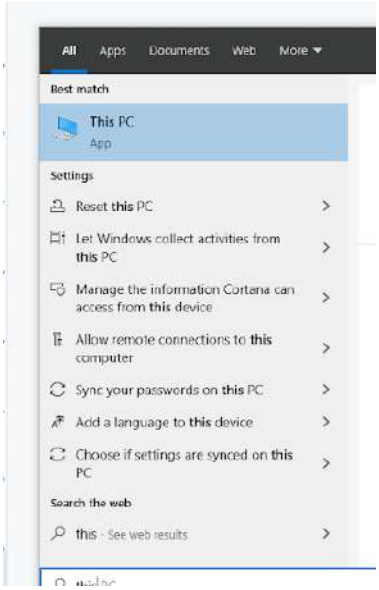
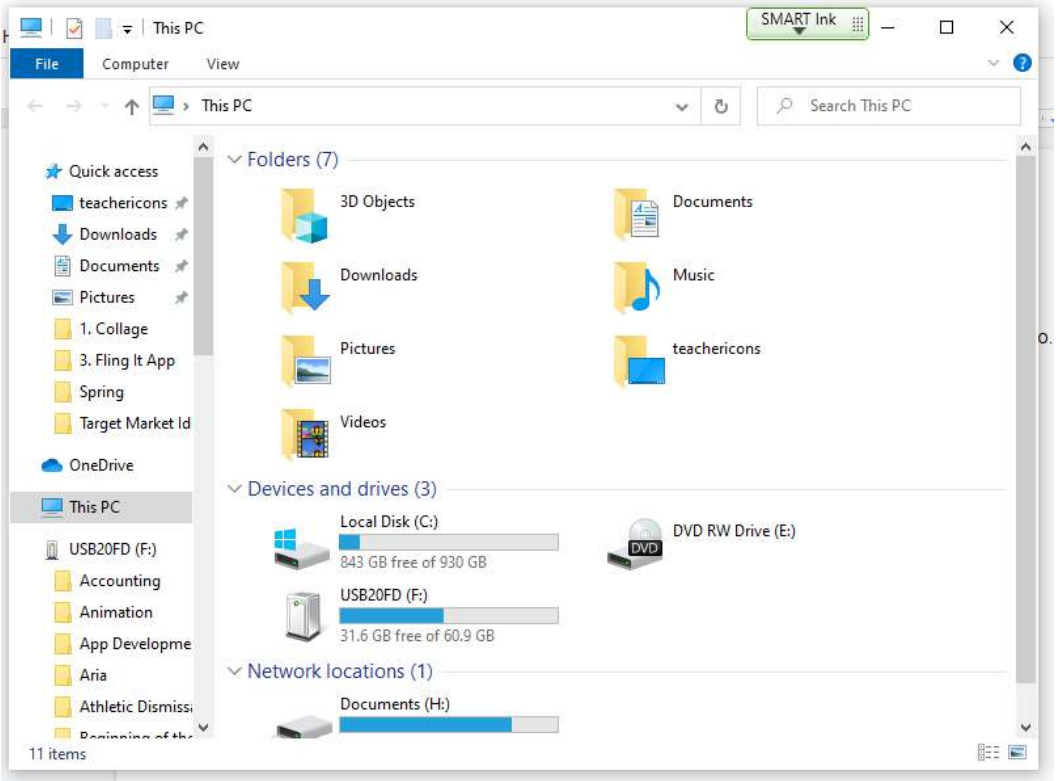
Chapter 1 Practice Directions

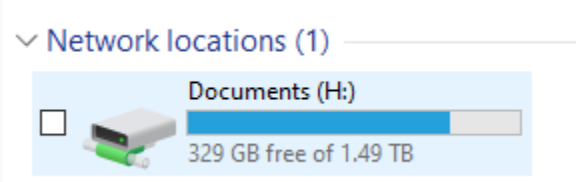
1. First, we need to create a folder in our student directory to save all of our Python files to. Click your Windows button in the bottom left corner and search for “This PC”.



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

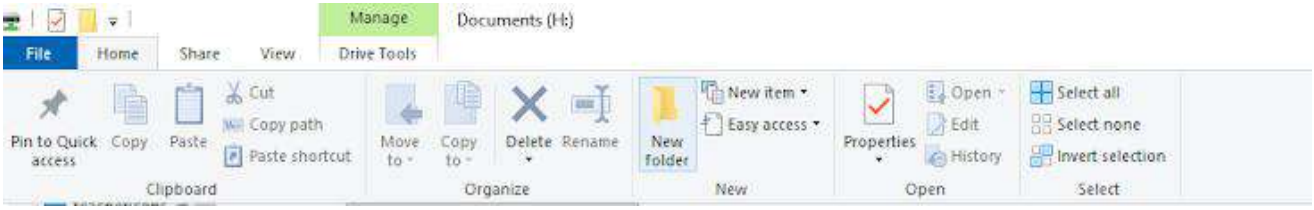


3. Locate your student drive in the menu. Most students have an H: drive. However, some students have a V: drive. You will see your username attached to your student drive. If you don't have a student drive, please let me know. This probably means that we need to have your student drives pulled over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

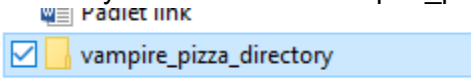


4. Double-click on your H: or V: drive to open it.

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

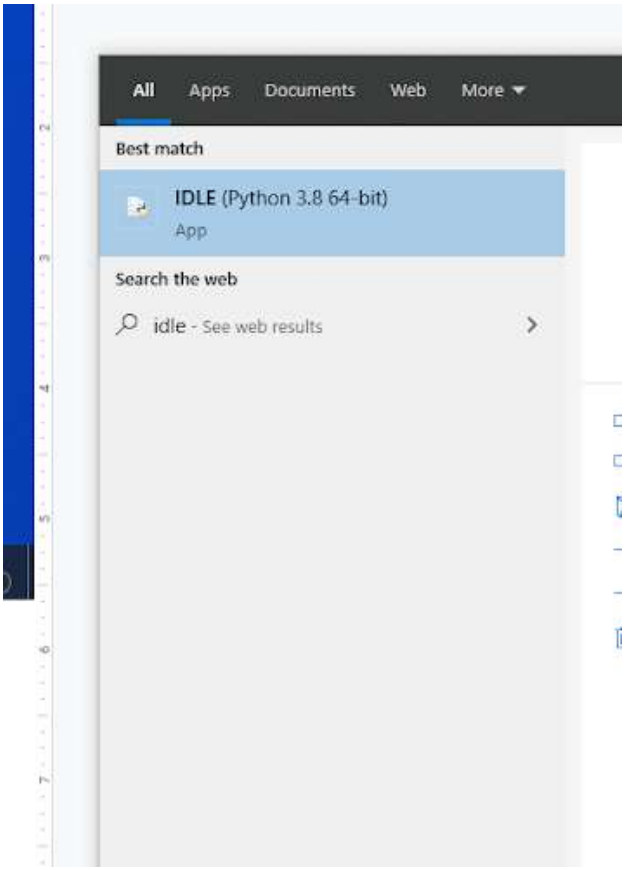


6. Rename your new folder "vampire_pizza_directory" and then press ENTER.



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

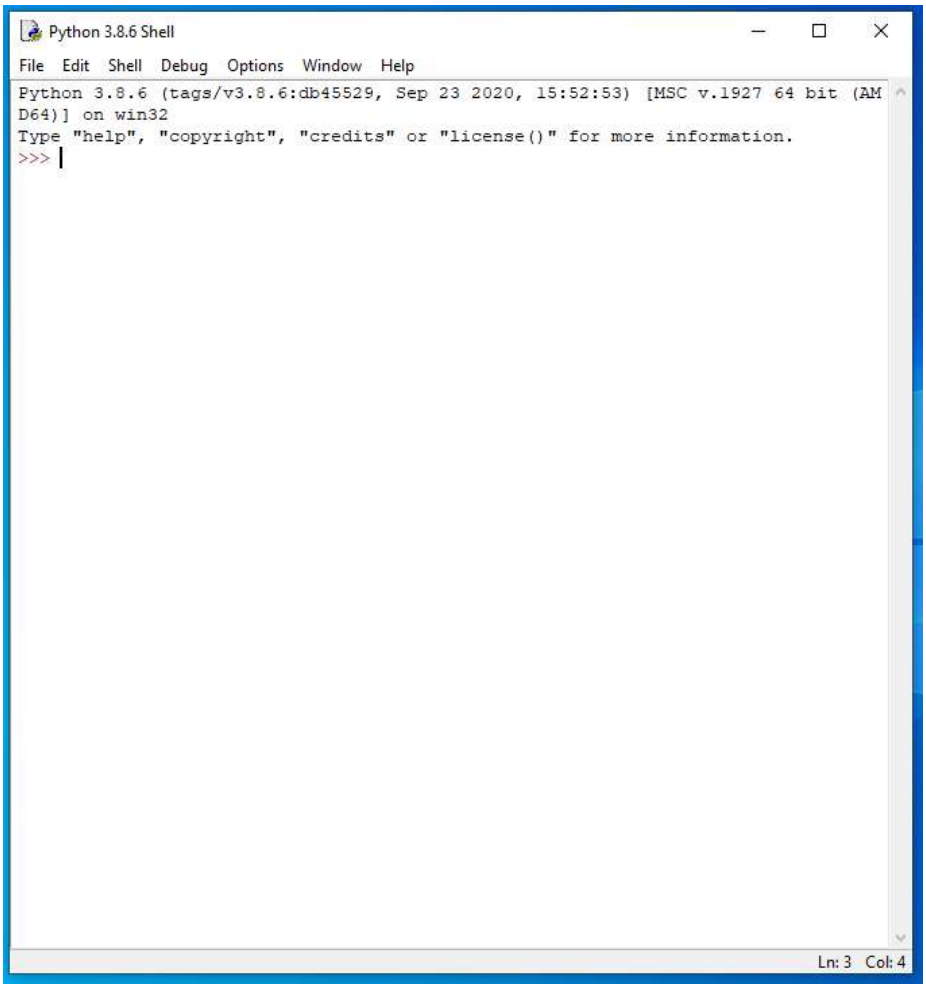
8. Using your Windows button menu, find and launch your IDLE program.



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.

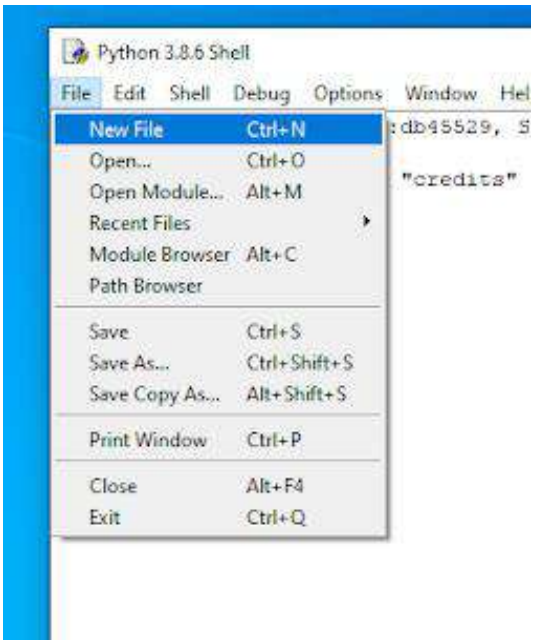
9. Your IDLE window should look something like this once it has launched.:



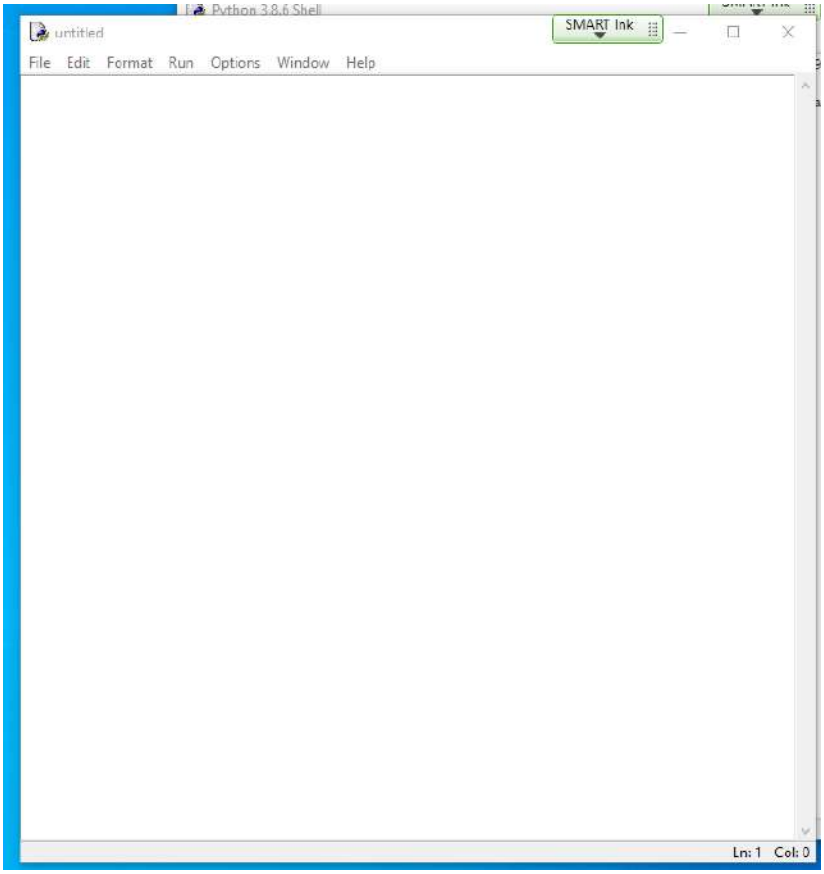
On Startup, IDLE will display the Python Shell, which can be used to give commands to the computer's operating system. Since we are viewing the shell through IDLE and not the actual command prompt window, the commands that we type into the Shell will not communicate directly with our operating system.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

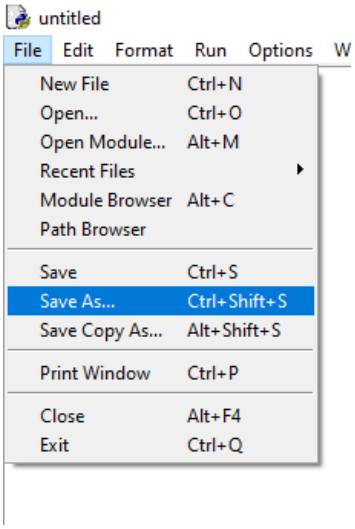
10. Go to File > New File to create a new Python file.



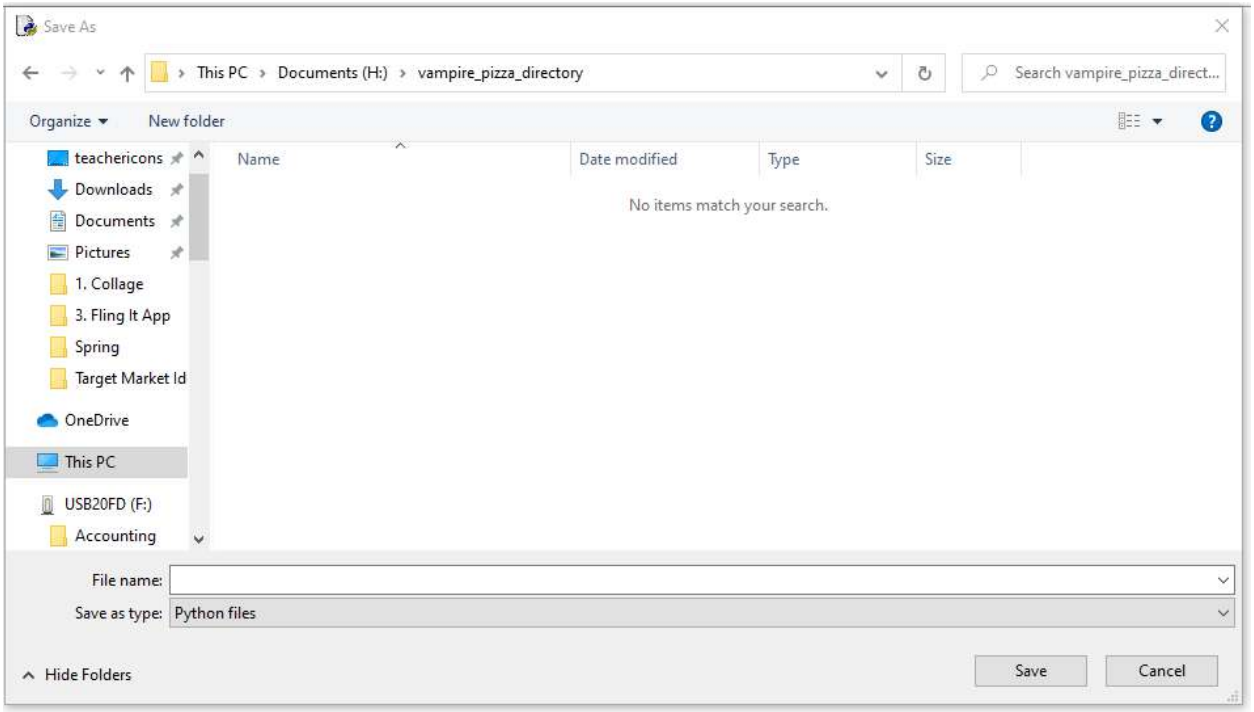
11. A blank Python file will open up. You can tell this apart from the Shell because the title bar will display the file name “untitled” right now because we haven’t saved it yet.



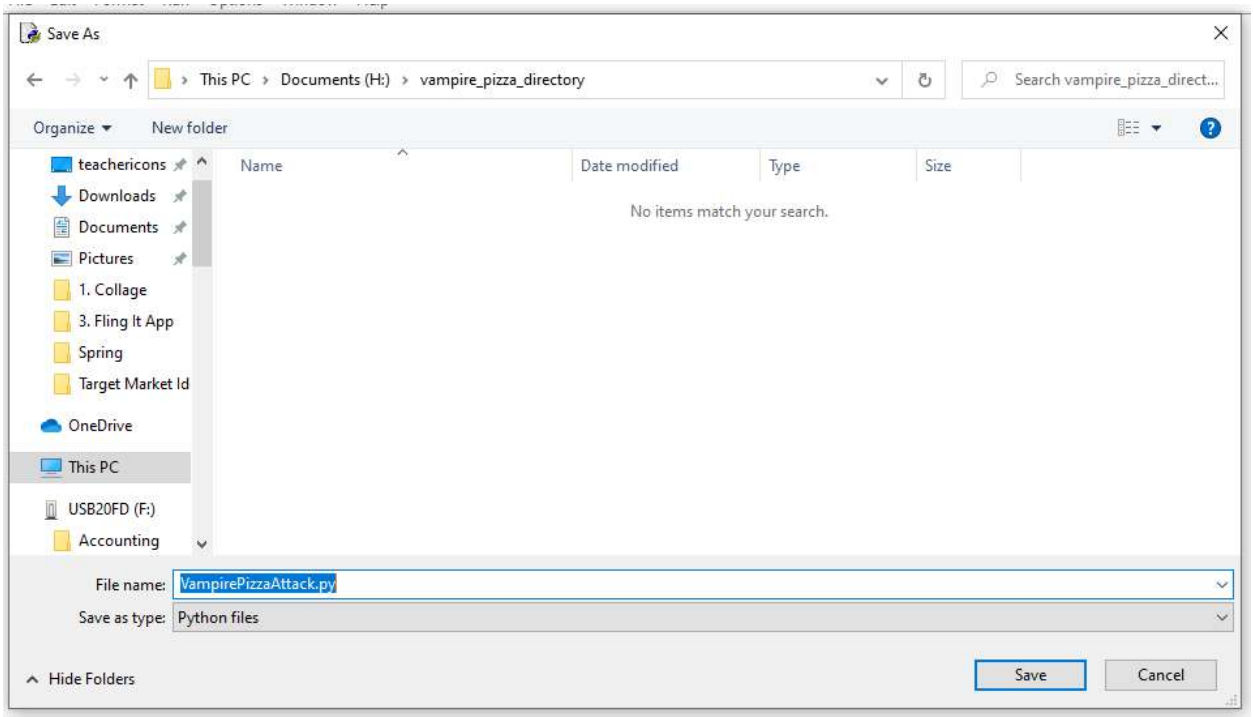
12. Go to File > Save As to save your file.



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

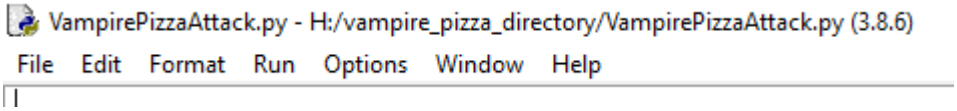


14. Type your new file name in the File Name box. Your new file name is VampirePizzaAttack.py



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 into your Python file:

```
1  #Import Libraries
2  import pygame
3  from pygame import *
4  #Initialize pygame
5  pygame.init()
```

The first and fourth lines contain comments. Anytime you would like to make a note within your code, you should use a comment. Programming comments always begin with the pound (#) sign. Comments are typically used to explain what different sections of code do.

Comments serve two main functions. They allow programmers to quickly find certain sections of code in order to modify them or locate bugs. They also allow programmers to pass off their program to another programmer without much hassle. The new programmers can read the comments left behind by the old programmers to understand what different portions of the code do.

In this case, the first comment indicates that the following two lines of code will import the game libraries that we need for this game. The second comment indicates that the code that follows will initialize the Pygame library.

Line 2 imports the Pygame library. A library is an extra set of commands or methods that is included with Python that doesn't automatically import into every Python file. Libraries were created because Python files were starting to get relatively large and have long load times, and programmers were trying to find creative solutions to minimize the load time for Python programmers. By minimizing the commands that are automatically included in all Python files, load times were improved. Programmers still have the ability to import different libraries into their program to gain access to some of the commands that they used to have access to automatically, but with each library import, load time typically increases. Every library needs to be imported before programmers can access it or use commands included in it. Please note: In Python, libraries are also called packages or modules.

Line 3 indicates that you want to import the entire Pygame library into your game. The asterisk symbol means to import "all", or to import the entire library. Sometimes, if programmers only want to use one or two commands from a library, they will only import those commands instead of the entire library.

Line 5 initializes Pygame. This makes the Pygame library that we imported on Lines 2 and 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.

19. Press ENTER twice.

20. Type the following code shown on Lines 7 – 13 of the screenshot below.

```
6
7   #---------------------------------------------------------------
8   #Define constant variables
9
10  #Define the parameters of the game window
11  WINDOW_WIDTH = 900
12  WINDOW_HEIGHT = 400
13  WINDOW_RES = (WINDOW_WIDTH, WINDOW_HEIGHT)
```

Line 7 contains a comment with a bunch of dashes. Whenever you are building a long program, comments with dashes can serve to separate once section of code from another and make things easier to find. These types of separations aren't necessary, but they do make things easier to find when you have a long document.

Line 8 contains another comment. Eventually, this section will have more constant variables than just the three that are included in this chapter. We are setting this code up for that now so we don't have to go back and edit this later.

Line 9 was left blank just to make things easier to read.

Line 10 contains another comment describing what the constant variables underneath it do.

Line 11 creates the constant variable called WINDOW_WIDTH and initializes its value (sets its initial value) to be equal to 900 pixels. Variables are used to store different pieces of information in the game. Variables are often used when we want to use data over and over again or when we want to use data that will change. Constant variables are variables whose values you DO NOT want to change throughout the course of the game. For this reason, constant variables are usually created in all capital letters. The capital letters are a way of telling other programmers not to mess with the values of that particular variable.

Variables can be named virtually anything you want, but cannot include spaces. However, remember that you have to use the same variable name, with the same capitalization and spelling, throughout the entire program. It is good practice to make sure that whatever you name your variable, your name contains some type of description or indication of what type of data that variable will hold.

So, as a review, the variable on Line 11 is called WINDOW_WIDTH because it contains the pixel width of the game window. It is also in all capital letters to indicate that it is a constant variable with a value that should not be changed.

Lines 12 and 13 create two more constant variables: WINDOW_HEIGHT and WINDOW_RES. The value of the WINDOW_HEIGHT variable is set to 400 pixels. It is also a constant variable that should not be changed. The value of the WINDOW_RES variable is set to be the value of the window's width variable and the window's height variable. This will be important later, so we will refer back to Line 13 soon.

21. Press ENTER twice.

22. Type the code that you see on Lines 15 – 20 of the screenshot below. Please note that I have included Lines 13-14 in the screenshot also, but you have already typed these in previous steps. I just wanted to make sure you knew where in the code you should be working.

```
13  WINDOW_RES = (WINDOW_WIDTH, WINDOW_HEIGHT)
14
15  #---------------------------------------------------------------
16  #Load assets
17
18  #Create window
19  GAME_WINDOW = display.set_mode(WINDOW_RES)
20  display.set_caption('Attack of the Vampire Pizzas!')
```

Line 15 contains another comment separator to separate the next code into another section.

Line 16 contains a comment explaining that we are going to load game assets in this section. We will eventually be loading more assets on this page, so we are setting up the code structure now so that it will make more sense to use in the future.

Line 17 is a blank line.

Line 18 contains a comment explaining that the code below it will create the game window.

Line 19 creates a new constant variable called GAME_WINDOW and sets its value to be the result of the display.set_mode method that uses the WINDOW_RES argument. 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 GAME_WINDOW variable. The window's size (its width and its height) is equal to the value of the WINDOW_RES variable. Remember

you set the WINDOW_RES variable on Line 13 to be equal to the window's height and the window's width variable values.

Line 20 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 "Attack of the Vampire Pizzas!".

23. Press ENTER twice.

24. Type the code that you see on Lines 22 - 27 in the following screenshot:

```
20  display.set_caption('Attack of the Vampire Pizzas!')
21
22  #---------------------------------------------------------------
23  #Start main game loop
24
25  #Game loop
26  game_running = True
27  while game_running:
```

Line 22 contains another comment separator to separate the next code into another section.

Line 23 contains a comment explaining that we are going to start the main game loop in this section of code.

Line 24 is blank.

Line 25 indicates that the code that follows is the game loop code, which is what the game will loop through until it is stopped.

Line 26 creates a variable called game_running and sets its value to be True. Notice that this variable is in lowercase letters, meaning that this is not a constant variable. When variables are in lowercase letters or are not constant, their values can (and most likely will) change throughout the course of the game. In this case, we will change the value of the game_running variable to either True or False throughout the game to indicate whether the game is currently running or being played.

Line 27 begins a while loop. The code under this while loop will execute while the game_running variable is equal to True. This is a bit confusing, but whenever you see something like "while running:" or "while y" or anything else, that means that the code underneath it will execute if that particular variable is set to True. If it isn't, this block of code (the code under the while loop) will be skipped. So, in other words, as long as the

game_running variable is set to True, the code under Line 27 will loop continuously. When the value of the game_running variable changes to False, that loop will end.

As a final note, make sure you have your colon at the end of Line 27. The loop will not program correctly if the colon is missing.

25. Press ENTER twice.

26. Type the code that you see on Lines 29 - 36 of the following screenshot:

```
27 while game_running:
28
29 #-------------------------------------------------------------
30 #Check for events
31
32     #Checking for and handling events
33     for event in pygame.event.get():
34         #Exit loop on quit
35         if event.type == QUIT:
36             game_running = False
```

Line 29 contains another comment separator.

Line 30 contains a comment explaining that this section of code will check for events. 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 31 is a blank line.

Line 32 contains a comment explaining that the code in the following section will check for and handle any events that happen in the game.

Line 33 creates a loop checks for any events that have occurred in the game. Python will register all events from the user into an event queue which can be accessed using the pygame.event.get() method. Every element in this queue is an event object. Line 33 creates a “for” loop that will check for and execute each event in the event queue.

Line 34 creates another comment explaining what the code in Lines 35 and 36 will do.

Line 35 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 36, you see the what will happen if the event.type==QUIT (Line 35) is detected in the event queue. If the event type is NOT EQUAL to QUIT, this line (Line 36) will be skipped and the program will continue on to the next line. If the event type is equal to QUIT, the game_running variable will be changed to False.

Please take note of the indentation used in the screenshot above. The indentation is very important.

27. Press ENTER twice.

28. Type the code that you see on Lines 38 – 42 of the screenshot below. Again, indentation is important.

```
36                game_running = False
37
38  #---------------------------------------------------------------
39  #Update display.
40      display.update()
41
42  #Close main game loop
```

Line 38 contains another comment separator.

Line 39 contains a comment describing what the following code does.

Line 40 will execute the display.update() method. This method updates the user's window/surface display to match the actions/changes that the computer has made.

Line 41 contains a blank line.

Line 42 contains a comment explaining that this is the end of the main game loop code.

29. Press ENTER once.

30. Type the code that you see on Lines 43 - 46 of the screenshot below.

```
42  #Close main game loop
43  #-------------------------------------------------------------
44
45  #Clean up game
46  pygame.quit()
```

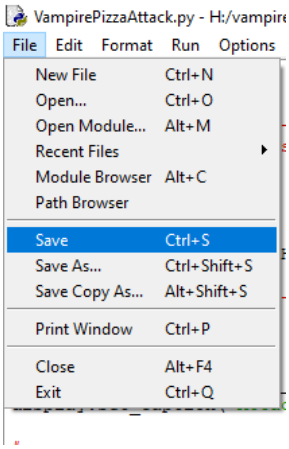
    Line 43 contains a comment separator.

    Line 44 is blank.

    Line 45 contains a comment explaining that the code below it will clean up and close out of the game.

    Line 46 exits pygame and closes out of the game window. This function will only run after the game loop has stopped running.

31. Go to File > Save.



32. At this point, you will not be able to see anything in your game window when you preview this program. For now, after you have saved your code you can close out of the Python file. You can also close out of the Python Shell if you still have it open.

Final Code:

```
#Import Libraries
import pygame
from pygame import *
#Initialize pygame
pygame.init()

#-----------------------------------------------------------
#Define constant variables

#Define the parameters of the game window
WINDOW_WIDTH = 900
WINDOW_HEIGHT = 400
WINDOW_RES = (WINDOW_WIDTH, WINDOW_HEIGHT)

#-----------------------------------------------------------
#Load assets

#Create window
GAME_WINDOW = display.set_mode(WINDOW_RES)
display.set_caption('Attack of the Vampire Pizzas!')

#-----------------------------------------------------------
#Start main game loop

#Game loop
game_running = True
while game_running:

#-----------------------------------------------------------
#Check for events

    #Checking for and handling events
    for event in pygame.event.get():
        #Exit loop on quit
        if event.type == QUIT:
            game_running = False

#-----------------------------------------------------------
#Update display.
    display.update()

#Close main game loop
#-----------------------------------------------------------

#Clean up game
pygame.quit()
```