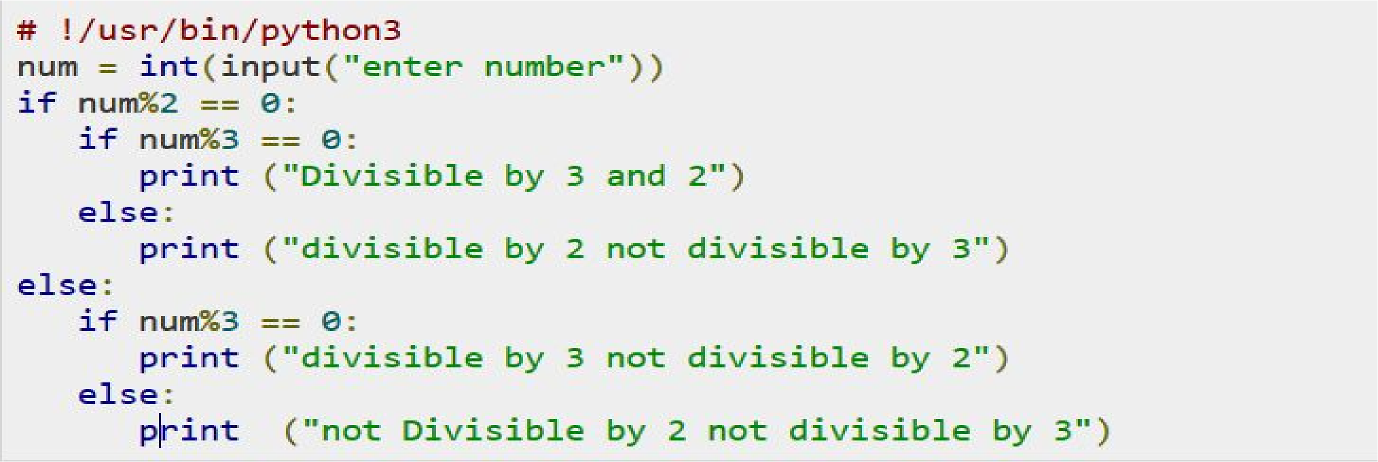
When the above code is executed, it produces the following result –



```
Enter amount: 600

Discount 30.0

Net payable: 570.0
```

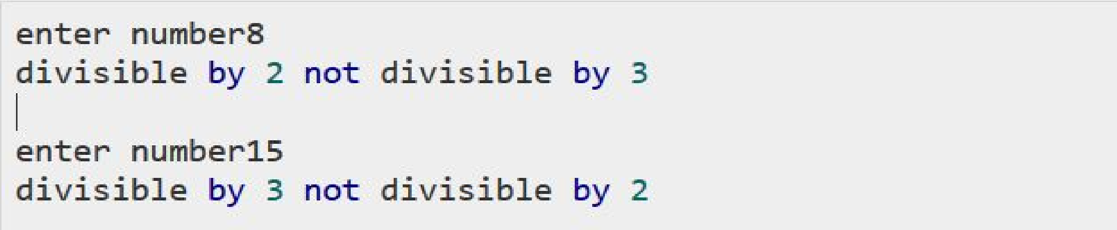
Decision Making (Nested if)

Ex:



```
# !/usr/bin/python3
num = int(input("enter number"))
if num%2 == 0:
    if num%3 == 0:
        print ("Divisible by 3 and 2")
    else:
        print ("divisible by 2 not divisible by 3")
else:
    if num%3 == 0:
        print ("divisible by 3 not divisible by 2")
    else:
        print  ("not Divisible by 2 not divisible by 3")
```

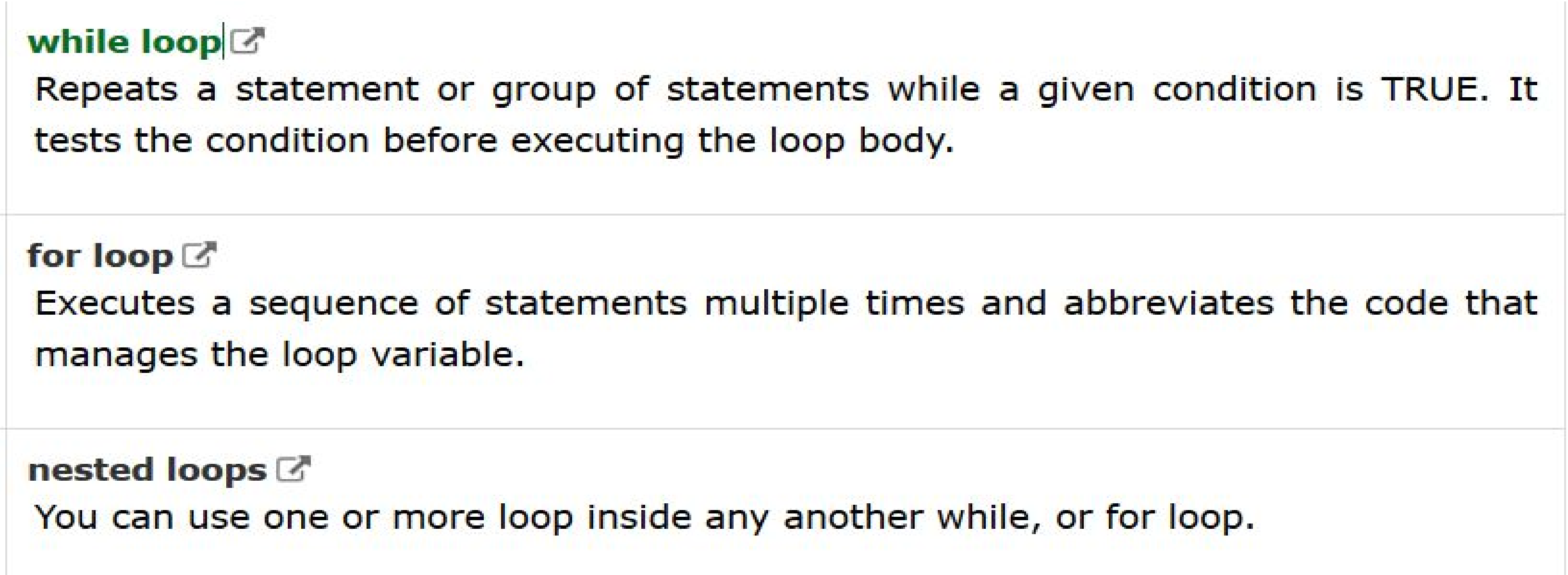
Output:



```
enter number8
divisible by 2 not divisible by 3

enter number15
divisible by 3 not divisible by 2
```

Loops

In general, statements are executed sequentially − The first statement in a function is executed first, followed by the second, and so on. There may be a situation when it’s necessary to execute a block of code several number of times.



while loop

Repeats a statement or group of statements while a given condition is TRUE. It tests the condition before executing the loop body.

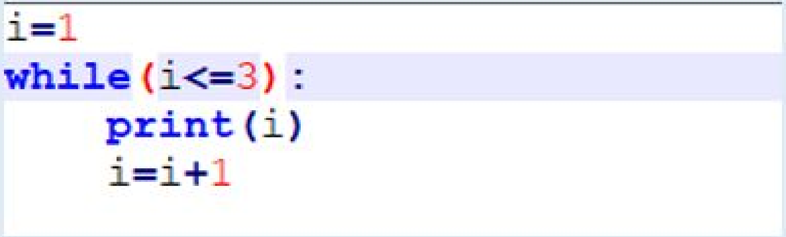
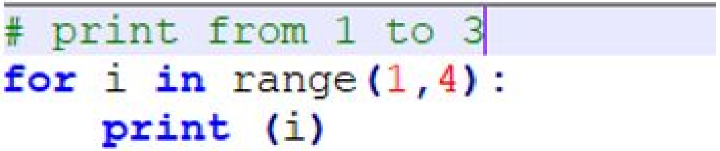
for loop

Executes a sequence of statements multiple times and abbreviates the code that manages the loop variable.

nested loops

You can use one or more loop inside any another while, or for loop.

Loops

<table>
<tr><th>Types</th><th>Example</th><th>Output</th></tr>
<tr><td>while</td><td>
<pre>
i=1
while(i<=3):
    print(i)
    i=i+1
</pre></td><td>1
2
3</td></tr>
<tr><td>for</td><td>
<pre>
print from 1 to 3
for i in range(1,4):
    print (i)
</pre></td><td>1
2
3</td></tr>
</table>

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.

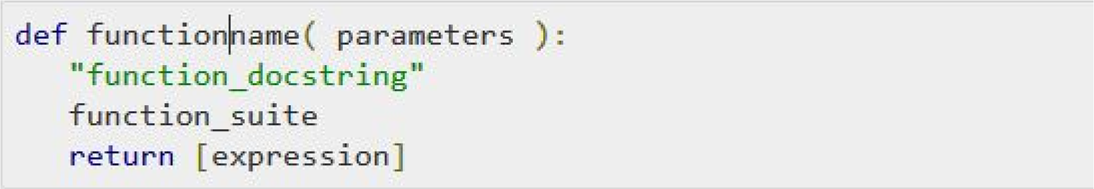
Defining a Function

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.
- 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.

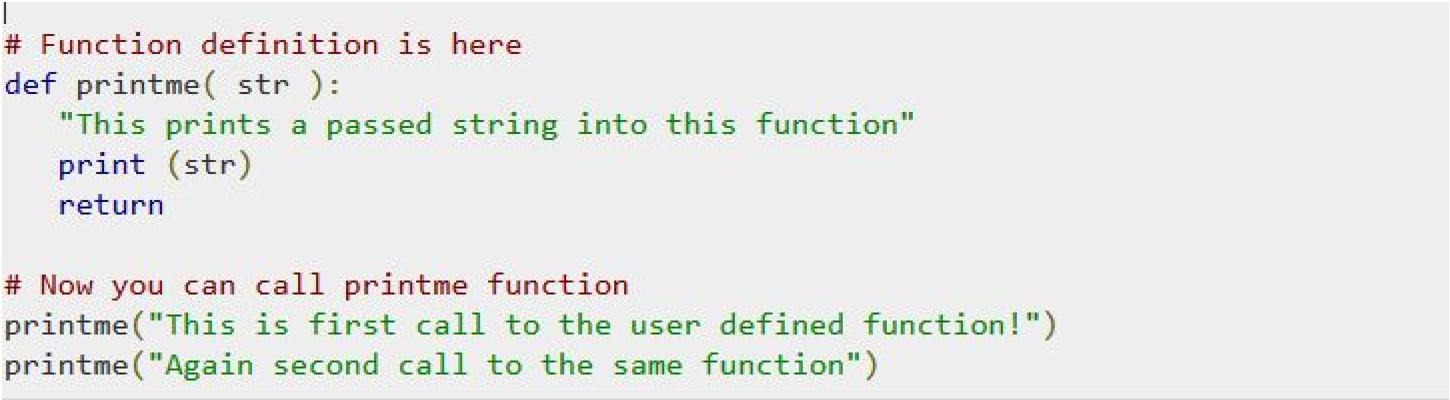
Function Call

Function definition:



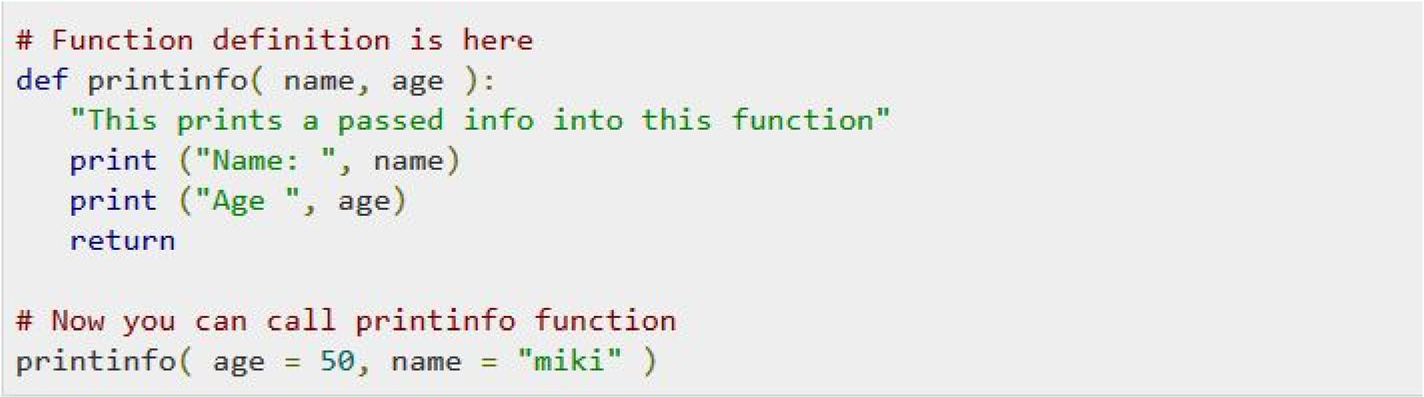
```
def functionname( parameters ):
   "function_docstring"
   function_suite
   return [expression]
```

Example:

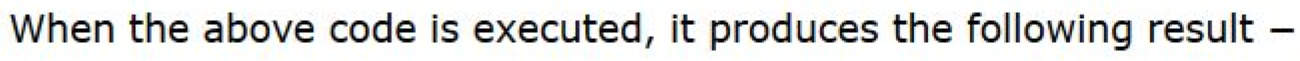


```
# Function definition is here
def printme( str ):
   "This prints a passed string into this function"
   print (str)
   return

# Now you can call printme function
printme("This is first call to the user defined function!")
printme("Again second call to the same function")
```

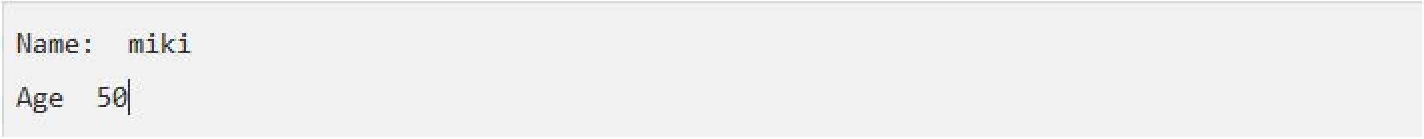
Function- Arguments



```
# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print ("Name: ", name)
   print ("Age ", age)
   return

# Now you can call printinfo function
printinfo( age = 50, name = "miki" )
```



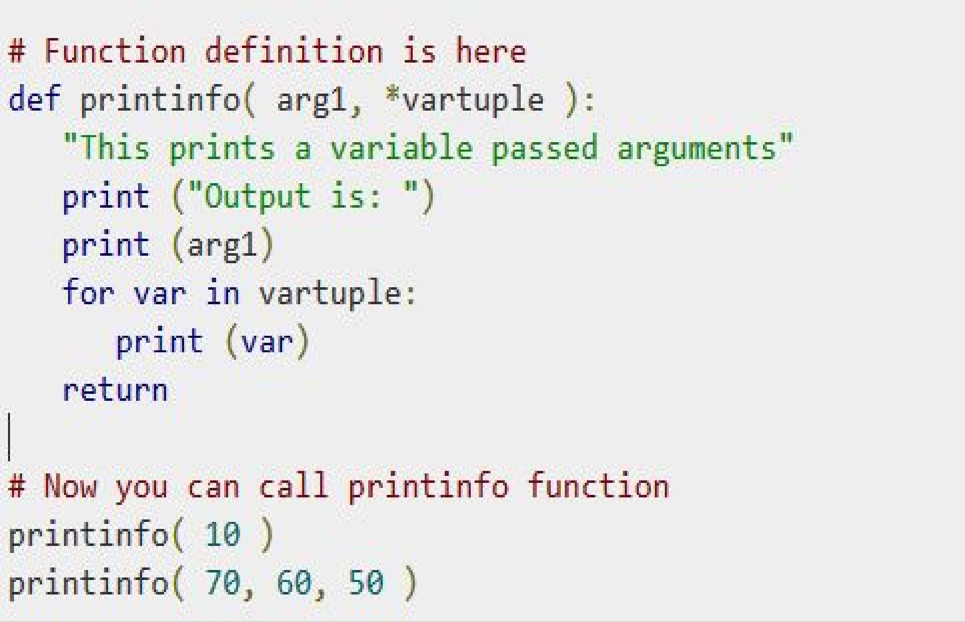
When the above code is executed, it produces the following result –



```
Name:  miki

Age  50
```

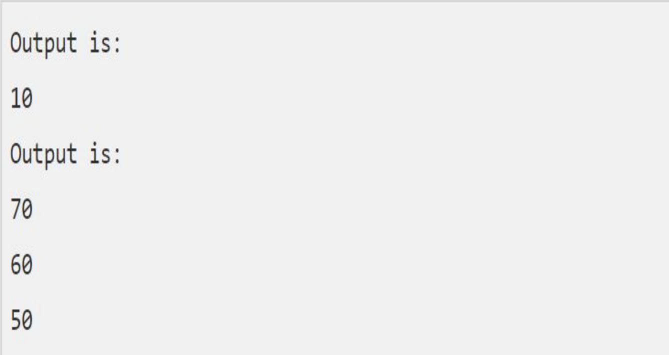
Function- Variable length arguments

Example



```
# Function definition is here
def printinfo( arg1, *vartuple ):
   "This prints a variable passed arguments"
   print ("Output is: ")
   print (arg1)
   for var in vartuple:
      print (var)
   return

# Now you can call printinfo function
printinfo( 10 )
printinfo( 70, 60, 50 )
```

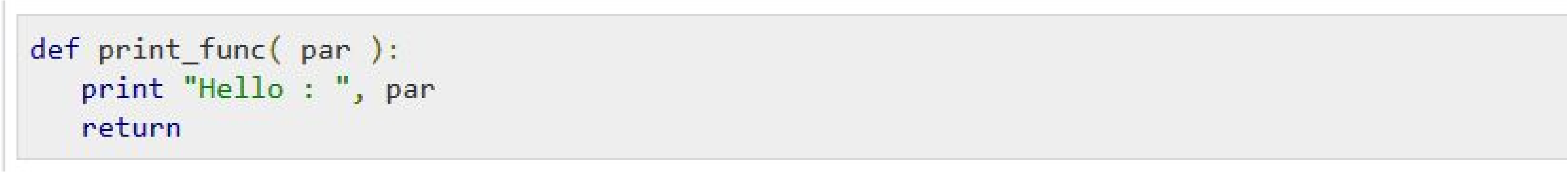
Output:



```
Output is:
10
Output is:
70
60
50
```

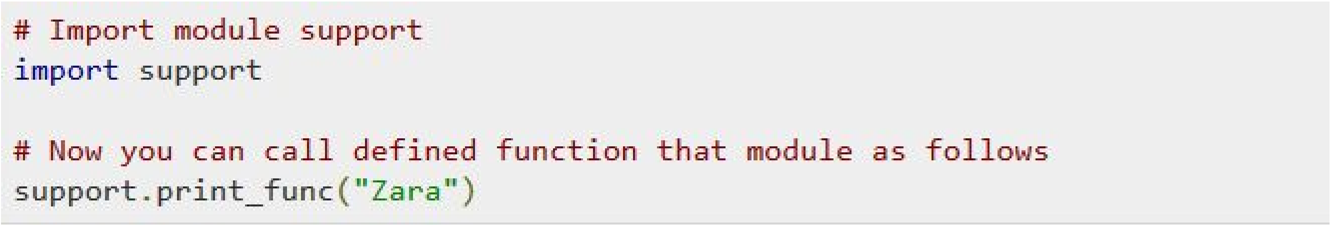
Modules

A module allows to logically organize Python code. Grouping related code into a module makes the code easier to understand and use. A module is a Python object with arbitrarily named attributes. Python code for a module named aname normally resides in a file FILE_NAME.py. Here is an example of a simple module, support.py –



```
def print_func( par ):
   print "Hello : ", par
   return
```

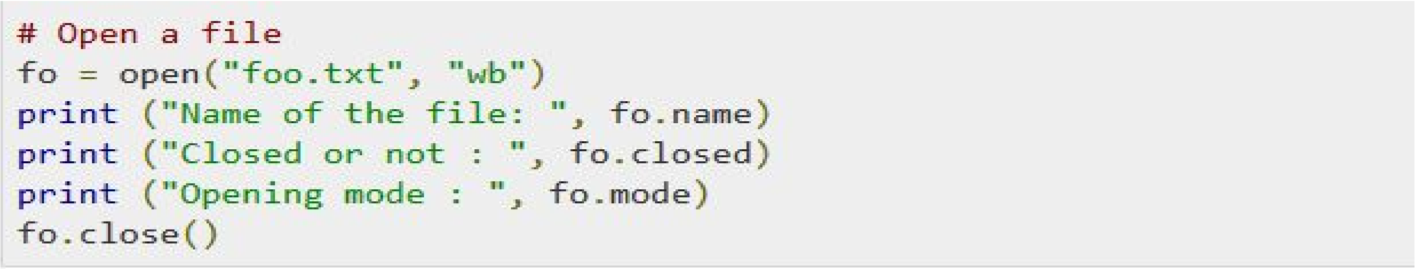
When the interpreter encounters an import statement, it imports the module if the module is present in the search path.



```
# Import module support
import support

# Now you can call defined function that module as follows
support.print_func("Zara")
```

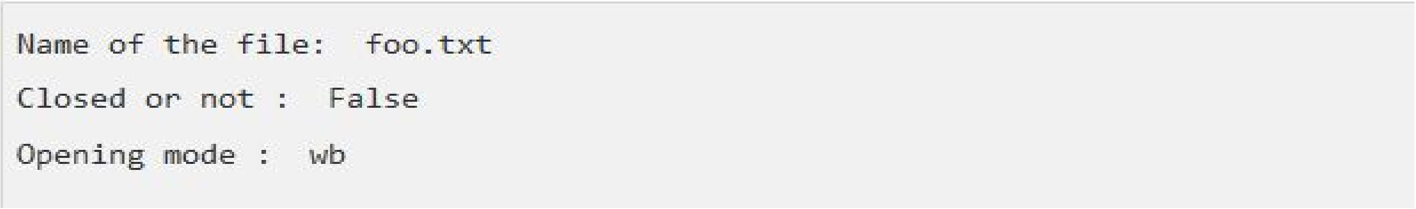
File I/O

Python provides basic functions and methods necessary to manipulate files by default. Most of the file manipulation can be completed using a **file** object.



```
# 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)
fo.close()
```

This produces the following result –

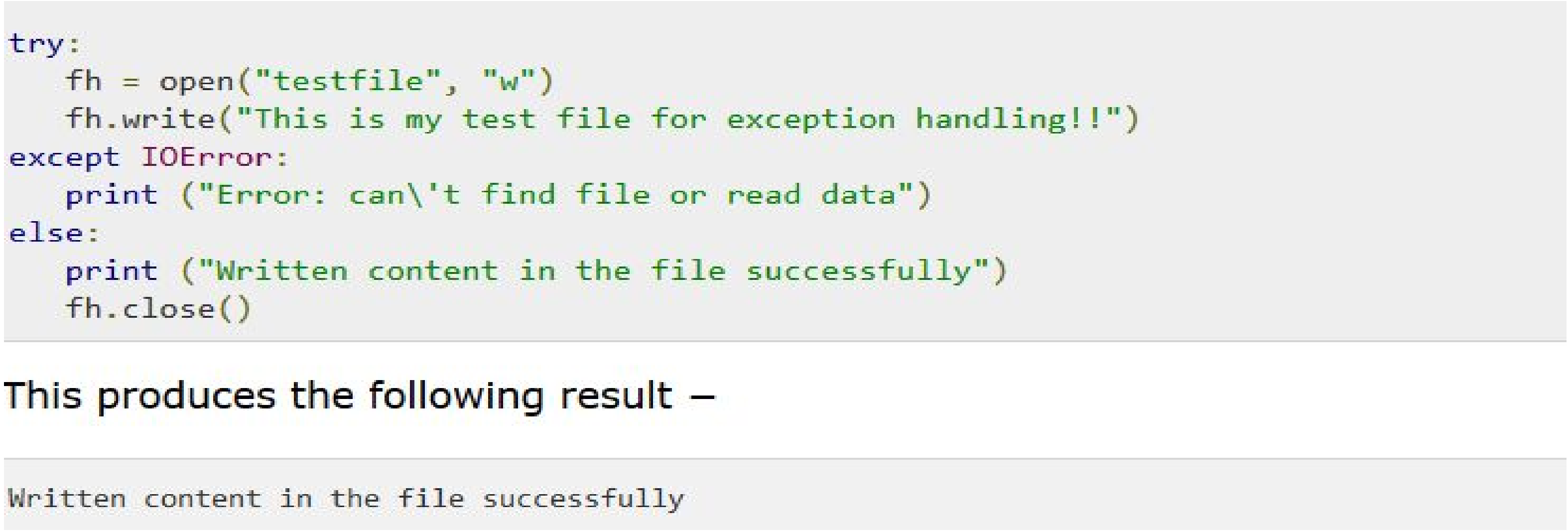


```
Name of the file:  foo.txt

Closed or not :  False

Opening mode :  wb
```

Exceptions

Python provides two very important features to handle any unexpected error in your Python programs and to add debugging capabilities in them –

- **Exception Handling** - Basic error handling i.e Division by zero, File Read/Write error etc.
- **Assertions** - Advanced topic



```
try:
   fh = open("testfile", "w")
   fh.write("This is my test file for exception handling!!")
except IOError:
   print ("Error: can\'t find file or read data")
else:
   print ("Written content in the file successfully")
   fh.close()
```

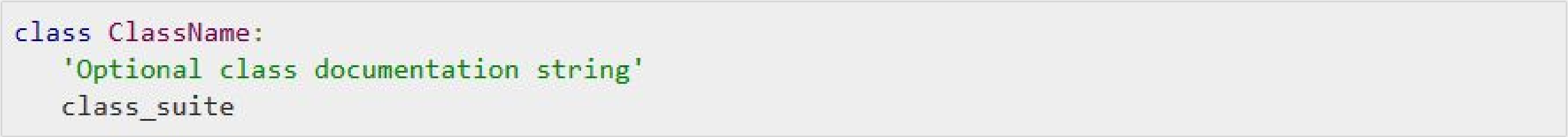
This produces the following result –

```
Written content in the file successfully
```

Class (OOP)

Python has been an object-oriented language since the time it existed. Due to this, creating and using classes and objects are downright easy.

The class statement creates a new class definition. The name of the class immediately follows the keyword class followed by a colon as follows –

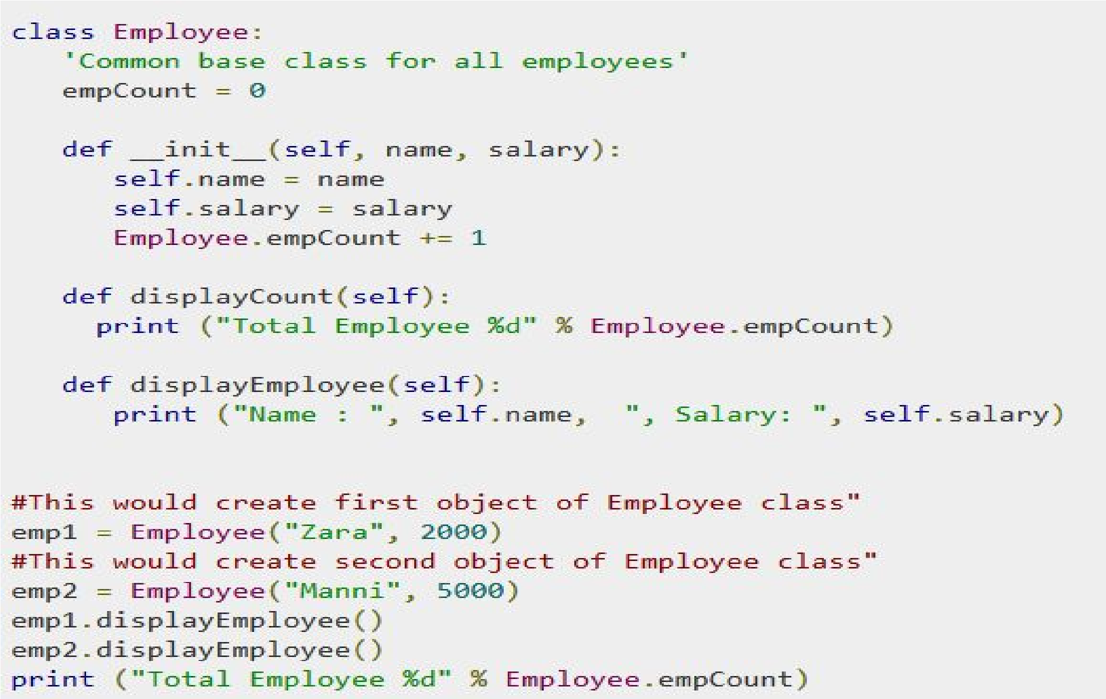


```
class ClassName:
   'Optional class documentation string'
   class_suite
```

- The class has a documentation string, which can be accessed via ***ClassName.__doc__***.
- The ***class_suite*** consists of all the component statements defining class members, data attributes and functions.

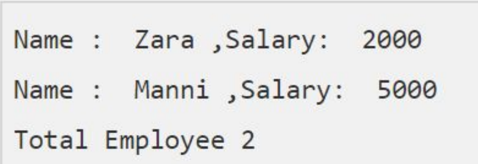
Class (OOP)

Example Code



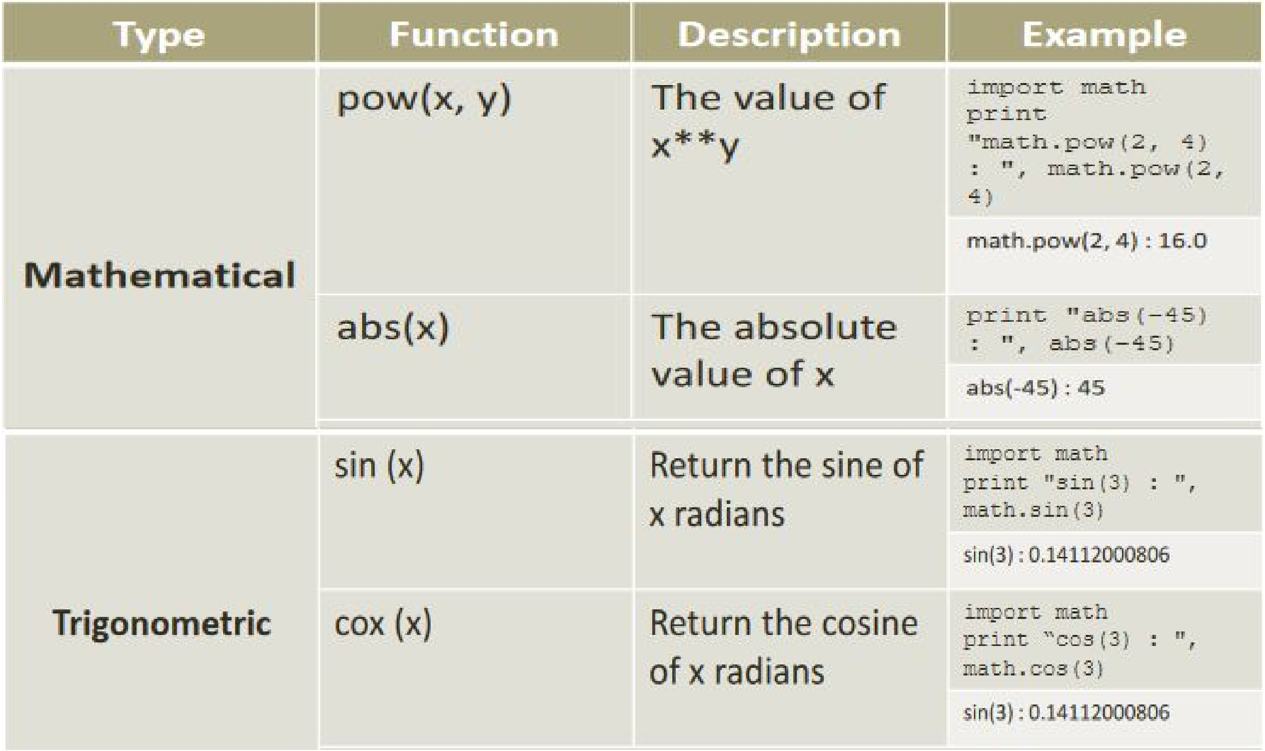
```
class Employee:
   'Common base class for all employees'
   empCount = 0

   def __init__(self, name, salary):
      self.name = name
      self.salary = salary
      Employee.empCount += 1

   def displayCount(self):
     print ("Total Employee %d" % Employee.empCount)

   def displayEmployee(self):
      print ("Name : ", self.name,  ", Salary: ", self.salary)


#This would create first object of Employee class"
emp1 = Employee("Zara", 2000)
#This would create second object of Employee class"
emp2 = Employee("Manni", 5000)
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

Output



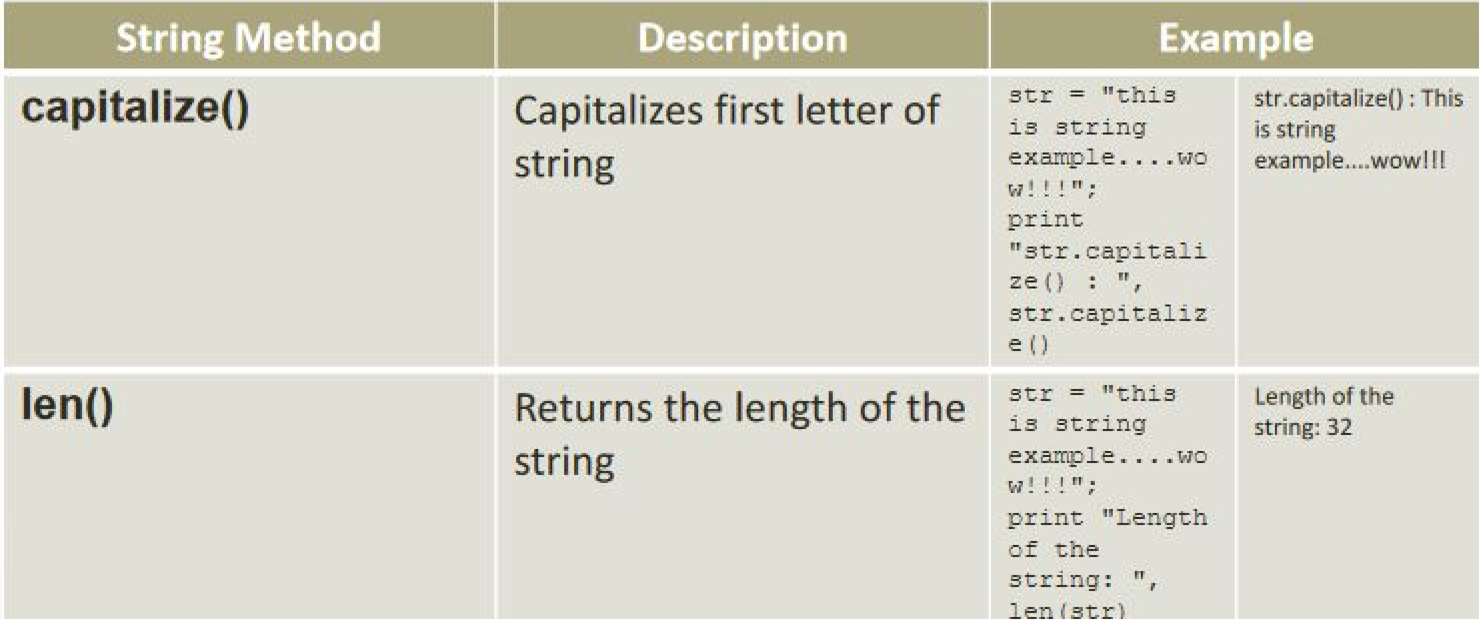
```
Name :  Zara ,Salary:  2000
Name :  Manni ,Salary:  5000
Total Employee 2
```

Numbers - Functions [3]



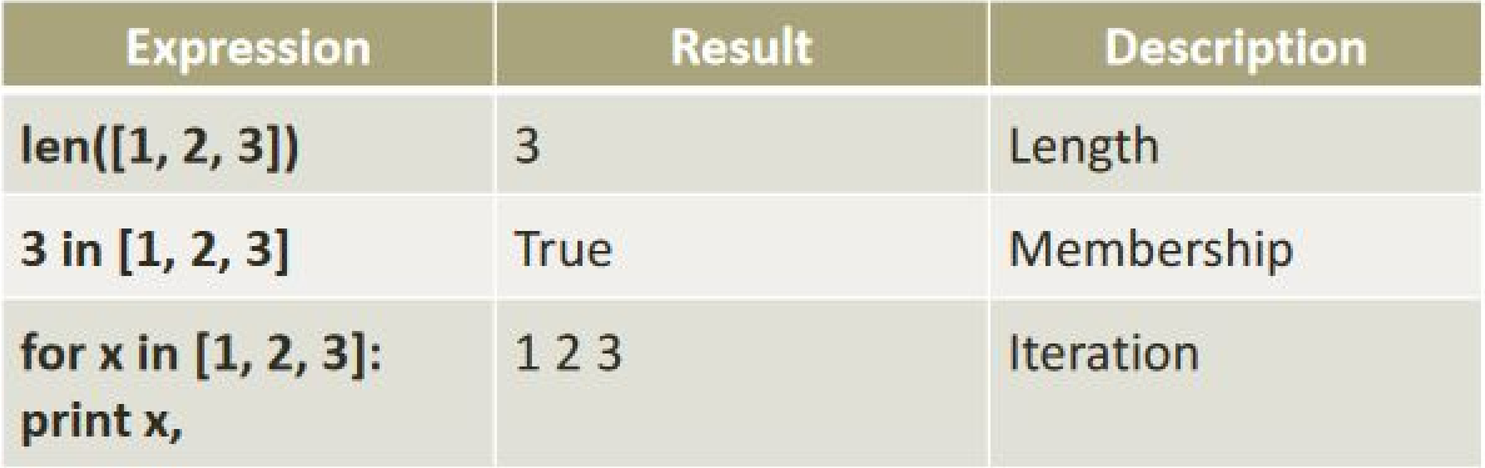
Type	Function	Description	Example
Mathematical	pow(x, y)	The value of x**y	import math print "math.pow(2, 4) : ", math.pow(2, 4) math.pow(2, 4) : 16.0
	abs(x)	The absolute value of x	print "abs(-45) : ", abs(-45) abs(-45) : 45
Trigonometric	sin (x)	Return the sine of x radians	import math print "sin(3) : ", math.sin(3) sin(3) : 0.14112000806
	cox (x)	Return the cosine of x radians	import math print “cos(3) : ", math.cos(3) sin(3) : 0.14112000806

String - Functions [3]



String Method	Description	Example	
capitalize()	Capitalizes first letter of string	`str = "this is string example....wow!!!"; print "str.capitalize() : ", str.capitalize()`	str.capitalize() : This is string example....wow!!!
len()	Returns the length of the string	`str = "this is string example....wow!!!"; print "Length of the string: ", len(str)`	Length of the string: 32

List - Operators and functions [3]



Expression	Result	Description
len([1, 2, 3])	3	Length
3 in [1, 2, 3]	True	Membership
for x in [1, 2, 3]: print x,	1 2 3	Iteration

Advanced Libraries

★ **Numpy-** support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays [12]

★ **Scipy-** provides many user-friendly and efficient numerical routines such as routines for numerical integration and optimization [11]

★ **Matplotlib** - 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms [9]

★ **Pandas-** high-performance, easy-to-use data structures and data analysis tools [8]

★ **Scikit Learn(sklearn)** - Machine learning libraries built on top of NumPy, SciPy and matplotlib [10]

For more reference: https://wiki.python.org/moin/UsefulModules

Resources and References

1. http://learnpython.org/
2. **https://www.tutorialspoint.com/python3/**
3. https://www.slideshare.net/MoitraSabya/python-basic-77418012?qid=e78f7c5d-51b8-4a70-a48d-32b1aeb35548&v=&b=&from_search=1
4. https://www.python.org/
5. https://www.slideshare.net/mariczhuck/austin-python-meetup-2017-how-to-stop-worrying-and-start-a-project-with-python-3?qid=83f645fe-c735-4b00-aeee-c9224c8294d7&v=&b=&from_search=11
6. https://www.jetbrains.com/pycharm/
7. https://www.continuum.io/downloads
8. http://pandas.pydata.org/
9. https://matplotlib.org/
10. http://scikit-learn.org/stable/
11. https://www.scipy.org/
12. http://www.numpy.org/