SI 506 Lecture 19

TOPICS

1. Challenges

Vocabulary

- **JSON**. JSON (JavaScript Object Notation) is a lightweight datainterchange format for exchanging information between systems.
- **Nested Loop**. A `for` or `while` loop located within the code block of another loop.

Data

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.

Today's data comprises a list of 200 JSON documents that represent the most recent NY Times articles published by the Science Desk and covering the subject of Pyschology and Psychologists.

An example JSON document named `nyt-article-example.json` is included in today's lecture files. You should review it and familiarize yourself with its structure and name-value pairs.

💡 Certain name-value pairs have been removed from the JSON documents in the interests of brevity. In addition, a "person" object containing all `null` values has also been removed in order to eliminate the need to introduce exception handling in your code.

Challenge 01

Task: Provide article subject counts employing a single nested `for` loop and a helper function named `get_article_subjects`. Write the counts to a JSON file. This challenge represents a refactoring (e.g., revision) of a previous challenge.

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. Implement the function named `get_article_subjects`. The function defines a single parameter named `article` and returns a list of "subject" string values contained in an article's "keywords" list. Review the function's docstring to better understand it's expected behavior.

   ❗ Recall that you must filter out keyword dictionaries with a "name" value that does not equal "subject".

3. After implementing `get_article_subjects` return to `main`. Create an empty accumulator dictionary named `subject_counts`. Loop over `articles` and for each article encountered call the function `get_article_subjects` passing it the appropriate argument. Assign the return value to variable named `article_subjects` (inside the loop).

4. Then implement a *nested* loop that iterates over `article_subjects`. Check if the "subject" string is used as a key in the `subject_counts` dictionary. If the key is *not* found in the `dict_keys` object add a *new* key-value pair assigning the subject element as the key and `1` as the value. Otherwise, increment the matching key-value pair by `1`.

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

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

Challenge 02

Task: Provide a list of article authors, filtering out duplicate entries. Write the list of authors to a CSV file.

1. In `main` create an empty accumulator list named `authors`. Loop over the `articles` list and in an inner loop, access the "person" list stored in the "byline" dictionary and loop over it.

2. For each "person" dictionary encountered extract the first, middle, and last name values and store in a tuple named `name` as follows:

   ```
   (< lastname >, < firstname >, < middlename >)
   ```

3. Then check whether or not the `name` tuple is in the `authors` list. If it is *not* a member of the list append it.

   💡 Sequences of the same type support comparison by position. Corresponding elements/items in each sequence are compared *lexicographically* as described in the Python value comparisons documentation:

   > Lexicographical comparison between built-in collections works as follows:
   >
   > For two collections to compare equal, they must be of the same type, have the same length, and each pair of corresponding elements must compare equal (for example, [1, 2] == (1, 2) is false because the type is not the same).
   >
   > Collections that support order comparison are ordered the same as their first unequal elements (for example, [1, 2, x] <= [1, 2, y] has the same value as x <= y). If a corresponding element does not exist, the shorter collection is ordered first (for example, [1, 2] < [1, 2, 3] is true).

   Tuple comparison

   ```
   >>> name01 = ('Brody', 'Jane', 'E.')
   >>> name02 = ('Brody', 'Jane', 'E.')
   >>> name01 == name02
   True
   ```

```
>>> name03 = ('Angier', 'Natalie', None)
>>> name01 == name03
False
```

4. After exiting the outer loop uncomment the provided dictionary comprehension in order to return a new dictionary named `authors` with key-value pairs sorted by last name, first name, and then middle name (the latter converted to `''` if `None`).

   💡 Most author records do not include a middle name value. Since sorting on `None` triggers a runtime `TypeError` exception, I replace each `None` encountered with a blank string (`''`) by passing the expression `x[2] or ''` to the built-in `str` function in my `lambda` expression. Since `None` is "falsy" the blank string is returned.

5. Call the function `write_csv` and write `authors` to a file named `stu-nyt-authors.csv`. Also pass in a `headers` argument comprising a sequence containing the strings "last_name", "first_name", and "middle_name".

Challenge 03

Task: Group articles by author and write data to a JSON file.

1. In `main` create an empty accumulator list named `citations`. Loop over the `articles` list and in an inner loop, access the "person" list stored in the "byline" dictionary and loop over it as in the previous challenge.

2. For each "person" dictionary encountered extract the first, middle, and last name values and store in a list (not tuple) named `name` as follows:

```
[< lastname >, < firstname >, < middlename >]
```

3. In the inner loop block, create a key composed of the `name` items separated by underscores (_). If the "middlename" value is `None` *exclude* it from the string:

```
'< lastname >_< firstname >_< middlename >'  <-- 'Brody_Jane_E.'
```

   or

```
'< lastname >_< firstname >'  <-- 'Angier_Natalie'
```

   Assign the string to a variable named `key`.

4. In the inner loop block, build an f-string that represents the article citation formatted as follows:

```
< pub_date > - < headline main >
```

Assign the string to a variable named `value`.

💡 Review the example article JSON document for the required name-value pairs.

5. Check if `key` can be found among the keys in the `citations` dictionary. If the key is *not* found in the `dict_keys` object add a *new* key-value pair assigning `key` as the key and put `value` in a list and assign the list as the value. Otherwise, append `value` to the list assigned to the matching key-value pair.

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

Challenge 04

Task: Duplicate authors exist in the `authors` list due to the use of uppercase names. Refactor Challenge 03 to correct this issue.

1. The author Carey Benedict has 44 citations to their name but due to the use of uppercase characters used in one of Carey's "person" records, both the `authors` and `citations` list fail to account for this issue (Carey is list twice). Return to challenge 03 and in the inner loop add code that checks whether or not each last name, first name, and middle name value is rendered in uppercase. If not call the appropriate `str` method to return a capitalized version of the name value. Place this code before creating the `key` value.

   ❗ In your `if` statement be sure to check for the presence of `None` as a value and ignore it (i.e., do not call the `str` method on it or a runtime exception will occur).

2. After implementing the fix call the function `write_json` and write `citations` encoded as JSON and update the file `stu-nyt-citations.json`. Then check Carey's record.