Exemplar_ Lists & tuples

November 23, 2023

Exemplar: Introduction to lists and tuples

0.1 Introduction

In this lab, you will practice creating, modifying, and working with data structures in Python. This will develop your knowledge of different kinds of data structures and the different operations that you can perform with them to answer questions about the data. This will help you prepare for projects you may encounter where you will need to use data structures to store and keep track of data.

For this activity, you are part of a research team that focuses on air quality, and you have received a few data points collected by the U.S. Environmental Protection Agency (EPA). Your goal is to store and organize this data, so that it can be accessed and updated easily.

In this lab, you'll be working specifically with the state names and county names of places that reported air quality index readings.

0.2 Tips for completing this lab

As you navigate this lab, keep the following tips in mind:

- `### YOUR CODE HERE ###` indicates where you should write code. Be sure to replace this with your own code before running the code cell.
- Feel free to open the hints for additional guidance as you work on each task.
- To enter your answer to a question, double-click the markdown cell to edit. Be sure to replace the "[Double-click to enter your responses here.]" with your own answer.
- You can save your work manually by clicking File and then Save in the menu bar at the top of the notebook.
- You can download your work locally by clicking File and then Download and then specifying your preferred file format in the menu bar at the top of the notebook.

0.3 Task 1: Create lists and use tuples

As you continue your work with the EPA, you want to create an ordered data structure containing pairs of values, which can be iterated over to gain insights from the data.

0.3.1 1a: Create lists

You'll work with state names and county names indicated below.

state_name	county_name
Arizona	Maricopa
California	Alameda
California	Sacramento
Kentucky	Jefferson
Louisiana	East Baton Rouge

- In this task, assign two variables:
  1. `state_names` - a `list` of each state in the `state_name` column in the table above, in order, as strings
  2. `county_names` - a `list` of each county in the `county_name` column in the table above, in order, as strings

[1]:
```
# 1. ### YOUR CODE HERE ###
state_names = ["Arizona", "California", "California", "Kentucky", "Louisiana"]

# 2. ### YOUR CODE HERE ###
county_names = ["Maricopa", "Alameda", "Sacramento", "Jefferson", "East Baton␣
↪Rouge"]
```

Hint

Refer to what you've learned about the syntax for instantiating lists.

0.3.2 1b: Use a loop to combine the lists into a single list of tuples

- Use `state_names` and `county_names` to:
  - Create a new list of tuples, where each tuple contains a pair of state name and county name.
  - Assign the new list to a variable called `state_county_tuples`.
- Print the result

Expected result:

```
[OUT] [('Arizona', 'Maricopa'),
       ('California', 'Alameda'),
       ('California', 'Sacramento'),
       ('Kentucky', 'Jefferson'),
       ('Louisiana', 'East Baton Rouge')]
```

[2]:
```
### YOUR CODE HERE ###
state_county_tuples = []
for i in range(len(state_names)):
    state_county_tuples.append((state_names[i], county_names[i]))
```

```
state_county_tuples
```

[2]:
```
[('Arizona', 'Maricopa'),
 ('California', 'Alameda'),
 ('California', 'Sacramento'),
 ('Kentucky', 'Jefferson'),
 ('Louisiana', 'East Baton Rouge')]
```

Hint 1

There are several ways to accomplish this task. One way could involve looping over the elements of each list and, with each iteration, assigning a tuple consisting of `(state_names[i], county_names[i])`, which could then be used to build the `state_county_tuples` list.

Hint 2

Use the `len()` function and the `range()` function to generate the appropriate number of iterations for a `for` loop.

Hint 3

1. Instantiate `state_county_tuples` as an empty list
2. Write a `for` loop to iterate over `range(len(state_names))`
3. With each iteration, append `(state_names[i], county_names[i])` to `state_county_tuples`

0.3.3 1c: Do the same thing using the zip() function

Python has a built-in function to make the above process much easier. It's called the `zip()` function. This function accepts any number of iterable objects as arguments. If the arguments are all of equal length, the function returns an iterator object of tuples, where each tuple contains `element[i]` of each argument.

You can then either loop over the iterator object or pass it to the `list()` function to unpack its values.

Refer to the zip() Python documentation for more information.

Here's an example:

[3]:
```python
# RUN THIS CELL
a = ['a', 'b', 'c']
b = [1, 2, 3]
c = zip(a, b)

print(c)
print(list(c))
```

```
<zip object at 0x7f00b4224910>
[('a', 1), ('b', 2), ('c', 3)]
```

Use the `zip()` function to generate the same output created in Task 1b.

1. Use `state_names` and `county_names` to:
    - Create a new list of tuples, where each tuple contains a pair of state name and county name.
    - Assign the new list to a variable called `state_county_zipped`.
2. Check that `state_county_zipped` is the same as `state_county_tuples`.

[4]:
```
# 1. ### YOUR CODE HERE ###
state_county_zipped = list(zip(state_names, county_names))

# 2. ### YOUR CODE HERE ###
state_county_zipped == state_county_tuples
```

[4]:
```
True
```

Hint 1

Refer to the example of the `zip()` function provided above.

Hint 2

Don't forget to convert the result of the zip operation to a list.

Hint 3

1. Use `list(zip(state_names, county_names))` and assign the result to `state_county_zipped`.

2. Test for equality with `state_county_tuples` using the `==` operator.

0.4 Task 2: Use list comprehension to convert to list of lists

Since tuples are immutable and can't be changed, converting tuples to lists is a practice data professionals use so they can make adjustments to the data, if necessary.

- Use a list comprehension to convert `state_county_tuples` from a list of tuples to a list of lists. Assign the result to a variable called `state_county_lists`.

- Print the result.

Expected result:

```
[OUT] [['Arizona', 'Maricopa'],
        ['California', 'Alameda'],
        ['California', 'Sacramento'],
        ['Kentucky', 'Jefferson'],
        ['Louisiana', 'East Baton Rouge']]
```

[5]:
```
### YOUR CODE HERE
state_county_lists = [list(i) for i in state_county_tuples]
state_county_lists
```

```
[5]: [['Arizona', 'Maricopa'],
      ['California', 'Alameda'],
      ['California', 'Sacramento'],
      ['Kentucky', 'Jefferson'],
      ['Louisiana', 'East Baton Rouge']]
```

Hint 1

Refer to what you've learned about list comprehension.

Hint 2

Remember that list comprehensions are like loops written in reverse:

```
[{operation to perform} for {element} in {iterable}]
```

Hint 3

There are multiple ways to perform this operation. The simplest uses the `list()` function on each element in `state_county_zipped`.

Another way is to use indexing to unpack each tuple into a list.

```
[[i[0], i[1]] for i in state_county_tuples]
```

0.5 Task 3: Unpacking an iterable

Data professionals often use the technique of unpacking to work with individual elements of iterable objects. As you continue in your work as an analyst, you are asked to iterate through your list of state/county pairs to identify only the counties in California.

As a refresher, here is the data you have been working with:

state_name	county_name
Arizona	Maricopa
California	Alameda
California	Sacramento
Kentucky	Jefferson
Louisiana	East Baton Rouge

0.5.1 3a: Unpacking in a loop

- Write a loop that unpacks each tuple in `state_county_tuples` and, if the state in the tuple is `California`, add the corresponding county to a list called `ca_counties`.

Expected output:

```
[OUT] ['Alameda', 'Sacramento']
```

[6]:
```
### YOUR CODE HERE ###
ca_counties = []
for state, county in state_county_tuples:
    if state=='California':
        ca_counties.append(county)
ca_counties
```

[6]:
```
['Alameda', 'Sacramento']
```

Hint 1

Refer to what you learned about unpacking tuples, `for` loops, conditional statements, and `list` methods.

Hint 2

To unpack a tuple in a loop, you must assign each of its elements to a variable. Each tuple in `state_county_tuples` has two elements, so instead of writing `for a in x`, you must write `for a, b in x`.

Hint 3

- Instantiate the `ca_counties` list before you begin writing your loop.
- With each iteration of the loop:
  1. Unpack each tuple in `state_county_tuples`:

  `for state, county in state_county_tuples:`

  2. Check whether `state` is California:

  `if state == 'California':`

  3. If it is, append `county` to `ca_counties`:

  `ca_counties.append(county)`

0.5.2 3b: Unpacking in a list comprehension

Now, use a list comprehension to accomplish the same thing as what you did in Task 3a.

- In a list comprehension, unpack each tuple in `state_county_tuples` and, if the state in the tuple is `California`, add the corresponding county to the list comprehension.
- Assign the result to a variable called `ca_counties`.
- Print `ca_counties`.

Expected output:

```
[OUT] ['Alameda', 'Sacramento']
```

[7]:
```
### YOUR CODE HERE ###
```

```
ca_counties = [county for (state, county) in state_county_tuples if␣
↪state=='California']
ca_counties
```

```
[7]: ['Alameda', 'Sacramento']
```

Hint 1

Refer to what you learned about list comprehension syntax. Also refer to previous tasks and examples in this lab.

Hint 2

It can be helpful to progress towards the goal in smaller steps. For example, first try to create a list comprehension that unpacks the `county` from each `state, county` pair, resulting in a list of *all* the counties.

If you can do that, then tweak the list comprehension by adding a conditional statement that only keeps the `county` if its corresponding `state` is `California`.

Hint 3

Consider the following syntax:

```
[___ for (___, ___) in _________ if __________]
```

0.6 Conclusion

What are your key takeaways from this lab?

- Lists and tuples are important iterable data types in Python that share many characteristics.
- Tuples in Python are useful for storing data in ordered sequences that are preserved and cannot be modified after creation.
- The `zip()` function is very useful for combining iterable objects element-wise.
- Tuples and lists can be unpacked.
- List comprehensions are quick and efficient ways to execute loop-like processes in a single line of code that results in a list.

Congratulations! You've completed this lab. However, you may not notice a green check mark next to this item on Coursera's platform. Please continue your progress regardless of the check mark. Just click on the "save" icon at the top of this notebook to ensure your work has been logged.

```
[ ]:
```