5th-July

{ String is stored inside memory (RAM)
& file " " " hard-disk

Python-tutorial 8

- Writing to a file:

```
my_file = open('test21.txt', mode='w')
my_file.write('This is a new line') → 18
```

- Iterating through a file:

```
%%writefile test51.txt
First line MADHU
Second line NIDHI
myfile = open('test51.txt')
myfile.read()
→ 'First line MADHU \n Second line NIDHI \n'
```

(∵ we are getting output like this, not exactly what we wrote). So, ↓

```
for line in open('test51.txt'): # myfile:
    print(line)
```

First line MADHU
Second line NIDHI

- String IO: (String Input Output)

```
message = 'This is string'   (∵ random string)
type(message) → str
f = StringIO(message)   (∵ Pass message to StringIO)
type(f) → _io.StringIO
```

Converting string to StringIO benefits with us we can use file like operations, like read, write etc.

```
f.read() → 'This is string'
f.read() → ''
f.seek(4)
f.read() → '_is string'
f.write('This is second line.') → 40
f.seek(4)
f.read() → '_is string This is second line.'
```

- **Errors and Exception Handling:** (imp)

If you make error in any line, further code won't be executed/run.
So, we always want that if our code fails somewhere in between, it allow further code to get executed & this can be managed by 'exception handling'.

error → compile-time error
→ run-time error

Note: (always use except block with try block otherwise there will be syntatical error. If code inside try block is correct then compilor won't go to except block.)
[try, except, else, finally → blocks]

```
try:
    f = open('text_21.txt')
    print(f.read())
except: print("WRONG")
```

Compile-time error → syntax
↳ (won't allow to run without compilation).
run-time error →
↳ code is correct but the kind of data provided is not.

↳ This is a new line.
(∵ Since, code inside try block was right).

(*)

```
try:
    f = open('text21.txt')
    print(f.write("MADHU"))
except:
    print("WRONG")
print:("HI")
```

→ WRONG
HI

(∵ so, even if the code inside try is wrong, further code execution is done).

(*)

```
try:
    f = open('text21.txt)
    print(f.write("MADHU"))
except:
    print("WRONG")
try:
    b = int(input("Enter integer"))
    c = 5/b
    print(c)
except:
    print("SORRY")
```

→ WRONG
Enter integer [2]
2.5

WRONG
Enter integer [0]
SORRY

Note: indentation is very important in Python.

- nested try-except:

```
try:
    WRONG → (if wrong, then nested try-except won't be executed)
    try:
    except:
except:
    -
```

```
try:
    WRONG
except:
    -
    try:
    except:
```
(This time nested try-except will get executed.)

- ~~nest~~ try-except-else: (else is optional)
(else will be executed only if try block is executed successfully.)

```
try:
    b = int(input("Enter value"))
    c = 6/b
    print(c)
except:
    print("SORRY")
else:
    print("execute only if try is right")
```

Enter value [0]
SORRY

Enter value [2.]
3
execute only if try is right

- finally: (execute itself for sure, doesn't depends on whether try is right/wrong.)
(optional block)

(*) enter a & b integers, keep on asking for integers until we pass proper input :-

```
while True:                    (until true).
    try:
        a = int(input('FIRST'))
        b = int(input('SECOND'))
        div = a/b
        print(div)
        break                  (break entire loop)
    except:
        print('INVALID INPUT')
        print('PRINT NUMBERS AGAIN')
        continue
```

(*)

```
try:
    5/0
except Exception as e:
    print("some issue:", e)
```

(many kinds of exceptions) (super class)

our message, system message

some issue: division by zero