JSON is a syntax for storing and exchanging data. JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange.

JSON is text, written with JavaScript object notation. (JSON) is a standardized format commonly used to transfer data as text that can be sent over a network. It’s used by lots of APIs and Databases, and it’s easy for both humans and machines to read. JSON represents objects as name/value pairs, just like a Python dictionary.

Python has a built-in package called json, which can be used to work with JSON data.

The json library can parse JSON from either strings or files. The library parses JSON into a Python dictionary or list. It can also convert Python dictionaries or lists into JSON strings.

**Parsing JSON - Covert JSON Strings to Python Object**

One of the most common task which we perform on JSON is to convert it to Python object. json library provides loads function to achieve it.

*# take a JSON string (json\_string) and convert it to Python object (parsed\_json).*

​

*#loads - will convert json object to python object*

*#dumps - will convert python object to json object*

​

**import** json

​

json\_string **=** '{"name":"Asha","id":123}'

print(type(json\_string))

​

pyt\_obj **=** json.loads(json\_string) *#for pickling we used load and for json we need to use loads method*

print(pyt\_obj) *#normal string is converted to json string which will be of dictionary type*

print(type(pyt\_obj))

print(pyt\_obj["name"]) *#get the values through keys*

print(pyt\_obj["id"])

​

**for** key,values **in** pyt\_obj.items(): *#Can iterate by loops and access using items(),keys()or values() method*

print(key ,values)

<class 'str'>

{'name': 'Asha', 'id': 123}

<class 'dict'>

Asha

123

name Asha

id 123

*# JSON string with another data type (list)*

**import** json

json\_string **=** '["firstname","Asha",{"lastname":"Talari"}]' *#string consisting of list embeded with dictionary items*

print(type(json\_string))

pyt\_obj **=** json.loads(json\_string)

print(type(pyt\_obj),pyt\_obj)

​

print(pyt\_obj[2]['lastname'])

*# print(json\_string[2]) #Can get the value of the dictionary through index*

**for** key,val **in** pyt\_obj[2].items():

print(key,val)

​

<class 'str'>

<class 'list'> ['firstname', 'Asha', {'lastname': 'Talari'}]

Talari

lastname Talari

**with** open("jsoncovt.json","w") **as** filehand: *#python object to Json*

json.dumps(filehand.write(json\_string)) *#converted to json object and stored in json file*

print(type(json\_string)) *#json string /object*

<class 'str'>

**with** open("jsoncovt.json","r") **as** filehand:

pyth\_obj **=** json.loads(filehand.read()) *#convert json object /string in to python object*

print(pyth\_obj)

print(type(pyth\_obj)) *#python object*

['firstname', 'Asha', {'lastname': 'Talari'}]

<class 'list'>

Following is the table to understand the relationship between Python and JSON Data and can be useful to understand

the state of data after serialization or deserialization.

​

Python                   JSON

​

dict                     object

list, tuple               array

None                     null

False                     false

True                     true

int, float               number

str                       string

De-serialization - Python object to JSON string

It is the process of converting python objects to JSON objects/string. It can be achieved by using json.dumps.

​

**import** json

normal\_dict **=** {'firstname':"asha",

'lastname':'talari',

'title':['mr','mrs']

}

​

*#with out any quatation so it is python object*

print(type(normal\_dict),normal\_dict)

json\_obj **=** json.dumps(normal\_dict) *#converted to json string object*

print(json\_obj)

print(type(json\_obj))

​

​

<class 'dict'> {'firstname': 'asha', 'lastname': 'talari', 'title': ['mr', 'mrs']}

{"firstname": "asha", "lastname": "talari", "title": ["mr", "mrs"]}

<class 'str'>

normal\_list **=** ['Asha','Talari']

​

print(type(normal\_list))

print(normal\_list) *#normal python object has single quotes*

json\_array **=** json.dumps(normal\_list)

print(type(json\_array))

print(json\_array) *#json object will have double quotes*

<class 'list'>

['Asha', 'Talari']

<class 'str'>

["Asha", "Talari"]

JSON to Python object conversion follows the following table

​

JSON           Python

object         dict

array           list

string         str

number (int)   int

number (real)   float

true           True

false           False

null           None

In all the above example, if we save the JSON string to a file then entire content is stored in a single line and will not look pretty. json.dumps provides additional methods to achieve it. In the following examples we are going to learn about them

indent

indent is an argument of dumps which provide proper indents to the JSON elements as shown in the below examples

**import** json

​

normal\_lst **=** ["Asha",'Talari','Dheeraj']

print(type(normal\_lst))

​

json\_string **=** json.dumps(normal\_lst,indent **=** 4) *#with 4 indentation*

print(json\_string)

print(type(json\_string))

<class 'list'>

[

"Asha",

"Talari",

"Dheeraj"

]

<class 'str'>

sort\_keys

sort\_keys when set to True sorts the Python object dictionary based on keys while converting them to JSON as shown in the below example

normal\_dict **=** {"firstname":"Asha","middlename":'Talari',"lastname":'Dheeraj'} *#sorted as per keys*

print(type(normal\_dict))

​

json\_string **=** json.dumps(normal\_dict,sort\_keys**=** **True**,indent **=** 4) *#with 4 indentation*

print(json\_string)

print(type(json\_string))

<class 'dict'>

{

"firstname": "Asha",

"lastname": "Dheeraj",

"middlename": "Talari"

}

<class 'str'>

*# Lets take an example of a Python dictionary which has key as tuple, which is a valid*

normal\_dict **=** {("Asha","Dheeraj"):["Admin","HR"],"Mahe":"HR"}

print(normal\_dict)

​

print(normal\_dict['Mahe']) *#this will give correct value where as*

print(normal\_dict[('Asha','Dheeraj')]) *#will give key error*

print(type(normal\_dict))

​

json\_str **=** json.dumps(normal\_dict,indent **=** 4)

print(json\_str)

print(type(json\_str))

​

*#we will be getting error while converting because keys can not be tuple as keys as for one single key there can be multiple values so if we want*

*#to skip this tuple then we can use skipkeys*

​

{('Asha', 'Dheeraj'): ['Admin', 'HR'], 'Mahe': 'HR'}

HR

['Admin', 'HR']

<class 'dict'>

**---------------------------------------------------------------------------**

**TypeError** Traceback (most recent call last)

**<ipython-input-2-8eb55ab4fde8>** in <module>

7 print**(**type**(**normal\_dict**))**

8

**----> 9** json\_str **=** json**.**dumps**(**normal\_dict**,**indent **=** **4)**

10 print**(**json\_str**)**

11 print**(**type**(**json\_str**))**

**c:\users\asha.t\appdata\local\programs\python\python37-32\lib\json\\_\_init\_\_.py** in dumps**(obj, skipkeys, ensure\_ascii, check\_circular, allow\_nan, cls, indent, separators, default, sort\_keys, \*\*kw)**

236 check\_circular**=**check\_circular**,** allow\_nan**=**allow\_nan**,** indent**=**indent**,**

237 separators**=**separators**,** default**=**default**,** sort\_keys**=**sort\_keys**,**

**--> 238 \*\*kw).encode(obj)**

239

240

**c:\users\asha.t\appdata\local\programs\python\python37-32\lib\json\encoder.py** in encode**(self, o)**

199 chunks **=** self**.**iterencode**(**o**,** \_one\_shot**=True)**

200 **if** **not** isinstance**(**chunks**,** **(**list**,** tuple**)):**

**--> 201** chunks **=** list**(**chunks**)**

202 **return** **''.**join**(**chunks**)**

203

**c:\users\asha.t\appdata\local\programs\python\python37-32\lib\json\encoder.py** in \_iterencode**(o, \_current\_indent\_level)**

429 **yield** **from** \_iterencode\_list**(**o**,** \_current\_indent\_level**)**

430 **elif** isinstance**(**o**,** dict**):**

**--> 431 yield** **from** \_iterencode\_dict**(**o**,** \_current\_indent\_level**)**

432 **else:**

433 **if** markers **is** **not** **None:**

**c:\users\asha.t\appdata\local\programs\python\python37-32\lib\json\encoder.py** in \_iterencode\_dict**(dct, \_current\_indent\_level)**

374 **continue**

375 **else:**

**--> 376 raise TypeError(f'keys must be str, int, float, bool or None, '**

377 f'not {key.\_\_class\_\_.\_\_name\_\_}')

378 **if** first**:**

**TypeError**: keys must be str, int, float, bool or None, not tuple

normal\_dict **=** {("Asha","Dheeraj"):["Admin","HR"],"Mahe":"HR"}

*# x = {("Asha"):["Admin","HR"]}*

*# print(x)*

print(type(normal\_dict))

json\_str **=**" "

**try**:

json\_str **=** json.dumps(normal\_dict,skipkeys **=** **True**,indent **=** 4) *#The trouble causing keys were been skipped*

print(json\_str)

**except** Exception **as** e:

print(e)

{'Asha': ['Admin']}

<class 'dict'>

{

"Mahe": "HR"

}

*#Few more examples*

**import** json

student **=** {"103":{"class":'V', "Name":'Rohit', "Roll\_no":7},

"102":{"class":'V', "Name":'David', "Roll\_no":8},

"101":{"class":'V', "Name":'Samiya', "Roll\_no":12}}

​

json\_str **=** json.dumps(student,sort\_keys **=True**,indent**=**2) *#with out indentation it will display in a straight line*

print(json\_str)

​

{

"101": {

"Name": "Samiya",

"Roll\_no": 12,

"class": "V"

},

"102": {

"Name": "David",

"Roll\_no": 8,

"class": "V"

},

"103": {

"Name": "Rohit",

"Roll\_no": 7,

"class": "V"

}

}

**import** json

normal\_x **=** "red","blue","green"

print(normal\_x)

​

print(type(normal\_x))

​

json\_str **=**json.dumps(normal\_x)

print(type(json\_str))

print(json\_str)

('red', 'blue', 'green')

<class 'tuple'>

<class 'str'>

["red", "blue", "green"]

**import** json

​

normal\_lst **=**[2,3,4,5]

y**=**1

**for** x **in** normal\_lst:

y **=** y **\***x

print(y)

​

print(type(normal\_lst))

json\_str **=** json.dumps(normal\_lst, separators**=**(". "," ")) *#the default separator is , and this can be replaced by any sep*

*# #between the numbers spacing can also be mentioned*

print(json\_str)

​

json\_str**=** json.dumps(normal\_lst) *#With indent it will provide indent so ti will be in sinle line*

print(type(json\_str))

print(json\_str)

​

y**=**1

**for** x **in** json\_str: *#if we do like this on str object , it will error*

y **=** y**\***x

​

print(y)

​

​

120

<class 'list'>

[2. 3. 4. 5]

<class 'str'>

[2, 3, 4, 5]

1[2, 3, 4, 5]

**import** json

​

nor\_bool **=** **True**

print(type(nor\_bool))

print(nor\_bool)

​

json\_bool **=** json.dumps(nor\_bool)

print(json\_bool)

print(type(json\_bool))

​

**if** json\_bool: *#true*

print("its value is true")

**else**:

print("its value is not true")

<class 'bool'>

True

true

<class 'str'>

its value is not true

*# Single quotes are not allowed inside the JSON string to denote string*

​

**import** json

json\_data **=** """{"103": {"class": "V", "Name": "Samiya", "Roll\_n": 12},

"102": {"class": "V", "Name": "David", "Roll\_no": 8},

"101": {"class": "V", "Name": "Rohit", "Roll\_no": 7}}"""

**try**:

js **=** json.loads(json\_data)

**except** Exception **as** je:

print(je)

*#solution is either use string method replace to relace single quotes or use literal\_eval from ast module*

**import** ast

**import** json

​

data **=** json.dumps(ast.literal\_eval(json\_data),indent **=** 4) *#will evalaute single quotes to double*

print(data)

{

"103": {

"class": "V",

"Name": "Samiya",

"Roll\_n": 12

},

"102": {

"class": "V",

"Name": "David",

"Roll\_no": 8

},

"101": {

"class": "V",

"Name": "Rohit",

"Roll\_no": 7

}

}