Working with Composite Data Types

Lab overview

A *composite* data type is any data type that comprises primitive data types. If you like food, you can visualize a composite data type as a turducken, which is a dish that consists of a chicken that is stuffed into a duck, which is stuffed into a turkey. In this lab, you will create a data type that consists of a string that is in a dictionary, which is in a list.

In this lab, you will:

- Use numeric data types
- Use string data types
- Use the dictionary data type
- Use the list data type
- Use a `for` loop
- Use the **print()** function
- Use the `if` statement
- Use the `else` statement
- Use the `import` statement

Estimated completion time

45 minutes

Accessing the AWS Cloud9 IDE

1. Start your lab environment by going to the top of these instructions and choosing **Start Lab**.

   A **Start Lab** panel opens, displaying the lab status.

2. Wait until you see the message *Lab status: ready*, and then close the **Start Lab** panel by choosing the **X**.

3. At the top of these instructions, choose **AWS**.

   The AWS Management Console opens in a new browser tab. The system automatically logs you in.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

> **Note:** If a new browser tab does not open, a banner or icon at the top of your browser typically indicates that your browser is preventing the site from opening pop-up windows. Choose the banner or icon, and choose **Allow pop ups**.

4. In the AWS Management Console, choose **Services** > **Cloud9**. In the **Your environments** panel, locate the **reStart-python-cloud9** card, and choose **Open IDE**.

   The AWS Cloud9 environment opens.

> **Note:** If a pop-up window opens with the message *.c9/project.settings have been changed on disk*, choose **Discard** to ignore it. Likewise, if a dialog window prompts you to *Show third-party content*, choose **No** to decline.

Creating your Python exercise file

5. From the menu bar, choose **File > New From Template > Python File**.

   This action creates an untitled file.

6. Delete the sample code provided from the template file.

7. Choose **File > Save As...**, provide a suitable name for the exercise file (for example, **composite-data.py**), and save it under the **/home/ec2-user/environment** directory.

Accessing the terminal session

8. In your AWS Cloud9 IDE, choose the **+** icon and select **New Terminal**.

   A terminal session opens.

9. To display the present working directory, enter `pwd` . This command points to **/home/ec2-user/environment**.

10. In this directory, locate the file that you created in the previous section.

Creating a car inventory data

Comma-separated values (CSV) is a file format that's used to store tabular data, such as data from a spreadsheet. You will work with the CSV file from the following block.

11. From the menu bar, choose **File > New File**.

    This action creates an untitled file.

12. Choose **File > Save As...**, and save the file as `car_fleet.csv`

13. Copy and paste the following text block into the **car_fleet.csv** file and save the file.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

17

```
vin,make,model,year,range,topSpeed,zeroSixty,mileage
TMX20122,AnyCompany Motors, Coupe, 2012, 335, 155, 4.1, 50000
TM320163,AnyCompany Motors, Sedan, 2016, 240, 140, 5.2, 20000
TMX20121,AnyCompany Motors, SUV, 2012, 295, 155, 4.7, 100000
TMX20204,AnyCompany Motors, Truck, 2020, 300, 155, 3.5, 0
```

Tip: If a pop-up window opens with the message *Native Clipboard Unavailable*, use the keyboard, not the browser menu, to perform copy and paste actions. For example, on Windows, use CTRL+C and CTRL+V to copy and paste, respectively. On Mac, use Command+C and Command+V.

Creating a car inventory program

Defining the dictionary

You will read in the file by using a module called `csv`. Additionally, you will make a deep copy of the data to store in memory by using a module called `copy`.

14. From the navigation pane of the IDE, choose (double-click) the **.py** file that you created in the previous *Creating your Python exercise file* section.
15. First, import the modules that you will use:

```
import csv
import copy
```

16. Next, define a dictionary that will serve as your composite type for reading the tabular data:

```
myVehicle = {
    "vin" : "<empty>",
    "make" : "<empty>" ,
    "model" : "<empty>" ,
    "year" : 0,
    "range" : 0,
    "topSpeed" : 0,
    "zeroSixty" : 0.0,
    "mileage" : 0
}
```

17. You will use a `for` loop to iterate over the initial keys and values of the dictionary.

```
for key, value in myVehicle.items():
    print("{} : {}".format(key,value))
```

> **Note:** The **items()** function belongs to the dictionary data type. The **items()** function tells the `for` loop to traverse the collection owned by the dictionary data type.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17

18. Define an empty list to hold the car inventory that you will read:

```
myInventoryList = []
```

19. Save the file.

Copying the CSV file into memory

You will read in the data from disk (hard drive) and make an in-memory (random access memory, or RAM) copy. In a computer, a *hard drive* stores data long term, including when the power is turned off. RAM refers to temporary memory that is faster, but it is erased when the computer's power is turned off.

You will be introduced to the `with open` syntax statement, which keeps a file open while you read data. It will automatically close the CSV file when the code inside the `with` block is finished running.

You will also use a new way of formatting a string. Instead of using double quotation marks and **.format** to pass in the variables, you can use a single quotation mark and write in the variables between the "{}" symbols.

Finally, **csv.reader()** is a function that you are using from the `csv` library that you imported with the `import csv` statement.

Most of the rest of the code should be familiar.

Now, return to the Python file:

20. Enter the following code:

```
with open('car_fleet.csv') as csvFile:
    csvReader = csv.reader(csvFile, delimiter=',')
    lineCount = 0
    for row in csvReader:
        if lineCount == 0:
            print(f'Column names are: {", ".join(row)}')
            lineCount += 1
        else:
            print(f'vin: {row[0]} make: {row[1]}, model: {row[2]}, year: {row[3]}, range: {ro
w[4]}, topSpeed: {row[5]}, zeroSixty: {row[6]}, mileage: {row[7]}')
            currentVehicle = copy.deepcopy(myVehicle)
            currentVehicle["vin"] = row[0]
            currentVehicle["make"] = row[1]
            currentVehicle["model"] = row[2]
            currentVehicle["year"] = row[3]
            currentVehicle["range"] = row[4]
            currentVehicle["topSpeed"] = row[5]
            currentVehicle["zeroSixty"] = row[6]
            currentVehicle["mileage"] = row[7]
            myInventoryList.append(currentVehicle)
            lineCount += 1
    print(f'Processed {lineCount} lines.')
```

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Though this code seems like a large amount of code to process, it mostly comprises statements that you saw in earlier labs. You have a `for` loop with an `if-else` statement followed by a `print()` statement at the end.

17 However, the following line needs further explanation:

```
currentVehicle = copy.deepcopy(myVehicle)
```

By default, Python does a *shallow copy* of complex data types. A shallow copy refers, or points, to the storage location of the *myVehicle* dictionary variable. Without this line, you would have only one storage box, and only the last item in the list would be copied into memory. This line makes sure that new storage boxes are created in memory to store the new tabular data that is being read.

Printing the car inventory

You will finish the Python script by printing the car inventory from the *myInventoryList* variable.

21. Return to the Python script, and enter the following code:

```
for myCarProperties in myInventoryList:
    for key, value in myCarProperties.items():
        print("{} : {}".format(key,value))
        print("-----")
```

22. Save the file.
23. Choose the **Run** button in the menu bar to run the program.
24. Confirm that the script runs correctly and that the output displays as you expect it to.
25. Review the code for reading in the tabular data from the CSV file one more time. Understanding this section of the code is key to this lab.

Congratulations! You have worked with composite data types in Python.

End Lab

Congratulations! You have completed the lab.

26. Choose **End Lab** at the top of this page, and then select Yes to confirm that you want to end the lab.

    A panel indicates that *DELETE has been initiated... You may close this message box now.*

27. A message *Ended AWS Lab Successfully* is briefly displayed, indicating that the lab has ended.

Additional Resources

For more information about AWS Training and Certification, see https://aws.amazon.com/training/ (https://aws.amazon.com/training/).

Your feedback is welcome and appreciated. If you would like to share any suggestions or corrections, please provide the details in our AWS Training and Certification Contact Form (https://support.aws.amazon.com/#/contacts/aws-training).

- ○ Yes
- ○ No

< Rubric: 6 - Composite Data Types | Points: 0 >

Previous Next