Mr. Jones’ Dark and Dangerous Dungeon

# Introduction

You are thrown into darkness, stumbling as you land on shaking feet. There is a glow, a smell of burning and a strange warmth. You look around trying to locate the source of the pale red light and the faint but strengthening smell. The room is dimly illuminated by the glow behind you. As you turn, the dim illumination turns with you...always remaining behind you...very strange.

Wait!

What!?

Your trousers are on fire!

You beat franticly at your backside and put out the smoldering material. Phew. That was close...and getting extremely uncomfortable. There is now a cool draft blowing through the hole in your trousers. At least that is better than the heat that was increasing rapidly before. The fire must have been from a stray spark of magical energy, a leftover from the spell that bought you here.

You are now in total darkness.

In your pocket you feel a box of matches. You strike one. The light of the match flare surprises you. You look around quickly. In the dim light of the flickering flame, you glimpse glints of light reflected from the damp dripping rock walls. All too quickly the match burns out and you are plunged back into the dark once again.

You strike another match. This time, you were expecting the flare and have time to catch sight of a pile of rags wrapped around a wooden handle, lying on the floor. It’s an unlit torch. Your match burns out and the light dies again. You have one match left.

You sink to your knees and inch carefully over to where you saw the torch. Reaching out around the cold, damp floor you feel for the torch, hoping it is not too wet to light.

Your fingers reach out and you grasp the torch. You stand unsteadily on the uneven, rocky floor. With your last remaining match, you light the torch. It is bright, and smoky (the smell...it reminds you of something...oh yes – your trousers). The room...or chamber you now realize...is lit by flickering light with shadows dancing over stone walls hung with mosses, molds and other unpleasant things. Dank, green-black and ominous. To your left, a dark passage leads off to who-knows-where. To your right a stout oak door stands slightly ajar. The room on the other side, equally dark and forbidding. What do you do?

# Design and Dice

You will write a text-based adventure game that will allow the user to explore Mr Jones’ Dark and Dangerous Dungeon. The user will meet strange monsters and collect magical artifacts. The monsters are terrifying and dangerous (hence the name of the dungeon...it’s also dark).

Will the user get out alive or will they be consumed by one of the terrible beasts that the mysterious and mighty Mr Jones keeps patrolling his domain?

There is a lot of complex coding here that I want you to try to work out as you go along. You will need to learn about and to understand the following:

* Input
* Output
* Variables and data types
* While loops
* For loops
* If Statements
* Sub programs
* Python dictionaries
* Python lists

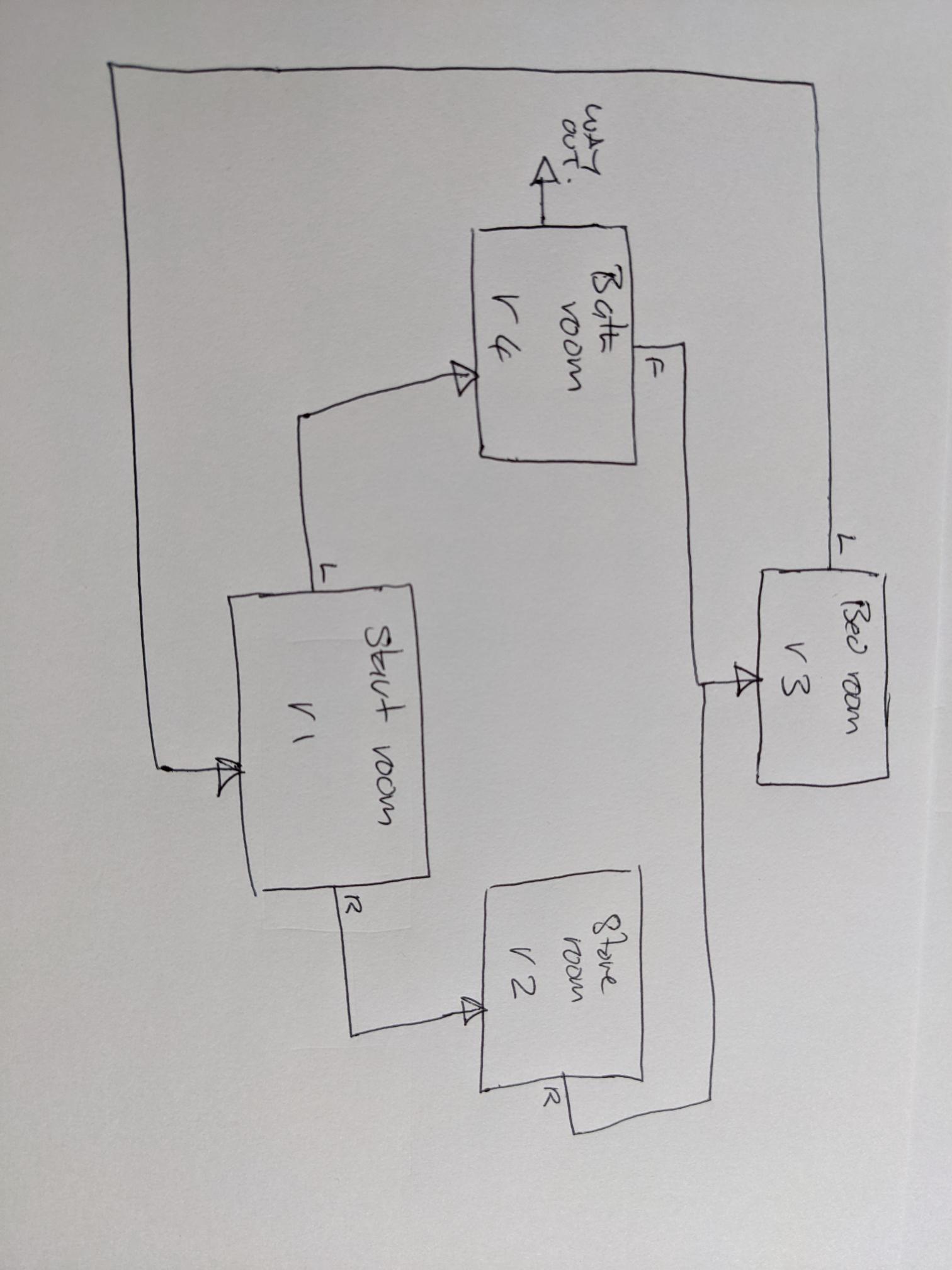
DON’T PANIC – this will probably take you all term because you will be learning as you go. There is plenty of help available. Work through the project rather than jumping to the end. Read the help sections in w3school so you understand the functionality we are using. As we will use lots of subroutines, start here:

<https://www.w3schools.com/python/python_functions.asp>

If you run into difficulties, read the error message, talk to other people writing their versions and then, if still stuck, ask me.

The first thing you need to do is to design your layout and your characters. This is an essential part. Keep it simple to start with. You can add more rooms, monsters, and objects later to make the game bigger and more complex but start simple to get it working.

Here is my maze:



## Design your map, objects, and monsters.

Start small to get your program working. You can always add more monsters, rooms etc later. Just have 3 rooms, monsters, and objects to start with. All rooms must have the same attributes. We will store this data in a structure in the program called a dictionary (a complex variable data type). The attributes need to be:

* Name The unique name of the room
* Description A suitably terrifying description of what the user sees when they enter the room
* LeftRoom The name of the room that can be accessed on the left. This can be “none”. IF it is not none, it must be the unique name of another existing room or “Way Out”. This is the end of the game – you won.
* RightRoom The name of the room that can be accessed on the right. This can be “none”. IF it is not none, it must be the unique name of another existing room or “Way Out”. This is the end of the game – you won.
* ForwardRoom The name of the room that can be accessed in front of you. This can be “none”. IF it is not none, it must be the unique name of another existing room or “Way Out”. This is the end of the game – you won.
* Objects This is a list (an array) of object names in the room. Each must be an object that your program creates (we’ll come to this in a moment).
* Monsters This is a list (an array) of Monster names in the room. Each must be a monster that your program creates (we’ll come to this in a moment).

You will also need a set of data for a room called “Way Out”:

Objects are also stored in dictionaries. All objects must have the same attributes:

* Name The unique name of the object
* Description A suitable description the user will see when they encounter the object
* Damage A number indicating the amount of damage the object can do to the monster. If negative, it will make the monster stronger (you need to use another object to defeat the monster).
* Health Add to user health when object is picked up – food or magic potion. Can be negative for poisons.

Monsters are also stored in dictionaries. All monsters must have the same attributes:

* Name The unique name of the monster
* Description A suitably terrifying description the user will see when they encounter the monster
* Damage The amount of damage the monster will do per hit
* Health The starting amount of health of the monster. When it reaches zero, the monster dies

Once you have designed your rooms, objects and monsters you need to start writing your program.

## Start your program

Open IDLE and import the random library. This is a library of different python functions that allows you to generate (amongst other things) random numbers. You will need to work out the health that you start with (how much damage did your burning trousers do to you). Your health will be determined by a dice throw.

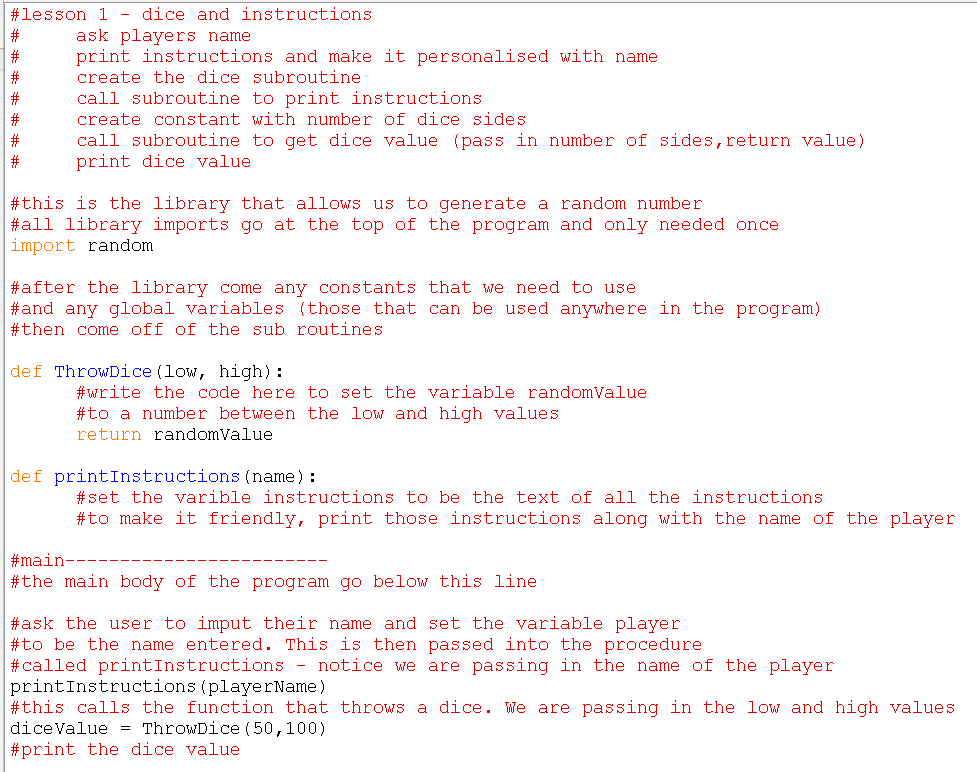
You need a whole number (an integer) between 50 and 100. Below is the structure of the program but I want you to write the code to get a random integer between a low and a high value. This will help you

<https://www.w3schools.com/python/ref_random_randint.asp>

Ask the user to input their name. Use this name when printing out the introduction and instructions.

Call the procedure to print the instructions and print the value of the dice thrown to make sure that your program is working.

Template for your program. Note the subroutines ThrowDice() and printInstructions():



# Initialization of the player, rooms, objects, and monsters

Each of these things will be stored in a dictionary and your program will rely on each thing having the correct number of attributes and those attributes having values that match the names of other things such as room names.

## Dictionaries.

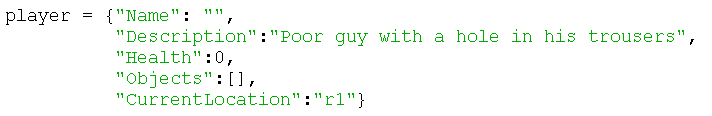
Read the following and, in your program create a dictionary for the main character – The player who will negotiate the dungeon.

<https://www.w3schools.com/python/python_dictionaries.asp>

Your player dictionary will need the following attributes:

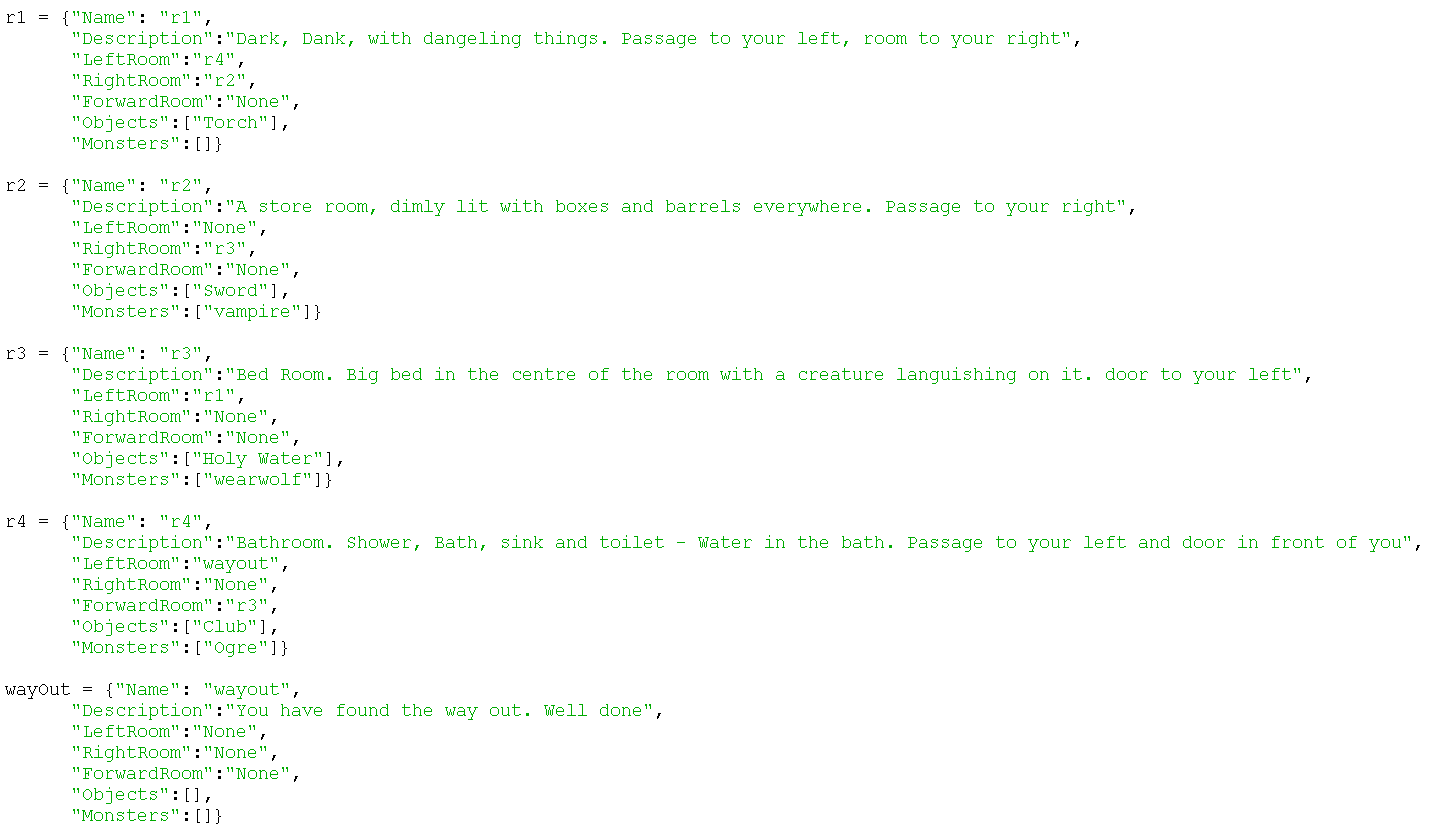
* Name Your name. This is set after the user enters their name
* Description A brief description, including a hole burnt into your trousers
* Health Your health. This is initially set to a value between 50 and 100 after the first dice roll. This will decrease with a hit from a monster and will increase when you find food objects
* objects a list of objects that you have found.
* CurrentLocation Current room name

Create the player dictionary at the top of your program (below the import). It will look something like this:



In the main body of your program, Set the player name attribute equal to the name of the user and the health attribute equal to the diceValue of the first throw.

This is the room data for my maze:



At the top of your program again, below the import but above all of the subroutine definitions, create dictionaries for each room, object, and monster. Don’t forget that the left, right and forward room names in each room dictionary must be the name of another room that you have created. E.g. “r1”, “r2” etc.

Create a dictionary for each object. Remember to only use the attributes above and make each object name unique and the same as the dictionary variable holding the related data.

Create a dictionary for each monster. Create a dictionary for each room. When creating the left, right and forward rooms, make sure they are either existing rooms or say “None”.

In the main body of your program, print out each of your rooms, monsters, and objects to make sure they all look right. You can always remve these temporary print statements later when you know everything is working.

In the main body of your program, create a variable called currentRoom and set this to your first room - r1. This will be a dictionary because you have set it equal to a dictionary. This is different to the currentRoom attribute in the dictionary for the player.

# Game loop and function keys

## Game Loop

Every game needs a game loop, whether it is Halo, Minecraft, Call of Duty or Chess. This will go around repeatedly, waiting for the next move or action from the play until the game is over. For our game, the game will stop when the player dies (player loses) or the player finds the way out (player wins).

Because we don’t know how many turns the player will take, we need to control our loop with a condition (or more than one condition). The loop needs to repeat until either the players' health is zero (or less) or the player finds the way out. We will therefore use a while loop as this is called a condition-controlled loop. Read the following to understand loops (note: we will not use “Break” or “Continue” in our code as this is poor coding practice)

<https://www.w3schools.com/python/python_while_loops.asp>

In the main body of your program add your while loop. This needs to loop while player[“Health”] >0 and player[“CurrentRoom”] != “Way Out”. Inside your while loop you will need to:

Print out the current room details

Ask the user what they want to do

## Validating input

The player must only be able to enter L,R,F, B or I. These will perform the following functions:

L – go to room on the left if it exists

R – go to room on the right if it exists

F – go to room in front if it exists

B – go back to previous room (functionality that we won’t implement yet)

I – print out the details of the current room

You will need to write a subprogram that asks this, validates this input, and only returns a value if it is one of those five. If the user enters a value that is not one of those five, the program will say that this is wrong and ask again. This will need a validation loop in the verification function. Think about how to do this by writing the things that need to happen down on paper. You will need to go into a loop and stay in that loop until a valid code has been entered. Once you have cracked this little (but complex) piece of code, you can use this technique to validate data entered for the rest of your computing career – it's an import code snippet.

Your validation subroutine will be a function that returns the value entered. First create a boolean variable and set it too False. Next start your while loop that will repeat until that variable is True. Inside the loop ask the user to enter the allowed values. If the value input is one of the allowed values, set the condition variable to True (this will stop the loop). If the user has not entered a valid value, print out an error message. This will keep the condition value as False and repeat the ask. Outside the loop, return the value entered. I have called this validation routine getAction().

# Actions

Everything we do for the next sections will be within the game loop. There are many things to do, most of which will need their own subprogram. Many subroutines will call other subroutines.

## Information

If the user enters “I” for their action, you will need to print out the information for the current room. Create a subprogram called printInformation() and pass it the currentRoom variable:

printInformation(currentRoom)

This will print out the description from the currentRoom[“Details”] attribute.

## Movement

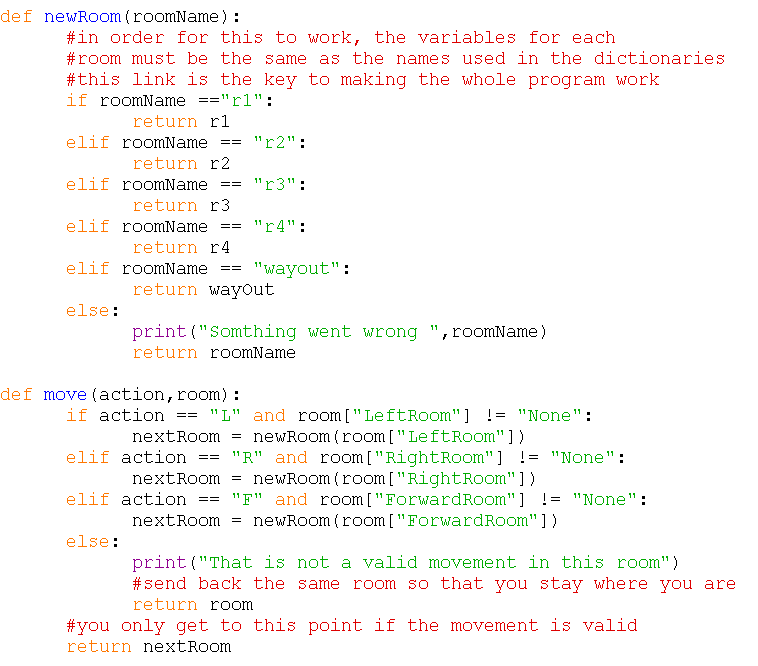
Add elif for L, R, F and B. Add an else that prints out “Error unknown action”. If your validation works successfully, you will never get here but it is a “belt-n-braces" action...just in case.

For L, R and F you will need to check that the current room has a room available at that direction. If not, you will need to print an error message. If so, you will need to move to that room by setting the currentRoom to the variable with the same name as the room in that direction (see below).

## The Next Room

To move to the next room, you need to find the room name in the correct attribute of the current room. For example, if you type L, you will need to get the room name in the LeftRoom attribute and then return the dictionary for that name. This will need a couple of new functions.

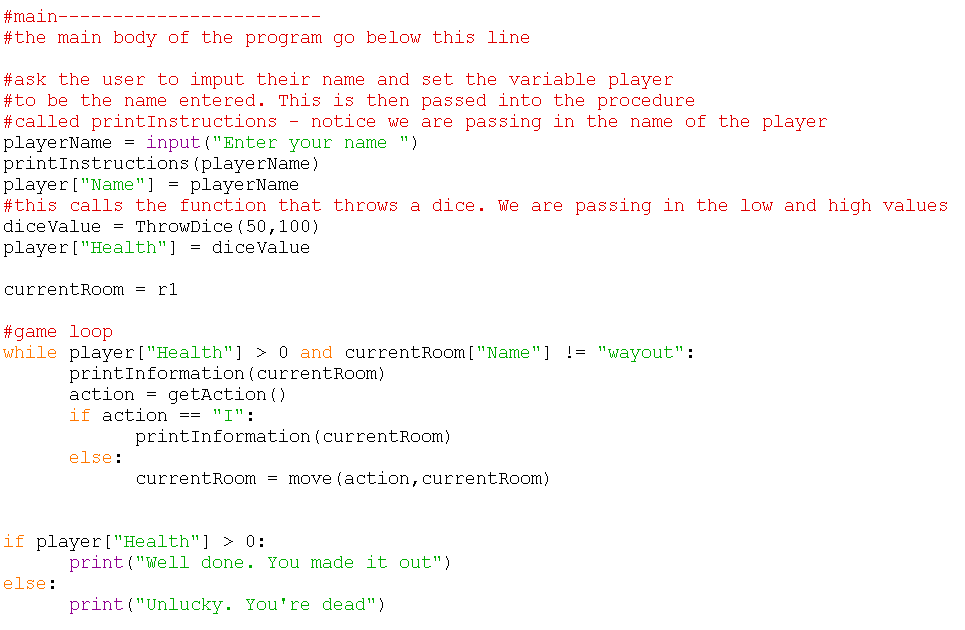
Here are the two funtions that control your movement (note: The green “r1” Is a string value, just a name. The black r1 is the whole dictionary with the same name.:



Note that they reference the actual names of the room dictionaries that you created before. It is essential that this all matches up.

# Objects

Your main game loop should now look like this (note that I have added the win/lose “if” statement at the bottom):



As you move to a new room, you will want to examine the objects that may be in there. Make a change to the printInformation() procedure so that it now prints out the details of any objects in the room. If there are objects in the room, you can ask the player if they want to pick them up. If the user enters “Y” then add the object name to the list of objects that the player has - player[“objects”] and delete it from the list of objects in the room. For example: r1[“objects”]. For this you will need to look at appending to a list and deleting from a list.

<https://www.w3schools.com/python/ref_list_append.asp>

<https://www.w3schools.com/python/ref_list_remove.asp>

## Picking up objects

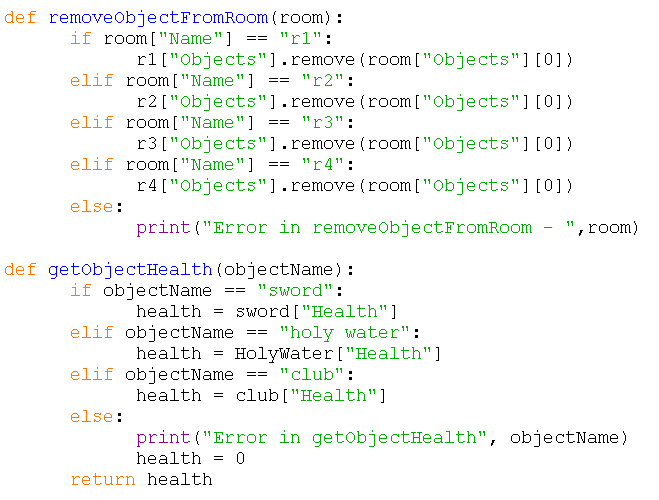
In the “else” section of your main game loop, add a subroutine to get any objects in the room. If there are, ask the user if they want to pick them up. If they pick them up and there is a health change, alter your players' health by the amount on the object. You could, if you wanted, add a dice roll here that would increase the health impact. If you do this remember to set the lower and upper bounds of the roll to 1 and 6 respectively.

In order to find out if there are objects in the room, your program can find the length of the list that holds these objects for example, r1[“Object”]. If the length of this list is greater than zero, your program can ask the user if they want to pick up the object.

<https://www.w3schools.com/python/ref_func_len.asp>

Write a new validation routine to check that the user only enters N or Y when asked if they want to pick up the object or not. This will be similar to the validation routine that you wrote for movements. Call your new function inputYN(text). The text input parameter is the test that will be used in the input command within the function. It will return only Y or N and loop until the user enters one of those two options.

The subroutines relating to objects also need to specify the object name. Here are the routines for removing an object from the room data and getting health from the object data (note the error handling):



# Fighting monsters

## Battling monsters

As above, this will be a subroutine that is called from the same place in the game loop as the object collection. It is up to you whether you want this before or after the object selection. If before, you might die before getting the object that could save you! If there is a monster in the room that you enter, it will attack you straight away. You will need to throw a dice and calculate the damage it does. You will also need to ask the user which of their objects in their list they want to use to battle with. They can only use what they have. Throw a dice, multiply by the damage attribute of the object and reduce the monster's health down to zero. Once zero (either the monster or the player) the battle is over.

As above, you will need code that gets the dictionary for the specific monster based on the name of the monster in the room[“monster”] attribute.

## Selecting a weapon

You will need to code a subroutine to get the weapon and another to get the damage for the weapon that you want to use to attack the monster. This means that the user will need to choose a weapon that you have in your list. You will need to use a for loop to loop through the list of all of your available objects.

<https://www.w3schools.com/python/python_for_loops.asp>

<https://www.w3schools.com/python/python_lists_loop.asp>

If you manage to kill the monster, it will need to be removed from the room. Use a subroutine similar to the removal of objects above to remove the monster from the room.

Print the list index alongside the weapon (use a counter variable), you can then ask the user to input the ID of the weapon they want to use (as an extra you should put validation around this input).

Then you will need another subroutine to get the damage that the selected weapon can do.

## Battling the monster

Once you have your monster details and your weapon damage you can set up a conditional loop to repeat the attacks.

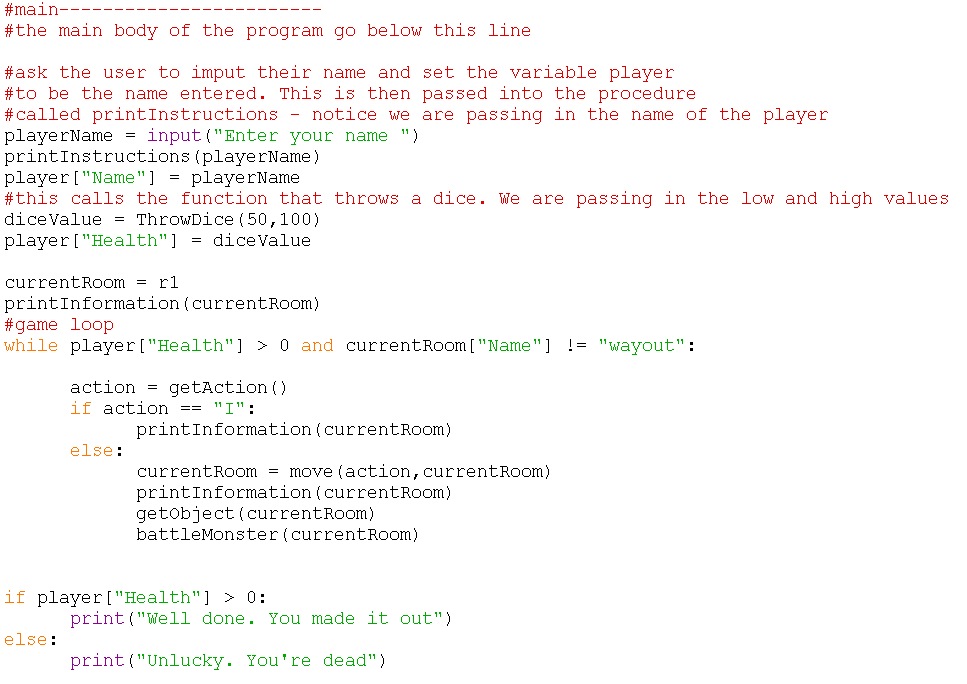
while monster["Health"] > 0 and player["Health"] > 0:

For each attack, the health of either the player or the monster needs to be reduced by the dame \* dice roll. The loop will stop once either the monster's health or the player's health is zero or below.

Print out suitable information such as the health of both parties after each attack.

# Final Program Outline.

Your main section and game loop will now look like this:



You will have data for Rooms, Objects, and monsters

You will have a list of subroutines like this:

def ThrowDice(low, high):

return randomValue

def printInstructions(name):

def getAction():

return action

def inputYN(text):

return answer

def printInformation(room):

def newRoom(roomName):

return room

def move(action,room):

return nextRoom

def removeObjectFromRoom(room):

def removeMonsterFromRoom(room):

def getObjectHealth(objectName):

return health

def getObject(room):

def getMonster(monster):

return Monster

def weaponDamage(weapon):

return Weapon

def getWeaponDamage():

return weaponDamage(weapon)

def battleMonster(room):

# Extras

There are many ways to improve your program. Here are some suggestions

Very Easy – Change the language in things such as the monster battles so that it is more dynamic, and action orientate. For example, rather than “the monster attacks you” make it “The monster dives at you, fangs claws and weapons whirling viciously at you”,

Easy – Add more rooms, objects and monsters. You will need to make changes to a number of subroutines as well so that they can reference the new names.

Easy - Multiply health by dice through when picking up an object

Easy- Add something to say to the monster.

Easy- Add something for the monster to say back.

Medium – Add validation for weapon selection. This needs to be an integer only and must be from the list provided nothing else.

Medium – Amend the room description when the monster is killed to say that the room now has a dead <name of monster> in it.

Medium – Add multiple monsters per room or multiple objects per room.

Medium - Add specific weapons that impact different monsters in different ways. For example, the holy water may do damage only to the vampire and just get other monsters wet! This will need a new attribute to object dictionary that can take the name of the monster, it is effective again. “ALL” for every monster or “None” for objects that are not weapons such as food.

Medium – Add the “Back” functionality. You will need to create a new variable in the game loop to hold the previous room.

Hard – put validation routines into a separate library and import them as needed

Hard - Make the map data into a library function so you can swap between different maps. Ask the user which map they want to use.

Very hard - Rewrite the whole project using object orientated coding to get rid of the hard coded rooms, monsters, and objects.

Hard. Needs OO first - Put all map data into a file.