Python File Handling

Theory

Types of data used for I/O:

- Text - '12345' as a sequence of unicode chars
- Binary - 12345 as a sequence of bytes of its binary equivalent

Hence there are 2 file types to deal with

- Text files - All program files are text files
- Binary Files - Images,music,video,exe files

How File I/O is done in most programming languages

- Open a file
- Read/Write data
- Close the file

Writing to a file

case 1 : file is not present

In [14]:
```
f=open('sample1.txt','w')
f.write('I am writing in a file which not exist earlier')
f.close()
```

In [27]:
```
# writing multiple lines in a file
f=open('sample1.txt','w')
```

```
f.write("this is line 1\n")
f.write("this is line 2")
f.close()
```

Case 2 : file already present

In [33]:
```
f=open('sample2.txt','w')
f.write('this file was already present')
f.close()
```

How exactly open() works ?

When we do f.open() it loads the file from ROM to RAM and from RAM file move to buffer from where it get read char by char.

- Problem with 'w' mode - it clear the previous content from file if exist

Append mode

In [49]:
```
f=open('sample2.txt','a')
f.write("\nthis line going to be append in sample2.txt")
f.close()
```

Writing multiple line from list

In [53]:
```
L = ['hello\n','hi\n','how are you\n','I am fine']
f=open('sample3.txt','w')
f.writelines(L)
f.close()
```

Reading from a file

In [61]:
```
f=open('sample3.txt','r')
content=f.read()
```

```
print(content)
f.close()
```

```
hello
hi
how are you
I am fine
```

In [63]:
```
# reading upto n character
f=open('sample3.txt','r')
content=f.read(10)
print(content)
f.close()
```

```
hello
hi
h
```

reading line by line

In [70]:
```
f=open('sample3.txt','r')
print(f.readline(),end='')
print(f.readline(),end='')
f.close()
```

```
hello
hi
```

reading whole line by line

In [73]:
```
f=open('sample3.txt','r')
while True:
    data=f.readline()
    if data=='':
        break
    print(data,end='')
f.close()
```

```
hello
hi
how are you
I am fine
```

Using Context Manager (With)

- It's a good idea to close a file after usage as it will free up the resources
- If we dont close it, garbage collector would close it
- with keyword closes the file as soon as the usage is over

writing using with

In [108…
```
with open('sample4.txt','w') as f:
    f.write("this is not exist earlier, created using with")
```

In [110…
```
with open('sample4.txt','a') as f:
    f.write("\nthis is next line")
```

reading using read()

In [118…
```
with open('sample4.txt','r') as f:
    print(f.read())
```

```
this is not exist earlier, created using with
this is next line
```

In [120…
```
with open('sample4.txt','r') as f:
    print(f.read(6))
```

```
this i
```

reading using readline()

In [127…
```
with open('sample4.txt','r') as f:
    print(f.readline(),end='')
    print(f.readline())
```

```
this is not exist earlier, created using with
this is next line
```

moving n char by char

In [129…
```
with open('sample4.txt','r') as f:
    print(f.read(6))
    print(f.read(6))
```

```
this i
s not
```

Benefit of reading char by char is to load a big file easily

In [133…
```
big_L = ['hello world ' for i in range(1000)]

with open('big.txt','w') as f:
  f.writelines(big_L)
```

In [ ]:
```
with open('big.txt','r') as f:
    chunk=10
    while True:
        content=f.read(chunk)
        if len(content)==0:
            break
        print(content)
```

Seek and Tell function

- tell() - it tells how much char we proceed and what next
- seek() - helps to move cursor anywhere in content

Using seek and tell during read

In [152…
```
with open('sample4.txt','r') as f:
    f.seek(15)
    print(f.read(10))
    print(f.tell())
    print(f.read(10))
    print(f.tell())
```

```
st earlier
25
, created
35
```

Using seek and tell during write

In [156…
```
with open('sample5.txt','w') as f:
  f.write('Hello')
  f.seek(0)
  f.write('Xa')
```

Problem with working in text mode

- can't work with binary files like images
- not good for other data types like int/float/list/tuples

In [ ]:
```
# working with binary file
with open('screenshot1.png','r') as f:
  f.read()
# will not work because utf-8 codec cannot be decode by 0x89
```

Working with binary files

Working with images

In [171…
```
# copy a image
with open('backiee-122217-landscape.jpg','rb') as f:
    with open('copy.jpg','wb') as wf:
        wf.write(f.read())
```

Working with other data types

In [ ]:
```
with open('sample4.txt','w'):
    f.write(12)
# it will not work with other data type like int,float,list,tuple etc.
# even if we store a dict in a string format then it can store it but cannot revert back to dict
```

In [183…
```
with open('sample6.txt','w') as f:
  f.write('5')
```

In [186…
```
with open('sample6.txt','r') as f:
  print(int(f.read()) + 5)
```

```
10
```

In [188…
```
# more complex data
d = {
    'name':'nitish',
     'age':33,
     'gender':'male'
}

with open('sample7.txt','w') as f:
  f.write(str(d))
```

In [193…
```
with open('sample7.txt','r') as f:
  print(type(f.read()))
```

```
<class 'str'>
```

Serialization and Deserialization

- **Serialization** - process of converting python data types to JSON format
- **Deserialization** - process of converting JSON to python data types

What is JSON?

```json
{
    "d": {
        "results": [
            {
                "__metadata": {
                    "type": "EmployeeDetails.Employee"
                },
                "UserID": "E12012",
                "RoleCode": "35"
            }
        ]
    }
}
```

Serilization using json module

In [199…
```python
import json
L=[1,2,3,4]
with open('demo1.json','w') as f:
    json.dump(L,f)
```

In [202…
```python
# dict
d = {
    'name':'nitish',
     'age':33,
     'gender':'male'
}

with open('demo2.json','w') as f:
  json.dump(d,f,indent=4)
```

Deserialization

In [206…
```python
# deserialization
import json

with open('demo2.json','r') as f:
  d = json.load(f)
  print(d)
  print(type(d))
```

```
{'name': 'nitish', 'age': 33, 'gender': 'male'}
<class 'dict'>
```

In [208…
```python
import json
with open('demo1.json','r') as f:
    l=json.load(f)
    print(l,type(l))
```

```
[1, 2, 3, 4] <class 'list'>
```

In [210…
```python
# serialize and deserialize tuple
import json

t = (1,2,3,4,5)

with open('demo3.json','w') as f:
  json.dump(t,f)
```

In [213…
```python
# serialize and deserialize a nested dict

d = {
    'student':'nitish',
     'marks':[23,14,34,45,56]
}

with open('demo4.json','w') as f:
  json.dump(d,f)
```

Serialization and Deserialization of custome objects

In [218…
```
class Person:

  def __init__(self,fname,lname,age,gender):
    self.fname = fname
    self.lname = lname
    self.age = age
    self.gender = gender
```

In [220…
```
person=Person('Abhishek','Keshri',26,'male')
```

In [234…
```
import json

def show_object(person):
    if isinstance(person,Person):
        return {'name':person.fname+' '+person.lname,'age':person.age,'gender':person.gender}

with open('demo5.json','w') as f:
    json.dump(person,f,default=show_object,indent=4)
```

In [236…
```
# deserializing
import json

with open('demo5.json','r') as f:
  d = json.load(f)
  print(d)
  print(type(d))
```

```
{'name': 'Abhishek Keshri', 'age': 26, 'gender': 'male'}
<class 'dict'>
```

Pickling

`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.

In [241…
```
class Person:
```

```
  def __init__(self,name,age):
    self.name = name
    self.age = age

  def display_info(self):
    print('Hi my name is',self.name,'and I am ',self.age,'years old')
```

In [243…
```
p = Person('nitish',33)
```

In [245…
```
# pickle dump
import pickle
with open('person.pkl','wb') as f:
  pickle.dump(p,f)
```

In [247…
```
# pickle load
import pickle
with open('person.pkl','rb') as f:
  p = pickle.load(f)

p.display_info()
```

```
Hi my name is nitish and I am  33 years old
```

Pickle Vs Json

- Pickle lets the user to store data in binary format. JSON lets the user store data in a human-readable text format.