EECS268:Lab1

Navigation

Home
Information

Syllabus
Schedule
Lecture Archive

Classwork

Labs
Submitting Work

Contents

Due time

This lab is due one week from start of your lab.

Required Files

- driver.py
  - Contain a main function
- executive.py
  - Contain an Executive class
  - Handle file I/O and user interaction
  - Creates a list of board games by extracting the information from file
- boardgame.py
  - Contain a BoardGame class
- Provide input file (https://people.eecs.ku.edu/~jwgibbo/eecs268/2022spring/labs/lab01/EECS268-2022-spring-board-game-data.tsv)

Overview

Pull up a chair and pop that Mountain Dew, it's time to play some board games! The provided input file (https://people.eecs.ku.edu/~jwgibbo/eecs268/2022spring/labs/lab01/EECS268-2022-spring-board-game-data.tsv) is Dr. Gibbons' actual board game collection. Inside you'll find grounds for divorce, I mean you'll find a listing of the games I own, my rating, the public's rating, and other useful information. Your job is to pull all the information out the file and let the user query the data in various ways.

This lab will only depend on your 168 knowledge:

- Class defining and object creation
  - Initializing an object member variables, and member methods
- List of objects
- File I/O
- Basic terminal I/O

Please use the C++ to Python document (https://docs.google.com/document/d/1BYeChQ5Tt9qxm5uY5CzJ4H5nTY6kafobsGCU7Y984GA/edit?usp=sharing) to either as a primer or a refresher.

Mountain Dew is not affiliated with KU, Dr. Gibbons, or the State of Kansas. It's also kind of gross.

File format

The file contains several rows, each row contains the information for a single game.

Each game has the following format <name> <gibbons-rating> <people's rating> <year published> <min players> <min playtime>

All data here has been provided by Board Game Geek (https://boardgamegeek.com/)

NOTES:

- I modified the names to help you read them in easier (spaces replaced with underscores and all names are lowercase)
- All ratings are between 0 and 10
- Some aren't standalone games, but are expansions, you don't have to worry about distinguishing them

Sample layout from the file

```
10_minute_heist:_the_wizard's_tower     2017    6       5.87304 5       10
1st_&_goal      2011    7.5     6.02351 4       150
7_wonders_duel  2015    8       7.97914 2       30
adventure_time:_love_letter     2015    6.5     6.3181  4       20
arkham_horror:_the_card_game    2016    9.5     7.92726 2       120
arkham_horror:_the_card_game_â€“_return_to_the_night_of_the_zealot      2018    3       6.09202 2       120
arkham_horror:_the_card_game_â€“_the_dunwich_legacy:_expansion  2017    7       7.32741 2       120
arkham_horror:_the_card_game_â€“_the_forgotten_age:_expansion   2018    8       6.22352 4       120
arkham_horror:_the_card_game_â€“_the_path_to_carcosa:_expansion 2017    9       6.83716 2       120
bang!_the_dice_game     2013    7.5     6.81194 8       15
```

Reading the file should a fairly easy task for you. Recall using string methods like split.

Please note, when we test your program we will use different input files, but the format will be the same as the one provided.

I encourage you to come up with your own test files.

You may assume the files will be properly formatted.

User Interactions

The user will launch your program and pass enter the file name containing the player data from the terminal.

Once the file is read in, provide the user with a menu in order to do the following:

```
Name (year) [GR=gibbons-rating,PR=public-rating,MP=min-players,MT=min-playtime]
```

For example:

```
10_minute_heist:_the_wizard's_tower (2017) [GR=6, PR=5.87996, MP=2, MT=10]
1st_&_goal (2011) [GR=7.5, PR=6.05092, MP=2, MT=120]
```

Unless otherwise stated, when printing a player to the screen, use the following format:

User's Menu

1. Print all games (same order as from file)
2. Print all games from a year
   - Obtain a year from the user and either print all the games from that year or print "No games found"
3. Print a ranking range
   - Obtain a ranking range (e.g. 1 - 10) from the user and print all in that ranking range (Gibbons rating)
4. The People VS Dr. Gibbons
   - The goal of this is see where Dr. Gibbons and the people disagree
   - Obtain a number (0-10, decimals allowed) from the user and print all games where the people's rating and Dr. Gibbons rating are separated by that much or more
   - Example, if the user wanted to see all games where the people's rating and Dr. Gibbons' rating differed by more than 1.5, they would enter 1.5 at the prompt then you see games like Pandemic listed
   - NOTE: This option doesn't care which rating is higher, it just prints games where ratings differ by a threshold set by the user
5. Print based on play time
   - Obtain a play time (in minutes) from the user
   - Print all games that have a min play time of that value or lower
6. Exit the program

Classes

In 268, your main function should be very concise, passing control to some kind of Executive class that will then run the show (example main below).

You need to think about how you represent a single entry from the file. Remember, we are making object oriented code, so you need to solve all the problems you encounter by using/create a class!

Things to think about:

- How can I represent a single game?
- What kind of data does a game have, and how will I access it?
  - Remember, we're not in the business of public member variables that house critical information
- What kind of information does a game need at construction time?
- What will contain all of the games?

You'll need a class (or classes) to handle other important things:

1.
   - You can assume if the file was able to be opened that it is formatted correctly
2. Reading the data from file and storing it
3. Present the user with a menu and handle the interactions
   - Don't assume the user is going to give you good input at the menu! Make sure they choose valid menu choices
   - You are not responsible for throwing or catching exceptions.

Goals for Day of Lab Release

Before lab is over, I would hope you have a good handle on the following:

- How many classes your going to create and what responsibilities each will have
- A BoardGame class created
- The beginnings of an Executive class
- Basic testing of reading information from the file, even if it's not being stored just yet
  - Being able to get the content out of file and print it to the screen is a good first step
  - When you're ready try loading it into an array of objects and then verifying it was store correctly
- A main that takes a file name from the command line and passes it to the Executive

Dos and Don'ts

- Do run your code often!
  - I'm talking about running after you write any significant portion of code
  - Define a method? Run it.
  - Made a new class with a skeleton definition? Run it.
- Do test your code
- Do use the debugger

- Don't try to write the entire lab and run it for the first time
- Don't forget to give your program different input files that you've created/edited yourself

Debugger

I want to encourage you to use the debugger over shoving prints in random places in code to figure out what's going on. If you have IDLE downloaded, you can turn on the debugger from the menu. Some easy steps to utilizing the debugger is

1. Set a breakpoint
2. Run debugger so that your code freezes when hitting the break point
3. Look at the values of the variables at that break point to verify their accuracy
4. Use the step or next commands to advance in the code
   - Be aware step will potentially enter code that you didn't author

driver.py

Main will be in charge of very little. It will import the needed files, but as soon as possible it will hand control over to an Executive class.

```
from executive import Executive

def main():
  file_name = input("Enter the name of the input file: ")
  my_exec = Executive(file_name)
  my_exec.run()

if __name__ == "__main__":
  main()
```

Submission instructions

Send a tar file with all the necessary files (.py files and input files) in a zipfile or tarball file to your GTA. Consult your GTA for preferred submission platform (i.e. canvas, email, blackboard)

Creating a File Archive Using Tar

The standard Unix utility for created archived files is **`tar`**. Tar files, often called **tarballs**, are like zip files.

1. Create a folder to hold the files you will be sending in. The folder should be named like *LastName-KUID-Assignment-Number*:

   ```
   mkdir Smith-123456-Lab-01
   ```

2. Now, copy the files you want to submit into the folder:
3. Tar everything in that directory into a single file:

   ```
   tar -cvzf Smith-123456-Lab-01.tar.gz Smith-123456-Lab-01
   ```

   That single command line is doing a number of things:
   - `tar` is the program you're using.
   - `-cvzf` are the options you're giving to `tar` to tell it what to do.
     - `c`: **create** a new tar file
     - `v`: operate in **verbose** mode (show the name of all the files)
     - `z`: **zip** up the files to make them smaller
     - `f`: create a **file**
   - `Smith-123456-Lab-01.tar.gz`: the name of the file to create. It is customary to add the `.tar.gz` extension to tarballs created with the `z` option.
   - `Smith-123456-Lab-01`: the directory to add to the tarball

Please note that it is **your** responsibility to make sure that your tarball has the correct files. You can view the contents of a tarball by using:

```
tar -tvzf filename.tar.gz
```

Rubric

Grades will be assigned according to the following criteria:

- [15pts] Class Design
  - Each class' role should be well defined
  - Avoid create "god classes" or classes that are in charge of way too much, break the problem down into several classes
- [10pts] Code documentation
  - Files need comments (e.g. Author, date, modifed time)
  - Methods need descriptions as well
  - Readability
- [5pts] Readability of output and stability of interaction
- [5pts] Input validation
  - Check for bad ranges on good types of input
- [55pts] User interactions
  - [10pts] interaction 1
  - [10pts] interaction 2
  - [10pts] interaction 3
  - [10pts] interaction 4
  - [13pts] interaction 5
  - [2pts] exiting
- [10pts] No random crashes (assume correct input types and valid files)

Retrieved from "https://wiki.ittc.ku.edu/ittc_wiki/index.php?title=EECS268:Lab1&oldid=23705"

- This page was last edited on 25 January 2022, at 21:17.