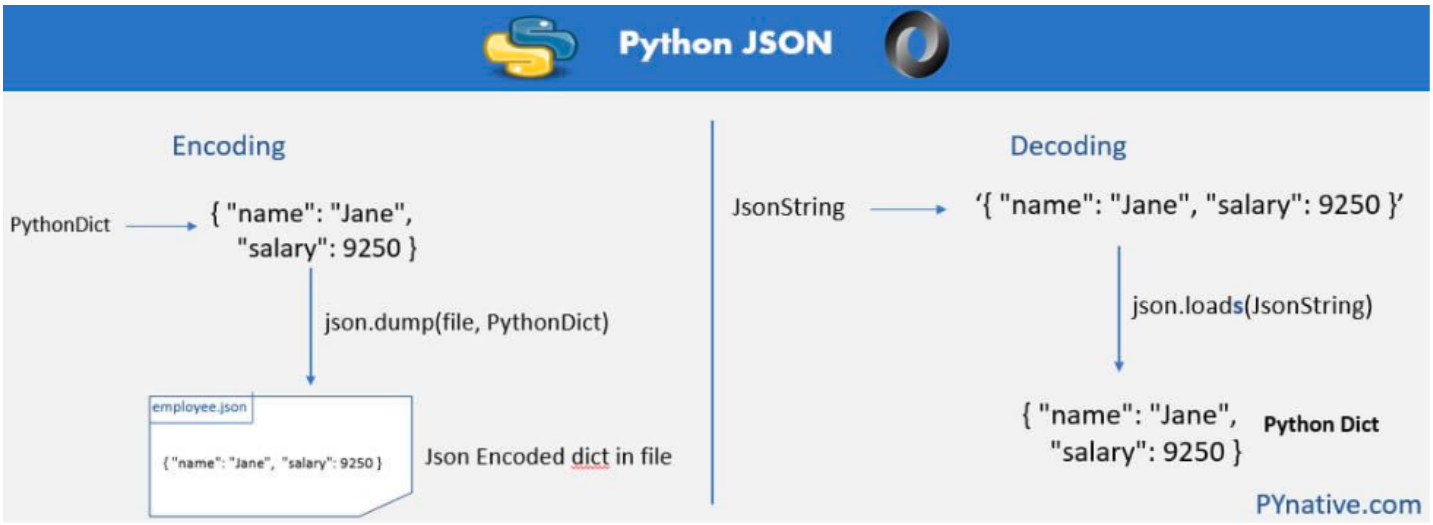
Python JSON Concept:

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

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



Python JSON Tutorial

Mapping between JSON and Python entities while Encoding

Now let's see how to convert all Python primitive types such as a `dict`, `list`, `set`, `tuple`, `str`, numbers into JSON formatted data. Please refer to the following table to know the mapping between JSON and Python data types.

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

Mapping between JSON and Python data types

Example:

Import the json module.

```
import  json
```

Parse JSON - Convert from JSON to Python:

If you have a JSON string, you can parse it by using the json.loads() method.

Example: Convert from JSON to Python:

```
import json
# some JSON:
x =  '{ "name" : "John" , "age" : 30 , "city" : "New York"}'
# parse x:
y = json.loads(x)
# the result is a Python dictionary:
print(y["age"])
```

Output:

30

Convert from Python to JSON

If you have a Python object, you can convert it into a JSON string by using the json.dumps() method.

Example: Convert from Python to JSON:

```
import json
# a Python object (dict):
x = {
  "name": "John",
  "age": 30,
  "city": "New York"
}
# convert into JSON:
y = json.dumps(x)
# the result is a JSON string:
print(y)
```

Output: {"name" : "John" , "age" : 30 , "city" : "New York"}

Example: Convert Python objects into JSON strings, and print the values.

```
import json
print(json.dumps({"name": "John", "age": 30}))
print(json.dumps(["apple", "bananas"]))
```

```
print(json.dumps(("apple", "bananas")))
print(json.dumps("hello"))
print(json.dumps(42))
print(json.dumps(31.76))
print(json.dumps(True))
print(json.dumps(False))
print(json.dumps(None))
```

Output:

```
{ "name" :  "John" ,  "age" :  30}
["apple", "bananas"]
["apple", "bananas"]
"hello"
42
31.76
true
false
null
```

Example: Convert a Python object containing all the legal data types:

```
import json
x = {
  "name" :  "John",
  "age" :  30,
  "married" :  True,
  "divorced" :  False,
  "children" :  ("Ann","Billy"),
  "pets" :  None,
  "cars" : [
    {"model" :  "BMW 230", "mpg": 27.5},
    {"model" :  "Ford Edge", "mpg": 24.1}
  ]
}
print(json.dumps(x))
```

Output: {"name": "John", "age": 30, "married": true, "divorced": false, "children": ["Ann","Billy"], "pets": null, "cars": [{"model": "BMW 230", "mpg": 27.5}, {"model": "Ford Edge", "mpg": 24.1}]}

Format the Result:

The example above prints a JSON string, but it is not very easy to read, with no indentations and line breaks.

The json.dumps() method has parameters to make it easier to read the result:

Example:

Use the **indent parameter** to define the numbers of indents:

For example, **json.dumps(x, indent=4)**

Code:

```
import json
x = {
  "name": "John",
  "age": 30,
  "married": True,
  "divorced": False,
  "children": ("Ann","Billy"),
  "pets": None,
  "cars": [
    {"model": "BMW 230", "mpg": 27.5},
    {"model": "Ford Edge", "mpg": 24.1}
  ]
}
# use four indents to make it easier to read the result:
print(json.dumps(x, indent=4))
```

Output:

```
{
    "name" :  "John",
    "age" :  30,
    "married" :  true,
    "divorced" :  false,
    "children" : [
        "Ann",
        "Billy"
    ],
    "pets": null,
    "cars": [
        {
            "model": "BMW 230",
            "mpg": 27.5
        },
        {
```

```
        "model": "Ford Edge",
        "mpg": 24.1
      }
   ]
}
```

Order the Result:

The json.dumps() method has parameters to order the keys in the result:

Example:

Use the sort_keys parameter to specify if the result should be sorted or not:

For example, json.dumps(x, indent=4, sort_keys=True)

Example:

```
import json
x = {
  "name": "John",
  "age": 30,
  "married": True,
  "divorced": False,
  "children": ("Ann","Billy"),
  "pets": None,
  "cars": [
   {"model": "BMW 230", "mpg": 27.5},
   {"model": "Ford Edge", "mpg": 24.1}
  ]
}
# sort the result alphabetically by keys:
print(json.dumps(x, indent=4, sort_keys=True))
```

Output:

```
{
   "age": 30,
   "cars": [
      {
         "model": "BMW 230",
         "mpg": 27.5
      },
      {
         "model": "Ford Edge",
         "mpg": 24.1
```

```
    }
  ],
  "children": [
    "Ann",
    "Billy"
  ],
  "divorced": false,
  "married": true,
  "name": "John",
  "pets": null
}
```


You can also define the separators, default value is (", ", ": "), which means using a comma and a space to separate each object, and a colon and a space to separate keys from values:

Example:

Use the separators parameter to change the default separator:

For example**, json.dumps(x, indent=4, separators=(". ", " = "))**