homework3

April 23, 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 first develop a program that can convert the hybrid data into a homogenous format. 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:

[2]:
```
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'},
   ('Alice', 'Boston', 3000.0, '2023-05-01'),
   ('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')
]
```

a) Write a function called `convert_trips` to convert the trip data into a homogenouse format with behavior that follows the documentation string below:

```
def convert_trips(trips):
    """
    Convert a list of trips containing dictionaries and tuples to a list of dictionaries.

    Parameters
    ----------
    trips : list
        Trip database containing dictionaries and tuples, where each dictionary or tuple
        represents information about a specific trip.

    Returns
    -------
    output_list: list of dicts
        Trip database as a list of dictionaries.

    """
```

Test your function by performing a loop over each trip in the `output_list` and printing the trip data (i.e. the dictionary) to the screen.

b) Write a function called `query_trips` consistent with the behavior in the following documentation string.

```
def query_trips(trips, filter_dict):
    """
    Manage business trip data.

    Parameters
    ----------
    trips : list
        Trip database containing a list of dictionaries with information about each trip.
    filter_dict : dict, optional
        If provided, return the list of trips that match the key-value pairs in
        this dictionary. The allowed keys in the dictionary are 'name',
        'destination', and 'date'. Multiple keys can be provided.

    Returns
    -------
    output_list: list of dicts
        List of trips that match the input parameters.

    Examples
    --------
    >>> filter_dict = {'name': 'Alice', 'destination': 'Boston'}
    >>> alice_trips = manage_trips(trips, filter_dict))
```

```
    Returns the list:

    alice_trips = [{'name': 'Alice', 'destination': 'Boston', 'mileage': 1322.0, 'date': '2023
                   {'name': 'Alice', 'destination': 'Boston', 'mileage': 3000.0, 'date': '2023
    """
```

Note: 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.

Test your code on the following example usage cases:

```
# Test filtering trips by destination, should print trips to New York
filter_dict = {'destination': 'New York'}
filtered_trips = query_trips(converted_trips, filter_dict)

# Test filtering trips by name, should print trips by Noah
filter_dict = {'name': 'Noah'}
filtered_trips = query_trips(converted_trips, filter_dict)

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

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

For each of these test, put the code in a separate code cell, and loop over each trip in `filtered_trips` and print the trip data (i.e. the dictionary) to the screen.

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 $x_1 \equiv 1$. Write Python code to determine the largest Fibonacci number less than 10^6?

1.3 Problem 3: 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.

As discussed at the end of the Week3 lecture notes, recursive functions are Python functions that call themselves.

a) Code up a **recursive** 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.

    """
```

b) 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.

c) Up to what order of the derivative n can you compute before you start getting numerical errors?

d) In your own words, what do you think is the source of these errors? Why do they set in at large values of n?