lab1_sp25

January 21, 2025

1 Lab Assignment 1: How to Get Yourself Unstuck

1.1 DS 6001

1.2 Problem 1

Please read the following blog post describing some common Python errors: https://www.tutorialsteacher.com/python/error-types-in-python. Then for each of the following situations, state the what type of exception is Python likely to produce. (I'm looking for specific Exceptions that appear in the table on this website.) [0.5 points each]

1.2.1 Part a

I accidentally type two closing parentheses)) instead of one at the end of a line of code.

- SyntaxError

1.2.2 Part b

I found some code where someone created a useful function using the `def` statement. I copy that code and paste it into my own script. However, the copy-and-pasting somehow got rid of all my tabs so that every line is aligned on the left.

- IndetationError

1.2.3 Part c

I use Python to calculate the number of atoms within every star in every galaxy in the universe.

- OverflowError

1.2.4 Part d

I divide π by the number of Super Bowls won by the Cleveland Browns.

- ZeroDivisionError

1.2.5 Part e

I am hungry, so I reset my Python kernel and start a new script. I write one line of code: `print(tacos)`.

- NameError

1.2.6 Part f

I download the entire internet so I can run a regression on it.

- MemoryError

1.2.7 Part g

I take the square root of the words "twenty five".

- TypeError

1.2.8 Part h

Something entirely different and unexpected happens, causing Python to stop short, its words caught in its mouth, as if reason itself boarded the last train to Seattle in the chill of an unforgiving winter's night. There is no explanation, nor can there ever be. Only one thing is certain: something went wrong.

- RuntimeError

1.3 Problem 2

Use the built-in docstrings or `help()` text for each method to answer the following questions:

1.3.1 Part a

I am loading data from a CSV file using `pd.read_csv()`. But it is taking a long time. Find a parameter that can speed up this method by using the C engine instead of the Python one. [1 point]

- use parameter `engine = 'c'`

1.3.2 Part b

I create the following dictionary:

```
mytrips = {'Portland': 'So many bicycles',
           'Nashville': 'Lost in Opryland Mall again',
           'Richmond': 'Got tattoo and became a hipster',
           'San Diego': 'Chased off beach by sea lion'}
```

How can I see only the cities I've visited (the keys) but not the travel notes (the values)? [1 point]

- `mytrips.keys()` will return just the cities (the keys).

1.3.3 Part c

I am using `px.scatter()` from plotly express to create an HTML enabled scatterplot. How would I add a box plot to the x-axis to visualize the distribution of the x variable? [1 point]

- use the option `marginal_x = 'box'`

1.4 Problem 3

For the following questions, find some examples of people using Stack Overflow badly, either by being toxic while answering a question, or by asking a poorly worded or researched question. You will have to do some digging as these sorts of behaviors aren't the first ones that pop up in a normal search. For toxic answers, look down the page for responses that are NOT the ones accepted by the question-asker. (Remember, these responses are pretty well hidden to us, but they came through loud and clear for the original asker.) For poor questions, look for posts with few responses or maybe some downvotes.

1.4.1 Part a

Search through the questions on Stack Overflow tagged as Python questions: https://stackoverflow.com/questions/tagged/python. Find a question in which an answerer exhibits passive toxic behavior as defined in this module's notebook. Provide a link, and describe what specific behavior leads you to identify this answer as toxic. [1 point]

- https://stackoverflow.com/questions/79360448/unable-to-connect-to-hadoop-namenode-at-192-168-99-10050070-due-to-connection-t
- The final two sentences of this post are passively toxic to me. Saying 'you don't need it to read a CSV file' is rude because, well, maybe they *do* need it to do that. Also the final line is extremely condescending by saying the asker is 'following blogs from 14 years ago'. Heck, maybe it's a good blog. Lighten up a little, man.

1.4.2 Part b

Search through the questions on Stack Overflow tagged as Python questions: https://stackoverflow.com/questions/tagged/python. Find a question in which a questioner self-sabotages by asking the question in a way that the community does not appreciate. Provide a link, and describe what the questioner did specifically to annoy the community of answerers. [1 point]

- https://stackoverflow.com/questions/79370056/open-window-maked-by-qt-designer-from-tkinter
- At the time I'm answering this question, the stack overflow question I've linked has no responses but one downvote. Although it's easy to see why this user probably won't get a great answer: there's hardly even a question asked, it's basically just a google search for the title and then some code. People will likely not appreciate this as it makes the responders do a lot of extra work trying to figure out the issue.

1.5 Problem 4

These days there are so many Marvel superheros, but only six superheros count as original Avengers: Hulk, Captain America, Iron Man, Black Widow, Hawkeye, and Thor. I wrote a function, `is_avenger()`, that takes a string as an input. The function looks to see if this string is the name of one of the original six Avengers. If so, it prints that the string is an original Avenger, and if not, it prints that the string is not an original Avenger. Here's the code for the function:

```
[28]: def is_avenger(name):
          if name=="Hulk" or "Captain America" or "Iron Man" or "Black Widow" or␣
       ↪"Hawkeye" or "Thor":
              print(name  + "'s an original Avenger!")
          else:
              print(name + " is NOT an original Avenger.")
```

To test whether this function is working, I pass the names of some original Avengers to the function:

```
[29]: is_avenger("Black Widow")
```

```
Black Widow's an original Avenger!
```

```
[30]: is_avenger("Iron Man")
```

```
Iron Man's an original Avenger!
```

```
[31]: is_avenger("Hulk")
```

```
Hulk's an original Avenger!
```

Looks good! But next, I pass some other strings to the function:

```
[32]: is_avenger("Spiderman")
```

```
Spiderman's an original Avenger!
```

```
[33]: is_avenger("Beyonce")
```

```
Beyonce's an original Avenger!
```

Beyonce is a hero, but she was too busy going on tour to be in the Avengers movie. Also, Spiderman definitely was NOT an original Avenger. It turns out that this function will display that any string we write here is an original Avenger, which is incorrect. To fix this function, let's turn to Stack Overflow.

Part a The first step to solving a problem using Stack Overflow is to do a comprehensive search of available resources to try to solve the problem. There is a post on Stack Overflow that very specifically solves our problem. Do a Google search and find this post. In your lab report, write the link to this Stack Overflow page, and the search terms you entered into Google to find this page.

- https://stackoverflow.com/questions/20002503/why-does-a-x-or-y-or-z-always-evaluate-to-true-how-can-i-compare-a-to-al
- I searched 'stack overflow if or statement always true python', which led me here. At the top of this question was a note saying this question already has answers at the link I supplied above.

Part b Suppose that no Stack Overflow posts yet existed to help us solve this problem. It would be time to consider writing a post ourselves. In your notebook, write a good title for this post. Do NOT copy the title to the posts you found for part a. (Hint: for details on how to write a good title see the slides or https://stackoverflow.com/help/how-to-ask) [1 point]

- Why does `if a or b or c == d` always evaluate True?

Part c One characteristic of a Stack Overflow post that is likely to get good responses is a minimal working example. A minimal working example is code with the following properties:

1. It can be executed on anyone's local machine without needing a data file or a hard-to-get package or module
2. It always produces the problematic output
3. It using as few lines of code as possible, and is written in the simplest way to write that code

Write a minimal working example for this problem. (Hint: the problem at hand has nothing to do with the Avengers, really.) [2 points]

[7]:
```
a = 'hello'

if a == 'hi' or 'hola' or 'hey':
    print('true')
```

```
true
```

1.6 Problem 5

This problem focuses on how to use an LLM as a code assistant for coding tasks. In this course you are free to use any LLM you want, or none at all, so long as you remember that LLM code assistants are wonderfuly useful if you know how to code and quite destructive to your ability to learn the basics if you do not, so pay attention to your use of AI and catch yourself if you see yourself using these tools as crutches.

However, for this problem only, please download

- VS Code
- The Python plug-in for VS Code
- The Jupyter plug-in for VS Code
- And the Codeium plug-in for VS Code (after creating a free account on codeium.com)

Please review section 1.9 of Surfing the Data Pipeline with Python to learn about how to use the autocomplete, in-line prompting, and external chat functionality that Codeium provides.

Once you have these tools installed, we will use them to address the following problem:

The Collatz Conjecture states that if you take a positive integer and apply the following algorithm:

- If even, divide by two
- If odd, multiply by 3 and add 1

And then repeat these steps for the result, then eventually every number (no matter how large) will converge to a sequence of 4, 2, 1, 4, 2, 1, … This statement is a conjecture because it is not proven for all positive integers.

Source: https://xkcd.com/710/

1.6.1 Part a

Use the Codeium external chat window to ask for a Python function that applies the Collatz algorithm described above one time to a given positive integer. Write your prompt in your notebook and paste the code it returns in a code cell. [2 points]

- Could you write me a python function that simulates the Collatz Conjecture?

[8]:
```
def collatz(n):
    """
    Simulates the Collatz Conjecture for a given number n.

    Args:
        n (int): The number to start the simulation with.

    Returns:
        list: A list of numbers representing the Collatz sequence.
    """
    sequence = [n]
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        sequence.append(n)
    return sequence
```

1.6.2 Part b

Copy the function you received in part a in a new code cell. Place your cursor somewhere inside the function prior to the `return` statements, and use in-line prompting to ask Codeium to add a `ValueError` if the inputted number is negative and a `TypeError` if the inputted number is not an integer. [2 points]

[9]:
```
def collatz(n):
    """
    Simulates the Collatz Conjecture for a given number n.

    Args:
        n (int): The number to start the simulation with.

    Returns:
        list: A list of numbers representing the Collatz sequence.
    """
    if n < 0:
        raise ValueError("n must be a positive integer")

    if not isinstance(n, int):
        raise TypeError("n must be an integer")
    sequence = [n]
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        sequence.append(n)
    return sequence
```

1.6.3 Part c

To test whether applying the Collatz function repeatedly will eventually result in an output of 1, write a while loop that begins with a user-specified positive integer and applies the function from part b over and over until a value of 1 is achieved. Instead of prompting the LLM to create this loop, try coding it yourself but use the autocompletion feature as much as you can. [2 points]

[29]:
```
def testcollatz(i):
    seqs = []
    iters = i
    while i > 1:
        seqs.append(collatz(i))
        i = i - 1

    if sum([seq[-1] for seq in seqs]) == iters-1:
        print('All sequences end in 1.')

    return seqs
```

[33]:
```
testcollatz(10)
```

```
All sequences end in 1.
```

[33]:
```
[[10, 5, 16, 8, 4, 2, 1],
 [9, 28, 14, 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1],
 [8, 4, 2, 1],
 [7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1],
 [6, 3, 10, 5, 16, 8, 4, 2, 1],
 [5, 16, 8, 4, 2, 1],
 [4, 2, 1],
 [3, 10, 5, 16, 8, 4, 2, 1],
 [2, 1]]
```