**Chapter 8 – Save/Load and Room Generation**

The end is approaching and our game is close to being complete. If you have been able to keep up until now, you are doing a great job. This final chapter should be a breeze if you were able to overcome the last 2 comfortably. Without delay, let’s move onto the main game.

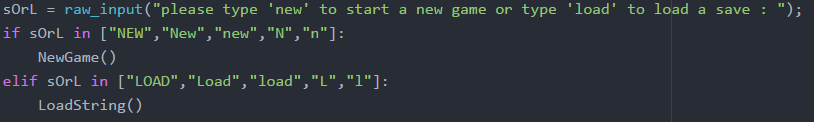
Room generation

We will create a function game() and declare it like all the previous times we have crated functions (remembering to call in the global variables).

We will start by issuing a few print statements, indicating the game has started:

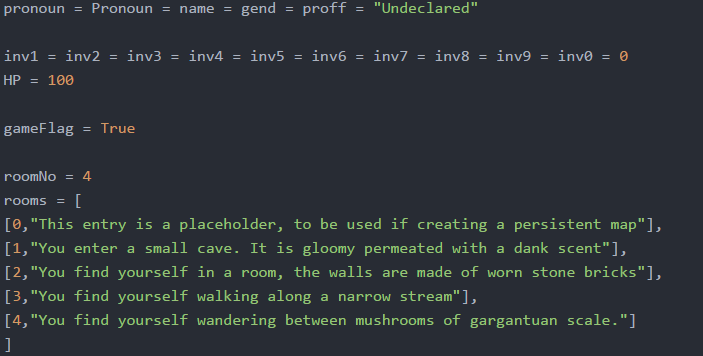
Screen Clipping

Next we will ask them if they wish to start a new game or load an existing game. If they choose a new game we simply call the NewGame() function we made in **chapter 4**. If not we will call the LoadString() function which we will make later. Therefore we should get this setup:



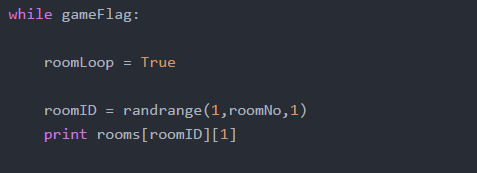
Now if recall previous chapters, we had a gameFlag variable to keep track of whether our game was active or not. This is most often updated when the player dies where we set it to false. This is where we will utilise the variable and create a loop that repeats while it is true.

At this point you should create some rooms based on the theme of your game and all you need to do is just give them and ID and a description. We want there to be global and therefore we will put them at the beginning of our code:



Since we are here, it would be a good time to also add our gameFlag variable as true as well as setting all other values as a default.

Now that we have our list of rooms, we use a random number generator again to choose one and then print the decription:

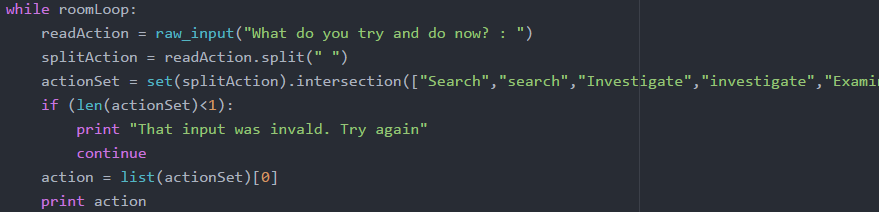


Note that we made a roomLoop variable here so that we know if we are still in the same room. Whenever we need to exit the room, we will set it to false and if the gameFlag is still true, then another random room is chosen and so on.

Now that we have chosen a room, we need to ask the user what they want to do in this room. To do this what we will do is ask for an input, see if any of the words used match our dictionary of commands and then choose the first commend they have entered. To do this we will a particular set function and split function.

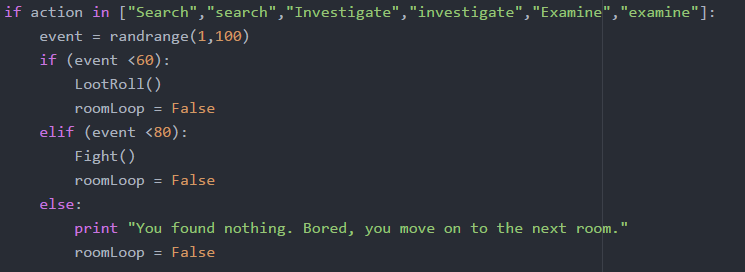
These are the words we will look out for:

["Search","search","Investigate","investigate","Examine","examine","Next","next","Hurry","hurry","Avoid","avoid","Fight","fight","Run","run","Help","help","Inventory","inventory","Save","save","Quit","quit"]

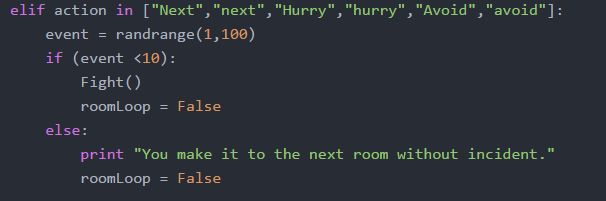
This is what the code will look like:

1. Ask for the input
2. Use the ‘.split’ function on their input. What this does is split a string into a list based on a specific character you choose. Here we choose the split the string based on spaces, hence giving a list of words
3. We then use a set function (set(list1).intersection(list2)). What this does is compare the two lists and makes a new list where only elements that are shared by both lists are included. The first list is the list of words the user input and the second list is our dictionary of commands.
4. If the new list we get has no elements (i.e. none of the words matched) we tell the user that we don’t recognise the word. Then we use the keyword ‘continue’ which essentially resets the loop from the beginning.
5. If they do enter a matching word, we take the first word from the list and use that as out action.

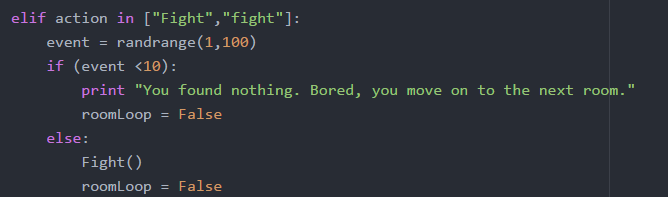
Now that we have a matching word, we just have to crate if statements that cover every possibility from our dictionary. Each command will have a bunch of scenarios what can happen and we will use a random number generator with different possibilities having different chances of occurring. Bear in mind the chances of an event happening is up to you!. Lets start with searching:



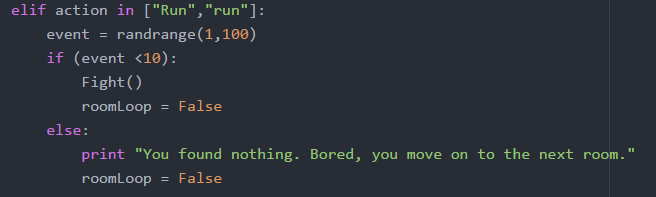
For our first scenario we choose a random number between 1 and 100. We give them a 60% chance of a loot roll, 20% chance of them running into a monster and 20% chance of finding nothing. You should notice that we have already made functions for all these events in previous chapters so all we have to do is call them. Also after event, we make the ‘roomLoop’ false so that once the event is over, a new room will be generated.

Our next scenarios is moving to the next room:

Just so it isn’t too easy, we give a 10% chance of a fight occurring.



If the user wishes to fight there is a 90% chance of the event occurring.

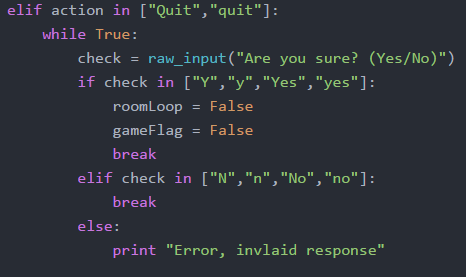


If the player tries to run we set a 10% chance of a fight.

Screen Clipping

Screen ClippingIf the player wants to see their items, we just call our PrintInventory() function. Since they would still be in the same room, we don’t set our room loop to false.

If the user needs help, we give them a list of words they can use. If you want the player to experiment more, you can give hints instead of the actual words.

We should also provide the player an option to quit. Do do this all we do is set the ‘gameFlag’ and ‘roomFlag’ to false:

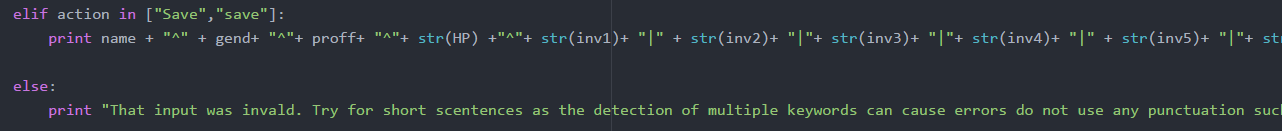
We continuously ask until we get a valid response and once we do, we use the keyword ‘break’ to exit the loop.

Our final scenario is to save the game to do this we will save the game in a string which follows a strict format. This helps us later on when we want to load the game.

Save/Load

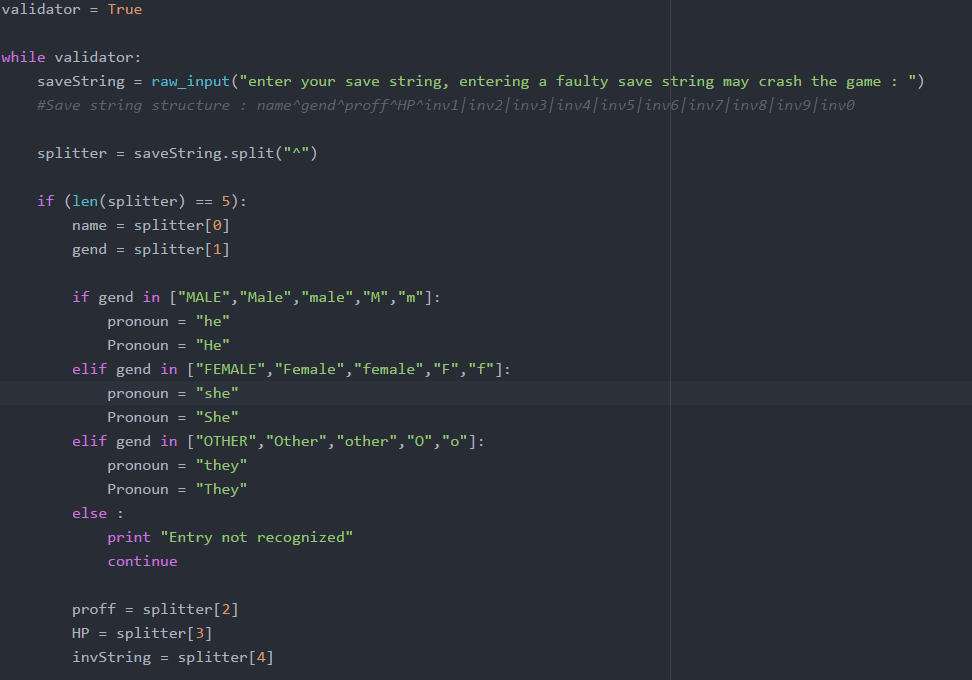
The format we plan to use is that every non-inventory item will be separated by a ‘^’ and every inventory item will be separate by ‘|’. Therefore the string we print will look like this:

name + "^" + gend+ "^"+ proff+ "^"+ str(HP) +"^"+ str(inv1)+ "|" + str(inv2)+ "|"+ str(inv3)+ "|"+ str(inv4)+ "|" + str(inv5)+ "|"+ str(inv6)+ "|"+ str(inv7)+ "|" + str(inv8)+ "|"+ str(inv9)+ "|"+ str(inv0)

And don’t forget to add that to the end of the input section:

Just for safety we have added an extra else at the end in case they input an invalid option. Full statement is in the chapter documentation.

Now our game() function is fully complete and the only thing left to do now is to write the load function. We will call it LoadStirng() and set it up like our previous functions. We will have a validator variable that will allows us to tell the user if their input is wrong. We do this twice by first checking if the first portion of the save has 5 elements and then seeing if the inventory has 10 elements.

We will use the ‘.split. function we used last time to produce a list of elements. First we will split by ‘^’ which should give us a list with 5 elements.

Since we know the order of our items, simply assign the values from our list into the appropriate variabes. The final item of the list (invString = splitter[4]) is our inventory string. Se will just follow the same procedure but this time using ‘|’ in our splitter:

Remember how we had if statements to check the length of each section – at the bottom we add the else cases which simply tell them to try again if there is a typo.

Now all the correct variables should be loaded and the player can start the game from where they left off.

THE END

Well done for making it this far, our game is at the point where we can play. We just need to add one single line right at the end to ensure our game starts up:

Screen Clipping

If you don’t remember how to launch the game, refer back to the introduction chapter which helps you get set up with Python and how to run files.

Also remember to go back and have a go at some of the extension tasks. They really help develop the game further so that it looks more polished with further features. Don’t forget, this is your game so if you have an idea you should try to add it into the game. If you are not sure how to add extra features, you can try find other tutorials in Python which may help you accomplish your goal!