readline(*size=-1*, /)
Read until newline or EOF and return a single str. If the stream is already at EOF, an empty string is returned.
If *size* is specified, at most *size* characters will be read.

Example:

```
try:
   fobj=open("e:\\student.txt","r")
   while True:
      stud=fobj.readline()
      if stud=='':
         break
      stud=stud.split()
      if int(stud[2])>=40 and int(stud[3])>=40:
         result="pass"
      else:
         result="fail"
      print(f'{stud[0]}\t{stud[1]}\t{stud[2]}\t{stud[3]}\t{result}')
except:
   print("error in reading")
```

Output

```
1       naresh      60      70      pass
2       suresh      80      90      pass
3       kishore     40      30      fail
4       ramesh      60      34      fail
5       kiran 70    80      pass
```

Random Access

seek(offset, whence=SEEK_SET, /)
Change the stream position to the given offset. Behaviour depends on the whence parameter. The default value for whence is SEEK_SET.

SEEK_SET or 0: seek from the start of the stream (the default); offset must either be a number returned

by TextIOBase.tell(), or zero. Any other offset value produces undefined behaviour.

SEEK_CUR or 1: “seek” to the current position; offset must be zero, which is a no-operation (all other values are unsupported).

SEEK_END or 2: seek to the end of the stream; offset must be zero (all other values are unsupported).

tell()
Return the current stream position as an opaque number. The number does not usually represent a number of bytes in the underlying binary storage.

Example:

```
# Random Accessing
fobj=open("e:\\file1.txt","r")
print(fobj.tell())
print(fobj.read(1))
print(fobj.tell())
print(fobj.read(1))
fobj.seek(5)
print(fobj.read(1))
fobj.seek(10)
print(fobj.read(3))
```

Output

```
0
J
1
y
n
656
```

Example:

```
# Updating file
```

```
fobj=open("e:\\file1.txt","r+")
fobj.write('J')
fobj.seek(5)
fobj.write('Y')
fobj.close()
```

Output
Update the content of file

Note: r+ mode is used for update (without truncating the file)

CSV file (csv module)

The so-called CSV (Comma Separated Values) format is the most common import and export format for spreadsheets and databases.

The csv module implements classes to read and write tabular data in CSV format. It allows programmers to say, "write this data in the format preferred by Excel," or "read data from this file which was generated by Excel," without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.

CSV file is text file.

The csv module's reader and writer objects read and write sequences. Programmers can also read and write data in dictionary form using the DictReader and DictWriter classes.

csv.writer(csvfile)
Return a writer object responsible for converting the user's data into delimited strings on the given file-like object. csvfile can be any object with a write() method. If csvfile is a file object, it should be opened with newline=''

```
import csv

fobj=open("e:\\emp.csv","w",newline='')
cw=csv.writer(fobj)
while True:
    empno=int(input("EmployeeNo :"))
    ename=input("EmployeeName :")
    job=input("Job :")
    sal=float(input("Salary :"))
    row=[empno,ename,job,sal]
    cw.writerow(row)
    ans=input("Add another employee?")
    if ans=="no":
        break
fobj.close()
```

Output

```
EmployeeNo :1
EmployeeName :naresh
Job :hr
Salary :50000
Add another employee?yes
EmployeeNo :2
EmployeeName :suresh
Job :acc
Salary :4000
Add another employee?yes
EmployeeNo :3
EmployeeName :kishore
Job :clerk
Salary :6000
Add another employee?no
```

csv.reader(csvfile)
Return a reader object that will process lines from the given csvfile. A csvfile must be an iterable of strings, each in the reader's defined csv format. A csvfile is most commonly a file-like object or list. If csvfile is a file object, it should be opened with newline=''

Example:

```
# Reading data from csv file

import csv

fobj=open("e:\\employee.csv","r",newline='')
cr=csv.reader(fobj)
for row in cr:
    print(row)

fobj.close()
fobj=open("e:\\employee.csv","r",newline='')
cr=csv.reader(fobj)
employee_list=list(cr)
print(employee_list)
```

Output

```
['empno', 'ename', 'salary']
['1', 'aaa', '5000']
['2', 'bbb', '6000']
['3', 'ccc', '8000']
['4', 'ddd', '9000']
['5', 'eee', '4500']
[['empno', 'ename', 'salary'], ['1', 'aaa', '5000'], ['2', 'bbb', '6000'], ['3', 'ccc', '8000'], ['4', 'ddd', '9000'], ['5', 'eee', '4500']]
```

DictWriter

csv.DictWriter(f, fieldnames)
Create an object which operates like a regular writer but maps dictionaries onto output rows.

Example:

```
# Creating csv file

import csv

fobj=open("e:\\stud.csv","w",newline='')
dw=csv.DictWriter(fobj,fieldnames=["rollno","name"])
dw.writeheader()
while True:
    rollno=int(input("Rollno: "))
    name=input("Name :")
    data={'rollno':rollno,'name':name}
    dw.writerow(data)
    ans=input("Add another student?")
    if ans=="no":
        break

fobj.close()
```

Output

```
Rollno: 1
Name :naresh
Add another student?yes
Rollno: 2
Name :suresh
Add another student?yes
Rollno: 3
Name :kishore
Add another student?no
```

csv.DictReader(f, fieldnames=None)

Create an object that operates like a regular reader but maps the information in each row to a dict whose keys are given by the optional fieldnames parameter.

Example:

```
import csv

fobj=open("e:\\stud.csv","r",newline='')
dr=csv.DictReader(fobj)
for row in dr:
    print(row)
```

Output

```
{'rollno': '1', 'name': 'naresh'}
{'rollno': '2', 'name': 'suresh'}
{'rollno': '3', 'name': 'kishore'}
```

JSON (json module)