ಶ್ರೀ ಗುರುಪ್ರಸಾದ್ ಎಸ್
ಶ್ರೀ ನಿಲಯ ಬಿಂಬಿಸಿರಿಸಿ

INDEX

NAME: GURUPRASAD.S STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: _____

S. No.	Date	Title	Page No.	Teacher's Sign / Remarks
		PYTHON APPLICATION		
		PROGRAMMING		
		Sub Code: 15CS664		
	1-13, 15	PYTHON FOR EVERY BODY - Charles R. Severance		
	15, 16, 17	Allen B Downey - Thinx Python.		

#> import this
zen of Python

Python

It is a general purpose, interpreted, dynamic, object oriented Programming language, some features are:

- → elegant Syntax & code readability
- → less lines of code → easy maintainance
- → large set of standard libraries
- → interactive mode - easy to test
- → Easily Extended by adding Pre compiled modules of C/C++
- → Supports OOP, code can be grouped into modules & Packages etc..

Installing Python

Windows

- → Download & install from www.python.org
- → Run in Command Prompt by > py or by using IDLE
- → Create Program in file and save it as .py and Run module F5

Linux

```
# wget http://www.python.org/ftp/Python/3.6.3/Python-3.6.3.tgz
# tar xvf Python-3.6.3.tgz
# cd Python-3.6.3
# .configure --enable-optimizations
# make -j8
# make altinstall
# Python3.6   - runs python in cmd Prompt
```

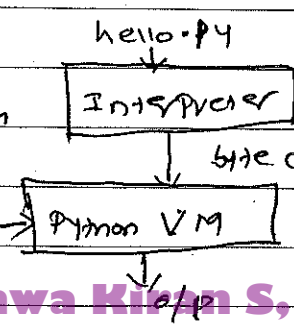
- → Create Pg file and save as .py

```
# Python hello.py ↵
```

Internal working of Python

Interpreter checks the syntax of the Program

Python Virtual M/c will generate the target code.



The key differences b/w Python 2.x and 3.x

→ Python was developed by Guido Van Rossum in 1980s and first released in 1991

→ He combined the features of C and ABC Languages it was named after Comedy TV series Monty Python's Flying Circus

→ Python 2.0 was released in 2000 by adding features such as garbage collection, support for unicode

→ Python 3.0 was released in 2008

Print function

2.x	– Print	Print 'Hello, world'
3.x	– Print()	Print ('Hello, world')

Integer Division

2.x	3.x
3/2 = 1	3/2 = 1.5
3/2.0 = 1.5	3/2.0 = 1.5

Unicode

2.x has ASCII str() types separate unicode() but no byte type

3.x has Unicode (utf-8) strings 2 byte classes byte and bytearray

Xrange

2.x Xrange() is used to create iterable objects to iterate infinately

3.x range() is implemented like xrange() and xrange() does not exist

range() got new __contains__ method that speed up

Raising Exception

2.x raise IOError, "file Error"

3.x raise IOError("file error")

Handling exceptions.

2.x

```
try
    let_us_cause_a_NameError
    except NameError, err:
    Print err, ' -> our Error message'
```

3.x

```
try
    let_us_cause_a_NameError
    Except NameError as err:
    Print (err, ' -> our Error message')
```

next and .next() methods

2.x supports both next() and .next() methods

3.x supports only .next() method

round()

2.x rounds to the nearest no Eg 16.5 to 17

3.x rounds to nearest Even no Eg 16.5 to 16

Comments in Python

- Single line Comment

single quotes - ''' till ''' multiline Comment

; is used to write multiple statements per line

Print ("Hello") ; Print ("world")

\ is used to write single statement in multiple line

Print (\
"hello")

KWD
ID
Literals
Spl symbols
operators

triple double quotes — """ allows string to span multiple lines

Eg: Print ("""Hello
world""")

Python Character set

a-z A-Z
0-9
spl symbols, white spaces \t \n \x0b \x0c \r

Literals 072 - octal 0XAB - hexidecimal
78 - int literal 21.98 - float literal 'Q' char "Hello" - string
type() - function used to know type of value/variable
type ('Hello') - str
type (123) - int

① **Keywords** import keyword Print (keyword.kwlist)
Python has 33 keywords
and | as | assert | break | class | continue | def | del | elif | except
False | finally | for | from | global | if | import | in | is | lambda | None
nonlocal | not | or | Pass | raise | return | True | try | while | with | yield

② **Identifiers/variables**
user defined names, start with character followed by char/num
/underscore, any length, cannot start with _

Data types

Integers
Number +/- ve without decimal point
octal repn by prefixing 0 O
Hexa decimal repn by prefixing 0x or 0X
Eg: 25 072
-73 0xAB

int() function/constructor is used to declare, initilize and convert other data type to int

Eg: int (x=0) int() → 0 int("25", 8) → 21
int (23.99) int (25) → 25
int ("25") int ("25") → 25

Float number

Contains number with decimal point. Int can be repn as decimal / scientific notations

Eg: 25.3
3.7 e 1. = 37.0

float() constructor is used to declare, initilize and convert to float

float (y = 3.2) float() → 0.0 float("23") → 23.0
float ('10.2') float (23) → 23.0

Complex number

Complex number is of the form a+bJ or x+yJ where a & b or x & y are real part and J is imaginary part.

(2+3J) → (2+3J) Complex() → 0J
2+3J → 2+3J Complex (2) → (2+0J)
(2+3J).real → 2.0 Complex ("2+3J") → (2+3J)
(2+3J).img → 3.0 complex ("2") → (2+0J)
Complex(2,3)

Complex() used to declare/initilize/convert complex nos
.real ~~img~~ field tells the real part and imaginary part
.imag

operator + is used for concatination of string

X = "abc"
Y = "xyz"

X+Y abcxyz

The str type

String type (str) allows us to store non-numric values in addition to numric values

String is created by using single/double/triple quotes

Str() constructor can be used to create/convert to str

X = "abc"

X = Str("abc")	'''Hello "world" from 'India''''
D = 'Hello'	Str(12.5) Str()
D = "Hello"	'12.5' ''

If string has single quotes enclose it in double quotes
If string has double quotes enclose it in single quotes
If string has both single & double quotes enclose it in ''' quotes

Boolean type

It represents boolean value True or False. True represented by 1 and False by 0

bool() is used for conversion of int/string to boolean

Eg:

bool(0)	bool(10)	bool(0.0)	bool(-12.5)	bool()	bool("")
False	True	False	False	False	

X = True | X = bool(True) | 5==4 False | 6>4 True

NOTE

→ In Python variables are not declared they are created by allignment
→ The type of a variable may change during the program
→ The operations on variables depend on the type of variable
→ int | float | str | bool | complex are the data types
→

Operators

+ - * / // % ** — arithmatic

== != <> <= >= — relational.

and, not, or — Logical

& | ~ ^ << >> — Bitwise

NOTE: Precedance of arithmatic operator evaluation is PMDAS and associativity is from L → R

Operations on int.

2 ** 3 — 8	10//3 — 3 (int division)	2-3 → -1
2 *3 — 6	10%3 — 1	
10/3 — 3.333	2+3 — 5	

functions

pow(2,3) — 8	math.factorial(5) → 120		
divmod(10,3) — (3,1)	2&3 → 2	2>>3	0
abs(-10) → 10	2|3 → 3	2<3	2==3
bin(12) → 0b1100	2^3 → 1	2<=3	2!=3
oct(12) → 0o14	~2 → -3	2>3	
hex(12) → 0xc	2<<3 → 16	2>=3	

operations on float

1.2 ** 0.1	math.floor(2.3)
2.3 *3	math.ceil(2.3)
7.5/3 — 2.5	round(2.3) → 2.0
7.5//3 — 2.0	math.trunc(2.3)
4.5%1.2 — 0.900	math.exp(2.3)
2.1+3.3	math.radians(180)
2.1-3.3	math.sqrt(2.3)
pow(1.2, 0.1)	math.sin(0)
divmod(7.5, 3) - (2.0, 1.5)	math.cos(0)
abs(-1.2	math.tan(0)

Operations on Complex

1j **2

(2+3j) * (5+6j)	cmath.polar(2+3j) - rectangular to polar coordinates
(2+3j) / (8+3j)	
~~(20+30j) // (2+3j)~~	cmath.rect(3.600...0.9...) - reverse -
~~(5+8j) % (2+3j)~~	abs(3+4j)
(2+3j) + (5+6j)	cmath.phase(3+4j) Angle (θ)
(2+3j) - (5+6j)	

String operations

'aBc'.lower()	'abc'.islower()
'aBc'.upper()	'abc'.isnumeric()
'hello WORLD'.title()	'abc'.isalnum()
'hello WORLD'.capitalize()	'abc'.isidentifier()
'hello WORLD'.swapcase()	'abc'.istitle()
'abc'.isalpha()	'abc'.isprintable()
'abc'.isupper()	'abc'.isspace()

Operations on bool type

bool(0) - F	bool() - F
bool(10) - T	and / or / not operations are short circuit ops
bool(0.0) - F	T and F - F
bool(-12.5) - T	F and T - F
bool("hello") - T	T and T - T
bool("") - F	

T or F - T	2 or 5 - 2	not F - T
F or T - T	5 or 2 - 5	not T - F
T or T - T	0 or 5 - 5	not 2 - F
F or F - F		not 0 - T

bool is sub type of int hence all operators work on int works on bool

Variables

1. Variables can be created when and where required
2. The value of variable can change at any time
3. The type of variable can change at any time.
4. We can find the data type of values allocated with variable
5. The operations on a variable depend on the type of data it is currently holding and can change at any time
6. Values are objects. Variables are reference to these objects

Basic input and output

Print()

The Print() displays the contents on screen.

Syntax: Print(argument)

argument can be value of type: int, float, str, bool

Eg: Print("hello") print(12) Print(True) Print(1.5*3)

Print ("hello", "world") Print ("2+3 = ", 2+3)

Print ("Hello", "world", sep = ',')

Print("hello", "world", sep = "<->")

Print ("Python, "is", "a", "dynamic", "language" \n \t \t \n

Print ("Welcome", "to", "Python", sep= "...", end="\n...\n

format()

format (item, format-specifier)

item - no or string format-specifier - string specifier how to format

format (x, ".2f") format(x, "10.2f")

Print (format (10.234567, "10.2f") Right Justified

Print (format (—— " ——, "<10.2f") left

Print (format (20, "10x")) Right Hexadecimal

— " —— " —, "<10x")) left hexa

d - decimal

o - octal

Print ("format ("Hello world", "25s") left "<25s" right

Print (format (0.31456, "10.2%"))

Print (format (31.2345, "10.2e"))

input ()

1 input() is used to accept i/p from user.

Var_name = input()

Var_name = input ('string') - Prompt the string and wait until user enter data

Eg: name = input ("Enter your Name")

language = input ("Enter language")

Print ('{} is learning {}'.format (name, language))

Eg: accept a number print the type of number.

num1 = input ("Enter a num")

Print ("the type of num is")

Print (type (num1))

num1 = int (num1)

Print (the type of num is")

Print (type (num1))

Area of Rectangle

len = int (input ("Enter len"))

bre = int (input ("Enter breadth"))

Print ("The area of rectangle")

Print (len * bre)

multiple i/p

a, b, c = input ("Please enter 3 nos\n").split (',')

Print ("the nos entered\n" + a + "\n" + b + "\n" + c + "\n")

Assignment

Variable = Expression

2 = 1	E = (5+10+25+30)	P = q = R = 100
2 (i/p) 1	E (o/p) 150	

Swap number

```
p, q = input("Enter two nos\n").split(',')
t = p
p = q
q = t
print (p, q)
```

or `p, q = q, p`

eval()

accepts string as parameter and returns python expression

`Eval('Print ("Hello")')` returns Hello

Eval() can also be used to return int value of the string passed

`x = int(input ("Enter the num"))` we have as

`x = Eval(input ("Enter the num"))`

Eg:
```
Name = input ("Enter name")
Age = Eval(input ("Enter age"))
Gender = input ("Enter gender")
Hight = Eval(input ("Enter hight"))
Print ("We Detaile\n")
Print ("Name", Name)
Print ("Age", Age)
Print ("Gender", Gender)
Print ("Hight", Hight)
```

Eg:
```
import math                    # hypotenuse of Rt angled triangle
b, h = eval (input ("Enter base and hight of triangle"))
area = math.sqrt(b*b + h*h)
Print ("Area is ", area)
```

→ ord('a') - give ASCII chr(97) - give char of ASCII

max(12, -12, 1) ⇒ 12 max('a', 'B', 'c') ⇒ c

Grams to kg

```
w1 = eval(input("Enter weight in grams"))
w2 = w1 // 1000        # gets no of kgs
w3 = w1 % 1000         # gets grams
Print ("Weight = ", w2, " kg and ", w3, " g")
```

Reverse of no

```
Num = eval (input ("Enter 4 digit number"))
r1 = num % 10
q1 = num // 10
r2 = q1 % 10
q2 = q1 // 10
r3 = q2 % 10
q3 = q2 // 10
r4 = q3 % 10
Print ('Reverse of', num, ' is:', r1, r2, r3, r4)
```

Bit wise operators

1 & 3 ⇒ 1 5 & 7 ⇒ 5

```
0001    0101
0011    0111
----    ----
0001    0101
```

1 | 3 ⇒ 3 5 | 7 ⇒ 7

```
0001    0101
0011    0111
----    ----
0011    0111
```

3 ^ 5 => 6	4 >> 2 = 1	4 << 2 => 16
0011	00000100	0000 0100
0101	→ 00000001	0001 0000
0110	N>>S = N/2^S	N<<2 = N*2^S

Compound assignment

integer floor division

+= | -= | *= | /= | //= | **= | %=

Evaluate $z = \dfrac{\frac{104(ab+c)}{d} - 0.8 + 2b}{(x+a)\left(\frac{1}{2}\right)}$

1 USD = 2.20 pound convert given i/p

```
p,q,r = Eval(Input('Enter 3 nos:'))
Print('p=', p, 'q=', q, 'r=', r,'.)
Print('(p>q>r) is', p>q>r)
Print('(p<q<r) is', p<q<r)
Print('(p<q) and (q<r) is'; (p<q) and (q<r))
```

Decision making Statements

Python supports following decision making statement

1. if statement
2. if-else
3. nested if
4. Multi way - if - elif - else

if statements

indentation is important

```
if condition:
    Blocks
```

```
Num = Eval(Input("Enter a num"))
if (num > 0):
    Num = Num*Num
    Print (num)
```

if - else statement

```
if condition:
    if_Block
else:
    else_Block
```

```
basic = float(input("Enter basic salary"))
if basic >= 10000:
    hra = 20/100 * basic
    da = 85/100 * basic
    cca = 10/100 * basic
else:
    hra = 15/100 * basic
    da = 75/100 * basic
    cca = 8/100 * basic
gross = basic + da + hra + cca
print("gross salary is", gross)
```

Nested - if

```
if condn1:
    stmt1
else:
if condn2:
    stmt2
else:
    stmt3
else:
    stmt4
```

```
n1, n2, n3 = Eval(input("Enter 3 nos"))
if n1 > n2:
    if n2 > n3:
        print('n1 greater than all')
    else:
        print('n2 greater than n3')
else:
    print('n1 is small')
```

```
Y = int(input('Enter year'))
if (Y % 100 == 0):
    if (Y % 400 == 0):
        print('', Y, 'is a century leap year')
    else:
        print('', Y, 'is a century non leap year')
else:
    if (Y % 4 == 0):
        print('', Y, 'is a non century leap year')
    else:
        print('', Y, 'is a non century non leap year')

print('{} is a century leap year'.format(Y))
```

to put a int/float value/var in a string

mind the indentation

other way

if - elif - else

```
if condn 1:
    block 1
elif condn 2:
    block 2
elif condn 3:
    block 3
.....
else:
    block n
```

```
m = float(input('enter marks'))
if (m <= 100 and m >= 70):
    print('Distinction')
elif (m >= 60 and m <= 69):
    print('First class')
elif (m >= 50 and m <= 59):
    print('second class')
elif (m >= 35 and m <= 49):
    print('Pass class')
elif (m > 0 and m <= 34):
    print('Fail')
else:
    print('Invalid marks')
```

Ex 6 Implement Calculator

Conditional Expression (ternary operator)

Syntax:

w/o parantheses

Expression 1 if condition else Expression 2

Eg: num1, num2 = Eval (input ('Enter two numbers'))
print ('min =', num1) if num1 < num2 else print ('min =', num2)

Eg: n = Eval (input ('Enter a number'))
print ('even') if n%2 == 0 else print ('odd')

Day of the month Prg.

```
m = eval(input ('Enter the month no b/w 1 to 12:'))
flag = 1
if (m == 2):
    ~~print ('Enter the year')~~
    ~~y = input ()~~
    y = Eval (input ('Enter the year'))
    if ((y%100 == 0 and y%400 == 0) or (y%100 != 0 and y%4 == 0)):
        days = 29
    else:
        days = 28
elif m in (1, 3, 5, 7, 8, 10, 12):
    days = 31
elif m in (4, 6, 9, 11):
    days = 30
else:
    flag = 0
if flag == 1:
    print ('no of days in {} month is {}'. format (m, days))
else:
    print ('Invalid month')
```

Loop Control Statements

While Loop

```
while condition:
    statements
else:
    statement
```

```
while condn : statement
```

Note: Python don't have ++ or -- operators.

```
n = eval(input('Enter a num'))
i = 1
while i<=n:
    print(i)
    i = i+1
```

Palindrome

```
n = int(input('enter a number'))
r=0
x=n
while n>0:
    d = n%10
    n = n//10
    r = r*10+d
print('The reverse of ',x,' is ',r,')
if x==r:
    print('The number is palindrome')
```

use of break and else in while to check given no is prime or not

```
n = eval(input('enter the number'))
i=2
while i<=n/2:
    if n%i==0:
        isPrime = False
        break
    i = i+1
else:
    isPrime = True
if isPrime:
    print('number is prime')
else:
    print('number is composite')
```

range() function

In-built function used to generate list of integers

it can have one/two/three parameters

range(begin, end, step)

Eg: list(range(1, 6))

[1, 2, 3, 4, 5]

Example: list(range(1, 20, 2)) - Print 1-20 with difference of 2

range(5) | range(1, 5) | range(1, 10, 2) | range(5, 0, -1) | range(-4, 4) | range(1, 1) | range(0)

For Loop

Syntax:

```
for var in sequence:
    statements
```

```
for i in range(1, 6):
    Print(i)
```

A - Z

```
for i in range(65, 91, 1):
    Print(Chr(i))
```

10 - 1

```
for i in range(10, 0, -1):
    Print(i)
```

Sum nos not div by 2/3/5

```
Sum = 0
n = Eval(Input("Enter range"))
for i in (1, n, 1):
    if n%2 == 0 or n%3 == 0 or n%5 == 0:

    else:
        Print(i)
        sum = sum + i
Print("the sum is", sum)
```

```
n = int(input("Enter the value"))
if (n==1):
    Print(0)
elif (n==2):
    Print(0,1)
elif (n>2):
    n1 = 0
    n2 = 1
    Print(n1, n2)
for i in range(n+1):
    fib = n1+n2
    Print(fib)
    n1 = n2
    n2 = fib
else:
    Print("Invalid")
```

Nested for loop

```
*
* *
* * *
```

```
n = eval(input('enter the val of n'))
n1 = 0
for i in range(1, n+1, 1):
    n1 = i
    for j in range(1, i+1, 1):
        print("*", end=" ")
    print("\n")
```

for with else clause

else clause of for loop is entered either if the loop has exhausted or terminated due to break

```
for var in sequence:
    statements
else:
    statements
```

Generate prime nos

```
n = eval(input('enter the limit'))
for i in range(2, n+1):
    for j in range(2, i):
        if i % j == 0:
            break
        else:
            print(i)
```

Continue

```
for i in range(1, 4):
    for j in range(1, 4):
        for k in range(1, 4):
            if i==j or j==k or k==i:
                continue
            else:
                print (i,j,k)
```

Function

A function is a self contained block of one or more statements

Syntax:

```
def function_name (Parameters):
        statement
        statement
```

```
def hi()
    print ("Hello world")
    hi()
```

Re definition of function is possible

```
def hi()
    print ("bmsit")
```

function name w/o parantheses is reference to the function can be stored as variable and used later.

```
def hi()
    print ("hello")
g = hi
g()
```

o/p hello

Parameters to function Positional Parameters.

The function can accept one or more arguments

```
def fun_name (P1, P2, ...):
    statements
```

The parameters in the function are assigned according to their position

```
def fun2 (name, age):
    Print ('the name is ', name, ' and age is ', age, '')
← fun2 ('guru', 38)
← fun2 (38, 'guru')
  fun2 (20, 20)
```

The type of the parameter is not considered.

Key word argument

The user can use the parameter name to pass the argument w/o need to follow the order of the parameters.

```
def si (P, t, r):
    Print ("Interest:", (P*t*r/100))

si (1000, 9, 5)
si (P=1000, t=5, r=9.5)
si (1000 t=5, r=9.5)
si (P=1000, 5, 9.5) X Invalid Syntax error
si (1000, P=2000, t=5, r=9.5) X Invalid. Runtime error
```

Default Arguments

These arguments values are assumed to be present if not provided explicitly

```
def area (pi = 3.14, r = 1):
    print ('area of circle is:', pi*r*r)
area()
area (r = 2.3)
area (3.1415, 4.5)
area (3.1415, r = 23)
```

Local & global scope of variable

eg 1

```
p = 20
def demo():
    q = 10
    print ("the prod is", p*q)
```

eg 2:

```
demo()
print ("the local var is", q)    X invalid.

def demo():
    q = 10
    print ('local', q, 'global', p, )

p = 30
demo()
```

global & local with same name

Eg 3:
```
def demo():
    q = 20
    Print(q)    // val in fun
q = 30
demo()
Print(q)    // val outside fun
```

Eg 4:
```
~~ated~~ q = 20
def demo():
    global q
    q = 30
    Print("inside fun", q)
demo()
Print("outside fun", q)
```

Additional arguments

We can specify additional arguments which are not part of function prototype by using dictionary

Eg:
```
def fun3(a, b, **x):
    Print(a, b)
    for k, v in x.items():
        Print("the value of {} is {}".format(k, v))
fun3(1, 2, c=3, d=4, e=5)
fun3(1, c=4, b=8, n=20)
```

Variable arguments

The function in Python can be made to accept variable no of arguments using a tuple prefixed by *

```
def sum (*x):
    s = 0
    for i in x:
        s += i
        Print ("sum is", s)
sum (2, 3)
sum (22, 33, 44, 55, 66)
sum (2.3, 4.3, 5.6)
```

Variable arguments with Positional Parameters

Known no of arguments + optional sequence of addn'l values

```
def sum (a, b, *x):
    s = a+b
    for i in x:
        s += i
    Print (s)
sum (2, 3)        sum ()  - Error
sum (2, 3, 4, 5, 6, 7)
```

```
def sum (*x, a, b)
```

No positional parameters after var args.

Default and Variable Arguments ✗

```
def sum (a, b=100, *x):
    s = a+b
    for i in x:
        s += i
    Print (s)
sum (2)
sum (2, 3, 4, 5, 6, 7)
sum (b=2, a=4, 6, 78)
sum (a=7, b=2, 10, 11, 12)
```

✗

Variable arguments followed by default arguments

default arguments specified after varargs should be specified using keyword arguments

```
def sum(*x, n=False):
    s=0
    for i in x:
        s+=i
    if n:
        s=-s
    print(s)
sum(2,3,4)
sum(2,5,8,9, n=True)
```

Variable arguments followed by keyword arguments

```
def sum(a,b=2,*c,**d):
    Print(a,b)
    for i in c:
        Print(i)
    for k,v in d.items():
        Print("{} Value is {}".format(k,v))
sum(1,2,3,4,5,6,7, x=8, y=9, z=10)
```

Return

The function return back after exn by default. The return statement can be used to return value or only control when ever needed in function.

Syntax

```
return value
```

a = ( ) (0)

a = [ ]

a = (10, 20, 30)

Date 26

```
def min(a,b):
    if (a>b):
        return a
    elif (b>a):
        return b
    else:
        return "both are Equal"
print(min(100,85))
```

~~def isPrime(n):~~

~~for i in range(2, int(n/2))~~

Eg2

For loop Example Pgm

Ceaser.py

```
msg = input("Enter the msg")
newmsg = ''
shift = input("Enter the shift")
for ch in msg:
    if ch >='a' and ch<='z':
        p = ord(ch) - ord('a')
        p = (p+shift) % 26
        newch = chr(p + ord('a'))
        newmsg = newmsg + newch
    else:
        if ch>='A' and ch<='Z':
            p = ord(ch) - ord('A')
            p = (p+shift) % 26
            newch = chr(p + ord('A'))
            newmsg = newmsg + newch

print(newmsg)
```

Prime series.py

```
def isPrime(x):
    for i in range(2, x//2, 1):
        if x % i == 0
            return False
    return True
n1, n2 = input("Enter 2 num").split(' ')
n1 = int(n1); n2 = int(n2)
for i in range(n1, n2+1, 1):
    if isPrime(i):
        print('{} is Prime'.format(i))
    else:
        print('{} is not Prime'.format(i))
```

Even parity $\boxed{\text{if else}}$

```
import sys
n = input ("Enter 8 digit number")
if (n.count('0') + n.count('1') != 8 or len(n) != 8):
    Print ('there is no 8 bits')
else:
        ones = n.count('1')

if ones % 2 == 0:
    Print ("Parity to add is 0")
else
    Print ("Parity to add is 1")
```

Returning multiple Values

multiple values can be returned from the function as a single collection - which could be tuple/list of dictionary / object.

Eg:

```
x, y = input ("Enter two numbers") . split (',')
x = int (x) ; y = int (y)
def calc (a, b):                    fn def
    t = (a+b, a-b, a*b, a/b)
  return t

res = calc (x, y)                   fn call
Print (res) ← tuple

Print ('sum = {}, diff = {}, Prod = {}, quot = {}'. format (res[0],
                                res[1], res[2], res[3]))
                                    ↳ list
```

Returning Dictionaries

```
def calc (x, y):
    t = {'sum': x+y, 'diff': x-y, 'Prod': x*y, 'quot': x/y}
    return t
a, b = input ('enter two nos').split (',')
a = int (a) ; b = int (b)
res = calc (a, b)
Print (res)
Print ('sum = {}, diff = {}, Prod = {}, quot = {}'.format(res['sum'],
                res['diff'], res['Prod'], res['quot']))
```

(+)

Recursive Functions

```
def fact (n):
    if n == 1:
        return 1
    Else
        return n*fact (n-1)
n = Eval (input ('enter num'))
r = fact (n)
Print (r)
```

max = 500 stacks

fibonacci

```
def fibonacci (n):
    ~~if (n==1):~~ x = 0; y = 1; sum = 0
    ~~Print~~
    if n == 1:
        Print (x)
    elif n == 2
        Print (x, y)
    else:
        Print (x, y)
        for i in range (3, n+1, 1):
            sum = x + y
            Print (sum)
            x = y
            y = sum
n = Eval (input ('enter num'))
fibonacci (n)
```

Nested functions

A function defined within in a function, the inner function is local to outer function

Eg1:
```
def f():
    Print('this is f')
    def g()
        Print ('this is g')
    g()

f()
g() - value Error.
```

Eg2:
```
import math
def area (a, b, c):
    def s_compute():
        s = (a+b+c)/2
    return s
    s = s_compute()
    ar = math.sqrt(s*(s-a)*(s-b)*(s-c))
    return ar
x, y, z = input('enter 3 sides').split(',')
x = float(x); y = float(y); z = float(z)
a = area (x, y, z)
Print (a)
```

$^nC_r = \frac{n!}{(n-r)!} * r!$

Outer function as Generator

```
import math
def power(n):
    def inpow(x):
        return int(math.pow(x,n))
    return inpow
square = power(2)
cube = power(3)
print('Equation: ax^3 + ax^2 + cx + d')
a,b,c,d = input('enter the values of a,b,c,d').split(',')
a,b,c,d = int(a), int(b), int(c), int(d)
x = int(input('enter the value of x'))
res = a*cube(x) + b*square(x) + c*x + d
print(res)
```

power() is generator function returns reference to function stored in square & cube

power() returns inpow() hence square(x) and cube(x) can be used

Lambda Expression

→ mechanism used for creating anonymous function

→ used to do simple task fit within an expression w/o formally defining a function.

Syntax: lambda parameters : expression

Eg: sq = lambda x : x**2
sq(4)

Unpacking argument list

Collection can be tuples, lists, sets dictionaries

* collection - tuples, lists, sets

** collection - dictionaries

When collection is passed as parameter to function, it is received as single object not as individual elements

Eg:
```
def f(x):
    print(x)
```

tuple

f((1,2,3,4,5)) o/p (1,2,3,4,5)

If we want the individual values in the collection then it has to be unpacked

Eg:
```
def calc(x,y)
    return (x+y, x-y, x*y, x/y)
```

```
L[2,3]
calc(*L)
```

Eg:
```
def calc(x,y)
    return (x+y, x-y, x*y, x/y)
```

```
D = {'x':2, 'y':3}
calc(**D)
```

Module-2

Strings

String is a immutable sequence of characters; string is a object of the class str.

s1 = str() s1 = ""

s2 = str("Hello") s2 = "Hello"

String is immutable means unchangable i.e string object once created cannot be changed.

Eg: str1 = "Hello"
str1[0] = "A"
Print (str1) raises Error.

Eg: str1 = "Knowledge"
str1 = str1 + "base"
before & after id (str1)

str1 = "Hello" str2 = "Hello" id (str1) id (str2)

String built in functions

a = "hello this is new STRING"
len (a) - returns length
min (a) - returns smallest char
max (a) - returns largest char.

Index [] operator.

s1 = "Python" s1[0] 'p' s1[1] 'y'
s1[-1] 'n' s1[-6] 'p' // negative indexing

Traversing with loops

```
s = "India"
for c in s:
    print (c)
```

```
S = 'I LOVE PYTHON PROGRAMMING'
for c in range (0, len (s), 2):
        Print ( s[c], end = " ")
```

```
S = 'India'
i = 0
while i < len(s):
    Print(S[i], end = " ")
    i+= 1
```

STRING OPERATORS

Slice operator:

Syntax:

String Variable [Start : End]

Slice returns subset of string called slice b/w the two indices

S = "BMSIT-BANGALORE"

S [4 : 10] T-BANG S[6:] BANGALORE S[:6] BMSIT-

Slicing with step size

Syntax

String variable [Start : End : Step_size]

S [0 : len (s) : 2] BSTBNAOE

S [::] Prints entire string

S [::-1] Prints the string in reverse order

S [-1 : -16 : -1] Prints the string from -1 to -16

S [:-1] Prints entire string except last character.

Strings + * and in operator

+ → Concatination

S1 = "hello" S2 = "BMSIT" S1+S2 hello BMSIT

* -> Concatenate the same string multiple times

S1 = "Hello"

S2 = 3*S1 ' S2 contains Hello 3 times.

in and not in : used to check whether a string is present in another string

S1 = "computer science and Engineering"

"and" in S1 — True

"science" in S1 — True

"Hello" in S1 — False

"Hello" not in S1 — True

Prg: Read two strings and find the words which are common.

```
S1 = input("Enter string 1") ; S2 = input("enter string 2")
for i in S1:
    if i in S2:
        Print (i, end="")
```

String Comparision

operators such as ==, <, >, <=, >= != are used to compare strings

Eg: S1 = abcd ; S2 = ABCD

S1 != S2 — TRUE

S1 > S2 — True

S1 == S2 — False

format method

The following are the different ways the string variables can be replaced in the place holder

Eg: Name = input("Enter name") ; Place = input("Enter Place")

```
Print ("my name is %s and I am from %s" % (name, Place))
Print ("my name is {} and I am from {}".format (name, Place))
Print ("I am from {1} and my name is {0}".format (name, place))
Print ("I am from {b} and my name is {a}".format (a=name, b=place))
```

split()

Returns list of words in the string, breaks string into smaller str

eg: c = input ('enter company name')

~~[illegible]~~ c = c.split()

for i in c:

print (i, end = "\n")

Testing String

str.isalnum()	str.islower()
str.isalpha()	str.isupper()
str.isdigit()	str.isspace()

Searching substring in a string

S = "Python Programming"

S.endswith ("ing") — True

S.endswith ("Java") — False

S.startswith ("Py") — True

S.startswith ("thon") — False

S.find ("thon")	S.find (n)	S.rfind ("n")	S.count ("n"
2	5	16	2

Converting string to another string

S = "hello"

S.capitalize()	S.upper()	S.lower()	S.title()	S.swapcase
Hello	HELLO	hello	Hello World	HELLO

S = "I brought two pencils two pens and two scales"

S.replace ("two", "three") — replace two by three all

S.replace ("two", "three", 2) — replaces first two appearance of two

Removing unwanted Characters from String

strip() - remove the white spaces \t \n

s = " \thello \t \t \t"

s.strip() s.lstrip() s.rstrip()

hello hello \t \t \t " \t hello"

s1 = "@@@ Hello @$$"

s.strip('@$') s.lstrip('@$') s.rstrip('@$')

Hello Hello@$$ @@@ Hello

Formatting Strings

s = "Hello World"

s.center(145)

' Hello World '

s.ljust(145)

'Hello World '

s.rjust(145)

' Hello World'

Eg: Check whether record word is reverse of given word

```
def rev(s1, s2):
    if s1 == s2[::-1]:
        return True
    else:
        return False
w1 = ("enter word1")
w2 = ("enter word2")
x = rev(w1, w2)
print(x)
```

File Handling

A file is a logical container of data, stored as a single unit in file system on a storage device.
Files can either be:
(i) Accessed directly by user
(ii) opened, viewed, edited by using specific applications
(iii) Can be used internally by an applications.

Various operations on file
i) Create file
ii) open file
iii) Reading from file
iv) writing into file.
v) Closing the file

Text v/s Binary Files

Text files contains human readable text,
Binary files are encoded binary data not human readable
Both text and binary files are stored as binary only
Eg: text files → abc.txt, prg1.py, mno.doc
Binary file → song.mp3 pic.jpg movie.3gp

Opening & Closing Files.

Open()

Syntax file_obj = open ("pathname", "mode")

Pathname – specifies the filename with path
mode
r - open for read only
w - open for write, create file if not exist
a - open for append, create file if not exist
rb - open binary file for read

wb - open binary file for write
ab - open binary file for append

open - returns file object on successfull opening of file
returns error if fails to open the file due to reasons.

Eg: f = open("abc.txt", "r")
f = open("abc.txt", "ab")

Close()

To close already opened file
syntax: fileobject.close()

Eg: f = open("abc.txt", "w")
f.closed # to check file is closed/not
False
f.close()
f.closed
True

Reading from Text File

The contents of the opened file can be read using the file object using different ways.

i) Reading entire content at one go
ii) Reading file contents one line at a time
iii) Reading file contents a chunck at a time
iv) Reading file contents a character at a time

(i) Reading the Entire Content of file

The entire content of the file can be read as a string or as a list

Reading content of file as String

```
fname = input ("Enter the file name")
try :
    f = open (fname, "r")
    c = f.read()
    Print (c)
except Exception as e:
        Print ("unable to open file {}".format(fname))
        Print ("Reason : {}".format (str(e)))
```

Reading content of file as list of lines

```
fname = input ("Enter the file name").
try
        f = open (fname, "r")
        c = list (f.read())
        for l in c:
            Print (l, end = " ")
except Exception as e:
        Print ("unable to open file {}".format(fname))
        Print ("Reason {}".format (str (e)))
```

Note: The above method may lead to unstable program if the size of the file is too large and memory might be insufficient

ii) Reading a line at a time iterating through the file

The file is a sequence of lines that can be directly iterated upon.

```
fname = input("Enter filename")
try:
    f = open(fname, "r")
    for l in f:
        Print(l, end="")
except Exception as e:
    Print("unable to open file: {}".format(fname))
    Print("Reason: {}".format(str(e)))
```

readlines()

Syntax

file object.readline([size])

The method reads entire file and returns a list of strings with each item being line of the file

```
f = open(fname, "r")
~~for a in~~
c = f.readlines()
for l in c:
    Print(l, end="")
```

readline()

Syntax

fileobject.readline([size])

The method reads single line from file and returns.

```
f = open(fname, "r")
while 1:
    c = f.readline()
    Print(c, end="")
```

Reading arbitrary amount of data

The required no of bytes can be read from the file by using read(bytes) method.

Eg:
```
f = open (fname, "r")
while 1:
    c = f.read (100)
    if not c: break
    Print (c, end = " ")
```

Writing to Text files

Writing to text file is done using the function write()

Syntax:

file object . write (string)

file object . close()

NOTE: For write to complete mandatory to close the file

Eg: Write factorial of 1 to 10 into file

```
import math
fname = input ("Enter filename")
try
    f = open (fname, "w")
    for i in range (1,11):
    f.write ("The factorial of {} is {}\n". format(i, math. factorial(i)))
f. close()
except Exception as e:
    Print ("unable to open file {} . format (fname))
    Print ("Reason {}". format (str(e)))
else:
    Print ("File is written")
```

Seek()

seek() is used to set/reset the current file position while read or write operation

Syntax:

file object.seek(offset, whence)

Whence can be:

0 → SEEK_SET | 1 → SEEK_CUR | 2 → SEEK_END

NOTE: when file is opened in r/w/a mode the non zero offset seek can be performed only on SEEK_SET but will through exception for SEEK_CUR and SEEK_END hence use rb/wb/ab modes and use .decode('utf-8') with read/write

Eg:

```
fname = input("Enter filename")
f = open(fname, "rb")
c = f.read(10).decode('utf-8')
print(c)
print(f.tell())
f.seek(-10,2)
print(f.tell())
c = f.read(10).decode('utf-8')
print(c)
f.close()
```

f.tell() - returns the position of the file object

f.name() - returns the name of the file pointed by f

Reading and writing Binary File

To read and write into a binary file $\boxed{\text{Pickle}}$ module is used

Pickle module is capable of converting object into a stream of bytes and write into a file also capable of reading sequence of bytes from a binary file and converting them into an object

To write

Pickle.dump (object, file)

writes the object into the specified file (opened)

To read.

Pickle.load (file)

reads the ~~object~~ byte from the opened file and returns an object

Eg: Employee database

```
import pickle
class Employee:
    def __init__(self, name, id, degn):
        self.name = name
        self.id = id
        self.degn = degn
f = open("Employee.dat", "ab")
name = input("Enter name")
id = input("Enter id")
degn = input("Enter degn")
pickle.dump(employee(name, id, degn), f)
f.close()
try:
    f = open("Employee.dat", "rb")
    while 1:
        e = pickle.load(f)
        Print("Name:{} id:{} Designation:{}".format(e.name, e.id, e.degn))
except EofError: pass
```

def __init__(self) - is a constructor used to initize the attributes of the class

head.py

```
fname = input("Enter filename")
try:
    f = open(fname, "r")
    for i in range(10):
        l = f.readline()
        if not l: break
        print(l, end="")
except Exception as e:
        print("Unable to open file{}".format(fname))
        print("Reason {}".format(str(e)))
```

wc.py

```
fname = input("Enter file name")
lc, wc, cc = 0, 0, 0
f = open(fname, "r")
while 1
        l = readline()
        if not l: break
        lc += 1
        wc += len(l.split())
        cc += len(l)
print("lines = {}, words = {}, characters = {}".format(lc, wc, cc))
```

Module 3

Lists, Dictionaries, Tuples, Regular Expressions

Lists

List is a ordered sequence of elements that can by dynamically altered
ordered - each element has an index based on it's position in list
sequence - elements are arranged in order based on index
dynamically orderd - each item in the list can be changed / list itself can be changed i.e items can be added, deleted, replaced.

Creating list

[ ] is used to denote the list

L = list([5, 2, 3])	type (l)
l = [5, 2, 3]	<class list>
l o/p [5, 2, 3]	

Accessing list Elements

Each Element in the list has an index : index increase from L→R from 0 to n-1 and R-L from -1 to -n

Eg: l = [3, 4, 5]

l[0] → 3 l[1] → 4 l[2] → 5 l[-1] → 5 l[-2] → 4 l[-3] → 3

Elements of list can be freely modified.

l = [3, 4, 5]	l[2] = l[0] + l[1]
l[0] = 90	l → [90, 4, 94]
l → [90, 4, 5]	

Counting list Elements

len() function can be used to count no of elements in list

l = [4, 5, 6, 8, 9, 10]	l1 = [ ]
len(L)	len(l1)
6	0

Iterating and Searching in list

```
l = [2, 3, 4, 5, 6, 7]
for i in l:
        print(i)
```

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

4 in l	5 in l	6 not in l	2 in l
True	True	False	False

Count accourance in list

count() tells how many instances of element is present

```
l = [3, 4, 3, 5, 3, 6, 3, 8]
```

l.count(3)	l.count(8)	l.count(2)
4	1	0

Locating Elements in list

list.index(x, i, j) optional

- list.index(x) - searches and returns the index of first accourance of x in the list
- i - starts searching from the specified index instead of 0
- j - stops searching ~~from~~ at the specified index instead of n-1

Eg:

```
l = [5, 2, 3, 2]
```

l.index(2)	l.index(4)
1	not in list

```
l = [5, 2, 3, 2]
```

l.index(2, 2)	l.index(3, 0, 2)
3	not in list

List slice

Syntax

list [start : end]

will Extract sub-list Extracted from a list b/w start and End.

l = [5, 2, 3, 2]

l[1:3]	l[1:]	l[:3]	l[:]	l[::-1]
2, 3	2, 3, 2	5 2 3	5, 2, 3, 2	2, 3, 2, 5

Replacing / inserting Elements in list using slice

l = [5, 2, 3, 2]	l = [5, 2, 3, 2]	l = [5, 2, 3, 2]
l[1:3] = [3, 2]	l[1:3] = [3, 0, 0, 2]	l[1:3] = [9]
l → [5, 3, 2, 2]	l → [5, 3, 0, 0, 2, 2]	l → [5, 9, 2]

l = [5, 2, 3, 2]	l = [5, 2, 3, 2]	l = [5, 2, 3, 2]
l[1:1] = [9, 8, 7]	l[1:2] = []	l[2:] = []
l → [5, 9, 8, 7, 2, 3, 2]	l → [5, 3, 2]	l → [5, 2]

l = [5, 2, 3, 2]	l = [5, 2, 3, 2]
l[:2] = []	l[:] = []
l → [3, 2]	l → []

Adding and Deleting Elements

Append Elements

list.append(x) - adds one or more Elements at the end of list

l = [5, 2, 3]	l[len(l):] = [9]
l.append(9)	$\boxed{\text{list.append}(x) = \text{list}[\text{len}(\text{list}):] = [x]}$
l → 5, 2, 3, 9	

list.extend(L) – adds all elements of list L to the list

l1 = [1,2,3,4]	list.extend(L) = list[len(list):] = L
l2 = [5,6,7,8]	l1[len(list):] = l2
l1.extend(l2)	
l1 → 1,2,3,4,5,6,7,8	

Inserting Elements

list.insert(i,x) – adds the element x at index i

l = [5,2,3]	list.insert(i,x) = list[i:i] = [x]
l.insert(2,9)	l[2:2] = [9]
l → 5,2,9,3	list.append(x) = list.insert(len(list), x)

Note: If the index is beyond the end of the list the element is inserted at the end

l = [5,2,3,4]

l.insert(50,9)

l → 5,2,3,4,9

Deleting Elements

del list[index] – deletes the element at specified index

l = [5,2,3]	l = [1,2,3,4,5,6]	l = [1,2,3,4,5,6]	del l
del l[1]	del l[1:4]	del l[:]	l
l → 5,3	l → [1,5,6]	l → [ ]	error

list.remove(x) – removes the first occurance of x in the list

l = [5,2,3,2]	l.remove(8)
l.remove(2)	not in list
l → 5,3,2	

list.pop(index) – deletes the element at the specified index

If index not specified delete the last element

l = [5,2,3,2]	
l.pop(1)	l.pop()
l → 5,2,3	l → 5,2,3

list.clear() - removes all elements of the list

```
l = [5,2,3,2]
l.clear()
l -> []
```

Adding, Multiplying & copying list

Adding Lists

l1 = [1,2,3] l2 = [4,5,6]	l = [3,4] l += l2 = l.extend(l2)
l = l1 + l2	l += [7,8]
l -> 1,2,3,4,5,6	l -> 3,4,7,8

Multiply lists

```
l = [2,3]                    list * n = n * list
l * = 3
l -> [2,3,2,3,2,3 2,3]
```

Assigning and copying lists

l1 = [5,2,3]	l1 = [5,2,3]
l2 = l1	l2 = l1.copy()
l2 -> 5,2,3	l2 -> 5,2,3
l1[0] = 9	l1[0] = 9
l1 l2	l1 l2
9,2,3 9,2,3	9,2,3 5,2,3

difference b/w l2 = l1 and l2 = l1.copy()

Other operations on lists

l = [1, -1, 20, 3, 55, 86]

min(l)	max(l)	l.reverse()	l.sort()
-1	86	86, 55, 3, 20, -1, 1	-1, 1, 3, 20, 55, 86

l.sort(reverse = True)
86, 55, 20, 3, 1, -1

Nested List

A list contains elements which are references to objects and these objects could be of any type including lists. A list within a list is nested list

```
l1 = [1, 2, 3]
l2 = [4, 5]
l3 = [6, 7, 8, 9]
l = [0, l1, l2, l3, 10, 100]
l -> [0, [1, 2, 3], [4, 5], [6, 7, 8, 9], 10, 100]
```

```
type(l[0])      int
type(l[1])      list

l[3,2]          8
l[1,1]          2
```

Eg:

```
d = int(input("Enter a digit b/w 0-9"))
w = ["zero", "one", "two", "three", "four", "five", "six", "seven",
     "eight", "nine"]
Print("{} is {}".format(d, w[d]))
```

TUPLES

Tuples are immutable ordered sequence of elements

immutable - contents of tuples cannot be changed.

ordered - Each element is indexed based on posⁿ

Sequence - ordered by indicies, traversed by indicies

Creating Tuples

tuple() - constructor	(,)	$e_1, e_2, e_3,$

t_1 = tuple (1, 2, 3)	t_2 = (5, 6, 7)	t_3 = 8, 9, 10	Empty t_4 = ()
$t_1 \rightarrow (1, 2, 3)$	$t_2 \Leftrightarrow (5, 6, 7)$	$t_3 \rightarrow (8, 9, 10)$	$t_4 \rightarrow ()$

Singleton Tuple - a tuple with one element

t_5 = (5)	type (t_5) → int	both the cases it is int object is created not tuple
t_6 = 8	type (t_6) → int	
t_5 = (5,)	type (t_5)	→ tuple
t_6 = 8,	type (t_6)	

Accessing tuple Elements

t = (8, 9, 7, 6, 5)

t[0] ⇒ 8 t[1] → 9 | t[-1] → 5 | t[-2] → 6

Indexing 0 to n-1 L→R and -1 to -n from R→L

t[0] = 20 // Error as tuple is immutable

Counting tuple Elements & Iterating through tuples

t = (5, 2, 3, 4, 6)

len (t) → 5

```
t = (5, 2, 3, 4, 6)
for i in t:
    Print (i)
```

Searching in tuples

in and not in

t = (2, 3, 8, 9, 2)

2 in t	4 in t	4 not in t	8 not in t
True	False	True	False

count()

t = (5, 2, 3, 2, 3, 5)

t.count(5)	t.count(3)	t.count(8)
2	2	0

index()

tuple.index(x, i, j)

x → element

i → start pos

j → end pos

t = (5, 2, 3, 8, 9, 2)

t.index(3)	t.index(8)	t.index(6)
2	3	Error.

t.index(2, 3)	t.index(8, 1, 3)
2	Error.

Tuple slice

tuple[start : end]

start → default 0 if needed end $\boxed{\text{len}(t)}$

t = (5, 2, 3, 2)

t[1:3]	t[:]	t[:-1]	t[len(t):]
(2, 3)	(5, 2, 3, 2)	(5, 2, 3, 2)	()

Adding and Multiplying tuples

+, += are used for concatination

*, *= are used to multiply tuples

$t_1 = (1,3)$	$t_2 = (7,8)$	$t_3 = t_1 + t_2$	$t_3 \rightarrow (1,3,7,8)$
$t_1 = (1,3)$	$t_1 += (7,8)$	$t_1 = (1,3,7,8)$	

$t = (5,2,3) * 3$	$t = (5,2,3,5,2,3,5,2,3)$

Assigning and copying tuples

tuples are immutable i.e. we cannot add/remove/change elements of tuples,

If the element in tuple is reference to some object then the contents of the object can be changed.

$t1 = (5, [\,], 3)$	$t1[1].append(2)$
$t2 = t1$	$t1 \rightarrow (5, [2], 3)$
$t2 \rightarrow (5, [\,], 3)$	

Other operations on tuples

$t = (5, 2, 3, -1, 25)$

$min(t) \rightarrow -1$	$max(t) \rightarrow 25$

sorted(t)	sorted(t, reverse = True)
$(-1, 2, 3, 5, 25)$	$(25, 5, 3, 2, -1)$

Lists & tuples are inter convertable:

$l = [5,2,3]$	$l = list(t)$
$t = tuple(l)$	l
$t \rightarrow (5,2,3)$	$[5, 2, 3]$

List	Tuple
Dynamically alterable	Immutable
Homogeneous	Heterogeneous
use when contents change	use when contents are freezed
||| to C/C++ arrays	||| to statically initialized C/C++ array

Program to Print first n fibonacci nos using tuples.

```
t1 = t2 = 1
n = int(input("How many terms?:"))
for i in range(n):
    print(t1)
    t1, t2 = t2, t1+t2
```

zip() function

zip() - creates list of tuples

A1 = [1, 2, 3] A2 = [x, y, z]

list(zip(A1, A2))

[(1, x), (2, y), (3, z)]

```
L1 = ['Black', 'White', 'Grey']
L2 = [255, 0, 100]
for color, code in zip(L1, L2):
    print(color, code)
```

o/p

(Black, 255)

(White, 0)

(Grey, 100)

Dictionaries

Dictionary is a collection of key value pair where key should be unique.
Keys are members of set which keep track of values.

Creating dictionary

using dict() function:

```
D = dict([('apple':'red'), ('grapes':'green')])
D
{'apple':'red', 'grapes':'green'}
```

In the dictionary is a list consisting of tuples, each tuple has two values.

using { }

```
d1 = {1:'green', 2:'red', 3:'blue'}
d1
{1:'green', 2:'red', 3:'blue'}
```

```
type(d1)
<class dict>
```

{ } is used to create sets also:

[{ }] - with multiple values seperated by [,] is a [set]

[{ }] - with multiple elements seperated by [: and ,] is [dictionary]

Eg: d2 = {1,2,3,4,5} type(d2) <class set>
d3 = {1:'a', 2:'b', 3:'c'} type(d3) <class dict>
d4 = { } type(d4) <class dict> d4 { }
d5 = set() type(d5) <class set> d5 set()

Accessing dictionary elements

```
d = {1:'a', 2:'b', 3:'c', 4:'d', 5:'e'}
d[1]   a
d[4]   d
```

d[1] = "guru"

d {1: guru, 2: 5 ....}

d[8] $\boxed{\text{Key Error: 8}}$ accessing dictionary key beyond limit / non existing key

len() - counting dictionary elements

len(d) → 5

Iterating through keys of dictionary (two ways)

(1)

```
for k in d:
    print(k)
```

1 2 3 4

Using $\boxed{\text{d.keys()}}$

```
k = d.keys()
for i in k:
    print(i)
```

1 2 3 4

Iterating through values of dictionary (two ways)

(1) $\boxed{\text{dict.values()}}$

```
v = d.values()
for i in v:
    print(i)
```

a b c d e

(2) Iterate through keys & print values

```
for k in d:
    print(d[k])
```

a b c d e

Iterating through key value pair

$\boxed{\text{dict.items()}}$

```
for k, v in d.items():
    print(k, v)
```

1 a 2 b 3 c 4 d 5 e

NOTE: The keys in the dictionary must be unique while the values of the keys not need to be unique

Searching in Dictionary

Check for Existing of a key in dictionary

d = {1:'a', 2:'b', 3:'c', 4:'d', 5:'e'}

4 in d	8 in d	3 not in d	8 not in d
True	False	False	True

Extract values of keys using [] & d.get()

① k = '3'
V = d[k]
print (v)

② d.get (3) | d.get(8) | d.get(1, 6)
'c' | no o/p | a

d.get (8,9) o/p 9

Extract key given its value

There is no direct method but can be extracted by computation,

V = 'e'
for k in d:
    if d[k] == V:
        print (k)

Adding and deleting Elements

① By assignment: assigning a value to a new key will add an element to dictionary, if key already exists changes the value

d = {1:'a', 2:'b', 3:'c', 4:'d', 5:'e'}
d [1] = 'abc'
d
{1: 'abc', 2:'b' ....}
d[6] = 'xyz'
d

(2) using d.setdefault()

d.setdefault(key)

If the key exists return the value associated with key,
If the key don't exist create the key with value 'None'

d.setdefault(3)	d.setdefault(8)
'c'	d {1:'abc', 2:'b' ..... 8:None}

d.setdefault(key, default)

If key exists return the value associated with key
If the key don't exist create the key with specified default value

d.setdefault(4)	d.setdefault(9,'mno')
'd'	d {1:'abc', 2:'b' .... 9:'mno'}

Deleting elements

(1) using del

del dict[key] deletes specified key, value pair

Eg: del d[3] d {1:'abc', 2:b, 4:'d' ....}

(2) dict.popitem() - deletes the last item from the dictionary will raise KeyError if called on empty dictionary, returns the deleted element as a tuple

d.popitem()	d.popitem()
(9,'mno')	(8:'green')

(3) dict.pop(key): deletes the specified key value pair, raise key error if key don't exist. return the value of key.

d.pop(1)	d.pop(2)
'abc'	'b'

(4) d.clear() : deletes all the elements of the dictionary.

d.clear() | d {}

Regular Expression :

→ A Regular Expression is a special text for describing search pattern

→ It is used for search, verify, find & replace string & format data

→ The module re provides support for Perl-like regular expression in Python.

→ As different ~~words~~ characters have spl meaning in regular expression we use raw string r'expression'

Regular Expression Patterns

NOTE: To get help!
>>> import re
>>> help(re)

^ Begining of line

$ End of line

. any single character

[...] matches any single char in brackets eg [a-z]

[^...] matches any single char not in brackets [^e-g]

re* match 0 or more appearance of preceding expr^n

re+ match 1 or more appearance

re? match 0 or 1 occurance of previous expr^n

re{n} match exactly n occurance of —"—

re{n,} match n or more occurance

re{n,m} match least n and max m occurance

a|b match either a or b

(re) - groups regular expr^n

(?imx) - i m x option is on while grouping

(?-imx) - i m x option is off

(?#...) - Comment

(?=re) - specifies posn using pattern

\w - matches word [A-Z a-z 0-9 _]

\W - matches nonword

\s - matches space [\t\n\r\f]

\d - matches digit [0-9]
\D - matches non digit
\A - matches beginning of string
\Z or \z matches end of string
\G - matches where last match finished
\b - matches word boundries
\B - matches non word boundries
\n \t - matches new line tab etc

<.*> Greedy repetation matches all characters
<.*?> Non greedy matches till >

match()
The function match RE pattern in the string with optional flags

RE, target string, options using bitwise OR

re.match(Pattern, String, flags)

The function returns match object on success and None on failure
groups() or group(num) can be used to access sub-group of match object

groups() - returns entire match
group(num) - return specific subgroup in num.

eg: str = "cats are smarter than dogs"
mo = re.match(r"(.*) are (.*?) .*", str, re.M|re.I)
if mo:
    Print(mo.groups())
    Print(mo.group(1))
    Print(mo.group(2))

Option flags or RE Modifiers

They control various aspects of matching R.E

re.I - Perform Case insensitive matching
re.L - affects \w and \W and \b and \B
re.M - ^ matches End of line ^ Start of of any line
re.S - . matches any thing including newline
re.U - Interprets letters according to Unicode Char set
re.X - ignores white space treats # as Comment marker

Search()

RE tgt str modifiers

re.Search(Pattern, Str, flags)

It searches the appearance of pattern in string and returns the match object on success and None on failure

Eg: Str = "Cats are smarter than dogs"

```
so = re.search(r'(.*) are(.*?) .*', re.M|re.I)
if so:
  Print (so.groups())
  Print (so.groups(1))
  Print (so.groups(2))
else
    Print ("No match")
```

Note: match() finds the pattern at the beginning of string while search() checks anywhere in string

findall()

re.findall(regex, string)
will return array of all nonoverlaping regx matches in the string

Eg:
```
import re
Str = "Sat, hat, mat, rat, Pat"
a = re.findall("[h-m]at", Str)
for i in a:
    Print(i)
```

o/p
Sat
rat
Pat

Compile() and Sub()

re.Compile("regx")
If the same regular Expression has to be used more than once we should compile it into a regular Expression object.

The re object returned by Compile Provides following function.
search() match(), findall() finditer() sub() split()

re.sub(~~regex~~, replacement, subject)

It Performs search and replacement of regex across the subject

Eg:
```
import re:
Str = "sat, hat, mat, rat, Pat"
r = re.compile("[r]at")
Str = r.sub("food", Str)
Print(Str)
```

o/p sat hat mat food Pat

finditer()

re.finditer(regx, str)

It generates an iterator to iterate over matched object. this object has lots of useful info about the match. which can be accessed using functions:

m.group() - returns regd part of matched string
m.start() - offset of the start of match
m.end() - offset of end of match
m.span() - returns both start and end of match

eg:
```
import re
str = "we have to inform him the latest information"
for i in re.finditer(r"inform", str):
    t = i.span()
    print(t)
```

o/p (11, 17) (34, 40)

Prg to create Dictionary of name and age

```
import re
Nameage = """Guru is 40 and Veena is 38
            Shridhar is 8 and Vedant is 9"""

age = re.findall(r'\d{1,3}', Nameage)
name = re.findall(r'[A-Z][a-z]*', Nameage)
x=0
agedict = {}
for i in name:
    agedict[i] = age[x]
    x = x+1
```

Pg to replace newline by space

```
import re
rstr = '''
this is line1
now this is line2
finally this is line3'''

regx = re.compile("\n")
rstr = regx.sub(" ", rstr)
print (rstr)
```

match 1 character

```
rstr = "123456A$#"
print (re.findall(r"\d{5}" rstr))
print (re.findall(r"\D" rstr))
print (re.findall(r"\W" rstr))
'''' \w \s \S \W \D
```

web scrape

```
import re
import urllib.request
url = "http://www.summet.com/dmxi/html/codesamples/addresses.html"
response = urllib.request.urlopen(url)
html = response.read() ; htmlstr = html.decode()
pdata = re.findall (r"((\d{3}) \d{3}-\d{4}", htmlstr)
for i in pdata:
    print(i)
```

*** End of Module 3 ***

Module - 4

Classes & objects Classes & functions Classes & Methods

8 Principles of OOP

① **Class:** A class is a design of a real world entity, comprises of attributes (data) and behaviour (methods)

② **Object:** An object is an instance of class
The objects are identical in terms of design but the attribute values are different hence the behaviour also varies based on attribute values.

③ **Data Encapsulation:**
The class encapsulates attributes (data) and methods (code that acts on data) into a single unit.

④ **Data Hiding:**
Every object should represent a real world entity and the data of objects be hidden and only accessible by the methods of the objects.

⑤ **Data Abstraction:**
The implementation details of methods are hidden and only the interface be exposed. User need to know only interface and impl^n can be changed w/o affecting interface.

⑥ **Polymorphism:** It is multiple form of same entity, performing different operations from different implementations based on the data using same method name.

(i) **Operator overloading:** Performing diff operations by using the same operator with different operands

(ii) Dynamic Polymorphism: Performing different operations using different implementation depending on the type of invoking object.

(7) Inheritance:

A class acquires all the features and properties of another class.

Reusablity, Extensablity and Compartmentalization are the uses of inheritance

(8) Message Passing

Inter object communication during runtime is implemented by message passing (making function calls with arguments i.e messages)

Defining Classes

Syntax:

```
class className:
    Statements
```

The statements can be Blank lines, Comments, class variables, class function

Eg
```
class A:
    pass
class A:
    """ This is our first implemenation of class"""
    pass
A.__doc__
```

o/p
```
'This is our ----
```

```
A.dir()
```

Lists all the modules that can be invoked on the class.

Instantiating Classes

To create object of the well defined class

Syntax:

Var = Classname()

Eg: a = A() a b/f __main__.A object

~~dir Date()~~ type(a)

* <class '__main__.A'>

a.__doc__

Instance Variables

The independent variables of the object are instance variable

NOTE: In C++ and Java class definition specifies the instance variable but in Python object can create its own instance variable

The instance variables are created using methods

Eg:
```
class Date:
    def setDate(self, d, m, y):
        self.day, self.month, self.year = d, m, y

d.Date()
d.setDate(1, 2, 2000)
```

d.day	d.month	d.year
1	2	2000

self: is the reference to the invoking object passed implicitly first argument of method.

Every instance variable belonging to the invoking object is explicitly preceded by reference self.

Instance Methods

They are the functions of class that are invoking objects (instance variables) using parameter self.

Eg:
```
class Date:
    def setDate(self, d, m, y):
        self.day, self.month, self.year = d, m, y
    def getDay(self): return self.day
    def getMonth(self): return self.month
    def getYear(self): return self.year

d = Date()
d.setdate(1, 2, 2000)
print("{}-{}-{}".format(d.getDay(), d.getMonth(), d.getYear
```

Eg2:
```
class Date:
    def setDate(self, d, m, y):
        if self.valid(d, m, y):
        self.day, self.month, self.year = d, m, y
        else
            print("Invalid Date")
    def getDay(self): return self.day
    def getMonth(self): return self.month
    def getYear(self): return self.year
    def print(self):
        print("{}-{}-{}".format(getDay(), getMonth(), getYear()))
    def valid(self, d, m, y):
        if y<1 or y>9999: return False
        if m<1 or m>12: return False
        dim = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
        if self.isLeap(y): dim[2] = 29
        if d<1 or d>dim[m]: return False
```

```
def isLeap(self, y):
    return y%4==0 and(not y%100==0 or y%400==0)
def adddays (self, days):
  d, m, y = self.day, self.month, self.year
  dim = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
  if self.isLeap (y) : dim[2] = 29
  for i in range(days):
      d = d+i
      if d> dim[m]:
          d = 1
          m = m+1
          if m>12:
              m = 1
              y = y+1
              if isLeap (y) : dim [2] = 29
              else
                  dim [2] = 28
  result = Date()
  result. setDate(d, m, y)
  return result

d1 = Date()
d1. setDate (1, 2, 2000)
d2 = d1. adddays (100)
d2 = Print ()
```

Class Variables

They are the variables that belongs to class and are shared across all instances of that class.
They are accessible by using classname/objectname

Eg: class A — class variable
x = 10

A.x o/p 10

a = A()
b = A()

a.x	b.x
10	10

a.x = 20

A.x	a.x	b.x
10	20	10

NOTE: When an object changes/initiates the value of a class variable it creates further created an instance variable not a class variable

A.x = 77

A.x	a.x	b.x
77	20	77

Class Functions

They are the functions defined inside the class.
They are invoked using class name (class object) but not using the reference to invoking object - self

Eg:
```
class A:
    x = 10
    def inc():
        A.x = A.x + 1
>>> A.x        10
>>> A.inc()
```

```
a = A()
b = A()
a.x        b.x
11         11
```

```
a.inc()
```
Error:
Attribute Error.

```
class Date:
    dim = [0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
    def valid (d, m, y):
        Date.dim[2] = 28
        if y<1 and y > 9999 : return False
        if m <1 and m >12 : return False
        if Date.isLeap(y) : Date.dim[2] = 29
        if d<1 and d > Date.dim[m] : return False
    return True

    def isLeap (y):
        if y % 100 == 0 and y % 400 == 0:
            return True
        elif y % 100 != 0 and y % 4 == 0 :
            return True
        else:
            return False

    def setDate (self, d, m, y):
        if Date.valid (d, m, y):
            self.day, self.month, self.year = d, m, y
        else
            Print ("Invalid Date")
    def getDay (self) : return self.day
    def getmonth (self) : return self.month
    def getyear (self) : return self.year
    def Print (self):
    Print ("{}-{}-{}".format(self.getDay(), self.getmonth()
                                self.getyear()))

    def addday (self, days):
        d, m, y = self.days, self.month, self.year
        if (Date.isLeap(y)) : Date.dim[2] = 29
```

class variable

class function

class function

```
for i in range(days):
    d = d+1
    if (d > Date.dim[m]):
        d = 1
        m = m+1
        if m > 12:
            m = 1
            y = y+1
            if Date.IsLeap(y): Date.dim[2] = 29
            else:
                Date.dim[2] = 28
result = Date()
result.setDate(D, m, y)
return result

d1 = Date()
d1.setDate(1, 2, 2000)
d2 = d1.add(100)
d2.print()
```

<u>Instance methods as special class function</u>

```
class B:
    def f(self):
        Prin("Hello")
```

B.f()	a = B()	B.f(a)
Error	a.f()	Hello
	Hello	

Public, Private and Protected Members

① **Public:** Public members are accessable everywhere where the class/object is accessable.
By default all the class members are public

② **Protected:**
These members are accessible in the class which defines and in the subclasses that inherit it
The protected members are prefixed with _ (one underscore)
$\boxed{\_\text{member}}$

③ **Private:**
These members are accessable only within the class in which it is defined
Private members start with 2 underscore and end with at most (maximum) 1 underscore
$\boxed{\_\_\text{member}\_}$ will be replaced by $\boxed{\_\text{Classname}\_\_\text{member}}$
This change of name is called name mangling - used to prevent direct access to member by it's name

```
class A:
    def set(self, x, y):
        self.x = x
        self.__y = y
    def print(self):
        print("{},{}".format(self.x, self.__y))
```

In date prg convert dim, days, months and years into private
date_private.py

```
a = A()
a.set(2,3)
a.print()
```

o/p 2,3

```
print(a.x)      o/p 2
print(a.y)      Error
print(a.__y)    Error
print(a._A__y)  o/p 3
```

Constructors and Destructors

Constructors:

It is a instance method (with self) that is auto invoke when instance is created and perform initialization.

The constructors are identified by special name __init.

```
Class A:
    def __init__(self):
        print("constructor is called")

a = A()
b = A()
```

} o/p Constructor is called

In date add constructor. - date_constructor.py.

```
    def __init__(self, d, m, y):
        self.setDate(1, 1, 1970)
        self.setDate(d, m, y)

d1 = Date(1, 2, 200)
d1.print()                o/p  1-2-2000
d2 = Date(90, 90, 90)
d2.print()                o/p  Invalid date  1-1-1970
```

Destructors.

It is a instance method that is auto invoked when an object is destroyed using del, it's name __del__

```
Class A:
    def __del__(self):
        print("Destructor called")

a = A()     b = A()
del(a)      del(b)        o/p  destructor called,
```

Inheritance:

Simple Inheritance

(derived class) (base/super class)

one class inherits from other existing class

Syntax:

```
class derived class (base class):
    class defination
```

Eg:

```
class A:
    Pass
class B(A):
    Pass
```

Private Public Protected with inheritance

```
class A:
    def __f1(self): Print("A.f1")        # Private
    def f2(self): Print("A.f2")          # Public
    def _f3(self): Print("A.f3")         # Protected

class B(A):
    def __g1(self): Print("B.g1")
    def g2(self): Print("B.g2")
    def _g3(self): Print("B.g3")

b = B()
```

b.__f1() Error not accessible outside class

b.f2() o/p A.f2 Public accessible everywhere

b._f3() o/p A.f3 Protected accessible in derived class & Private

b.__g1() Error not accessible

b.g2() o/p B.g2

b._g3() o/p B.g3

Public - accessible inside & outside the class

Private - accessible inside class ~~or outside~~

Protected - accessible inside class & its sub class

Function overloading

The function with same name in both base and derived class.

There will be a reference to instance of derived class which is used to invoke specific function based on object

The base class function can be invoked in derived class by using `super()` . function()

```
class A:
    def f(self):
        print ("A.f")
class B(A):
    def f(self):
        print ("B.f")
        super().f()
b = B()                 o/p     B.f
b.f()                           A.f
A.f(b) direct invocation        A.f
```

Constructors and Destructors in Simple Inheritance

In Inheritance Base class can exist independently but the derived class is dependent on base class

When an derived class instance is created automatically a base class instance is also created.

When derived class is constructed base class is constructed first then derived class but when derived class is destroyed base class is destroyed after. (*)

C++ automatically enforces this but in Python it is Programmer responsiblity.

Eg:
```
class A:
    def __init__(self):
        print("A is constructed")
    def __del__(self):
        print("A is destroyed")
class B(A):
    def __init__(self):
        super().__init__()
        print("B is constructed")
    def __del__(self)
        print("B is destroyed")
        super().__del__()

b = B()
del(b)
```

Multiple Inheritance

A class derive from 2 or more base classes is multiple inheritance

Syntax

class className (baseClass1, baseClass2, [baseClass3] ...)

Minimum two base classes are mandatory.

Eg:
```
class A:
    def fa(self):
        print("A called")
class B:
    def fb(self):
        print("B called")
class C(A, B):
    def fc(self):
        self.fa()
        self.fb()
        print("C called")
```

```
c = C()
c.fc()
```

o/p

A called

B called

C called

Function overloading in Multiple Inheritance

In multiple inheritance the super() call will be the first base class

```
class A:
    def f(self):
        print("A")
class B:
    def f(self):
        print("B")
class C(A,B):
    def f(self):
        super().f()
        print("C")
c = C()
c.f()
```

if we change the fun name to b() then o/p will print B, C as the next base class ~~with~~ f() ~~become~~ will be invoked. as f() impl" not found in A.

o/p A
C.

Dynamic Polymorphism

Subclasses can perfectly substitute the base classes. The mechanism of deciding which function to invoke based on involving object is dynamic polymorphism.

```
Class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        pass
class dog(Animal)
    def __init__(self):
        super().__init__("Dog")
    def speak(self):
```

```
class cat(Animal):
    def __init__(self):
        super().__init__("cat")
    def speak(self):
        Print("Meow Meow")

def introduce(Animal):
    Print("Hi This animal is :animal.name)
    Print("This animal says")
        animal.speak()

anima = dog()
introduce(animal)
animal = cat()
introduce(animal)
```

Attribute Handling

① hasattr() - tells whether particular instance has particular attribute or not

$\boxed{\text{hasattr(object, attribute)}}$

```
class A:
    def __init__(self)
        self.x = 0

a = A()
hasattr(a,'x')
True
hasattr(a,'y')
False
```

② getattr() : returns the value of attribute if exists
returns default (if provided) and attribute not exists
returns Error if attribute not exists and no default provide

Syntax: getattr (object, attribute, default)

```
class A:
    def __init__(self):
        self.x = 4

a = A()
getattr(a,'x', 2)
4
getattr (a,'y', 2)
2
getattr (a,'y')
Error.
```

③ Setattr() - Set the value of attribute

Setattr(object, attribute, value)

```
Class A:
    def __init__(self):
        self.x = 2

a = A()
setattr(a, 'x', 8)
getattr (a,'x')
8
setattr(a,'y', 10)
getattr (a,'y')
10
```

Sets the attribute value to the specified value if it exists.
If attribute dont exist it creates the attribute with new value.

④ delattr()

Syntax delattr(object, attribute)

Delete the attribute if exists else attribute Error.

```
class A:
    def __init__(self):
        self.x = 0
a = A()
>>> a.x
0
delattr(a, 'x')
>>> a.x        No attribute Error.
```

Standard Attributes

There are certain standard attributes that is always present within each class:

__name__	name of class
__doc__	Documentation string of class
__bases__	tuple containing base classes of the class
__module__	name of the module which class belongs to
__dict__	Dictionary containing namespace of the class

```
class A:
    pass
class B:
    pass
class C(A, B):
    "A class to demonstrate standard attributes"
    x = 10
    def __init__(self):
        pass
```

```
def f(self):
    pass
print(C.__name__)
print(C.__doc__)
print(C.__module__)
print(C.__dict__)
```

Magic Functions

They are the functions that are automatically invoked without explicitly naming them.

① **Constructors & Destructors**

`__init__` constructor `__del__` destructor.

```
class A:
    def __init__(self): print("created")
    def __del__(self): print("destroyed")

a = A()
created
del(a)
destroyed
```

② **Stringification**

To convert object to a string

$\boxed{\text{str(object)}}$

It calls the method `__str__` of that object. If the object don't contain this method it uses the nearest method of super class

```
class A:
    pass
a = A()
```

o/p

```
<__main__.A object at 0x211144C50
```

```
class A:
    def __str__(self):
        return "Hi"
a = A()
str(a)
```

o/p Hi

Operator Overloading

operators can work with user defined class/objects call operator overloading

if o is object and i is integer o+i will result in

```
class A:
    Pass
o = A()
i = 5
o+i
```

Error

if we overload the operator + using the magic method __add__

```
class A:
    def __add__(self, x):
        Pass
o = A()
i = 5
o+i
No Error
```

i+o to perform this we method __radd__ reverse add

```
class A:
    def __radd__(self, x):
        Pass
o = A()
i = 5
```

+	__add__	__radd__
-	__sub__	__rsub__
*	__mul__	__rmul__
//	__floordiv__	__rfloordiv__
/	__truediv__	__rtruediv__
%	__mod__	__rmod__
divmod()	__divmod__	__rdivmod__
** pow()	__pow__	__rpow__

Unary operators

-	__neg__
+	__pos__
abs()	__abs__

Type conversion

int()	__int__	complex()	__complex__
float()	__float__	str()	__str__
bool()	__bool__	bytes()	__bytes__

index(), bin(), oct(), hex() __index__

Comparison operators

<	__lt__	>=	__ge__
>	__gt__	==	__eq__
<=	__le__	!=	__ne__

Assignment

+=	__iadd__	** =	__ipow__
-=	__isub__	& =	__iand__
* =	__imul__	| =	__ior__
// =	__ifloordiv__	^ =	__ixor__
/ =	__itruediv__	<< =	__ilshift__
% =	__imod__	>> =	__irshift__