SI 506 Lecture 23

1. Dictionary comprehension
    1. Basic syntax
    2. Simple example
2. Transforming values
3. Conditional statements
    1. Challenge 01
    2. `if-else` statements
    3. `if-elif-else` statements
4. Nested loops
5. Another example

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 Climate Desk.

An example JSON document named `nyt-article-climate-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 Dictionary comprehension

> A compact way to process all or part of the elements in an iterable and return a dictionary with the results.

Source: https://docs.python.org/3/glossary.html

1.1 Basic syntax

```
new_dict = {key: val for element in iterable}

new_dict = {key: val for key, value in dict.items()}
```

1.2 Simple example

If you were asked to provide a JSON file comprising word counts for each article in the `nyt_articles` list you might opt for the following simple data structure:

💡 Each article "web_url" is a unique identifier.

```
article_word_counts = {}
    for article in nyt_articles:
        article_word_counts[article['web_url']] = article['word_count']

write_json('./stu-nyt-article_word_counts-v1p0.json', article_word_counts)
```

You could obtain the same results employing a dictionary comprehension:

```
article_word_counts = {article['web_url']: article['word_count'] for
article in nyt_articles}

write_json('./stu-nyt-article_word_counts-v1p1.json', article_word_counts)
```

2.0 Transforming values

You can pass a function or call an object method in a dictionary comprehension in order to transform values. In the following `for` loop and dictionary comprehension examples both the article "web_url" and "pub_date" values are transformed in order to return a dictionary in which a shortened publication date (i.e., Year-Month-Day) is mapped to a key comprising the article URL reduced to the resource path (e.g., "lula-brazil-rainforest-climate") minus the file extension (i.e., '.html'):

```
article_pub_dates = {}
    for article in nyt_articles:
        resource_path = article['web_url'].split('/')[-1][:-5] # resource
path only minus extension
        # resource_path = Path(article['web_url']).stem # better
        pub_date = article['pub_date'].split('T')[0]
        article_pub_dates[resource_path] = pub_date

    write_json('./stu-nyt-article_pub_dates-v1p0.json', article_pub_dates)
```

The dictionary comprehension is compact and expressive:

```
article_pub_dates = {
    Path(article['web_url']).stem: article['pub_date'].split('T')[0]
    for article in nyt_articles
    }
```

💡 For `pathlib.Path` attributes see Geir Arne Hjelle, "Python 3's pathlib Module: Taming the File System" (Real Python, Apr 2018).

3.0 Conditional statements

A dictionary comprehension can specify one or more conditional statements in order to assign a subset of a dictionary to a new dictionary.

```
new_dict = {key: val for element in iterable if condition}

new_dict = {key: val for key, value in dict.items() if condition}
```

The `nyt_articles` list contains multimedia listings. These records can be accessed by filtering on the "document_type" as the following `for` loop illustrates:

```
multimedia = {}
  for article in nyt_articles:
      if article['document_type'].lower() == 'multimedia':
          multimedia[article['web_url']] = article
```

The same can be achieved by employing a dictionary comprehension:

```
multimedia = {
    article['web_url']: article
    for article in nyt_articles
    if article['document_type'].lower() == 'multimedia'
    }
```

💡 Note that the typical comprehension variables `key` and `val` or `k` and `v` are conventions; you are free to employ comprehension variable names that better express the nature of the data as in the previous example.

3.1 Challenge 01

Task: Convert a conventional `for` loop to 1) a `for` loop with a nested dictionary comprehension and 2) a list comprehension containing a nested dictionary comprehension.

1. Convert the following `for` loop that appends "shortened" article dictionaries to the list named `articles_short` to a `for` loop that appends each shortened article dictionary by employing a dictionary comprehension that leverages the `keep_keys` tuple instead of a dictionary literal. Append the new list to the variable named `articles_short`.

```
articles_short = []
for article in nyt_articles:
    articles_short.append(
        {
            'web_url': article['web_url'],
            'pub_date': article['pub_date'],
            'document_type':  article['document_type'],
            'type_of_material': article['type_of_material'],
            'word_count': article['word_count']
            }
        )

# TODO Uncomment
# write_json('./stu-nyt-articles_short-v1p0.json', articles_short)

# Alternative (for loop / dict comprehension)
keep_keys = ('web_url', 'pub_date', 'document_type',
'type_of_material', 'word_count')
articles_short = []

# TODO Implement for loop and dict comprehension

# TODO Uncomment
write_json('./stu-nyt-articles_short-v1p1.json', articles_short)
```

2. After completing the task uncomment the function call to `write_json()` and write the list to a JSON file. Review the file output.

3. Next, convert the `for` loop / dict comprehension code to a list comprehension that features a nested dictionary comprehension. Assign the comprehension to the variable named `articles_short`.

```
# Alternative (list and dict composition combined)
articles_short = None # TODO write comprehension

# TODO Uncomment
write_json('./stu-nyt-articles_short-v1p2.json', articles_short)
```

4. After completing the task uncomment the function call to `write_json()` and write the list to a JSON file. Review the file output.

5. Compare the three companion files (versions 1.0-1.2). They *must* match each other line for line, character for character, indent for indent.

3.2 `if-else` statements

You can employ `if-else` logic in a dictionary comprehension. The `if-else` logic is placed *before* the `for` statement and employs the ternary form of the `if-else` operator.

```
new_dict = {
    key: (some_val_if_true if condition else some_other_val)
    for key, val in dict_.items()
    }

new_dict = {
    (key if condition else default_key): (some_val_if_true if condition
else some_other_val)
    for key, val in dict_.items()
    }
```

The example below creates a new dictionary based on the articles in the `article_short` list. Depending on the word count the value (a `dict`) mapped to the key defines the read time as either "LONG" or "SHORT":

```
articles_read_time = {
    article['web_url'].split('/')[-1][:-5]: (
        {'word_count': article['word_count'], 'read time': 'LONG'}
        if (article['word_count']) >= 750
        else {'word_count': article['word_count'], 'read time': 'SHORT'}
        )
    for article in articles_short if article['document_type'].lower() ==
'article'
    }
```

3.3 `if-elif-else` statements

The `elif` statement is *not* recognized inside a dictionary comprehension. You can mimic `if-elif-else` logic by employing multiple `else` statements.

Dividing the article read times into three categories (SHORT, MEDIUM, LONG) requires the following adjustment:

```
articles_read_time = {
    article['web_url'].split('/')[-1][:-5]: (
        {'word_count': article['word_count'], 'read time': 'LONG'}
        if article['word_count'] >= 1000
        else {'word_count': article['word_count'], 'read time': 'MEDIUM'}
        if 500 <= (article['word_count']) < 1000
        else {'word_count': article['word_count'], 'read time': 'SHORT'}
        )
    for article in articles_short if article['document_type'].lower() ==
'article'
    }
```

💡 If dictionary comprehension readability is concern then consider relocating the business logic (e.g., categorizing economies) to a function and then use it to transform the data by calling it from inside the dictionary comprehension.

4.0 Nested loops

You can embed nested loops in a dictionary comprehension. The outer loop is listed first followed by the inner loop:

```
new_dict = {key: val for outer_element in outer_loop for inner_element in
inner_loop if condition}
```

In the following examples NYT article author names are extracted and stored in a list of dictionaries:

```
authors = {
    '< web_url >': [< author 01 >],
    '< web_url >': [< author 01 >, < author 02 >, ...],
    ...
}
```

Each author is represented by a dictionary comprising three key-value pairs ("firstname", "middlename" and "lastname"). These values are accessed via `article['byline']['person']`. Since an article can contain coauthors the author dictionaries are stored in a list and mapped to the article's "web_url" identifier.

You could write a nested `for` loop to populate the `authors` list:

```
authors = {}
    for article in nyt_articles:
        val = []
        for person in article['byline']['person']:
            val.append(
```

```
                {
                    'firstname': person['firstname'],
                    'middlename': person['middlename'],
                    'lastname': person['lastname']
                    }
                )
        authors[article['web_url']] = val
```

You could reimplement the nested loop as list comprehension that features a nested dictionary comprehension:

```
authors = {
    article['web_url']: [
        {
            'firstname': author['firstname'],
            'middlename': author['middlename'],
            'lastname': author['lastname']
            }
        for author in article['byline']['person']
        ]
    for article in nyt_articles}
```

💡 Nested dictionary comprehensions can get ugly. Check out this example in stackoverflow.com:

```
data = {outer_k: {inner_k: myfunc(inner_v) for inner_k, inner_v in
outer_v.items()} for outer_k, outer_v in outer_dict.items()}
```

5.0 Another example

The following example code groups the climate articles by publication month and then computes the monthly average word count per article

```
months = {i: [] for i in reversed(range(1, 12))}

print(f"\n5.0.1 months dict = {months}")

for article in articles_short:
    month = datetime.strptime(article['pub_date'], '%Y-%m-
%dT%H:%M:%S%z').month
    if month in months.keys():
        months[month].append(article)
    else:
        months[month] = [article]
```

```
# Filter out non-articles (e.g., multimedia)
monthly_word_count_totals = {i: 0 for i in reversed(range(1, 12))}
for month, articles in months.items():
    for article in articles:
        if article['document_type'].lower() == 'article':
            monthly_word_count_totals[month] += (article['word_count'])

# Return lists of word counts
monthly_word_counts = {i: [] for i in reversed(range(1, 12))}
for month, articles in months.items():
    for article in articles:
        if article['document_type'].lower() == 'article':
            monthly_word_counts[month].append(article['word_count'])

# Total word counts per month
monthy_word_counts_sum = {month: sum(counts) for month, counts in
monthly_word_counts.items()}

# Average word counts per month (floor division)
monthy_word_counts_avg = {
    month: (sum(counts) // len(counts))
    for month, counts in monthly_word_counts.items()
    }
```