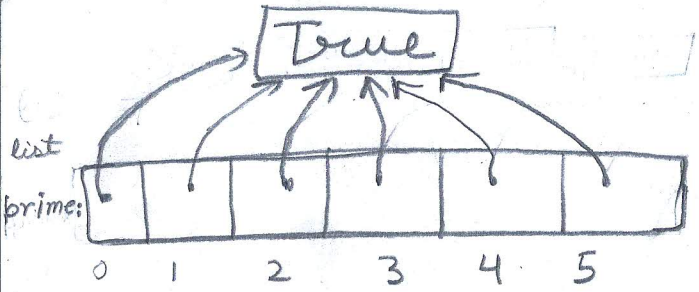


IV	**Sieve of Eratosthenes:** generate prime numbers with the help of an algorithm given by the Greek Mathematician named Eratosthenes, whose algorithm is known as Sieve of Eratosthenes. **File I/O :** File input and output operations in Python Programming **Exceptions and Assertions** **Modules :** Introduction , Importing Modules , **Abstract Data Types :** Abstract data types and ADT interface in Python Programming. **Classes :** Class definition and other operations in the classes , Special Methods (such as _init_, _str_, comparison methods and Arithmetic methods etc.) , Class Example , Inheritance , Inheritance and OOP.

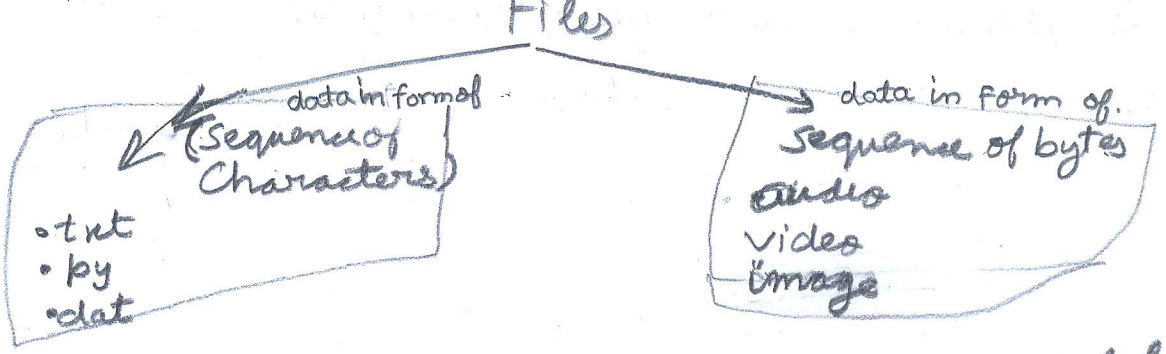
Python program to print all prime numbers smaller than or equal to n using Sieve of Eratosthenes.

```
def sieveOfEratosthenes(n):
    prime = [True for i in range(n+1)]
    p = 2
    while (p**2 <= n):
        if prime[p] == True          # update all multiples of p
            for i in range(p**2, n+1, p):  # multiples >= square
                prime[i] = False

        p = p+1
    for p in range(2, n+1):
        if prime[p]:
            print(p)

sieveOfEratosthenes(5)  <- function call.
```

O/p:
2
3
5
7

<- these are the prime numbers smaller than or equal to 10.

File Input/Output : File input and output operations in Python Programming



In basic file handling we dont require import statement.

3 logical steps of file:

① open → f = open ('filename', 'mode')
② Processing
③ close.

path of file eg. 'file1.txt'

f is reference variable pointing to that file.

7 modes (works on text files)

- 'r' — read only
- 'w' — write only (not read), create empty file if no file exists, erases data
- 'a' — no reading, new lines at end, no erase
- 'r+' — read only, write only, modify, FileNotFound error.
- 'w+' — write/read, 'w'
- 'a+' — reading and writing, no erase, creates new file if no file exists
- 'x' — exclusive write mode, always creates new file then write. If same name file exists then it gives error. Use 'x' mode when you know that file name does not exist

7 modes (works on binary files.)

- rb — b means binary
- wb
- ab
- rb+
- wb+
- ab+
- xb+

File Properties

```
f = open('file1.txt', 'r')      ---- (1)
file processing                 ---- (2)
f.close()                       ---- (3)
```

(2) If we open a file that does not exist then: eg. `f = open('file1.txt', 'r')`

FileNotFoundError

```
f = open('file1.txt', 'r')
f.name        ---> will give   'file1.txt'
f.mode        ---> will give mode   'w'
f.readable()  ---> Can we read this file?   False
f.writable()  ---> Can we write in this File?   True
```

eg. we create a file called file1.txt

```
Tom - Salesman
Pam - Accountant
Sam - Security
Jim - Salesman
```

app.py

```
f1 = open("file1.txt", "r")
print(f1.read())        # all lines read
print(f1.readline())    # reads single line (first)
f1.close()
```

Writing to file

app.py

```
f1 = open("file1.txt", "a")
f1.write("\nMoby - officer")
f1.write("\nNelly - Advisor")
f1.close()
```

new entry at end ---> file1.txt

```
Tom -
Pam -
Sam -
Jim -
Moby - officer
Nelly - Advisor
```

```
f1 = open("file1.txt", "w")
f1.write("Bob - Mechanic")
f1.close()
```

overwrite ---> file1.txt

```
Bob - Mechanic
```

```
f1 = open("file2.txt", "w")
f1.write("Andy - Programmer")
f1.close()
```

new file created ---> file2.txt

```
Andy - Programmer
```

Exceptions and Assertions

Exception is a runtime error.

Exception handling means to handle exceptional situations.

```
x = int(input("Enter first number"))   ← int type
y = input("Enter second number"))      ← str type
print("Sum is", x+y)                   ← addition not allowed
```

Because we cannot add an int and a str.

O/p: TypeError: . . unsupported operand type(s) for +: 'int' and 'str'

```
① x = int(input("Enter first number"))
② y = int(input("Enter second number"))
③ print("Division result", x/y)
```

O/p: Enter first number 10
Enter second number 0

ZeroDivisionError: . . message . . . .

names of classes

Like we have int type
Str type etc.
We have TypeError type
ZeroDivisionError type

These all are pre-defined classes in Python.

type means class.
Class means type.

In Python when we create a class, it also means that we are creating a data type or simply type.

There are so many types in Python because there are so many classes in Python

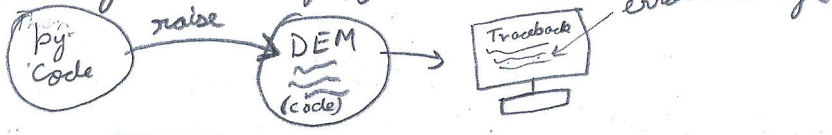
These 2 classes and many more are called exception classes.

When this line ③ was executing, because user has entered 0 as value for y, then Python automatically will raise exception. (by creating object of ZeroDivisionError class)

During runtime, if any error comes, now this error is denoted by an exception class.

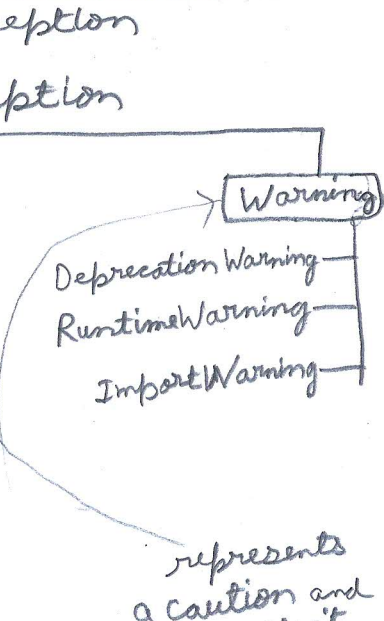
these are pre-defined class (ie we have not made in our program code.

Exception raising means that out of total exception classes that are in Python eg. TypeError, ZeroDivisionError, etc. Python will check if there is a problem in your code. Then Python will check that the exception is of what category. Then after identifying this: the object of class representing this category will be made by Python and raised.



After raising the control goes to DefaultExcept mechanism, and the pre-written code inside DEM's result is seen by user in the output! Traceback . . ZeroDivisionError . . . . message

All exceptions are represented as classes in Python. The exceptions which are already available in Python are called 'built-in' exceptions. The base class for all built in exceptions is 'BaseException' class. From BaseException class, the sub class 'Exception' is derived. From Exception class, the sub classes 'StandardError' and 'Warning' are derived.



An error should be compulsory handled otherwise the program will not execute

represents a caution and even though it is not handled, the program will execute.

IMPORTANT EXCEPTION CLASSES in Python

Exception handling:

```
x = int(input("Enter a number"))
y = int(input("Enter another number"))
try:
    z = x/y
    print("Division result", z)
except ZeroDivisionError:
    print("Invalid attempt of division")
```

Probability of coming Exception is more here.

problematic code we place inside try block.

its solution we place inside except block

try block → `z = x/y` / `print("Division result", z)`

except block → `print("Invalid attempt of division")`

o/p: Enter a number 10
Enter another number 0
Invalid attempt of division

```
try:
    x = int(input("Enter a number"))
    y = 1/x
finally:
    print("We are not raising exception.")
print("Inverse is", y)
```

o/p: Enter a number: 5
We are not raising exception.
Inverse is 0.2

- We can write a try block without any except blocks.
- We cannot write except blocks without a try block.
- else block and finally blocks are optional
- finally block is always executed

```
try:
    statements
except Exception(classname)1:
    ==
except Exception(classname)2:
    ==
else:
    statements
finally:
    statements
```

In one try block, there can be multiple except blocks.
At a time an exception will occur, then exactly 1 except block will execute.

None of the except block executes: When in try block, any line is raising an exception (ie an object is raised). Now we match class name of object with every multiple except blocks class name. Class name. If yes then our except block will execute (Except mechanism).

If all these cases fail then, DEM (Default Except Mechanism) will execute.

Handling Syntax Error.

```
try:
    date = eval(input("Enter date : "))
except SyntaxError:
    print('Invalid date entered')
else:
    print('You entered: ', date)
```

o/p: Enter date: 2007, 9, 9 ← eval() function will evaluate comma seperated values as tuple type.
You entered: (2007, 10, 9)

o/p: Enter date: 2007, 9b, 9) ← wrong syntax entered by user.
Invalid date entered.

Handling AssertionError without message

```
try:
    x = int(input('Enter a number between 5 and 10 '))
    assert x>=5 and x<=10
    print('The number entered: ', x)
except AssertionError:
    print('The condition is not fulfilled')
```

o/p: Enter a number between 5 and 10 15
The condition is not fulfilled.

Assertion:

The assert statement is useful to ensure that a given condition is True. If it is not true, it raises AssertionError.

assert Condition, message

If the condition is False, then the exception by the name AssertionError is raised along with the 'message' written in the assert statement. If 'message' is not given in the assert statement, and the condition is False, then also AssertionError is raised without message.

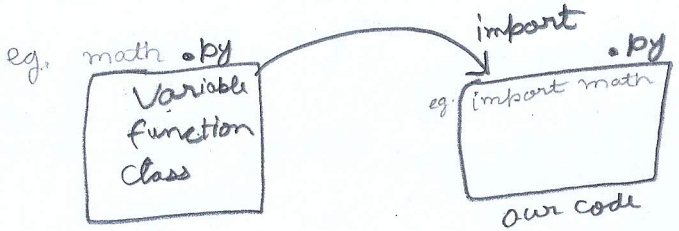
Handling AssertionError with message.

```
try:
    x = int(input('Enter a number between 5 and 10 '))
    assert x>=5 and x<=10, "Your input is incorrect"
    print('The number entered: ', x)
except AssertionErr as obj:
    print(obj)
```

o/p: Enter a number between 5 and 10 15
Your input is incorrect

Modules: Introduction, Importing Modules.

Module is a Python file consisting of Python code.
Module can define functions, classes and variables.



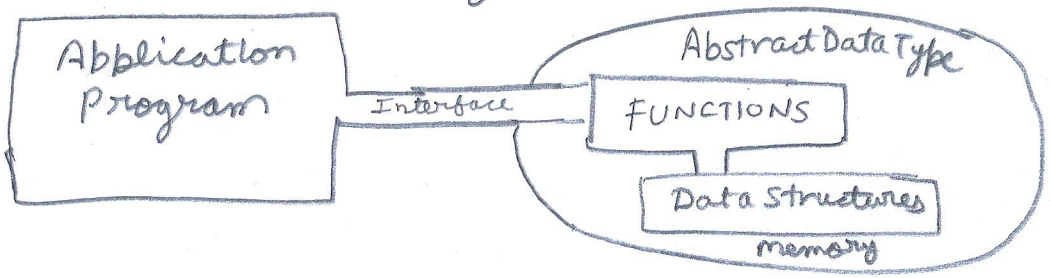
eg. math module contains factorial function
∴ we import :

```
>>> import math
>>> math.factorial(5)
120
>>> math.sqrt(25)
5.0
>>> math.pi
3.141592653589793
```

```
>>> from math import factorial
>>> factorial(5)
120

>>> import math as m
>>> m.factorial(5)
120

>>> from math import factorial, sqrt
>>> sqrt(81)
9.0
>>> factorial(9)
362880
```

Classes are the Python representation for "Abstract Data Types" (ADT)

An ADT involves both data and operations on that data.
ADT only mentions what operations are to be performed but not how these operations will be implemented. It does not specify how data will be organised in memory and what algorithms will be used for implementing the operations. It is called "abstract" because it gives an implementation-independent view. The process of providing only the essentials and hiding the details is known as abstraction.



CLASSES & OBJECTS.

Data: numbers, strings, list...

Not all of data / information / things can be represented using strings, numbers, boolean.

eg. a phone, a computer, a person... ie data types available are not covering these

✓ So what we do in classes and objects is that we can create our own data types. eg. mobile phone datatype to represent a phone. So I can store all the information I want to know about my phone inside of that datatype. In Python we create a class for it.

Student.py

```
Class Student:
    def __init__(self, name, major, gpa, isOnProbation):
        self.name = name
        self.major = major
        self.gpa = gpa
        self.isOnProbation = isOnProbation
```

self is referring to actual object eg. Student1

the name we passed in eg. "Jack"

the major

Class is a overall template, i.e. it defines what a Student is.

An object is an actual Student with actual information

Attributes of a class are (variable + function)

eg self.name is attribute of class Student

app.py

```
from Student import Student
Student1 = Student("Jack", "Law", 3.4, False)   # a Student object will be created,
                                                # object is just an instance of a class.
print(Student1.name)
print(Student1.gpa)
Student2 = Student("Jill", "Math", 3.6, True)
print(Student2.gpa)
```

Student Object

We can access attributes from inside of this object.

Inheritance : We can define a bunch of attributes inside of a class. And then we create another class and inherit all of those attributes.

So I can have one class that can have all the functionality of another class without having to physically write out any of the same attributes.

(i.e variables (fields) } Attribute
function (method) }

chef.py

```
class Chef:
    def roti(self):
        print("chef makes roti")
    def dal(self):
        print("chef makes dal")
    def sambhar(self):
        print("chef makes sambhar")
    def kadhi(self):
        print("chef makes generic kadhi")
```

app.py

```
from chef import Chef
C = Chef()  # object c of class Chef created
C.roti()    # service for object c

from chineseChef import ChineseChef
CC = ChineseChef()  # object cc of class ChineseChef created
CC.special()  # service for object c

C.kadhi()
CC.kadhi()  # override the service of
            # Chef class.
```

chineseChef.py

```
from chef import Chef
class ChineseChef(Chef):  # 4 services of Chef copied
    def special(self):
        print("chinese chef makes manchurian")
    def kadhi(self):  # override dish
        print("chinese chef makes thick gravy kadhi!!!")
```

chinese chef makes thick gravy kadhi!!!

(i.e. we are considering the
def kadhi(self): of
chineseChef.py
(not of chef.py)
(in overriding)

Class Functions can modify objects of that class, or give us specific information about that object.

student.py

```
class Student:
    def __init__(self, name, major, gpa):
        self.name = name
        self.major = major
        self.gpa = gpa

    def onHonor(self):
        if self.gpa >= 3.6:
            return True
        else:
            return False
```

I can define a function inside this Student class, then all of my Student objects can access it.

this function (method) provides service to the objects of this class

app.py

```
from student import Student
Student1 = Student("Jack", "Law", 3.4, False)
Student2 = Student("Jill", "Math", 3.6, True)
print(Student2.onHonor())
```