pyt

Full Name: **ANSWER KEY** _____

Email Address (UW Net ID): _____@uw.edu

Section: _____

CSE 160 Spring 2024 - Final Exam

Instructions:

- You have **110 minutes** to complete this exam. This exam has been written with an expectation of taking approximately 50-60 minutes, however you have the full 110 minutes to complete it.
- The exam is **closed book**, including no calculators, computers, phones, watches or other electronics.
- You are allowed a single sheet of notes for yourself.
- We also provide a syntax reference sheet.
- Turn in *all sheets* of this exam, together and in the same order when you are finished.
- A couple of extra pages have been provided to you at the end of the exam. You may use these as scratch paper.
- When time has been called, you must put down your pencil and stop writing.
    - **Points will be deducted if you are still writing after time has been called.**
- You may only use parts and features of Python that have been covered in class up to this point.
- You may ask questions by raising your hand, and a TA will come over to you.

Good luck!

Question	**Topic**
Question 1	Errors
Question 2	Testing
Question 3	Reading Code
Question 4	Dictionaries
Question 5	Classes

Question 1) While going to your TA's office hours for HW 6, you ask for help in understanding the following error message (the full directory is truncated by '. . .'):

```
Traceback (most recent call last):
  File "...\homework6\fishing.py", line 213, in <module>
    d = parse_data(in_file)
        ^^^^^^^^^^^^^^^^^^
  File "...\homework6\fishing.py", line 37, in parse_data
    value = row[year]
            ~~~^^^^^^
KeyError: 1995
```

For reference this is the .csv file which you passed in as your data:

```
country,country code,measure,1995,1996,1997,1998,1999,2000
Canada,CAN,consumption,22.62,23.41,23.23,24.1,24.97,23.49
Canada,CAN,farmed,65207,72376,81676,91046,112916,127665
Canada,CAN,population,29137000,29442000,29746000,30020000,30270000,30530000
Canada,CAN,wild caught,948410,1195329,1274391,1291796,1277027,1121610
```

A. What is the name of the function which contains the error? **parse_data**

Here are some numbered lines from the function which caused the error, with most of the actual code not shown.

```
30|  with open(filename) as csv_file:
31|      csv_reader = csv.DictReader(csv_file)
32|      ...
33|      ...
34|      ...
35|      ...
36|      ...
37|      ... value = row[year]
38|      ...
39|      ...
40|      ...
```

B. At the line where the error occurred, write out the code that caused the error after the ... (don't worry about indentation). You should only fill in **one** line.

C. Given what you know about `csv.DictReader` and the data shown on the previous page, why might this error have occurred? (hint: what is the default data type of the values inside a `csv.DictReader` object?)

The key "1995" (as an integer) was not found in the parsed row of the CSV file. This is because csv.DictReader by default gives all keys and values as strings.

D. In two lines or less, write code that would also produce a `KeyError`. It does not have to be relevant to this problem, any `KeyError` will suffice.

```
d = {'a': 1}
print(d['b'])
```

Phew! Now that you have figured out this error hopefully your TA doesn't put a question about HW 6 on the final…

Question 2) During the Midterm, you helped a vet clinic write a function for their dog's weights. For reference, it had the following header and docstring:

```
def average_weight(patient_data):
    '''
    The function average_weight() finds the average weight of dogs in order to
    track the changes in canine obesity over time. It takes in a list of
    dictionaries (patient_data), looks at dogs' weights, and returns a float
    that represents the average dog weight. These dictionaries are in the form
    {"Patient": -, "Weight": -, "Species": -}. With no dogs present in the
    dictionary, return 0. Valid species values are "Canine", "Feline", and
    "Avian".
    '''
```

This function has not been tested though and now needs to be. Write three tests along with an explanation of what each test is focusing on. Each test must test distinctly different aspects of the function.

Note: Inputs and outputs must be specified in the test and a clear and precise reason must be given to achieve full points.

Test	Reason
`assert average_weight([{'Patient': 'Lucky', 'Height': 23, 'Weight': 30, 'Species': 'Dog'}] == 30`	**Test with only one patient**
`assert [{'Patient': 'Lucky', 'Weight': 30, 'Species': 'Dog'}, {'Patient': 'Shadow', 'Weight': 25, 'Species': 'Cat'}] == 30`	**Test with multiple species**
`assert [{'Patient': 'Shadow', 'Weight': 25, 'Species': 'Cat'}] == 0`	**Test with no dogs present**

Question 3) Consider the function below. It has a handful of issues, bugs, or errors. On the following page, identify the line number that has a problem, state why it's an error, and describe how to fix it.

```
def determine_winner(votes):
    '''
    Determine the Eurovision winner based on the votes from
    different countries.

    Inputs:
    votes (dict): A dictionary where keys are country names and values
                  are dictionaries of contestant names and the
corresponding
                  points given by that country to them.
    Example Input:
     votes = {
         'France': {'Contestant_A': 12, 'Contestant_B': 10},
         'Germany': {'Contestant_A': 8, 'Contestant_C': 6},
         'Italy': {'Contestant_B': 7, 'Contestant_C': 4, 'Contestant_A':
     2}
     }

    Returns:
        The name of the contestant with the highest total score (string)
    '''

1.  total_scores = {}
2.  winner = None
3.  max_vote = None
4.
5.  for scores in votes.items():
6.      for contestant in scores():
7.          points = scores[contestant]
8.          if contestant not in total_scores:
9.              total_scores[contestant] = points
10.            total_scores[contestant] += points
11.
12.    for contestant in total_scores.keys():
13.         temp_score = total_scores[contestant]
14.         if (max_score < temp_score or max_score = None)
15.            max_score = temp_score
16.            winner = contestant
17.
18.    return winner
```

(Answers here are for Question 3, outlined on the previous page.)

Issues:

- **Line 3: max_vote is not used elsewhere. Adjust so either it is all max_score or all max_vote**
- **Line 5: when unpacking a tuple (using .items), must provide two variables. Cannot use scores in line 6 otherwise.**
- **Line 6: dictionary is not callable, so cannot use scores()**
    - **Remove the () or use .keys()**
- **Line 9/10: when new keys are created, the number of points will be added twice. Could potentially result in incorrect totals and an incorrect winner.**
    - **Inaccurate total but correct output**
- **Line 10: incorrect indentation (move left to align with line 8)**
- **Line 14: cannot compare None type, so must reverse order of logic checks**
- **Line 14: missing colon after if statement**
- **Line 14: cannot use = None to compare, must use “is None”**

Question 4) Suppose you are a quantitative financial analyst interested in the overall performance of various stocks in your stock profile.

Consider an input dictionary mapping stock codes (strings) to their values (list of floats) over the past *N* months, where *N* is any number of months (and could be different for every stock). Write a function `stock_stats(stock_dict)` that returns a *new* dictionary mapping stock codes to a dictionary containing the all-time mean, median, and mean absolute deviation. The new dictionary should be sorted alphabetically by key. (**Note:** even though dictionaries are considered "unordered," Python will maintain the keys in the order that they were inserted in.)

For example,

```
stock_dict = {"AAPL": [188.68, 191.73, 173.52],
              "MSFT": [374.64, 407.48, 430.52],
              "GOOG": [175.51, 176.57, 176.44]}

stock_stats(stock_dict)
```

Output:

```
{
    'AAPL': {'mean': 184.64, 'median': 188.68, 'absdev': 7.41},
    'GOOG': {'mean': 176.17, 'median': 176.44, 'absdev': 0.44},
    'MSFT': {'mean': 404.21, 'median': 407.48, 'absdev': 19.71}
}
```

Mean absolute deviation is calculated as the sum of the differences between each value in the list and the list's mean, over the number of values in the list. More formally:

$$\frac{\sum_{0}^{n-1} \left| x_i - m \right|}{n}$$

where x_i is the "i-th" (by index) item from the list, m is the list's mean, and n is the number of items in the list. For this problem, you may use the `sum` function to get the sum of a list.

Write your answer to this question on the following page.

```
def stock_stats(stock_dict):
    # Your Answer for Question 4 goes here:

        stats = {}
        for k in sorted(stock_dict):
            vals = stock_dict[k]
            n = len(vals)
            mean = sum(vals) / n

            stats[k] = {
                "mean": mean,
                "median": sorted(vals)[n // 2],
                "absdev": sum([abs(x - mean) for x in vals]) / n,
            }
        return stats
```

Question 5:

You are given the following class ‘Bookshop’ initialized with a name, city, and starting inventory. The class also contains methods (member functions) that allow you to add, remove, update, and search for books.

```
class Bookshop():
   def __init__(self, name, city):
       '''
       Creates a Bookshop with a given name and city, both strings.
       '''
       self.name = name
       self.city = city
       self.inventory = []
       print('Started a new bookshop in', city + ':', name + '!')

   def remove_book(self, title):
       '''
       Removes a book from the inventory by title.
       Parameters: title: Title of the book to remove.
       Returns: True if title is found, otherwise returns False.
       '''
       for book in self.inventory:
           if book['title'] == title:
               self.inventory.remove(book)
               print('Book "' + title + '" removed from the inventory.')
               return True
       print('Book "' + title + '" not found in the inventory.')
       return False

   def add_book(self, title, author, price, stock):
       '''
       Adds a new book to the inventory.

       Parameters:
           title: Title of the book.
           author: Author of the book.
           price: Price of the book.
           stock: Number of copies available.
       '''
       self.inventory.append({"title": title, "author": author,
                              "price": price, "stock": stock})
       print('Book "' + title + '" by, author, 'added to the inventory.')

   def list_inventory(self):
       '''
       Lists all the books in the inventory.
       Returns: A list of all books in the inventory.
       '''
       if not self.inventory:
           return 'The inventory is empty.'
       return self.inventory
```

A) Many books in the book store will be written by the same author, and we might want to know which authors are represented in this store. Write a new method, `unique_authors`, that returns a list of the authors sorted by last name. For this problem, you may assume that all author names are in the format `"First Last"`, that is: only a single space between given and surname, the latter being what to sort by.

```
def unique_authors(self):
    '''
    Returns a list, ordered by last name, of the unique author names.
    '''
    # Your solution to Question 5(a) goes here:

     authors = set()
     for book in self.inventory:
         authors.add(book['author'])
     return sorted(authors, key=sort_key)


   def sort_key():
        return x.split()[-1]
```

B) Write a new method, `update_stock`, for the Bookshop class that updates the inventory quantity for a given title. The function header and docstring are provided below. In addition to the input/return requirements specified in the docstring, `update_stock` needs to:

- Change a book's quantity (stock) to the new value, then return True.
- Return False if the given title doesn't exist in the inventory.
- Print "`Stock for book "{title}" updated to {new_stock}.`" when the update is successful (replacing `{title}` and `{new_stock}` with those variables respectively).
- When the update is not successful (e.g., the book is not found), print "`Book "{title}" not found in the inventory.`" replacing `{title}` with the title of the book.

```
def update_stock(self, title, new_stock):
    '''
    Updates the stock of a specific book.
    Returns: True if update is successful (title exists),
             otherwise returns False (title does not exist)

    Parameters:
    title: Title of the book.
    new_stock: New stock quantity.
    '''
    # Your solution to Question 5(b) goes here:

     for book in self.inventory:
         if book['title'] == title:
             book['stock'] = new_stock
             print(f'Stock for book "{title}" updated to {new_stock}.')
             return True
     print(f'Book "{title}" not found in the inventory.')
     return False
```

C) For this problem, write code that would produce the following output:

```
Started a new bookshop in Seattle: Ada's Technical Books!
Book "The Plot Thickens" by Paige Turner added to the inventory.
Book "Bug Off" by Deb Ugger added to the inventory.
[{'title': 'The Plot Thickens', 'author': 'Paige Turner', 'price': 17.99,
'stock': 10}, {'title': 'Bug Off', 'author': 'Deb Ugger', 'price': 20.0,
'stock': 5}]
['Paige Turner', 'Deb Ugger']
Stock for book "Bug Off" updated to 2.
Book "The Plot Thickens" removed from the inventory.
[{'title': 'Bug Off', 'author': 'Deb Ugger', 'price': 20.0, 'stock': 2}]
```

That is,

- Creates a new Bookshop
- Adds two books to the inventory.
- Prints the bookshop's inventory.
- Prints the known unique authors.
- Makes a couple of inventory changes.
- Prints the inventory again.

Your code must use the `Bookshop` class as defined at the beginning of this problem. You may assume that the functions you wrote in parts A and B of this problem are complete and correct (no matter what you actually wrote for them). This answer does not require any new functions or methods.

```
bookshop = Bookshop("Ada's Technical Books", "Seattle")

bookshop.add_book("The Plot Thickens", "Paige Turner", 17.99, 10)
bookshop.add_book("Bug Off", "Deb Ugger", 20.00, 5)

print(bookshop.list_inventory())
print(bookshop.unique_authors())
bookshop.update_stock("Bug Off", 2)
bookshop.remove_book("The Plot Thickens")
print(bookshop.list_inventory())
```