2a.

My program was written in Python. The Purpose of my program is to play a game in which the user is battling a boss, for entertainment. The program works as a rock, paper, scissors game in which the user inputs their choice and the computer chooses one for the boss. The boss and hero take damage every time their choice loses, until their total health hits zero. My video illustrates a playthrough of the program from start to completion. The videos shows the user playing the game against the computer until the boss' health falls below 0, this leads to the game ending and a victory screen revealing itself. The video details my code through screen updates, user input, and mathematical updates shown on screen.

2b.

I completed my project entirely by myself. Before I began to write my program, I brainstormed different ideas. I wanted a program that would accurately reflect what I had learned this year so I chose a battle that involved a changing screen, as well as some math. I began my project by creating an outline of what the screen would look like. I had trouble starting because I was not well-versed in screen-making with pygame, so I had to research how to do it on Google to continue. After getting the screen set up, I had to add sprites. In order to do this, I found images that fit my theme well, and projected them into the screen. This process took a lot of trial and error because I had to continually change the positions of images, moving it a few pixels, and testing its positioning. After all of the graphics were in place, I set out setting up a damage system. This part took a while because I was unsure how to properly write a rock, paper, scissors game. I looked up my old code, and after I found that it worked, I set out on writing the code.

2c.

I wrote the winning() function on my own. This function was imperative for the program because it decided whether the boss or the hero won the last duel. The first function that is called in winning(), guess(), lets the user choose whether they would like to choose rock, paper, or scissors which is returned and stored as a variable. The guess() function is a logical function because it takes advantage of 'if' statements to make sure the user's choice is one of the allowed choices. The second function I called, guess1(), has the computer choose rock, paper, or scissors and store it in another variable. The output from this function is compared against the user's choice in the following lines in the victor() command. The victor command is a logical command because it uses 'if' and 'elif' statements to decide which users input in the winning input. After the victor command chooses a winner, the rest of the winning() function is used to decide who will take damage, and how much damage they will take. The function does this by finding the loser of the duel, and subtracting a random amount of damage from a certain range.

```
def winning():
    guesshero = guess()
    guessboss = guess1()
    winner = victor(guesshero, guessboss)
    if winner == 0:
        print("tie, no damage.")
    if winner == 1:
        randam = int(random.randrange(11,16))
        print("Hero is victorious, Boss takes " + str(randam) + " damage.")
        boss.health = boss.health - randam
    if winner == 2:
        randam = int(random.randrange(9,14))
        print("Boss is victorious, Hero takes " + str(randam) + " damage.")
        hero.health = hero.health - randam
```

```
def guess1():
    guesss = ['rock', 'paper', 'scissors']
    guessnum = random.randrange(0,3)
    guessalpha = guesss[guessnum]
    return(guessalpha)
```

```
def guess():
    guesss = ['Rock', 'rock', 'Paper', 'paper', 'Scissors', 'scissors']
    while True:
        shoot = input("Rock, Paper, Scissors, shoot: ")
        if shoot in guesss:
            return(shoot)
        else:
            print("That is not a valid guess.")
```

```
def victor(guesshero, guessboss):
    rocks = ['rock', 'Rock']
    papers = ['paper', 'Paper']
    scissors = ['scissors', 'Scissors']
    if guesshero in rocks:
        if guessboss in rocks:
            print("Rock ties rock.")
            return(0)
        elif guessboss in papers:
            print("Paper beats rock.")
            return(2)
        else:
            print("Rock beats scissors.")
            return(1)
    if guesshero in papers:
        if guessboss in rocks:
            print("Paper beats rock.")
            return(1)
        elif guessboss in papers:
            print("Paper ties paper.")
            return(0)
        else:
            print("Scissors beat paper.")
            return(2)
    if guesshero in scissors:
        if guessboss in papers:
            print("Scissors beat paper.")
            return(1)
        elif guessboss in rocks:
            print("Rock beats scissors.")
            return(2)
        else:
            print("Scissors tie scissors.")
            return(0)
```

2d.

I wrote the draw_hero_health() function all on my own. This program is an abstraction because it simplifies the process of drawing the hero's health bar. Drawing the health bar is a process that must be done every time the game deals damage. This abstraction helps manage complexity of the program because it takes a process that is done a lot, and moves it out of the way of the main body of code. This program uses math in order to decide how long the health bar is, and how much of the bar is filled. It does this by having a maximum length of 300, and a strict height of 30. In order to decide how much to fill the bar, the function takes the amount of health the hero has left, and divides it by the total amount of health the user started with.

```
def draw_hero_health(surf, x, y, pct):
    if pct < 0:
        pct = 0
    BAR_LENGTH = 300
    BAR_HEIGHT = 30
    fill = (pct / hero.healthx) * BAR_LENGTH
    outline_rect = pygame.Rect(x, y, BAR_LENGTH, BAR_HEIGHT)
    fill_rect = pygame.Rect(x, y, fill, BAR_HEIGHT)
    pygame.draw.rect(surf, GREEN, fill_rect)
    pygame.draw.rect(surf, WHITE, outline_rect, 2)
```