JSON

JSON is a syntax for storing and exchanging data.

JSON is text, written with JavaScript object notation.

## **JSON in Python**

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

### **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.

The result will be a **Python dictionary**.

### **Example**

Convert from JSON to Python:

|  |
| --- |
| **JSON01.py** |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9****10** | **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"]) |

## **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.

|  |
| --- |
| **JSON02.py** |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9****10****11****12****13****14** | **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) |

You can convert Python objects of the following types, into JSON strings:

* dict
* list
* tuple
* string
* int
* float
* True
* False
* None

### **Example**

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

|  |
| --- |
| **JSON03.py** |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9****10****11** | **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**)) |

When you convert from Python to JSON, Python objects are converted into the JSON (JavaScript) equivalent:

|  |  |
| --- | --- |
| **Python** | **JSON** |
| dict | Object |
| list | Array |
| tuple | Array |
| str | String |
| int | Number |
| float | Number |
| True | true |
| False | false |
| None | null |

Example

Convert a Python object containing all the legal data types:

|  |
| --- |
| **JSON04.py** |
| **Line** | **Code** |
| **1****2****3****4****5****6****7****8****9****10****11****12****13****14****15****16** | **import** jsonx = { "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)) |

## **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:

json.dumps(x, indent=4)

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:

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

**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:

json.dumps(x, indent=4, sort\_keys=**True**)