6.1 I/O Operations: Reading keyboard input , Printing to screen

Reading Keyboard Input

Python provides two built-in functions to read a line of text from standard input, which by default comes from the keyboard. These functions are −

- raw_input
- input

The *raw_input* Function

The raw_input([prompt]) function reads one line from standard input and returns it as a string.

```
>>> str = raw_input("Enter your input: ")
Enter your input: mmpolytechnic
>>> print(str)
mmpolytechnic
>>>
```

The input Function

The input([prompt]) function is equivalent to raw_input, except that it assumes the input is a valid Python expression and returns the evaluated result to you.

```
>>> str = input("Enter your name: ")
Enter your name: purva
```

Error:

NameError: name 'purva' is not defined(“for string input” instead of input use raw_input for accepting string value from user)

```
>>> str = input("Enter your input: ")
Enter your input: 1
>>> print(str)
1
```

"6.2 File Handling: Opening file in different modes, accessing file contents using standard library functions, Reading and writing files, closing a file, Renaming and deleting file, Directories in python, File and related standard functions

File handling is an important part of any web application.

Python has several functions for creating, reading, updating, and deleting files.

Opening file in different modes

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

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

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

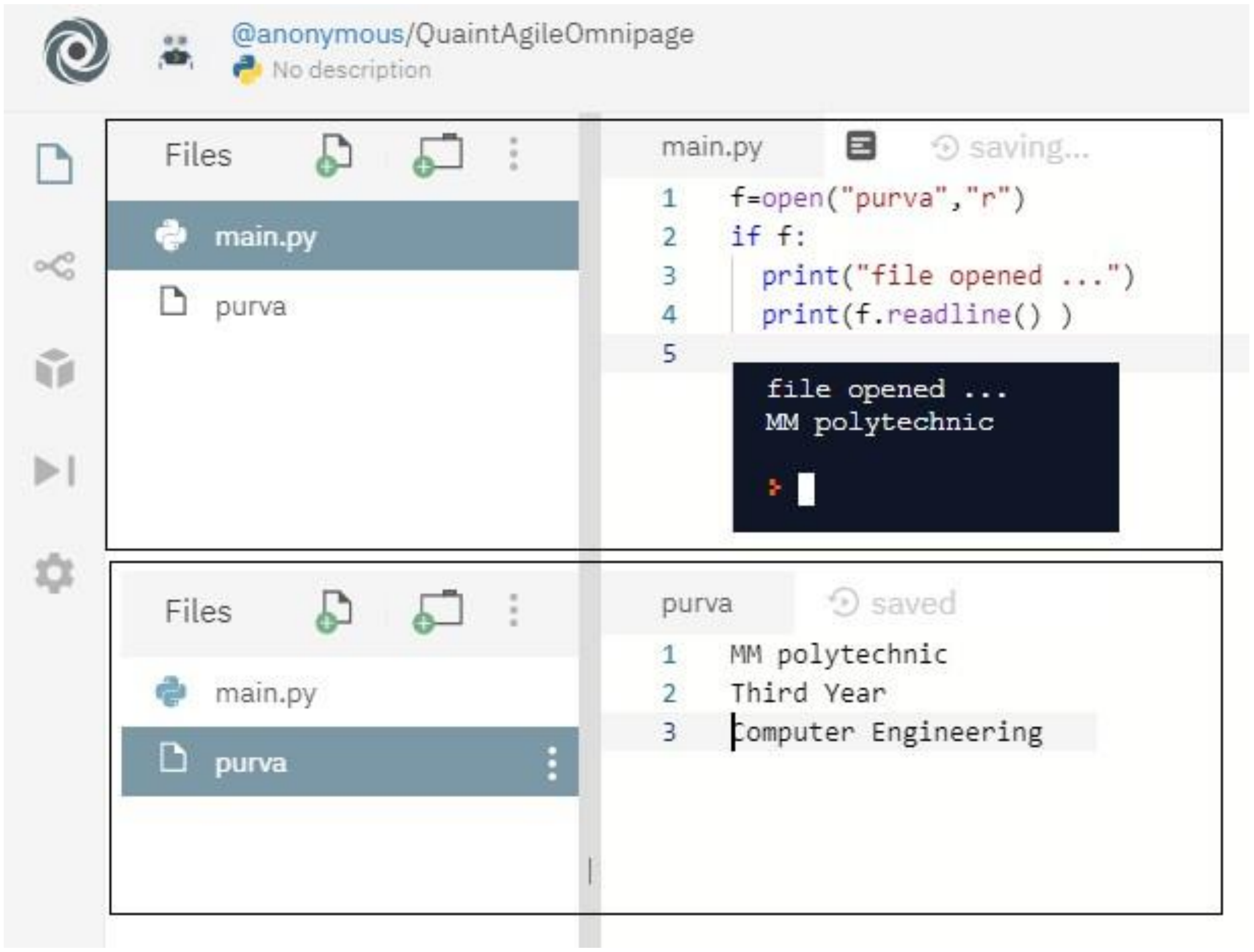
"w" - Write - Opens a file for writing, creates the file if it does not exist

"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)



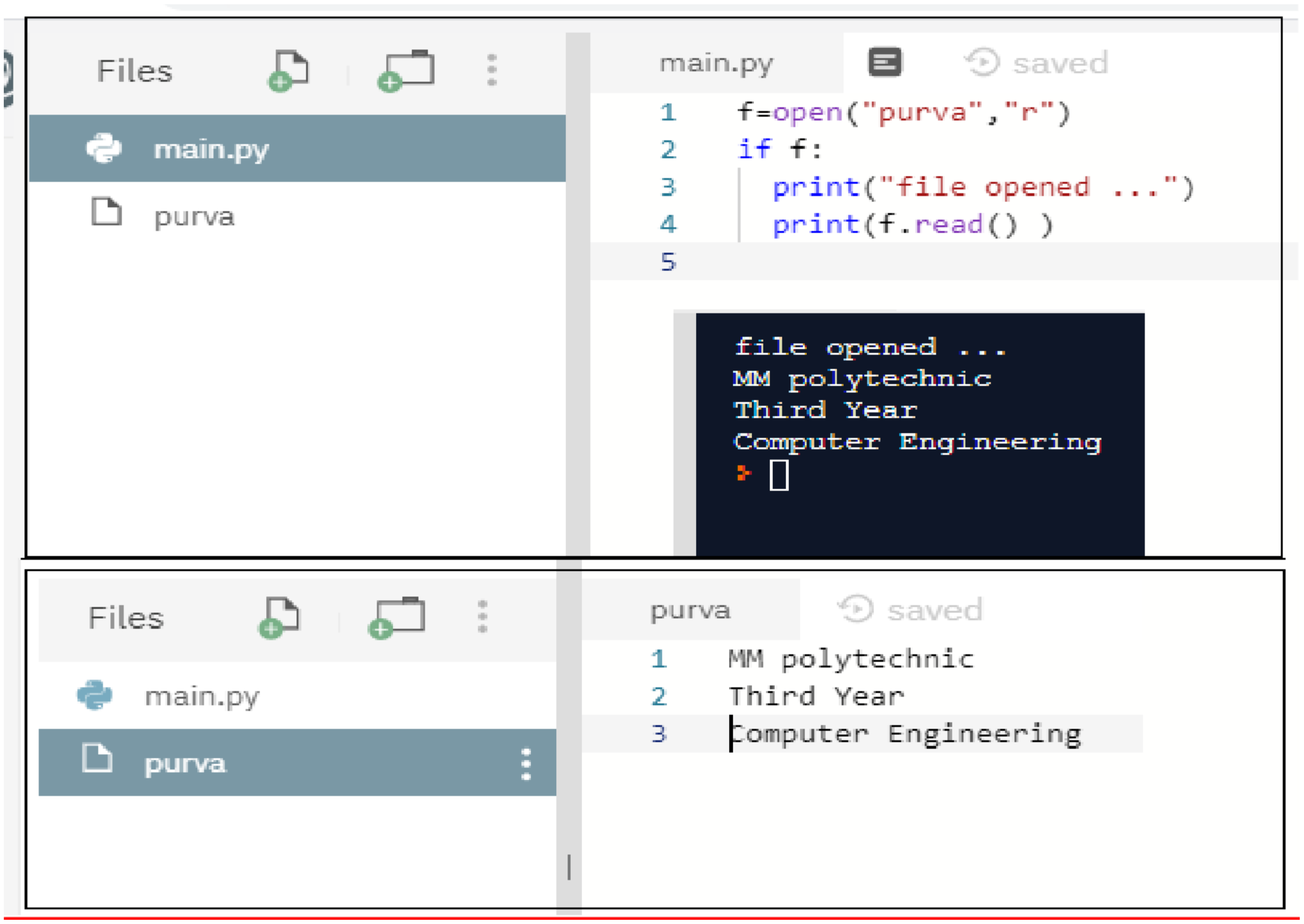
```
f=open("purva","r")
if f:
    print("file opened ...")
    print(f.readline() )
```

```
file opened ...
MM polytechnic

> 
```

```
MM polytechnic
Third Year
Computer Engineering
```

Accessing file contents using standard library functions Open a File on the Server

Assume we have the following file, located in the same folder as Python:



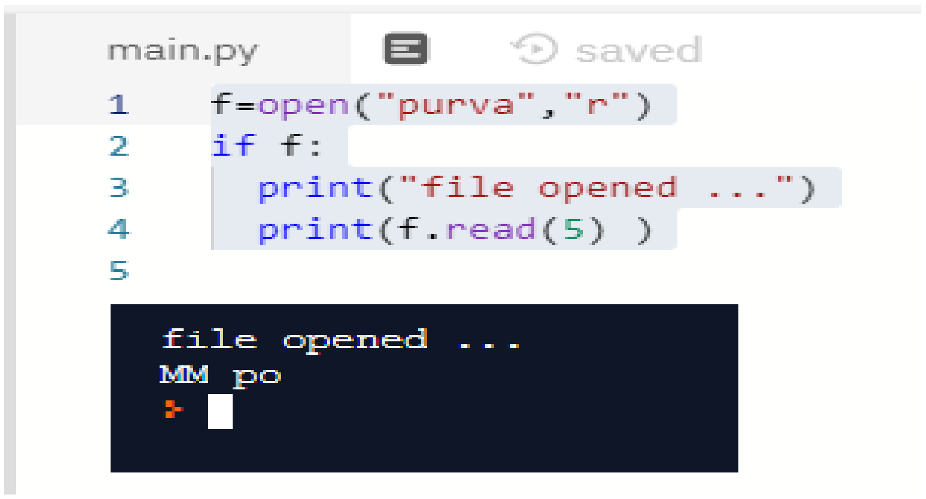
```
f=open("purva","r")
if f:
  print("file opened ...")
  print(f.read() )
```

```
file opened ...
MM polytechnic
Third Year
Computer Engineering
```

```
MM polytechnic
Third Year
Computer Engineering
```

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:

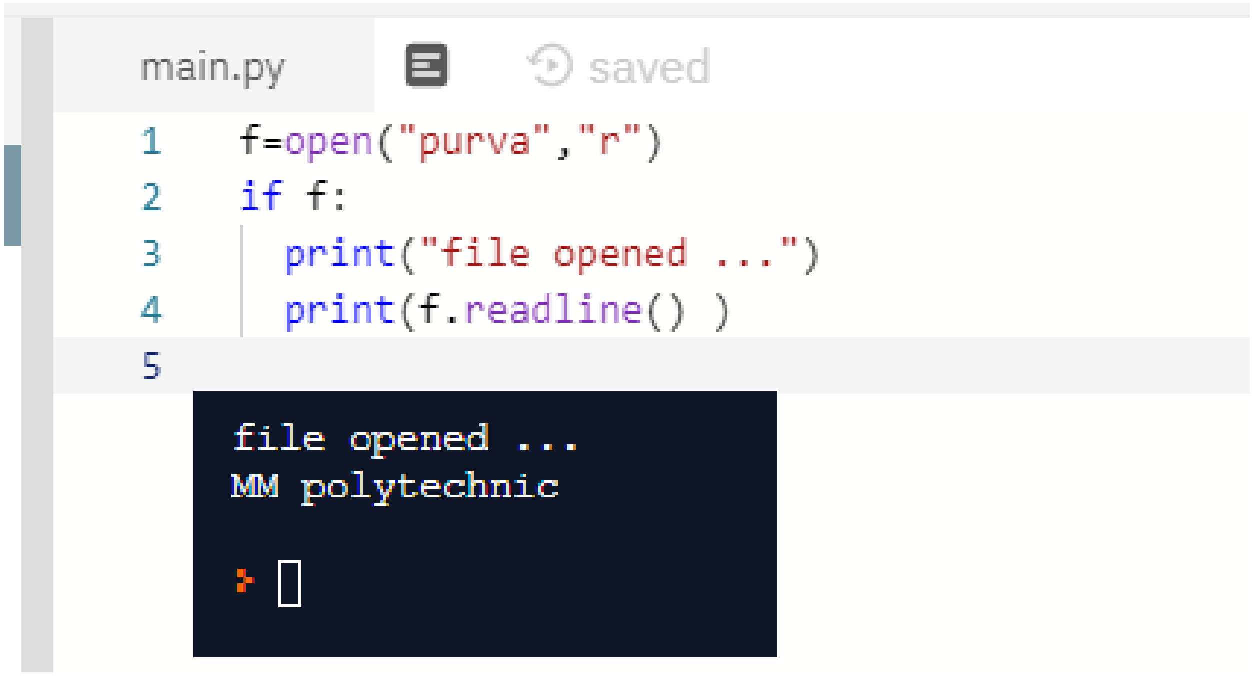


```
f=open("purva","r")
if f:
  print("file opened ...")
  print(f.read(5) )
```

```
file opened ...
MM po
```

Readline

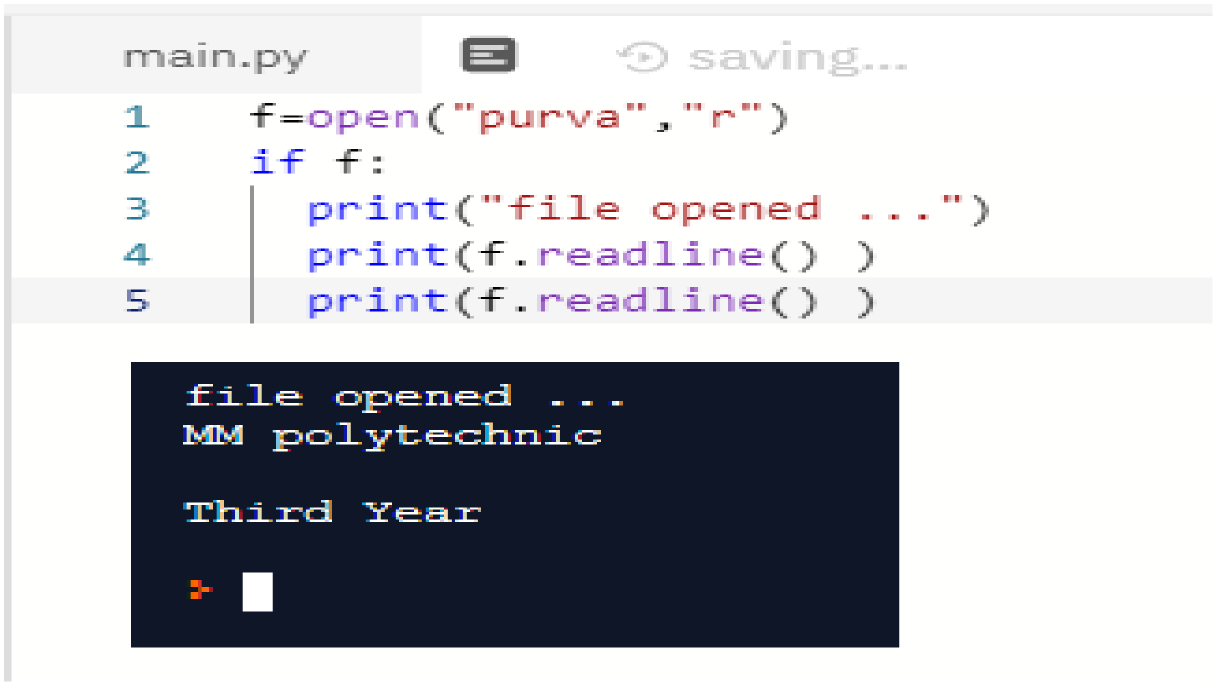
You can return one line by using the readline() method:



```python
f=open("purva","r")
if f:
  print("file opened ...")
  print(f.readline() )
```

```
file opened ...
MM polytechnic

>
```

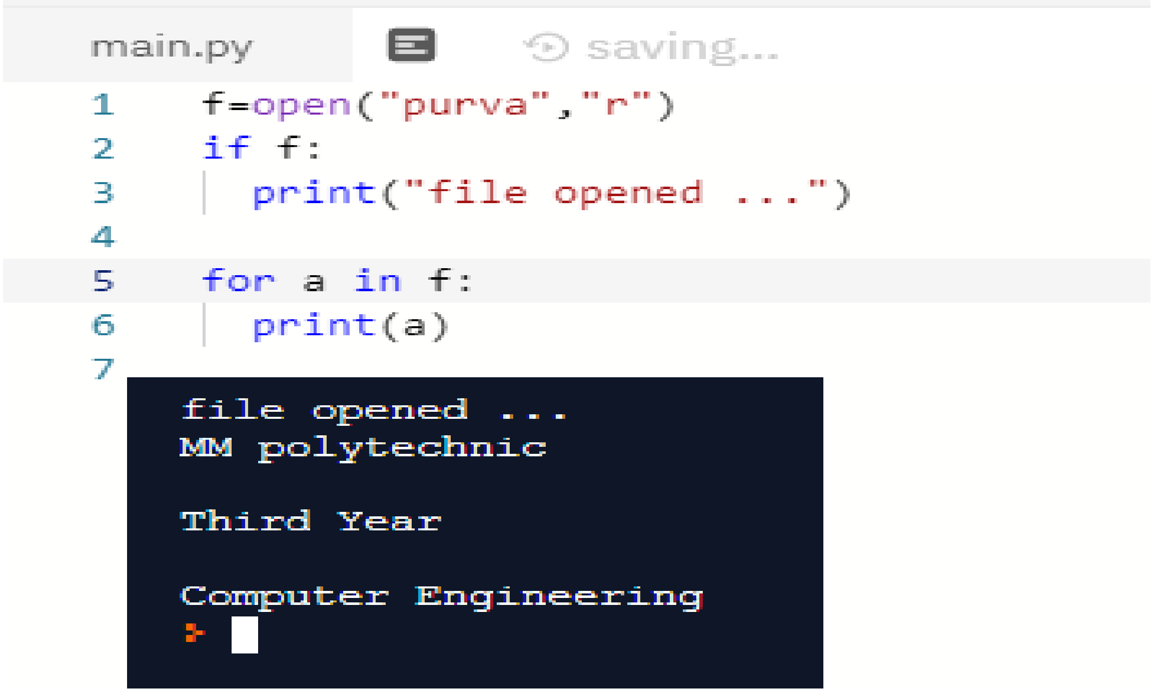
Calling readline() 2 times



```python
f=open("purva","r")
if f:
  print("file opened ...")
  print(f.readline() )
  print(f.readline() )
```

```
file opened ...
MM polytechnic

Third Year

>
```

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

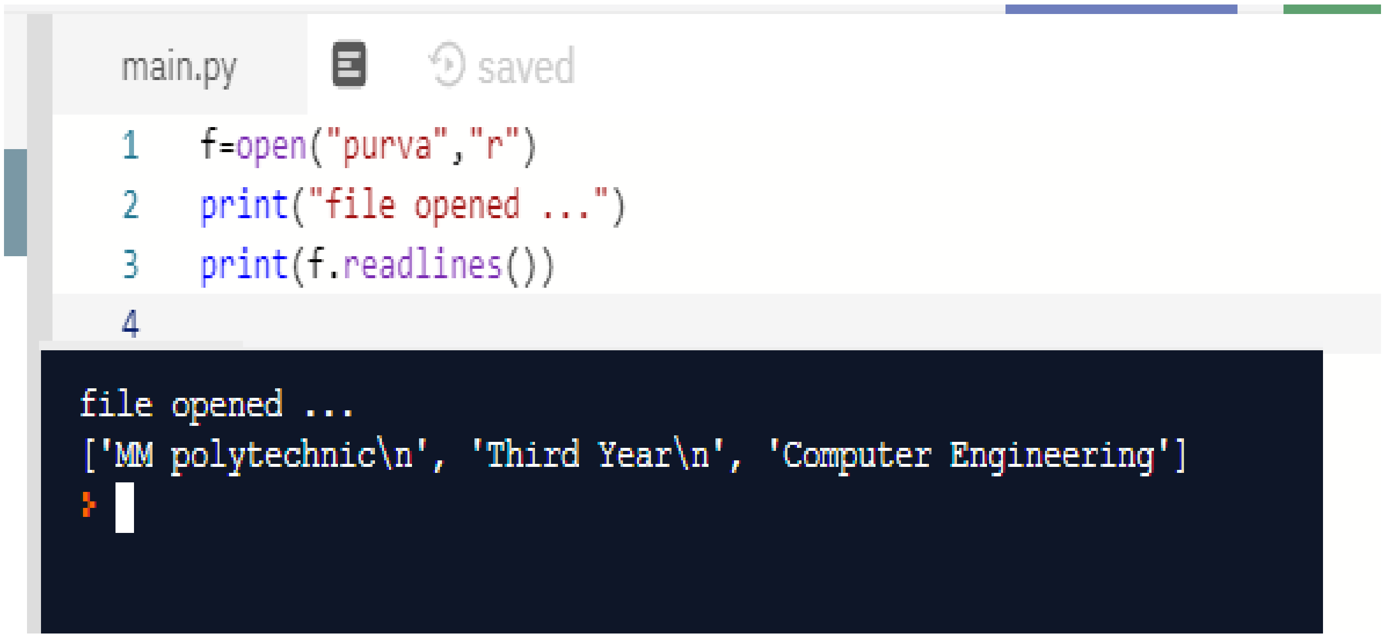


```
f=open("purva","r")
if f:
  print("file opened ...")

for a in f:
  print(a)

```

```
file opened ...
MM polytechnic

Third Year

Computer Engineering
```

Readlines

Read and return a list of lines from the file. Reads in at most n bytes/ characters if specified.



```
f=open("purva","r")
print("file opened ...")
print(f.readlines())

```

```
file opened ...
['MM polytechnic\n', 'Third Year\n', 'Computer Engineering']
```

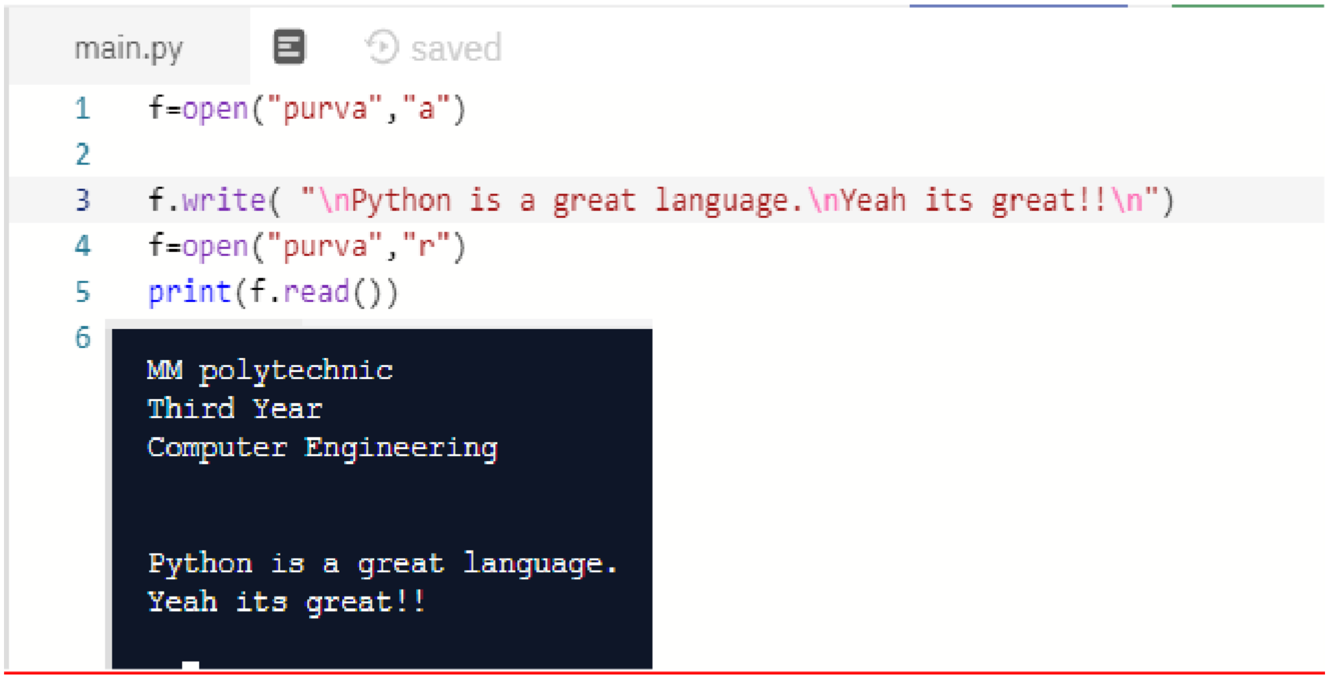
Reading and writing files

The write() Method

The write() method writes any string to an open file.

Python strings can have binary data and not just text.

The write() method does not add a newline character ('\n') to the end of the string –



main.py saved

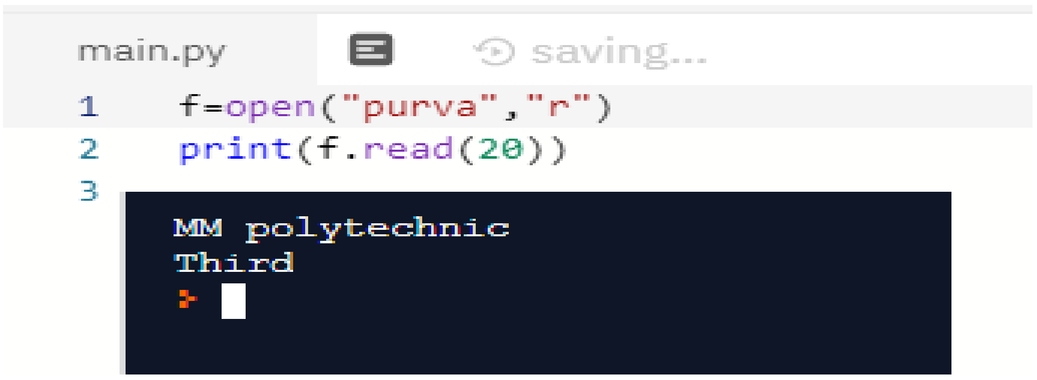
```python
f=open("purva","a")

f.write( "\nPython is a great language.\nYeah its great!!\n")
f=open("purva","r")
print(f.read())
```

```
MM polytechnic
Third Year
Computer Engineering


Python is a great language.
Yeah its great!!
```

The read() Method

The read() method reads a string from an open file.

This method starts reading from the beginning of the file and if count is missing, then it tries to read as much as possible, maybe until the end of file.



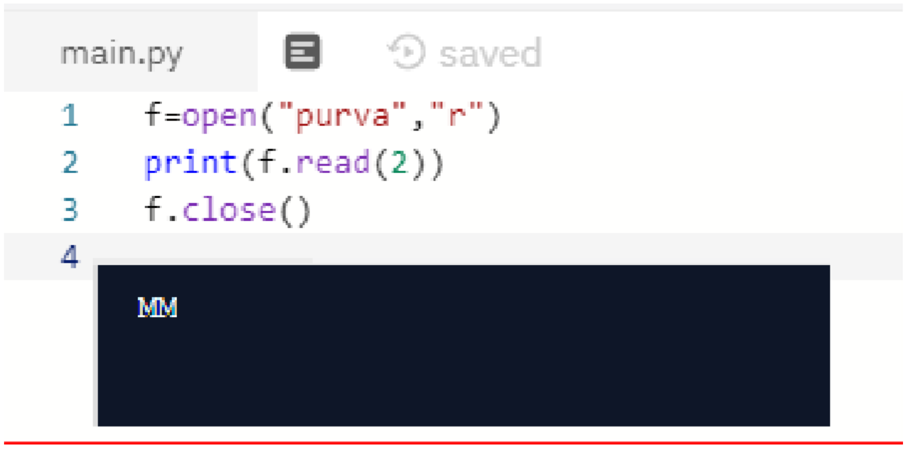
main.py saving...

```python
f=open("purva","r")
print(f.read(20))
```

```
MM polytechnic
Third
```

Closing a file

Python automatically closes a file when the reference object of a file is reassigned to another file. It is a good practice to use the close() method to close a file.



```python
f=open("purva","r")
print(f.read(2))
f.close()
```

```
MM
```

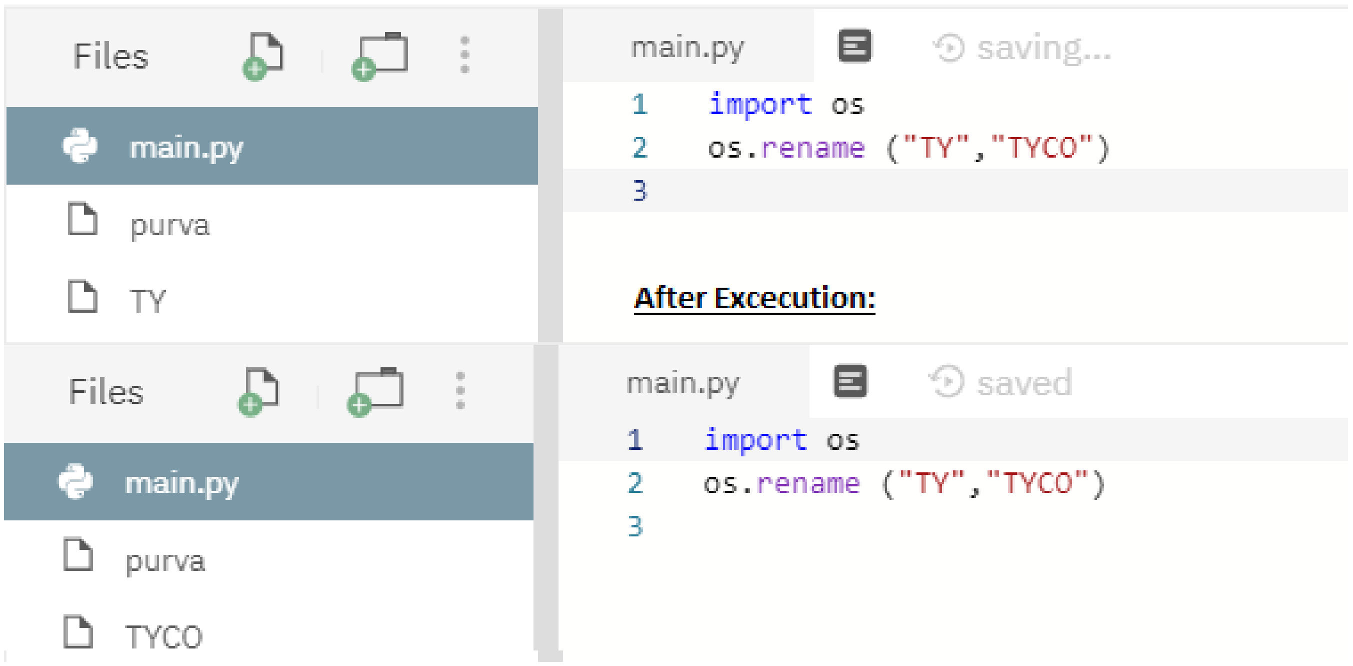
Renaming and deleting file

Python os module provides methods that help you perform file-processing operations, such as renaming and deleting files.

To use this module you need to import it first and then call any related functions.

The rename() Method

The rename() method takes two arguments, the current filename and the new filename.



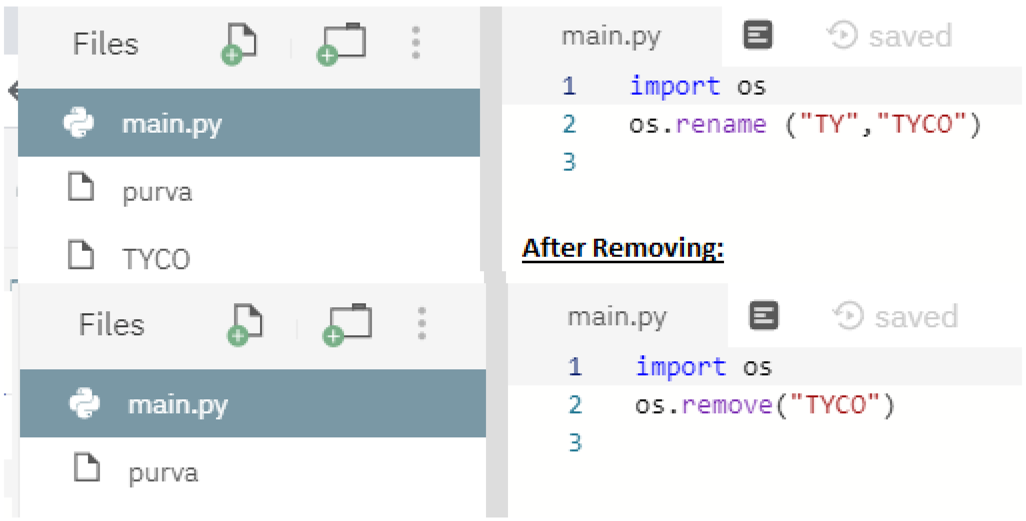
```
import os
os.rename ("TY","TYCO")
```

After Excecution:

```
import os
os.rename ("TY","TYCO")
```

The remove() Method

You can use the remove() method to delete files by supplying the name of the file to be deleted as the argument.



```
import os
os.rename ("TY","TYCO")
```

After Removing:

```
import os
os.remove("TYCO")
```

"6.3 Exception Handling:

An exception can be defined as an abnormal condition in a program resulting in the disruption in the flow of the program.

Python provides us with the way to handle the Exception so that the other part of the code can be executed without any disruption. However, if we do not handle the exception, the interpreter doesn't execute all the code that exists after that.

Common Exceptions

A list of common exceptions that can be thrown from a normal python program is given below.

ZeroDivisionError: Occurs when a number is divided by zero.

NameError: It occurs when a name is not found. It may be local or global.

IndentationError: If incorrect indentation is given.

IOError: It occurs when Input Output operation fails.

EOFError: It occurs when the end of the file is reached, and yet operations are being performed.

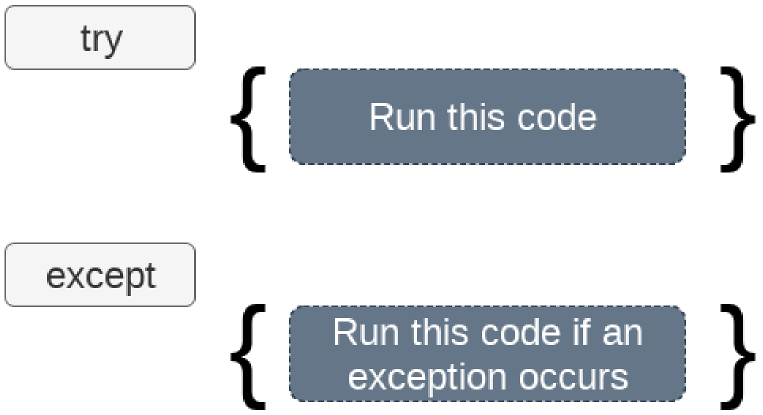
Exception Handling- ‘ try: except:’statement

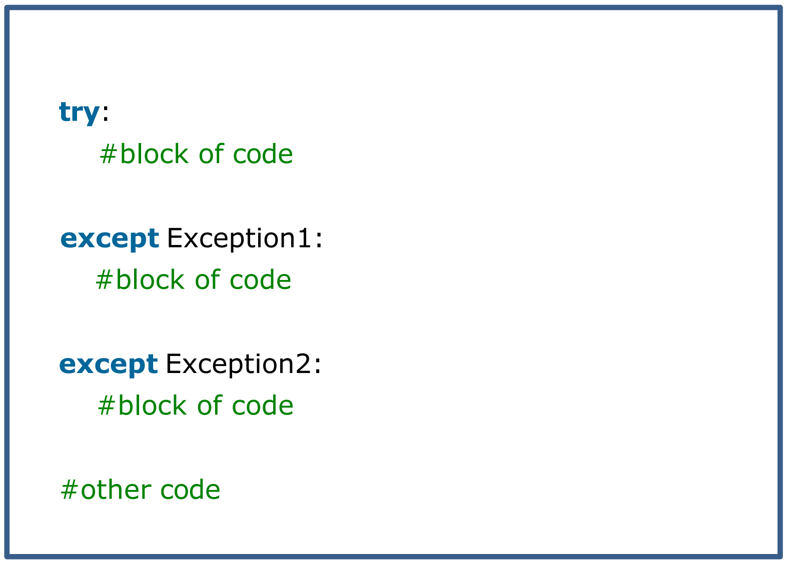
- The **try** block lets you test a block of code for errors.
- The **except** block lets you handle the error.
- The **finally** block lets you execute code, regardless of the result of the try- and except blocks.

When an error occurs, or exception as we call it, Python will normally stop and generate an error message.

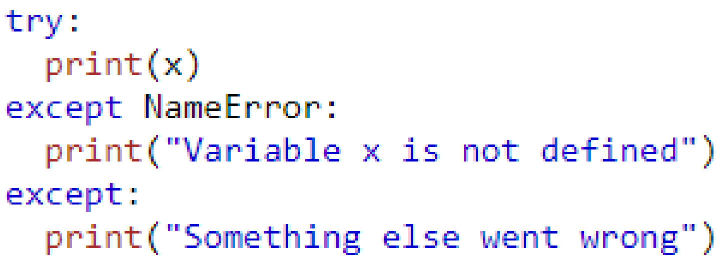
These exceptions can be handled using the try statement:

Using multiple exceptions:

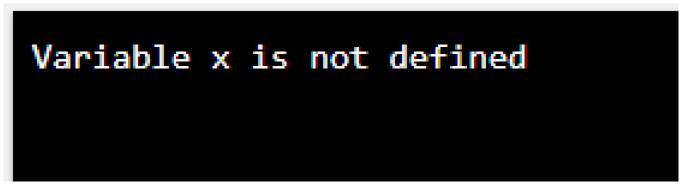




```
try:
    #block of code

except Exception1:
    #block of code

except Exception2:
    #block of code

#other code
```



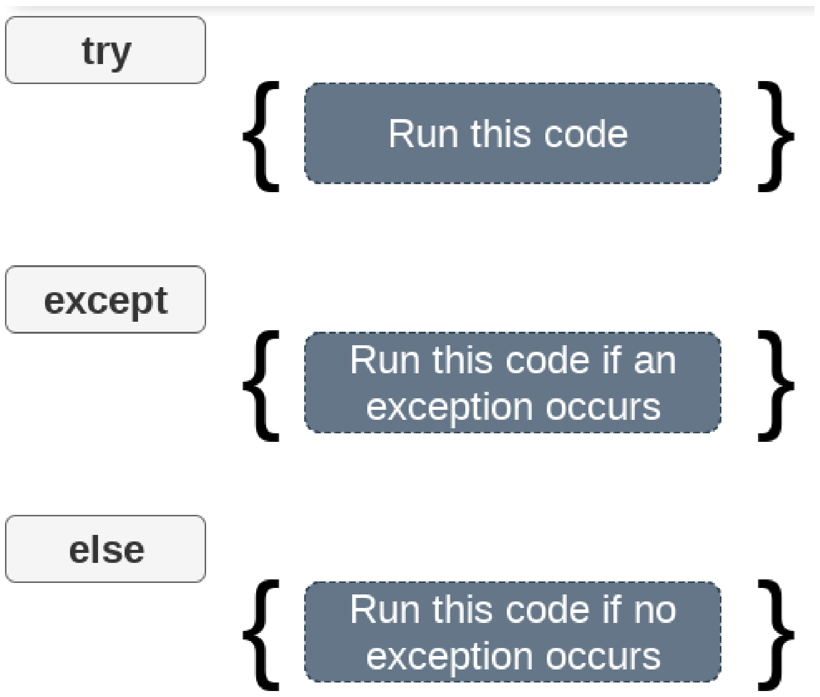
```
try:
  print(x)
except NameError:
  print("Variable x is not defined")
except:
  print("Something else went wrong")
```

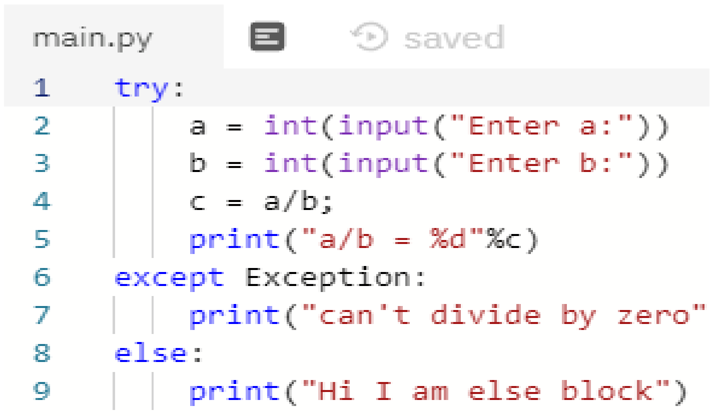


```
Variable x is not defined
```

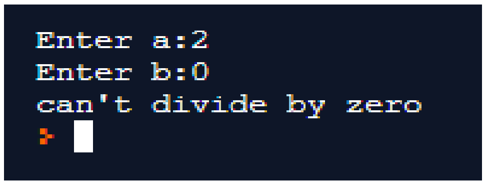
The use the else statement with the try-except statement, place the code which will be executed in the scenario if no exception occurs in the try block.

The syntax to use the else statement with the try-except statement is given below.

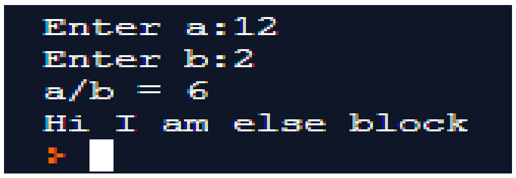

try
Run this code
except
Run this code if an exception occurs
else
Run this code if no exception occurs



```python
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")
```

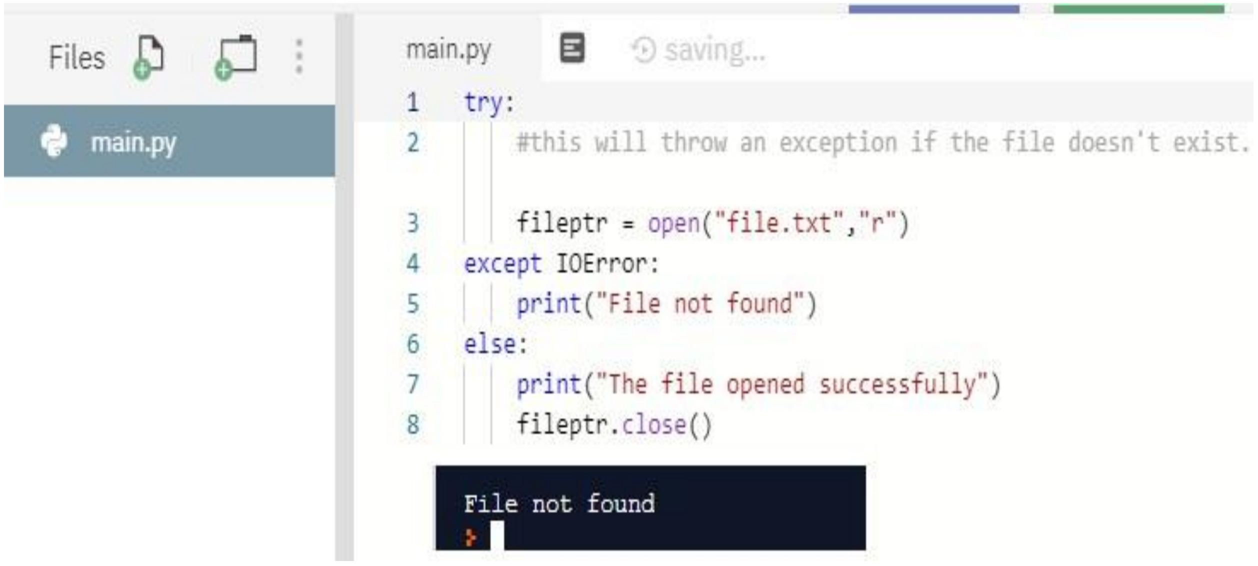


```
Enter a:2
Enter b:0
can't divide by zero
```



```
Enter a:12
Enter b:2
a/b = 6
Hi I am else block
```

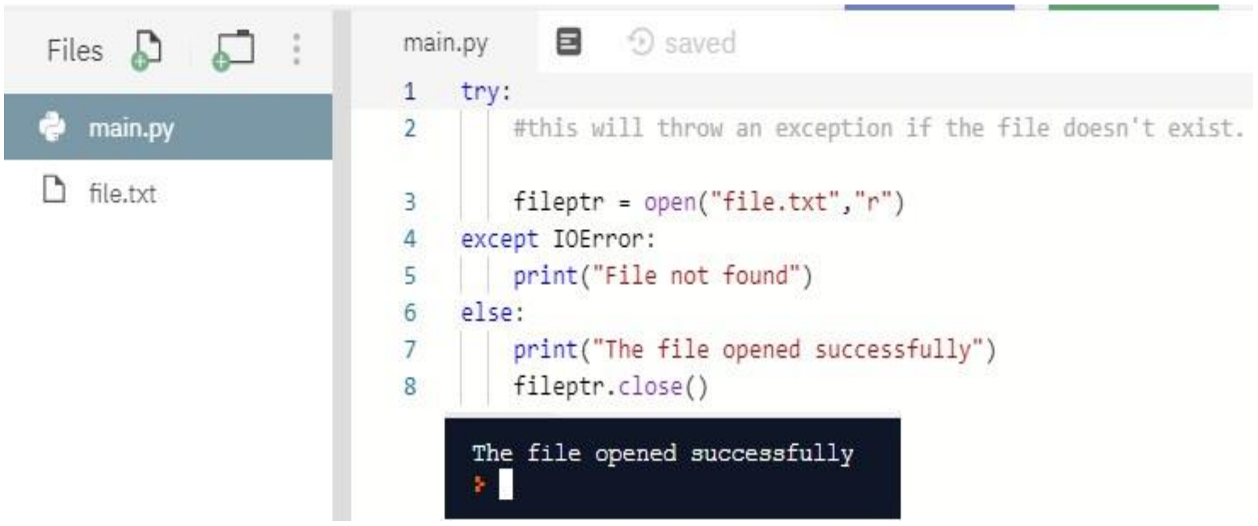
If file not present:



```
try:
    #this will throw an exception if the file doesn't exist.

    fileptr = open("file.txt","r")
except IOError:
    print("File not found")
else:
    print("The file opened successfully")
    fileptr.close()
```

```
File not found
>
```

If file present:



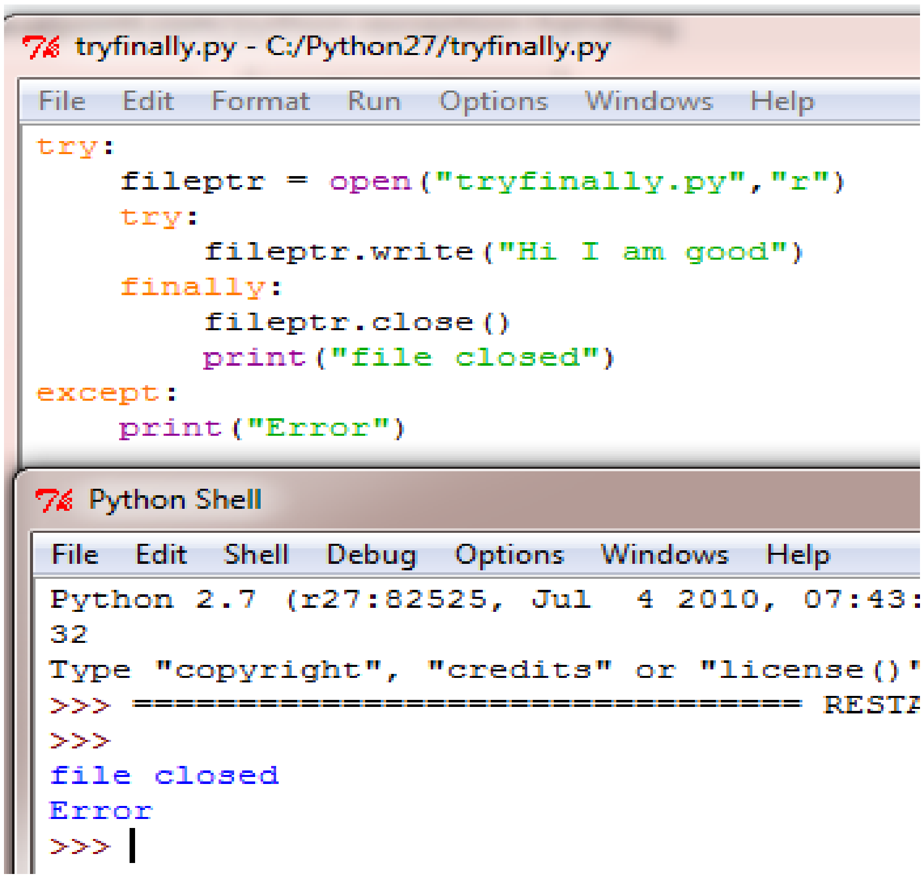
```
try:
    #this will throw an exception if the file doesn't exist.

    fileptr = open("file.txt","r")
except IOError:
    print("File not found")
else:
    print("The file opened successfully")
    fileptr.close()
```

```
The file opened successfully
>
```

The finally block

The finally block with the try block in which, we can pace the important code which must be executed before the try statement throws an exception.

(in given example we open file which is present in directory or filename where writing code.)


tryfinally.py - C:/Python27/tryfinally.py
File Edit Format Run Options Windows Help

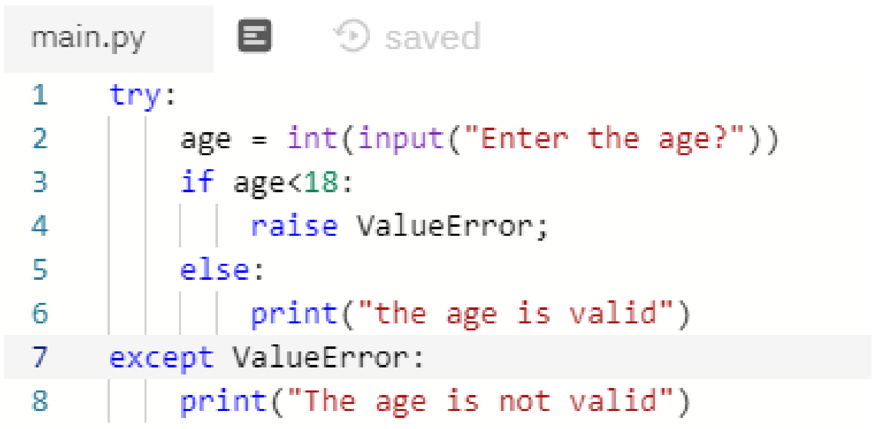
```
try:
    fileptr = open("tryfinally.py","r")
    try:
        fileptr.write("Hi I am good")
    finally:
        fileptr.close()
        print("file closed")
except:
    print("Error")
```

Python Shell
File Edit Shell Debug Options Windows Help

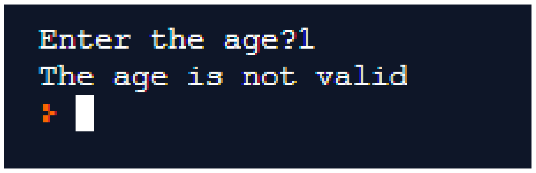
```
Python 2.7 (r27:82525, Jul  4 2010, 07:43:
32
Type "copyright", "credits" or "license()"
>>> ================================ RESTA
>>>
file closed
Error
>>> 
```

Raise an exception

To throw (or raise) an exception, use the raise keyword.



```
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")
```



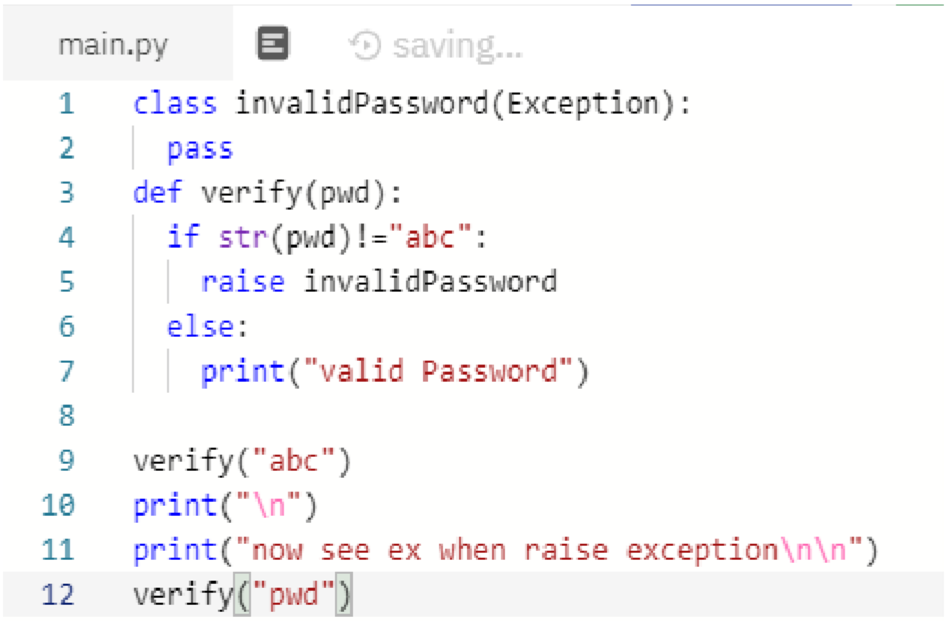
```
Enter the age?1
The age is not valid
```

User defined exceptions.

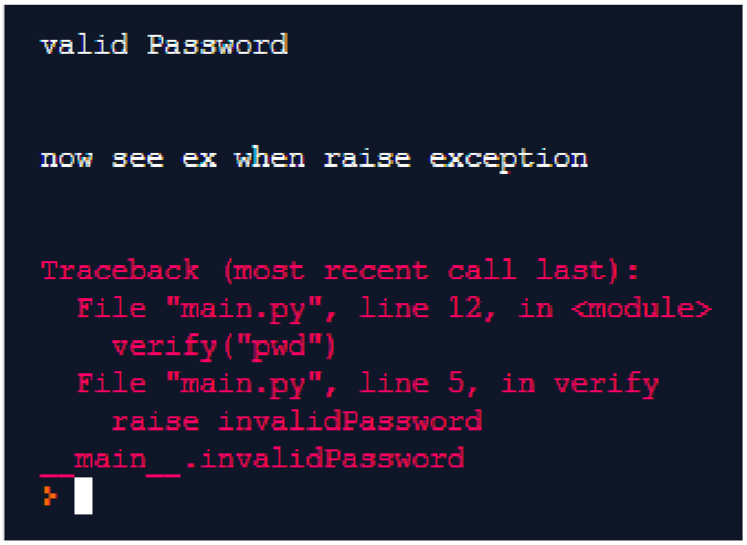
Python has many built-in exceptions which forces program to output an error when something in it goes wrong.

However, sometimes need to create a custom exception that serves purpose.

In Python, users can define such exceptions by creating a new class. This exception class has to be derived, either directly or indirectly, from Exception class.



```python
class invalidPassword(Exception):
  pass
def verify(pwd):
  if str(pwd)!="abc":
    raise invalidPassword
  else:
    print("valid Password")

verify("abc")
print("\n")
print("now see ex when raise exception\n\n")
verify("pwd")
```



```
valid Password


now see ex when raise exception


Traceback (most recent call last):
  File "main.py", line 12, in <module>
    verify("pwd")
  File "main.py", line 5, in verify
    raise invalidPassword
__main__.invalidPassword
›
```