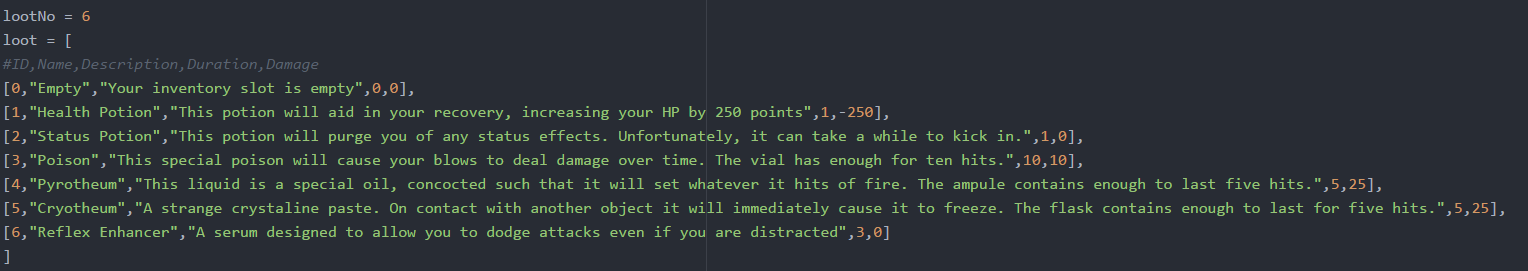
**Chapter 5 – Inventory and Loot**

From the programming side of things, we want to keep the inventory simple and easy to manage while in combat. If we cast our minds back to **chapter 3** where in the first example we showed the following list:



Currently we have 6 items and the comment in the code tells us what each part means. If you are struggling to understand the above section, feel free to return to **chapter 3** where it is explained in depth.

Notice each item in our game has a *unique* ID number in the first element of each sublist. That number will be assigned to an inventory variable if it is present within that slot. If the slot has no item, we will use 0.

The first step is to globally declare each inventory slot and when the game is launched we will make each slot 0 be default. If the player want to load a save then we can change that through the use of a function but when we create the variables we want a simple placeholder value.

If we want to declare the inventory slots globally we must make sure it is not in any function and it is good practice to put this before any functions within the code i.e. near the top. You can assign each variable on a separate line by giving it the value 0 but there is a shorter way of writing it in Python:

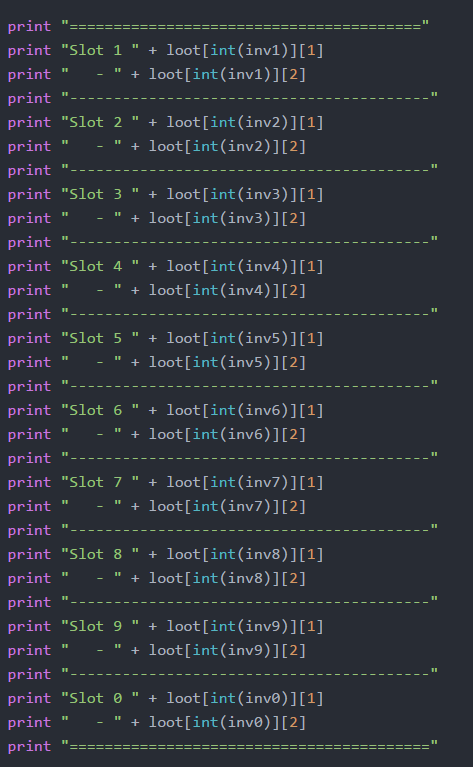
Screen Clipping

The list at the beginning of the chapter should also be global so applying the same idea, we would also write this at the beginning before any functions. The items themselves are once again up to you because they should fit the theme of the game but if you are unsure about what values to use just put in some rough estimates. Once you have completed the mechanics of the game, you can do testing and readjust the values later depending on how hard you want the game to be.

Inventory display function

We want to allow the user to check their inventory at various points in the game and creating a separate function to do that is the best course of action. We will name the function ‘PrintInventory’ and set up the function like how we learnt in **chapter4** where we remember to call in the global variables.

How you format the inventory is up to you but we will focus on how to show the correct information for each slot. Here is our example:

Here we use print statements in a logical sequence which generally follows the idea of:

*Slot number – Item Name*

*Item Description*

*Dash Divider*

So in your position the hardest thing to understand is how to make sure the correct item is return for each slot.

If you look back to our list of items, the ID of each item is also the position it lies in the list ‘loot’ (noting list indexes start at 0). Since we also use that exact same number whenever an item of that type is held we can use the numbers stored in each ‘invX’ variable.

Lets say our ‘inv1’ slot contains a health potion. Therefore the value in inv1 = 1.

Remember how our ‘loot’ list was a list within a list. This is also known as a 2 dimensional list which can take 2 indexes (list[x][y]). The x value will tell us what sublist to look and the y value will tell us which element in that sublist we should look at.

Essentially you can think of them as coordinates.

So back to our example, if we have a specific number in ‘inv1’ the x number will be the number itself because our sublists within ‘loot’ are ordered on the same number scheme. Once thing we have to do to make sure no errors appear is to cast the value to an integer so the computer knows it’s a number. This is done by writing int(variable).

So for the item in inventory slot one would be references like this loot[int(inv1)][y]. Our next goal is to figure out what we should put in the place of Y.

If we go back to our print structure we said we wanted to print the item name and the item description. If we go back to the comment in our list ‘loot’ our second element was the item name and third was item description. Hence the indexes would be 1 and 2 respectively and I would use loot[int(inv1)][1] and loot[int(inv1)][2]. Using this information I have made my format for displaying one inventory slot above.

Next we just repeat for all the other inventory slots, remembering to change ‘invX’ for each slot.

Loot Randomisation Function

Since we are still on the topic of loot, here would be a good point to also create a function that handles loot drops when a player searches a room or wins a battle. By making a function it becomes easy to just add calls to the function whenever we think a loot event should happen.

We want our loot to be random selected but as a programmer implementing randomness is difficult. Luckily there is a library in Python where someone has made their own functions which we can use. This library is called random and more specifically we want the randrange() function.

To do this we write:

from random import randrange

as the very first line of our program.

The randrange function has 3 parameters which you must give (randrange(x,y,z)). The ‘x’ value is the lower bound, the ‘y’ value is the upper bound and ‘z’ is the step interval within the range. If we only want to produce whole numbers from our range, our step interval would be 1.

Since we have a variable containing the number of possible loot items our randomiser will look like the following

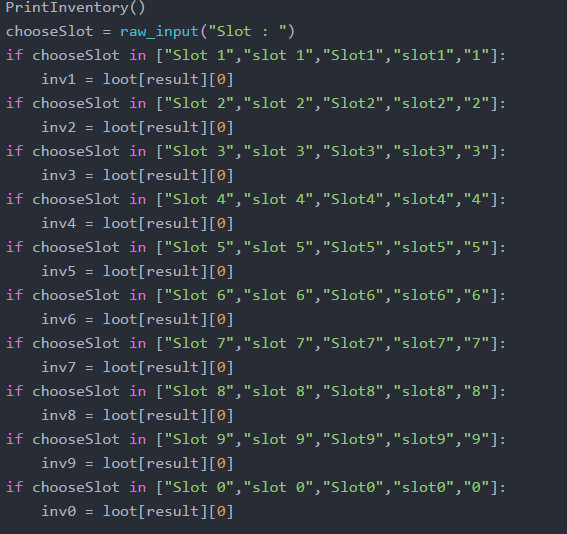
result = randrange(1,lootNo,1)

This very useful as we can just change one variable and our lootroll() function does not have to be updated if we add extra items.

The next step is to tell the player what item they got which is done using the same principles we learnt in the first party of the chapter.

print ("You found a " + loot[int(result)][1] + " where would you like to store it? putting something in a filled slot will destroy the previous object.")

To make things easy, we will show what the player already has and guess what, we simply just call our ‘PrintInventory’ function. Then we will ask which slot they want to choose and store it in a variable. Using a series of if statements we eventually put the item in the correct slot. Everything below should be familiar if we use everything learnt in the chapter:

the slot is stored in ‘chooseslot’

an if statement with a list is used to check for their input

if there is a match we assign the inventory slot with the correct number.

And with that our lootroll() function is complete.

The next chapter will be one of the hardest and will require two chapters to be covered. In terms of programming knowledge, nothing new will be required but it does require a strong understanding of everything covered so far. Feel free to go back to previous chapters and try mini exercises until you are confident.

EXTENSION

Currently all items of the loot have an equal chance of being produced but as the game designer you may want to make it harder to get certain items. This is called adding weighting to items and adds more depth to the game. Can you think of how to implement it? (If you are struggling, a hint is provided in the code for this chapter on the Github link. Alternatively, you can return to this chapter later on as future chapters will teach you how to do this.)