UNITED
GROUP OF INSTITUTIONS

Python Programming
Unit 4
(KNC-302)

Prepared By
Abhishek Kesharwani
Assistant Professor,UCER Naini,Allahabad

UNIT 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.

Sieve of Eratosthenes

Algorithm to find all the prime numbers less than or equal to a given integer *n* by Eratosthenes' method

1. Create a list of consecutive integers from 2 to *n*: (2, 3, 4, …, *n*).
2. Initially, let *p* equal 2, the first prime number.
3. Starting from p^2, count up in increments of *p* and mark each of these numbers greater than or equal to p^2 itself in the list. These numbers will be $p(p+1), p(p+2), p(p+3)$.

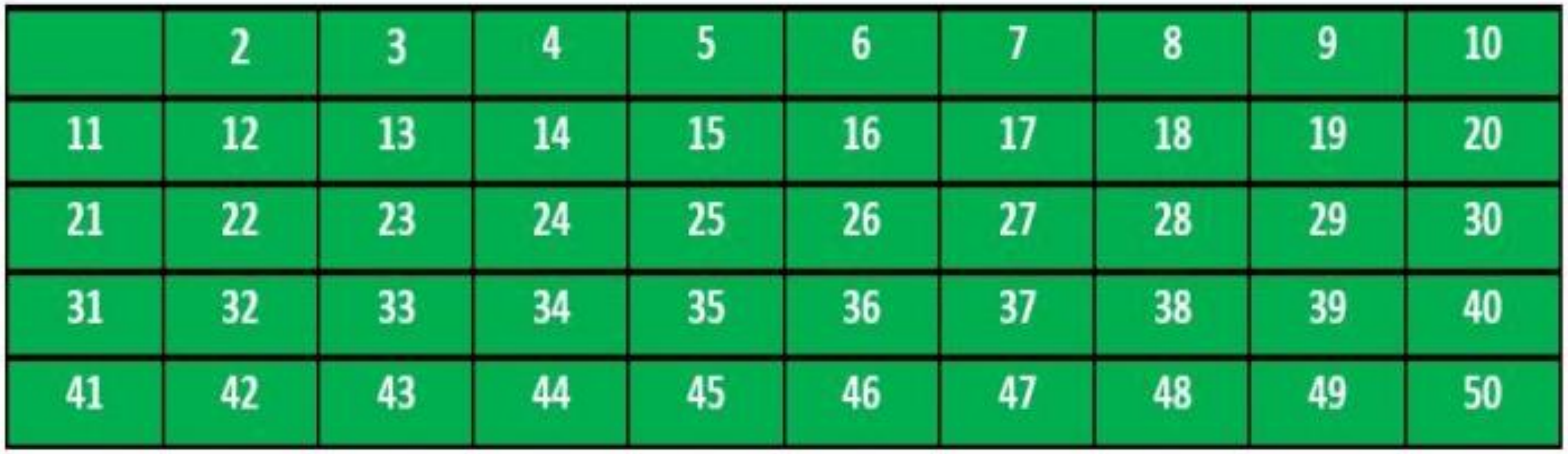
Sieve of Eratosthenes

4. Find the first number greater than *p* in the list that is not marked. If there was no such number, stop. Otherwise, let *p* now equal this number (which is the next prime), and repeat from step 3.

Example:

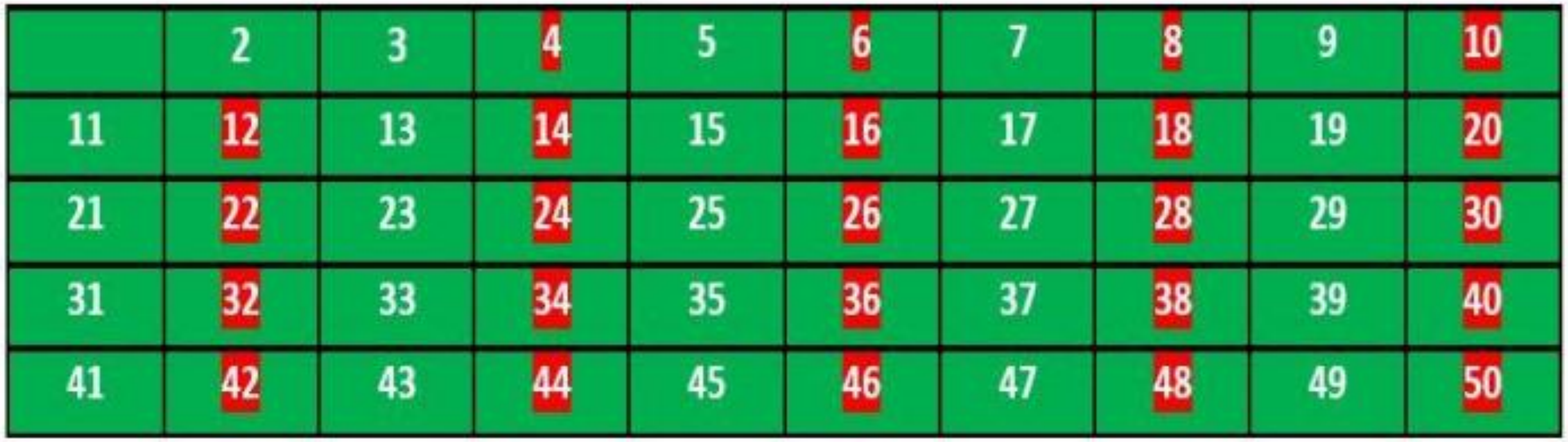
Let us take an example when n = 50. So we need to print all print numbers smaller than or equal to 50.

We create a list of all numbers from 2 to 50.

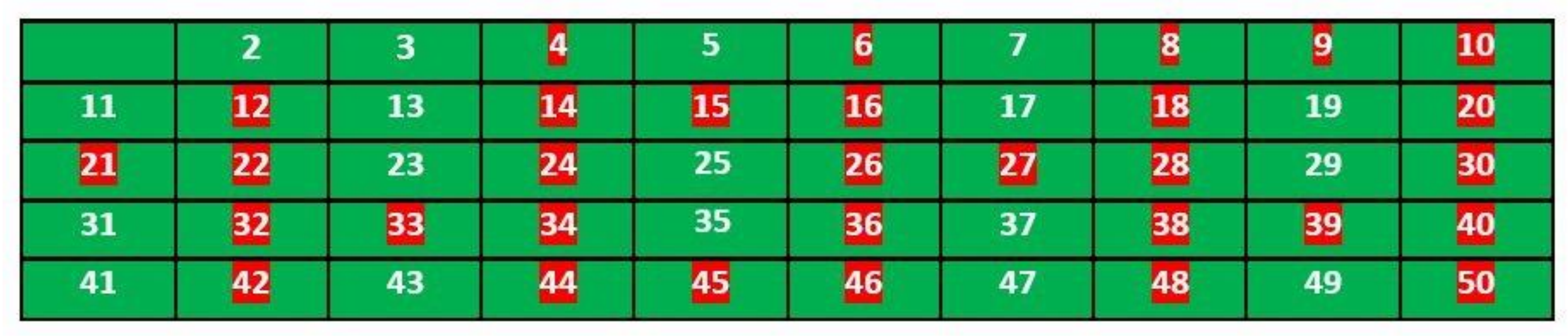


	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

According to the algorithm we will mark all the numbers which are divisible by 2.

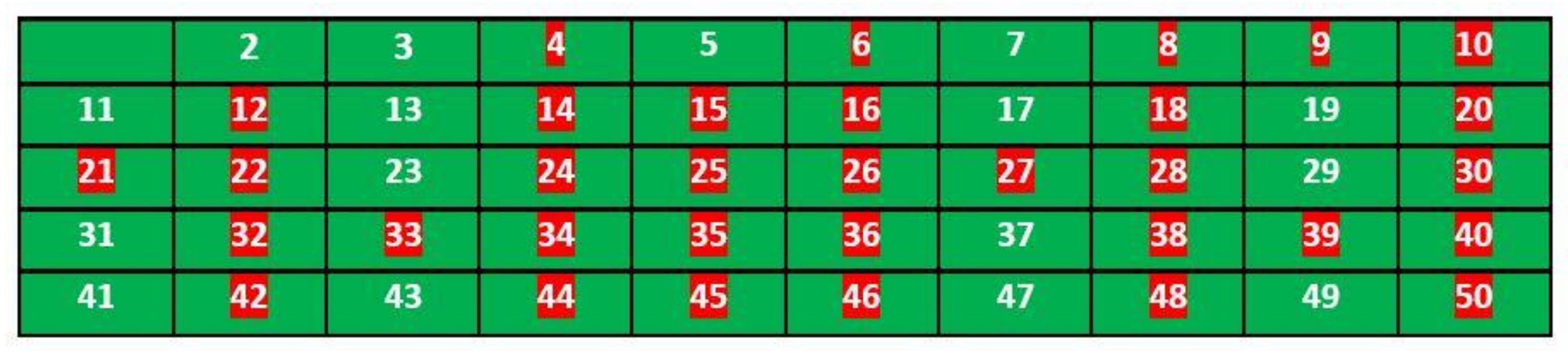


	2	3	🟥 4	5	🟥 6	7	🟥 8	9	🟥 10
11	🟥 12	13	🟥 14	15	🟥 16	17	🟥 18	19	🟥 20
21	🟥 22	23	🟥 24	25	🟥 26	27	🟥 28	29	🟥 30
31	🟥 32	33	🟥 34	35	🟥 36	37	🟥 38	39	🟥 40
41	🟥 42	43	🟥 44	45	🟥 46	47	🟥 48	49	🟥 50



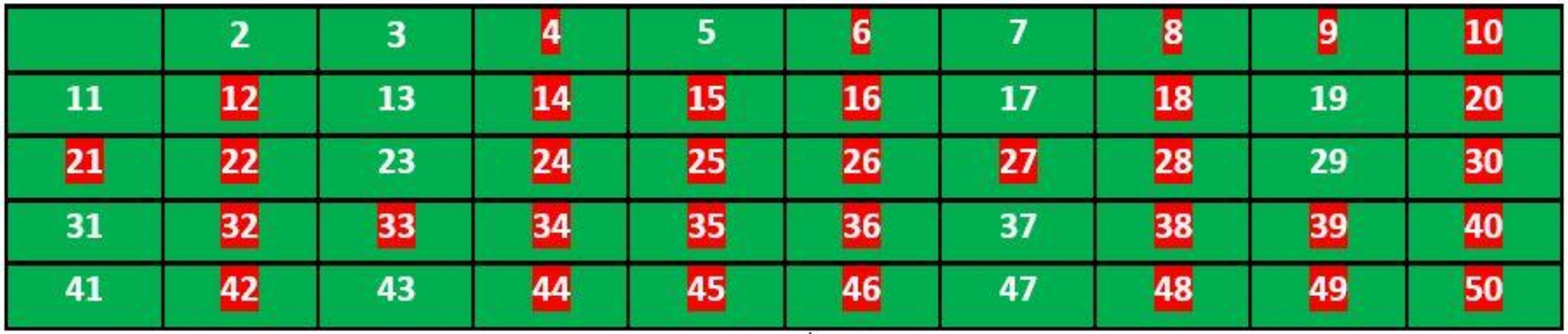
	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Now we move to our next unmarked number 3 and mark all the numbers which are multiples of 3 and are greater than or equal to the square of it.



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

We move to our next unmarked number 5 and mark all multiples of 5 and are greater than or equal to the square of it.



	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

So the prime numbers are the unmarked ones: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47.

Sieve of Eratosthenes

```
def sieve(n):
    global primes
    primes=[True]*(n+1)
    primes[0]=False
    primes[1]=False

    for j in range(2,n+1):
        if primes[j]==False:
            continue
        for i in range (j*j,n+1,j):
            primes[i]=False
```

```
n=int(input('input n'))
sieve(n)
for i in range (2,n+1):
    if primes[i]:
        print(i,end=' ')
```

Output

```
input n 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59
   61 67 71 73 79 83 89 97
```

Python File Open

- File handling is an important part of any web application.
- Python has several functions for creating, reading, updating, and deleting files.

The key function for working with files in Python is the open() function.

The open() function takes two parameters; filename, and mode.

Create a New File

To create a new file in Python, use the open() method, with one of the following parameters:

- "x" - Create - will create a file, returns an error if the file exist
- "a" - Append - will create a file if the specified file does not exist
- "w" - Write - will create a file if the specified file does not exist

There are four different methods (modes) for opening a file:

- "r" - Read - Default value. Opens a file for reading, error if the file does not exist
- "a" - Append - Opens a file for appending, creates the file if it does not exist
- "w" - Write - Opens a file for writing, creates the file if it does not exist
- "x" - Create - Creates the specified file, returns an error if the file exists

In addition you can specify if the file should be handled as binary or text mode

- "t" - Text - Default value. Text mode
- "b" - Binary - Binary mode (e.g. images)

Syntax

To open a file for reading it is enough to specify the name of the file:

```
f = open("demofile.txt")
```

```
f = open("demofile.txt", "rt")
```

Because "r" for read, and "t" for text are the default values, you do not need to specify them.

Note: Make sure the file exists, or else you will get an error.

Opening a file

- Python provides the open() function which accepts two arguments, file name and access mode in which the file is accessed.
- The function returns a file object which can be used to perform various operations like reading, writing, etc.

Example

```
#opens the file file.txt in read mode
fileptr = open("file.txt","r")

if fileptr:
    print("file is opened successfully")
```

The close() method

- Once all the operations are done on the file, we must close it through our python script using the close() method.
- Any unwritten information gets destroyed once the close() method is called on a file object.

Reading the file

- To read a file using the python script, the python provides us the read() method.
- The read() method reads a string from the file. It can read the data in the text as well as binary format.

Example

```
fileptr = open("file.txt","r");

content = fileptr.read();
print(type(content))

print(content)

fileptr.close()
```

Read Only Parts of the File

By default the read() method returns the whole text, but you can also specify how many characters you want to return:

Example

- Return the 5 first characters of the file:

```
f = open("demofile.txt", "r")
print(f.read(5))
```

Looping through the file

```
fileptr = open("file.txt","r");
for i in fileptr:
    print(i) # i contains each line of the file
```

By looping through the lines of the file, you can read the whole file, line by line:

Write to an Existing File

To write to an existing file, you must add a parameter to the open() function:

"a" - Append - will append to the end of the file

"w" - Write - will overwrite any existing content

Delete a File

To delete a file, you must import the OS module, and run its os.remove() function:

Example

Remove the file "demofile.txt":

```
import os
os.remove("demofile.txt")
```

Check if File exist

```
import os
if os.path.exists("ak123"):
  os.remove("demofile.txt")
else:
  print("The file does not exist")
```

To avoid getting an error, you might want to check if the file exists before you try to delete it:

Delete Folder

To delete an entire folder, use the os.rmdir() method:

Example

Remove the folder "myfolder":

```
import os
os.rmdir("myfolder")
```

File Pointer positions

- Python provides the tell() method which is used to print the byte number at which the file pointer exists.

Modifying file pointer position

- In the real world applications, sometimes we need to change the file pointer location externally since we may need to read or write the content at various locations.
- seek() method which enables us to modify the file pointer position externally.

Example

```
fileptr = open("file2.txt","r")

print("The filepointer is at byte :",fileptr.tell())
 content = fileptr.read();
 print("After reading, the filepointer is at:",filep
tr.tell())
```

Output

```
The filepointer is at byte : 0
After reading, the filepointer is at 26
```

Modifying file pointer position

```
# open the file file2.txt in read mode
fileptr = open("file2.txt","r")

#initially the filepointer is at 0
print("The filepointer is at byte :",fileptr.tell())

#changing the file pointer location to 10.
fileptr.seek(10);

#tell() returns the location of the fileptr.
print("After reading, the filepointer is at:",fileptr.tell())
```

Output

```
The filepointer is at byte : 0
After reading, the filepointer is at 10
```

Exceptions

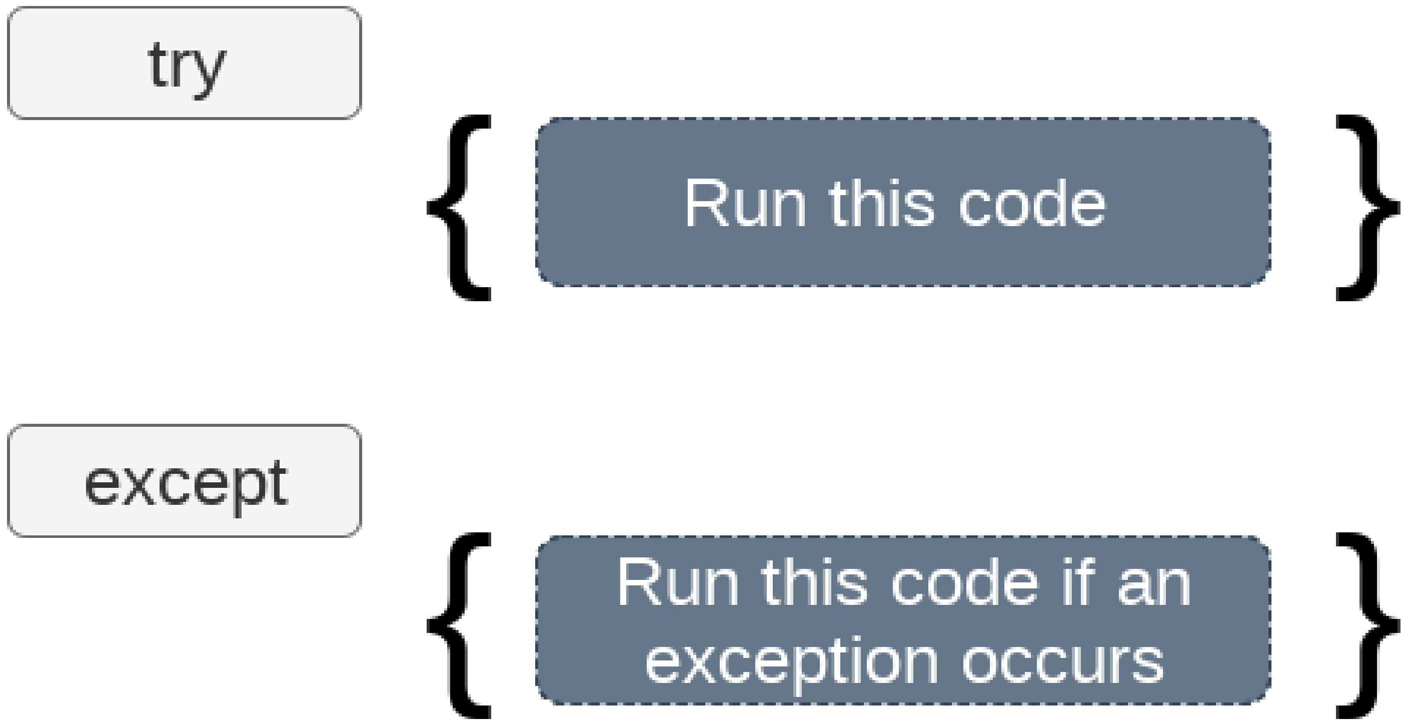
- An exception can be defined as an abnormal condition in a program resulting in the disruption in the flow of the program.
- Whenever an exception occurs, the program halts the execution, and thus the further code is not executed.
- Therefore, an exception is the error which python script is unable to tackle with.
- Python provides us with the way to handle the Exception so that the other part of the code can be executed without any disruption.

Common Exceptions

1. **ZeroDivisionError:** Occurs when a number is divided by zero.
2. **NameError:** It occurs when a name is not found. It may be local or global.
3. **IndentationError:** If incorrect indentation is given.
4. **IOError:** It occurs when Input Output operation fails.
5. **EOFError:** It occurs when the end of the file is reached, and yet operations are being performed.

Exception handling in python

- If the python program contains suspicious code that may throw the exception, we must place that code in the try block.
- The try block must be followed with the except statement which contains a block of code that will be executed if there is some exception in the try block.


try
Run this code
except
Run this code if an exception occurs

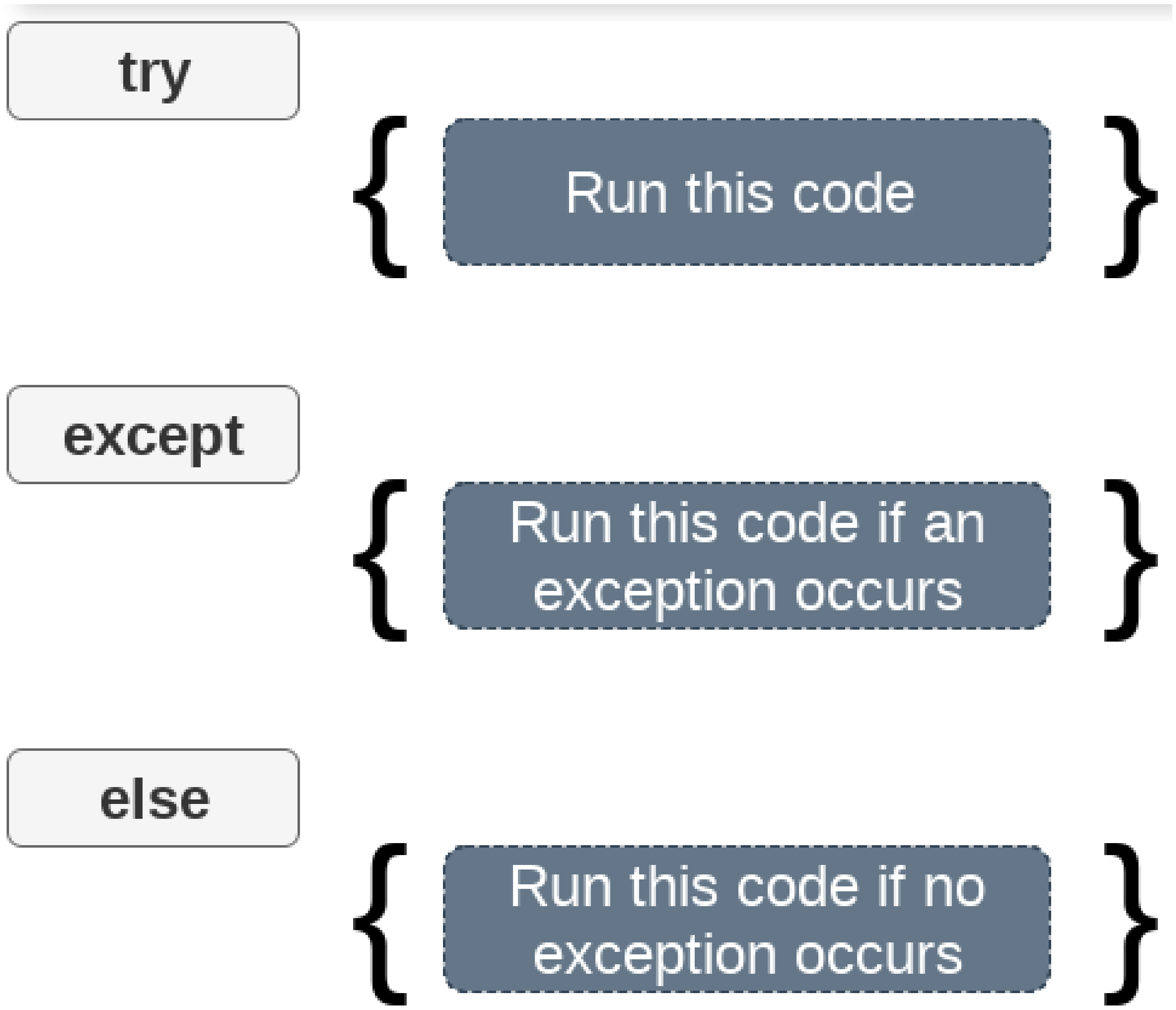
Syntax

```
try:
    #block of code

except Exception1:
    #block of code

except Exception2:
    #block of code

#other code
```


try
Run this code
except
Run this code if an exception occurs
else
Run this code if no exception occurs

try except else

- We can also use the else statement with the try-except statement in which, we can place the code which will be executed in the scenario if no exception occurs in the try block.

Example

```
try:
    a = int(input("Enter a:"))
    b = int(input("Enter b:"))
    c = a/b;
    print("a/b = %d" %c)
except Exception:
    print("can't divide by zero")
else:
    print("Hi I am else block")
```

Output

```
Enter a:10
Enter b:2
a/b = 5
Hi I am else block
```

Points to remember

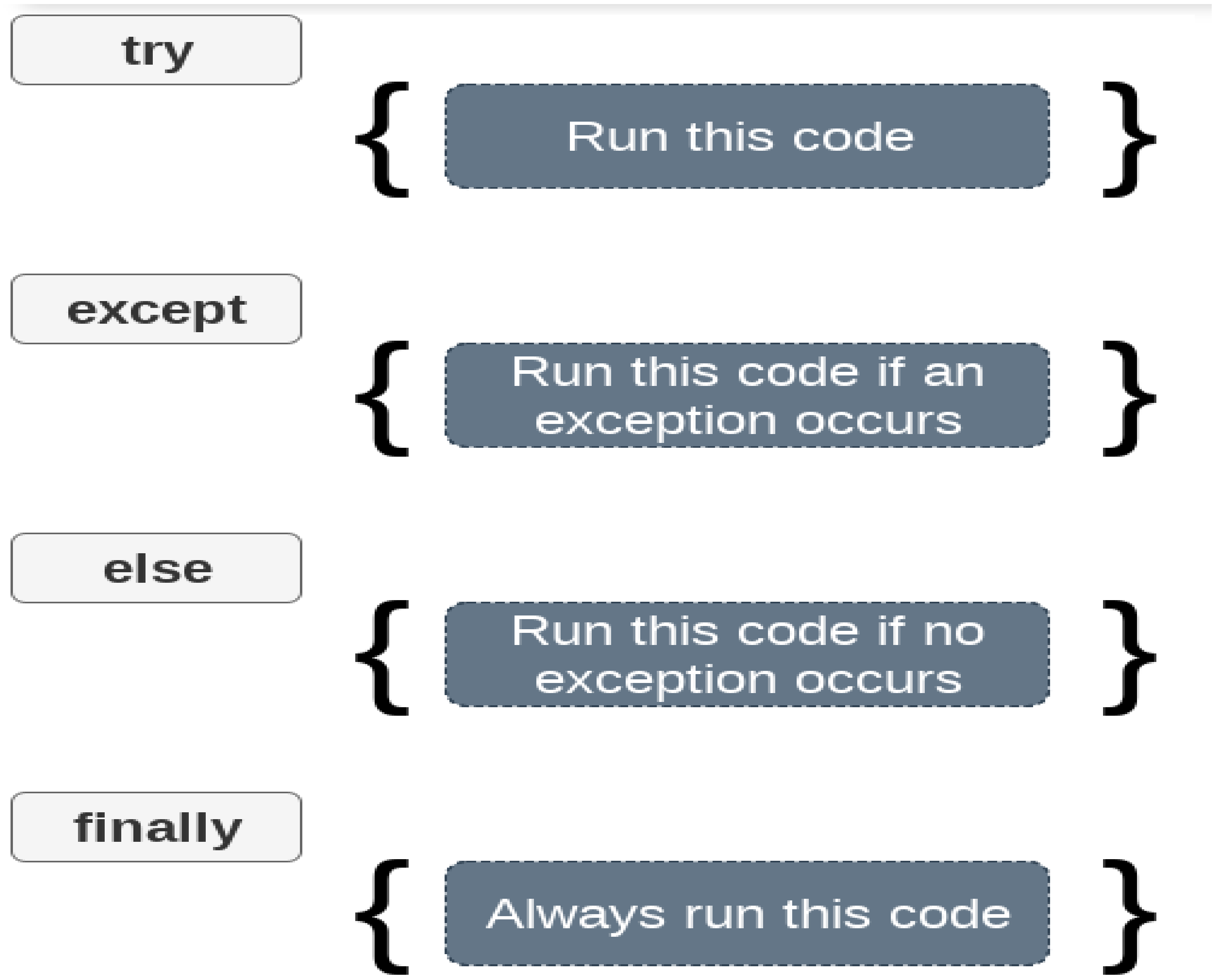
- We can declare multiple exceptions in the except statement since the try block may contain the statements which throw the different type of exceptions.
- We can also specify an else block along with the try-except statement which will be executed if no exception is raised in the try block.
- The statements that don't throw the exception should be placed inside the else block.

Example

```
try:
    #this will throw an exception if the f
ile doesn't exist.
    fileptr = open("file.txt","r")
except IOError:
    print("File not found")
else:
    print("The file opened successfully")
    fileptr.close()
```

The finally block

- We can use the finally block with the try block in which, we can place the important code which must be executed before the try statement throws an exception.


try
{ Run this code }
except
{ Run this code if an exception occurs }
else
{ Run this code if no exception occurs }
finally
{ Always run this code }

Example

```
try:
    a = int(input("Enter a:"))
    b = int(input("Enter b:"))
    c = a/b;
    print("a/b = %d"%c)
except:
    print("can't divide by zero")
else:
    print("Hi I am else block")
finally:
    print("finally block is always
executed")
```

Declaring multiple exceptions

```
try:
    #block of code

except (<Exception 1>,<Exception 2>,<Exception 3>,...<Exception n>)
    #block of code

else:
    #block of code
```

Example

```
try:
    a=10/0;
except ArithmeticError,StandardError:
    print "Arithmetic Exception"
else:
    print "Successfully Done"
```

Raising exceptions

An exception can be raised by using the raise clause in python. The syntax to use the raise statement is given below.

syntax

raise Exception_class,<value>

Example

```
try:
    age = int(input("Enter the age?"))
    if age<18:
        raise ValueError;
    else:
        print("the age is valid")
except ValueError:
    print("The age is not valid")
```

Example

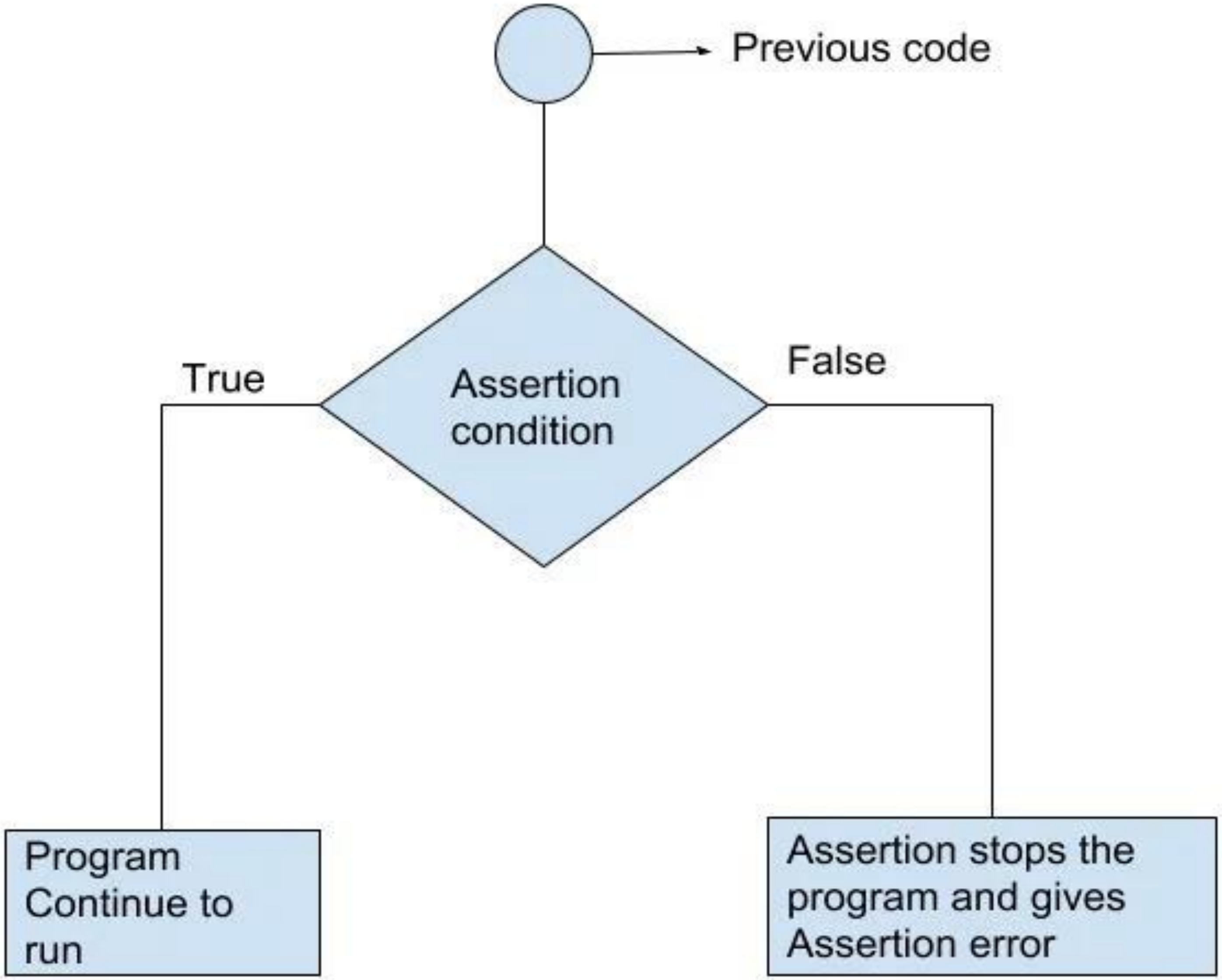
```
try:
    a = int(input("Enter a?"))
    b = int(input("Enter b?"))
    if b is 0:
        raise ArithmeticError
    else:
        print("a/b = ",a/b)
except ArithmeticError:
    print("The value of b can't be 0")
```

Output

```
Enter a 10
Enter b 0
The value of b can't be 0
```

Assertions

- Assertions are statements that assert or state a fact confidently in your program.
- For example, while writing a division function, you're confident the divisor shouldn't be zero, you assert divisor is not equal to zero.
- Assertions are simply boolean expressions that check if the conditions return true or not.
- If it is true, the program does nothing and moves to the next line of code.
- However, if it's false, the program stops and throws an error.


Previous code
Assertion condition
True
False
Program Continue to run
Assertion stops the program and gives Assertion error

Python assert Statement

- Python has built-in assert statement to use assertion condition in the program.
- assert statement has a condition or expression which is supposed to be always true.
- If the condition is false assert halts the program and gives an AssertionError.

Syntax for using Assert in Pyhton:

- assert <condition>
- assert <condition>,<error message>

```python
def avg(marks):
    assert (len(marks) != 0,"List is empty.")
    return sum(marks)/len(marks)

mark2 = [55,88,78,90,79]
print("Average of mark2:",avg(mark2))

mark1 = []
print("Average of mark1:",avg(mark1))
```

```
Average of mark2: 78.0
AssertionError: List is empty.
```

Key Points to Remember

- Assertions are the condition or boolean expression which are always supposed to be true in the code.
- assert statement takes an expression and optional message.
- assert statement is used to check types, values of argument and the output of the function.
- assert statement is used as debugging tool as it halts the program at the point where an error occurs.