SI 506: Lecture 18

TOPICS

1. JSON
2. `json` module
3. Nested loops

Vocabulary

- **Nested Loop**. A `for` or `while` loop located within the code block of another loop.

1.0 JSON

JSON (JavaScript Object Notation) is a lightweight data interchange format for exchanging information between systems.

JSON consists of two basic data structures and several value types.

1.1 JSON data structures

1. An *unordered* set of name-value pairs known as an *object* and denoted by curly braces (`{}`).
2. An *ordered* list of values known as an *array* and denoted by square brackets (`[]`).

1.2 JSON value types

1. string
2. number
3. boolean
4. array `[]`
5. object `{}`
6. null

1.3 JSON example

The New York Times provides an Article Search API (Application Programming Interface) that permits keyword searching and retrieval of JSON representations of NY Times articles. Below is a example of one such JSON document.

💡 Certain name-value pairs have been removed from the JSON document below in the interests of brevity. For instance only the top 3 keywords (out of 10) have been included in the example.

```
{
  "abstract": "Ten years ago, psychologists proposed that a wide range of
people would suffer anxiety and grief over climate. Skepticism about that
idea is gone.",
  "web_url": "https://www.nytimes.com/2022/02/06/health/climate-anxiety-
therapy.html",
```

```
  "source": "The New York Times",
  "headline": {
    "main": "Climate Change Enters the Therapy Room",
    "kicker": null,
    "content_kicker": null,
    "print_headline": "Anxiety Over Climate Change Lands on the
Therapist’s Couch"
  },
  "keywords": [
    {
      "name": "subject",
      "value": "Anxiety and Stress",
      "rank": 1,
      "major": "N"
    },
    {
      "name": "subject",
      "value": "Psychology and Psychologists",
      "rank": 2,
      "major": "N"
    },
    {
      "name": "subject",
      "value": "Global Warming",
      "rank": 3,
      "major": "N"
    }
  ],
  "pub_date": "2022-02-06T10:00:12+0000",
  "document_type": "article",
  "news_desk": "Science",
  "section_name": "Health",
  "byline": {
    "original": "By Ellen Barry",
    "person": [
      {
        "firstname": "Ellen",
        "middlename": null,
        "lastname": "Barry",
        "qualifier": null,
        "title": null,
        "role": "reported",
        "organization": "",
        "rank": 1
      }
    ],
    "organization": null
  },
  "type_of_material": "News",
  "word_count": 1940
}
```

2.0 json module

Like the `csv` module the Python standard libary's `json` module provides enhanced functionality for working with JSON files. JSON is a lightweight data interchange format for exchanging information between systems.

To use the `json` module you must import it:

```
import json
```

There are four `json` module functions; two of which are of particular interest to us:

Function	Purpose
`json.load()`	*Deserializes* (decodes) a text or binary file that contains a JSON document to a `dict` or `list`.
`json.dump()`	*Serializes* (encodes) an object as a JSON formatted stream to be stored in a file.
`json.loads()`	*Deserializes* (decodes) a string, bytes, or bytearray containing a JSON document to a `dict` or `list`.
`json.dumps()`	*Serializes* (encodes) an object to a JSON formatted string.

Since you will also be working with JSON documents between now and the end of the semester implmenting a function that can read a JSON document as well one that can write a JSON document to a file will prove useful.

2.1 Reading JSON files (`json_load()`)

The function `read_json()` reads a JSON document per the provided filepath, calls the json module's `json.load()` function in order to *decode* the file data as a `dict` or a `list` (of dictionaries), before returning the decoded data to the caller.

```
def read_json(filepath, encoding='utf-8'):
    """Reads a JSON document, decodes the file content, and returns a list
or dictionary if
    provided with a valid filepath.

    Parameters:
        filepath (str): path to file
        encoding (str): name of encoding used to decode the file

    Returns:
        dict/list: dict or list representations of the decoded JSON
document
    """

    with open(filepath, 'r', encoding=encoding) as file_obj:
        return json.load(file_obj)
```

2.2 Writing to a JSON file (`json_dump()`)

The function `write_json()` accepts a dictionary or a list of dictionaries, calls the json module's `json.dump()` function in order to *encode* the passed in data as JSON before writing the encoded data to the target file.

```
def write_json(filepath, data, encoding='utf-8', ensure_ascii=False,
indent=2):
    """Serializes object as JSON. Writes content to the provided filepath.

    Parameters:
        filepath (str): the path to the file
        data (dict)/(list): the data to be encoded as JSON and written to
the file
        encoding (str): name of encoding used to encode the file
        ensure_ascii (str): if False non-ASCII characters are printed as
is; otherwise
                            non-ASCII characters are escaped.
        indent (int): number of "pretty printed" indention spaces applied
to encoded JSON

    Returns:
        None
    """

    with open(filepath, 'w', encoding=encoding) as file_obj:
        json.dump(data, file_obj, ensure_ascii=ensure_ascii,
indent=indent)
```

Challenge 01

Task: Read in the NY Times JSON article data, extract articles written in 2022, and write the articles that meet the filter condition to a file as JSON.

1. In `main` call the function `read_json` and provide it with the appropriate filepath in order to retrieve NY Times Science Desk articles filtered on the subject "Psychology and Psychologists". Assign the return value to a variable named `articles`.

2. Loop over `articles` and in the loop block write an `if` statement that identifies articles with a publication year of 2022. Append each 2022 article to a list (name your choice).

    Review the JSON file `nyt-article-example.json` for the appropriate publication date name-value pair in order to derive the dictionary key name to use in your `if` statement.

   ! Each article contains an ISO-8601 date formatted string as the following example illustrates:

   ```
   2022-02-06T10:00:12+0000
   ```

💡 In your `if` statement utilize a `str` method to access the year portion (i.e., "2022") of the string. Alternative approaches could involve use of the `datetime` module or the third-party library `dateutil`, but these are out-of-scope for this challenge.

3. After exiting the loop, call the function `write_json` and write the "accumulator" list encoded as JSON to a file named `stu-nyt-articles-2022.json`.

3.0 Nested loops

What if you were asked to identify articles in the `articles` list that possess one or more keywords? Note, that each JSON document representing a NY Times article contains a list of keyword objects. For example a 2018 article by Paula Span entitled "Dementia Is Getting Some Very Public Faces" includes the following "keywords" JSON list:

```
"keywords": [
  {"name": "subject", "value": "Alzheimer's Disease", "rank": 1, "major":
"N"},
  {"name": "subject", "value": "Elderly", "rank": 2, "major": "N"},
  {"name": "subject", "value": "Dementia", "rank": 3, "major": "N"},
  {"name": "subject", "value": "Psychology and Psychologists", "rank": 4,
"major": "N"},
  {"name": "subject", "value": "Disabilities", "rank": 5, "major": "N"},
  {"name": "subject", "value": "Celebrities", "rank": 6, "major": "N"},
  {"name": "persons", "value": "O'Connor, Sandra Day", "rank": 7, "major":
"N"}
]
```

If we wanted to return articles in the `articles` list that where tagged with either "Dementia" and/or "Alzheimer's Disease" we could write a function that performs a keyword look up (e.g., `has_keywords(keywords, filters)` and then call it from inside a `for` loop in order to identify dementia-related articles. Alternatively, we could implement a nested loop.

💡 # Dementia is a generic term that a range of neurological conditions impacting the brain including Alzheimer's Disease.

A nested loop refers to a loop located within the code block of another loop. During each iteration of the "outer" loop, the "inner" loop will execute, completing all its iterations (unless terminated early) *prior* to the outer loop commencing its next iteration, if any.

```
for < element > in < sequence >:
    # indented block
    for < element > in < element (itself a sequence) >
        < statement A >
        < statement B >
        ...
```

The following example illustrates the pattern in action. The outer loop iterates over the `articles` list. The inner loop iterates over each article's "keywords" list. If the keyword is a subject keyword and the keyword value is either "Alzheimer's Disease" or "Dementia" a match is obtained and the article is appended to the list `dementia_articles`.

❗ The data set contains five (5) keyword "name" values: "subject", "glocations", "organizations", "persons", and "creative_works" so filtering on "subject" is required.

💡 Note the use of the `break` statement inside the inner loop. Since an article can contain both keyword objects it is imperative that no additional loop iterations occur after the first match is obtained in order to avoid duplicate append operations.

```
# Get all articles touching on Dementia
dementia = []
for article in articles:
    for keyword in article['keywords']:
        if keyword['name'] == 'subject' and keyword['value'] in
("Alzheimer's Disease", 'Dementia'):
            dementia.append(article)
            break # avoid appending duplicates due to either/or membership
check

# Write to file
write_json('stu-nyt-dementia-articles.json', dementia_articles)
```

Challenge 02

Task: Return a list of *unique* keyword values and write the results to a text file.

1. In `main`, create an empty list named `subjects`. Then implement a nested loop that iterates over each article's keywords list.
2. Inside the nested loop block, write an `if` statement that encompasses the following two conditions:
    1. The keyword dictionary's "name" value equals "subject", and
    2. The keyword dictionary's "value" is not a member of `subjects`

    If both conditions return `True` append the keyword "value" to the `subjects` list.
3. After exiting the outer loop write the `subjects` list to a text file named `stu-subjects.txt`. Employ a `with open()` statement to accomplish the task.

    💡 If you forget how to write a list to a text file review **lecture 12, section 3.5**.

    Requirements

    1. *Prior* to writing the list to a file.Perform an alphanumeric sort of the list using the built-in function `sorted()`.

2. ❗ Be sure to add a trailing `\n` newline to every string element written to the file in order to avoid writing each string to a single line.

```
#MeToo Movement
ANOREXIA NERVOSA
ANTIDEPRESSANTS
Addiction (Psychology)
Advertising and Marketing
...
```

not

```
#MeToo MovementANOREXIA NERVOSAANTIDEPRESSANTSAddiction
(Psychology)Advertising and Marketing ...
```

Challenge 03

Task: Obtain keyword subject counts across all articles. Store in a dictionary and write to a file as JSON.

1. In `main`, create an empty dictionary named `subject_counts`. Then implement a nested loop that iterates over the `subjects`, `articles`, and `keyword` lists in order to obtain the subject counts to be assigned to `subject_counts` as key-value pairs.

   Requirements

   1. Your solution *must* feature three (3) loops, two of which are nested.
      1. Outer: Loop over each subject (`str`) in `subjects`.
      2. Inner (1): Loop over each article (`dict`) in `articles`.
      3. Inner (2): Loop over each keyword (`dict`) in the article's keywords (`list`). Each keyword dictionary contains the following key-value pairs:

```
{
    'name': 'subject',
    'value': 'Therapy and Rehabilitation',
    'rank': 5,
    'major': "N"
}
```

   2. Inside the innermost loop implement the following conditional logic to either add a new key-value pair *or* increment an existing key-value pair in the `subject_counts` dictionary:
      - Check whether or not the keyword "name" value equals "subject" *and* the keyword "value" value equals the outer loop subject element string.

- If the above compound condition resolves to `True` then *inside* the `if` statement block, check if the outer loop subject element matches a key in the `subject_counts` `dict_keys` object.

```
if < expressions >:
    if < expression >:
        ...
    ...
```

If the key is *not* found in the `dict_keys` object, either

- add a *new* key-value pair assigning the subject element as the key and `1` as the value, *or*
- increment an existing key-value pair (i.e., with a key name that matches the outer loop's subject element string) by `1`.

2. After exiting the outer loop uncomment the dictionary comprehension in order to return a new dictionary `subject_counts` with key-value pairs sorted by value (descending order) and then by key (alphanumeric, ascending order).

3. Call the function `write_json` and write `subject_counts` encoded as JSON to a file named `stu-nyt-subject_counts.json`.

Challenge 04

Task: Refactor the Challenge 03 nested loop by relocating the innermost loop's keyword "name" and "value" check to a function named `has_subject`. Then obtain keyword subject counts across all articles. Store in a dictionary and write to a file as JSON.

1. In `main`, create an empty dictionary named `subject_counts_fx`. Then implement a nested loop that iterates over the `subjects` and `articles` in order to obtain the subject counts to be assigned to `subject_counts` as key-value pairs.

2. Implement the function named `has_subject`. The function defines two parameters:

    - subjects (`list`): list of "subject" string elements
    - subject (`str`): string to locate in the `subjects` list

    If the passed in `subject` is found in the passed in `subjects` list, the boolean `True` is returned by the function to the caller; otherwise `False` is returned.

    💡 Transfer the following conditional logic to the function implemented in the innermost loop in the previous Challenge code:

    - Check whether or not the keyword "name" value equals "subject" *and* the keyword "value" value equals the outer loop subject element string.

3. After implementing the function `has_subject` return to `main`. Loop over the `subjects` and `articles` lists as in the previous challenge. However *replace* the innermost article keywords loop

with an `if` statement that encompasses a call to the function `has_subject`.

4. If the above condition resolves to `True` then *inside* the `if` statement block, incorporate the same conditonal logic employed in the previous challenge to either
    - add a *new* key-value pair assigning the subject element as the key and `1` as the value, *or*
    - increment an existing key-value pair (i.e., with a key name that matches the outer loop's subject element string) by `1`.
5. After exiting the outer loop uncomment the dictionary comprehension in order to return a new dictionary `subject_counts_fx` with key-value pairs sorted by value (descending order) and then by key (alphanumeric, ascending order).
6. Uncomment the `assert` statement to confirm that `subject_counts` is the equivalent of `subject_counts_fx`. If `False` a runtime exception will be generated.