Exercise: File Finder

Table of Contents

Background

Often times we find ourselves needing to parse through files or data that are not in a format that is easily readable by humans. In these cases, we can use Python to help us parse through the data and extract the information we need. In this exercise, we will be using Python to parse through a file containing data about items placed within a home and create an interface for a user to interact with the data.

Instructions

For this and all future exercises the driver does the typing, while the navigator watches for errors and helps to plan ahead for the next steps. Drivers and navigators can switch roles as often as they would like throughout the pair programming session according to the pair programming conventions. Both driver and navigator should be focused and actively engaged on the programming task throughout the session.

Use any VS Code to write a Python script named `finder.py`. To make this script testable on GradeScope, please use the template provided within the assignment instructions. Specifically you will be completing the code within the body of the `main`. Specific functionality and instructions are detailed below.

`main()`

- Arguments
  - name (str) : The name of the object that a user is wanting to find.
- Functionality
  - Convert the `HOME_DATABASE` string into an array strings by splitting by new lines.
  - Iterate through the array of strings and split the string on commas.
    - Save the name of the item, location of the item, date last moved, and time last moved into variables.

- If the name of the item matches the name passed into the function, print the location, date last moved, and time last moved in the following format:
  - The `item` were found in the `location` and were placed there on `date last moved` at `time last moved`

NOTE The example file in this assingment follows the common structure for a CSV, assume that the columns are the following: `name,location,date last moved,time last moved`

Template

```
import sys
import argparse

HOME_DATABASE = """shoes,floor,8/23/23,10:00am
papers,cubbard,7/23/23,1:00am
picture frames,wall,6/23/23,10:00pm
tshirts,cubbard,1/13/23,9:00am
soccer balls,basket,2/9/22,12:00pm
kitchen tables,floor,3/23/23,4:00pm
cabinets,corner,5/30/23,10:30pm"""

def main():
    pass

def parse_args(args_list):
    """Takes a list of strings from the command prompt and passes them through as
arguments

    Args:
        args_list (list) : the list of strings from the command prompt

    Returns:
        args (ArgumentParser)
    """
    #For the sake of readability it is important to insert comments all throughout.
    #Complicated operations get a few lines of comments before the operations
commence.
    #Non-obvious ones get comments at the end of the line.
    #For example:
    #This function uses the argparse module in order to parse command line arguments.
    parser = argparse.ArgumentParser() #Create an ArgumentParser object.

    #Then we will add arguments to this parser object.
    #In this case, we have a required positional argument.
    #Followed by an optional keyword argument which contains a default value.
    parser.add_argument('object', type=str, help="Please enter the name that we are
searching for.")
```

```
    args = parser.parse_args(args_list) #We need to parse the list of command line
arguments using this object.

    return args

if __name__ == "__main__":

    #If name == main statements are statements that basically ask:
    #Is the current script being run natively or as a module?

    #It the script is being run as a module, the block of code under this will not
beexecuted.
    #If the script is being run natively, the block of code below this will be
executed.

    arguments = parse_args(sys.argv[1:]) #Pass in the list of command line arguments
to the parse_args function.

    #The returned object is an object with those command line arguments as attributes
of an object.
    #We will pass both of these arguments into the main function.
    #Note that you do not need a main function, but you might find it helpfull.
    #You do want to make sure to have minimal code under the 'if __name__ ==
"__main__":' statement.

    print(main(arguments.object))
```

Running your program

Your program is designed to run from the terminal. To run it, open a terminal and ensure you are in the directory where your script is saved.

The program takes two required command-line arguments: Object Name.

Below are some examples of how to use the program. The examples assume you are using macOS and your program is called `finder.py`. If you are using Windows, replace `python3` with `python`.

Basic usage

```
python3 finder.py papers
```

Output:

```
The papers were found in the cubbard and were placed there on 7/23/23 at 1:00am
```

NOTE The program will not proceed until you provide inputs.

Basic Usage with Bad Input

Inputs:

```
python3 finder.py bad
```

Output:

```
ValueError: Sorry, could not find your item named bad within the database
```

Submitting your script

Whether or not the instructional team member has had a chance to look over your script, at the end of the session, the driver should email a copy of the script to the navigator. If you did not complete the script during pair programming time, you may complete the script together at a later time, or each of you may complete the script individually. **It is not okay for one team member to complete the script for the other person without that person's participation.**

Both driver and navigator should upload the script to GradeScope, even if the instructional team member has signed off on the script.