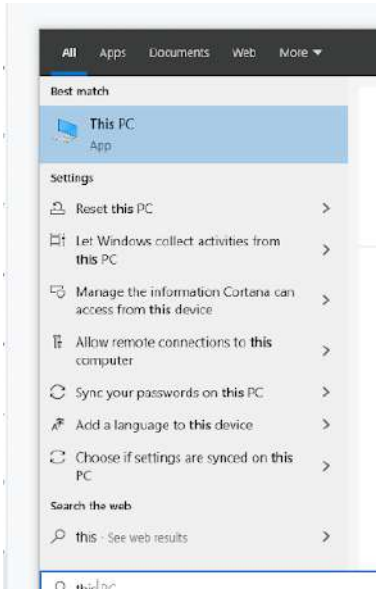
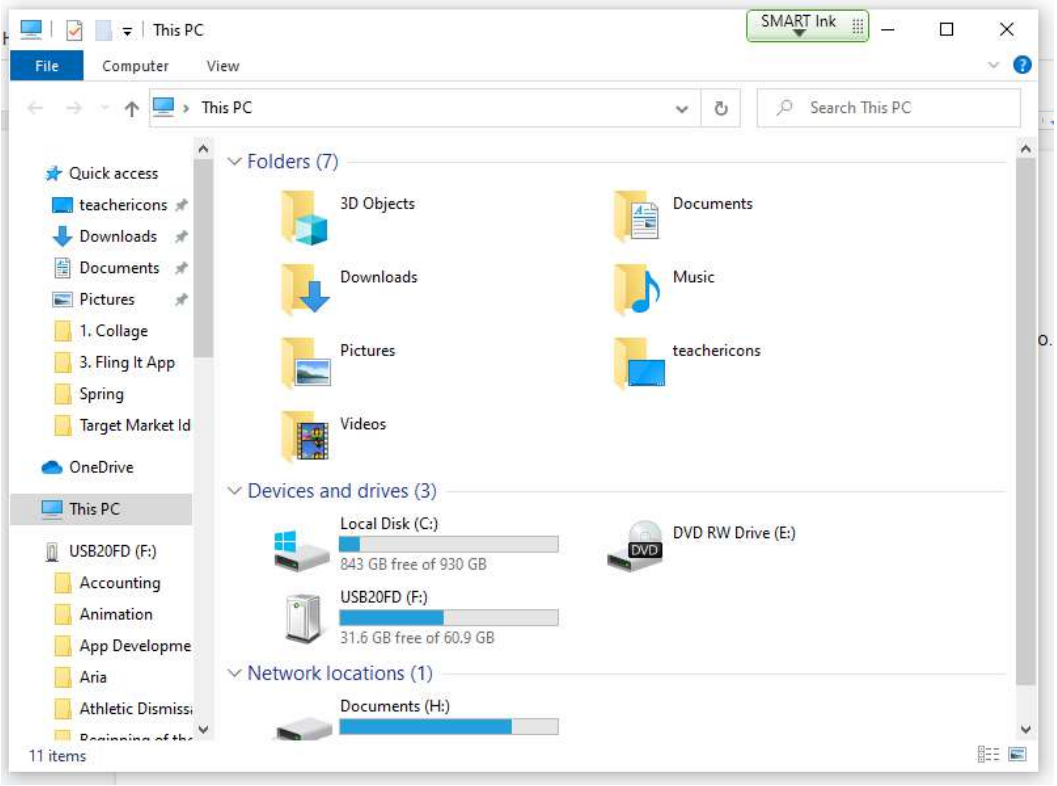
Coding a Balloon Burst Game in Pygame

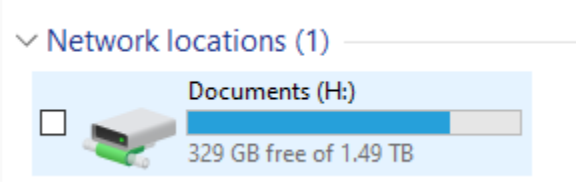
1. First, we need to create a folder in our student directory to save all of our Python files to. Click your Windows button in the bottom left corner and search for “This PC”.



2. Click on “This PC” to pull up your Windows Explorer window.

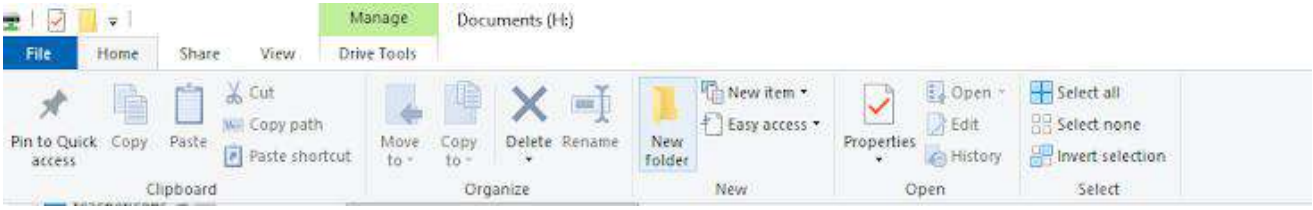


3. Locate your student drive in the menu. Most students have an H: drive. However, some students have a V: drive. You will see your username attached to your student drive. If you don't have a student drive, please let me know. This probably means that we need to have your student drives pulled over from the Monroe server.

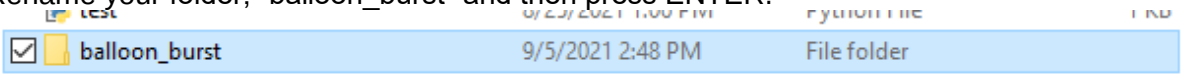


4. Double-click on your H: or V: drive to open it.

5. Click on Home > New Folder to create a new folder.

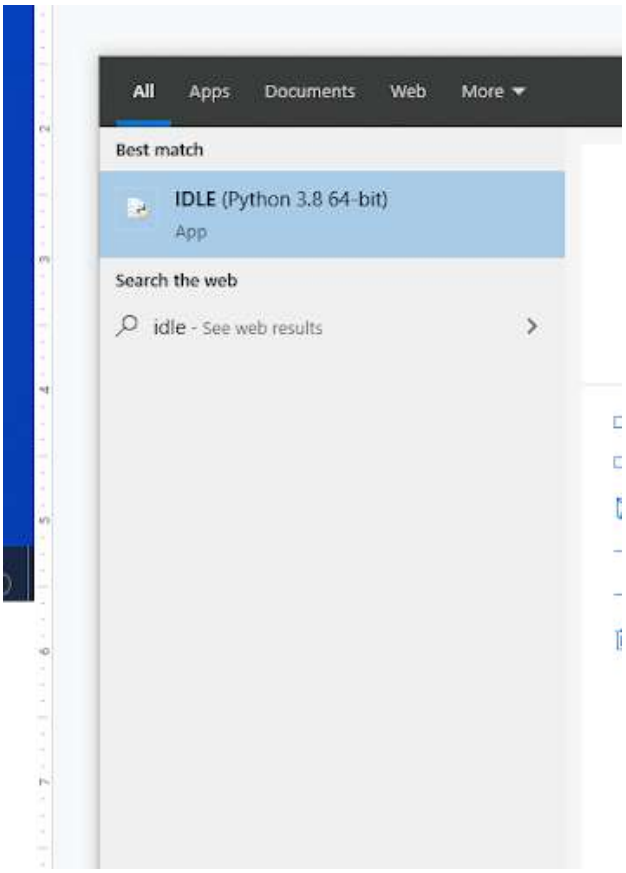


6. Rename your folder, "balloon_burst" and then press ENTER.



7. You can close out of your Windows Explorer window after you have made your folder.

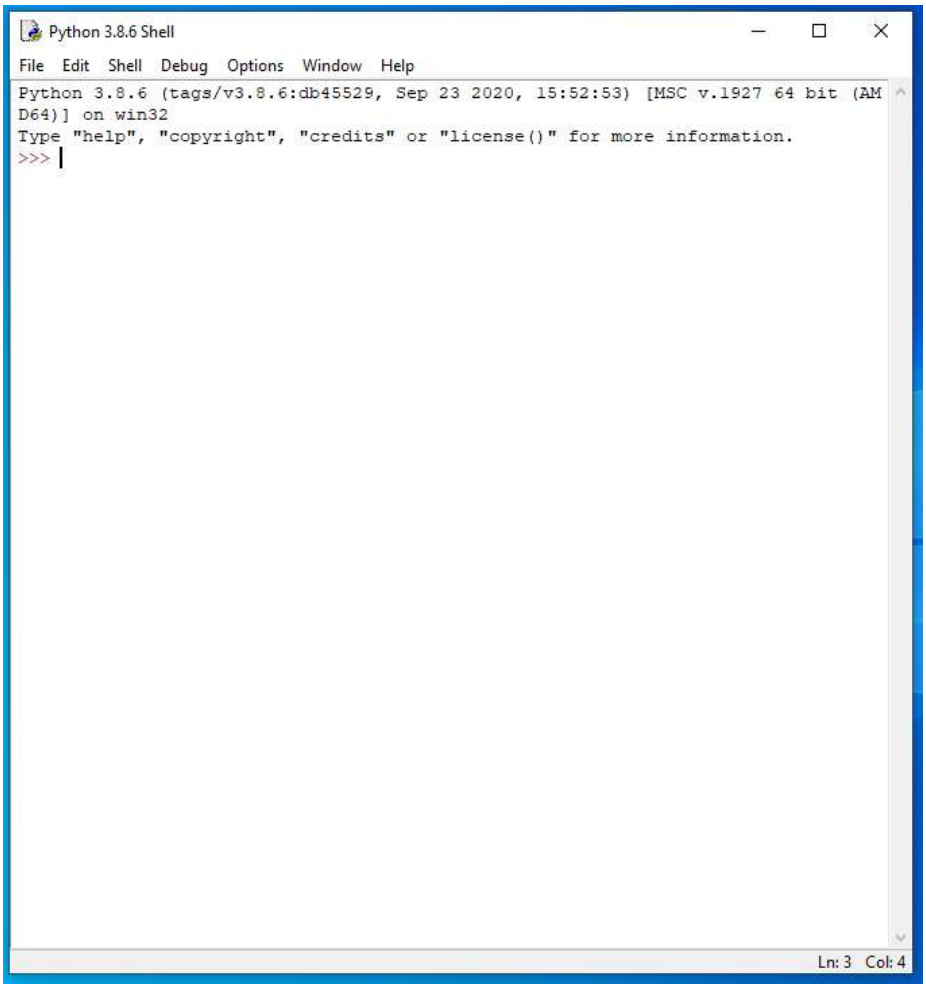
8. Using your Windows button menu, find and launch your IDLE program.



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.

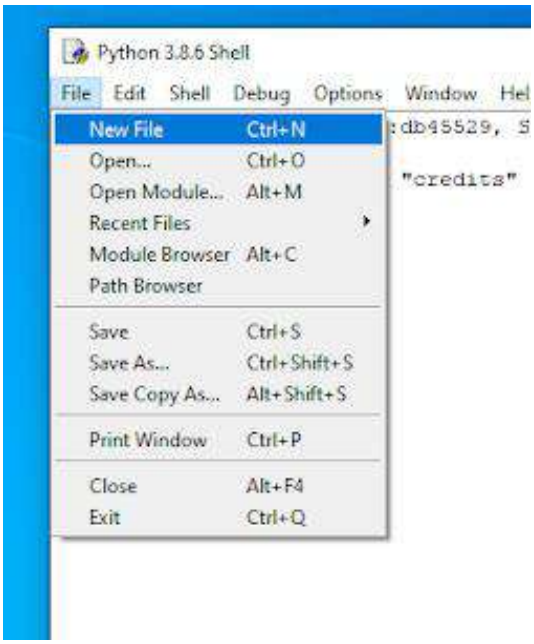
9. Your IDLE window should look something like this once it has launched.:



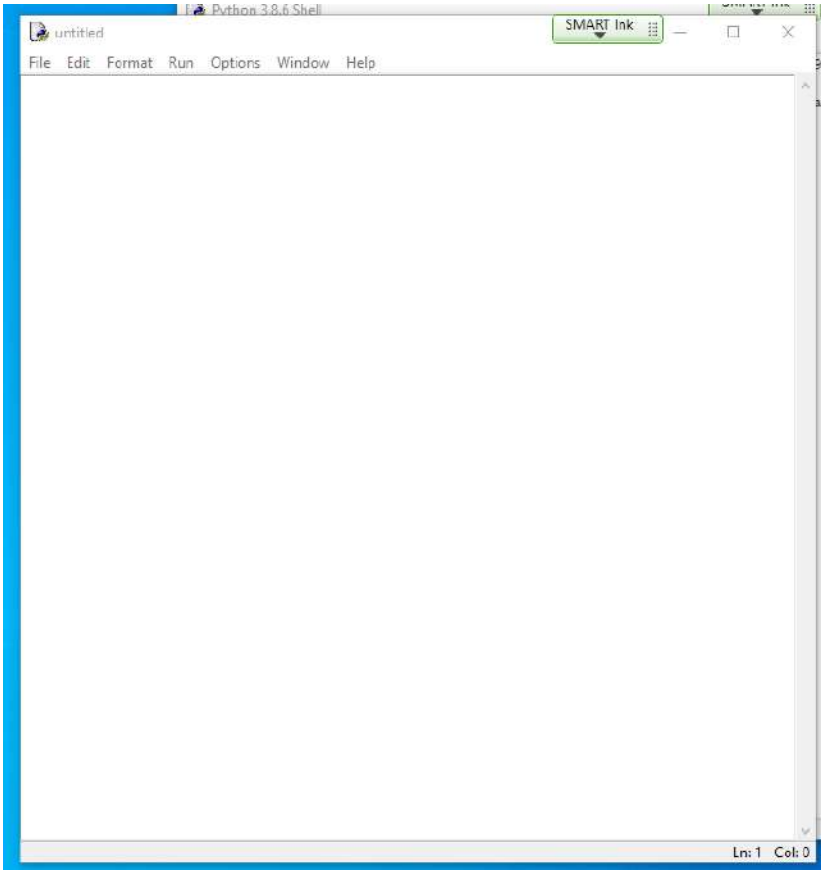
On Startup, IDLE will display the Python Shell, which can be used to give commands to the computer's operating system. Since we are viewing the shell through IDLE and not the actual command prompt window, the commands that we type into the Shell will not communicate directly with our operating system. 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.

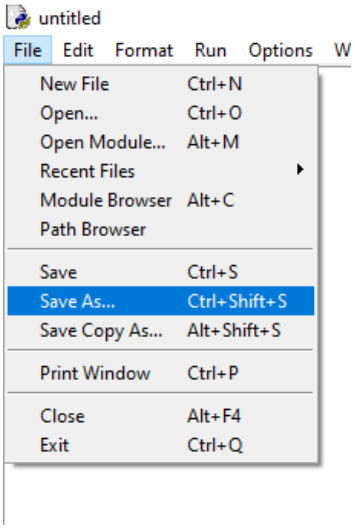
10. Go to File > New File to create a new Python file.



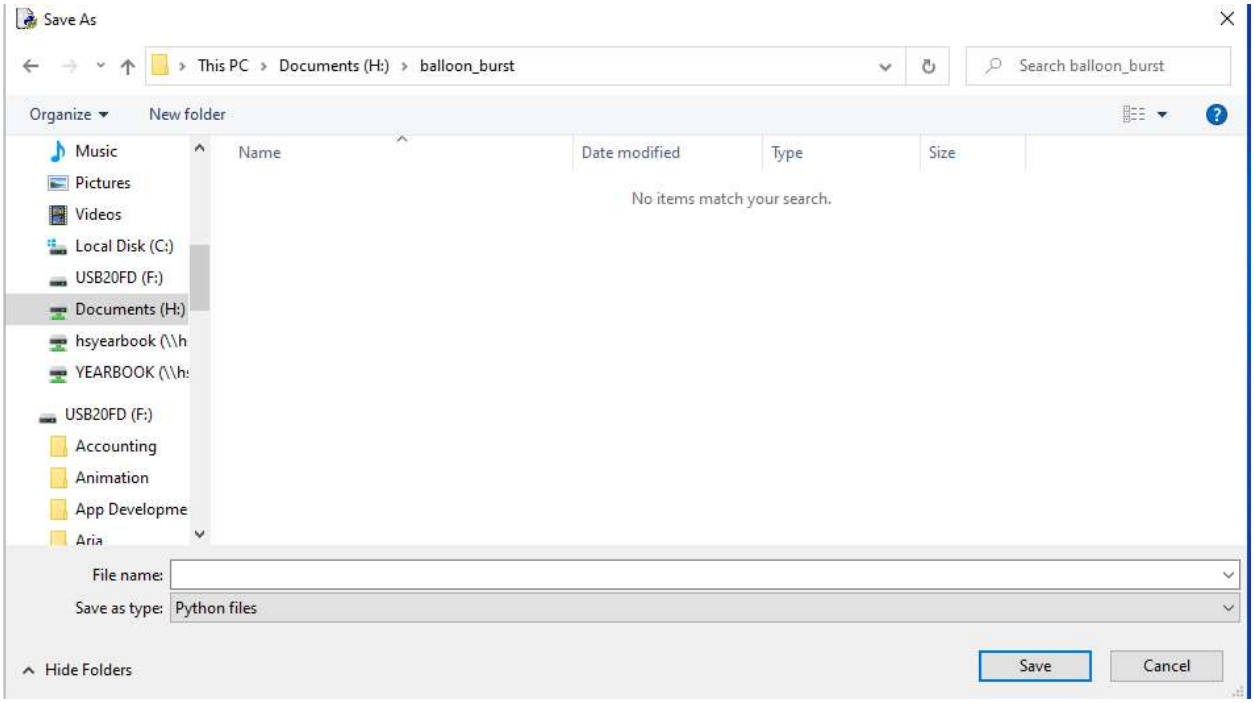
11. A blank Python file will open up. You can tell this apart from the Shell because the title bar will display the file name “untitled” right now because we haven’t saved it yet.



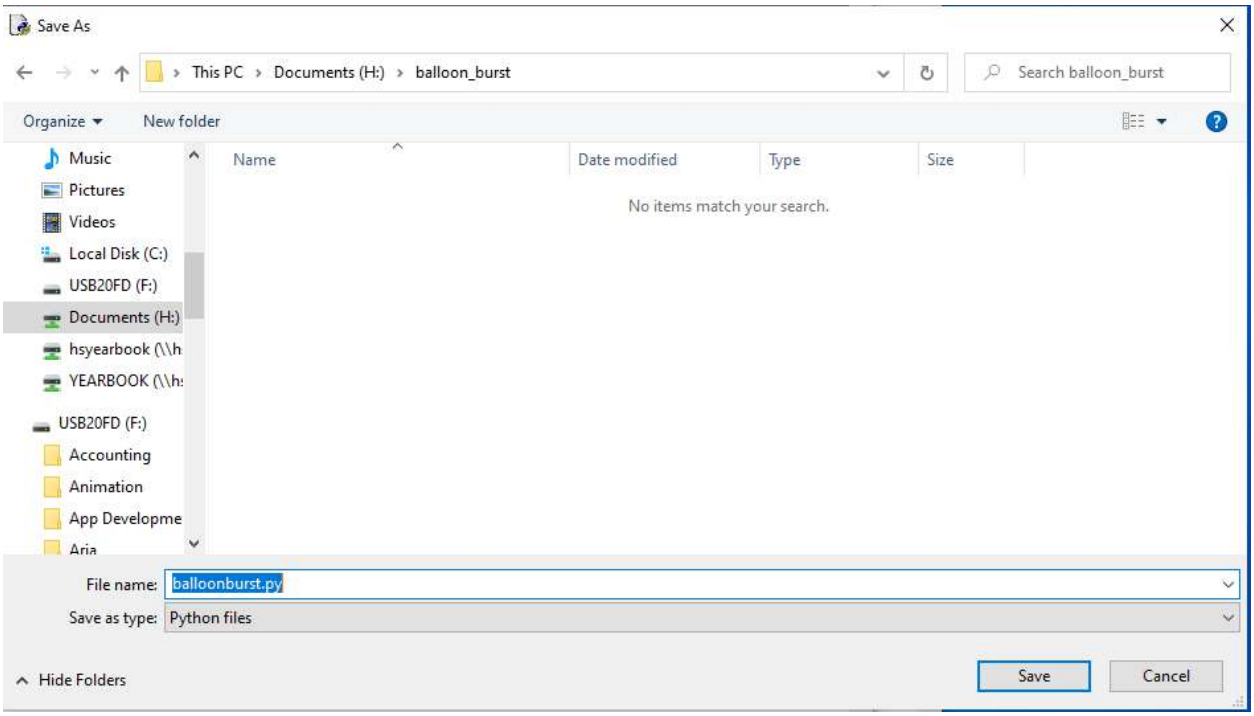
12. Go to File > Save As to save your file.



13. Navigate to your H: or V: drive and select the balloon_burst folder that you created earlier. Double-click the folder to open it.

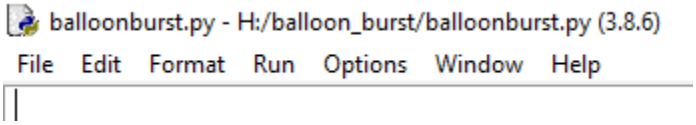


14. Type your new file name in the File Name box. Your new file name is balloonburst.py



15. Click the Save button.

16. You should now notice that your new Python file has been renamed. You will see the new name in the title bar along the top of the file.



17. We will now begin to code the first part of our game. I like to make my coding window larger so that I can see all of my code a bit better, but that is a personal decision. Remember as we move through these exercises that your spelling, capitalization, and indentation should match. If it doesn't, your program likely won't work.

18. Type the following lines of code (Lines 1 – 16) into your Python file.

```
1  #Basic Pygame Structure
2
3  import pygame
4
5  #Define Classes (sprites) here
6
7  pygame.init()
8
9  screen = pygame.display.set_mode([800,400])
10 pygame.display.set_caption("Balloon Burst")
11 background_image = pygame.image.load("SkyBackground.png").convert()
12 pygame.mouse.set_visible(False)
13 done = False
14 clock = pygame.time.Clock()
15 black = (0, 0, 0)
16 white = (255, 255, 255)
```

Line 1 contains a comment explaining that you are going to layout the basic Pygame structure of code in the lines below.

Line 3 imports the pygame module into the file. This ensures that Python will reference the pygame module of code whenever the game is run.

Line 5 contains another comment. You will eventually define different classes or sprites at the beginning of the program code under this comment. For now, don't worry about the different classes in the game.

Line 7 initializes Pygame. This makes the Pygame library that we imported on Line 3 do the initial, behind-the-scenes setup that it needs in order to run. Initializing variables or objects prepares them to be used by the program for the first time. It is good practice to initialize every library/module you import to make sure it is up and running at the start of your program.

Line 9 creates the screen variable and sets its value to be the result of the display.set_mode method that uses the dimensions specified in the argument (in this case, 800 pixels wide and 400 pixels tall). The display.set_mode method creates a display Surface using the arguments it is given in the parentheses. In this case, the display.set_mode method creates a game window and assigns that window's size to the screen variable.

Line 10 changes the title of the display window to whatever argument you give it in the parentheses. In this case, the title of the display window will be "Balloon Burst".

Line 11 creates a new variable called background_img. The variable's value is set to be equal to the SkyBackground.png image. Remember, all image files must be saved in the same folder as your code file or it won't work. The image.load() command is used to load

an image into your game. The command finishes with the convert() method, that converts the background image into a format that is used by whatever display the user is on. When you use the .convert() function, the image will load faster on the screen every time it is blitted. If the formats of the image in the game do not match the user's display, then the game will automatically have to convert formats every time that image is blitted, which takes a lot of processing power. If the image it converted at the beginning when it is loaded into the program, this avoids the program having to convert the image every time.

The .convert() command is not crucial to coding. The code will work without it. However, the game will be slower to load if it has to blit and convert the image several times throughout its game loop.

Line 12 will set the user's mouse visibility to False, meaning that the cursor isn't shown on the screen.

Line 13 will create a variable called done, which is set to False.

Line 14 creates another variable called clock. The variable's value is set to be equal to the value of the time.Clock() method from the pygame module. Every Pygame file comes with the time module built in, and the time module is actually a collection of various different objects/methods that can be used for different timing components of games. Each object/method within the time module provides a variety of different functionality options. In this case, Line 14 sets up a new clock object in the game and assigns that object to the variable called clock. The clock object gives the programmer the ability to track time, in seconds and milliseconds, for anything in the game.

Lines 15 and 16 create two new variables called black and white. The values of these variables contain the RGB number values for the colors black and white.

19. Press ENTER twice.

20. Type the code that you see on Lines 18 – 32 of your code.

```
16 white = (255, 255, 255)
17
18 #Define additional Functions and Procedures here
19
20 #------------- Main Program Loop -------------
21 while done == False:
22
23     for event in pygame.event.get():
24         if event.type == pygame.QUIT:
25             done = True
26
27     #Update sprites here
28
29     pygame.display.flip()
30     clock.tick(20)
31
32 pygame.quit()
```

Line 18 contains a comment. You will eventually define additional functions and procedures in this game, and Line 18 creates a special section for you to type those functions and procedures into.

Line 19 is a blank line.

Line 20 contains another comment indicating that the Main Program Loop for the game will be starting.

Line 21 contains a “while” loop. This loop will continue to run while the value of the done variable is equal to False. Remember, you created the “done” variable on Line 13 and set its initial value to False.

Line 22 is blank.

Line 23 begins a “for” loop. This loop will check for the events that occur in the game. Events are things that can happen in a game. Things like button presses and quitting the game are events. In this section of code, we will be programming behaviors that happen when certain events take place.

Line 24 creates an if function that checks to see if the event.type is equal to the value of QUIT. Python has a variety of events that it generates based on different actions the user takes. One of the major events it uses is the QUIT event. The QUIT event is generated whenever the user closes out of the Pygame window or whenever the game asks to be closed. If the user closes out of the Pygame window or the system asks to

close, the QUIT event is generated in the event queue, which is then picked up by the pygame.event.get(): method.

On Line 25, you see the what will happen if the event.type==QUIT (Line 24) is detected in the event queue. If the event type is NOT EQUAL to QUIT, this line (Line 25) will be skipped and the program will continue on to the next line. If the event type is equal to QUIT, the done variable will be changed to True.

Line 26 is blank.

Line 27 contains a comment indicating that the code to update your sprites will, eventually, be typed in that section.

Line 28 is blank again.

Line 29 contains the display.flip() command. This command can be a bit confusing. This command does not actually flip the contents of your game window. The display.flip() command will update the contents of the entire display. In other words, it will redraw all of the sprites and background on your display.

Line 30 contains the clock.tick command. This command is set to 20 frames. What this means is that the program will run at a rate of 20 frames per second. This means that the display, which updates every frame, will update at a rate of 20 times per second. Remember that this game loop will run every frame, and the display needs to be updated at the end of each frame to reflect the new settings on the display (for example, the new position of the sprite or other elements in your game or the new score or time left in the game).

Line 31 is blank.

Line 32 contains the pygame.quit() command which will stop the game and close the window.

21. Click at the end of Line 27 (at the end of the “#Update sprites here” comment)

```
26
27      #Update sprites here
28
29      pygame.display.flip()
30      clock.tick(20)
31
32  pygame.quit()
```

22. Press ENTER twice.

23. Type the code that you see on Line 30 of the screenshot below. You will have to move your cursor down to Line 30 and ensure that the indentation matches what is shown in the screenshot.

```
27      #Update sprites here
28
29
30      screen.blit(background_image, [0,0])
31      pygame.display.flip()
32      clock.tick(20)
33
34  pygame.quit()
```

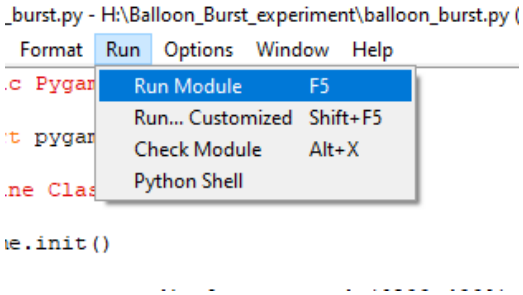
    Line 30 will blit, or draw, the image stored in the background_image variable (the variable we created on Line 11 that uses the SkyBackground.png image) on top of the screen surface that we created on Line 9. Remember, all items, including images and sprites, need a surface to display on. Before you can set a background image for your game, you need to create a surface to display it on.

    The blit command you typed on Line 30 contains coordinates which are used to position one image over the other.

    The “screen” surface that we created on Line 9 has a resolution of 800x400. The image in the “background_image” variable, the SkyBackground image, has a resolution of 800x400 as well. Items are placed at a certain location in a game window using x and y coordinates. The top left corner of the game screen/window has the coordinate of 0, 0. The y coordinate increases as items move down further away from the top corner of the screen and the x coordinate increases as it moves to the right across the game window. To place one surface over another of the same size, we would position it using the coordinates of 0, 0. This would place the top left corner of the image at the coordinate location of 0, 0.

24. Go to File > Save to save your file.

25. Go to Run > Run Module to test your game.



26. You should see a game window load using the SkyBackground image.

27. Close out of the game window.

28. Click at the end of Line 5.

```
1 #Basic Pygame Structure
2
3 import pygame
4
5 #Define Classes (sprites) here
6
7 pygame.init()
```

29. Press ENTER.

30. We will now create our first object class: the Balloon Class. Type the code that you see on Lines 6 – 36. This is a large chunk of code, but it is pretty repetitive.

```
5  #Define Classes (sprites) here
6  class Balloon(pygame.sprite.Sprite):
7
8      def __init__(self, x, y, direction, balloonType):
9          pygame.sprite.Sprite.__init__(self)
10
11         self.Direction = direction
12         self.BalloonType = balloonType
13
14         if balloonType == 1:
15             balloonImage = pygame.image.load("RedBalloon.png")
16             self.Speed = 3
17             self.Score = 5
18         if balloonType == 2:
19             balloonImage = pygame.image.load("YellowBalloon.png")
20             self.Speed = 7
21             self.Score = 5
22         if balloonType == 3:
23             balloonImage = pygame.image.load("GreenBalloon.png")
24             self.Speed = 5
25             self.Score = 10
26         if balloonType == 4:
27             balloonImage = pygame.image.load("BlueBalloon.png")
28             self.Speed = 10
29             self.Score = 0
30
31         self.image = pygame.Surface([26,50])
32         self.image.set_colorkey(black)
33         self.image.blit(balloonImage,(0,0))
34         self.rect = self.image.get_rect()
35         self.rect.x = x
36         self.rect.y = y
37
```

On Line 6, we declare a new class called Balloon. The class names always start with a capital letter. The Balloon class we created will inherit the attributes of a Pygame sprite object using the Sprite base class (we specify this on Line 9).

Line 7 is blank.

On Line 8, we use the method def __init__ to create a new instance of the Balloon class. In other words, to create a new object that is part of the Balloon class. When we call the def __init__ method, we will give it the values of the x and y coordinate location we want the object to be generated at, the direction we want the object to be moving (left or right), and the balloonType for the object (type 1, 2, 3, or 4). The balloonType variable will also be used to control the color of the balloon that is generated.

As mentioned before, Line 9 states that the Balloon class objects will inherit the attributes of the Pygame Sprite base class whenever they are created.

Line 10 is blank.

Line 11 creates a class property called Direction and assigns it the direction value that was passed to that particular class object during the __init__ method.

Line 12 does the same thing for the BalloonType class property. It creates the BalloonType class property and assigns it the balloonType value that was passed to that particular class object during the __init__ method.

Line 13 is blank.

Lines 14 – 29 create the class object settings for each different type of Balloon class object. Each different type of Balloon class object (balloonType 1, 2, 3, and 4) is assigned a different image, speed, and score value. Depending on the type of class object the game creates, the balloons generated will eithe rbe red, yellow, green, or blue and will have a different image, speed, and score value. *Note: We have not set the game up to generate the balloon class objects yet. Eventually, we will program the game to generate a random balloon object (either balloonType 1, 2, 3, or 4) for the user.*

Line 30 is blank.

Line 31 creates a new surface for the balloon class object and its image to generate on.

Line 32 uses the Colorkey() method to ensure that the transparent areas of the balloon png image files are actually transparent. When blitting the balloon surface into the game window, the pixels that have the same color as the colorkey will be transparent.

Line 33 will copy the chosen balloon's image onto the new surface created in Line 31.

Pygame uses Rect objects to store and manipulate rectangular areas of the surface and game window. Line 34 creates a new rect object from the balloon image we generated on Lines 31 – 33. This will allow the program to move and manipulate the balloon object by changing its rect position.

Lines 35 and 36 uses the new rect area created on Line 34 to manipulate the position of the balloon object by changing the rect's x and y coordinates. The x and y coordinates of the rect object are set to the x and y coordinates that were passed to that particular class object during the __init__ method.

31. Click at the end of Line 59 (after the “#Update sprites here” comment).

```
55      for event in pygame.event.get():
56          if event.type == pygame.QUIT:
57              done = True
58
59      #Update sprites here
60
61
62      screen.blit(background_image, [0,0])
63      pygame.display.flip()
64      clock.tick(20)
```

32. Press ENTER.

33. Type the code that you see on Lines 60 – 62 of the screenshot below. Ensure that your indentation matches what is shown in the screenshot below.

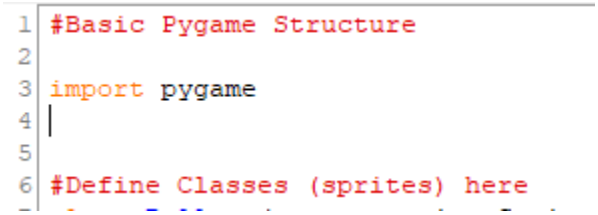
```
58
59      #Update sprites here
60      yCoord = random.randint(50, 350)
61      balloonType = random.randint(1, 4)
62      balloon = Balloon(0, yCoord, "right", balloonType)
63
```

Line 60 creates a variable called yCoord. This variable’s value will be a random integer between 50 and 350. This will generate a random y-coordinate position for each balloon. We have not imported the random pygame module yet. We will do that in the next set of steps. However, once we import the random pygame module, we can access the randint pygame function, which generates a random integer between the two numbers we give to it (in this case, between 50 and 350).

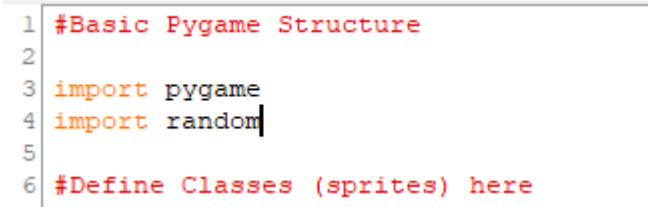
Line 61 creates a variable called balloonType. This variable’s value is also set to be a random integer between the numbers 1 and 4 (representing the fact that we have four different balloon types that we defined in our Balloon class – balloonType 1, balloonType 2, balloonType 3, and balloonType 4).

Line 62 creates a new class object instance called balloon. This instance calls the Balloon class object using the x coordinate of 0 (which will be all the way at the left edge of the game window), the y coordinate stored in the yCoord variable, the “right” direction to ensure that the balloon will begin by moving to the right, and the balloonType that is stored in the balloonType variable.

34. Click at the end of Line 3 in your code and press ENTER.



35. Import the random module into your game by typing the code shown on Line 4 of the screenshot below.



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

37. If you were to run the program now, you should see that no balloon appear yet. This is because we have not yet drawn the balloon objects on the display surface. This must happen once during each loop of the program.

    In any game, there are always events that occur multiple times. In this game, these events include: checking to see if the user has clicked on each balloon, in turn; Checking each balloon to see if it has touched the edge of the window and must change direction; Moving every balloon to the left or right; Checking to see if the user has clicked on any of the blue balloons in the game, which would end it; Drawing each balloon in turn on the window's surface.

    To simplify the handling of these repetitive events, Pygame allows objects to be grouped together. These groups can then be used to handle the repetitive events. Objects in Pygame programs can be added/removed from groups or copied from one group to another.

    In this game, we will create three groups. Each balloon object will be a member of two different groups.

    - Each new instance of a balloon will be added to either the "otherBalloons" group or the "blueBalloons" group. The "otherBalloons" group of objects will consist all ALL balloons that are NOT blue. The "blueBalloons" group of objects will consist of only the blue balloons.
    - In addition, each object will be added to the "allBalloons" group.

    Click at the end of Line 49 and press ENTER twice, then type the code that you see on Lines 51 – 54 of the screenshot below.

```
49 white = (255, 255, 255)
50
51 otherBalloons = pygame.sprite.Group()
52 blueBalloons = pygame.sprite.Group()
53 allBalloons = pygame.sprite.Group()
54
55 #Define additional Functions and Procedures here
```

    Lines 51 – 54 establish the three sprite groups I discussed above.

38. Click at the end of Line 67, then press ENTER.

```
64      #Update sprites here
65      yCoord = random.randint(50, 350)
66      balloonType = random.randint(1, 4)
67      balloon = Balloon(0, yCoord, "right", balloonType)
68      
69
70      screen.blit(background_image, [0,0])
```

39. Type the code that you see on Lines 68 – 72 of the screenshot below.

```
64      #Update sprites here
65      yCoord = random.randint(50, 350)
66      balloonType = random.randint(1, 4)
67      balloon = Balloon(0, yCoord, "right", balloonType)
68      if balloonType >=1 and balloonType <=3:
69          otherBalloons.add(balloon)
70      else:
71          blueBalloons.add(balloon)
72      allBalloons.add(balloon)
73
```

Lines 68 - 69 check to see if the balloonType of the object created is equal to the number 1, 2, or 3. If it is, it adds that sprite to the otherBalloons group (indicating that the sprite is either a red, yellow, or green balloon).

If the balloonType object is NOT 1, 2, or 3 (meaning that it is 4, the last balloonType category number we have), then the balloon sprite object is added to the blueBalloons group (Lines 70 and 71).

All balloons generated are added to the allBalloons group using the code on Line 72.

40. Click at the end of Line 74.

```
72      allBalloons.add(balloon)
73
74      screen.blit(background_image, [0,0])
75      pygame.display.flip()
76      clock.tick(20)
77
78  pygame.quit()
```

41. Press ENTER, then type the code that you see on Lines 75 and 76 of the screenshot below.

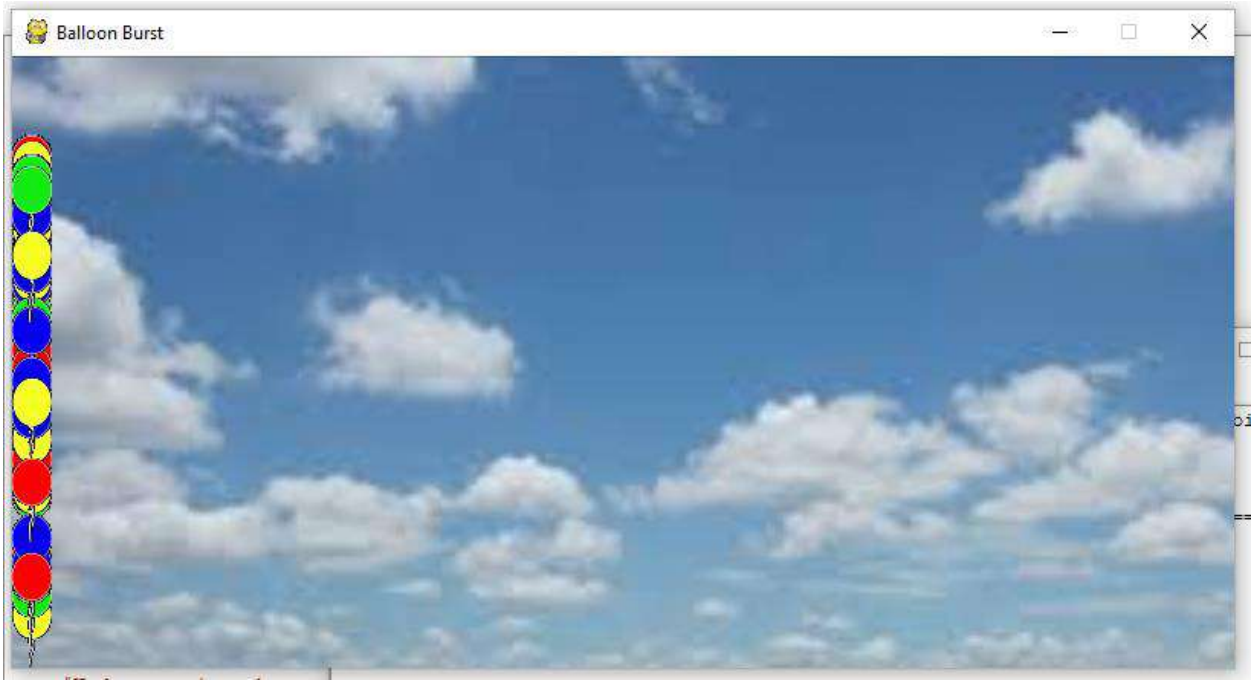
```
74      screen.blit(background_image, [0,0])
75      allBalloons.draw(screen)
76
77      pygame.display.flip()
78      clock.tick(20)
79
80  pygame.quit()
```

    Line 75 draws the balloons in the allBalloons group onto the screen surface.

    Line 76 is a blank link.

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

43. Go to Run > Run Module to preview your game. You should notice that the game now draws balloons at the left side of the game window. We have not programmed the balloons to move yet. Your mouse cursor should also disappear.



44. The last line of the main program loop (Line 78 – the “clock.tick(20)” line) ensures that the Balloon Burst game is running at 20 frames per second. As instances of our Balloon objects are being created inside the main loop of the game, this means that our balloons are being created at a rate of 20 balloons per second.

    If our game is to be playable, we need to slow the balloon generation down and also introduce some random element as to how fast the balloons are being created. To do this, we will use the same close that we use to control the frame rate to track and calculate random time intervals between balloon objects generating.

    However, first, we need to create a variable that keeps track of that random time interval that is calculated.

    Click at the end of Line 53, press enter twice, and then type the code you see on Line 55 of the screenshot below.

```
53 allBalloons = pygame.sprite.Group()
54
55 timeTillNextBalloon = random.randint(1000, 2000)
56
57 #Define additional Functions and Procedures here
```

    Line 55 creates a new variable called timeTillNextBalloon. Its value is set to a random integer between 1000 and 2000. When we pair this integer with our clock (which measures time in milliseconds), this variable will represent how many milliseconds will pass between balloons, and that number will be randomized.

45. Click at the end of Line 66 and press ENTER.

```
64             done = True
65
66     #Update sprites here
67
68     yCoord = random.randint(50, 350)
69     balloonType = random.randint(1, 4)
```

46. Type the two lines of code you see in the screenshot below.

```
65
66      #Update sprites here
67      if pygame.time.get_ticks() > timeTillNextBalloon:
68          timeTillNextBalloon += random.randint(300, 2500)
69      yCoord = random.randint(50, 350)
```

Line 67 checks to see if the time passed is greater than the number of milliseconds stored in the timeTillNextBalloon variable.

If it is, the timeTillNextBalloon variable will be increased by another random value generated using the randint function. This random value will be between 300 and 2500 milliseconds.

47. Modify the code on Lines 69 – 76 to match the indentation shown below.

```
65
66      #Update sprites here
67      if pygame.time.get_ticks() > timeTillNextBalloon:
68          timeTillNextBalloon += random.randint(300, 2500)
69          yCoord = random.randint(50, 350)
70          balloonType = random.randint(1, 4)
71          balloon = Balloon(0, yCoord, "right", balloonType)
72          if balloonType >=1 and balloonType <=3:
73              otherBalloons.add(balloon)
74          else:
75              blueBalloons.add(balloon)
76          allBalloons.add(balloon)
77
78      screen.blit(background_image, [0,0])
79      allBalloons.draw(screen)
80
81      pygame.display.flip()
82      clock.tick(20)
```

Placing the rest of our balloon generation code inside the “if” statement that determines if the appropriate amount of time has passed will ensure the code is only executed after the current time (pygame.time.get.ticks() method) s greater than the time stored in the timeTillNextBalloon variable.

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

49. Go to Run > Run Module to preview your game. You should now see that the balloons generate at a slower, but random, pace

50. Click at the end of Line 37 and press ENTER twice.

51. Type the code that you see on Lines 39 – 44 of the screenshot below. Ensure your indentation matches what is shown in the screenshot.

```
38
39      def moveBalloons(self):
40
41          if self.Direction =="right":
42              self.rect.x += self.Speed
43          if self.Direction =="left":
44              self.rect.x -= self.Speed
45
46
47  pygame.init()
```

Line 39 creates another method inside the Balloon object class. This method is called moveBalloons.

Line 40 is a blank line.

Line 41 creates an if statement to check to see if the value of the balloon's Direction property is "right." If it is, Line 42 will add the object's speed property to the current x-coordinate position of the object's rect area. For example, if the object rect's x-coordinate is 10 and its speed is 5, then the rect's x coordinate position will be changed to 15 (its current value of 10 + the speed's value of 5).

Line 43 checks to see if the balloon's direction property is "left." If it is, Line 44 will subtract the object's speed property from the current x-coordinate position of the object's rect area. Remember, if the balloon is moving to the left in the game window, it's x-coordinate should be decreasing, which is why you would subtract the speed instead of adding it like you did on Line 42.

52. Click at the end of Line 83, then press ENTER twice.

```
83          allBalloons.add(balloon)
84
85
86
87      screen.blit(background_image, [0,0])
88      allBalloons.draw(screen)
```

53. Type the comment and code you see on Lines 85 – 87 of the screenshot below.

```
83          allBalloons.add(balloon)
84
85      # Move each balloon in the allBalloons group
86      for balloon in (allBalloons.sprites()):
87          balloon.moveBalloons()
88
89      screen.blit(background_image, [0,0])
```

Line 85 contains a comment.

Line 86 creates a “for” loop that will run on all balloon objects in the allBalloons group.

Line 87 will run the moveBalloons method from the balloon class. Since all balloons are put into the allBalloons group, this method will update the position of all the balloon objects using the moveBalloons method.

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

55. Go to Run > Run Module to preview your game. You should now see that balloons are generated at a random speed and move from left to right across the screen. However, they do not stop at the right edge of the screen to change directions. We will fix this next.

56. Click at the end of Line 83 and press ENTER twice.

57. Type the code that you see on Lines 85 – 90 of the screenshot below.

```
83          allBalloons.add(balloon)
84
85      #Check if balloon sprites have reached edge of screen
86      for balloon in (allBalloons.sprites()):
87          if balloon.rect.x <0:
88              balloon.Direction = "right"
89          if balloon.rect.x > 774:
90              balloon.Direction = "left"
91
92      # Move each balloon in the allBalloons group
```

Line 85 contains another comment.

Line 86 will begin a “for” loop that will run on every balloon in the allBalloon sprite group.

Line 87 will check to see if the balloon rect’s x-coordinate position is less than 0. If it is, Line 88 will set the balloon object’s direction property to “right.”

Line 89 will check to see if the balloon rect’s x-coordinate position is greater than 774. If it is, Line 90 will set the balloon object’s direction property to “left.” Notice we used the

number 774 instead of 800, even though the game window is 800 pixels wide. This is because the balloon graphics are 26 pixels wide. Since the rect's x and y coordinate positions are based off of the graphic's top left hand edge, we have to allow for the width of the graphic in our code.

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

59. Go to Run > Run Module to preview your game. You should notice that when the balloons reach the right edge of the game window, they change direction and start moving to the left.

This takes care of our balloon generation. Now, we need to give the player a dart to pop the balloons. This involves creating a dart, blitting the dart on the screen, and setting balloon behavior for whenever the balloon is popped with a dart.

60. Click at the end of Line 45, then press ENTER twice. You may need to backspace your cursor once or twice until it is aligned all the way at the left margin, as shown in the screenshot below.

```
39      def moveBalloons(self):
40
41          if self.Direction =="right":
42              self.rect.x += self.Speed
43          if self.Direction =="left":
44              self.rect.x -= self.Speed
45
46
```

61. Type the code that you see on Lines 46 – 56 of the screenshot below.

```
46  class Dart(pygame.sprite.Sprite):
47
48      def __init__(self):
49          pygame.sprite.Sprite.__init__(self)
50          dartImage = pygame.image.load("Dart.png")
51          self.image = pygame.Surface([24,19])
52          self.image.set_colorkey(black)
53          self.image.blit(dartImage, (0, 0))
54          self.rect = self.image.get_rect()
55          self.rect.x = 388
56          self.rect.y = 190
```

Line 46 declares anew class named Dart. The Dart class will inherit the properties of the Sprite Pygame base class, meaning that objects in the Dart class will be sprites.

Line 47 is blank.

Line 48 initiates a new instance of the Sprite object.

In Line 49, we ensure that all new objects under this class inherit the properties of the Sprite Pygame base class.

Line 50 creates a new variable called dartImage. That variable is set to be the Dart.png image that you were given.

Line 51 creates a new surface for the dart class object and its image to generate on.

Line 52 uses the Colorkey() method to ensure that the transparent areas of the dart image files are actually transparent. When blitting the dart surface into the game window, the pixels that have the same color as the colorkey will be transparent.

Line 53 will copy the dart's image onto the new surface created on Line 51.

Pygame uses Rect objects to store and manipulate rectangular areas of the surface and game window. Line 54 creates a new rect object from the dart image we generated on Lines 51 – 53. This will allow the program to move and manipulate the dart object by changing its rect position.

Lines 55 and 56 uses the new rect area created on Line 54 to manipulate the position of the dart object by changing the rect's x and y coordinates. The x and y coordinates of the rect object are set to the x and y coordinates that were passed to that particular class object during the __init__ method.

62. Click at the end of Line 74, as shown in the screenshot below.

```
73
74 timeTillNextBalloon = random.randint(1000, 2000)
75
76 #Define additional Functions and Procedures here
```

63. Press ENTER twice.

64. Type the code that you see on Lines 76 – 78 of the screenshot below.

```
74 timeTillNextBalloon = random.randint(1000, 2000)
75
76 dart = Dart()
77 darts = pygame.sprite.Group()
78 darts.add(dart)
79
80 #Define additional Functions and Procedures here
```

Line 76 creates a new dart object using the Dart() class. This object will be referred to as dart by the program.

Line 77 creates a new sprite group called darts.

Line 78 will add the dart object that was generated by the program to the darts sprite group.

65. Click at the end of Line 113, as shown in the screenshot below.

```
112     screen.blit(background_image, [0,0])
113     allBalloons.draw(screen)
```

66. Press ENTER.

67. Type the code that you see on Line 114 of the screenshot below.

```
112     screen.blit(background_image, [0,0])
113     allBalloons.draw(screen)
114     darts.draw(screen)
```

Line 114 will draw the dart objects in the darts group onto the screen surface. Remember, we cannot see the dart unless we use the draw command. We are putting the command at the bottom of the program loop so that all positions and groups are updated before we draw the objects on the screen. Also, we are putting the dart draw command after the balloon draw command so that we place the dart over or in front of the balloon images in the game window.

68. Click at the end of Line 74, as shown in the screenshot below.

```
73
74 timeTillNextBalloon = random.randint(1000, 2000)
75
```

69. Press ENTER.

70. Type the code that you see on Line 75 of the screenshot below.

```
74 timeTillNextBalloon = random.randint(1000, 2000)
75 mousePosition = [0]*2
76
77 dart = Dart()
```

Line 75 creates a mousePosition array and sets its value to be 0, 0. Arrays function like lists. They can contain a number of different objects of different data types. They can hold more than one value at a time. Each list in an array is surrounded by square brackets ([ ]). When you see an array value multiplied by two, it essentially creates two items in that list with that value.

For example, the [0]*2 array could ALSO be written as [0, 0]. The “*2” portion of the code tells the program to include two items with the value of 0 in the array/list.

71. Click at the end of Line 88, as shown in the screenshot below.

```
83 #------------- Main Program Loop -------------
84 while done == False:
85
86     for event in pygame.event.get():
87         if event.type == pygame.QUIT:
88             done = True
```

72. Press ENTER twice.

73. Type the code you see on Lines 89 – 91 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot below.

```
85      for event in pygame.event.get():
86          if event.type == pygame.QUIT:
87              done = True
88
89          if event.type == pygame.MOUSEMOTION:
90              mousePosition[:] = list (event.pos)
91              dart.moveDart(mousePosition)
```

    Line 90 checks to see of the mouse is moving. If it is, the program loop executes the behaviors in the code on Lines 90 – 91.

    Line 91 copies the x and y coordinates of the mouse from the event queue into the array of two values called mousePosition. Essentially, Line 90 records the x and y coordinate positions of your mouse cursor.

    Line 92 calls the moveDart() method, which we will program in the next step. The moveDart method will use the mousePosition coordinates that you saved on Line 91.

74. Click at the end of Line 56, as shown in the screenshot below.

```
48      def __init__(self):
49          pygame.sprite.Sprite.__init__(self)
50          dartImage = pygame.image.load("Dart.png")
51          self.image = pygame.Surface([24,19])
52          self.image.set_colorkey(black)
53          self.image.blit(dartImage, (0, 0))
54          self.rect = self.image.get_rect()
55          self.rect.x = 388
56          self.rect.y = 190
57
58
59  pygame.init()
```

75. Press ENTER twice.

76. Press ENTER twice.

77. Type the code you see on Lines 58 – 60 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot.

```
46 class Dart(pygame.sprite.Sprite):
47
48     def __init__(self):
49         pygame.sprite.Sprite.__init__(self)
50         dartImage = pygame.image.load("Dart.png")
51         self.image = pygame.Surface([24,19])
52         self.image.set_colorkey(black)
53         self.image.blit(dartImage, (0, 0))
54         self.rect = self.image.get_rect()
55         self.rect.x = 388
56         self.rect.y = 190
57
58     def moveDart(self, mousePosition):
59         self.rect.x = mousePosition[0]
60         self.rect.y = mousePosition[1]
61
62
63 pygame.init()
```

Line 58 creates a new class method called moveDart. The moveDart method will use the mousePosition it is passed by the program to run.

Line 59 updates the x rect position of the object to the first mousePosition coordinate in the list. Python starts counting at the number 0, the first list item in any list will have an index value of 0, the second list item in any list will have an index value of 1, and so on. Since you are wanting to change the x position of the dart object to the first coordinate in the mousePosition list, you will reference the coordinate with the index value of 0.

Line 60 updates the y rect position of the object to the second mousePosition coordinate in the list. Python starts counting at the number 0, the first list item in any list will have an index value of 0, the second list item in any list will have an index value of 1, and so on. Since you are wanting to change the y position of the dart object to the second coordinate in the mousePosition list, you will reference the coordinate with the index value of 1.

78. Click at the end of Line 79, as shown in the screenshot below.

```
76 allBalloons = pygame.sprite.Group()
77
78 timeTillNextBalloon = random.randint(1000, 2000)
79 mousePosition = [0]*2
80
81 dart = Dart()
```

79. Press ENTER.

80. Type the code shown on Line 80 of the screenshot below.

```
76 allBalloons = pygame.sprite.Group()
77
78 timeTillNextBalloon = random.randint(1000, 2000)
79 mousePosition = [0]*2
80 score = 0
81
82 dart = Dart()
```

Line 80 creates a variable called score and sets its initial value to 0. As the player score points, this value will change.

81. Click at the end of Line 97, as shown in the screenshot below.

```
88 #------------- Main Program Loop -------------
89 while done == False:
90
91     for event in pygame.event.get():
92         if event.type == pygame.QUIT:
93             done = True
94
95         if event.type == pygame.MOUSEMOTION:
96             mousePosition[:] = list (event.pos)
97             dart.moveDart(mousePosition)
98
99     #Update sprites here
```

82. Press ENTER twice.

83. Type the code that you see on Lines 99 – 106 of the screenshot below. Ensure your indentation matches the indentation shown in the screenshot.

```
88  #------------- Main Program Loop -------------
89  while done == False:
90
91      for event in pygame.event.get():
92          if event.type == pygame.QUIT:
93              done = True
94
95          if event.type == pygame.MOUSEMOTION:
96              mousePosition[:] = list (event.pos)
97              dart.moveDart(mousePosition)
98
99          if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
100             hitBalloons = pygame.sprite.groupcollide(blueBalloons, darts, False, False)
101             if len(hitBalloons) > 0:
102                 done = True
103             hitBalloons = pygame.sprite.groupcollide(otherBalloons, darts, False, False)
104             for balloon in (hitBalloons):
105                 score += balloon.Score
106             pygame.sprite.spritecollide(dart, allBalloons, True, collided = None)
107
108     #Update sprites here
```

Line 99 checks to see if the mouse button is being clicked or held down. The event.button == 1 checks to see that the mouse button is down (or it has been clicked). If the button has been clicked, the code will proceed with the behaviors programmed on Lines 100 – 106.

Line 100 will create the hitBalloons variable using the Groupcollide() method. This will check to see if the items in the blueBallons group and the darts group are colliding with each other at the time the mouse button was clicked. The two Boolean variables (False and False) on Line 100 will delete (kill) any object in either group that has collided with objects of the other group in the list (either a blueBalloon or a dart, respectively).

For the Groupcollide() method, it is important to remember that any item in the first group listed (in this case, the blueBalloons group) will be part of the returned list by that method. So, while both the dart and the blueBalloons will be deleted, only the blueBalloon objects that have collided with the dart will be returned. Any objects in the blueBalloon group that have been collided with will be added to the hitBalloons group.

Line 101 checks to see if the length of the hitBalloons group is greater than 0. If it is, the “done” variable will be changed to True, meaning that the game will be over. Remember, according to Line 89, the main program loop will only run while the done variable is equal to False. We want the game to end if the player hits a blue balloon, so we need to change the done variable to True if there are any blueBalloons in the list (or any other words, if there are more than zero items on the hitBalloons list after check for collisions between the blueBalloon and dart objects).

If there are no blueBalloons on the hitBalloons list, the code will resume running on Line 103. This will check to see if any of the other balloons in the otherBalloons list (the red, green, or yellow balloons) have collided with the dart object. If they have, the dart and

the other balloons will be deleted and the other balloons will be added to the hitBalloons list.

Line 104 will use the hitBalloons listing (the list of red, yellow, and green balloons that have been hit by the dart) to increase the score by whatever that balloon's score value is (Line 105).

Line 106, the last line in this code, will reset the collision settings for the dart and the balloons in the allBalloons group. This will prepare the loop for the next frame so that the game can properly sense which items have been collided with during each frame.

84. Click at the end of Line 72, as shown in the screenshot below.

```
65 screen = pygame.display.set_mode([800,400])
66 pygame.display.set_caption("Balloon Burst")
67 background_image = pygame.image.load("SkyBackground.png").convert()
68 pygame.mouse.set_visible(False)
69 done = False
70 clock = pygame.time.Clock()
71 black = (0, 0, 0)
72 white = (255, 255, 255)
73
74 otherBalloons = pygame.sprite.Group()
```

85. Press ENTER.

86. Type the code you see on Line 73 of the screenshot below.

```
71 black = (0, 0, 0)
72 white = (255, 255, 255)
73 font = pygame.font.Font(None, 36)
74
75 otherBalloons = pygame.sprite.Group()
```

Line 73 creates a font variable. This variable's value is set to be a new font object. The font will not have any assigned font face, meaning that it will use the system defaults to display the font. It will be size 36 pt text.

87. Click at the end of Line 134, as shown in the screenshot below.

```
132     screen.blit(background_image, [0,0])
133     allBalloons.draw(screen)
134     darts.draw(screen)
135
136     pygame.display.flip()
137     clock.tick(20)
```

88. Press ENTER.

89. Type the text that you see on Lines 135 – 137 of the screenshot below.

```
132     screen.blit(background_image, [0,0])
133     allBalloons.draw(screen)
134     darts.draw(screen)
135     #Add the score to the screen
136     textImg = font.render(str(score), 1, white)
137     screen.blit(textImg, (10, 10))
138
139     pygame.display.flip()
140     clock.tick(20)
```

Line 135 contains a comment.

To display text, Pygame creates an image of the test which is then blitted on to the screen display at the given coordinates. Line 136 creates a textImage variable that is set to be the rendered font image of the score (since the score is an integer, it must be converted to a string value in order to be displayed as text). The number 1 on Line 136 is equal to the Boolean value of True, meaning that antialiasing will be turned on. This will ensure the numbers appear smooth in the rendered image. The font color is also set to white.

Line 137 blits the textImg onto the screen surface at the coordinates of 10 and 10.

90. Click at the end of Line 73, as shown in the screenshot below.

```
65 screen = pygame.display.set_mode([800,400])
66 pygame.display.set_caption("Balloon Burst")
67 background_image = pygame.image.load("SkyBackground.png").convert()
68 pygame.mouse.set_visible(False)
69 done = False
70 clock = pygame.time.Clock()
71 black = (0, 0, 0)
72 white = (255, 255, 255)
73 font = pygame.font.Font(None, 36)
74
75 otherBalloons = pygame.sprite.Group()
```

91. Press ENTER twice.

92. Type the code you see on Line 75 of the screenshot below.

```
73 font = pygame.font.Font(None, 36)
74
75 popSound = pygame.mixer.Sound("pop.wav")
76
77 otherBalloons = pygame.sprite.Group()
```

The pygame.mixer.Sound method on Line 75 creates a new sound object in the game using the pop.wav sound file you have been given. In this case, the sound object is named popSound. Note: The sound file must be saved in the same folder as all of your images and your Pygame code in order for it to work in the game.

93. Click at the end of Line 108, as shown in the screenshot below.

```
92 while done == False:
93
94     for event in pygame.event.get():
95         if event.type == pygame.QUIT:
96             done = True
97
98         if event.type == pygame.MOUSEMOTION:
99             mousePosition[:] = list (event.pos)
100            dart.moveDart(mousePosition)
101
102        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
103            hitBalloons = pygame.sprite.groupcollide(blueBalloons, darts, False, False)
104            if len(hitBalloons) > 0:
105                done = True
106            hitBalloons = pygame.sprite.groupcollide(otherBalloons, darts, False, False)
107            for balloon in (hitBalloons):
108                score += balloon.Score
109            pygame.sprite.spritecollide(dart, allBalloons, True, collided = None)
110
111    #Update sprites here
```

94. Press ENTER.

95. Type the code that you see on Line 109 of the screenshot below.

```
94      for event in pygame.event.get():
95          if event.type == pygame.QUIT:
96              done = True
97
98          if event.type == pygame.MOUSEMOTION:
99              mousePosition[:] = list (event.pos)
100             dart.moveDart(mousePosition)
101
102         if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:
103             hitBalloons = pygame.sprite.groupcollide(blueBalloons, darts, False, False)
104             if len(hitBalloons) > 0:
105                 done = True
106             hitBalloons = pygame.sprite.groupcollide(otherBalloons, darts, False, False)
107             for balloon in (hitBalloons):
108                 score += balloon.Score
109                 popSound.play()
110             pygame.sprite.spritecollide(dart, allBalloons, True, collided = None)
111
```

Line 109 will play the sound stored in the popSound object, which is your pop.wav sound. This sound will play every time a red, yellow, or green balloon is hit and the score is updated.

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

97. Go to Run > Run Module. You should be able to play a complete game now. Your balloons will move to the left or right across the game window. You will have a dart that pops balloons according your mouse position. If you happen to hit a blue balloon, the game will end. If you hit a red, yellow, or green balloon, you will hear a pop sound and see your score increase.

98. Attach your Python file to your assignment on Google Classroom and submit it.