File handling

When we normally want to open a .txt (text) file our operating system helps us.

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 along with the path and mode.

Open ("path of the file", mode)

- 'r' (read)
- 'a' (append)
- 'w' (write)
- 'x' (create)

There are four different modes for opening a file:

'r' - Read - Default value. opens a file for reading, error if 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 a specified file, error if file already exists.

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

't' - Text - Default value. Text mode

'b' - Binary - Binary mode (eg. images)

* When we open a file using python syntax it creates a file-object.

* we have few methods for the file-object.

read() method

read() method is used to read the file.
For this the file needs to be opened in the read mode first.

```
Open ("file-path", 'r')
                    ↳ read mode
```

↓ path_name it accepts

- double backward slash
- single forward slash

* path_name does not accept single backward slash so we view file in raw format.

```
Open (r "file-path", 'r')
```

Open function creates a file_object so

```
f = open (r "file-path", 'r')
```

so if we want to read the file_object

```
f.read()
```
→ this output is not exact

for exact output we use print function.

```
Print (f.read())
```

* we can only read once. After reading we have to close it.

close() method : After using any file object method once we need to close the file-object to clear the buffer.

```
f.close()
```

write() method : we need to open the file in append mode or write mode for us to be able to write to the file.

write() method in 'a' mode appends to the text already existing in the file.

```
f = open(r"path", "a")
f.write("....")
f.close()
```

write() method in 'w' mode clears the file and writes fresh in it.

```
f = open(r"path", "w")
f.write("....")
f.close()
```

* we need to close after writing.

Create mode "x"

If we want to create a new file we need to mention the file path and file name to be created in the open function.

```
open(r"file-path", 'x')
```

Q) Create file1 add some content to file1 and read file1.

```
f = open(r"filepath file1.txt", "x")

f = open(r"filepath file1.txt", "w")
```

```
f.write("This is the first file 9 created")
f.close()

f = open(r"filepath file1.txt", "x")

Print(f.read())
f.close()
```

→ "This is the first file 9 created".

we have more options like reading few characters like x.read(5) which only reads first five characters.

Also if we want to read linewise when the text file has multilines we can use a method called readline()

```
x.readline()
```

This method can be executed multiple times until we read all the lines. After last line Python will not return anything.

we have another advanced mode called "r+" which is read & write mode. Here when we are writing it will append to the existing file.

Here the read & write can be done one after another without closing the file

in between.

* Jupyter notebook has issues sometimes during file handling but pycharm does not have any issues.
* For now we can only handle text (.txt) files.
* In python we do not have an inbuilt function to delete the file.
* Always the file needs to be closed before performing the next operation.
* To delete a txt file in python we can take help of 'os' module.

import os

os.remove(file-path) → we can use this to delete the file.

more 'os' module methods:

os.getcwd() → gets current working directory

↳ C:\\Users\\.... gives file-path

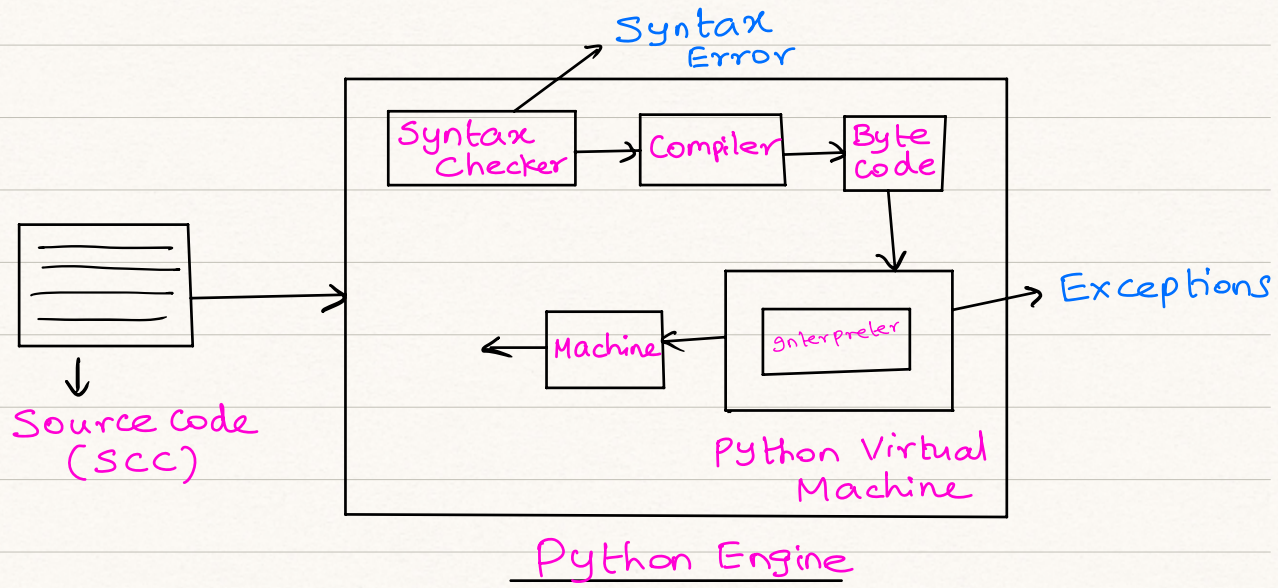
os.listdir() → in current file path list all the files.

os.chdir(new file-path) → changes the directory to the new file-path

os.rmdir(file-path) → removes the folder specified.

* This is ctrl+delete we cannot restore it.

Exception Handling:



Exceptions are errors raised when byte code is converted to the machine code by the interpreter in the python virtual machine during the execution of the python code.

* These are logical errors and we as programmers can handle them.
* Syntax errors cannot be handled and we simply have to correct the syntax.

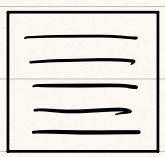
* when an exception occurs python normally stops and generates an error message.
* TO handle exceptions we use exception handling.

These exceptions can be handled using some blocks.

- The try block lets you test a block of code for errors.
- The except block lets you handle the error.
- The else block lets you execute the code when there is no error.
- The finally block lets you execute the code, irrespective of whether the error occurs or not.

try block: It tries to find out where the error happens.

Syntax

try:
    ....
    
    all the code } try block

ex:- try:
        Print(x) → x is not defined here (NameError)

try:
    1/0 → Number cannot be divided by zero. (ZeroDivisionError)

try:

`x = "asdf` → ✗ this is Syntax error & can only be corrected not handled.

```
try:
    x = [1, 2, 3, 4]
    x[5]        → Index out of range (IndexError)
```

There are many kinds of logical errors that we can handle using the Exception-class.

except block: This is a catcher of the exception thrown by the try block.

Syntax:

```
except Exception-class:
    print()        ← message
```

except block always follows the try block. except block will handle the errors and prints message after handling.

* we will have 'n' no.of Exception classes which will know what kind of error it is.
* try only finds the error & gives to except block and does nothing else.

ex:- 1/0 → ZeroDivisionError.

* except block handles the error and prints any message we want.

ex:- except ZeroDivisionError:
        print("you cannot divide a number by '0'")

* we can write 'n' no.of except blocks handling each kind of error.

ex:- try:
        x = "asdf"
        Yo
        x[20]
        print('hi')
    except IndexError:
        print("string index out of range")
    except ZeroDivisionError:
        print("you cannot divide a number by '0'")

* If we want to execute one statement or print one message for all the errors we can use it like:

ex:- try:
        x = "asdf"
        Yo
        x[20]
        print('hi')
    except (IndexError, ZeroDivisionError,..):

```
        print("you are having an error, handle
                                        it")
```

Else block: we can use 'else' keyword to define a block of code to be executed if no errors were raised.

* else block is optional.

Syntax

```
else:
    print("....")
```

ex:-

```
try:
    x = "asdf"
    Yo
    x[20]
    print('hi')
except (IndexError, ZeroDivisionError,..):
    print("you are having an error, handle
                                        it")
else:
    print("no error")
```

* except block should be present between try and else blocks.

finally block: The finally block, if specified will be executed regardless if the try block

raises an error or not.

ex:-
```
try:
    x = "asdf"
    Yo
    x[20]
    print('hi')
except (IndexError, ZeroDivisionError,..):
    print("you are having an error, handle
                                    it")
else:
    print("no error")
finally:
    print ("The try except is finished")
```

* This can be useful to close objects and cleanup resources.

ex:- try to open and write to a file that is not writable

```
try:
    f.open ("demofile.txt", "r")
    try:
        f.write ("Lorum Ipsum")
    except:
        print("something went wrong while
                                writing to file")
```

```
finally:
    f.close()
except:
    print("something went wrong when
           opening the file")
```

* Exception handling is usually used in app development.
* Datascience does not need deep exception handling.

To Raise our own concern : (Not used much)

As a python programmer we can choose to throw an exception if a condition occurs.
To throw (or raise) an exception, use the 'raise' keyword.

Syntax:

```
raise Exception-class ("your message")
```

ex :-

```
x = "hello"
if not type(x) is int:
    raise TypeError("only integers are allowed")
```

* we can define what kind of error to raise, and the text to print to user.

** 'raise' is not commonly used because it makes the program slow by making the python do multiple things.

* Instead better to use 'None' keyword for this exception.