Contents

[0.27 - How the game works, The current structure, and the current best strategy to win 1](#_Toc4088541)

[How the game works/what I’ve learned about the Game 1](#_Toc4088542)

[The Game Structure/execution 3](#_Toc4088543)

[Game Initialization 3](#_Toc4088544)

[Game Structure 3](#_Toc4088545)

[Best Strategy to Win for speed running or otherwise 3](#_Toc4088546)

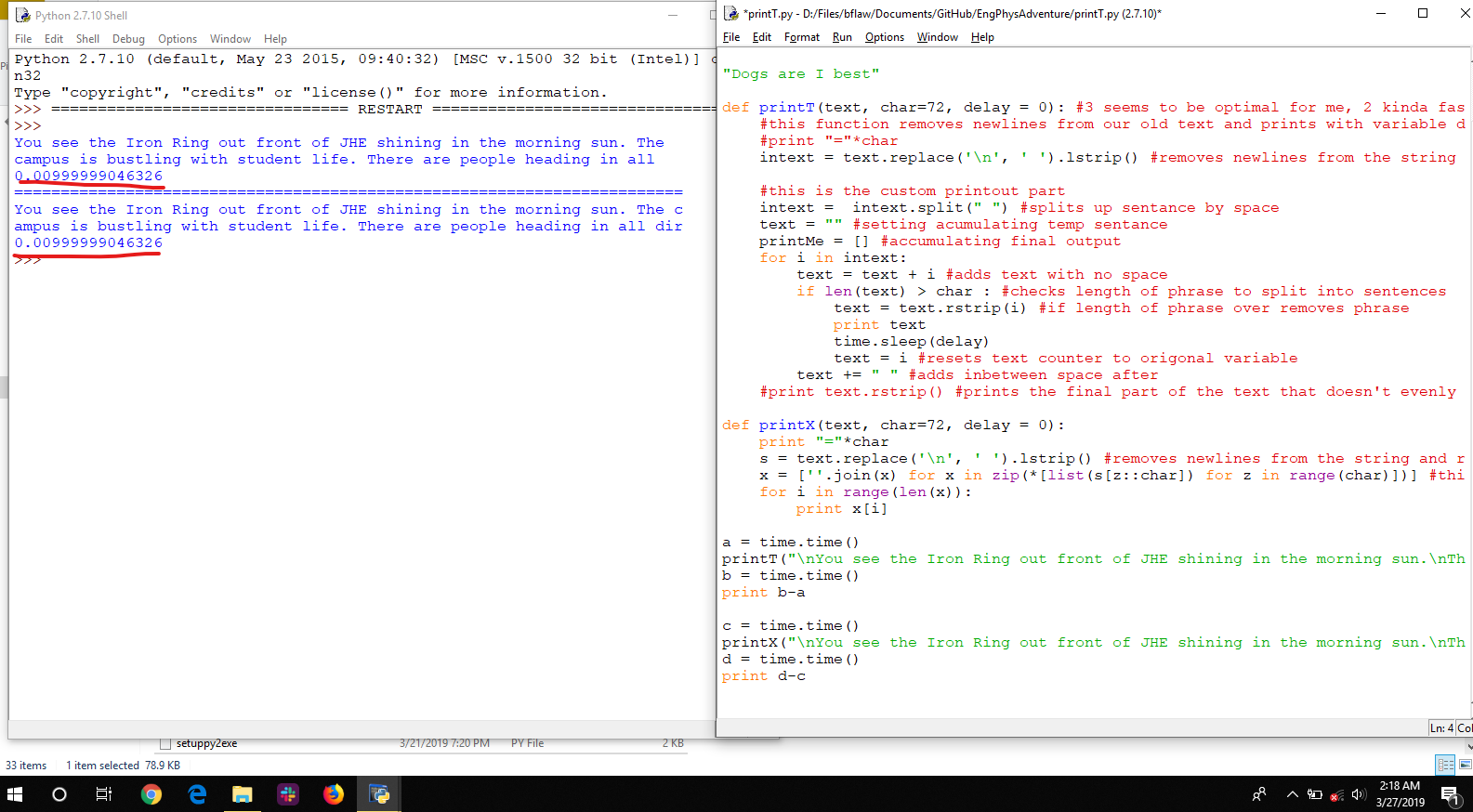
[0.26.1 – SpeedRun Update 3](#_Toc4088547)

# 0.29 – printT (special print),

## printT

PrintT(text, character limit, delay) is a custom line printing module for the game that will allow for using old written lore with manually entering newlines or text blocks with newlines at all (as long as there’s spacing between words). This function automatically scales to any defined character width screen, can have custom delays, breaks up paragraphs automatically (as long as using 5 consistent . ! ? punctuation marks), and can be overwritten using an arbitrary delimiter (\S) to start a new line or (\S) (\S) to enter an empty line with a delay in-between. I maybe have spent more time creating this (5 hours) than total time saved in waiting but I learned a lot about python text parsing, function pass by reference/ global variables/proper non-global variable use, doc strings, and performance testing.

Although I should have done some research before creating it I simply assumed this would be too specific an algorithm to find. I did use an algorithm found online to try to parse the characters quickly but in the end gave no performance difference on the single test I performed (see below). I will talk more to the performance after.



### printT Code

The code uses a simple non-intelligent algorithm remove all new lines and replace with spaces and then to split all words in the input text into a list of “words” with no spaces. “Word” is in quotes here because it may contain punctuation or any other manner of things, its just separated by spaces so it’s key to make sure that nothing is jumbled together or else it might overflow (or at least the longest word would be the character limit. Then it basically loops through all the words and builds up phrases one word at a time until it either matches or exceeds the character limit. If it exceeds it that overflowing word is removed and added to the next phrase with all extra spaces removed. The paragraph sensor works by just counting exclamation points, question marks, and peroids in each “word”(with the exclusion of Dr.) until it reaches 5 then stops the phrase and adds an extra line. If more exclusions and specific words need to be added they can be. The (\S) parser took some time but in the end simply looks in each “word” and if it finds the (\S) in it removes it and ads the beginning of the word to the end of the phrase before printing and clearing it. IF there was a word after the (/S) it ads it to the next phrase unless it’s a space.

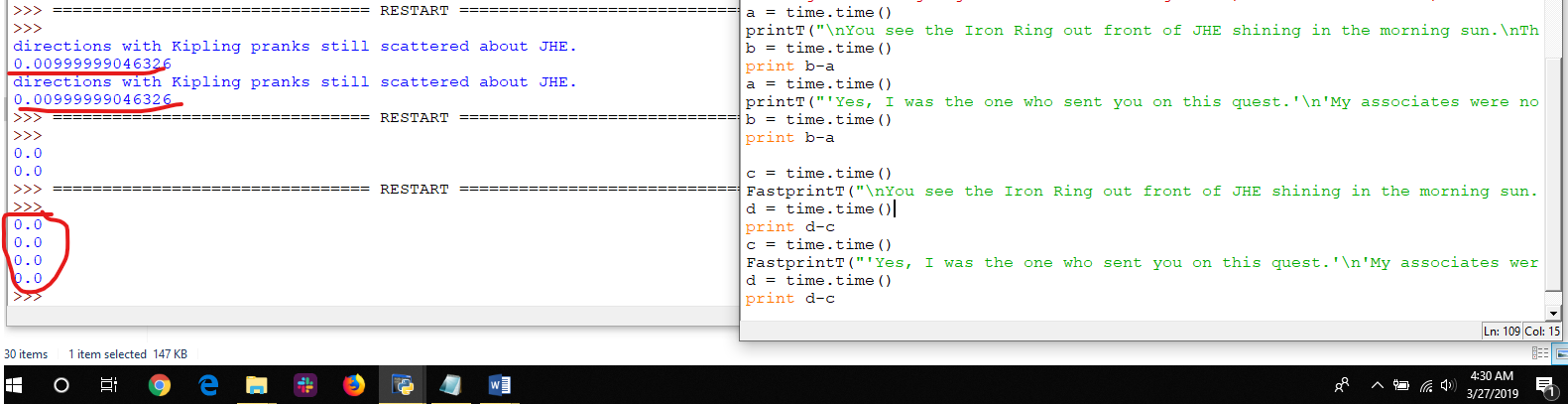
There is some tricky spacing and parsing implementation that took me some time and I coded in jank manor. So for further implementation and examples see the code.

### printT Performance

Using the time method, tested differences in the function in a tic-toc fashion before outputting them. Subject to processor load noise but ran multiple times to confirm input.

I tested the full printT and then a reduced version with less checks to look for a speed difference and saw none, even zero, within cpu noise time.

It was discovered that even on a long paragraph all the checks and looping through words took no identifiable time as seen below. The main delay was from printing output so once the printing was silenced the time for calculation went to zero. This shows that most of the algorithms (on the scale of EBTA) are probably limited by the text output so should not worry too much about scale. But this was fun and reassuring to see. Possibly there will be some calculation time over the course of the whole game text but this is hard to test.



# 0.28 – Music, settings, play testing and feedback

## 0.28.1/2 – Save file problems

After fixing the save file from moving the quest file has shown mean just how violtile/finicky this game is that part of it breaks just by me forgetting how to do dictionary’s correctly. Also how I don’t have full handle on global variables and how we should not be using them at all to make it more clear.

The main counter to making sure it works properly is play testing with different scenarios including: completing the evil mode, good mode, the completions mode, and testing save functions.

Also moving the quest mode to 0.28.1 made the end of the game break with having to call the function.

## 0.28 – Music, Marginal Improvements, and Feedback

Made a bunch of small improvements and mainly music. Music is unfortunately just a simple player that is threaded that can’t stop or play other music. The best option I see for threaded music (without making our own thread which is reasonable) is figuring out how to get pygame to compile in a reasonable way.

Moving the quest file to another function I had to shuffle around lots of thing including the imports but I did it in a similar way to the creative mode imports. This also made problems with the definition of quests global function and saving because I forgot how save files work.

Altogether not too much big structure change but a lot of minor details that added up including making the settings a global function that are defined in the opening file. Trying to keep the program modular but hoping with further help the game can be greatly improved in code structure and best practices for reading/understanding, and making changes without breaking everything. Might help to make node charts to show how everything is connected so when changing things test everything connected.

Still want to personally figure out global variables and how we can use object inheritance to our advantage. Other than that figured out how to do a mid-way custom print function that is basically just a mod overlay for the current print function. Also got a lot of good feedback to be implemented for Kipling!

Current Green console settings in R,G,B is (52,255,52)

# 0.27 - How the game works, The current structure, and the current best strategy to win

## How the game works/what I’ve learned about the Game

So while working on the game to make the save modes and restructuring the game I figured out how the game works in terms of structure (which isn’t very apparent on first glance) so I wanted to document that and my changes.

The main thing to know about the game is the variables and the structure. I’ll start with the variables which I have not changed except for adding one global dictionary called GAMEINFO.

The backend of the game was mostly written by Mitch Lemeiux and utilizes dictionaries and objects. For those unfamiliar as I was a dictionary in python is simply a list that is indexed by a key instead of it’s number.

For example

Animals = [ “Dogs”, “cats”, “frogs”] where animals is a list of items where you acesss the string “Dogs” by using x[0] where it’s the 0th element

Animals = {“playerpet” : “Dogs”, “npc pet”: “cats”, “extra pet”: “frogs”} where animals is now a dictionary with syntax { key : value}. So to access the string “Dogs” by using x [“playerpet”].

Dictionaries are valuable to use over lists because it’s more apparent in the code what information you’re trying to access. For example it’s much more clear to know what item you’re referring to if your dictionary is keyed by the item and you see ITEMS[“Eng Phys Shirt”] vs ITEMS[362].

The game also uses object oriented programming (OOP) with several game class to define the types of objects used. Hopefully the reader of this is familiar with objects and understands the advantages of using them so I will not go over them.

The main variables in the game that store all the information are also global (defined in the top of the GameFunctions.py file except for QUESTS which is at the bottom) and are in allcaps. These variables are global so they can be passed between functions by simply definining them in the function as “global PLAYER” for instance to be able to access the player’s attributes. Otherwise variables defined and used in the function are local to that function unless passed as an attribute to the function and then returned by the fucntion. The variables are acessed by the main file by using “from GameFunctions import \*” which imports all the variables, functions, and imports of the GameFunctions file.

Bellow is the definition of the main global variables in the game

1. PLAYER #The main character. player is an object instance of class character.
2. ITEMS #All the items. This a dictionary of objects of class equipment keyed by their lowcase equipment name (item.name). Remember the lowercase, may trip you up if referencing upercase version in the file.
3. MAPS #All the locations. A tuple of objects of class Map inxed by there x,y,z coordinate (MAPS[x][y][z])
4. INTERACT #All the interactables (stationary things that need something). This a dictionary of objects of class Interact keyed by their lowcase name (interact.name). Remember the lowercase, may trip you up if referencing upercase version in the file.
5. QUESTS #Quest statuses. This is a dictionary of flags (1 or 0) for the status of the quest keyed by quest name.
6. ENEMIES #All the npcs. This a dictionary of objects of class Enemy keyed by their lowcase equipment name (item.name.lower()). Remember the lowercase, may trip you up if referencing upercase version in the file.
7. GAMEINFO #Miscellaneous game info. Dictionary of all sorts of variables

You can 5 of the main variables are dictionaries which store the instance objects of all the different classes for the whole game. PLAYER is the main player object that is your player and MAPS is a special array that is indexed by it’s [x][y][z] coordinate for example MAPS[x][y][z] is one location (to be honest I’m not sure exactly how maps works but I’ll let you know when I do).

The dictionaries are filled with instance objects from each of the classes seen bellow. They all have different attributes (object variables) and methods (object functions) which a full list that can be seen in the GameClasses.py file.

1. class Equipment:
2. class Character:
3. class Enemy:
4. class Interact:
5. class Map:

Knowing these basics, seeing how it’s done in the code, and following the description in the Github readme (see copy pasted below) you should be able to follow how to add things to the game (but ask questions if you don’t).

In general try to keep this structure and put any other long ascii display or mode into another file:

EngPhysAdventure \_\_\_\_ = the setup, main loop, and ending. Run this to run the game.

GameFunctions.py = The main mechanics of the game and the quests. All non-class functions. Where the global variables are dfined

GameClasses.py = Class definitions and their coresponding methods.

Startup.py = All the map locations, items, npcs (called enemies), and interactables are defined. Also creates the dictionaries of them.

AsciiArt.py = Where all of the ascii art display files are

## The Game Structure/execution

Because the game as become much more non-linear with the ability to restart, save, continue at the end, restart at the end, and even play the game in the game I want to outline how it now runs from variable initialization to how the game runs while you’re playing it.

### Game Initialization

### Game Structure

One thing to say about the game right now is it seems very unrobust. If varaibles and definitions aren’t in the right order in the code the game will break or glitch. This makes me suspect that with increasing complexity the game will become even more finicky which may be able to be solved with better organizations/functional code. If there’s anything you see that looks bad/unrefined/can be done better please feel free to change it in a spate branch. I am by no means the most experienced at writing code, using python, or software development. In the future to make it easier manage the software development side I’d like to use better defined test cases, follow a proper python style guide, and think about code optimization

-Brendan Fallon

## Best Strategy to Win for speed running or otherwise

If doing Tyler Kashak just beat all the people. If otherwise should do the nuke quest first as it’s the easiet to get the gamma glove before having to face the tough Kenrick and Dr. Soleymani. From there best to do the silicon quest and then the optics last. Can get it down to five minutes if you want

-Brendan Fallon

# 0.26.1 – SpeedRun Update