Python Programming

Vasudevan T V

Course Objectives

- To develop proficiency in python programming language
- To understand the various data structures available in python
- To test and debug code written in python
- To implement object oriented programming concepts in python
- To develop web based applications using python

Module I
Introduction

- Created by Guido Van Rossum, a Dutch Programmer
- First released in 1991
- Named after the comedy series Monty Python's Flying Circus shown in BBC
- Guido Van Rossum was a big fan of this series
- Python 2.0 released in 2000 is not currently supported
- Python 3.0 released in 2008 is the currently supported version

Features of Python

- It is an interpreted, high level programming language
- It is a multi-paradigm programming language
- It fully supports structured programming and object oriented programming
- It partially supports functional programming and logic programming

Philosophy of Python

Its core philosophy is summarised in the document The Zen of Python, which contains 20 software principles. This includes

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Identifiers

- A python identifier is a name used to identify a variable, function, class, module or other object
- An identifier can contain letters (lower case or upper case), digits and underscores
- It must not start with a digit
- Python is a case sensitive programming language
- Hello and hello are different identifiers

Reserved Words (Keywords)

- keywords are having special meaning and they cannot be used as identifiers
- They contain lower case letters only
- and, as, assert, break, class, continue, def, del, elif, else, except, exec, finally, for, from, global, if, import, in, is, lambda, not, or, pass, print, raise, return, try, while, with, yield

Running Python

Method 1 - Interactive Mode

```
$ python3
```

Running Python

Method 1 - Interactive Mode

```
$ python3
Python 3.5.2 (default, July 17 2020, 14:04:10)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>>
```

Running Python

Method 1 - Interactive Mode

```
$ python3
Python 3.5.2 (default, July 17 2020, 14:04:10)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>> print("Good Morning")
```

Running Python

Method 1 - Interactive Mode

```
$ python3
Python 3.5.2 (default, July 17 2020, 14:04:10)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for
more information.
>>> print("Good Morning")
Good Morning
```

Running Python

Method 2 - Execute Program from Command Line

Store the program with the extension .py and execute it using python3 command

```
$ python3 salute.py
```

Running Python

Method 3 - Using a GUI Integrated Development Environment

- IDLE is a Python IDE for Linux distributions
- PythonWin is a Python IDE for Windows

Running Python

Method 4 - Run Python Online

- https://www.python.org/shell

python.org is the official website of python software foundation, a non-profit organisation that is devoted to the development of python

Input/Output Functions in Python

- input() is used to accept input from user
- It returns value as string
- print() is used to output values

Example

```
>>> text = input()
Python Programming
>>> print(text)
Python Programming
>>> print("Python Programming")
Python Programming
>>> print(12345)
12345
```

Input/Output Functions in Python

We can also display a prompt message to input

Example

```
>>> name = input("Enter your name:")
Enter your name: Vasudevan T V
>>> print(name)
Vasudevan T V
```

Arithmetic Operators

Operation	Operator
Addition	+
Subtraction	-
Multiplication	*
Division	/
Truncating Division	//
Exponentiation	**
Modulus	%
Negation	-

/ performs floating point division

// performs integer division

Precedence of Operations

1. Parentheses
2. Exponentiation
3. Negation
4. Multiplication, Division, Truncating Division, Modulus
5. Addition, Subtraction

Note

Operators with the same precedence are evaluated from left to right

Data Types

- Fundamental Data Types
- These are the basic data types
- Number
- Boolean
- String
- Collection Data Types
- They contain collection of elements
- List
- Tuple
- Set
- Dictionary

Number Data Types

- int(integer)
- float (floating point number)
- complex (complex number)

Number Data Types

```
>>> number = 100 # This is a comment
>>> type(number) # type() is used to display the data type
<type 'int'>
>>> number = -21.9
>>> type(number)
<type 'float'>
>>> number = 5 + 6j
>>> type(number)
>>> <type 'complex'>
```

Boolean Data Type

- They contain boolean values

Examples

```
>>> a = True
>>> type(a)
<class 'bool'>
>>> b = False
>>> type(b)
<class 'bool'>
```

String Data Type

- A string can be enclosed within single or double quotes

Examples

```
>>> string1 = 'Good Morning'
>>> type(string1)
<type 'str'>
>>> string2 = "Python Programming"
>>> type(string2)
<type 'str'>
>>> string3 = 'mca'
>>> string3.upper() # capitalise each word in the string
'MCA'
>>> string1.replace('Morning','Afternoon') # replace word
'Good Afternoon'
```

List Data Type

- A list is an ordered group of elements, which can be of different types

Examples

```
>>> list1=['abc', 200, 2+3j, 23.67, 'def']
>>> type(list1)
<type 'list'>
>>> print(list1[0]) # This will print the first element
abc
>>> print(list1[1:3]) # This will print elements starting
from second till fourth (not including fourth)
[200, (2+3j)]
>>> print(list1[2:]) # This will print all elements of
the list starting from third element
[(2+3j),23.67,'def']
```

List Data Type

Examples

```
>>> list2 = ['abc', 100, 23.45]
>>> list3 = ['def', 200, 67.89]
>>> print(list2 * 2) # This will print the elements of
the list twice
['abc', 100, 23.45, 'abc', 100, 23.45]
>>> print(list2 + list3) # It will print the concatenated
list
['abc', 100, 23.45, 'def', 200, 67.89]
>>> list4 = [10, 20, 30] # A homogeneous list
>>> list4.append(40) # This will append 40 to the list
>>> print(list4)
[10, 20, 30, 40]
>>> print(max(list4)) # Prints the largest value of the
list
40
```

Tuple Data Type

- A Tuple is a read-only list
- The elements of a tuple are enclosed within parantheses

Examples

```
>>> list1 = ('abc', 100, 34.56) # Tuple
>>> type(list1)
<type 'tuple'>
>>>print(list1)
('abc', 100, 34.56)
>>> list2 = ['abc', 100, 34.56] # List
>>> list1[1] = 10 # Invalid Operation
>>> list2[1] = 10 # Valid Operation
```

Set Data Type

- A set is an unordered group of unique elements

Examples

```
>>> fruit = {'apple','banana','mango'}
>>> type(fruit)
<type 'set'>
>>> print(fruit) # element order is determined by Python
set(['mango', 'banana', 'apple'])
>>> fruit.add('pineapple') # add element
>>> print(fruit)
set(['mango', 'pineapple', 'banana', 'apple'])
>>> fruit.remove('banana') # remove element
>>> print(fruit)
set(['mango', 'pineapple', 'apple'])
```

Set Data Type

- We can perform the usual mathematical set operations in sets

Examples

```
>>> A = {1, 2, 3}
>>> B = {3, 4, 5, 6}
>>> A | B # Union
set([1, 2, 3, 4, 5, 6])
>>> A & B # Intersection
set([3])
>>> A - B # Difference
set([1, 2])
```

Dictionary Data Type

- A dictionary is an unordered group of elements which are accessed using an associated key value
- Here the elements are stored along with the corresponding keys

Examples

```
>>> temps = {'sun':30,'mon':31,'tue':30,'wed':32,'thu':33,
'fri':32,'sat':31}
>>> type(temps)
<type 'dict'>
>>> print(temps)
{'wed': 32, 'sun': 30, 'thu': 33, 'tue': 30, 'mon': 31,
'fri': 32, 'sat': 31}
>>> temps['sun']
30
```

Data Type Conversions

- Built-in functions are available to convert one data type to another

Examples

```
>>> x = 10
>>> type(x)
<type 'int'>
>>> z = float(x) # converts to float
>>> type(z)
<type 'float'>
>>> print(z)
10.0
```

Data Type Conversions

Examples

```
>>> w = str(x) # converts to string
>>> type(w)
<type 'str'>
>>> print(w)
10
>>> list1 = ['abc',100,10.5]
>>> type(list1)
<type 'list'>
>>> list2 = tuple(list1) # converts to tuple
>>> type(list2)
<type 'tuple'>
>>> list3 = list(list2) # converts to list
>>> type(list3)
<type 'list'>
```

Conditional Execution

- Simple if statement

```
if x > 0 :
    print('x is positive')
```

- if..else statement

```
if x % 2 == 0:
    print('x is even')
else:
    print('x is odd')
```

Conditional Execution

- if .. elif .. [else] statement

```
if marks >= 90:
    print('Grade A')
elif marks >=80:
    print('Grade B')
elif marks >= 70:
    print('Grade C')
elif marks >= 60:
    print('Grade D')
elif marks >= 50:
    print('Grade E')
else:
    print('Grade F')
```

Conditional Execution

- nested if .. else statement

```
if x == y:
    print('x and y are equal')
else:
    if x < y:
        print('x is less than y')
    else:
        print('x is greater than y')
```

Looping

- for loop

```
x = [ 1, 2, 3, 4, 5]
for i in x:
    print(i)      # print integers from 1 to 5


for i in [ 1, 2, 3, 4, 5 ]:
    print(i)      # same output as above

for c in 'Python':   # print each character
    print(c)          # of the string 'Python'
```

Looping

- for loop

```
for i in range(1,6):  # range is a built-in function
    print(i)           # print integers from 1 to 5

for i in range(1,11,2):  # print odd integers
    print(i)              # from 1 to 9

for i in range(5,0,-1):  # print integers
    print(i)              # from 5 to 1
```

Looping

- while loop

```
i = 1
while i < 6 :          # print integers from 1 to 5
    print(i)
    i=i+1

while True :                    # This program terminates
    line = input('> ')  # only when done is typed
    if line == 'done' :
        break
    print(line)
print('Done!')
```

Looping

- Loop Control Statement - break
  This will terminate the loop and transfer control to the statement immediately after the loop

```
value = 5                #---- for loop version ----#
for i in range(1,11):
    if i == value:
        print('found')
        break
else:                  # executed when the iteration is
    print('not found')  # fully over
```

Looping

- Loop Control Statement - break

```
i = 1                          #---- while loop version ----#
value = 5
while i < 11:
    if i == value:
        print('found')
        break
    i = i + 1
else:
    print('not found')
```

Looping

- Loop Control Statement - continue
  This will skip the remaining part of current iteration and go to the beginning of loop

```
for letter in 'Python':          # for loop example
    if letter=='h':
        continue
    print('Current Letter:',letter)

i = 0                            # while loop example
while i < 6:
    i = i + 1
    if i == 3:
        continue
    print(i)
```

Looping

- Loop Control Statement - pass
  - It indicates a null operation
  - This can be used when a statement is required syntactically, but nothing should happen
  - This can also be used in places where code will be written in future

```
for letter in 'Python':
    if letter == 'h':
        pass # null operation
    else:
        print('Current Letter:', letter)
```

Looping

- Nested Loops

```
for i in range(1,6):
    for j in range(1,i+1):
        print(j,end="")
    print()
```

- By default **print()** will insert newline character after every printing
- end="" in **print()** omits this
- **print()** without arguments will simply insert a newline character

Looping

- Nested Loops

```
for i in range(1,6):
    for j in range(1,i+1):
        print(j,end="")
    print()
```

```
Output
------
1
12
123
1234
12345
```

Module 2
Function Definition and Calling

Format - With Arguments

```
def functionname ( arg1, arg2, ... ) :
    statement1
    statement2
    ...
```

Example

```
>>> def add ( num1, num2 ) :
        return num1 + num2
>>> add(20,30)
50
```

Function Definition and Calling

Format - Without Arguments

```
def functionname ( ) :
    statement1
    statement2
    ...
```

Example

```
>>> def display_message( ) :
        print("Good Morning")
        print("Welcome to Python Programming")
>>> display_message()
Good Morning
Welcome to Python Programming
```

Default Argument Values

Example

```
>>> def arithmetic_sequence(start,end,increment=1):
        i=start
        while i<= end :
            print(i,end="")
            i = i + increment

>>> arithmetic_sequence(1,10)
1 2 3 4 5 6 7 8 9 10
>>> arithmetic_sequence(1,10,2)
1 3 5 7 9
```

More on Arguments

- All the arguments are passed by reference to a function
- Hence the function can change the original values in the calling function

Example

```
>>> def changeme( mylist ):
        mylist.append(40)
        print("Values inside the function: ", mylist)
>>> mylist=[10,20,30]
>>> changeme(mylist)
Values inside the function:  [10, 20, 30, 40]
>>> mylist
[10, 20, 30, 40]
```

Variable Length Arguments

- We can pass variable number of arguments to a function
- The variable name preceded by * holds all the values of variable arguments

Example

```
>>> def printinfo( arg1, *args ):
        print("Output is: ")
        print(arg1,end="")
        for var in args:
            print(var,end="")
>>> printinfo(10)
Output is:
10
>>> printinfo(10,20,30,40,50)
Output is:
10 20 30 40 50
```

Keyword Arguments

- While calling a function, usually argument values are passed based on parameter positions
- We can also pass argument values using parameter names.Here, we need not remember relative positions of parameters

Example

```
>>> def printinfo( name, age ):
        print("Name: ", name)
        print("Age: ", age)
>>> printinfo('John',30)        # positional arguments
Name: John
Age: 30
>>> printinfo(age=30,name='John') # keyword arguments
Name: John
Age: 30
```

Lambda Functions

- They are anonymous functions defined using lambda keyword

Format

lambda arg1 [,arg2,.....argn]:expression

Example

```
>>> sum = lambda a,b : a+b     # function definition
>>> sum(5,10)                  # function call
15
>>> print (lambda a, b: a + b)(5, 10)  # both together
15
```

Recursive Functions

- A function which calls itself is a recursive function

Example

```
>>> def factorial(n):
        if n==0 :
            return 1
        else :
            return n * factorial(n-1)

>>> factorial(5)
120
```

Entry Point

- Entry point to a python program is the start of the source code
- main() does not have any significance in python
- However, we can define an explicit main() as given below

Example

```
def main():
    print("Good Morning")

# __name__ is a special variable that contains the
# name of the currently executing module
# If it is the main program, __name__ will be having
# the value "__main__"

if __name__== "__main__":
    main()

print("Welcome to python programming")
```

Module

- A module is a file containing python code
- It is a way of logically organising python code
- Grouping related code into a module makes it easier to understand use
- A module can contain variables, functions, classes and runnable code
- Python Standard library contains several bulit-in modules
- We can also define our own user defined modules
- A module can be used in a python program using import statement

Example

import *module name*

Module

Example

```
>>> import math              # import the math module
>>> print(math.sqrt(10))

>>> import math as Mathematics
>>> print(Mathematics.sqrt(10))

>>> from math import sqrt    # import sqrt() only
>>> print(sqrt(10))

>>> from math import cos as cosine
>>> print(cosine(0))
```

Module

Example

```
# import all functions in math #
# functions can be used without module name #
>>> from math import *
>>> print(sqrt(10))

>>> import keyword, calendar  # import two modules

# This will return true if a string is a keyword #
>>> keyword.iskeyword('if')

# This will return true if a year is leap year #
>>> calendar.isleap(2021)
```

time Module

- This module provides various time related functions

Example

```
>>> import time

# This will print the time elapsed in seconds #
# since the beginning of time for computers #
# ie.1970 January 1 00:00:00 #
>>> print(time.time())

# This will print  #
# the current date and time in the format #
# 'Thu Sep 16 08:30:00 2021' #
>>> print(time.asctime())
```

time Module

Example

```
# example.py #

import time
start = time.time()
time.sleep(3) # suspend the execution for 3 seconds
stop = time.time()
print(stop - start)

# output #

3.00524902344
```

datetime Module

- This module provides various date and time related classes and functions

Examples

```
# This will print today's date in YYYY-MM-DD format #
>>> import datetime
>>> today=datetime.date.today()
>>> print(today)

# This will print today's date in DD-MM-YYYY format #
>>> print(today.strftime("%d-%m-%Y"))

# This will print today's date in the format #
# 'Thursday September 16' #
>>> print(today.strftime("%A %B %d"))
```

datetime Module

Examples

```
# This will print date after 30 days #
>>> no_of_days=datetime.timedelta(days=30)
>>> print(today + no_of_days)

# This will print how many days and time are there #
# till 2022 Jan 26, 10:00 #
>>> today = datetime.datetime.today()
>>> rday = datetime.datetime(2022, 1, 26, 10, 0)
>>> print(rday - today)
```

Creating a Module

Example

```
# Module welcome.py #
print("Good Morning")
def funct1():
    print("Welcome to MESCE")
def funct2():
    print("Welcome to MCA Department")

# Using the above module in a program #
import welcome
welcome.funct1()
welcome.funct2()
```

Locating a Module

- An imported module is to be located and loaded into memory
- Python first searches for modules in the current directory
- If it is not found, it searches the directories specified in the PYTHONPATH environment variable
- If it is still not found, a Python installation-specific path is searched (e.g., C:\Python32\Lib)
- If the module is still not found, an error is reported

Namespaces

- A namespace is a container holding all the defined names in python
- It enables programs to avoid name clashes between various identifiers
- A name clash can occur when multiple modules containing identifiers with the same name are imported into a program
- There are three namespaces in python viz. Built-in Namespace, Global Namespace and Local Namespace

Namespaces

- The built-in namespace contains the names of all built-in functions and constants etc.
- The global namespace contains the names of all identifiers in the currently executing module
- The local namespace contains the names of all identifiers in the currently executing function
- Python looks for an identifier, first in the local namespace, then in the global namespace, and finally in the built-in namespace.

Scope of a Variable

- Scope of a variable is that part of the code where it is visible

1. Local Scope
2. Global Scope
3. Enclosing Scope

Local and Global Scope

- Scope of a variable is that part of the code where it is visible

Example

```
a=1                 # Global Variable
def func():
    b=2             # Local Variable
    print(a)         # print 1 when function is called
    print(b)         # print 2 when function is called
func()
print(a)             # print 1
print(b)             # shows error
```

Enclosing Scope

- A variable scope that is not local or global is non-local or enclosing scope

Example

```
def func1():
    a = 1  # a is local to func1() and
           # non-local to func2()
    def func2():
        b = 2
        print(a)
        print(b)
    func2()
    print(a)
func1()
```

Global Keyword

Example

```
a = 1                # global variable
def func2() :
    a = 2            # local variable
    print(a)          # prints 2
func2()
print(a)             # prints 1
```

Global Keyword

Example

```
a = 1                # global variable
def func2() :
    global a         # global variable
    a = 2            # global variable
    print(a)          # prints 2
func2()
print(a)             # prints 2
```

Nonlocal Keyword

- This is used for accessing a nonlocal variable

Example

```
def func1():
    a = 1               # nonlocal variable for func2()
    def func2():
        a = 2           # local variable
        print(a)        # print 2
    func2()
    print(a)            # print 1
func1()
```

Nonlocal Keyword

Example

```
def func1():
    a = 1              # nonlocal variable for func2()
    def func2():
        nonlocal a     # nonlocal variable
        a = 2
        print(a)       # print 2
    func2()
    print(a)           # print 2
func1()
```

Packages

- A package is a collection of modules
- Python standard library contains several built-in packages such as
  - numpy - performs high-level mathematical functions
  - scipy - used for scientific computing
  - matplotlib - used for plotting publication quality figures
- We can also have user-defined packages

Creating a Package

- We will create a package with the below given two modules

greetings.py

```
def func1():
    print("Hello")
def func2():
    print("Good Morning")
```

welcome.py

```
def func3():
    print("Welcome to MESCE")
def func4():
    print("Welcome to MCA Department")
```

Creating a Package

- Create a directory called package1
- Place the above two modules under this directory
- Now package1 is a package containing modules called greetings and welcome
- Now create the following file inside package1

```
# __init__.py  #
from greetings import func1
from greetings import func2
from welcome import func3
from welcome import func4
```

- This will import all the functions in these modules when the package is imported

Importing a Package

- Now the package can be imported as shown below

```
import package1
# functions are accessed #
package1.func1()
package1.func2()
package1.func3()
package1.func4()
```

Output

```
Hello
Good Morning
Welcome to MESCE
Welcome to MCA Department
```

Exception Handling

- An exception is an undesirable situation in which the normal instruction execution flow of a program is disrupted
- Example 1 - An instruction tries to divide a number by zero
- Example 2 - Trying to open a file which does not exist
- Example 3 - Trying to import a module / package that do not exist / cannot be located
- If an exception is not handled properly, the program will terminate at once

Exception Handling

- Python code for handling an exception contain four blocks

```
try:        # The try block
    contains the code where an error/exception is
    likely to arise
except:     # The except block
    handles an exception that arises in the try block
else:       # The else block
    contains the code that will execute
    if no error occurs in the try block
finally:    # The finally block
    contains the code that will execute
    whether an error occurs or not in the try block
```

- We can have multiple except blocks for handling different types of exceptions

Exception Handling

Program

```
# This program tries to convert
# a non number string to an integer
myString = "This string is not a number"
print("Converting myString to an integer...")
myInt = int(myString)
print(myInt)
print("Done")
```

Output

```
Converting myString to an integer...
Traceback (most recent call last):
File "python", line 4, in <module>
ValueError: invalid literal for int() with base 10:
'This string is not a number'
```

Exception Handling

Program

```
try: # run the code in this block until an error occurs
  myString = "This string is not a number"
  print("Converting myString to an integer...")
  myInt = int(myString)
  print(myInt)
except: # if an error occurs, jump to here
  print("Can't convert; myString not a number")
print("Done")
```

Output

```
Converting myString to an integer...
Can't convert; myString not a number
Done
```

Exception Handling

Program

```
myString = "This string is not a number"
try: # run the code in this block until an error occurs
  print("Converting myString to an integer...")
  print(1/0)
  myInt = int(myString)
  print(myInt)
except ValueError: # handling ValueError Exception
  print("Can't convert; myString not a number")
except ZeroDivisionError: # ZeroDivisionError Exception
  print("Can't divide by zero")
print("Done")
```

Output

```
Converting myString to an integer...
Can't divide by zero
Done
```

Exception Handling

Program

```
myString = "1"
try: # run the code in this block until an error occurs
  print("Converting myString to an integer...")
  myInt = int(myString)
  print(myInt)
except: # if an Exception occurs, jump to here
  print("Can't convert; myString not a number")
else: # This block runs if there is no error
  print("No error occurred")
finally: # Runs whether there is an error or not
  print("Done")
```

Output

```
Converting myString to an integer...
1
No error occurred
Done
```

Exception with Arguments

- An exception can have an argument
- This argument gives additional information about the problem

```
try:
    You do your operations here;
    ......................
except ExceptionType, Argument:
    You can print value of Argument here...
```

Exception with Arguments

Program

```
myString = "This string is not a number"
try: # run the code in this block until an error occurs
  print("Converting myString to an integer...")
  myInt = int(myString)
  print(myInt)
except ValueError, Argument: # ValueError Exception Block
  print("Can't convert; myString not a number")
  print(Argument)
print("Done")
```

Output

```
Converting myString to an integer...
Can't convert; myString not a number
invalid literal for int() with base 10:
'This string is not a number'
Done
```

Raising an Exception

- We can manually raise (throw) an exception using raise statement

Program

```
try:
    raise ValueError # This will raise (throw)
                     # a ValueError exception
except ValueError:
    print('There was an exception.')
```

Output

```
There was an exception.
```

Built-in Exceptions

- Python standard library contains several built-in exceptions
- They can be raised based on various error situations
- A ValueError exception is raised when a function gets an argument of correct type but improper value
- A ZeroDivisionError exception is raised when division or modulo operation by zero takes place

User-defined Exceptions

Program

```
# define Python user-defined exceptions

# Raised when the input value is too small
# derived from the built-in exception Exception
class ValueTooSmallError(Exception):
  pass

# Raised when the input value is too large
# derived from the built-in exception Exception
class ValueTooLargeError(Exception):
  pass
```

User-defined Exceptions

Program - Continues

```
# user guesses a number until he/she gets it right
number = 10
while True:
  try:
    i_num = int(input("Enter a number: "))
    if i_num < number:
      raise ValueTooSmallError
    elif i_num > number:
      raise ValueTooLargeError
    break
  except ValueTooSmallError:
    print("This value is too small, try again!")
  except ValueTooLargeError:
    print("This value is too large, try again!")
print("Congratulations! You guessed it correctly.")
```

Assertions

- They are boolean expressions that will check whether a condition returns true or false
- If the condition returns true, it does nothing and the next line of code is executed
- If the condition returns false, the program stops and AssertionError is displayed
- It can be used as a debugging tool, as it shows at which point an error has occurred

Assertions

Program

```
def avg(marks):
  assert len(marks) != 0  # Assertion
  return sum(marks)/len(marks)

marklist1 = [6,4,12,13,15]
print("Average of marklist1 =", avg(marklist1))
marklist2 = []
print("Average of marklist2 =", avg(marklist2))
```

Output

```
Average of marklist1 = 10
Average of marklist2 =
Traceback (most recent call last):
  File "python", line 9, in <module>
  File "python", line 2, in avg
AssertionError
```

Assertions

Program

```
def avg(marks):  # Assertion with error message
  assert len(marks) != 0, "List is Empty"
  return sum(marks)/len(marks)

marklist1 = [6,4,12,13,15]
print("Average of marklist1 =", avg(marklist1))
marklist2 = []
print("Average of marklist2 =", avg(marklist2))
```

Output

```
Average of marklist1 = 10.0
Traceback (most recent call last):
  File "assertion.py", line 8, in <module>
    print("Average of marklist2 =", avg(marklist2))
  File "assertion.py", line 2, in avg
    assert len(marks) != 0, "List is Empty"
AssertionError: List is Empty
```

Module 3
File Handling

- Writing to Files
  - Contents, if any, are overwritten
  - Original contents are lost
- Appending to Files
  - Contents are written to the end of file
  - Original contents are not lost
- Reading Files

Writing to Files

```
# SimpleFileWriting-1.py #
myInt1=10
myInt2=20
myInt3=30
# Open OutputFile.txt in write mode
outputFile = open ("OutputFile.txt", "w")
# Write myInt1 to outputFile
outputFile.write(str(myInt1))
# Write myInt2 to outputFile
outputFile.write(str(myInt2))
# Write myInt3 to outputFile
outputFile.write(str(myInt3))
# Close outputFile
outputFile.close()

**Output**
102030
```

Writing to Files

```
# SimpleFileWriting-2.py #
myInt1=10
myInt2=20
myInt3=30
outputFile = open ("OutputFile.txt", "w")
outputFile.write(str(myInt1) + "\n")
outputFile.write(str(myInt2) + "\n")
outputFile.write(str(myInt3) + "\n")
outputFile.close()

**Output**
10
20
30
```

Writing to Files

```
# WriteListToFile-1.py #
myList=["David","Lucy","Addison","Wesley","Diana"]
# Open OutputFile.txt in write mode
outputFile = open ("OutputFile.txt", "w")
# For each name in myList
for name in myList:
  # Write name to the file on its own line
  outputFile.write(name+"\n")
# Close outputFile
outputFile.close()

**Output**
David
Lucy
Addison
Wesley
Diana
```

Writing to Files

```
# WriteListToFile-2.py #

myList=["David","Lucy","Addison","Wesley","Diana"]
# Open OutputFile.txt in write mode
outputFile = open ("OutputFile.txt", "w")
# join myList with "\n", then write to the file
outputFile.write("\n".join(myList))
# Close outputFile
outputFile.close()

** Output **
David
Lucy
Addison
Wesley
Diana
```

Appending to Files

```
# AppendToFile.py #                    # OutputFile.txt #
myInt1=40                              # 10
myInt2=50                              # 20
myInt3=60                              # 30
# Open OutputFile.txt in append mode
outputFile = open ("OutputFile.txt", "a")
outputFile.write(str(myInt1) + "\n")
outputFile.write(str(myInt2) + "\n")
outputFile.write(str(myInt3) + "\n")
outputFile.close()
** Output **
10
20
30
40
50
60
```

Reading Files

```
# SimpleFileReading.py #
# Open OutputFile.txt in read mode
inputFile = open ("OutputFile.txt", "r")
myInt1 = int(inputFile.readline())
myInt2 = int(inputFile.readline())
myInt3 = int(inputFile.readline())
inputFile.close()
print(myInt1)
print(myInt2)
print(myInt3)
```

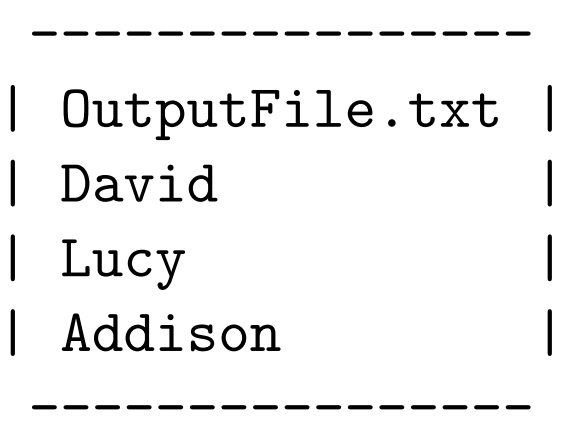
```
** Output **
10
20
30
```

```
---------------
| OutputFile.txt |
| 10             |
| 20             |
| 30             |
---------------
```

Reading Files

```
# LoadFileToList.py #
myList = [ ]
inputFile = open ("OutputFile.txt", "r")
for line in inputFile:
  # add line to myList, stripping out whitespace
  myList.append(line.strip())
inputFile.close()
print(myList)
```

```
** Output **
['David','Lucy','Addison']
```



tell() method in File

- It is used to get the current position of the file pointer(file handle)(file object)
- File pointer is like a cursor, which points to a specific location in the file
- When a file is opened in read or write mode, it points to the beginning of the file
- When a file is opened in append mode, it points to the end of the file

Syntax

```
f.tell()
```

- f is the file pointer

tell() method in File

```
# tell.py #
f1 = open ("file1.txt", "w")
print(f1.tell())
f1.write("Python")
f1.close()
f1 = open ("file1.txt", "r")
print(f1.tell())
f1.close()
f1 = open ("file1.txt", "a")
print(f1.tell())
f1.close()

** Output **
0
0
6
```

seek() method in File

- It is used to change the position of the file pointer to a specified position

Syntax

f.seek(offset[,reference point])

- f is the file pointer
- offset is the number of positions to move forward
- reference point specifies the location from which we change the position of the file pointer
- reference point can be 0, 1 or 2
- 0 indicates beginning of the file (default)
- 1 indicates current position of file pointer (in binary files only)
- 2 indicates end of the file (in binary files only)

seek() method in File

```
# seek.py #
f1 = open ("file1.txt", "w")
f1.write("Python Programming")
f1.close()
f1 = open ("file1.txt", "r")
f1.seek(7)
# Prints the entire line from the current position of file
# pointer
print(f1.readline())
f1.close()

** Output **
Programming
```

Object Oriented Programming

- All values in python are represented as objects
- numeric values, strings, lists etc. are objects
- The built-in function id is used to find the location in which an object is stored (reference value of a variable)

```
>>> n = 10
>>> id(n) # This will return the memory location of n
505498136
```

Class Definition

```
class Person: # Define a class Person
  # special method that initialises an object of Person
  def __init__ (self):
    self.firstname = "no first name"
    self.lastname = "no last name"
    self.eyecolour = "no eye colour"
    self.age = -1
  def getdetails(self):
    print("First Name:", self.firstname)
    print("Last Name:", self.lastname)
    print("Eye Colour:", self.eyecolour)
    print("Age:" , self.age)
  def setdetails(self, firstname, lastname, eyecolour,age):
    self.firstname = firstname
    self.lastname = lastname
    self.eyecolour = eyecolour
    self.age = age
```

Creating Objects

```
# Creating objects of Person
myPerson1 = Person()
print("Person 1")
print("--------")
myPerson1.setdetails("Addison", "Wesley", "Brown", 30)
myPerson1.getdetails()

myPerson2 = Person()
print("Person 2")
print("--------")
myPerson2.setdetails("Guido", "Van Rossum", "Blue", 62)
myPerson2.getdetails()
```

Creating Objects

```
Output
------


Person 1
--------
First Name: Addison
Last Name: Wesley
Eye Colour: Brown
Age: 30
Person 2
--------
First Name: Guido
Last Name: Van Rossum
Eye Colour: Blue
Age: 62
```

Encapsulation and Data Hiding

```
# Define a Class Person
class Person:
  # special method that initialises an object of Person
  def __init__ (self):
    self.__firstname = "Guido" # Private Member
    self.__lastname = "Van Rossum" # Private Member
    self.eyecolor = "Blue"
    self.age = 62
# Creating an object of Person
myPerson = Person()
print(myPerson.firstname) # Not Allowed
print(myPerson.lastname)  # Not Allowed
print(myPerson.eyecolor) # Allowed
print(myPerson.age) # Allowed
```

Inheritance, Method Overriding and Polymorphism

```
# Polymorphism.py #

import math
class Shape:  # Superclass
  def __init__(self,x,y):
    self.__x = x
    self.__y = y
  def getXYLoc(self):
    return(self.__x,self.__y)
  def setXYLoc(self,x,y):
    self.__x = x
    self.__y = y
  def calcArea(self):
    raise NotImplementedError("Method not implemented")
```

Inheritance, Method Overriding and Polymorphism

```
# Polymorphism.py continued #

class Circle(Shape):  # Subclass
  def __init__(self,x,y,r):
    Shape.__init__(self,x,y)
    self.__radius = r
  def calcArea(self):  # Method Overriding
    return math.pi * self.__radius ** 2

class Square(Shape): # Subclass
  def __init__(self,x,y,s):
    Shape.__init__(self,x,y)
    self.__side = s
  def calcArea(self):  # Method Overriding
    return self.__side ** 2
```

Inheritance, Method Overriding and Polymorphism

```
# Polymorphism.py continued #

shape = 1
if shape == 1:
  fig = Circle(0,0,1)
elif shape == 2:
  fig = Square(0,0,2)

print(fig.calcArea())  # Polymorphism
```

Module 4
Regular Expressions: Introduction

- Regular expressions are used to identify whether a pattern exists in a string or not
- They can also be used for modifying a string
- They are handled in Python using re module

Match() Function

- It searches for a pattern in the beginning of a string
- If it is found, the match object is returned
- Otherwise, None will be returned

Match() Function

```
* Program *
import re
# Searching for a pattern in the beginning of a string
result = re.match("monsoon", "monsoon times")
if result:
  print("Pattern Found in the beginning")
else:
  print("Pattern Not Found in the beginning")
result = re.match("times", "monsoon times")
if result:
  print("Pattern Found in the beginning")
else:
  print("Pattern Not Found in the beginning")

* Output *
Pattern Found in the beginning
Pattern Not Found in the beginning
```

Search() Function

- It searches for a pattern anywhere in a string
- If it is found, the match object is returned
- Otherwise, None will be returned

Search() Function

```
* Program *
import re
# Searching for a pattern anywhere in the string
result = re.search("mes", "monsoon times")
if result:
  print("Pattern found in the string")
else:
  print("Pattern not found in the string")
result = re.search("mca", "monsoon times")
if result:
  print("Pattern found in the string")
else:
  print("Pattern not found in the string")

* Output *
Pattern found in the string
Pattern not found in the string
```

Search and Replace

```
* Program *
import re
DateOfBirth = "01-01-2000"
# This will replace - with / in DateOfBirth
DateOfBirth = re.sub("-","/",DateOfBirth)
print(DateOfBirth)

* Output *
01/01/2000
```

Regular Expression Modifiers

- They provide additional options while matching

```
* Program *
import re
# case sensitive search
result = re.search("MES", "monsoon times")
# This will ignore case while searching
# re.I - a regular expression modifier
result = re.search("MES", "monsoon times", re.I)
```

Regular Expression Pattern - Examples

```
import re
# Searches for mes in the beginning of string
result = re.search("^mes", "monsoon times")
# Searches for mes in the end of string
result = re.search("mes$", "monsoon times")
# Checks if there is a single character between m and s
# in the string
result = re.search("m.s", "monsoon times")
# Checks if there is a pair of characters between s and n
# in the string
result = re.search("s..n", "monsoon times")
```

Character Classes / Special Character Classes

```
# Character Class can be specified within brackets
# Any character inside is matched
result = re.search("b[aeiou]t", "bat")

# Special Character Classes
# \d will match any decimal digit
result = re.search("file\d", "file1.txt")
# \D will not match any decimal digit
result = re.search("file\D", "files.txt")
# \w will match any alphanumeric character
result = re.search("first\wname", "first-name")
# \W will not match any alphanumeric character
result = re.search("first\Wname", "first-name")
```

Repetition Cases

```
# Matches 0 more repetitions of b after a
result = re.search("ab*", "a")
# Matches 1 or more repetitions of b after a
result = re.search("ab+", "a")
# Matches 0 or 1 occurrences of b after a
result = re.search("ab?", "a")
# \d{3} match exactly 3 digits
result = re.search("\d{3}", "234")
# \d{3,} match 3 or more digits
result = re.search("\d{3,}", "23")
# \d{3,5} match 3,4 or 5 digits
result = re.search("\d{3,5}", "234")
```

findall() method

- findall() finds all non-overlapping occurrences of a pattern in a string and returns a list of all matches

```
* Program *
import re
print(re.findall("car", "car"))
print(re.findall("car", "carrying a car"))
print(re.findall("aa","aaaa"))

* Output *
['car']
['car', 'car']
['aa', 'aa']
```

compile() method

- compile() compiles a regular expression pattern into a regular expression object
- This object can be used for matching using various methods
- This will make the execution faster, if the same pattern is used several times in a program

```
* Program *
import re
c = re.compile("car")
print(c.findall("car"))
print(c.findall("carrying a car"))
```

Database Programming

- Creating Tables
- Insert Operation
- Read Operation
- Update Operation
- Delete Operation

Creating Tables

```
* Program *
import MySQLdb # Module used for connecting to mysql
# Open database connection
# Arguments(machine name,username,password,database name)
db = MySQLdb.connect("localhost","username","pwd","test" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Create table, Triple quoted string for multiple lines
sql = """CREATE TABLE EMPLOYEE (
         FIRST_NAME  CHAR(20) NOT NULL,
         LAST_NAME  CHAR(20),
         AGE INT,
         SEX CHAR(1),
         INCOME FLOAT )"""
cursor.execute(sql)
# disconnect from server
db.close()
```

Insert Operation

```
* Program *
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","username","pwd","test" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to INSERT a record into the database.
sql = """INSERT INTO EMPLOYEE(FIRST_NAME,
         LAST_NAME, AGE, SEX, INCOME)
         VALUES ('Guido', 'Van Rossum', 65, 'M', 20000)"""
try:
   cursor.execute(sql)
   db.commit() # Commit changes in the database
except:
   db.rollback() # Rollback if there is any error
# disconnect from server
db.close()
```

Read Operation

```
import MySQLdb
db = MySQLdb.connect("localhost","username","pwd","test" )
cursor = db.cursor()
sql = "SELECT * FROM EMPLOYEE"
try:
   cursor.execute(sql)
   results = cursor.fetchall() # Fetch all the rows
   for row in results:
      fname = row[0]
      lname = row[1]
      age = row[2]
      sex = row[3]
      income = row[4]
      print("fname=%s,lname=%s,age=%d,sex=%s,income=%d" %\
            (fname, lname, age, sex, income ))
except:
   print("Error: unable to fetch data")
db.close()
```

Update Operation

```
* Program *
import MySQLdb
db = MySQLdb.connect("localhost","username","pwd","test" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to UPDATE required records
sql = "UPDATE EMPLOYEE SET INCOME = 30000"
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit changes in the database
   db.commit()
except:
   # Rollback if there is any error
   db.rollback()
# disconnect from server
db.close()
```

Delete Operation

```
import MySQLdb
# Open database connection
db = MySQLdb.connect("localhost","username","pwd","test" )
# prepare a cursor object using cursor() method
cursor = db.cursor()
# Prepare SQL query to DELETE required records
sql = "DELETE FROM EMPLOYEE"
try:
   # Execute the SQL command
   cursor.execute(sql)
   # Commit your changes in the database
   db.commit()
except:
   # Rollback in case there is any error
   db.rollback()
# disconnect from server
db.close()
```

Module 5
GUI Programming

Important Options

- Tkinter
- wxPython
- JPython

Tkinter Introduction

- It is the standard GUI library for python
- It provides a fast and easy way to create GUI applications
- Tkinter provides a powerful object-oriented interface to the Tk GUI toolkit
- Tk is a user interface toolkit that makes it easy to build desktop graphical user interfaces
- Tk is cross-platform, It can run on Windows, Mac and Linux

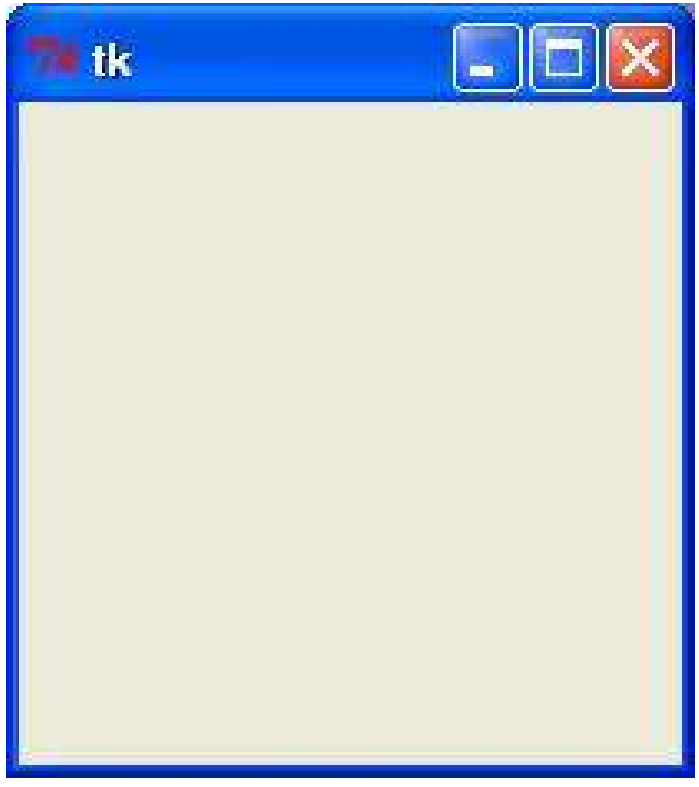
Tkinter and Python Programming

- Example - TkinterWindow.py

```
# Imports the tkinter module
import tkinter
# Create the GUI application main window
w = tkinter.Tk()
# Enter the main event loop to take action against
# each event triggered by the user.
w.mainloop()
```

Tkinter and Python Programming

- Output - TkinterWindow.py



Tkinter Widgets

- Tkinter widgets are controls used in a GUI application. They include
  - Buttons
  - Labels
  - Text Box etc

Tkinter Widgets - Button

- The Button widget is used to add buttons in a Python application

Syntax

widget variable = tkinter.Button (*parent window name, options*)

- options - the list of options for this widget

Tkinter Widgets - Button

- Example - TkinterButton.py

```
# Imports the Tkinter module
import tkinter
# Create the GUI application main window
w = tkinter.Tk()
# Create a button in the main window with text "Hello"
B = tkinter.Button(w, text ="Hello")
# This method organises the widget before placing in the
# parent window
B.pack()
# Enter the main event loop to take action against
# each event triggered by the user.
w.mainloop()
```

Tkinter Widgets - Button

- Output - TkinterButton.py

Tkinter Widgets - Label

- The Label widget is used to provide a caption for other widgets in a Python application

Syntax

widget variable = tkinter.Label (*parent window name, options*)

- options - the list of options for this widget

Tkinter Widgets - Entry

- The Entry widget is used to display a single-line text box for accepting values from a user.

Syntax

widget variable = tkinter.Entry (*parent window name, options*)

- options - the list of options for this widget

Tkinter Widgets - Label and Entry

- Example - TkinterLabelTextbox.py

```
# Imports every object in the Tkinter module
from tkinter import *
# Create the GUI application main window
w = Tk()
# Create a label widget with text "User Name" in the
# window
L1 = Label(w, text="User Name")
# This method organises the widget to the left of the
# parent window
L1.pack(side = LEFT)
# Create an entry widget with border size 5 pixels in the
# window
# Default border size is 2 pixels
E1 = Entry(w, bd=5)
```

Tkinter Widgets - Label and Entry

- Example - TkinterLabelTextbox.py - continued

```
# This method organises the widget to the right of the
# parent window
E1.pack(side = RIGHT)
# Enter the main event loop to take action against
# each event triggered by the user.
w.mainloop()
```

Tkinter Widgets - Label and Entry

- Output - TkinterLabelTextbox.py

Tkinter Widgets - Checkbutton

- The Checkbutton widget is used to display a number of options as checkboxes.
- The user can select multiple options at a time.

Syntax

widget variable = tkinter.Checkbutton (*parent window name*, *options*)

- options - the list of options for this widget

Tkinter Widgets - Checkbutton

- Example - TkinterCheckbutton.py

```
# Imports every object in the Tkinter module
from tkinter import *
# Create the GUI application main window
w = Tk()
# Creates Tkinter integer variables to hold the current
# status of checkbuttons
CheckVar1 = IntVar()
CheckVar2 = IntVar()
# Create a Checkbutton in the parent window with
# text "Music"
C1 = Checkbutton(w, text = "Music", variable = CheckVar1,
onvalue = 1, offvalue = 0, height=5, width = 20)
```

Tkinter Widgets - Checkbutton

- Example - TkinterCheckbutton.py - continued

```
# Create a Checkbutton in the parent window with
# text "Video"
C2 = Checkbutton(w, text = "Video", variable = CheckVar2,
onvalue = 1, offvalue = 0, height=5, width = 20)
# This method organises the widget before placing in the
# parent window
C1.pack()
C2.pack()
# Enter the main event loop to take action against each
# event triggered by the user
w.mainloop()
```

Tkinter Widgets - Checkbutton

- Output - TkinterCheckbutton.py

Tkinter Widgets - Radio Button

- Radio button widget also provides multiple options
- Here the user can select only one option at a time

Syntax

widget variable = tkinter.Radiobutton (*parent window name*, *options*)

- options - the list of options for this widget

Tkinter Widgets - Radio Button

- Example - TkinterRadioButton.py

```
# Imports every object in the Tkinter module
from tkinter import *
# Create the GUI application main window
w = Tk()
# var is the control variable, an integer variable
# shared by all radio buttons
var = IntVar()
# Create a radio button widget with text "Option 1"
# in the window, var having value 1
R1 = Radiobutton(w, text="Option 1",variable=var,value=1)
# pack() method organises the widget before it is placed
# in the parent window, if no argument is specified it
# will be centered
R1.pack()
```

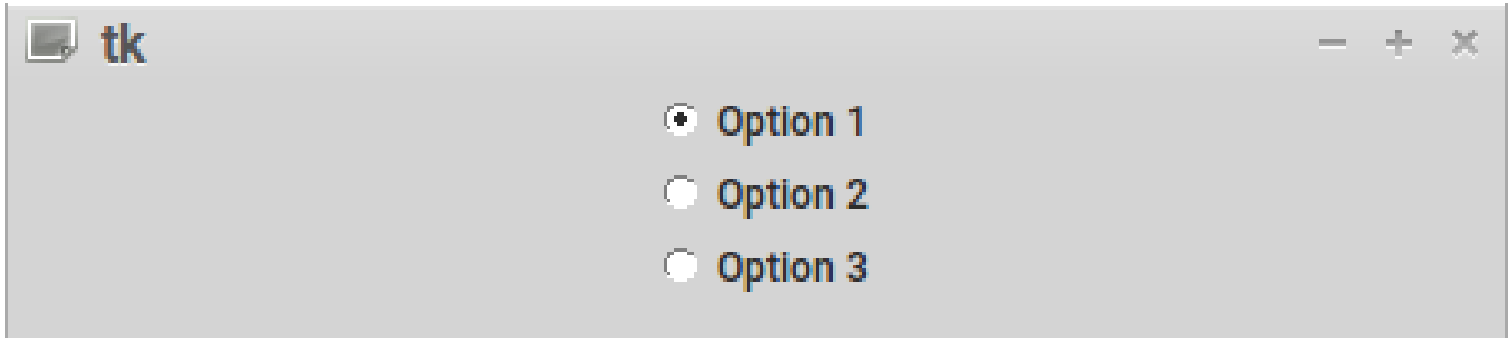
Tkinter Widgets - Radio Button

- Example - TkinterRadioButton.py - continued

```
# Create a radio button widget with text "Option 2"
# in the window, var having value 2
R2 = Radiobutton(w, text="Option 2",variable=var,value=2)
R2.pack()
# Create a radio button widget with text "Option 3"
# in the window, var having value 3
R3 = Radiobutton(w, text="Option 3",variable=var,value=3)
R3.pack()
# Enter the main event loop to take action against each
# event triggered by the user
w.mainloop()
```

Tkinter Widgets - Radio Button

- Output - TkinterRadioButton.py



Tkinter Widgets - Menu

- Used for creating different kinds of menus

Syntax

widget variable = tkinter.Menu(*parent window name, options*)

- options - the list of options for this widget

Tkinter Widgets - Menu

- Example - TkinterMenu.py

```
# Imports every object in the Tkinter module
from tkinter import *
# Create the GUI application main window
w = Tk()
# Create a Menu widget in the window
menu = Menu(w)
# Display the menu
w.config(menu=menu)
# Create a toplevel menu bar
menubar = Menu(menu)
# Create a main menu "File"
menu.add_cascade(label="File", menu=menubar)
# Create a submenu "New"
menubar.add_command(label="New")
# Create a submenu "Open"
menubar.add_command(label="Open")
```

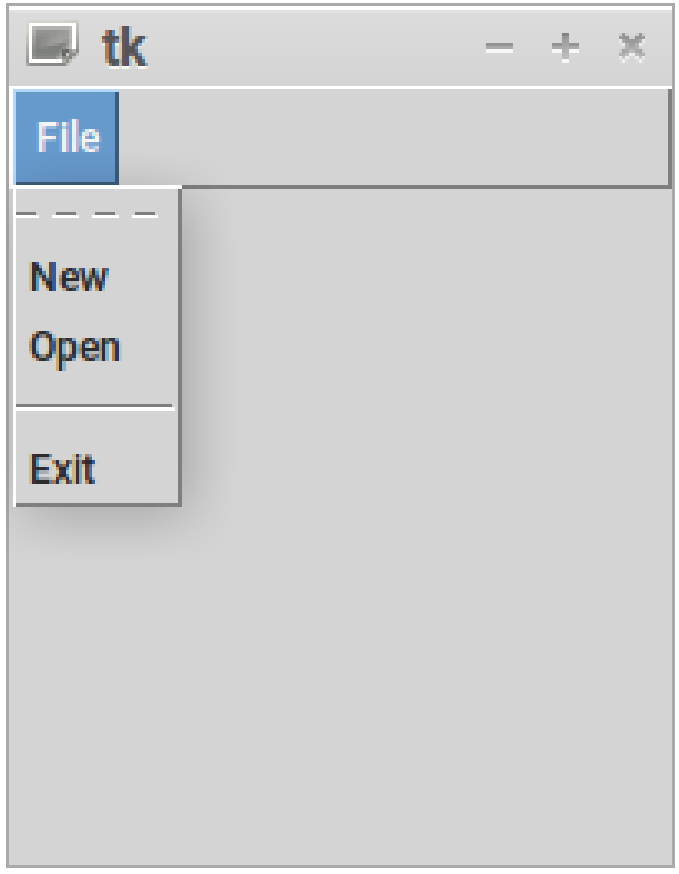
Tkinter Widgets - Menu

- Example - TkinterMenu.py - continued

```
# Adds a separator line to the menu
menubar.add_separator()
# Create a submenu "Exit", which when clicked will quit
# the window
menubar.add_command(label="Exit", command=w.quit)
# Enter the main event loop to take action against each
# event triggered by the user
w.mainloop()
```

Tkinter Widgets - Menu

- Output - TkinterMenu.py



Web Development

- In this section we will see how python is used for retrieving and processing information from world wide web
- A world wide web follows the client-server architecture used in computing
- A web client is a program that retrieves information from the web, by sending requests to a web server
  Example - Browser
- A web server is a program running on an information provider's host computer
- It responds to these requests by providing the required data
- The standard protocol used for web communication is Hyper Text Transfer Protocol (HTTP)

Python Web Client Tools

- A browser is used primarily for viewing and interacting with websites
- A web client can do more than that
- Here we will see web client tools used in python to retrieve, process, store and transmit information from web
- The address of a document in web is called Uniform Resource Locator (URL)
- Python uses urllib package for dealing with URLs
- urllib package contains the following modules

1. urllib.request for opening and reading URLs
2. urllib.error containing the exceptions raised by urllib.request
3. urllib.parse for parsing URLs
4. urllib.robotparser for parsing robots.txt files

urllib.request Module

- This module is used for opening and reading URLs

Example

```
# This python code opens the python.org website
# and reads the first 300 bytes of it

>>> import urllib.request
>>> f = urllib.request.urlopen('http://www.python.org/')
>>> print(f.read(300))
```

urllib.request Module

- This module is used for opening and reading URLs

Example

```
# This python code opens the python.org website
# and reads the first 300 bytes of it

>>> import urllib.request
>>> f = urllib.request.urlopen('http://www.python.org/')
>>> print(f.read(300))
b'<!doctype html>\n<!--[if lt IE 7]>   <html class="no-js
ie6 lt-ie7 lt-ie8 lt-ie9">   <![endif]-->\n<!--[if IE 7]>
<html class="no-js ie7 lt-ie8 lt-ie9">   <![endif]-->\n
<!--[if IE 8]>      <html class="no-js ie8 lt-ie9">
<![endif]-->\n<!--[if gt IE 8]><!--><html class="no-js"'
```

urllib.parse Module

- This module is used for parsing URLs

```
# This python code parses the specified URL into
# 6 components
>>> import urllib.parse
>>> f = urllib.parse.urlparse('https://docs.python.org/3/
faq/index.html')
>>> f
ParseResult(scheme='https', netloc='docs.python.org',
path='/3/faq/index.html',params='',query='',fragment='')
>>> f.scheme   # Network Protocol
'https'
>>> f.netloc   # Location of Server
'docs.python.org'
>>> f.path
'/3/faq/index.html'   # Path to a specific web page
```

urllib.parse Module

- This module is used for parsing URLs

```
>>> f = urllib.parse.urlparse('http://www.mysite.com/
admin/UpdateUserServlet;user')
>>> f
ParseResult(scheme='http', netloc='www.mysite.com',
path='/admin/UpdateUserServlet', params='user', query='',
fragment='')

# params refers to parameters for last path element
# They begin with ;
```

urllib.parse Module

- This module is used for parsing URLs

```
# A query  refers to  & delimited set of key=value pairs

# A query begins with ?

>>> f = urllib.parse.urlparse('https://www.amazon.in/s?
k=shirt&ref=nb_sb_noss_2')
>>> f
ParseResult(scheme='https', netloc='www.amazon.in',
path='/s', params='', query='k=shirt&ref=nb_sb_noss_2',
fragment='')
```

urllib.parse Module

- This module is used for parsing URLs

```
# A fragment refers to a specific location within a
# web page, It is followed by # character

>>> f = urllib.parse.urlparse('https://docs.python.org/3/
library/urllib.parse.html#structured-parse-results')
>>> f
ParseResult(scheme='https',netloc='docs.python.org',
path='/3/library/urllib.parse.html', params='', query='',
fragment='structured-parse-results')


# If we are specifying fragment in a URL, you will
# be taken to that specific location within the page
```

Web Client

- A Simple Program for Accessing Live Cricket Scores from Cricinfo

score.py

```
# A module for downloading and parsing syndicated feeds
import feedparser
# Parses the given URL
scores = feedparser.parse('http://static.cricinfo.com/rss/
livescores.xml')
# Access the RSS feed element 'title'
for score in scores.entries:
     print(score.title)
```

sample output

```
Lions 324/10  v Cape Cobras 162/4 &  115/10 *
Warriors 231/10 &  124/10  v Titans 63/2 &  293/10 *
Knights 208/4 &  424/10  v Dolphins 202/6 &  162/10 *
```

Web Server

- We can create a simple HTTP Web Server using http.server module in Python

```
# This command will start the server in the current
# directory
>>> python3 -m http.server
Serving HTTP on 0.0.0.0 port 8000 ...
```

- Now you can see the file structure of the current directory by entering this IP Address along with port number in your browser

  `http://0.0.0.0:8000/`

Web Service

- Application Programming Interface(API) is a standard that facilitates communication between two or more computer programs.
- Open Notify is an open source project that provides APIs for NASA's Data
- We will see a python program to retrieve the number of people in the space now using an Open Notify API

Web Service

People_in_Space.py

```
# Used for Sending HTTP Requests
import requests
# APIs retrieve data in json (java script object notation)
# format, used for storing structured data
people = requests.get('http://api.open-notify.org/
astros.json')
# json() decodes json format into a dictionary
people_json  = people.json()
# To print the number of people in space
print("Number of people in space:",people_json['number'])
# To print the names of people in space using a for loop
for p in people_json['people']:
    print(p['name'])
```

Web Service

Sample Output

```
Number of people in space: 3
Sergey Ryzhikov
Kate Rubins
Sergey Kud-Sverchkov
```

Module 6
Introduction to scipy

- scipy is a built-in package in python used for scientific computing
- Example - scipyConstants.py
- This program prints various scientific and mathematical constants

```
# This will import the scipy.constants subpackage covering
# physical and mathematical constants
import scipy.constants
print("Pi = ", scipy.constants.pi)
print("Golden Ratio = ", scipy.constants.golden)
print("Speed of light in vacuum = ", scipy.constants.c)
print("Planck Constant = ", scipy.constants.h)
print("Gravitational Constant = ", scipy.constants.G)
print("Avogadro Constant = ", scipy.constants.Avogadro)
```

Introduction to scipy

- Output - scipyConstants.py

```
Pi =  3.14159265359
Golden Ratio =  1.61803398875
Speed of light in vacuum =  299792458.0
Planck Constant =  6.62607004e-34
Newton's Gravitational Constant =  6.67408e-11
Avogadro Constant =  6.022140857e+23
```

Introduction to numpy

- numpy is a built-in package in python used for performing high level mathematical functions
- Example 1 - Sorting an Array

```
>>> import numpy
>>> array1 = numpy.array([2,1,5,4,3,8,7,6,9])
>>> array1
array([2, 1, 5, 4, 3, 8, 7, 6, 9])
>>> numpy.sort(array1)
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Introduction to numpy

- Example 2 - Matrix Addition

```
>>> import numpy
>>> matrix1 = numpy.array([[1,2],[3,4]])
>>> matrix1
array([[1, 2],
       [3, 4]])
>>> matrix2 = numpy.array([[5,6],[7,8]])
>>> matrix2
array([[5, 6],
       [7, 8]])
>>> matrix1 + matrix2
array([[ 6,  8],
       [10, 12]])
```

Introduction to matplotlib

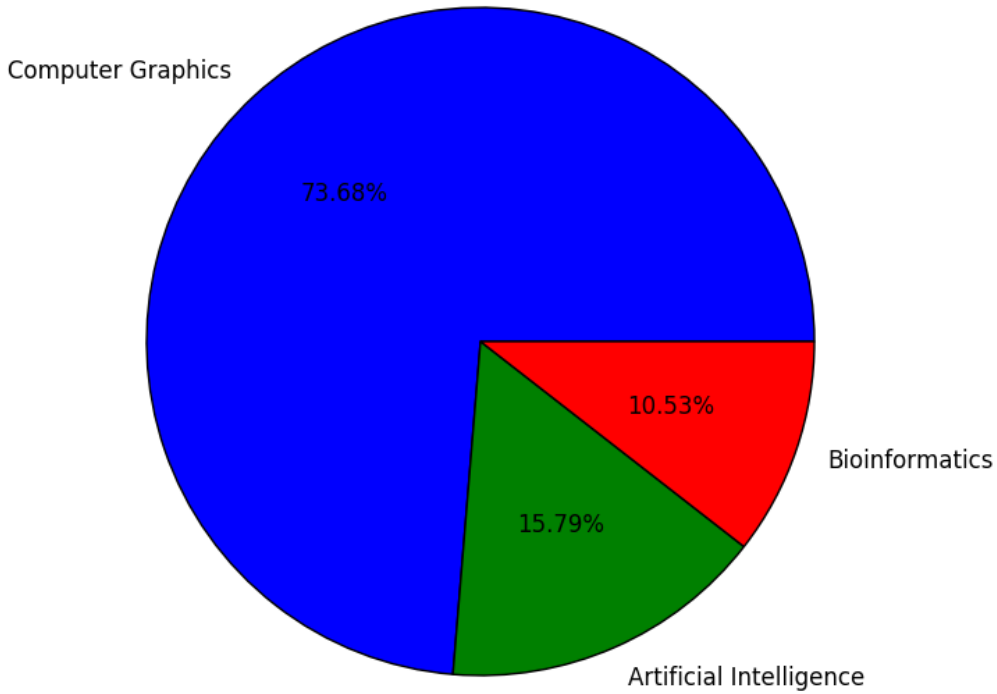
- matplotlib is a built-in package used for plotting publication quality figures
- Example - pie_chart.py

```
# pyplot is a module used in the package matplotlib to
# plot various figures
import matplotlib.pyplot as plt
# create a new figure
fig = plt.figure()
# list parameters[x0,y0,width,height]
ax = fig.add_axes([0,0,1,1])
# equal scaling of axis
ax.axis('equal')
# list of courses
courses = ['Computer Graphics', 'Artificial Intelligence',
'Bioinformatics']
# no of students enrolled in each course
students = [42,9,6]
```

Introduction to matplotlib

- Example - pie_chart.py - continued

```
# draw the pie chart, autopct displays percentage value in
# each pie wedge
ax.pie(students, labels = courses,autopct='%.2f%%')
# show the pie chart
plt.show()
```



Web Frameworks

- A web framework is a software framework that provides a standard way to build and deploy web applications on the world wide web.
- A web application(web app) is an application software that runs on a web server
- They are accessed by the user through a web browser with an active internet connection.
- Examples - Webmail, Online Shopping, Online Banking

Introduction to Django

- Django is a Python based web framework
- Started by Adrian Holovaty and Simon Willison as an internal project at the Lawrence Journal-World newspaper in 2003.
- It was publicly released in 2005 and named as Django after the jazz guitarist Django Reinhardt
- Since 2008, Django Software Foundation is maintaining Django as a free and open source application
- Well known sites that use Django - Instagram, Mozilla, Washington Times

Creating an App Using Django

- Example - Hello.py

```
# This module contains system specific parameters and
# functions
import sys
# configure django settings
from django.conf import settings
settings.configure(
 DEBUG=True,
 SECRET_KEY='thisisthesecretkey',
 ROOT_URLCONF=__name__,
 MIDDLEWARE_CLASSES=(
 'django.middleware.common.CommonMiddleware',
 'django.middleware.csrf.CsrfViewMiddleware',
 'django.middleware.clickjacking.XFrameOptionsMiddleware',
                    ),
 )
```

Creating an App Using Django

- Example - Hello.py - continued

```
# displays 'Hello World' associated with a URL
from django.conf.urls import url
from django.http import HttpResponse
def index(request):
    return HttpResponse('Hello World')
urlpatterns = (url(r'^$', index),)

if __name__ == "__main__":
   from django.core.management
   import execute_from_command_line
   execute_from_command_line(sys.argv)
```

Creating an App Using Django

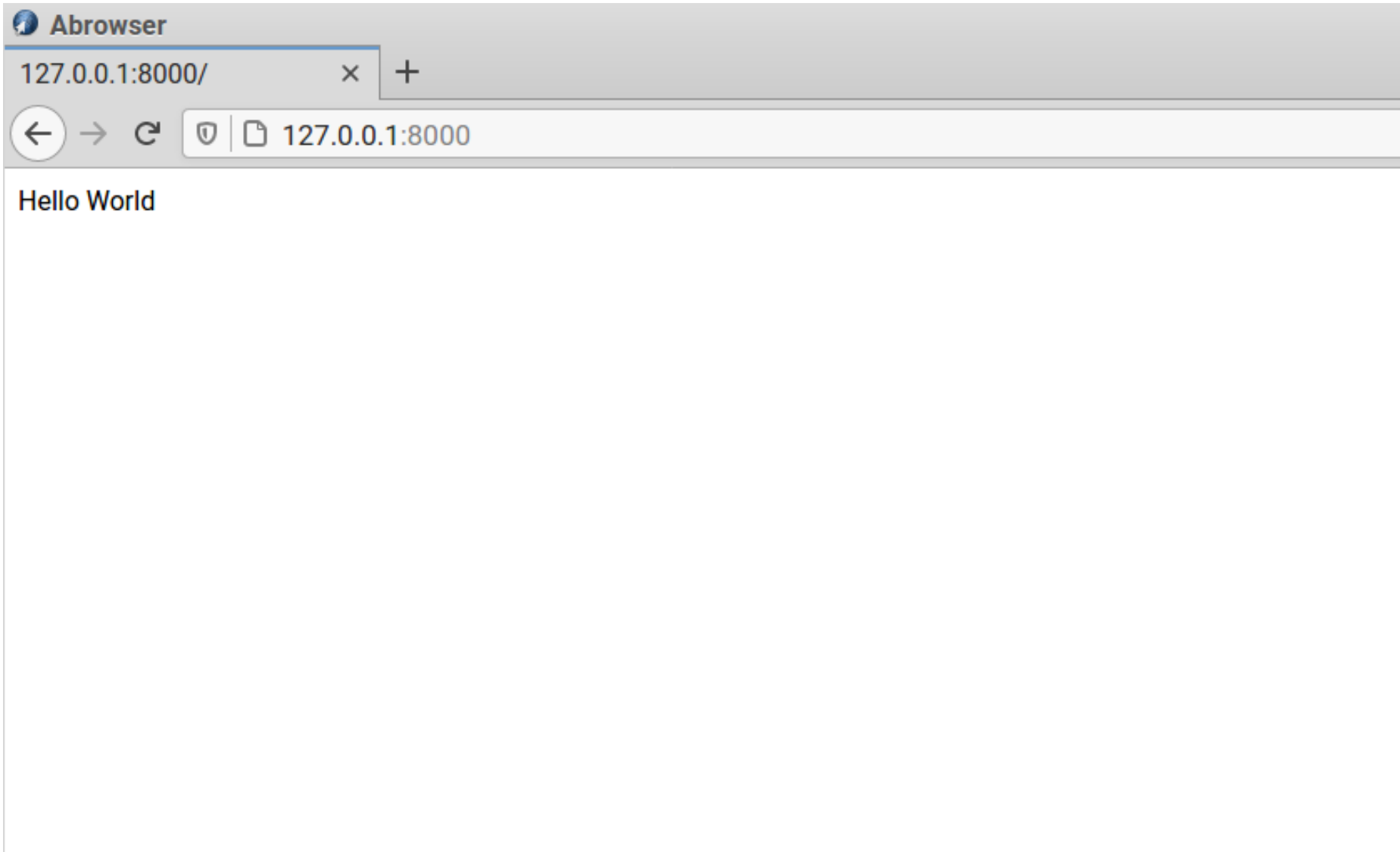
- Output - Hello.py - continued

```
>>> python3 Hello.py runserver
Performing system checks...

System check identified no issues (0 silenced).
December 29, 2020 - 12:47:31
Django version 1.8.7, using settings None
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Creating an App Using Django

- Output - Hello.py - continued (when opened in a web browser)
- **http://127.0.0.1:8000** or **http://localhost:8000**



Bibliography

1. Introduction to Computer Science using Python , Charles Dierbach, Wiley, 2015
2. Core Python Applications Programming (Third Edition), Wesley J Chun, Pearson
3. Think Python, Allen Downey, Green Tea Press
4. https://numpy.org
5. https://www.tutorialspoint.com
6. https://www.geeksforgeeks.org
7. Python Programming - Wikibooks