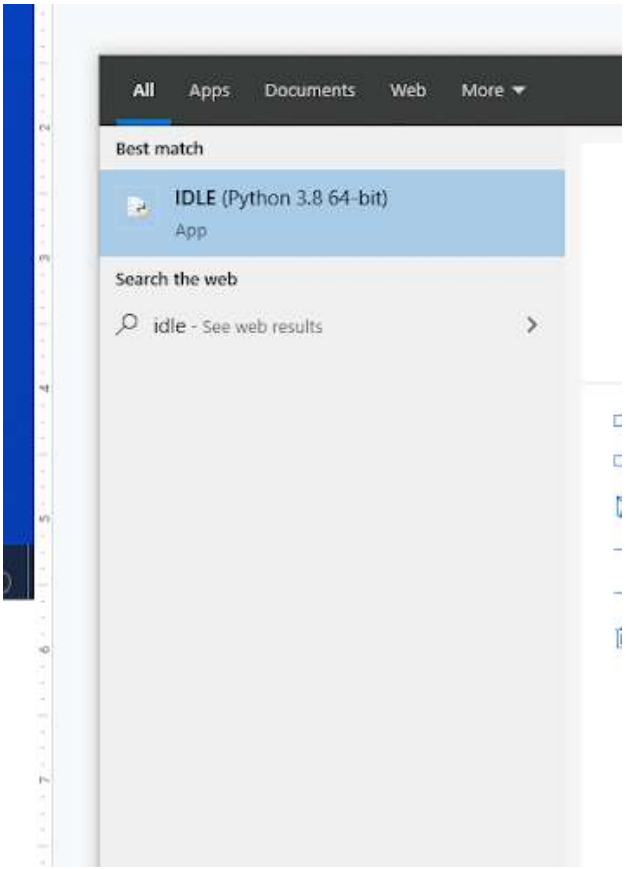
**Part 3**
**Space Mission Directions**

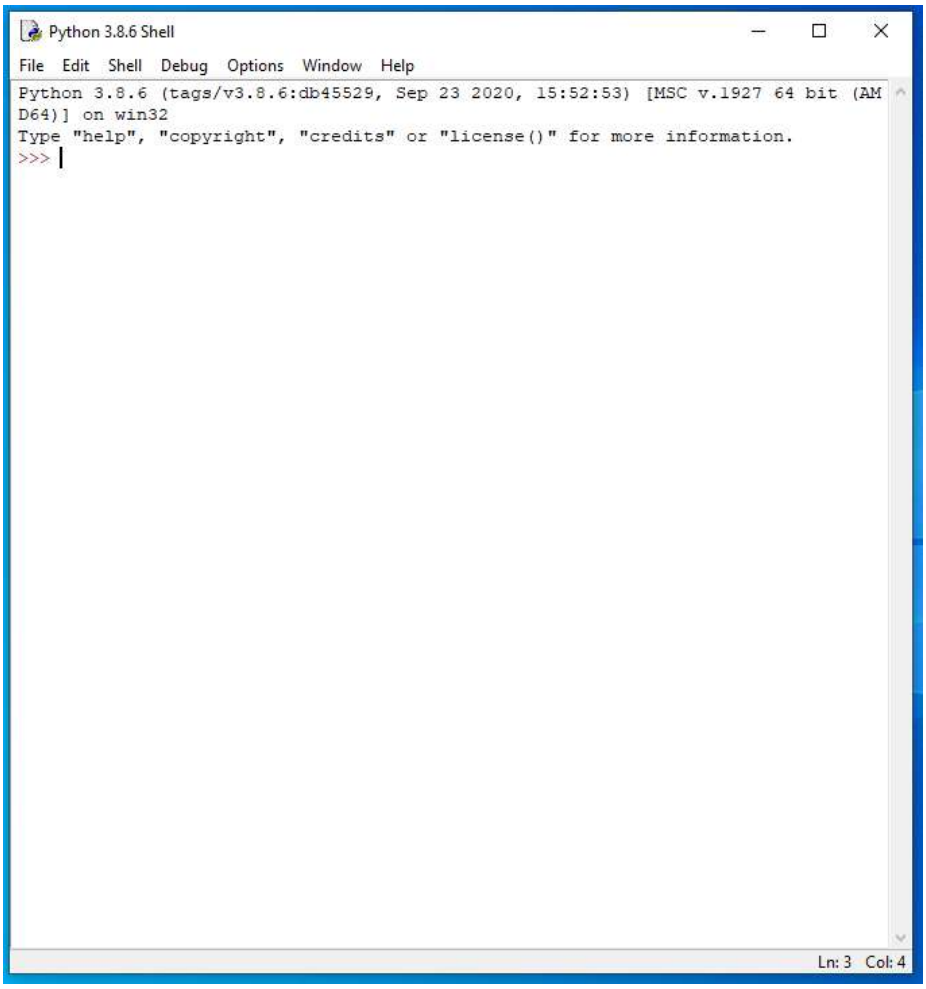
1. Navigate out to the Google Classroom for this class.
2. Locate the Space Mission Part 3 assignment.
3. Click on the, “Chapter 6 Scenery Code to Copy/Paste” file that is attached to the assignment.
4. You will want this file later on in the exercise. Keep it handy.
5. We are now ready to start adding code to our file. Using your Windows button menu, find and launch your IDLE program.




IDLE is the integrated development environment associated with Python. It is made up of a code editor where you type your code along with other helpful tools that allow you to write, save, and test run programs.

IDLE is designed to recognize Python code, compile Python code, and provide basic debugging tips to programmers if there are problems with their code.

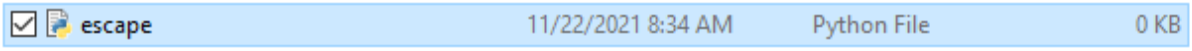
6. Your IDLE window should look something like this once it has launched.:




On Startup, IDLE will display the Python Shell, which can be used to give commands to the computer's operating system. Since we are viewing the shell through IDLE and not the actual command prompt window, the commands that we type into the Shell will not communicate directly with our operating system. However, you can type similar commands in the Python Shell directly from the Python program (not through IDLE) and, if you have permission to access the operating system's commands, you can communicate with the computer's operating system that way.

In IDLE, the shell is mainly used as a launching screen for other activities that we will do, like writing code for our game or debugging a file.

7. Go to File > Open and then browse in the Starting Files folder I gave you to find the escape python file that we have been working on.




8. Your escape.py file will open up.

9. Click at the end of Line 25.

```
21 DEMO_OBJECTS = [images.floor, images.pillar, images.soil]
22
23 LANDER_SECTOR = random.randint(1, 24)
24 LANDER_X = random.randint(2, 11)
25 LANDER_Y = random.randint(2, 11)
26
27
28 ################
```

10. Press ENTER twice.

11. Type the code you see on Line 27 of the screenshot below.

```
23 LANDER_SECTOR = random.randint(1, 24)
24 LANDER_X = random.randint(2, 11)
25 LANDER_Y = random.randint(2, 11)
26
27 TILE_SIZE = 30
28
29
30 ################
31 ##    MAP     ##
32 ################
```

Line 27 creates a new variable called TILE_SIZE. This variable's value is set to 30.

12. Scroll and click at the end of Line 230.

```
228 items_player_may_carry = list(range(53, 82))
229 # Numbers below are for floor, pressure pad, soil, toxic floor.
230 items_player_may_stand_on = items_player_may_carry + [0, 39, 2, 48]
231
232
233 ################
234 ## MAKE MAP   ##
235 ################
```

13. Press ENTER three times.

14. Navigate to the code document you were given on Google Classroom.

15. Select all of the text in the document, copy it, and then paste it starting on Line 233. It should paste code on Lines 233 – 277.

```
254           [16,1,3], [12,8,6], [12,9,4], [12,9,8],
255           [15,4,6], [12,7,1], [12,7,11]],
256       36: [[4,3,1], [9,1,7], [8,1,8], [8,1,9],
257           [5,5,4], [6,5,7], [10,1,1], [12,1,2]],
258       37: [[48,3,1], [48,3,2], [48,7,1], [48,5,2], [48,5,3],
259           [48,7,2], [48,9,2], [48,9,3], [48,11,1], [48,11,2]],
260       38: [[43,0,2], [6,2,2], [6,3,5], [6,4,7], [6,2,9], [45,1,10]],
261       39: [[38,1,1], [7,3,4], [7,6,4], [5,3,6], [5,6,6],
262           [6,3,9], [6,6,9], [45,1,11], [12,1,8], [12,1,4]],
263       40: [[41,5,3], [41,5,7], [41,9,3], [41,9,7],
264           [13,1,1], [13,1,3], [42,1,12]],
265       41: [[4,3,1], [10,3,5], [4,5,1], [10,5,5], [4,7,1],
266           [10,7,5], [12,1,1], [12,1,5]],
267       44: [[46,4,3], [46,4,5], [18,1,1], [19,1,3],
268           [19,1,5], [52,4,7], [14,1,8]],
269       45: [[48,2,1], [48,2,2], [48,3,3], [48,3,4], [48,1,4], [48,1,1]],
270       46: [[10,1,1], [4,1,2], [8,1,7], [9,1,8], [8,1,9], [5,4,3], [7,3,2]],
271       47: [[9,1,1], [9,1,2], [10,1,3], [12,1,7], [5,4,4], [6,4,7], [4,1,8]],
272       48: [[17,4,1], [17,4,2], [17,4,3], [17,4,4], [17,4,5], [17,4,6], [17,4,7],
273           [17,8,1], [17,8,2], [17,8,3], [17,8,4],
274           [17,8,5], [17,8,6], [17,8,7], [14,1,1]],
275       49: [[14,2,2], [14,2,4], [7,5,1], [5,5,3], [48,3,3], [48,3,4]],
276       50: [[45,4,8], [11,1,1], [13,1,8], [33,2,1], [46,4,6]]
277       }
```

The code above contains the scenery dictionary, which describes the scenery in each room. Scenery is equipment that stays in the same place throughout the game and includes furniture, pipes, and electronic equipment.

Our scenery dictionary will use the room numbers as the key. Each scenery item is listed in the scenery dictionary, sorted by room number. For each room number, the dictionary stores a list, with a square bracket at the start and end of it. Each item in that list is another list that tells the program where in the room to put each scenery object.

First in the list is the object number. This is the same as the number that is used as the key in the objects dictionary. For example, object number 5 represents a table (object number 5 in the objects dictionary).

The second two numbers represent the object's x and y position in each room.

The back wall is usually in row 0, so we will typically start by placing objects at row 1 for the y position. The largest useful number (on the y-axis) will be the room height minus 2 – we subtract 1 because the map positions start at 0 and subtract another 1 for the space the front wall occupies.

The number in the x-position area tells the program how far across the room from left to right the object should be. Again, a wall is usually in position 0. The largest useful number will generally be the room width minus 2.

16. Click on Line 279. Ensure your code has blank lines on Lines 278, 279, 280, and 281, as shown in the screenshot below.

```
275     49: [[14,2,2], [14,2,4], [7,5,1], [5,5,3], [48,3,3], [48,3,4]],
276     50: [[45,4,8], [11,1,1], [13,1,8], [33,2,1], [46,4,6]]
277     }
278
279
280
281
282 ################
283 ## MAKE MAP  ##
284 ################
```

17. Type the code you see on Lines 279 – 286 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
275     49: [[14,2,2], [14,2,4], [7,5,1], [5,5,3], [48,3,3], [48,3,4]],
276     50: [[45,4,8], [11,1,1], [13,1,8], [33,2,1], [46,4,6]]
277     }
278
279 checksum = 0
280 check_counter = 0
281 for key, room_scenery_list in scenery.items():
282     for scenery_item_list in room_scenery_list:
283         checksum += (scenery_item_list[0] * key
284                      + scenery_item_list[1] * (key + 1)
285                      + scenery_item_list[2] * (key + 2))
286         check_counter += 1
287
288
289
290 ################
291 ## MAKE MAP  ##
292 ################
```

This bit of code serves as a safety measure, called a checksum. It will check that all the data is present and correct by making a calculation involving the data and then checking the result against the correct answer. If there's a mistake in the data you've entered, this bit of code wills top the program until you fix it. This stops your game from running with bugs in it.

Line 279 creates a variable called checksum and sets its value to 0.

Line 280 creates another variable called check_counter and sets its value to 0 as well.

Line 281 begins a "for" loop that will run for each dictionary key and room_scenery_list item in the scenery_items dictionary list.

Line 282 begins a nested "for" loop that will run on each scenery item in the list for each room.

Line 283 begins a function that will multiply the first scenery item in the list by the key for that scenery item. Line 284 will add 1 to the key number and multiply the second scenery item in the list by that new number. Line 285 will add 2 to the key number and multiply the third scenery

item in the list by that new number. The total of all of these calculations will be added to the checksum value.

Line 286 will add 1 to the check_counter variable's value.

18. Press ENTER.

19. Type the code you see on Lines 287 – 290 of the screenshot below. Ensure you type the code all the way at the left margin, as shown.

```
275     49: [[14,2,2], [14,2,4], [7,5,1], [5,5,3], [48,3,3], [48,3,4]],
276     50: [[45,4,8], [11,1,1], [13,1,8], [33,2,1], [46,4,6]]
277     }
278
279 checksum = 0
280 check_counter = 0
281 for key, room_scenery_list in scenery.items():
282     for scenery_item_list in room_scenery_list:
283         checksum += (scenery_item_list[0] * key
284                      + scenery_item_list[1] * (key + 1)
285                      + scenery_item_list[2] * (key + 2))
286         check_counter += 1
287 print(check_counter, "scenery items")
288 assert check_counter == 161, "Expected 161 scenery items"
289 assert checksum == 200095, "Error in scenery data"
290 print("Scenery checksum: " + str(checksum))
```

Line 287 will print the value of the check_counter variable along with the text, "scenery items".

Line 288 will use the assert function to check that the program has the right number of data items. Again, remember an assert statement is used to continue to execute the code if the given condition evaluates to True. In this case, if Line 288 (check_counter == 161) evaluates to true, that means that we have the appropriate number of scenery items in our list and have not made an error. If this statement evaluates to False, the execution of the game code will stop and the game will not run. The Error Text included at the end of Line 288 will also display.

Line 289 uses a similar assert function to check that the checksum value is 20095. This is the number that the checksum should be equal to if we've included all scenery items in our data. If the number does not equal this, the execution of the game code will stop and the error text included at the end of Line 289 will display.

Line 290 will print the text, "Scenery checksum: " along with the value of the checksum variable, converted to a string.

20. Press ENTER twice.

21. Type the code you see on Lines 292 – 296 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
279 checksum = 0
280 check_counter = 0
281 for key, room_scenery_list in scenery.items():
282     for scenery_item_list in room_scenery_list:
283         checksum += (scenery_item_list[0] * key
284                      + scenery_item_list[1] * (key + 1)
285                      + scenery_item_list[2] * (key + 2))
286         check_counter += 1
287 print(check_counter, "scenery items")
288 assert check_counter == 161, "Expected 161 scenery items"
289 assert checksum == 200095, "Error in scenery data"
290 print("Scenery checksum: " + str(checksum))
291
292 for room in range(1, 26):# Add random scenery in planet locations.
293     if room != 13: # Skip room 13.
294         scenery_item = random.choice([16, 28, 29, 30])
295         scenery[room] = [[scenery_item, random.randint(2, 10),
296                           random.randint(2, 10)]]
297
298
299
300
301 ################
302 ## MAKE MAP  ##
303 ################
```

Line 292 will begin a block of code that will add random scenery to the room. For each room, the random.choice() function chooses a random scenery item from items number 16, 28, 29, and 30 (shrub, large rock, small rock, and a crater). Remember rooms 1 – 26 are outdoor rooms outside the space station.

We leave Room 13 empty (Line 293) so that your character in the game has an empty space on the planet's surface (room 13) with no scenery objects.

Line 295 also adds a new entry to the scenery dictionary for the room. This entry contains the random scenery item and the random y and x positions for that item. The y and x positions place the item inside the room but not too near to the edge of the room.

After Lines 295 and 296 run, our scenery dictionary will contain information about the scenery in every room.

22. Press ENTER twice.

23. Type the code you see on Lines 298 – 305 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
287 print(check_counter, "scenery items")
288 assert check_counter == 161, "Expected 161 scenery items"
289 assert checksum == 200095, "Error in scenery data"
290 print("Scenery checksum: " + str(checksum))
291
292 for room in range(1, 26):# Add random scenery in planet location
293     if room != 13: # Skip room 13.
294         scenery_item = random.choice([16, 28, 29, 30])
295         scenery[room] = [[scenery_item, random.randint(2, 10),
296                           random.randint(2, 10)]]
297
298 # Use loops to add fences to the planet surface rooms.
299 for room_coordinate in range(0, 13):
300     for room_number in [1, 2, 3, 4, 5]: # Add top fence
301         scenery[room_number] += [[31, 0, room_coordinate]]
302     for room_number in [1, 6, 11, 16, 21]: # Add left fence
303         scenery[room_number] += [[31, room_coordinate, 0]]
304     for room_number in [5, 10, 15, 20, 25]: # Add right fence
305         scenery[room_number] += [[31, room_coordinate, 12]]
306
307
308
309
310
311 ################
312 ## MAKE MAP  ##
313 ################
```

Line 298 contains a comment.

Line 299 begins a block of code that will add fences to the rooms. All the place surface locations are 13 tiles high and 13 tiles wide, so we can use one loop (Line 299) to add the top, left side, and right side fences (Lines 300 – 305) to the appropriate rooms. Then, we will add the fences to the scenery dictionary's list of scenery items and item y and x-coordinate locations for that particular room.

24. Press ENTER twice.

25. Type the code you see on Lines 307 – 308 of the screenshot below.

```
298 # Use loops to add fences to the planet surface rooms.
299 for room_coordinate in range(0, 13):
300     for room_number in [1, 2, 3, 4, 5]: # Add top fence
301         scenery[room_number] += [[31, 0, room_coordinate]]
302     for room_number in [1, 6, 11, 16, 21]: # Add left fence
303         scenery[room_number] += [[31, room_coordinate, 0]]
304     for room_number in [5, 10, 15, 20, 25]: # Add right fence
305         scenery[room_number] += [[31, room_coordinate, 12]]
306
307 del scenery[21][-1] # Delete last fence panel in Room 21
308 del scenery[25][-1] # Delete last fence panel in Room 25
```

We don't want the side fence panels where the outside area joins the space station wall. The bottom-left corner of the room should be wall, so there shouldn't be a fence panel here. The loops we just used added a fence panel here, though.

Lines 307 and 308 will use the del function to delete the last item of scenery added to Room 21 and 25. The "-1" code is a programming shortcut for referring to the last item in a list.

26. Ensure that Lines 309 – 310 are blank and that your "MAKE MAP" comment begins on Line 311, as shown in the screenshot below. You may need to add or delete blank lines as necessary.

```
307 del scenery[21][-1] # Delete last fence panel in Room 21
308 del scenery[25][-1] # Delete last fence panel in Room 25
309
310
311 ###############
312 ## MAKE MAP  ##
313 ###############
```

27. Scroll and click at the end of Line 379.

```
373     if current_room <= MAP_SIZE - MAP_WIDTH: # If room is not on bottom row
374         room_below = GAME_MAP[current_room+MAP_WIDTH]
375         # If room below has a top exit, add exit at bottom of this one
376         if room_below[3]:
377             room_map[room_height-1][middle_column] = floor_type
378             room_map[room_height-1][middle_column + 1] = floor_type
379             room_map[room_height-1][middle_column - 1] = floor_type
```

28. Press ENTER twice.

29. Backspace your insertion point twice to line it up with the previous "if" statements on Lines 368 and 373.

30. Type the code you see on Lines 381 – 386 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
378             room_map[room_height-1][middle_column + 1] = floor_type
379             room_map[room_height-1][middle_column - 1] = floor_type
380
381     if current_room in scenery:
382         for this_scenery in scenery[current_room]:
383             scenery_number = this_scenery[0]
384             scenery_y = this_scenery[1]
385             scenery_x = this_scenery[2]
386             room_map[scenery_y][scenery_x] = scenery_number
387
388
389 ###############
```

Line 381 will check whether there's an entry for the current room in the scenery dictionary. This check is essential because some rooms in our game might not have any scenery, and if we try to use a dictionary key that doesn't exist, Python will stop the game with an error.

Line 382 will begin a loop that will cycle through scenery items for the rom and copies them into a list called this_scenery. The first time through the loop, this_scenery contains the list for the first scenery item. The second time, it will contain the list for the second item, and so on until it reaches the final scenery item for the current room.

Each scenery item has a list containing its object number, y position, and x position. The program extracts these details from this_scenery using the index numbers and puts them into variables called scenery_number (Line 383), scenery_y (Line 384), and scenery_x (Line 385).

Now the program has all the information it needs to add the scenery item to room_map. You might remember that room_map stores the object number of the item in each position in the room. It uses the y position and x position in the room as list indexes. This program uses the scenery_y and scenery_x values as list indexes to put the item scenery_number into room_map (Line 386).

31. Press ENTER twice.

32. Type the code you see on Lines 388 – 390 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
381     if current_room in scenery:
382         for this_scenery in scenery[current_room]:
383             scenery_number = this_scenery[0]
384             scenery_y = this_scenery[1]
385             scenery_x = this_scenery[2]
386             room_map[scenery_y][scenery_x] = scenery_number
387
388             image_here = objects[scenery_number][0]
389             image_width = image_here.get_width()
390             image_width_in_tiles = int(image_width / TILE_SIZE)
391
392
393
394 ###############
395 ## EXPLORER  ##
396 ###############
```

If all our objects were one tile wide, that is all we would need to do. But some objects are wider and cover several tiles. For example, a wide object positioned in one tile might cover two or more tiles to its right, but at the moment, the program only sees it in that one tile.

We need to add something to room_map in those additional spaces so the program knows the player can't walk on those tiles. I've used the number 255 to represent a space that doesn't have an object in it but also cannot be walked on. Why the number 255? It's a large enough number to give you space to add many more objects to the game if you want to, allowing for 254 items in the objects dictionary.

First, we need to figure out how wide an image is so we know how many tiles it fills. We use scenery_number as the dictionary key to get information about the object from the objects dictionary. (Line 388). We know the objects dictionary returns a list of information, the first item of which is the image. So we use the index 0 to extra the image and put it into the variable image_here.

Then we can use the program to find out the width of animage by adding the get_width() function after the image name (Line 389). We put that number into a variable called image_width.

Because we need to know how many tiles the image covers, Line 390 divides the image width (in pixels) by the tile size (in this case, 30 pixels) and makes it an integer (whole number). We must convert the number to an integer because we're going to use it in the range() function here shortly. The range() function will only work with whole numbers.

33. Press ENTER twice.

34. Type the code you see on Lines 392 – 393 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
381     if current_room in scenery:
382         for this_scenery in scenery[current_room]:
383             scenery_number = this_scenery[0]
384             scenery_y = this_scenery[1]
385             scenery_x = this_scenery[2]
386             room_map[scenery_y][scenery_x] = scenery_number
387 
388             image_here = objects[scenery_number][0]
389             image_width = image_here.get_width()
390             image_width_in_tiles = int(image_width / TILE_SIZE)
391 
392             for tile_number in range(1, image_width_in_tiles):
393                 room_map[scenery_y][scenery_x + tile_number] = 255
394 
395 
396 
397 ###############
```

If an image is 90 pixels wide, Line 390 will divide the image pixel size by the tile size (30) to store the result, 3, in the image_width_in_tiles variable. Then, Line 392 will create a loop that counts to 2 using the range() function. Remember, we give it a range of 1 to the image_width_in_tiles variable (which is 3), so that would result in the loop looping twice, in this instance.

We add the loop numbers to the x position of the object, and those positions in the room_map are marked with the number 255 (Line 393). Large objects that cover three tiles now have 255 in the next two spaces to their right.

35. Ensure that your "EXPLORER" comment begins on Line 397, as shown in the screenshot below. You may need to add or delete blank lines.

```
379             room_map[room_height-1][middle_column - 1] = floor_type
380 
381     if current_room in scenery:
382         for this_scenery in scenery[current_room]:
383             scenery_number = this_scenery[0]
384             scenery_y = this_scenery[1]
385             scenery_x = this_scenery[2]
386             room_map[scenery_y][scenery_x] = scenery_number
387 
388             image_here = objects[scenery_number][0]
389             image_width = image_here.get_width()
390             image_width_in_tiles = int(image_width / TILE_SIZE)
391 
392             for tile_number in range(1, image_width_in_tiles):
393                 room_map[scenery_y][scenery_x + tile_number] = 255
394 
395 
396 
397 ###############
398 ## EXPLORER  ##
399 ###############
```

36. Comment two “##” symbols at the beginning of Lines 405 – 410 to make these lines inactive in your code.

```
397 ###############
398 ## EXPLORER  ##
399 ###############
400
401 def draw():
402     global room_height, room_width, room_map
403     generate_map()
404     screen.clear()
405     ## room_map[2][4] = 7
406     ## room_map[2][6] = 6
407     ## room_map[1][1] = 8
408     ## room_map[1][2] = 9
409     ## room_map[1][8] = 12
410     ## room_map[1][9] = 9
411
412     for y in range(room_height):
```

Placing a comment at the beginning of these six lines will make this part of the code inactive. This would be the same thing as deleting these lines from your code entirely. However, we may want to reference back to this code later, so I chose to comment the lines instead of just delete them.

37. Click at the end of Line 413, as shown in the screenshot below.

```
410     ## room_map[1][9] = 9
411
412     for y in range(room_height):
413         for x in range(room_width):
414             image_to_draw = objects[room_map[y][x]][0]
415             screen.blit(image_to_draw,
416                 (top_left_x + (x*30),
417                  top_left_y + (y*30) - image_to_draw.get_height()))
```

38. Press ENTER.

39. Type the code you see on Line 414 of the screenshot below.

```
412     for y in range(room_height):
413         for x in range(room_width):
414             if room_map[y][x] != 255:
415             image_to_draw = objects[room_map[y][x]][0]
416             screen.blit(image_to_draw,
417                 (top_left_x + (x*30),
418                  top_left_y + (y*30) - image_to_draw.get_height()))
```

We need to make a small change to the code that displays the room so it doesn't try to draw an image for a floor space marked with 255. That space will be covered by an image to the left of it, and we don't have an entry in the objects dictionary for 255. We don't want the program to look for an image labeled with the number 255 and then throw an error.

The "if" statement that you created on Line 414 makes sure that the instructions in the code below it draw an object only if the object number is not 255.

40. Indent the code on Lines 415 – 418 as shown in the screenshot below.

```
412     for y in range(room_height):
413         for x in range(room_width):
414             if room_map[y][x] != 255:
415                 image_to_draw = objects[room_map[y][x]][0]
416                 screen.blit(image_to_draw,
417                     (top_left_x + (x*30),
418                      top_left_y + (y*30) - image_to_draw.get_height()))
```

41. Go to File > Save to save your game file.