SI 506 Lecture 19

TOPICS

1. Nested loop challenges
    1. Challenge 01
    2. Challenge 02
    3. Challenge 03
    4. Challenge 04
    5. Challenge 05

Vocabulary

- **JSON**. JSON (JavaScript Object Notation) is a lightweight data interchange 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 300+ JSON objects that represent the most recent NY Times articles published by the Science Desk that report on scientific research findings.

An example JSON document named `nyt-article-research-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.

1.0 Nested loop challenges

1.1 Challenge 01

Task: Provide article keyword subject counts employing a nested `for` loop and a helper function named `get_article_subjects`. Write the counts to a JSON file.

1. Implement the function named `get_subject_keywords`. The function defines a single parameter named `article` and returns a list of keyword "subject" string values contained in an article's "keywords" list. Review the function's docstring regarding its expected behavior, parameters, and return value.

💡 This function implements one of two nested loops that need to located inside the `articles` loop.

Function requirements and hints

1. The list that you return to the caller *must* include only keyword "value" strings accessed from keyword dictionaries with a "name" value that equals "subject".
2. You should also ensure that no duplicates of the keyword subject's "value" are appended to the local accumulator list.

💡 Tuesday's lecture includes code similar to what you should implement in the function block.

2. After implementing `get_subject_keywords()` return to `main`.
3. Call the function `read_json()` and provide it with the filepath argument `'./nyt-articles-research-20221102.json'` to retrieve NYT Science Desk "Research" articles. Assign the return value to a variable named `articles`.
4. Create an empty accumulator dictionary named `keyword_counts`.
5. Loop over `articles` and for each article encountered call the function `get_subject_keywords()` passing it the appropriate argument. Assign the return value to a variable named `keywords` (inside the outer loop).
6. Next, implement a *nested* inner loop that iterates over `keywords`. Check if the "keyword" (a `str`) is used as a key in the `keyword_counts` dictionary. If the key is *not* found in the `dict_keys` object add a *new* key-value pair assigning the keyword element as the key and `1` as the value. Otherwise, *increment* the matching key-value pair by `1`.
7. 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).
8. Uncomment the `write_json()` function call and write `subject_counts` encoded as JSON to a file named `stu-nyt-subject_counts.json`. Review your file output.

1.2 Challenge 02

Task: Everybody loves dinosaurs. Extract articles tagged with the keywords: Dinosaurs, Fossils, Paleontology, or Pterosaurs and write the articles to a CSV file.

1. In `main()` create an empty accumulator list named `paleontology`.
2. Loop over `articles` and retrieve each article's subject keywords by calling the function `get_subject_keywords()`. Assign the return value to a variable named `keywords`.
3. Next, implement a *nested* inner loop that iterates over `keywords`. If the keyword string matches **any** of the following keyword subjects:
    - Dinosaurs

   - Fossils
   - Paleontology
   - Pterosaurs

   append a dictionary to `paleontology` that contains the following "article" key-value pairs:

   1. headline_main (main headline value)
   2. byline (author's original byline value)
   3. web_url
   4. pub_date

   💡 Review the `nyt-article-research-example.json` file's key-value pairs. I also recommend that you pass a dictionary literal to the `list.append()` method.

4. In order to avoid appending duplicate dictionary representations of each article employ the appropriate **control statement** inside the `if` block to terminate the inner `keywords` loop whenever an article's subject keyword matches *any* of the keyword subjects listed above.

5. Call the function `write_dicts_to_csv()` and pass it the filepath `stu-nyt-paleontology.csv`, the paleontology list, and the keys from one of the list's dictionaries to serve as the CSV's header row. Review your file output.

1.3 Challenge 03

Task: Extract a list of all authors filtering out duplicate entries. Then write the list of authors to a CSV file.

1. Implement the function named `get_author_name`. The function defines a single parameter named `author` and returns a three-item tuple comprising the person's last name, first name, and middle name (if provided). Review the function's docstring regarding its expected behavior, parameters, and return value.

   Function requirements and hints

   1. Access and return the following article "person" key-value pairs in the following order:

   2. lastname

   3. firstname

   4. middlename

   5. Implement the function with a single line of code.

2. After implementing `get_author_name()` return to `main()`.

3. Create an empty accumulator list named `authors`.

4. Loop over the `articles` list. Implement a nested loop by looping over the "person" list stored in the article's "byline" dictionary.

   💡 Employ subscript operator chaining in your loop to access the byline "person" list.

5. Inside the inner loop call `get_author_name()` and pass it a "author" dictionary as the argument. Assign the return value (a `tuple`) to a variable named `name`.

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

   💡 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
   ```

7. After exiting the outer loop **uncomment** the "Sort Authors" variable assignment (i.e., `authors = [...]`). The new list is created using a list comprehension, the built-in `sorted()` function, and an anonymous `lambda` function to sort the list by last name, then first name, and, lastly, by the middle name.

   💡 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.

8. 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". Review your file output.

1.4 Challenge 04

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

1. In `main` create an empty accumulator list named `citations`.
2. Loop over the `articles` list. Implement a nested loop by looping over the byline "person" list stored in the article's "byline" dictionary in the same way as the previous challenge.
3. Inside the inner loop call `get_author_name()` and pass it the "author" dictionary as the argument. Assign the return value (a `tuple`) to a variable named `name`.
4. In the inner loop block, employ conditional statements to create a key composed of the `name` items formatted per the following rules:

    1. If the person possesses a "middlename" value include it in the string you build.

    ```
    '< lastname >, < firstname > < middlename >'  <-- 'Andrews, Robin George'
    ```

    2. Otherwise, if no "middlename" is provided (i.e., the value equals `None`) exclude it from the string you build.

    ```
    '< lastname >, < firstname >'  <-- 'Anthes, Emily'
    ```

    3. Assign the string to a variable named `key`.

5. After the `key` assignment, create a dictionary named `story` that contains the following article key-value pairs:

    1. pub_date
    2. headline_main (main headline value)
    3. web_url

    💡 Review the `nyt-article-research-example.json` file's key-value pairs.

6. Check if the `key` you created can be found among the keys in the accumulator `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 a list containing `story` as the value. Otherwise, append `story` to the list assigned to the matching key-value pair.

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

1.5 Challenge 05

Task: An NYT staff writer is duplicated in the `citations` list due to a missing middle name or "double-barrelled" name. Combine the citations and then delete the redundant key-value pair.

1. The NYT staff writer Roni Caryn Rabin is listed in `stu-nyt-citations.json` both as "Rabin, Roni Caryn" and "Rabin, Roni".

```
"Rabin, Roni": [
  {
    "pub_date": "2022-08-30T17:24:44+0000",
    "headline_main": "Paxlovid Cuts Covid Deaths Among Older People,
Israeli Study Finds",
    "web_url": "https://www.nytimes.com/2022/08/30/health/paxlovid-
efficacy-seniors.html"
  }
]
```

2. Access the `citations` list "Rabin, Roni Caryn" key-value pair and *insert* the single "Rabin, Roni" citation (a `dict`) into the "Rabin, Roni Caryn" list in the second (2nd) position.

3. After inserting the dictionary **remove** the "Rabin, Roni" key-value pair from `citations` by passing the dictionary as an argument to the appropriate built-in function.

4. Uncomment the `write_json()` function call and write the mutated `citations` list encoded as JSON to a file named `stu-nyt-citations-corrected.json`. Review your file output.