3.3 Lesson Plan - Python Deeper Dive

Overview

Today's class will dive deeper into some of Python's other capabilities that will be used throughout the rest of the course.

Instructor Notes

- The first part of today's class will continue to move along with Python, starting with a warm up exercise then moving into dictionaries, list comprehensions and functions.

- Some students may still be struggling with the pace and concepts of Python. Make sure TAs are circulating and giving extra attention to the students who need it. If class is moving ahead of schedule there may be extra time for students to either tackle some additional attached problems, or serve as a review for those that are struggling.

- Lastly as a reminder, slideshows *(when applicable)* are for instructor use only - when distributing slides to students, please first export the slides to a PDF file. You may then distribute the PDF file through Slack.

Sample Class Video *(Highly Recommended)*

- View an example class lecture visit *(Note: video may not reflect latest lesson plan)*

Class Objectives

- Students will be able to add, commit, and push code up to GitHub from the command line.
- Students will be able to create and use Python dictionaries.
- Students will be able to read data in from a dictionary.
- Students will be able to use List Comprehensions.
- Students will be able to write and reuse Python functions.
- Students will have a firm understanding of coding logic and reasoning.

1. Instructor Do: Welcome Students (0:05)

- Welcome students to class and let them know today's class will dive deeper into more of Python's capabilities. Today's class will also finish with more usage of Git and Github, particularly how to use the command line to add and commit files.

2. Students Do: Cereal Cleaner (0:10)

- To start today's class off, students will be creating an application that reads in cereal data from a CSV and then prints only those cereals that have more than 5 grams of fiber in them.
- Open up **`CerealCleaner`** within the terminal and run the code to show the end results of the application.

```
['All-Bran', 'K', 'C', '70', '4', '1', '260', '9', '7', '5', '320', '25',
'3', '1', '0.33', '59.425505']
['All-Bran with Extra Fiber', 'K', 'C', '50', '4', '0', '140', '14', '8',
'0', '330', '25', '3', '1', '0.5', '93.704912']
['Bran Flakes', 'P', 'C', '90', '3', '0', '210', '5', '13', '5', '190',
'25', '3', '1', '0.67', '53.313813']
['Fruit & Fibre Dates; Walnuts; and Oats', 'P', 'C', '120', '3', '2', '160',
'5', '12', '10', '200', '25', '3', '1.25', '0.67', '40.917047']
['Fruitful Bran', 'K', 'C', '120', '3', '0', '240', '5', '14', '12', '190',
'25', '3', '1.33', '0.67', '41.015492']
['Post Nat. Raisin Bran', 'P', 'C', '120', '3', '1', '200', '6', '11', '14',
'260', '25', '3', '1.33', '0.67', '37.840594']
['Raisin Bran', 'K', 'C', '120', '3', '1', '210', '5', '14', '12', '240',
'25', '2', '1.33', '0.75', '39.259197']
```

- **Files**
  - **`cereal.csv`**
  - **`cereal_bonus.csv`**
- **Instructions**
  - Read through `cereal.csv` and find the cereals that contain five grams of fiber or more, printing the data from those rows to the terminal.
- **Hint**
  - Every value within the csv is stored as a string and certain values have a decimal. This means that they will have to be cast to be used.
  - `csv.reader` begins reading the csv file at the first row. Explain that `next(csv_reader, None)` will skip the header row.
  - Integers are whole numbers and, as such, cannot contain decimals. Decimal numbers will have to be cast as a float or double .
- **Bonus**

- Try the following again but this time using `cereal_bonus.csv` , which does not include a header.

3. Instructor Do: Review Cereal Cleaner (0:05)

- Open **`cereal_solved.py`** and walk through the code with the class, answering whatever questions students have.

- Key points to cover when discussing this activity:

  - The importing of dependencies is done before anything else in the code. This is done to keep the code clean and makes it easier to read/understand in the future.
  - Point out how the variable `cereal_csv_path` is named in a very self-explanatory way. This is to ensure that future developers - including those who originally wrote this application - know precisely what this variable references.
  - The same is true of `csv_reader` which is used to hold the data read in from the CSV file.
  - The `next()` function is placed before the for loop to skip over the first row of data - the header.

```
import os
import csv

cereal_csv = os.path.join("..", "Resources", "cereal.csv")

# Open and read csv
with open(cereal_csv, newline="") as csvfile:
    csvreader = csv.reader(csvfile, delimiter=",")

    # Read the header row first (skip this part if there is no heade
    csv_header = next(csvfile)
    print(f"Header: {csv_header}")

    # Read through each row of data after the header
    for row in csvreader:

        # Convert row to float and compare to grams of fiber
        if float(row[7]) >= 5:
            print(row)
```

    - In order to check through all of the rows in the CSV file and find those that have more than five grams of fiber in them, a for loop containing an if statement is used.
    - Looking through the CSV file, one would find that the fiber column is stored within the 8th column. This means the if statement must check values stored at index 7.

- Check with the class to see what methods they used to come up with their solutions before moving onto the next activity.

End of Section