Massive Information & Knowledge Engineering

Python Dictionary & JSON

204113 Computer & Programming

Dr. Arnon Rungsawang
Dept. of computer engineering
Kasetsart University

https://mike.cpe.ku.ac.th/204113

Version 2024

Python Dictionary

Dictionary

- We use dictionary to find the meaning of an unfamiliar word or learning a language.
- In Python, a dictionary is a collection which is unordered, changeable and indexed.
- Each item in a dictionary refers to exactly one object.
  - Note that it is possible to map to an object that contains a set of other objects, like a set or list.
- In natural language dictionaries, in contrast, the word bat could either be
  - a flying mammal that lives in a cave or
  - a tool for playing baseball.

Creating a Python Dictionary

- There are several ways to create a Python dictionary, but the most common is to use a dictionary literal.
- A dictionary literal is a set of key/value pairs surrounded by curly brackets.

```
grade_map = {"A":90, "B":80, "C":70, "D":60}
minScore4A = grade_map["A"]
print(minScore4A)
```

  - In this example, each letter grade is a key.
  - Between each key and value, we have a colon, `:`.
  - Separating each key-value pair, we have either a comma or (to end the dictionary) a closing curly brace. More formally, we can write the syntax this way:

```
{key1: value1, …, keyN:valueN}
```

Look up and add item to a Python Dictionary

- To look up an item, we use the dictionary name, followed by the key name in square brackets.

```
grade_map = {"A":90, "B":80, "C":70, "D":60}
minScore4A = grade_map["A"]
print(minScore4A)
```

- We can add an item to Python dictionaries using an assignment operator and the same brackets operator we used to look up an item.

```
grade_map = {"A":90, "B":80, "C":70, "D":60}
grade_map["F"] = 0
print(grade_map)
```


Less common way to create a Dictionary

- We can also create a dictionary using the `dict()` constructor, and combine this with setting values individually:

```
good_grades = dict()
good_grades["A"] = 90
good_grades["B"] = 80
print(good_grades)
```

  - This way of creating dictionaries is much less common and is not recommended if we know the values of dictionary in advance.

Dealing with missing data

- When we accessing the item in a dictionary using a key, Python will raise a `KeyError` if that key is not in the dictionary.

```
grade_map = {"A":90, "B":80, "C":70, "D":60}
print(grade_map["G"])
```

```
Traceback (most recent call last):
  File "<string>", line 2, in <module>
KeyError: 'G'
```

- The way to avoid this error if you're unsure what keys the dictionary contains is to use the dictionary's `get()` method instead.
  - This method takes a key to retrieve, and instead of raising an error if the key is not in the dictionary, it returns None by default.
  - As an option, you can pass a second argument to the get method to return a different default.

```
print(grade_map.get("G"))
print(grade_map.get("G", 0))
print(grade_map.get("A", 0))
```

```
None
0
90
```


Iterating through a Dictionary

- The dictionary type has two methods: `keys()` and `values()`, which returns a list only of keys or only of values, respectively.
- Using each key in the `keys()` list, we can always look up the value and iterate the dictionary this way.

```
supPower = {"Superman": "strength", "Flash":\
            "speed", "Sue Storm": "invisiblity"}
for key in supPower.keys():
   print(f"{key}'s superpower is {supPower[key]}.")
```

```
Superman's superpower is strength.
Flash's superpower is speed.
Sue Storm's superpower is invisiblity.
```

Iterating through a Dictionary (2)

- Another way to do this is to use the dictionary's `items()` method. Here's what the `items()` method returns:

```
print(supPower.items())
```

```
dict_items([('Superman', 'strength'), ('Flash', 'speed'),
('Sue Storm', 'invisiblity')])
```

- The object looks like it contains a list of tuples, so this means we can unpack each item easily.

```
for key, value in supPower.items():
  print(f"{key}'s superpower is {value}.")
```

```
Superman's superpower is strength.
Flash's superpower is speed.
Sue Storm's superpower is invisiblity.
```


Usage of the built-in `get()` method

```
1  def count_words(text):
2    """returns a dictionary with the count
3       of each word in the text"""
4    counts = {}
5    text = text.lower()
6    words = text.split(" ")
7    for word in words:
8      if counts.get(word) == None:
9        counts[word] = 1
10     else:
11       counts[word] = counts[word] + 1
12   return counts
13
14 text = "this is the thing that this is "
15 text += "not the thing that it isn't."
16 result = count_words(text)
17 for key, value in result.items():
18   print(f"{key}: {value}")
```

```
this: 2
is: 2
the: 2
thing: 2
that: 2
not: 1
it: 1
isn't.: 1
```


Usage of the built-in `get()` method (2)

```
1  def count_words(text):
2    """returns a dictionary with the count
3       of each word in the text"""
4    counts = {}
5    text = text.lower()
6    words = text.split(" ")
7    for word in words:
8      if word not in counts.keys():
9        counts[word] = 1
10     else:
11       counts[word] = counts[word] + 1
12   return counts
13
14 text = "this is the thing that this is "
15 text += "not the thing that it isn't."
16 result = count_words(text)
17 for key, value in result.items():
18   print(f"{key}: {value}")
```

```
this: 2
is: 2
the: 2
thing: 2
that: 2
not: 1
it: 1
isn't.: 1
```


Usage of Python `defaultdict`

```
1  from collections import defaultdict
2
3  def count_words(text):
4    """returns a defaultdict with the count
5    of each word in the text"""
6    counts = defaultdict(int)
7    text = text.lower()
8    words = text.split(" ")
9    for word in words:
10     counts[word] = counts[word] + 1
11   return counts
12
13 text = "this is the thing that this is "
14 text += "not the thing that it isn't."
15 result = count_words(text)
16 for key, value in result.items():
17   print(f"{key}: {value}")
```

```
this: 2
is: 2
the: 2
thing: 2
that: 2
not: 1
it: 1
isn't.: 1
```

Dictionary Comprehension

- Like list comprehensions, dictionary comprehensions are a concise way to create dictionaries from sets of elements.
- They are convenient when we need to create a dictionary based on the values of another dictionary.
  - For example, suppose we have a dictionary that maps country codes to country names. If we want to create a dictionary that maps country names to country codes, we can simply use a dictionary comprehension:

```
codes2countries = {"US":"United States", "CA":\
  "Canada", "China":"CN", "Colombia":"CO",\
  "Mexico": "MX"}
countries2codes = {v:k for k,v in codes2countries.\
  items()}
print(countries2codes)
```

```
{'United States': 'US', 'Canada': 'CA', 'CN': 'China',
'CO': 'Colombia', 'MX': 'Mexico'}
```


Python JSON

Python JSON

- JSON (JavaScript Object Notation) is perhaps the most popular data-interchange format between a server and web application.
- Python has a built-in JSON package that lets us conveniently and quickly convert JSON to and from Python dictionaries, which share a similar key/value structure but are much easier to manipulate.
- In addition to (de)serializing dictionaries to and from JSON strings, the Python `json` module also includes methods to write and read Python dictionaries as Python files easily.

Python Dictionary to JSON string

```
1  import json
2
3  # Create a Python dictionary object
4  website={"url":"codesolid.com", "editor":\
5      "John Lockwood", "topics": ["Python",\
6      "Docker", "Lambda Functions",\
7      "Python for Beginners"]}
8
9  # json.dumps-dump the dictionary to a JSON string
10 json_string = json.dumps(website)
11 print(f"Dumps convered a type: {type(website)}")
12 print(f"to JSON as a type: {type(json_string)}")
13 print("String value:")
14 print(json_string)
```

```
Dumps convered a type: <class 'dict'>
to JSON as a type: <class 'str'>
String value:
{"url": "codesolid.com", "editor": "John Lockwood", "topics":
["Python", "Docker", "Lambda Functions", "Python for Beginners
"]}
```

- We use `json.dumps()` to encode Python dictionaries to JSON.
  - This method accepts a Python dictionary as its input and returns a JSON string.
  - To remember the name of this method, think about dumping a dictionary into a string (hence, “dumps”).

JSON string to Python Dictionary

```python
import json

# Create a Python dictionary object
website={"url":"codesolid.com", "editor":\
    "John Lockwood", "topics": ["Python",\
    "Docker", "Lambda Functions",\
    "Python for Beginners"]}

json_string = json.dumps(website)

# Reload it to a new dictionary and print the result
json_dict = json.loads(json_string)
print(f"Loads returns a type of {type(json_dict)}")
print(f"with a value of:\n{json_dict}")
```

```
Loads returns a type of <class 'dict'>
with a value of:
{'url': 'codesolid.com', 'editor': 'John Lockwood', 'topics':
['Python', 'Docker', 'Lambda Functions', 'Python for Beginners
']}
```

- We use `json.loads()` to decode JSON strings back to Python dictionaries.
  - This method accepts a JSON string as its input and returns a Python dictionary.

Writing a Python Dictionary to a JSON File

```python
import json

file_name = "website.json"

# A Python dicitonary as before:
website = {"URL": "codesolid.com",\
           "editor": "John Lockwood",\
           "topics": ["Python", "Docker",\
           "Lambda Functions",\
           "Python for Beginners"]}

# Write the dicitonary to a JSON file.
with open(file_name, 'w') as outfile:
    json.dump(website, outfile)
```

website.json - Notepad

File Edit Format View Help

```
{"URL": "codesolid.com", "editor": "John Lockwood", "topics":
["Python", "Docker", "Lambda Functions", "Python for Beginners"]}
```

- We use `json.dump()` to write a Python dictionaries to a JSON file.

Reading a JSON File to a Python Dictionary

```python
import json

file_name = "website.json"

with open(file_name, 'r') as infile:
    website = json.load(infile)

print(f"Loaded a {type(website)}")
print(f"with vaulues\n{website}")
```

website.json - Notepad

File Edit Format View Help

```
{"URL": "codesolid.com", "editor": "John Lockwood", "topics":
["Python", "Docker", "Lambda Functions", "Python for Beginners"]}
```

```
Loaded a <class 'dict'>
with vaulues
{'URL': 'codesolid.com', 'editor': 'John Lockwood', 'topics':
['Python', 'Docker', 'Lambda Functions', 'Python for Beginners
']}
```

- We use `json.load()` to load a JSON file into a Python dictionaries.

Python Object vs. JSON

- When we convert from Python objects to JSONs, 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

Python Object vs. JSON (2)

```
jsonStr = '[1,2,3]' # JSON ARRAY
pyList = json.loads(jsonStr)
pyList
```

```
[1, 2, 3]
```

```
try:
  jsonStr2 = '(4,5,6)' # not an array in JSON
  pyTuple = json.loads(jsonStr2)
except Exception as e:
  print(e)
print("-" * 40)
pyTuple = (4,5,6)
jsonStr2 = json.dumps(pyTuple)
jsonStr2 # here is array in JSON
```

```
Expecting value: line 1 column 1 (char 0)
----------------------------------------
'[4, 5, 6]'
```


Python Object vs. JSON (3)

```
import json

## examine these resulting JSON objects
print(json.dumps({"name": "John", "age": 30}))
print(json.dumps(["apple", "bananas"]))
print(json.dumps(("apple", "bananas")))
print(json.dumps("hello"))
print("-" * 40)
print(json.dumps(42), json.dumps(31.76))
print(json.dumps(True), json.dumps(False))
print(json.dumps(None), type(json.dumps(None)))
```

```
{"name": "John", "age": 30}
["apple", "bananas"]
["apple", "bananas"]
"hello"
----------------------------------------
42 31.76
true false
null <class 'str'>
```


Python Object vs. JSON (4)

```
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))
print(type(json.dumps(x)))
```

```
{"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}]}
<class 'str'>
```


Format the Result of JSON String

```
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, indent=4))
```

```
{
    "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 of JSON String (2)

```python
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 . and a space to separate objects,
# and a space, a = and a space to separate keys
# from their values:
print(json.dumps(x, indent=4, separators=(". "," = ")))
```

```
{
    "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 of JSON String

```python
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))
```

```
{
    "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
}
```


JSON Encoding Custom Object

```python
import json

class Person:
    def __init__(self, name, age):
        self.name = name
        self.age = age

class PersonEncoder(json.JSONEncoder):
    def default(self, obj):
        if isinstance(obj, Person):
            return {"name": obj.name, "age": obj.age}
        return super().default(obj)

# Create a custom object
person = Person("Ashutosh Krishna", 23)
# jsonStr = json.dumps(person) # <--
# TypeError: Object of type Person is not JSON serializable
# Encode the custom object using the custom encoder
json_str = json.dumps(person, cls=PersonEncoder)
# Print the encoded JSON string
print(json_str)
```

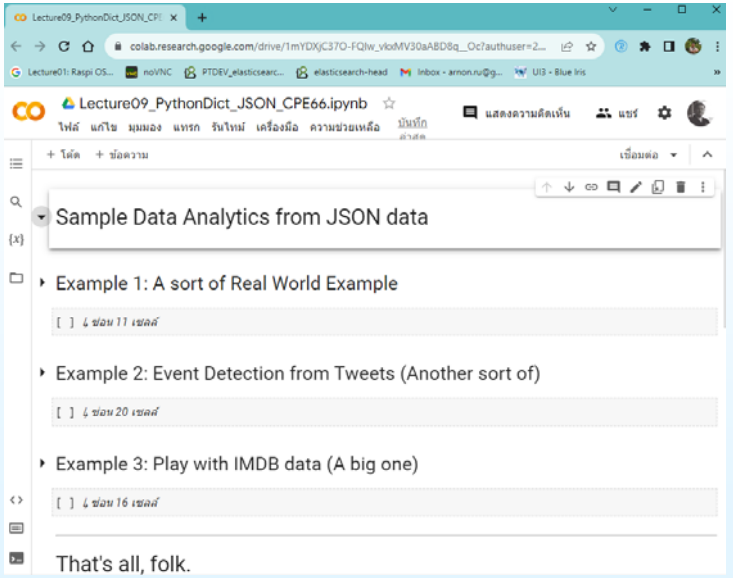
- JSONEncoder class allows us to customize the encoding process.
- To define how your custom object should be encoded into JSON format, we can extend the JSONEncoder and change its default method.

```
>>> %Run -c $EDITOR_CONTENT
  {"name": "Ashutosh Krishna", "age": 23}
>>>
```

REF: https://www.freecodecamp.org/news/how-to-use-the-json-module-in-python/

Sample Problem Solving

```python
import requests
import json

# Make the GET request to the horoscope API
url = 'https://horoscope-app-api.vercel.app/api/v1\
/get-horoscope/daily?sign=capricorn&day=today'
response = requests.get(url)
data = response.json()  # Convert the response to JSON

# Store the JSON data in a file
with open("horoscope_data.json", "w") as file:
    json.dump(data, file)

print("Data stored successfully!")

# Retrieve JSON data from the file
with open("horoscope_data.json", "r") as file:
    data2 = json.load(file)

# Access and process the retrieved JSON data
date = data2["data"]["date"]
horoscope_data = data2["data"]["horoscope_data"]

# Print the retrieved data
print(f"Horoscope for date {date}: {horoscope_data}")
```

To be continue..

つづく