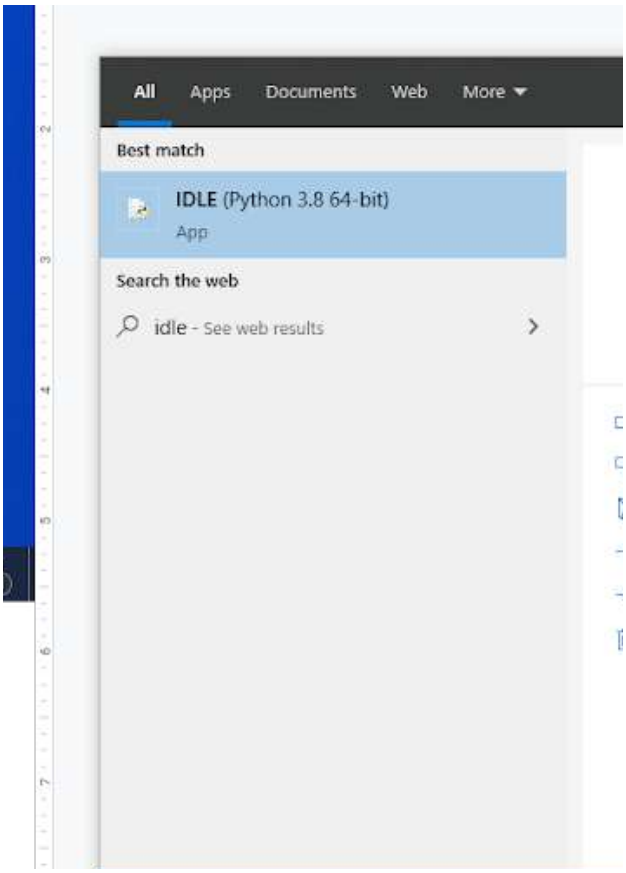
Part 6
Space Mission Directions

1. Navigate out to the Google Classroom for this class.

2. Locate the Space Mission Part 6 assignment.

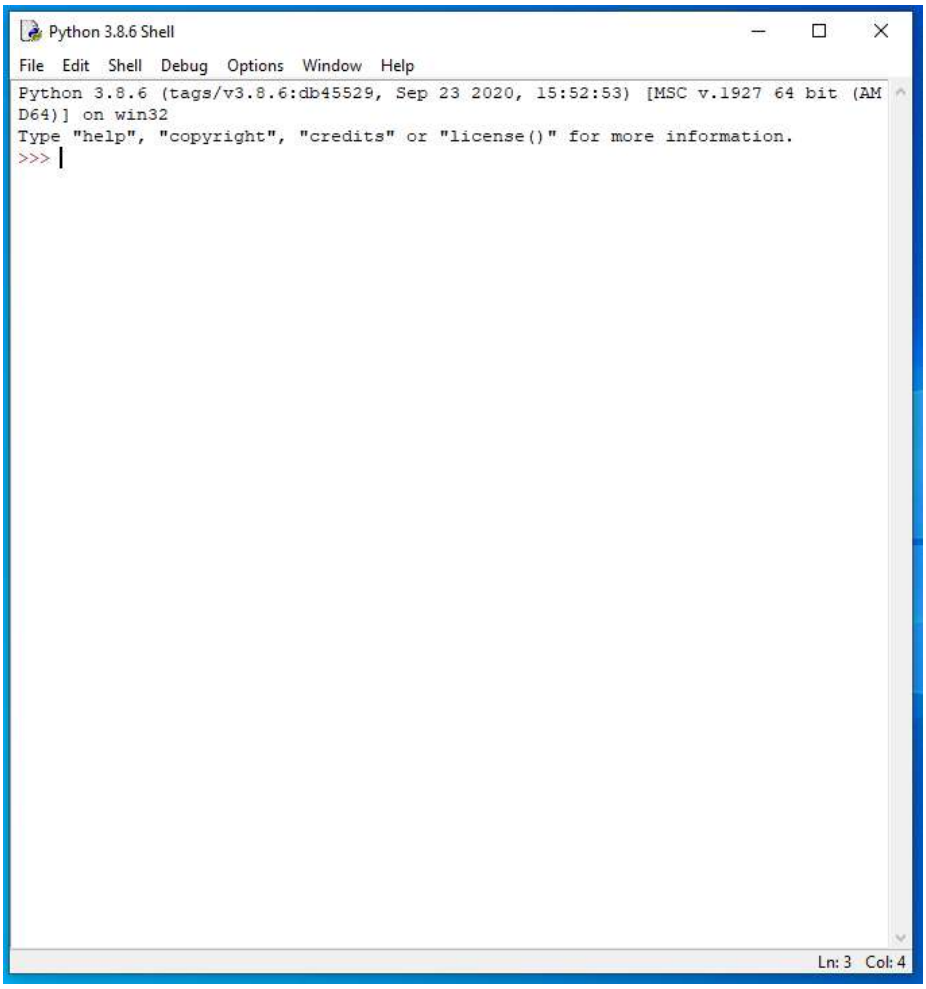
3. We are now ready to start adding code to our file. Using your Windows button menu, find and launch your IDLE program.



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.

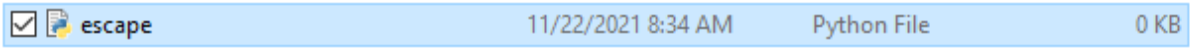
4. Your IDLE window should look something like this once it has launched.:



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.

5. 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.



6. Your escape.py file will open up.

7. Scroll and click at the end of Line 462.

```
443     if current_room in scenery:
444         for this_scenery in scenery[current_room]:
445             scenery_number = this_scenery[0]
446             scenery_y = this_scenery[1]
447             scenery_x = this_scenery[2]
448             room_map[scenery_y][scenery_x] = scenery_number
449 
450             image_here = objects[scenery_number][0]
451             image_width = image_here.get_width()
452             image_width_in_tiles = int(image_width / TILE_SIZE)
453 
454             for tile_number in range(1, image_width_in_tiles):
455                 room_map[scenery_y][scenery_x + tile_number] = 255
456 
457     center_y = int(HEIGHT / 2) # Center of game window
458     center_x = int(WIDTH / 2)
459     room_pixel_width = room_width * TILE_SIZE # Size of room in pixels
460     room_pixel_height = room_height * TILE_SIZE
461     top_left_x = center_x - 0.5 * room_pixel_width
462     top_left_y = (center_y - 0.5 * room_pixel_height) + 110
463 
464 
465 ###############
466 ## GAME LOOP ##
467 ###############
```

8. Press ENTER twice.

9. Type the code you see on Lines 464 – 475 of the screenshot below. Ensure your indentation, punctuation, and line spacing match what is shown in the screenshot.

```
457     center_y = int(HEIGHT / 2) # Center of game window
458     center_x = int(WIDTH / 2)
459     room_pixel_width = room_width * TILE_SIZE # Size of room in pixels
460     room_pixel_height = room_height * TILE_SIZE
461     top_left_x = center_x - 0.5 * room_pixel_width
462     top_left_y = (center_y - 0.5 * room_pixel_height) + 110
463
464     for prop_number, prop_info in props.items():
465         prop_room = prop_info[0]
466         prop_y = prop_info[1]
467         prop_x = prop_info[2]
468         if (prop_room == current_room and
469             room_map[prop_y][prop_x] in [0, 39, 2]):
470                 room_map[prop_y][prop_x] = prop_number
471                 image_here = objects[prop_number][0]
472                 image_width = image_here.get_width()
473                 image_width_in_tiles = int(image_width / TILE_SIZE)
474                 for tile_number in range(1, image_width_in_tiles):
475                     room_map[prop_y][prop_x + tile_number] = 255
476
477
478 ###############
479 ## GAME LOOP ##
480 ###############
```

In the new code, we start by setting up a loop to go through the items in the props dictionary, which we haven't created yet. For each item, the dictionary key goes into the variable prop_number, and the list with the position information goes into the list prop_info (Line 464).

To make the program easier to read, I've set up some variables to store the information from the prop_info list. The program extracts the information for the room number (and puts it into prop_room variable on Line 465) and the y and x positions (which go into the prop_y and prop_x variables on Lines 466 and 467).

We add a check to see whether the prop_room matches the room the player is in and whether the prop is sitting on the floor (Lines 468 and 469). The floor check puts the three different floor types in a list (0 for inside, 2 for soil, and 39 for the pressure pad in room 26). The program checks the prop's position to see what's in that location in the room map. If it's one of these floor types, it means the object is sitting on the floor in full view. If not, the prop is hidden inside an item of scenery and shouldn't be visible yet. For example, if a cabinet is in the prop's location instead of the floor, the prop won't be shown on screen. The player can still find the prop by examining the vabinet at that location, though.

If the prop is in the room and on the floor, the room map is updated with the prop number (Line 470).

Lines 471 – 473 calculate the image and the size of the prop being used.

Some props, like doors, are wider than one tile. We add the number 255 to any tiles that the prop covers other than the first one (Line 474). This is similar to the code we used to mark wide scenery earlier in the generate_map() function.

10. Uncomment all of the start_room() functions by erasing the "#" at the beginning of each comment (Lines 543, 553, 563, and 573). The uncommented code is shown in the screenshot below.

```
535 # check for exiting the room
536     if player_x == room_width: # through door on RIGHT
537         #clock.unschedule(hazard_move)
538         current_room += 1
539         generate_map()
540         player_x = 0 # enter at left
541         player_y = int(room_height / 2) # enter at door
542         player_frame = 0
543         start_room()
544         return
545 
546     if player_x == -1: # through door on LEFT
547         #clock.unschedule(hazard_move)
548         current_room -= 1
549         generate_map()
550         player_x = room_width - 1  # enter at right
551         player_y = int(room_height / 2) # enter at door
552         player_frame = 0
553         start_room()
554         return
555 
556     if player_y == room_height: # through door at BOTTOM
557         #clock.unschedule(hazard_move)
558         current_room += MAP_WIDTH
559         generate_map()
560         player_y = 0 # enter at top
561         player_x = int(room_width / 2) # enter at door
562         player_frame = 0
563         start_room()
564         return
565 
566    if player_y == -1: # through door at TOP
567         #clock.unschedule(hazard_move)
568         current_room -= MAP_WIDTH
569         generate_map()
570         player_y = room_height - 1 # enter at bottom
571         player_x = int(room_width / 2) # enter at door
572         player_frame = 0
573         start_room()
574         return
```

11. Click at the end of Line 574.

```
535 # check for exiting the room
536     if player_x == room_width: # through door on RIGHT
537         #clock.unschedule(hazard_move)
538         current_room += 1
539         generate_map()
540         player_x = 0 # enter at left
541         player_y = int(room_height / 2) # enter at door
542         player_frame = 0
543         start_room()
544         return
545 
546     if player_x == -1: # through door on LEFT
547         #clock.unschedule(hazard_move)
548         current_room -= 1
549         generate_map()
550         player_x = room_width - 1  # enter at right
551         player_y = int(room_height / 2) # enter at door
552         player_frame = 0
553         start_room()
554         return
555 
556     if player_y == room_height: # through door at BOTTOM
557         #clock.unschedule(hazard_move)
558         current_room += MAP_WIDTH
559         generate_map()
560         player_y = 0 # enter at top
561         player_x = int(room_width / 2) # enter at door
562         player_frame = 0
563         start_room()
564         return
565 
566    if player_y == -1: # through door at TOP
567         #clock.unschedule(hazard_move)
568         current_room -= MAP_WIDTH
569         generate_map()
570         player_y = room_height - 1 # enter at bottom
571         player_x = int(room_width / 2) # enter at door
572         player_frame = 0
573         start_room()
574         return
```

12. Press ENTER twice.

13. Type the code you see on Lines 576 – 590 of the screenshot below. Ensure your indentation, punctuation, and line spacing match what is being shown in the screenshot.

```
566     if player_y == -1: # through door at TOP
567         #clock.unschedule(hazard_move)
568         current_room -= MAP_WIDTH
569         generate_map()
570         player_y = room_height - 1 # enter at bottom
571         player_x = int(room_width / 2) # enter at door
572         player_frame = 0
573         start_room()
574         return
575
576      if keyboard.g:
577         pick_up_object()
578
579      if keyboard.tab and len(in_my_pockets) > 0:
580         selected_item += 1
581         if selected_item > len(in_my_pockets) - 1:
582             selected_item = 0
583         item_carrying = in_my_pockets[selected_item]
584         display_inventory()
585
586      if keyboard.d and item_carrying:
587         drop_object(old_player_y, old_player_x)
588
589      if keyboard.space:
590         examine_object()
591
592
593       # If the player is standing somewhere they shouldn't, move them back.
594      if room_map[player_y][player_x] not in items_player_may_stand_on: #\
```

Line 576 checks to see if the g key on the keyboard has been pressed. If it has, Line 577 executes the pick_up_object function.

Line 579 checks to see if the tab key has been pressed on the keyboard and the length of the in_my_pockets list variable is greater than 0. If this is true, the selected_item variable (which we haven't created yet) will be increased by one (Line 580). Line 581 checks to see if the value of the selected_item variable is greater than the length of the in_my_pockets list. If it is, than the select_item variable will be reset back to 0 and the player will view the first item in the in_my_pockets list again (the item with the index value of 0) (Line 582).

Line 583 will create the item_carrying variable and set its value to be equal to whatever item is currently selected (the selected_item) from the in_my_pockets list.

Line 584 will run the display_inventory function, which we haven't created yet.

Line 586 will check to see if the d key on the keyboard has been pressed and the item_carrying variable is true. If both of these conditions are true, then the drop_object method will run (Line 587) using the old_player_y and old_player_x coordinates that the program saved. We haven't written the drop_object method yet.

Line 589 will check to see if the space key has been pressed on the keyboard.

Line 590 will run the examine_object function (which we haven't written yet) if the space key has been pressed on the keyboard.

14. Ensure the "#If the player is standing…" comment is on Line 593.

15. Scroll and click at the end of Line 719.

```
712 def show_text(text_to_show, line_number):
713     if game_over:
714         return
715     text_lines = [15, 50]
716     box = Rect((0, text_lines[line_number]), (800, 35))
717     screen.draw.filled_rect(box, BLACK)
718     screen.draw.text(text_to_show,
719                      (20, text_lines[line_number]), color=GREEN)
720
721
722
723 ################
724 ##   START    ##
725 ################
```

16. Press ENTER three times.

17. Type the code you see on Lines 723 – 742 of the screenshot below. Ensure your indentation, punctuation, and line spacing matches what is shown in the screenshot.

```
712 def show_text(text_to_show, line_number):
713     if game_over:
714         return
715     text_lines = [15, 50]
716     box = Rect((0, text_lines[line_number]), (800, 35))
717     screen.draw.filled_rect(box, BLACK)
718     screen.draw.text(text_to_show,
719                      (20, text_lines[line_number]), color=GREEN)
720
721
722
723 ###############
724 ##   PROPS   ##
725 ###############
726
727 # Props are objects that may move between rooms, appear or disappear.
728 # All props must be set up here. Props not yet in the game go into room 0.
729 # object number : [room, y, x]
730 props = {
731     20: [31, 0, 4], 21: [26, 0, 1], 22: [41, 0, 2], 23: [39, 0, 5],
732     24: [45, 0, 2],
733     25: [32, 0, 2], 26: [27, 12, 5], # two sides of same door
734     40: [0, 8, 6], 53: [45, 1, 5], 54: [0, 0, 0], 55: [0, 0, 0],
735     56: [0, 0, 0], 57: [35, 4, 6], 58: [0, 0, 0], 59: [31, 1, 7],
736     60: [0, 0, 0], 61: [36, 1, 1], 62: [36, 1, 6], 63: [0, 0, 0],
737     64: [27, 8, 3], 65: [50, 1, 7], 66: [39, 5, 6], 67: [46, 1, 1],
738     68: [0, 0, 0], 69: [30, 3, 3], 70: [47, 1, 3],
739     71: [0, LANDER_Y, LANDER_X], 72: [0, 0, 0], 73: [27, 4, 6],
740     74: [28, 1, 11], 75: [0, 0, 0], 76: [41, 3, 5], 77: [0, 0, 0],
741     78: [35, 9, 11], 79: [26, 3, 2], 80: [41, 7, 5], 81: [29, 1, 1]
742     }
743
744
745 ###############
746 ##   START   ##
747 ###############
```

Lines 723 – 725 create a new section of code called PROPS.

Lines 727 – 729 contain comments describing the props dictionary that you are about to create.

Lines 730 – 742 contains a listing of props that the player can manipulate in the game. They are listed by prop number. Their coordinates include the room number they start out in and their y and x positions. The props dictionary lists the position locations for all the props, starting with some doors (20 to 24) and including a rescue ship (40) and the carryable items starting at 53.

There is just one oddity to draw your attention to. We count doors as props rather than scenery, because they're not always there: when they're open, they're removed from the room. Most doors stay open when they're opened until the game ends. However, the door that connects rooms 27 and 32 can also shut, meaning players can see it from both sides. As a result, we need

two props to represent this door, showing it in the top of room 27 and the bottom of room 32. These two doors are object numbers 25 and 26.

Prop 71 is the Poodle lander, which crash landed on the planet surface before the game began. We use the LANDER_Y and LANDER_X variables from the VARIABLES section of code to position the lander, because its location will change with each new game. The Poodle landed with such force that it might have become covered with Martian soil. It lives in room 0 until the player can dig it up.

18. Press ENTER twice.

19. Type the code you see on Lines 744 – 753 of the screenshot below. Ensure your indentation, punctuation, and line spacing matches what is shown in the screenshot.

```
###############
##   PROPS   ##
###############

# Props are objects that may move between rooms, appear or disappear.
# All props must be set up here. Props not yet in the game go into room 0.
# object number : [room, y, x]
props = {
    20: [31, 0, 4], 21: [26, 0, 1], 22: [41, 0, 2], 23: [39, 0, 5],
    24: [45, 0, 2],
    25: [32, 0, 2], 26: [27, 12, 5], # two sides of same door
    40: [0, 8, 6], 53: [45, 1, 5], 54: [0, 0, 0], 55: [0, 0, 0],
    56: [0, 0, 0], 57: [35, 4, 6], 58: [0, 0, 0], 59: [31, 1, 7],
    60: [0, 0, 0], 61: [36, 1, 1], 62: [36, 1, 6], 63: [0, 0, 0],
    64: [27, 8, 3], 65: [50, 1, 7], 66: [39, 5, 6], 67: [46, 1, 1],
    68: [0, 0, 0], 69: [30, 3, 3], 70: [47, 1, 3],
    71: [0, LANDER_Y, LANDER_X], 72: [0, 0, 0], 73: [27, 4, 6],
    74: [28, 1, 11], 75: [0, 0, 0], 76: [41, 3, 5], 77: [0, 0, 0],
    78: [35, 9, 11], 79: [26, 3, 2], 80: [41, 7, 5], 81: [29, 1, 1]
    }

checksum = 0
for key, prop in props.items():
    if key != 71: # 71 is skipped because it's different each game.
        checksum += (prop[0] * key
                     + prop[1] * (key + 1)
                     + prop[2] * (key + 2))
print(len(props), "props")
assert len(props) == 37, "Expected 37 prop items"
print("Prop checksum:", checksum)
assert checksum == 61414, "Error in props data"


###############
##   START   ##
###############
```

As with the scenery information, I've used a checksum on Lines 744 - 753 to help you spot whether you made an error entering the data. It might not be possible to play the game all the way to the end if a mistake is made here. The only prop missing from the checksum calculations is number 71, because its position uses different random numbers in each game.

20. Press ENTER three times.

21. Type the code you see on Lines 756 – 758 of the screenshot below.

```
744 checksum = 0
745 for key, prop in props.items():
746     if key != 71: # 71 is skipped because it's different each game.
747         checksum += (prop[0] * key
748                      + prop[1] * (key + 1)
749                      + prop[2] * (key + 2))
750 print(len(props), "props")
751 assert len(props) == 37, "Expected 37 prop items"
752 print("Prop checksum:", checksum)
753 assert checksum == 61414, "Error in props data"
754
755
756 in_my_pockets = [55]
757 selected_item = 0 # the first item
758 item_carrying = in_my_pockets[selected_item]
759
760
761 ###############
762 ##   START   ##
763 ###############
```

Line 756 creates a list variable called in_my_pockets. This variable stores all the items the player has picked up, also known as their inventory.

One of these items is always selected, so the player is ready to do something with it. The selected_item variable on Line 757 stores the currently selected item's index number in the in_my_pockets list.

The item_carrying variable (Line 758) stores the object number of the item the player has selected. You can think of the item_carrying variable as being the number of the object in their hands.

22. Press ENTER three times.

23. Type the code you see on Lines 761 – 774 of the screenshot below. Ensure your indentation, line spacing, and punctuation matches what is shown in the screenshot.

```
756 in_my_pockets = [55]
757 selected_item = 0 # the first item
758 item_carrying = in_my_pockets[selected_item]
759
760
761 ########################
762 ## PROP INTERACTIONS ##
763 ########################
764
765 def find_object_start_x():
766     checker_x = player_x
767     while room_map[player_y][checker_x] == 255:
768         checker_x -= 1
769     return checker_x
770
771 def get_item_under_player():
772     item_x = find_object_start_x()
773     item_player_is_on = room_map[player_y][item_x]
774     return item_player_is_on
775
776
777 ###############
778 ##   START   ##
779 ###############
```

Lines 761 – 763 create another section of the code called PROP INTERACTIONS.

Line 765 creates a mew method called find_object_start_x.

This function finds the start position of whatever object is at the player's position, going left to find the real object number if the location contains 255.

To do this, the function sets the variable checker_x to be the same as the player's x position (Lome 766). We use a loop that keeps going for as long as the room map contains 255 at the x position of checker_x and at the player's y position (Line 767). Inside that loop is a single instruction to reduce checker_x by 1, moving 1 tile to the left (Line 768). When the loop finishes, checker_x contains the left position where the object begins. That number is then sent back to the instruction that started the function (Line 769).

Line 771 creates another new function called get_item_under_player.

This function works out which object is at the player's position. Line 772 uses the find_object_start_x to find out where the object starts and stores the x position in the variable item_x. Then it looks at the room map data for that position to see what object is there (Line 773) and sends that number back to the instruction that started the function (Line 774).

24. Press ENTER twice.

25. Type the code you see on Lines 776 – 788 of the screenshot below. Ensure your indentation, line spacing, and punctuation match what is being shown in the screenshot.

```
771 def get_item_under_player():
772     item_x = find_object_start_x()
773     item_player_is_on = room_map[player_y][item_x]
774     return item_player_is_on
775
776 def pick_up_object():
777     global room_map
778     # Get object number at player's location.
779     item_player_is_on = get_item_under_player()
780     if item_player_is_on in items_player_may_carry:
781         # Clear the floor space.
782         room_map[player_y][player_x] = get_floor_type()
783         add_object(item_player_is_on)
784         show_text("Now carrying " + objects[item_player_is_on][3], 0)
785         sounds.pickup.play()
786         time.sleep(0.5)
787     else:
788         show_text("You can't carry that!", 0)
789
790
791
792 ###############
793 ##   START   ##
794 ###############
```

Line 776 creates another function called pick_up_object.

Line 777 establishes the room_map variable as a global variable so that the function can access and modify it.

Line 778 contains a comment.

The function pick_up_object will start when the player presses the get key (G) to pick up an item. It begins by running the get_item_under_player function to put the object number for the item at the player's position into the variable item_player_is_on (Line 779).

If the item is carryable, the rest of the function picks it up by clearing the floor space, figuring out what the floor type should be, alerting the player that they are now carrying that particular object, and playing a sound. It also runs the add_object function for the object in the item_player_is_on variable, which we will create in the next few steps. Otherwise, it will display a message telling the player that they cannot carry that object (Line 788).

26. Press ENTER twice.

27. Type the code you see on Lines 790 – 797 of the screenshot below. Ensure your indentation and punctuation match what is shown in the screenshot.

```
776 def pick_up_object():
777     global room_map
778     # Get object number at player's location.
779     item_player_is_on = get_item_under_player()
780     if item_player_is_on in items_player_may_carry:
781         # Clear the floor space.
782         room_map[player_y][player_x] = get_floor_type()
783         add_object(item_player_is_on)
784         show_text("Now carrying " + objects[item_player_is_on][3], 0)
785         sounds.pickup.play()
786         time.sleep(0.5)
787     else:
788         show_text("You can't carry that!", 0)
789 
790 def add_object(item): # Adds item to inventory.
791     global selected_item, item_carrying
792     in_my_pockets.append(item)
793     item_carrying = item
794     # Minus one because indexes start at 0.
795     selected_item = len(in_my_pockets) - 1
796     display_inventory()
797     props[item][0] = 0 # Carried objects go into room 0 (off the map).
798 
799 
800 ###############
801 ##   START   ##
802 ###############
```

Line 790 creates another function called add_object. This function would require the item to be entered whenever it is called.

Line 791 establishes the select_item and item_carrying variables as global variables.

Line 792 appends whatever item the player is currently holding to the in_my_pockets list, effectively adding that particular item to the player's inventory.

Line 793 will set the value of the item_carrying variable to whatever item was just added.

Line 794 contains a comment.

Line 795 sets the index value of the selected_item variable to the length of the list minus 1, which would be the last item in the list, or whatever item was just added.

Line 796 will run the display_inventory function, which we haven't written yet.

Line 797 will remove that particular item's image from the current room they are assigned and move that item to room 0 so that the item is no longer visible on the map.

28. Press ENTER twice.

29. Type the code you see on Lines 799 – 820 of the screenshot below. Ensure your indentation, punctuation, and line spacing match what is shown in the screenshot.

```
797     props[item][0] = 0 # Carried objects go into room 0 (off the map).
798
799 def display_inventory():
800     box = Rect((0, 45), (800, 105))
801     screen.draw.filled_rect(box, BLACK)
802
803     if len(in_my_pockets) == 0:
804         return
805
806     start_display = (selected_item // 16) * 16
807     list_to_show = in_my_pockets[start_display : start_display + 16]
808     selected_marker = selected_item % 16
809
810     for item_counter in range(len(list_to_show)):
811         item_number = list_to_show[item_counter]
812         image = objects[item_number][0]
813         screen.blit(image, (25 + (46 * item_counter), 90))
814
815     box_left = (selected_marker * 46) - 3
816     box = Rect((22 + box_left, 85), (40, 40))
817     screen.draw.rect(box, WHITE)
818     item_highlighted = in_my_pockets[selected_item]
819     description = objects[item_highlighted][2]
820     screen.draw.text(description, (20, 130), color="white")
821
822
823 ###############
824 ##   START   ##
825 ###############
```

Line 799 creates a new function called display_inventory.

The new display_inventory function starts by drawing a black box over the inventory area to clear it. If the player isn't carrying anything, the function returns without taking any further action because there are no items to display. (Lines 800 – 804)

There is only room to show 16 items on the screen, but the player could carry many more items than that. If the in_my_pockets list is too long to fit on the screen, the program shows it 16 items at a time. The player can select any of the items shown on the screen by pressing the TAB key to move through them, from left to right. If the last item displayed is selected and they press TAB, the next chunk of the list is shown. If the player presses TAB on the final item in the list, the start of the list appears again.

We store the part of the in_my_pockets list currently displayed on the screen in another list called list_to_show and use a loop to display it. The loop puts numbers into a variable called item_counter, which is used to extract the right image to draw each time, and also work out where to draw it.

The clever bit is working out which items should go into list_to_show. In the start_display variable, we store the index number for the first item in in_my_pockets that the program should draw (Line 806). The // operator divides the select item number by 16, rounding down. The result is then multiplied by 16 to get the index number for the first item in the batch. For example, if the selected item is number 9, you'll divide 9 by 16, round down, and then multiple by 16, getting a result of 0. That's the start of the list, which makes sense, because we know there's room for 16 items onscreen and that 9 is less than 16. If you wanted to see the group of items that includes item 22, you'd divide 22 by 16, round down, and multiply by 16. That's the start of the next batch, because the first batch has index numbers that range from 0 to 15.

We create the list_to_show list (Line 807) using a technique called list slicing, which is simply using just a part of a list. When you give Python two list indexes with a colon between them, the program will cut out that part of the list. The section we're using starts at the start_display index and finishes 15 items later. A list slice leaves out the last item, so we use start_display + 16 as the end point.

We also need another calculation to work out which item to highlight as the selected item from the new list. The item will have an index between 0 and 15, and we'll store it in selected_marker variable (Lines 808). We calculate it as the remainder after we divide the selected item number by 16. For example, if the selected item is number 18, it will be at index number 2 when the second group of items is displayed. Python has the modulo operator %, which you can use to get the remainder after a division.

As mentioned earlier, Lines 810 – 813 create a "for" loop to display the items in the list_to_show list variable on the screen. This loop will loop through each item in the list_to_show list to extract the proper image (Line 812) and to display the proper image at the appropriate location (Line 813).

To highlight the selected item on the screen, we draw a box around it using a Rect positioned at its left edge (Lines 815 – 816). Unlike the filled rectangle you've seen, this instruction draws a hollow box with a white edge (Line 817).

Lines 818 – 820 extra information from the selected item to find its description in the objects dictionary (the item with the index value of 2 from the objects dictionary). The description for the selected item is displayed underneath the inventory, so players can TAB through their items to read their descriptions again.

30. Press ENTER twice.

31. Type the code you see on Lines 822 – 835 of the screenshot below. Ensure your indentation and punctuation match what is shown in the screenshot.

```
815     box_left = (selected_marker * 46) - 3
816     box = Rect((22 + box_left, 85), (40, 40))
817     screen.draw.rect(box, WHITE)
818     item_highlighted = in_my_pockets[selected_item]
819     description = objects[item_highlighted][2]
820     screen.draw.text(description, (20, 130), color="white")
821
822 def drop_object(old_y, old_x):
823     global room_map, props
824     if room_map[old_y][old_x] in [0, 2, 39]: # places you can drop things
825         props[item_carrying][0] = current_room
826         props[item_carrying][1] = old_y
827         props[item_carrying][2] = old_x
828         room_map[old_y][old_x] = item_carrying
829         show_text("You have dropped " + objects[item_carrying][3], 0)
830         sounds.drop.play()
831         remove_object(item_carrying)
832         time.sleep(0.5)
833     else: # This only happens if there is already a prop here
834         show_text("You can't drop that there.", 0)
835         time.sleep(0.5)
836
837
838 ###############
839 ##   START   ##
840 ###############
```

Line 822 creates a new function called drop_object. This function will require the old_y and old_x variables to be entered when it is called.

Line 823 establishes the room_map and props variable as global variables.

The drop_object function needs two pieces of information: the player's old y and x positions. If the player moved through the wall function, this will be the position they were in before they tried to move. If not, these numbers will be the same position as where they currently are. We know this is a sensible place to drop an item that won't put the object inside a wall. The player's old position goes into the variables old_y and old_x within this function.

The program checks whether the room map at the player's old position is a type of floor (Line 824). If so, it's okay to drop a prop here, so the drop instructions are used. If not, the player sees a message telling them they can't drop objects there (Line 833). This will happen, for example, if there is already a prop in that position.

If the player can drop the item, we need to update the props dictionary. The variable item_carrying contains the numbers of the object the player is carrying. Its entry in the props dictionary is a list. The first list item (index 0) is the room the prop is in. Line 825 will update this to the current room where the prop was dropped. The second item (index 1) is its y position,

and the third item is its x position (index 2). Lines 826 – 827 will update the y and x position of the prop to be the player's old position.

The room map for the current room also needs to be updated, so the room contains the dropped item (Line 828). The game will show a message and play a sound to tell the player that they've successfully dropped something and then the item is removed from the inventory using the remove_object function (which we haven't written yet) (Lines 829 – 831). The game will also pause for half a second.

32. Press ENTER twice.

33. Type the code you see on Lines 837 – 847 of the screenshot below. Ensure your indentation and punctuation match what is shown in the screenshot.

```
822 def drop_object(old_y, old_x):
823     global room_map, props
824     if room_map[old_y][old_x] in [0, 2, 39]: # places you can drop things
825         props[item_carrying][0] = current_room
826         props[item_carrying][1] = old_y
827         props[item_carrying][2] = old_x
828         room_map[old_y][old_x] = item_carrying
829         show_text("You have dropped " + objects[item_carrying][3], 0)
830         sounds.drop.play()
831         remove_object(item_carrying)
832         time.sleep(0.5)
833     else: # This only happens if there is already a prop here
834         show_text("You can't drop that there.", 0)
835         time.sleep(0.5)
836
837 def remove_object(item): # Takes item out of inventory
838     global selected_item, in_my_pockets, item_carrying
839     in_my_pockets.remove(item)
840     selected_item = selected_item - 1
841     if selected_item < 0:
842         selected_item = 0
843     if len(in_my_pockets) == 0: # If they're not carrying anything
844         item_carrying = False # Set item_carrying to False
845     else: # Otherwise set it to the new selected item
846         item_carrying = in_my_pockets[selected_item]
847     display_inventory()
848
849
850 ###############
851 ##   START   ##
852 ###############
```

Line 837 creates a new function called remove_object. This function will require the item to be input whenever it is called.

Line 838 makes the selected item, in_my_pockets variable list, and item_carrying variables into global variables.

Line 839 uses the remove function to remove that particular item from the in_my_pockets list. If the player has dropped the item, it needs to be removed from their inventory.

Line 840 will update the selected item variable by decreasing it by 1.

Line 841 will check to see if the selected_item variable's value is less than 0. If so, Line 842 will set the selected_item variable to 0.

Line 843 will check to see if the length of the in_my_pockets list is equal to 0. If this is true, Line 844 will set the value of the item_carrying variable to False because the player doesn't have anything in their inventory.

If neither of the above conditions are true (the selected_item variable's value is more than 0 and the in_my_pockets list is longer than 0), the item_carrying variable will be set to be equal to the selected_item from the in_my_pockets list.

Line 847 will display the updated inventory on the screen using the display_inventory function.

34. Press ENTER twice.

35. Type the code you see on Lines 849 – 866 of the screenshot below. Ensure your indentation and punctuation match what is shown in the screenshot.

```
837 def remove_object(item): # Takes item out of inventory
838     global selected_item, in_my_pockets, item_carrying
839     in_my_pockets.remove(item)
840     selected_item = selected_item - 1
841     if selected_item < 0:
842         selected_item = 0
843     if len(in_my_pockets) == 0: # If they're not carrying anything
844         item_carrying = False # Set item_carrying to False
845     else: # Otherwise set it to the new selected item
846         item_carrying = in_my_pockets[selected_item]
847     display_inventory()
848
849 def examine_object():
850     item_player_is_on = get_item_under_player()
851     left_tile_of_item = find_object_start_x()
852     if item_player_is_on in [0, 2]: # don't describe the floor
853         return
854     description = "You see: " + objects[item_player_is_on][2]
855     for prop_number, details in props.items():
856         # props = object number: [room number, y, x]
857         if details[0] == current_room: # if prop is in the room
858             # If prop is hidden (= at player's location but not on map)
859             if (details[1] == player_y
860                 and details[2] == left_tile_of_item
861                 and room_map[details[1]][details[2]] != prop_number):
862                 add_object(prop_number)
863                 description = "You found " + objects[prop_number][3]
864                 sounds.combine.play()
865     show_text(description, 0)
866     time.sleep(0.5)
867
868
869 ###############
870 ##   START   ##
871 ###############
```

Line 849 creates another function called examine_opbject.

We start by getting the number of the object the player wants to examine and storing it in item_player_is_on variable (Line 850). At this point in the game_loop function, the player's position will be on or possibly inside the item they want to examine, if it's a piece of scenery. We use the find_object_start_x function to find the starting x position of the object and put the starting x position of the item into the variable left_tile_of_item (Line 851). If there isn't an object to examine at the player's location, the function finishes without taking any further action (Lines 852 – 853). Ignoring an empty space feels more natural than describing the floor, especially if you make a mistake with the controls. If there is an item at the player's location, the description of the object goes into the description variable, taken from the long description from the objects dictionary (Line 854).

The program then checks whether there's an item hidden inside the item the player is examining. We use a loop to go through all the items in the props dictionary (Line 855). If an item is in the current room at the player's position, but the room map at that position doesn't contain the prop number, it means the item is hidden (Lines 857 – 861). We therefore add the hidden object to the player's inventory (Line 862) and give the player a message that tells them they found something (Line 863). This message uses the object's short description to tell them what they've found. We also play a sound (Line 864).

At the end of the function, the description is shown (Line 865) and we've put a short pause here (Line 866) to stop it being immediately overwritten if the player holds the key down.

36. Ensure the "START" comment runs on Lines 869 – 871 of the code.

```
837 def remove_object(item): # Takes item out of inventory
838     global selected_item, in_my_pockets, item_carrying
839     in_my_pockets.remove(item)
840     selected_item = selected_item - 1
841     if selected_item < 0:
842         selected_item = 0
843     if len(in_my_pockets) == 0: # If they're not carrying anything
844         item_carrying = False # Set item_carrying to False
845     else: # Otherwise set it to the new selected item
846         item_carrying = in_my_pockets[selected_item]
847     display_inventory()
848 
849 def examine_object():
850     item_player_is_on = get_item_under_player()
851     left_tile_of_item = find_object_start_x()
852     if item_player_is_on in [0, 2]: # don't describe the floor
853         return
854     description = "You see: " + objects[item_player_is_on][2]
855     for prop_number, details in props.items():
856         # props = object number: [room number, y, x]
857         if details[0] == current_room: # if prop is in the room
858             # If prop is hidden (= at player's location but not on map)
859             if (details[1] == player_y
860                 and details[2] == left_tile_of_item
861                 and room_map[details[1]][details[2]] != prop_number):
862                 add_object(prop_number)
863                 description = "You found " + objects[prop_number][3]
864                 sounds.combine.play()
865     show_text(description, 0)
866     time.sleep(0.5)
867 
868 
869 ################
870 ##   START    ##
871 ################
```

37. Click at the end of Line 875.

```
869 ################
870 ##    START    ##
871 ################
872
873 generate_map()
874 clock.schedule_interval(game_loop, 0.03)
875 clock.schedule_interval(adjust_wall_transparency, 0.05)
```

38. Press ENTER.

39. Type the code you see on Line 876 of the screenshot below.

```
869 ################
870 ##    START    ##
871 ################
872
873 generate_map()
874 clock.schedule_interval(game_loop, 0.03)
875 clock.schedule_interval(adjust_wall_transparency, 0.05)
876 clock.schedule_unique(display_inventory, 1)
```

    Line 876 uses the clock.schedule_unique function to run the display_inventory method just once, after a delay of 1 second.

40. Go to File > Save to save your code.

Final Code:

```
# Escape

import time, random, math

###############
## VARIABLES ##
###############

WIDTH = 800 #window size
HEIGHT = 800

#PLAYER variables
PLAYER_NAME = "Alice"
FRIEND1_NAME = "Jack"
FRIEND2_NAME = "Matthew"
current_room = 31 # start room = 31

top_left_x = 100
top_left_y = 150

DEMO_OBJECTS = [images.floor, images.pillar, images.soil]

LANDER_SECTOR = random.randint(1, 26)
LANDER_X = random.randint(2, 11)
LANDER_Y = random.randint(2, 11)

TILE_SIZE = 80

player_y, player_x = 2, 5
game_over = False

PLAYER = {
    "left": [images.spacesuit_left, images.spacesuit_left_1,
             images.spacesuit_left_2, images.spacesuit_left_3,
             images.spacesuit_left_4
             ],
    "right": [images.spacesuit_right, images.spacesuit_right_1,
              images.spacesuit_right_2, images.spacesuit_right_3,
              images.spacesuit_right_4
              ],
    "up": [images.spacesuit_back, images.spacesuit_back_1,
           images.spacesuit_back_2, images.spacesuit_back_3,
           images.spacesuit_back_4
           ],
    "down": [images.spacesuit_front, images.spacesuit_front_1,
             images.spacesuit_front_2, images.spacesuit_front_3,
             images.spacesuit_front_4
             ]
    }

player_direction = "down"
player_frame = 0
player_image = PLAYER[player_direction][player_frame]
player_offset_x, player_offset_y = 0, 0

PLAYER_SHADOW = {
    "left": [images.spacesuit_left_shadow, images.spacesuit_left_1_shadow,
             images.spacesuit_left_2_shadow, images.spacesuit_left_3_shadow,
             images.spacesuit_left_3_shadow
             ],
    "right": [images.spacesuit_right_shadow, images.spacesuit_right_1_shadow,
              images.spacesuit_right_2_shadow,
              images.spacesuit_right_3_shadow, images.spacesuit_right_3_shadow
              ],
    "up": [images.spacesuit_back_shadow, images.spacesuit_back_1_shadow,
           images.spacesuit_back_2_shadow, images.spacesuit_back_3_shadow,
           images.spacesuit_back_3_shadow
           ],
    "down": [images.spacesuit_front_shadow, images.spacesuit_front_1_shadow,
             images.spacesuit_front_2_shadow, images.spacesuit_front_3_shadow,
             images.spacesuit_front_3_shadow
             ]
    }

player_image_shadow = PLAYER_SHADOW["down"][0]

PILLARS = {
    images.pillar, images.pillar_95, images.pillar_90,
    images.pillar_60, images.pillar_50
    }

wall_transparency_frame = 0

BLACK = (0, 0, 0)
BLUE = (0, 155, 255)
YELLOW = (255, 255, 0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
RED = (128, 0, 0)


###############
##    MAP    ##
###############

MAP_WIDTH = 5
MAP_HEIGHT = 10
MAP_SIZE = MAP_WIDTH * MAP_HEIGHT

GAME_MAP = [ ["Room 0 - where unused objects are kept", 0, 0, False, False] ]

outdoor_rooms = range(1, 26)
for planetsectors in range(1, 26): #rooms 1 to 25 are generated here
    GAME_MAP.append( ["The dusty planet surface", 13, 13, True, True] )

```

```
GAME_MAP  = [
        #["Room name", height, width, Top exit?, Right exit?]
        ["The airlock", 13, 5, True, False], # room 26
        ["The engineering lab", 13, 13, False, False], # room 27
        ["Doodle Mission Control", 9, 13, False, True], # room 28
        ["The viewing gallery", 9, 15, False, False], # room 29
        ["The crew's bathroom", 5, 5, False, False], # room 30
        ["The airlock entry bay", 7, 11, True, True], # room 31
        ["Left elbow room", 9, 7, True, False], # room 32
        ["Right elbow room", 7, 13, True, True], # room 33
        ["The science lab", 13, 13, False, True], # room 34
        ["The greenhouse", 13, 13, True, False], # room 35
        [PLAYER_NAME + "'s sleeping quarters", 9, 11, False, False], # room 36
        ["West corridor", 15, 5, True, True], # room 37
        ["The briefing room", 7, 13, False, True], # room 38
        ["The crew's community room", 11, 13, True, False], # room 39
        ["Main Mission Control", 25, 16, False, False], # room 40
        ["The sick bay", 12, 7, True, False], # room 41
        ["West corridor", 9, 7, True, False], # room 42
        ["Utilities control room", 9, 9, False, True], # room 43
        ["Systems engineering bay", 9, 11, False, False], # room 44
        ["Security portal to Mission Control", 7, 7, True, False], # room 45
        [FRIEND1_NAME + "'s sleeping quarters", 9, 11, True, True], # room 46
        [FRIEND2_NAME + "'s sleeping quarters", 9, 11, True, True], # room 47
        ["The pipeworks", 13, 11, True, False], # room 48
        ["The chief scientist's office", 9, 7, True, True], # room 49
        ["The robot workshop", 9, 11, True, False] # room 50
        ]

#simple sanity check on map above to check data entry
assert len(GAME_MAP)-1 == MAP_SIZE, "Map size and GAME_MAP don't match"


###############
##  OBJECTS  ##
###############

objects = {
    0: [images.floor, None, "The floor is shiny and clean"],
    1: [images.pillar, images.full_shadow, "The wall is smooth and cold"],
    2: [images.soil, None, "It's like a desert. Or should that be desert?"],
    3: [images.pillar_low, images.half_shadow, "The wall is smooth and cold"],
    4: [images.bed, images.half_shadow, "A tidy and comfortable bed"],
    5: [images.table, images.half_shadow, "It's made from strong plastic."],
    6: [images.chair_left, None, "A chair with a soft cushion"],
    7: [images.chair_right, None, "A chair with a soft cushion"],
    8: [images.bookcase_tall, images.full_shadow,
        "Bookshelves, stacked with reference books"],
    9: [images.bookcase_small, images.half_shadow,
        "Bookshelves, stacked with reference books"],
    10: [images.cabinet, images.half_shadow,
         "A small locker, for storing personal items"],
    11: [images.desk_computer, images.half_shadow,
         "A computer. Use it to run life support diagnostics"],
    12: [images.plant, images.plant_shadow, "A spaceberry plant, grown here"],
    13: [images.electrical1, images.half_shadow,
         "Electrical systems used for powering the space station"],
    14: [images.electrical2, images.half_shadow,
         "Electrical systems used for powering the space station"],
    15: [images.cactus, images.cactus_shadow, "Ouch! Careful on the cactus!"],
    16: [images.shrub, images.shrub_shadow,
         "A space lettuce. A bit limp, but amazing it's growing here!"],
    17: [images.pipe1, images.pipe1_shadow, "Water purification pipes"],
    18: [images.pipe2, images.pipe2_shadow,
         "Pipes for the life support systems"],
    19: [images.pipe3, images.pipe3_shadow,
         "Pipes for the life support systems"],
    20: [images.door, images.door_shadow, "Safety door. Opens automatically \
for astronauts in functioning spacesuits."],
    21: [images.door, images.door_shadow, "The airlock door. \
For safety reasons, it requires two person operation."],
    22: [images.door, images.door_shadow, "A locked door. It needs " \
         + PLAYER_NAME + "'s access card"],
    23: [images.door, images.door_shadow, "A locked door. It needs " \
         + FRIEND1_NAME + "'s access card"],
    24: [images.door, images.door_shadow, "A locked door. It needs " \
         + FRIEND2_NAME + "'s access card"],
    25: [images.door, images.door_shadow,
         "A locked door. It is opened from Main Mission Control"],
    26: [images.door, images.door_shadow,
         "A locked door in the engineering bay."],
    27: [images.map, images.full_shadow,
         "The screen says the crash site was Sector: " \
         + str(LANDER_SECTOR) + " // X: " + str(LANDER_X) + \
         " // Y: " + str(LANDER_Y)],
    28: [images.rock_large, images.rock_large_shadow,
         "A rock. Its coarse surface feels like a whetstone", "the rock"],
    29: [images.rock_small, images.rock_small_shadow,
         "A small but heavy piece of Martian rock"],
    30: [images.crater, None, "A crater in the planet surface"],
    31: [images.fence, None,
         "A fine gauze fence. It helps protect the station from dust storms"],
    32: [images.contraption, images.contraption_shadow,
         "One of the scientific experiments. It gently vibrates"],
    33: [images.robot_arm, images.robot_arm_shadow,
         "A robot arm. Used for heavy lifting"],
    34: [images.toilet, images.half_shadow, "A sparkling clean toilet"],
    35: [images.sink, None, "A sink with running water", "the taps"],
    36: [images.globe, images.globe_shadow,
         "A giant globe of the planet. It gently glows from inside"],
    37: [images.science_lab_table, None,
         "A table of experiments, analyzing the planet soil and dust"],
    38: [images.vending_machine, images.full_shadow,
         "A vending machine. It requires a credit.", "the vending machine"],
    39: [images.floor_pad, None,
         "A pressure sensor to make sure nobody goes out alone."],
    40: [images.rescue_ship, images.rescue_ship_shadow, "A rescue ship!"],
    41: [images.mission_control_desk, images.mission_control_desk_shadow, \
         "Mission Control stations."],
    42: [images.button, images.button_shadow,
         "The button for opening the time-locked door in engineering."],
    43: [images.whiteboard, images.full_shadow,
         "The whiteboard is used in brainstorms and planning meetings."],
    44: [images.window, images.full_shadow,
         "The window provides a view out onto the planet surface."],
    45: [images.robot, images.robot_shadow, "A cleaning robot, turned off."],
    46: [images.robot2, images.robot2_shadow,
         "A planet surface exploration robot, awaiting set-up."],
    47: [images.rocket, images.rocket_shadow, "A one-person craft in repair"],
    48: [images.toxic_floor, None, "Toxic floor - do not walk on!"],
```

```
    49: [images.drone, None, "A delivery drone"],
    50: [images.energy_ball, None, "An energy ball - dangerous!"],
    51: [images.energy_ball2, None, "An energy ball - dangerous!"],
    52: [images.computer, images.computer_shadow,
         "A computer workstation, for managing space station systems."],
    53: [images.clipboard, None,
         "A clipboard. Someone has doodled on it.", "the clipboard"],
    54: [images.bubble_gum, None,
         "A piece of sticky bubble gum. Spaceberry flavour.", "bubble gum"],
    55: [images.yoyo, None, "A toy made of fine, strong string and plastic. \
Used for antigrav experiments.", PLAYER_NAME + "'s yoyo"],
    56: [images.thread, None,
         "A piece of fine, strong string", "a piece of string"],
    57: [images.needle, None,
         "A sharp needle from a cactus plant", "a cactus needle"],
    58: [images.threaded_needle, None,
         "A cactus needle, spearing a length of string", "needle and string"],
    59: [images.canister, None,
         "The air canister has a leak.", "A leaky air canister"],
    60: [images.canister, None,
         "It looks like the seal will hold!", "A sealed air canister"],
    61: [images.mirror, None,
         "The mirror throws a circle of light on the walls.", "a mirror"],
    62: [images.bin_empty, None,
         "A rarely used bin, made of light plastic", "a bin"],
    63: [images.bin_full, None,
         "A heavy bin full of water", "a bin full of water"],
    64: [images.rags, None,
         "An oily rag. Pick it up by one corner if you must!", "an oily rag"],
    65: [images.hammer, None,
         "A hammer. Maybe good for cracking things open...", "a hammer"],
    66: [images.spoon, None, "A large serving spoon", "a spoon"],
    67: [images.food_pouch, None,
         "A dehydrated food pouch. It needs water.", "a dry food pack"],
    68: [images.food, None,
         "A food pouch. Use it to get 100% energy.", "ready-to-eat food"],
    69: [images.book, None, "The book has the words 'Don't Panic' on the \
cover in large, friendly letters", "a book"],
    70: [images.mp3_player, None,
         "An MP3 player, with all the latest tunes", "an MP3 player"],
    71: [images.lander, None, "The Poodle, a small space exploration craft. \
Its black box has a radio sealed inside.", "the Poodle Lander"],
    72: [images.radio, None, "A radio communications system, from the \
Poodle", "a communications radio"],
    73: [images.gps_module, None, "A GPS Module", "a GPS module"],
    74: [images.positioning_system, None, "Part of a positioning system. \
Needs a GPS module.", "a positioning interface"],
    75: [images.positioning_system, None,
         "A working positioning system", "a positioning computer"],
    76: [images.scissors, None, "Scissors. They're too blunt to cut \
anything. Can you sharpen them?", "blunt scissors"],
    77: [images.scissors, None,
         "Razor-sharp scissors. Careful!", "sharpened scissors"],
    78: [images.credit, None,
         "A small coin for the station's vending systems",
         "a station credit"],
    79: [images.access_card, None,
         "This access card belongs to " + PLAYER_NAME, "an access card"],
    80: [images.access_card, None,
         "This access card belongs to " + FRIEND1_NAME, "an access card"],
    81: [images.access_card, None,
         "This access card belongs to " + FRIEND2_NAME, "an access card"]
    }

items_player_may_carry = list(range(69, 82))
# Numbers below are for floor, pressure pad, soil, toxic floor.
items_player_may_stand_on = items_player_may_carry + [0, 39, 2, 45]


###############
##  SCENERY  ##
###############

# Scenery describes objects that cannot move between rooms.
# room number: [[object number, y position, x position]...]
scenery = {
    26: [[39,8,2]],
    27: [[33,5,5], [33,1,1], [33,1,8], [47,5,2],
         [47,3,10], [47,5,8], [42,1,6]],
    28: [[27,0,3], [41,4,3], [41,4,7]],
    29: [[7,2,6], [6,2,8], [12,1,13], [44,0,1],
         [36,4,10], [10,1,1], [19,4,2], [17,4,4]],
    30: [[34,1,1], [35,1,31],
    31: [[11,1,1], [19,1,8], [46,1,3]],
    32: [[48,2,2], [48,2,3], [48,2,4], [48,3,2], [48,3,3],
         [48,3,4], [48,4,2], [48,4,3], [48,4,4]],
    33: [[13,1,1], [13,1,9], [13,1,8], [13,1,10], [48,2,1],
         [48,2,7], [48,3,6], [48,3,11],
    34: [[37,2,2], [32,6,7], [37,10,4], [28,5,3]],
    35: [[16,2,9], [16,2,2], [16,3,9], [16,3,0], [16,8,9], [16,8,2], [16,1,8],
         [16,1,9], [12,8,6], [12,9,4], [12,9,8],
         [15,4,6], [12,7,1], [12,7,11]],
    36: [[4,3,1], [9,1,7], [5,1,8], [9,1,9],
         [5,5,4], [6,5,7], [10,1,1], [12,1,2]],
    37: [[48,3,1], [48,9,2], [48,7,1], [49,5,2], [49,5,3],
         [48,7,2], [48,9,2], [48,9,3], [48,11,1], [48,11,2]],
    38: [[43,0,2], [6,2,2], [6,3,6], [6,4,7], [6,2,5], [49,1,10]],
    39: [[35,1,1], [7,3,4], [7,6,4], [5,5,6], [5,6,6],
         [6,3,9], [6,6,9], [45,1,11], [12,1,5], [12,1,4],
    40: [[41,5,3], [41,5,7], [41,8,3], [41,9,7],
         [13,1,1], [13,1,8], [43,1,12]],
    41: [[4,3,1], [20,3,5], [4,5,1], [20,5,5], [4,7,1],
         [20,7,5], [12,1,11], [12,1,5]],
    44: [[46,4,3], [46,4,5], [18,1,1], [19,1,3],
         [19,1,5], [52,9,7], [14,1,5]],
    45: [[48,2,1], [48,2,2], [48,3,3], [48,3,4], [46,1,5], [46,1,1]],
    46: [[20,1,1], [5,1,2], [5,1,7], [5,1,8], [5,1,5], [5,5,3], [7,3,2]],
    47: [[5,1,1], [5,1,2], [10,1,3], [13,1,7], [5,4,4], [6,4,7], [4,1,8]],
    48: [[17,4,1], [17,4,2], [17,4,3], [17,4,4], [17,4,5], [17,4,6], [17,4,7],
         [17,5,1], [17,5,2], [17,5,3], [17,5,4],
         [17,5,5], [17,5,6], [17,5,7], [14,1,1]],
    49: [[14,2,3], [14,2,4], [7,5,1], [5,5,3], [48,3,3], [48,3,4]],
    50: [[45,4,5], [11,1,1], [13,1,5], [33,2,1], [46,4,6]]
    }
```

```
checksum = 0
check_counter = 0
for key, room_scenery_list in scenery.items():
    for scenery_item_list in room_scenery_list:
        checksum += (scenery_item_list[0] * key
                     + scenery_item_list[1] * (key + 1)
                     + scenery_item_list[2] * (key + 2))
        check_counter += 1
print(check_counter, "scenery items")
assert check_counter == 161, "Expected 161 scenery items"
assert checksum == 200095, "Error in scenery data"
print("Scenery checksum: " + str(checksum))

for room in range(1, 26):# Add random scenery in planet locations.
    if room != 13: # Skip room 13.
        scenery_item = random.choice([16, 28, 29, 30])
        scenery[room] = [[scenery_item, random.randint(2, 10),
                          random.randint(2, 10)]]

# Use loops to add fences to the planet surface rooms.
for room_coordinate in range(0, 13):
    for room_number in [1, 2, 3, 4, 5]: # Add top fence
        scenery[room_number] += [[31, 0, room_coordinate]]
    for room_number in [1, 6, 11, 16, 21]: # Add left fence
        scenery[room_number] += [[31, room_coordinate, 0]]
    for room_number in [5, 10, 15, 20, 25]: # Add right fence
        scenery[room_number] += [[31, room_coordinate, 12]]

del scenery[21][-1] # Delete last fence panel in Room 21
del scenery[25][-1] # Delete last fence panel in Room 25


###############
## MAKE MAP  ##
###############

def get_floor_type():
    if current_room in outdoor_rooms:
        return 2 # soil
    else:
        return 0 # tiled floor

def generate_map():
# This function makes the map for the current room,
# using room data, scenery data and prop data.
    global room_map, room_width, room_height, room_name, hazard_map
    global top_left_x, top_left_y, wall_transparency_frame
    room_data = GAME_MAP[current_room]
    room_name = room_data[0]
    room_height = room_data[1]
    room_width = room_data[2]

    floor_type = get_floor_type()
    if current_room in range(1, 21):
        bottom_edge = 2 #soil
        side_edge = 2 #soil
    if current_room in range(22, 26):
        bottom_edge = 1 #wall
        side_edge = 2 #soil
    if current_room > 25:
        bottom_edge = 1 #wall
        side_edge = 1 #wall

    # Create top line of room map.
    room_map=[[side_edge] * room_width]
    # Add middle lines of room map (wall, floor to fill width, wall).
    for y in range(room_height - 2):
        room_map.append([side_edge]
                        + [floor_type]*(room_width - 2) + [side_edge])
    # Add bottom line of room map.
    room_map.append([bottom_edge] * room_width)

    # Add doorways.
    middle_row = int(room_height / 2)
    middle_column = int(room_width / 2)

    if room_data[4]: # If exit at right of this room
        room_map[middle_row][room_width - 1] = floor_type
        room_map[middle_row+1][room_width - 1] = floor_type
        room_map[middle_row-1][room_width - 1] = floor_type

    if current_room % MAP_WIDTH != 1: # If room is not on left of map
        room_to_left = GAME_MAP[current_room - 1]
        # If room on the left has a right exit, add left exit in this room
        if room_to_left[4]:
            room_map[middle_row][0] = floor_type
            room_map[middle_row + 1][0] = floor_type
            room_map[middle_row - 1][0] = floor_type

    if room_data[3]: # If exit at top of this room
        room_map[0][middle_column] = floor_type
        room_map[0][middle_column + 1] = floor_type
        room_map[0][middle_column - 1] = floor_type

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

    if current_room in scenery:
        for this_scenery in scenery[current_room]:
            scenery_number = this_scenery[0]
            scenery_y = this_scenery[1]
            scenery_x = this_scenery[2]
            room_map[scenery_y][scenery_x] = scenery_number

            image_here = objects[scenery_number][0]
            image_width = image_here.get_width()
            image_width_in_tiles = int(image_width / TILE_SIZE)

            for tile_number in range(1, image_width_in_tiles):
                room_map[scenery_y][scenery_x + tile_number] = 255
```

```
    center_y = int(HEIGHT / 2) # Center of game window
    center_x = int(WIDTH / 2)
    room_pixel_width = room_width * TILE_SIZE # Size of room in pixels
    room_pixel_height = room_height * TILE_SIZE
    top_left_x = center_x - 0.5 * room_pixel_width
    top_left_y = (center_y - 0.5 * room_pixel_height) + 110

    for prop_number, prop_info in props.items():
        prop_room = prop_info[0]
        prop_y = prop_info[1]
        prop_x = prop_info[2]
        if (prop_room == current_room and
            room_map[prop_y][prop_x] in [0, 39, 2]):
                room_map[prop_y][prop_x] = prop_number
                image_here = objects[prop_number][0]
                image_width = image_here.get_width()
                image_width_in_tiles = int(image_width / TILE_SIZE)
                for tile_number in range(1, image_width_in_tiles):
                    room_map[prop_y][prop_x + tile_number] = 255


###############
## GAME LOOP ##
###############

def start_room():
    show_text("You are here: " + room_name, 0)

def game_loop():
    global player_x, player_y, current_room
    global from_player_x, from_player_y
    global player_image, player_image_shadow
    global selected_item, item_carrying, energy
    global player_offset_x, player_offset_y
    global player_frame, player_direction

    if game_over:
        return

    if player_frame > 0:
        player_frame += 1
        time.sleep(0.05)
        if player_frame == 5:
            player_frame = 0
            player_offset_x = 0
            player_offset_y = 0

#save player's current position
    old_player_x = player_x
    old_player_y = player_y

# move if key is pressed
    if player_frame == 0:
        if keyboard.right:
            from_player_x = player_x
            from_player_y = player_y
            player_x += 1
            player_direction = "right"
            player_frame = 1
        elif keyboard.left: #elif stops player making diagonal movements
            from_player_x = player_x
            from_player_y = player_y
            player_x -= 1
            player_direction = "left"
            player_frame = 1
        elif keyboard.up:
            from_player_x = player_x
            from_player_y = player_y
            player_y -= 1
            player_direction = "up"
            player_frame = 1
        elif keyboard.down:
            from_player_x = player_x
            from_player_y = player_y
            player_y += 1
            player_direction = "down"
            player_frame = 1

# check for exiting the room
    if player_x == room_width: # through door on RIGHT
        #clock.unschedule(hazard_move)
        current_room += 1
        generate_map()
        player_x = 0 # enter at left
        player_y = int(room_height / 2) # enter at door
        player_frame = 0
        start_room()
        return

    if player_x == -1: # through door on LEFT
        #clock.unschedule(hazard_move)
        current_room -= 1
        generate_map()
        player_x = room_width - 1  # enter at right
        player_y = int(room_height / 2) # enter at door
        player_frame = 0
        start_room()
        return

    if player_y == room_height: # through door at BOTTOM
        #clock.unschedule(hazard_move)
        current_room += MAP_WIDTH
        generate_map()
        player_y = 0 # enter at top
        player_x = int(room_width / 2) # enter at door
        player_frame = 0
        start_room()
        return
```

```
    if player_y == -1: # through door at TOP
        #clock.unschedule(hazard_move)
        current_room -= MAP_WIDTH
        generate_map()
        player_y = room_height - 1 # enter at bottom
        player_x = int(room_width / 2) # enter at door
        player_frame = 0
        start_room()
        return

    if keyboard.g:
        pick_up_object()

    if keyboard.tab and len(in_my_pockets) > 0:
        selected_item += 1
        if selected_item > len(in_my_pockets) - 1:
            selected_item = 0
        item_carrying = in_my_pockets[selected_item]
        display_inventory()

    if keyboard.d and item_carrying:
        drop_object(old_player_y, old_player_x)

    if keyboard.space:
        examine_object()


     # If the player is standing somewhere they shouldn't, move them back.
    if room_map[player_y][player_x] not in items_player_may_stand_on: #\
    #       or hazard_map[player_y][player_x] != 0:
        player_x = old_player_x
        player_y = old_player_y
        player_frame = 0

    if player_direction == "right" and player_frame > 0:
        player_offset_x = -1 + (0.25 * player_frame)
    if player_direction == "left" and player_frame > 0:
        player_offset_x = 1 - (0.25 * player_frame)
    if player_direction == "up" and player_frame > 0:
        player_offset_y = 1 - (0.25 * player_frame)
    if player_direction == "down" and player_frame > 0:
        player_offset_y = -1 + (0.25 * player_frame)


##############
##  DISPLAY  ##
##############

def draw_image(image, y, x):
    screen.blit(
        image,
        (top_left_x + (x * TILE_SIZE),
         top_left_y + (y * TILE_SIZE) - image.get_height())
        )

def draw_shadow(image, y, x):
    screen.blit(
        image,
        (top_left_x + (x * TILE_SIZE),
         top_left_y + (y * TILE_SIZE))
        )

def draw_player():
    player_image = PLAYER[player_direction][player_frame]
    draw_image(player_image, player_y + player_offset_y,
               player_x + player_offset_x)
    player_image_shadow = PLAYER_SHADOW[player_direction][player_frame]
    draw_shadow(player_image_shadow, player_y + player_offset_y,
                player_x + player_offset_x)

def draw():
    if game_over:
        return

    # Clear the game arena area.
    box = Rect((0, 150), (800, 600))
    screen.draw.filled_rect(box, RED)
    box = Rect ((0, 0), (800, top_left_y + (room_height - 1)*30))
    screen.surface.set_clip(box)
    floor_type = get_floor_type()

    for y in range(room_height): # Lay down floor tiles, then items on floor.
        for x in range(room_width):
            draw_image(objects[floor_type][0], y, x)
            # Next line enables shadows to fall on top of objects on floor
            if room_map[y][x] in items_player_may_stand_on:
                draw_image(objects[room_map[y][x]][0], y, x)

     # Pressure pad in room 26 is added here, so props can go on top of it.
    if current_room == 26:
        draw_image(objects[39][0], 9, 2)
        image_on_pad = room_map[9][2]
        if image_on_pad > 0:
            draw_image(objects[image_on_pad][0], 8, 2)

    for y in range(room_height):
        for x in range(room_width):
            item_here = room_map[y][x]
            # Player cannot walk on 255: it marks spaces used by wide objects.
            if item_here not in items_player_may_stand_on + [255]:
                image = objects[item_here][0]

                if (current_room in outdoor_rooms
                    and y == room_height - 2
                    and room_map[y][x] == 1) or \
                    (current_room not in outdoor_rooms
                    and y == room_height - 1
                    and room_map[y][x] == 1
                    and x > 0
                    and x < room_width - 1):
                    # Add transparent wall image in the front row.
                    image = PILLARS[wall_transparency_frame]

                draw_image(image, y, x)
```

```
                if objects[item_here][1] is not None: # If object has a shadow
                    shadow_image = objects[item_here][1]
                    # If shadow might need horizontal tiling
                    if shadow_image in [images.half_shadow,
                                        images.full_shadow]:
                        shadow_width = int(image.get_width() / TILE_SIZE)
                        # Use shadow across width of object.
                        for z in range(0, shadow_width):
                            draw_shadow(shadow_image, y, x+z)
                    else:
                        draw_shadow(shadow_image, y, x)

        if (player_y == y):
            draw_player()

    screen.surface.set_clip(None)

def adjust_wall_transparency():
    global wall_transparency_frame

    if (player_y == room_height - 2
        and room_map[room_height - 1][player_x] == 1
        and wall_transparency_frame < 4):
        wall_transparency_frame += 1 # Fade wall out.

    if ((player_y < room_height - 2
            or room_map[room_height - 1][player_x] != 1)
            and wall_transparency_frame > 0):
        wall_transparency_frame -= 1 # Fade wall in.

def show_text(text_to_show, line_number):
    if game_over:
        return
    text_lines = [15, 50]
    box = Rect((0, text_lines[line_number]), (800, 35))
    screen.draw.filled_rect(box, BLACK)
    screen.draw.text(text_to_show,
                     (20, text_lines[line_number]), color="GREEN")


###############
##   PROPS   ##
###############

# Props are objects that may move between rooms, appear or disappear.
# All props must be set up here. Props not yet in the game go into room 0.
# object number : [room, y, x]
props = {
    20: [31, 0, 4], 21: [26, 0, 1], 22: [41, 0, 2], 23: [39, 0, 5],
    24: [45, 0, 2],
    25: [32, 0, 2], 26: [27, 12, 6], # two sides of same door
    40: [0, 8, 6], 53: [45, 1, 5], 54: [0, 0, 0], 55: [0, 0, 0],
    56: [0, 0, 0], 57: [35, 4, 6], 58: [0, 0, 0], 59: [31, 1, 7],
    60: [0, 0, 0], 61: [36, 1, 2], 62: [36, 1, 6], 63: [0, 0, 0],
    64: [27, 8, 3], 65: [50, 1, 7], 66: [39, 5, 6], 67: [46, 1, 1],
    68: [0, 0, 0], 69: [30, 3, 3], 70: [47, 1, 3],
    71: [0, LANDER_Y, LANDER_X], 72: [0, 0, 0], 73: [27, 4, 6],
    74: [28, 1, 11], 75: [0, 0, 0], 76: [41, 2, 5], 77: [0, 0, 0],
    78: [35, 9, 11], 79: [26, 3, 2], 80: [41, 7, 5], 81: [29, 1, 1]
    }

checksum = 0
for key, prop in props.items():
    if key != 71: # 71 is skipped because it's different each game.
        checksum += (prop[0] * key
                     + prop[1] * (key + 1)
                     + prop[2] * (key + 2))
print(len(props), "props")
assert len(props) == 37, "Expected 37 prop items"
print("Prop checksum:", checksum)
assert checksum == 64114, "Error in props data"


in_my_pockets = [66]
selected_item = 0 # the first item
item_carrying = in_my_pockets[selected_item]


#######################
## PROP INTERACTIONS ##
#######################

def find_object_start_x():
    checker_x = player_x
    while room_map[player_y][checker_x] == 255:
        checker_x -= 1
    return checker_x

def get_item_under_player():
    item_x = find_object_start_x()
    item_player_is_on = room_map[player_y][item_x]
    return item_player_is_on

def pick_up_object():
    global room_map
    # Get object number at player's location.
    item_player_is_on = get_item_under_player()
    if item_player_is_on in items_player_may_carry:
        # Clear the floor space.
        room_map[player_y][player_x] = get_floor_type()
        add_object(item_player_is_on)
        show_text("Now carrying " + objects[item_player_is_on][3], 0)
        sounds.pickup.play()
        time.sleep(0.5)
    else:
        show_text("You can't carry that!", 0)

def add_object(item): # Adds item to inventory.
    global selected_item, item_carrying
    in_my_pockets.append(item)
    item_carrying = item
    # Minus one because indexes start at 0.
    selected_item = len(in_my_pockets) - 1
    display_inventory()
    props[item][0] = 0 # Carried objects go into room 0 (off the map).

def display_inventory():
    box = Rect((0, 45), (800, 105))
    screen.draw.filled_rect(box, BLACK)

    if len(in_my_pockets) == 0:
        return

    start_display = (selected_item // 16) * 16
    list_to_show = in_my_pockets[start_display : start_display + 16]
    selected_marker = selected_item % 16

    for item_counter in range(len(list_to_show)):
        item_number = list_to_show[item_counter]
        image = objects[item_number][0]
        screen.blit(image, (25 + (46 * item_counter), 90))

    box_left = (selected_marker * 46) - 3
    box = Rect((22 + box_left, 88), (40, 40))
    screen.draw.rect(box, WHITE)
    item_highlighted = in_my_pockets[selected_item]
    description = objects[item_highlighted][2]
    screen.draw.text(description, (20, 130), color="white")

def drop_object(old_y, old_x):
    global room_map, props
    if room_map[old_y][old_x] in [0, 2, 39]: # places you can drop things
        props[item_carrying][0] = current_room
        props[item_carrying][1] = old_y
        props[item_carrying][2] = old_x
        room_map[old_y][old_x] = item_carrying
        show_text("You have dropped " + objects[item_carrying][3], 0)
        sounds.drop.play()
        remove_object(item_carrying)
        time.sleep(0.5)
    else: # This only happens if there is already a prop here
        show_text("You can't drop that there.", 0)
        time.sleep(0.5)
```

```
def remove_object(item): # Takes item out of inventory
    global selected_item, in_my_pockets, item_carrying
    in_my_pockets.remove(item)
    selected_item = selected_item - 1
    if selected_item < 0:
        selected_item = 0
    if len(in_my_pockets) == 0: # If they're not carrying anything
        item_carrying = False # Set item_carrying to False
    else: # Otherwise set it to the new selected item
        item_carrying = in_my_pockets[selected_item]
    display_inventory()

def examine_object():
    item_player_is_on = get_item_under_player()
    left_tile_of_item = find_object_start_x()
    if item_player_is_on in [0, 2]: # don't describe the floor
        return
    description = "You see: " + objects[item_player_is_on][3]
    for prop_number, details in props.items():
        # props = object number: [room number, y, x]
        if details[0] == current_room: # if prop is in the room
            # If prop is hidden (= at player's location but not on map)
            if (details[1] == player_y
                and details[2] == left_tile_of_item
                and room_map[details[1]][details[2]] != prop_number):
                add_object(prop_number)
                description = "You found " + objects[prop_number][3]
                sounds.combine.play()
    show_text(description, 0)
    time.sleep(0.5)


##############
##  START   ##
##############

generate_map()
clock.schedule_interval(game_loop, 0.03)
clock.schedule_interval(adjust_wall_transparency, 0.05)
clock.schedule_unique(display_inventory, 1)
```