First Class After Mid Sem

Function python

syntax:-

```
def funs(arg. ):
    """docstring. - """
    S1;
    S2;
    S3;
    return [expression]
```

//pointing the docstring:-

```
print(funs_doc_)
```

scope and lifetime

Second Class After Mid Sem

Python:-

3 types of function argument:-

1) ~~Multiple Argument~~

```
def func1(name, message):
    print('Hi', name, message)
func1('Ajay', 'welcome')
```

Hi Ajay welcome.

2) default argument:- value assigned in the function it declaration itself.

```
def func1(name, message='welcome'):
    print('Hi', name, message)
func1('Ajay').
```

Hi Ajay welcome.

Page No.

def func1(name='Ajay', message): default arg. should be on right.
print('Hi', name, message).

func1(('This') → error

Date ___/___/_____

2) Keyword Argument :-

```
def fun1(name, message):
    print('Hi', name, message)

fun1(message='welcome', name='Ajay')

fun1(message='welcome', 'Ajay') → Error
```

keyword argument always come after positional argument.

~~fun1('Hi',~~

```
def fun1(good, name, message):
    print(good, name, message)

fun1('Hi', message='Hello', 'Ajay')
```

3) Arbitrary argument

```
def fun1(*names):
    for name in names:
        print(name)
fun1('Ajay', 'Rohit', 'Anjali')
```

Ajay
Rohit
Anjali

def ~~fun~~(a=10, b=15, c=20)
pass

~~fun1~~

Recursive Function:-

Date __/__/____

```
def fun1():
    S1;
    S2
    fun1()

fun1()
```

```
def factorial(n):
    if n==0 or n==1
        return 1
    else:
        return n*factorial(n-1)

factorial(5)
```

Lambda function:

```
lambda arg: expr

lambda n1,n2,n3: n1*n2*n3

lambda x: x*3
print(x(5))
```

filter (function, iterable) → Based on condition

```
l=[1,2,3,4,5,6,7,8,9]

list(filter(lambda x: x%3==0, l))
```

Page No.

Date ___/___/_____

map (function, iterable) → Based on operation.

l = [1, 2, 3, 4, 5, 6, 7, 1, 9]

print(map(lambda x: x**3, l))

Third Class After Mid Sem

Staffing:- Non local variable :- neither local nor global.

'Nonlocal' keyword is used to make a variable nonlocal.

```
def fun1():
    x=1
    def fun2():
        Nonlocal x=2
            print(x)
    print(x)
    fun2()
    print(x)
fun1()
print(x)
```

1.
2
2
error not define.

```
def fun1():
    x=1
    def fun2():
        Nonlocal x
            x=2
            print(x)
    print(x)
    fun2()
    print(x)
fun1()
print(x)
```

1
2
2
error.

jisme nonlocal uske bahar me change. just

Date ___ / ___ / ___

```
def fun1():
    x=1
    def fun2():
        x=2
        def fun3():
            Non local x
            x=3
            print(x)
        print(x)
        fun3()
    print(x)
    fun2()
    print(x)
fun1()
print(x)
```

	w nonlocal	w/o nonlocal
	1	1
fun1()	2	2
print(x)	3	3
	3	1
	error	error

```
def fun1():
    x=1
    def fun2():
        global x
        x=2
        print(x)
    print(x)
    fun2()
    print(x)
fun1()
print(x)
```

	1
	2
	1
	2

module:- collection of function which can be accessed from other file

N/AN

Date ___/___/______

$(A \cap B)' = A' \cup B'$

accessing a module:-

① import the module :- import math

```
print(math.pi)
```

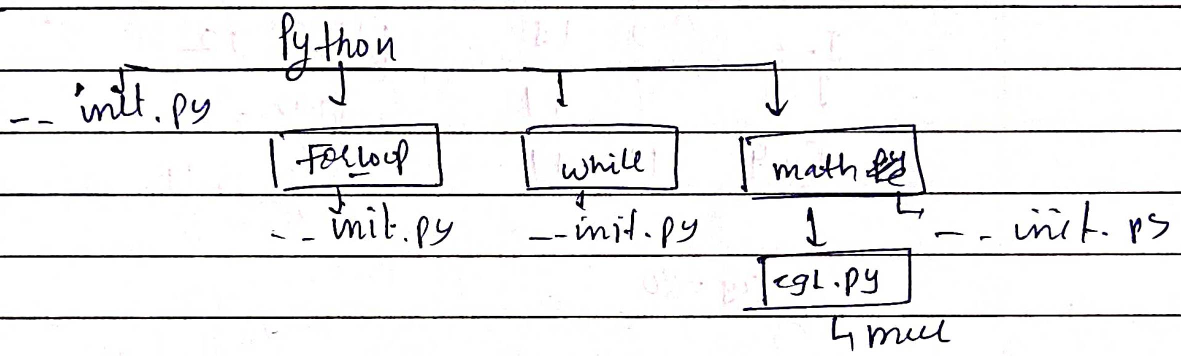
② import with rename:- import math as mt

```
print(mt.pi)
```

③ from math import pi, e, sin()

④ from math import

package:-




```
import python.math.eg1 as eg.
eg.mul(1,2)
```

Fourth Class after Mid sem is attached later

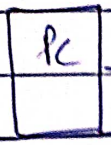
RISC Pipelining :- Python :-

Fifth Class After Mid Sem

Date __/__/__

Unit :- 4 File and Exception Handeling

File and directory :-



eg.py txt

I am in Ranchi

open function :-

v = open("eg.txt") → default in read mode.

v :- file object.

v = open("eg.txt", 'r')

↳ compute path to be specified.

↳ mode

print(v.read()) → print the whole content of txt file.

if → v.read(2) → read first two characters.

print(v.read(2))

print(v.read(2))

output :- I am

print(v.readlines()) → returns a list.

['Iam in ', 'Ranchi']

write mode / writing in a file :-

v = open("eg.txt", "w") ; → if file it doesn't exist then

v.write("IIIT Ranchi"); creates a file.

↳ The original .txt file will get overwritten in file.

append mode

v = open('eg.txt', "a")

v.write ("IIIT Ranchi")

output :- I am in

Ranchi IIIT Ranchi

Date ___/___/_____

exclusive location:-

x = open("eg.txt","x") → it also tells whether file already exist or not.

closing:-

v.close()

Position of a cursor / cursor location:-

v.tell() → tells position of cursor.

changing the position

v.seek(0)

Directory:- used with os module

To check current directory.

import os

print(os.getcwd())

→ returns current directory. (full path name)

changing the directory:-

os.chdir("compute path")

To check subdirectory

print(os.listdir())

↳ if no argument → then it returns the content of current directory

so for a directory → specify its path

Creating a directory

os.mkdir("C:\\Python/name")

↳ path ↳ name of directory

Page No. ____

Date __/__/____

Renaming a directory

Python → Java → directory name

```
os.rename("old name of directory", "new name of directory")
```

Removing a ~~directory~~ file

```
os.remove("name of file");
```

For directory

```
os.rmdir("directory-name").
```

It will only delete empty directory.

```
import shutil.
shutil.rmtree("directory name")
```

removes whole directory without whether it is empty or not.

Sixth Class After Mid Sem

Exception Handling:-

Try and except statement:-

```
l=[2, 2.5, 0, '2']
for i in l:
    try:
        print(i)
        x=1/i
        print(x)
    except:
        print("Error:")
```

output:-

```
2
0.5
2.5
0.4
0
Error: 0
2
Error: 2
```

To handle certain types of error:-

Date ___/___/_____

② for n in l:

```
try:
    print(n)
    x = 1/n
    print(x)
except TypeError:
    print("Error:", n)
except ZeroDivisionError:
    print("Error2:", n)
except:
    print("any other error")
else:
    print("No error")
```

2	2
0.5	No error
2.5	0.5
0.4	No error → ∵ no error in try block
0	2.5
error2: 0	0.4
z	No error
error: z	0
	error2: 0
	z
	error: z

④ for n in l:

```
try:
    print(n)
    x = 1/n
    print(x)
except:
    print("Error:", n)
finally:
    print("Executed finally")
```

2
0.5
Executed Finally
2.5
0.4
Executed Finally
0
error: 2
Executed Finally
z
error: z
Executed Finally

index, import,

Date ___ / ___ / ___

User-defined exception:- (raise keyword)

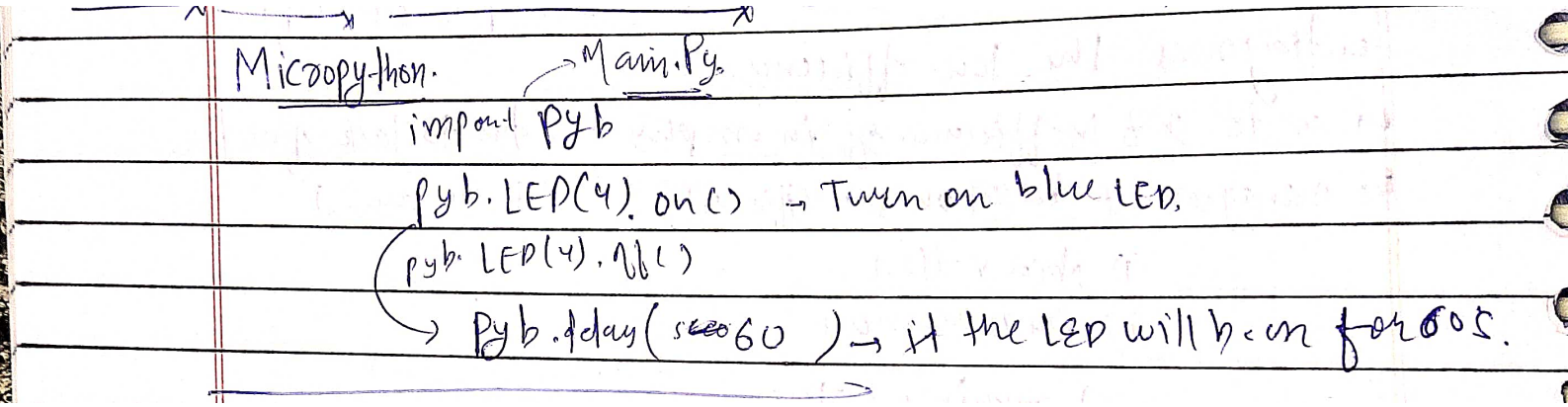
```
l = [2, 2.5]
for n in l:
    try:
        print(n)
        a = 1/n
        print(a)
        raise
    except:
        print("Error")
except ZeroDivisionError:
    print("Error zero")
```

	2
	0.5
	Error
	2.5
	0.4
	Error

```
class E(Exception):
    pass
for n in l:
    try:
        print(n)
        a = 1/n
        print(a)
        raise E
    except E:
        print("User exception")
```

	l = [2, 2.5, 0]
	2
	0.5
	User exception
	2.5
	0.4
	User exception
	0
	Error Zero

Seventh Class After Mid Sem



Fourth Class After Mid Sem

4th class

Date 20/10/22
Page

Date and Time Module

```
from datetime import date
                     year month day
a = date(2021, 11, 13)
print(a)
```

2021/11/13.

```
a = date.today( )
print(a.year)
```

2022

```
a = date.fromtimestamp(1)
```

Default value : 1 Jan 1970

```
a = date( )
print(a)
```

— Error

Time class

```
from datetime import time

a = time( )
print(a)
```

00:00:00

```
a = time(11)
print(a)
```

11:00:00

```
a = time(11, 12)
print(a)
```

11:12:00

```
a = time(11, 12, 13)
print(a)
```

11:12:13

→ microsecond

```
a = time(11,12,13,1)
print(a)
```

11:12:13.1

```
print(a.hour)
```

date and time class

```
from datetime import datetime
a = datetime(2011,12,13)
```

year, month, day

First three arguments are necessary

```
print(a)
```

2011/12/13 00:00:00

```
a = datetime.now()
print(a)
```

2022/10/20 03:30:00

→ timedelta class

```
a = date(2014,12,13)
b = date(2019,12,13)
c = b-a
print(c)
```

// days

365 days

a = timedelta(days=365)
b = timedelta(days=12)
c = b - a
print(c)

-353

strftime method

from datetime import date

a = date(2018, 11, 13)
c = a.strftime("%d-%m-%Y %H:%M:%S")
~~%H:%M:%S~~

print(c)

13-11-2018 00:00:00

c = a.strftime("%d %b, %Y")

13 Nov, 2018

%B - November

Two arguments are necessary

```
a = 20 oct, 2022
c = a.strptime(a, "%d %b, %Y")
print(c)
```

2022/10/20

```
a = oct 20, 2022
"%b %d, %y"
```

Unpacking of Arguments

Before pos slash you should have positional arguments.

After star you can have keyword arguments only.

```
def fun1(positional /, * keyword)
def fun1(*names)
    for name in names
        print('Hi', name)
n = ('Rohit', 'Ajay')
fun1(*n)
```

→ unpacking of a argument from a list tuple, string

```
def fun1 ( msg, msg1, name)
    print ( msg, name, msg1)

s = { msg : 'Hi', name: 'Rohit', msg1 : 'welcome'}

fun1 ( ** s )  > two stars required
```