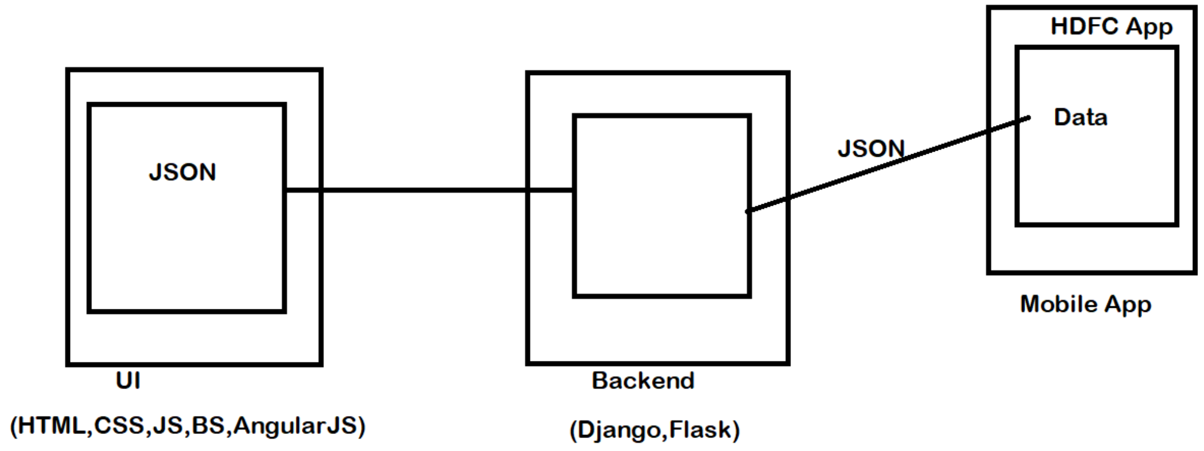
JSON file

JSON stands for Java Script Object Notation.
JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

To work with json file , python provides a predefined module called "json".

Json module provides encoders and decoders.



In json data is written as key and value pair.

Python	JSON
dict	object

Python	**JSON**
list, tuple	array
str	string
int, float, int- & float-derived Enums	number
True	true
False	false
None	null

Json module provides the following methods

1. dump Write object into file
2. load Read object from file
3. dumps Convert python object json string (not write inside file)
4. loads Convert json string into python object (not write inside file)

```
# Writing python data into json format
import json
emp_data={'empno':[1,2,3],
        'ename':['naresh','ramesh','kishore'],
        'salary':[1000,2000,3000]}

emp_str=json.dumps(emp_data)
print(type(emp_data))
print(type(emp_str))

print(emp_data)
print(emp_str)
```

```
with open("emp.json","w") as f:
    json.dump(emp_data,f)
    print("data saved inside file")
```

Output

```
<class 'dict'>
<class 'str'>
{'empno': [1, 2, 3], 'ename': ['naresh', 'ramesh', 'kishore'], 'salary':
[1000, 2000, 3000]}
{"empno": [1, 2, 3], "ename": ["naresh", "ramesh", "kishore"], "salary":
[1000, 2000, 3000]}
data saved inside file
```

Example:

```
# Reading json string and converting into python type
import json
emp_data={'empno':[1,2,3],
          'ename':['naresh','ramesh','kishore'],
          'salary':[1000,2000,3000]}

emp_str=json.dumps(emp_data)
print(emp_data,emp_str,sep="\n")
print(type(emp_data),type(emp_str))
emp_data1=json.loads(emp_str)
print(emp_data1)
print(type(emp_data1))

with open("emp.json","r") as f:
    emp_data2=json.load(f)
    print(emp_data2)
    print(type(emp_data2))
```

Output

```
{'empno': [1, 2, 3], 'ename': ['naresh', 'ramesh', 'kishore'], 'salary':
[1000, 2000, 3000]}
{"empno": [1, 2, 3], "ename": ["naresh", "ramesh", "kishore"], "salary":
[1000, 2000, 3000]}
<class 'dict'> <class 'str'>
{'empno': [1, 2, 3], 'ename': ['naresh', 'ramesh', 'kishore'], 'salary':
[1000, 2000, 3000]}
<class 'dict'>
{'empno': [1, 2, 3], 'ename': ['naresh', 'ramesh', 'kishore'], 'salary':
[1000, 2000, 3000]}
<class 'dict'>
```

Binary files

Binary file is collection of bytes.
A byte is an integer value which range from 0-255.
Bytes object represent integer value in byte. This representation is done using encoding standard (ASCII UTF-8).

Example: images, audio, video

File must be opened with suffix wb Write binary, rb read binary

Methods used for reading and writing binary data

1. write
2. read

Example:

```
# Creating binary file

with open("file1","wb") as f:
    b1=bytes([65,66,67,68,69,70])
    f.write(b1)

print("Data Saved inside file1")
```

Output

```
Data Saved inside file1
```

Example:

```
# Reading data from binary file

with open("file1","rb") as f:
    b=f.read()
    print(b)
    print(b[0],b[1],b[2],b[4])
```

Output

```
b'ABCDEF'
65 66 67 69
```

Example:

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

with open("e:\\django.png","rb") as f1:
    f2=open("e:\\django2.jpg","wb")
    b=f1.read()
    print(b)
    f2.write(b)
    f2.close()

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

Output

```
data is saved...
```

Pickle module

“pickle” is a default module which comes with python software. 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,” [1] or “flattening”; however, to avoid confusion, the terms used here are “pickling” and “unpickling”.

1. dump
2. load

Example of pickling

```
import pickle

with open("file2.ser","wb") as f:
    pickle.dump(65,f)
    pickle.dump(1.5,f)
    pickle.dump(1+2j,f)
    pickle.dump(True,f)

print("Data Saved...")
```

Output

```
Data Saved...
```

Example of unpickling

```
import pickle

with open("file2.ser","rb") as f:
    obj1=pickle.load(f)
    obj2=pickle.load(f)
    obj3=pickle.load(f)
    obj4=pickle.load(f)
    print(obj1,obj2,obj3,obj4,sep="\n")
```

Output

```
65
1.5
(1+2j)
True
```

Pickling and Unpicking custom objects

Emp.py

```
class Employee:
    def __init__(self,eno,en):
        self.__eno=eno
        self.__en=en
    def __str__(self):
        return f'{self.__eno},{self.__en}'
```

Filetest22.py (Pickling)

```
import emp
import pickle

with open("employee.ser","wb") as f:
    emp1=emp.Employee(101,"naresh")
    emp2=emp.Employee(102,"suresh")
    pickle.dump(emp1,f)
    pickle.dump(emp2,f)

print("employee objects are saved")
```

Filetest23.py (Unpickling)

```
import emp
import pickle

with open("employee.ser","rb") as f:
    emp1=pickle.load(f)
    emp2=pickle.load(f)
    print(emp1,emp2,sep="\n")
```