Erik Knighton

5/30/2020

IT FDN 100 A

Assignment07

<https://erikjkuw.github.io/IntroToProg-Python-Mod07/>

Erringly Diced Pickles

# Introduction

I honestly don't know what I was thinking. I assumed, because we weren't starting with any preexisting code, that we were tasked with building a program equally as complex, even moreso than last week. I watched the course video, I read the documents and the textbook, but somehow I got stuck in my own head and came up with something I'm proud of, but perhaps ashamed it took so long to construct.

In any case, I had talked previously about writing a script that would roll for ability scores in Dungeons and Dragons using a method taught to me: 8d6, rerolling dice under 4, and only two rerolls. You do this twice and then add and assign to your various abilities. The statistical value is in keeping with what is the standard array [15, 14, 13, 12, 10, 8], but allows higher starting values, which allows for optimizing characters without having to sacrifice too much. I already wanted to have such automation on hand to share with my adventuring group, so it seemed I could two-birds-one-stone this assignment.

# Getting Carried Away

To begin, I wrote some pseudo code to get an idea of how complicated this project might be. On the next page I have preserved it in its infancy, a testament to my naivety and hubris. Knowing we would have to pickle our data, I chose to model it on the two previous ToDo list assignments, where there would be a menu of choices, dictionary rows nested in lists, cryptic tuples to unpack and a whole mess of functions to declare as well as document (something I did not effectively flesh out last time).

Once I had completed the pseudo code, I created a template in PyCharm, and then did a side-by-side comparison with the organization of module 6. I decided not to copy and paste anything, favoring the clunkiness of tapping away in an effort to ingrain the knowledge preserved in the starter script that I had modified.

#### Pseudo Code:

#### Load existing data from .bin

#### Unpickle .bin file

#### Display Menu:

#### >(1) Roll a Character

#### >(2) View Characters

#### >(3) Save Characters to File

#### >(4) Exit

#### (1) Roll a Character

#### Ask user for Name of Character

#### Save input value as list name,

#### While loop count less than 3 (total rolls is 3)

#### or list length less than 6 (must reroll 3 or more dice):

#### Roll 8 dice

#### For loop:

#### > each die that rolls below a 4,

#### > delete and save 4+ to a list

#### > increase count by 1

#### > check list of rolls length

#### Print list of rolls

#### Reroll all lower than 4,

#### append 4+ to list of rolls,

#### increase count by 1,

#### check length

#### sort list highest to lowest

#### Print final list of 8 numbers, calculate total

#### Check total against Standard Array

#### (Exception - Total less than Standard Array,

#### ask DM for reroll)

#### Save list of rolls as value to dictionary key Ability Rolls

#### (2) View Characters

#### For loop:

#### Print each list

#### (Exception - No Existing Characters

#### default to (1) Build Character)

#### (3) Save Characters to File

#### Ask user y/n

#### If y, set saved binary variable to True

#### Pickle data

#### Shelve to store data in .dat file

#### (Exception - No Characters to save)

#### 

#### (4) Exit

#### (Exception - saved variable False, print warning)

I was able to classify my functions as before, though figuring out which ones would go under Processor or IO took some doing. I often found myself wanting to add print() statements everywhere as I used to do while testing each step, verifying their successful operation.

Eventually I got everything working. And I do emphasize eventually. I had to look a few things up, referencing the textbook for the random module, pickle, errors and exceptions, as well as ensuring I had a solid grasp of the concepts.

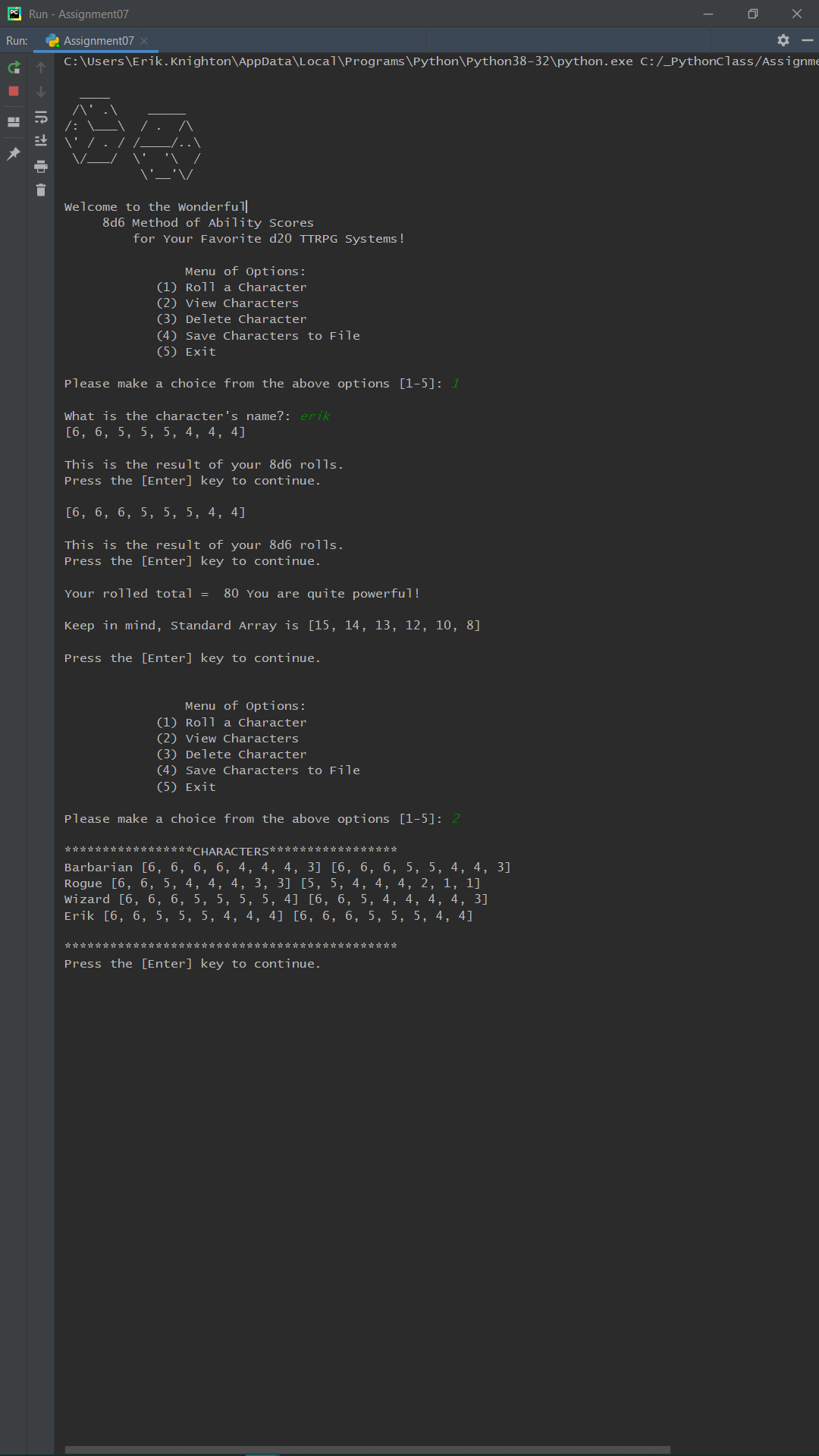
The output of my primary functions, that is to roll a bunch of dice and put them neatly in a dictionary under a character name, looks like so:

#### @staticmethod def dice\_roll(): """Rolls 8d6 dice, rerolling twice any numbers under 4 **:param** roll\_count: **:param** list\_of\_rolls: (list) of **:param** roll: **:return**: """ roll\_count = 0 list\_of\_rolls = [random.randint(1,6) for roll in range(8)] while roll\_count < 2: for roll in list\_of\_rolls: if roll < 4: list\_of\_rolls.remove(roll) roll = random.randint(1,6) list\_of\_rolls.append(roll) roll\_count += 1 return sorted(list\_of\_rolls, reverse=True), "\nThis is the result of your 8d6 rolls."

This function is run twice through, as is done when rolling physical dice. These two created lists are then stored alongside the character name as values with identifying keys:

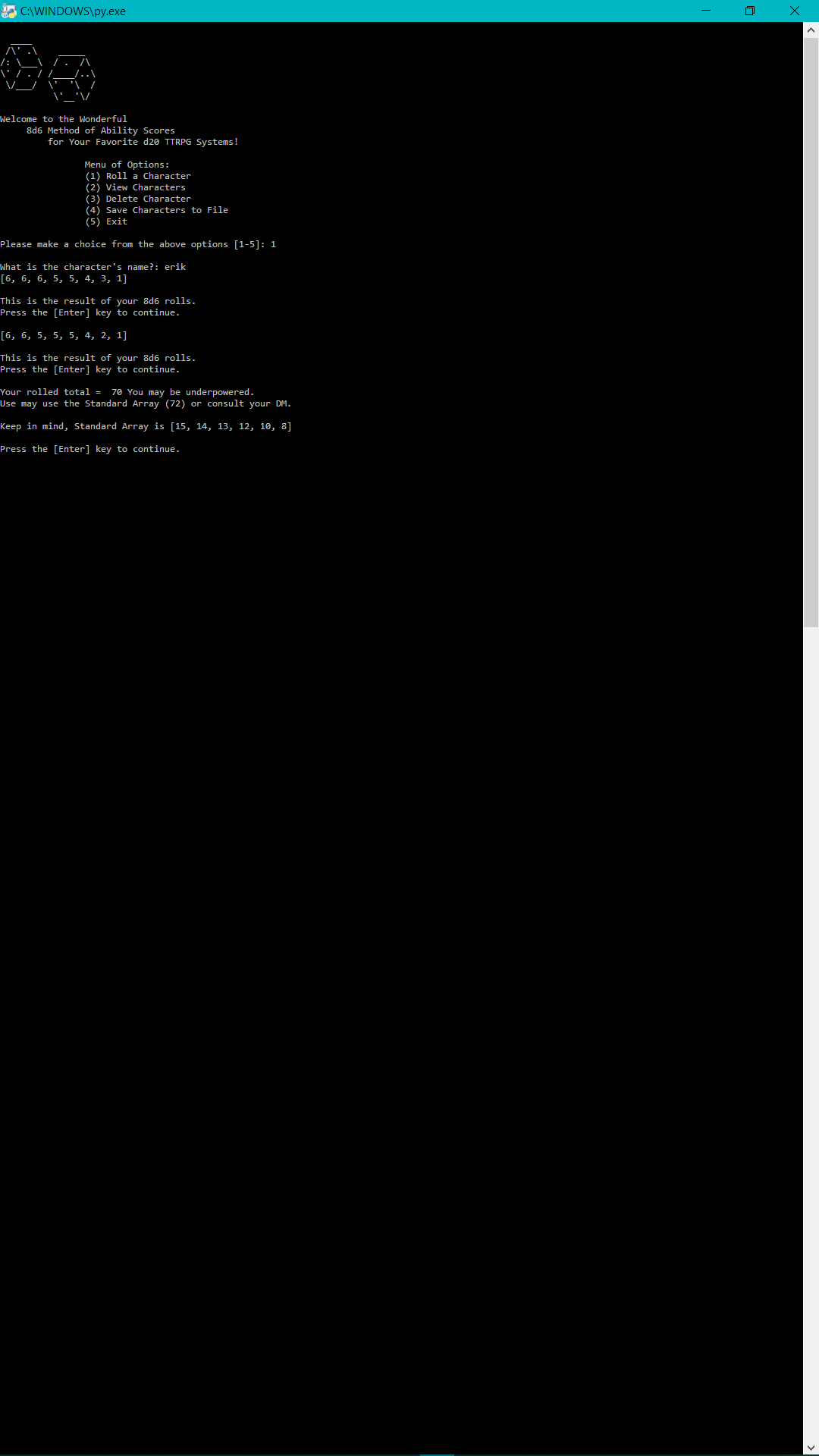
#### {Name : "character name", Dice Rolls 1 : [lstDiceRolls1], Dice Rolls 2 : [lstDiceRolls2]}

Here is a screen capture of the above code working in PyCharm (Figure 1). The code automatically unpickles the stored data when it opens, and so the storage file is populated already with a few generic characters, named after typical classes from Dungeons and Dragons type role playing games. Included is some ASCII art I pulled from <https://www.asciiart.eu/miscellaneous/dice> (External Site).



### Figure 1. Rolling a New Character, and Oh Look, It's Me!

Here, too, is the same code operating in the terminal (Figure 2). Though I do not show the error here, nor the pickling of the input data, there are figures further down the document showing both the raw code and its execution.



### Figure 2. Rolling Stats for Myself Again, with Slight Downgrade

\*It was here, after I had spent far too long poring over my script, getting everything in order, making sure all of the steps performed their requested functions successfully, and trying to make them both user-friendly as well as nicely organized on screen, that I watched the starter video. The video that says to keep it simple and write one or two code snippets that describe how to both pickle data and handle errors and exceptions. Because I had set my mind on using this script as exemplar of both of those required concepts, there was little use in backtracking now. I will publish this code on my GitHub page, and take whatever hit I should from erring towards unnecessary complexity. And where would I be if my long-windedness were unjustly fettered?

# In a Proverbial Pickle

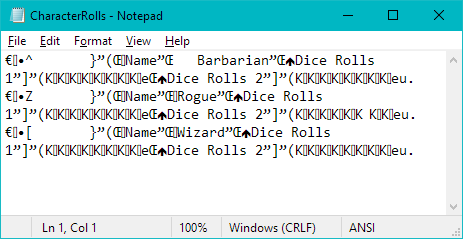
My script, as I said above, was modeled on the Task/Priority script from assignments 6 and 7. Therefore, pickling instead of writing to a text document was a simple leap in difficulty.

To pickle my rolled stats, as well as the assigned character name, I simply used the pickle.dump() function. The first time I ran this part of the code I created the file using the "wb+" access mode. This is important because it informed me that my read function should contain an exception for when the file doesn't exist or is empty, and therefore can still be created without breaking the program. I will detail that in the following section on error handling. Once serialized in the binary document, the .dat file displays the following (Figure 3) for the below character data:

Barbarian [6, 6, 6, 6, 4, 4, 4, 3] [6, 6, 6, 5, 5, 4, 4, 3]

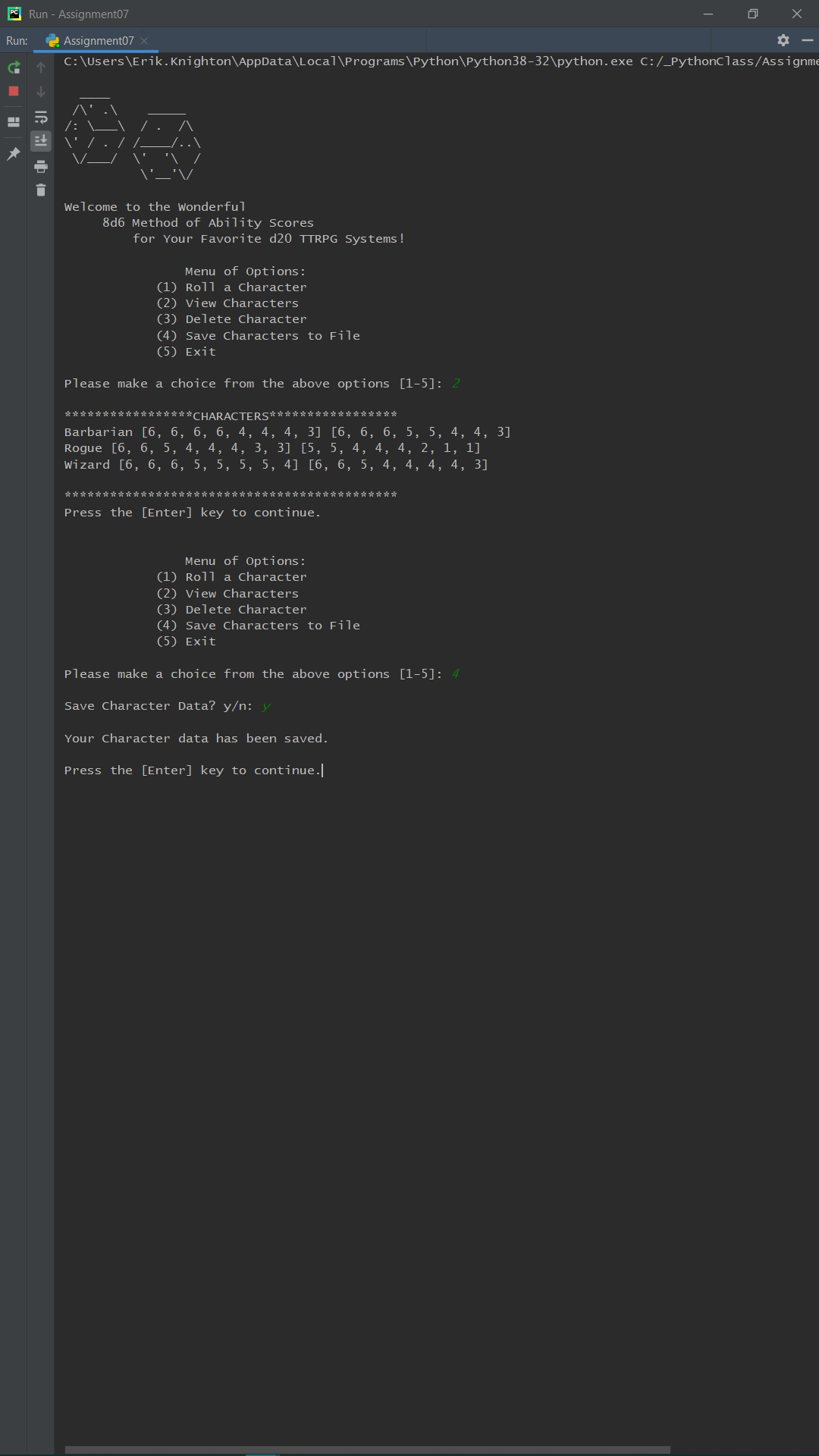
Rogue [6, 6, 5, 4, 4, 4, 3, 3] [5, 5, 4, 4, 4, 2, 1, 1]

Wizard [6, 6, 6, 5, 5, 5, 5, 4] [6, 6, 5, 4, 4, 4, 4, 3]



### Figure 3. Pickled Player Characters and Their Stats

In PyCharm there isn't much to see, but I ensured there was a message printed which would confirm successful preservation of the characters and their dice rolls (Figure 4). The data from the document in the above figure is loaded and then displayed with option 2, after which it is pickled and stored again through option 4 on the menu. The document is unchanged, though both rolling new characters and deleting characters work similarly to how they did in the ToDo List assignment.



### Figure 4. Showing Unpickled and Subsequently Repickled Data in PyCharm

# How to Handle the Exceptional

Unpickling my data proved to be taxing. I kept getting an error that there were integers and strings and gobbledygook aplenty that it couldn't unpack, or at least that is how I interpreted TypeError: unorderable types: int() < str(). I had gone into this program with the understanding that pickling would be easy, and therefore was truly puzzled.

I did some cyber-sleuthing, and could not figure out why my data was being denied its due unpickling. I went back over my code, str()ing my strings and int()ing my integers. Somehow it worked. Though it might have made a better example for handling an exception had I actually been able to puzzle it out. Either way, I was presented with another error, the one I mentioned earlier in reading my data.

However, I also determined that I needed to make a EOFError exception in my code to allow for an empty file, as well as a file that ends when the pickle module is expecting to deserialize more data. Here is what that code looks like, using a with/as clause (which I encountered on a previous assignment, and kept tucked away for a rainy day like today) inside a try/except clause:

#### list\_of\_rows = [] with (open(file\_name, "rb")) as file: while True: #Handling Exception encountered from empty file try: list\_of\_rows.append(pickle.load(file)) except EOFError: break return list\_of\_rows

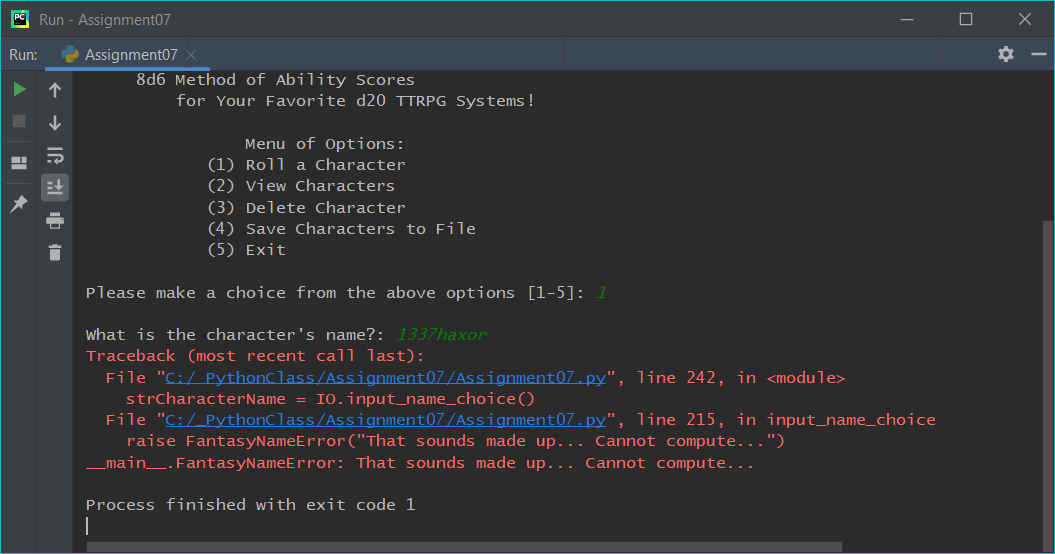
The other side of the exception-handling coin I wished to address was the custom error. There were multiple opportunities in my pseudo code where I thought I might be able to squeeze one in, but it turns out that most of the time all I needed was a if/else conditional. There was one exception, though, and that was to throw a monkey wrench into the system should some player opt for a character name with symbols or worse, numbers! And thus I scripted the custom error FantasyNameError:

#### class FantasyNameError(Exception):

#### pass

In order to take full advantage of this error, I had to search out how to verify if a string was, indeed, made up of only the alphabetical characters. The string method isalpha() gave me what I needed, which led me to this piece of code, which raises the error with a message when an offensive name warranted the code to cease altogether (Figure 5):

#### character\_name = str(input("What is the character's name?: ")).title() if character\_name.isalpha(): pass else: raise FantasyNameError("That sounds made up... Cannot compute...") return character\_name



### Figure 5: Custom Error Message for Silly Naming Conventions

The terminal, of course, doesn't display this error, instead terminating the program prematurely in accordance with prescriptive naming convention.

I believe that is the last of the error handling concepts I tackled in this module. Using finally was the only clause I didn't make use of. I am keeping it in mind for future assignments whereby a document might require closure only after passing all the exceptions or something. Next is taking on Jekyll with the GitHub website portion of the assignment. Plenty of opportunities to go astray there, as well.

# Websiteography

Here are some pages that I found to be especially useful, with a brief synopsis of each. There were times I ventured onto Stack Overflow or previewed a bit of a YouTube video, but largely I stuck with these so I wouldn't be persuaded to copy someone else's code.

## Pickling Links:

[Learn Python - Serialization](https://www.learnpython.org/en/Serialization)

I really enjoy the DataCamp widgets that let you toy with the code they offer in the examples on this page (something I wish I could put on my GitHub). Not only do I get a feeling for how the author approaches both using JSON and pickle, I have the opportunity to edit and implement my own variations.

[Tutorials Point - Data Persistence](https://www.tutorialspoint.com/python_data_persistence/python_data_persistence_pickle_module.htm)

As we approach object orientation, I thought it best to bookmark this site, as it gives a brief example of both the Pickler and Unpickler classes. I do not understand how these are supposed to be different, yet, but as I said, worth holding on to in the interim.

[RealPython - Pickle Module](https://realpython.com/python-pickle-module/)

Noteworthy in large part because there are some advanced techniques that I might not have known were related to pickling or serialization: like JSON. Also of note is that there are security issues with the pickle module. Unpickling unknown data is like eating pickles you found in the garden shed of a house you inherited from your long lost relative.

[Programiz - DocStrings](https://www.programiz.com/python-programming/docstrings)

I needed to brush up on docstrings anyway, and this was the only page to mention the pickle module. It's a simple list of all the available functions that the pickle module can perform, and the syntax for each. I was prompted to look up the difference between some, primarily the string versions dumps() and loads() versus their anything-goes counterparts.

## Exception Handling Links:

[Learn Python - Exception Handling](https://www.learnpython.org/en/Exception_Handling)

I realized when looking at this page just how valuable the PyCharm built-in warnings and error are. I could have had NameError, IndexError, ValueError all over the place, but PyCharm by and large just highlights them before you even run the code. Good to know what they look like, and how to implement them, though, with the DataCamp widgets.

[Tutorials Point - Python Exceptions](https://www.tutorialspoint.com/python/python_exceptions.htm)

A list of 29 existing errors, and when they are raised! Superb! Assert will be useful in the future, too, I'll bet. Mostly I appreciated the layman's approach to exception handling here. There's pseudo code, real code, and then step-by-step explanations of what is happening as you watch the code execute. Also a good reminder that you can have exept clauses without exceptions, or with many.

[Real Python - Exceptions](https://realpython.com/python-exceptions/)

Some of the concepts here are over my head, but I wanted to keep it in mind because the site itself is worthwhile, as are their many visual aids, videos, downloadables.

[Programiz - Exception Handling](https://www.programiz.com/python-programming/exception-handling)

I thought I could use some of the existing errors from the book, like ValueError, before I made my own to raise in instances of non-alphabetical naming. I looked here in large part because you get to run the code on the page, and see how it operates, and how the errors look.

# Summary

I took on more than I could chew for this assignment. I had a lot of fun, and learned a lot about why the scaffolding supplied in past assignments was provided to us. I am proud of the program I have developed, and I hope that in the future I can give it some GUI shine and polish. For now, I have made a decent stab at pickling data, and following up with how serialization works with that module. In equal measure I hope my brief overview of both exception handling and custom errors suffices for now. Admittedly I am at my wit's end. Hopefully this journey elicits some delight, and a sense of curiosity for those who are not familiar with role playing games as a hobby. I certainly did not mind so much missing the forest for the trees incorporating a favorite pastime.