Binary files
Binary file is a collection of bytes
In binary file we can write only bytes data

Example: Images, Audio, Video, …

In order perform operation on binary files, the file must be opened in binary mode.

wb → write bytes/binary
rb → read bytes/binary
ab → append bytes/binary

Example:
```
# Creating binary file
fobj=open("file1","wb")
b=bytes([65,66,67,68,69])
fobj.write(b)
print("data is written")
fobj.close()
```

Output
data is written

Example:
```
fobj=open("file1","rb")
data=fobj.read()
print(data)
for x in data:
    print(x)
fobj.close()
```

Output
```
b'ABCDE'
65
66
67
68
69
```

Example:

```
# Write a program to create copy of image

fobj1=open("a1.jpg","rb")
fobj2=open("a2.jpg","wb")

data=fobj1.read()
fobj2.write(data)

print("file copied...")
fobj1.close()
fobj2.close()
```

Output

```
file copied...
```

Pickling and Un pickling (pickle module) (OR) Serialization and De-Serialization

Pickling is process of converting object type into bytes.
Unpickling is process of converting bytes into object type.

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy. Pickling (and unpickling) is alternatively known as “serialization”, “marshalling,” or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

This process can be done using “pickle” module.
Pickle modules provides the following functions

1. dump()
2. dumps()
3. load()
4. loads()

Example of pickling

```
import pickle

fobj=open("file2.ser","wb")
pickle.dump(65,fobj)
pickle.dump(1.5,fobj)
pickle.dump(1+2j,fobj)
pickle.dump(True,fobj)
pickle.dump([10,20,30],fobj)

print("data is saved")
fobj.close()
```

Output

```
data is saved
```

Example:

```
import pickle

fobj=open("file2.ser","rb")
a=pickle.load(fobj)
b=pickle.load(fobj)
c=pickle.load(fobj)
d=pickle.load(fobj)
e=pickle.load(fobj)

print(a,b,c,d,e,sep="\n")

fobj.close()
```

Output

```
65
1.5
(1+2j)
True
[10, 20, 30]
```

Pickling and Un-Pickling User defined Data type or Custom objects

Emp.py
class Employee: def __init__(self):

```
        self.__empno=None
        self.__ename=None
        self.__salary=None
    def setData(self,e,en,s):
        self.__empno=e
        self.__ename=en
        self.__salary=s
    def printData(self):
        print(f'{self.__empno},{self.__ename},{self.__salary}')
```

Program for pickling

```
import pickle

fobj=open("file2.ser","rb")
a=pickle.load(fobj)
b=pickle.load(fobj)
c=pickle.load(fobj)
d=pickle.load(fobj)
e=pickle.load(fobj)

print(a,b,c,d,e,sep="\n")

fobj.close()
```

Output

```
EmployeeNo 1
EmployeeName naresh
Salary 45000
Add another employee?yes
EmployeeNo 2
EmployeeName suresh
Salary 65000
Add another employee?yes
EmployeeNo 3
EmployeeName kishore
Salary 56000
Add another employee?no
```

Program to unpickling

```
import pickle
import emp
fobj=open("emp.ser","rb")
```

```
emp1=pickle.load(fobj)
emp2=pickle.load(fobj)
emp3=pickle.load(fobj)

emp1.printData()
emp2.printData()
emp3.printData()

fobj.close()
```

Output

```
1,naresh,45000.0
2,suresh,65000.0
3,kishore,56000.0
```

Pickling and Un Pickling without writing inside file

```
import pickle

a=1.5
b=pickle.dumps(a)
print(a,b,type(a),type(b))
c=pickle.loads(b)
print(c,type(c))
c1=1+2j
b1=pickle.dumps(c1)
print(c1,type(c1))
print(b1,type(b1))
c2=pickle.loads(b1)
print(c2,type(c2))
```

Output

```
1.5
b'\x80\x04\x95\n\x00\x00\x00\x00\x00\x00\x00G?\xf8\x00\x00\x00\x00\x00\
x00.' <class 'float'> <class 'bytes'>
1.5 <class 'float'>
(1+2j) <class 'complex'>
b'\x80\x04\x95.\x00\x00\x00\x00\x00\x00\x00\x8c\x08builtins\x94\x8c\x07c
omplex\x94\x93\x94G?\xf0\x00\x00\x00\x00\x00\x00G@\x00\x00\x00\x00\x
00\x00\x00\x86\x94R\x94.' <class 'bytes'>
(1+2j) <class 'complex'>
```

Regular Expressions (re module)

What is regular expression?
A regular expression is special string which define search pattern or match pattern (OR) regular expression represents pattern which used for searching or matching within string.

Regular expressions, often shortened to "regex" or "regexp," are a powerful tool for pattern matching in text. They are sequences of characters that define a search pattern. Python's re module provides support for using regular expressions.

Pattern is described using special syntax or characters; these are classified into different categories

1. Character Literals
2. Meta Characters
3. Character classes
4. Quantifiers

Applications of regular expressions

1. Input validations
    a. User name validation
    b. Password validation
    c. Email validation
    d. Mobile No validation
    e. Domain name validation
    f. Date and time validation
2. Search Engine
3. Parsers
4. AI (Chat bots)

To work with regular expression python provides a default module called “re”. This module provides the following functions to work with regular expressions.

1. match()
2. search()
3. fullmatch()
4. findall()
5. split()

Syntax of creating regular expression

Regular expression is prefix with “r” or “R”.

Any string prefix with r is called raw string.

In Python, a raw string is a string literal that treats backslashes (`\`) as literal characters, rather than escape characters. This is achieved by prefixing the string with r or R.