homework3

January 24, 2024

Introduction to Computer Programming for the Physical Sciences

Joseph F. Hennawi

Winter 2024

Open a new Jupyter notebook

Name your notebook with your name and Homework 1

Open a Markdown cell at the top and write your name and Homework 1

Open a Markdown cell before each problem and write e.g. Problem 1, Problem 2(a), etc.

Please abide by the Policy and Guidelines on Using AI Tools

Once you finish the problems: 1) Restart the Python kernel and clear all cell outputs. 2) Rerun the notebook from start to finish so that all answers/outputs show up. 3) Save your notebook as a single .pdf file and upload it to Gradescope on Canvas by the deadline. No late homeworks will be accepted except for illness accompanied by a doctor's note.

1 Homework 3

1.1 Problem 1: Working with Python Data Containers

You are tasked with developing a program to manage information on business trips made by employees to various cities where your company has offices. Unfortunately, your company did not adopt a uniform method for recording this information, so you have to develop a program that can handle different types of data. Specifically, the business trip data is a python list containing a combination of dictionaries and tuples, where each dictionary or tuple represents information about a specific trip. The relevant data are the employee's name, destination city, round-trip mileage, and the date of the trip. The dictionaries contain the information in key-value pairs (i.e. name, destination, mileage, date), while the tuple contains the same information ordered sequentially (i.e. tuple_data = (name, destination, mileage, date)). The employee data is given in the code cell below:

[1]:
```
trips = [
    {'name': 'Alice', 'destination': 'New York', 'mileage': 200.0, 'date':␣
 ↪'2022-01-01'},
    ('Bob', 'Chicago', 300, '2022-02-01'),
    {'name': 'Charlie', 'destination': 'Los Angeles', 'mileage': 400, 'date':␣
 ↪'2022-03-01'},
    ('David', 'San Francisco', 500.0, '2022-04-01'),
    {'name': 'Eve', 'destination': 'Seattle', 'mileage': 620, 'date': '2022-05-01'},
    ('Frank', 'Boston', 705, '2022-06-01'),
    {'name': 'Grace', 'destination': 'Denver', 'mileage': 810, 'date': '2022-07-01'},
    ('Henry', 'Austin', 911, '2022-08-01'),
    {'name': 'Ivy', 'destination': 'Atlanta', 'mileage': 1000.0, 'date': '2022-09-01'},
    ('Jack', 'Miami', 1112.0, '2022-10-01'),
    {'name': 'Kate', 'destination': 'Dallas', 'mileage': 1200.0, 'date': '2022-11-01'},
    ('Liam', 'Houston', 134, '2022-12-01'),
    {'name': 'Mia', 'destination': 'Phoenix', 'mileage': 1400.0, 'date': '2023-01-01'},
    ('Noah', 'Las Vegas', 1526.0, '2023-02-01'),
    {'name': 'Olivia', 'destination': 'San Diego', 'mileage': 1600.0, 'date': '2023-03-01'},
    ('Peter', 'Portland', 1700, '2023-04-01'),
    ('Alice', 'Boston', 1322.0, '2023-01-01'),
    ('Bob', 'Los Angeles', 400, '2023-02-01'),
    {'name': 'Charlie', 'destination': 'Chicago', 'mileage': 500, 'date': '2023-03-01'},
    ('David', 'Miami', 600.0, '2023-04-01'),
    {'name': 'Eve', 'destination': 'Dallas', 'mileage': 700.0, 'date': '2023-05-01'},
    ('Frank', 'Houston', 800, '2023-06-01'),
    {'name': 'Grace', 'destination': 'Phoenix', 'mileage': 900.0, 'date': '2023-07-01'},
    ('Henry', 'Las Vegas', 1000, '2023-08-01'),
    {'name': 'Ivy', 'destination': 'San Diego', 'mileage': 1100.0, 'date': '2023-09-01'},
    ('Jack', 'Portland', 1200, '2023-10-01'),
    {'name': 'Kate', 'destination': 'New York', 'mileage': 1300.0, 'date': '2023-11-01'},
    ('Liam', 'Austin', 1400, '2023-12-01'),
    {'name': 'Mia', 'destination': 'Denver', 'mileage': 1500.0, 'date': '2024-01-01'},
    ('Noah', 'Atlanta', 1600, '2024-02-01'),
    {'name': 'Olivia', 'destination': 'Seattle', 'mileage': 1700.0, 'date': '2024-03-01'},
    ('Peter', 'San Francisco', 1800, '2024-04-01'),
    {'name': 'Alice', 'destination': 'Los Angeles', 'mileage': 210.0, 'date': '2022-02-01'},
    ('Bob', 'San Francisco', 310, '2022-03-01'),
    {'name': 'Charlie', 'destination': 'Seattle', 'mileage': 410, 'date':␣
↪'2022-04-01'},
    ('David', 'Boston', 510.0, '2022-05-01'),
    {'name': 'Eve', 'destination': 'Denver', 'mileage': 630, 'date':␣
↪'2022-06-01'},
    ('Frank', 'Austin', 715, '2022-07-01'),
    {'name': 'Grace', 'destination': 'Atlanta', 'mileage': 820, 'date':␣
↪'2022-08-01'},
    ('Henry', 'Miami', 921, '2022-09-01'),
    {'name': 'Ivy', 'destination': 'Dallas', 'mileage': 1010.0, 'date':␣
↪'2022-10-01'},
    ('Jack', 'Houston', 1122.0, '2022-11-01'),
    {'name': 'Kate', 'destination': 'Phoenix', 'mileage': 1210.0, 'date':␣
↪'2022-12-01'},
    ('Liam', 'Las Vegas', 144, '2023-01-01'),
    {'name': 'Mia', 'destination': 'San Diego', 'mileage': 1410.0, 'date':␣
↪'2023-02-01'},
    ('Noah', 'Portland', 1536.0, '2023-03-01'),
    {'name': 'Olivia', 'destination': 'San Francisco', 'mileage': 1610.0,␣
↪'date': '2023-04-01'},
    ('Peter', 'New York', 1710, '2023-05-01'),
    {'name': 'Anne', 'destination': 'Seattle', 'mileage': 220.0, 'date':␣
↪'2022-03-01'},
    ('Gerald', 'Boston', 320, '2022-04-01'),
    {'name': 'Charlie', 'destination': 'Denver', 'mileage': 420, 'date':␣
↪'2022-05-01'},
    ('Dirk', 'Austin', 520.0, '2022-06-01'),
    {'name': 'Eve', 'destination': 'Atlanta', 'mileage': 640, 'date':␣
↪'2022-07-01'},
    ('Joe', 'Miami', 725, '2022-08-01'),
    {'name': 'Grace', 'destination': 'Dallas', 'mileage': 830, 'date':␣
↪'2022-09-01'},
    ('Dirk', 'Houston', 931, '2022-10-01'),
    {'name': 'Ivy', 'destination': 'Phoenix', 'mileage': 1020.0, 'date':␣
↪'2022-11-01'},
    ('Henry', 'Las Vegas', 1132.0, '2022-12-01'),
    {'name': 'Kate', 'destination': 'San Diego', 'mileage': 1220.0, 'date':␣
↪'2023-01-01'},
    ('Alison', 'Portland', 154, '2023-02-01'),
    {'name': 'Mia', 'destination': 'San Francisco', 'mileage': 1420.0, 'date':␣
↪'2023-03-01'},
    ('Noah', 'New York', 1546.0, '2023-04-01'),
    {'name': 'Olivia', 'destination': 'Los Angeles', 'mileage': 1620.0, 'date':␣
↪'2023-05-01'},
    ('Peter', 'Chicago', 1720, '2023-06-01')
]
```

Write a function called manage trips consistent with the behavior in the following documentation string.

```python
def manage_trips(trips, append_new=None, filter_dict=None, display=False):
    """
    Manage business trip data.

    Parameters
    ----------
    trips : list
        List containing dictionaries and tuples, where each dictionary or tuple represents inf
    append_new : list or dict or tuple, optional
        If provided, append this new trip, OR this list of new trips to the list of trips and 
    filter_dict : dict, optional
        If provided, return a list of trips that match the key-value pairs in this dictionary.
    display : bool, optional
        If True, print the output_list of trips (a formatting example is provided in Examples)

    Returns
    -------
    output_list: list of dicts
        List of trips that match the input parameters. Note that the output_list should
        contain dictionaries, even if the intput trips or append_new trips contains tuples.

    Notes
    -----
    Note that only one of the parameters append_new, name, destination, or date can be provide

    Examples
    --------

    >>> alice_trips = manage_trips(trips, filter_dict={'name': 'Alice'}, display=True)

    Returns the list:

    alice_trips = [{'name': 'Alice', 'destination': 'New York', 'mileage': 200.0, 'date': '202
                   {'name': 'Alice', 'destination': 'Boston', 'mileage': 1322.0, 'date': '2022
                   {'name': 'Alice', 'destination': 'Los Angeles', 'mileage': 210.0, 'date': '


    and prints the output formatted as follows

            Name    | Destination | Mileage | Date
    ---------------------------------------------------------
            Alice   |  New York   |    200   | 2022-01-01
            Alice   |   Boston    |   1322   | 2022-01-02
            Alice   | Los Angeles |    210   | 2022-02-01
    ---------------------------------------------------------



    """
```

A few points: - Make sure that the output is formatted as shown in the example above. - Make sure that if multiple trip parameters (name, destination, date) are provided in filter_dict, the output is a list of trips that match ALL of those parameters. - Your code should perform error checking if the filter_dict contains a key that is not 'name', 'destination', or 'date'. - Note that for append_new, the input can be a single trip (i.e. a dictionary or tuple) or a list of trips (i.e. a list of dictionaries or tuples). - Make sure that your code still works if the input is a new list of trips modified by the append_new option.

Test your code on the following example usage cases:

```
# Test appending a new trip
new_trip = {'name': 'Zoe', 'destination': 'San Francisco', 'mileage': 2000.0, 'date': '2023-02
new_trip_list = manage_trips(trips, append_new=new_trip)
print(new_trip_list[-1])  # Should print the new trip

# Test that the code works on the new trip list, should print trips by Zoe
filter_dict = {'name': 'Zoe'}
filtered_trips = manage_trips(new_trip_list, filter_dict=filter_dict, display=True)

# Test filtering trips by destination, should print trips to New York
filter_dict = {'destination': 'New York'}
filtered_trips = manage_trips(trips, filter_dict=filter_dict, display=True)

# Test filtering trips by name, should print trips by Alice
filter_dict = {'name': 'Alice'}
filtered_trips = manage_trips(trips, filter_dict=filter_dict, display=True)


# Test filtering trips by date, should print trips on 2023-04-01
filter_dict = {'date': '2023-04-01'}
filtered_trips = manage_trips(trips, filter_dict=filter_dict, display=True)

# Test filtering trips by name and destination, should print trips by Alice to New York
filter_dict = {'name': 'Alice', 'destination': 'New York'}
filtered_trips = manage_trips(trips, filter_dict=filter_dict, display=True)
```

1.2 Problem 2: The Fibonacci Sequence

The Fibonacci sequence begins

$$0,\ 1,\ 1,\ 2,\ 3,\ 5,\ 8,\ 13,\ 21,\ 34,\ 55,$$

and is defined by the recursion relation

$$x_{n+1} = x_n + x_{n-1}$$

with $x_0 \equiv 0$ and $x_1 \equiv 1$. What is the largest Fibonacci number less than 10^6?

1.3 Problem 3: Cryptography

Cryptography is the study of how to make messages secret or how to read secret messages. A very simple encryption technique is called the Caesar cipher, which you can read up more about here. The basic idea is that each letter is replaced by a letter that is a certain number of letters away, so for example if the shift was 2, then A would become C, B would become D, etc., (and Z will become B).

Write a function that given a string and a shift, will return the encrypted string for that shift. Note that the same function should be able to decrypt a message, by passing it the negative shift. Your function should only accept and return lowercase letters (i.e. ignore the case) and spaces should not be changed.

- Use your function to decrypt following message, `'the pbatenghyngvbaf lbh unir fhpprrqrq va qrpelcgvat gur fgevat'` which was encrypted with a shift of 13.
- Decrypt the following message and find the shift used to encrypt it: `'gwc uivioml bw nqvl bpm zqopb apqnb'`

Hint: You may find the following functions from the Python standard library useful: `ord()`, `chr()`

1.4 Problem 4: Higher Order Derivatives

We can represent the n^{th} derivative of $f(x)$ as $f^{(n)}(x)$ where $f^0(x) \equiv f(x)$, $f^1(x) \equiv f'(x)$, $f^2(x) \equiv f''(x)$, etc.

Last week we learned how to compute numerical derivatives of a function $f(x)$, and found that the symmetric difference formula

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

performs significantly better than the forward difference formula. Code up a Python function to compute the n^{th} derivative of the function $f(x)$ using the symmetric difference formula and the recursion relation

$$f^{(n)}(x) \approx \frac{f^{(n-1)}(x+h) - f^{(n-1)}(x-h)}{2h}$$

Your function should be consistent with the following documentation string:

```
def deriv(f, x, n, h=1e-2):
    """
    Compute the nth derivative of a function f(x) using the symmetric difference formula.

    Parameters
    ----------
    f : function
        Function to take the derivative of.
```

```
x : float
    Point at which to evaluate the derivative.
n : int
    Order of the derivative.
h : float, optional
    Step size. The default is 1e-2.

Returns
-------
f^(n)(x): float
    The nth derivative of f(x) evaluated at x.

"""
```

- Test your function by computing the derivatives of e^x at $x = 1$ for $n = 0, 1, 2, 3, 4, ...10$ and compare to the analytic result. Print our your results to 16 decimal places. You can use the numpy function `np.exp()` to compute e^x.
- Up to what order of the derivative n can you compute before you start getting numerical errors?
- What is the source of these errors? Why do they set in at large values of n?