SI 506 Lecture 22

Topics

Data

Today's data is sourced from The World Bank. The Bank assigns the world's economies to four income groups based on the World Bank Atlas method for calculating Gross National Income (GNI) per capita in current USD using data from the previous year. The classifications are updated each year on 1 July. For the current 2022 fiscal year, the classifications are delineated as follows:

Fiscal Year	Classification	GNI per capita (2020)
2021-2022	`Low income`	<= $1,045.00 (USD)
2021-2022	`Lower middle income`	between $1,046.00 and $4,095.00 (USD)
2021-2022	`Upper middle income`	between $4,096.00 and $12,695.00 (USD)
2021-2022	`High income`	>= $12,696.00 (USD)

Source: World Bank Blogs, "New World Bank country classifications by income level: 2021-2022"; World Bank Country and Lendings Groups.

1.0 The list comprehension

> A compact way to process all or part of the elements in a sequence and return a list with the results.

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

💡 You can also write dictionary comprehensions but that is topic for the next lecture.

1.1 Basic syntax

```
new_list = [expression for element in sequence]
```

1.2 Simple example

If you were asked to return a list of country codes contained in the `wb-economies-2021_2022.json` file you would likely write the following code after reading the data file:

```
country_codes = []
for country in countries:
    country_codes.append(country['country_code'])
```

The code performs the following operations:

1. Instantiates an empty "accumulator" list named `country_codes`.
2. Loops over the `countries` list of nested dictionaries, each representing a country.
3. Append's each element's "country_code" value to the accumulator list.

The above represents a fine implementation that gets the job done. However, you can also utilize a *list comprehension* to accomplish the task—an approach that is arguably more elegant, and, depending on the scenario, a more performant way to create a new list from an existing list.

```
country_codes = [country['country_code'] for country in countries]
```

2.0 Transforming values

You can pass a function or call an object method in a list comprehension in order to transform values.

For example, assume that you need to access the ISO-3165-1 alpha-3 country codes in the nested country dictionaries comprising the `countries` list and store them in a new list named `country_codes`. However, you notice that roughly half of the codes employ lowercase rather than uppercase characters as defined by ISO-3165-1 alpha-3.

The lowercase codes should by converted to uppercase. You can employ a list comprehension and the `str.upper()` method to perform the conversion.

```
country_codes = [country['country_code'].upper() for country in countries]
```

💡 if your goal is to *mutate* the nested dictionaries comprising the `countries` list, a `for` loop is the preferred approach. Recall that a list comprehension is used to create a *new* list from an existing sequence.

3.0 Conditional statements

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

```
new_list = [expression for element in sequence if condition]
```

The following list comprehension returns only those economies located in the region "East Asia & Pacific":

```
east_asia = [country for country in countries if country['region'] ==
'East Asia & Pacific']
```

3.1 Challenge 01

Task: Write a list comprehension that returns economies located in either Latin America, the Carribbean or North America *and* are categorized as lower middle income economies by The World Bank.

1. Write a list comprehension that accesses all "lower middle income" economies located in "Latin America & Caribbean" *or* "North America".

   Assign the new list to a variable named `americas_lower_middle`.

2. Call the function `read_json` and write the `americas_lower_middle` list to the file `stu-americas-lower_middle.json`.

   💡 You can enhance readability by writing list comprehensions that exceed 80-100 characters and/or feature complex conditions or other expressions across multiple lines vertically.

   ```
   new_list = [
       expression
       for element in sequence
       if condition
       ...]
   ```

3.2 `if-else` statements

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

```
new_list = [expression if condition else expression for element in
sequence]
```

The example below returns a list of tuples that categorizes economies as either "High income" or "Middle/low income":

```
two_categories = [
    (country['country_name'], 'High income')
    if country['income_group'].lower() == 'high income'
    else (country['country_name'], 'Middle/low income')
    for country in countries
    ]
```

3.3 `if-elif-else` statements

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

Dividing World economies into three categories ("High income", "Middle income", or "Low income") can be accomplished by adding an additional `else` statement to the list comprehension:

```
three_categories = [
    (country['country_name'], 'High income')
    if country['income_group'].lower() == 'high income'
    else (country['country_name'], 'Middle income')
    if country['income_group'].lower() in ('lower middle income', 'upper
middle income')
    else (country['country_name'], 'Low income')
    for country in countries
    ]
```

However, you may well conclude that the above comprehension's `if-else-else` leads to diminished readability when compared to a `for` loop:

```
three_categories = []
    for country in countries:
        if country['income_group'].lower() == 'high income':
            three_categories.append((country['country_name'], 'High
income'))
        elif country['income_group'].lower() in ('lower middle income',
'upper middle income'):
            three_categories.append((country['country_name'], 'Middle
income'))
```

```
        else:
            three_categories.append((country['country_name'], 'Low
income'))
```

💡 If list 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 a list comprehension.

```
three_categories = [categorize_economy(country) for country in countries]
```

4.0 Nested loops

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

```
new_list = [expression for outer_element in outer_sequence for
inner_element in inner_sequence if
condition]
```

The World Bank also groups countries. After reading in the group data from the file `wb-groups-2021_2022.json` we can retrieve economies characterized as members of the "Arab World" (group code = "ARB") using a list comprehension that employs a nested loop:

💡 Review the two data sets to determine which key-value pair(s) can be used to link a country to a group.

```
arab_world = [
    country
    for group in groups
    for country in countries
    if group['group_code'] == "ARB"
    and group['country_code'] == country['country_code']
    ]
```

4.1 Challenge 02

Task: Employ a list comprehension and a function to create a new list of Sub-Saharan African economies that combine key-value pairs drawn from related country and group dictionaries.

1. In `main` call the function `read_json` and pass the filepath `wb-groups-2021_2022.json`. Assign the return value to `groups`.

2. Implement the function `add_group`. The function defines two parameters:

   - country (dict): represents a country
   - group (dict): represents a World Bank group linked to the country

   The function combines a select list of key-value pairs drawn from the passed in `country` and `group` *if and only if* the `country['country_code']` and `group['country_code']` values are equal. Review the function's docstring to better understand its behavior.

   ❗ Each new dictionary must contain the following key-value pairs inserted in the following order:

   ```
   {
      "country_name": "Ghana",
      "country_code": "GHA",
      "group_name": "Sub-Saharan Africa",
      "group_code": "SSF",
      "income_group": "Lower middle income"
   }
   ```

3. Write a list comprehension that employs a nested loop in order to identify economies in the `countries` list that are located in Sub-Saharan Africa (group_code = "SSF"). Transform the elements to be included in the new list by calling the function `add_group` inside the comprehension. Assign the new list to a variable named `sub_saharan_africa`.

4. Call the function `write_json` and write the `sub_saharan_africa` list to a file named `stu-sub_saharan_africa.json`.