2a.

The create task program that I developed is a Space Invaders game. It was created using the Python language with the Pygame module. The purpose of my game is to let users be a captain and control a spaceship while shooting as many fireballs as possible to gain points. The player should also avoid crashing their spaceship into incoming fireballs or avoid negative points by not shooting blue fireballs. The screen-recorded video illustrates the main features of my program such as starting the game, ability to shoot bullets, incoming fireballs, score tracking, text display, maneuvering, and the result of a user crashing their spaceship into a fireball or setting a new personal record.

2b.

When I started independently developing my program, I was confused as to which features of my game to start creating first. Over time, I was able to plan my developmental process efficiently by focusing on incrementally developing the features such as collisions, sound-effects, firing, score tracking and sprite movements one at a time. I tested these features when integrating them together for my finished game to run. In my developmental process, I faced challenges of varying magnitude. One challenging feature to develop was tracking collisions, as I had to find an effective method to track when the sprites (images) overlap. I did this by using the width, height, and coordinates of the sprites to essentially create a box containing the sprite. With the help of inequalities, I checked if the coordinates and sides of the spaceship boxes and fireball boxes overlapped. Another tricky feature to develop was tracking a player's score history to determine their high score. After a few attempts, I decided that the fastest method was to write the player's score to an external text file every time they crashed their spaceship. This file is then read and converted into a list of numbers to determine the high score using the built-in Python `max()` function.

2c.

My program's `gameLoop` algorithm integrates two sub algorithms, `keyPresses`, and `bulletCollision`, which function independently and in combination with one another. The `bulletCollision` algorithm tracks collisions respectively among moving sprites using the width, height, and coordinates of these sprites as parameters, to essentially construct a box containing the sprite. Finally, the algorithm determines if a collision occurred by checking if the coordinates and sides of the boxes overlap using inequalities. The `keyPresses` algorithm uses a method from the Pygame module for monitoring the player's keystrokes to move the spaceship object controlled by the player within the game window. When the player pushes an arrow key, the algorithm moves the spaceship sprite in the user's desired direction using mathematical operators and creates boundaries for the spaceship sprite using inequalities, thus not allowing the player to move the sprite outside the game window. The `bulletCollision` and `keyPresses` algorithms work in combination with one another to achieve the intended purpose of the program, as the `bulletCollision` algorithm returns a true boolean if the player shot a fireball, hence allowing the program to determine if the player's score should be changed, and the `keyPresses` algorithm allows the player to maneuver the spaceship and avoid crashing into fireballs.

```
def gameLoop():
    # Objects:
    spaceship = spaceShip(215, 400)
    bulletList = []
    fireballList = []
    score = 0
    scoreList = []
    gameRunning = True
    while gameRunning == True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                gameRunning = False
        playerHighScore = scoringMethod()
        # Bullets:
        for bullet in bulletList:
            if bullet.bullety < gameWindowHeight and bullet.bullety > 0:
                bullet.moveBullet()
            else:
                bulletList.pop(bulletList.index(bullet))

        # Fireballs:
        for fireball in fireballList:
            if fireball.firebally < gameWindowHeight and 
fireball.firebally > 0:
                fireball.moveFireball()
            else:
                fireballList.pop(fireballList.index(fireball))
            if fireball.fireballRandNum == 1:
                fireball.fireballSprite = 
pygame.image.load('fireball2.png')
```

```
                fireball.fireballPlus = False
            else:
                fireball.fireballSprite =
pygame.image.load('fireball.png')
                fireball.fireballPlus = True
        if len(fireballList) < 4:

fireballList.append(fireballClass(random.randrange((spaceship.spaceShipW
idth // 2), (gameWindowWidth - (spaceship.spaceShipWidth))), 1))

        moveBullets(bulletList, spaceship)
        keyPresses(spaceship)

        # Collisions and Crashing:
        for bullet in bulletList:
            for fireball in fireballList:
                collision = bulletCollision(bullet.bulletx,
bullet.bullety, bullet.bulletWidth, bullet.bulletHeight,
fireball.fireballx, fireball.firebally, fireball.fireballWidth,
fireball.fireballHeight)
                if collision == True:
                    fireball.fireballCollision(fireballList)
                    bullet.bulletCollision(bulletList)
                    if fireball.fireballPlus == True:
                        score += 1
                    else:
                        score -= 1

        for fireball in fireballList:
            crash = spaceshipCollision(spaceship.spaceShipx,
spaceship.spaceShipy, spaceship.spaceShipWidth,
spaceship.spaceShipHeight, fireball.fireballx, fireball.firebally,
fireball.fireballWidth, fireball.fireballHeight)
            if crash == True:
                scoreListMethod(scoreList, score)
                scoreFileWriteMethod(scoreList)
                if score > playerHighScore:
                    newRecordMethod()
                else:
                    gameOverMethod()

        # FPS:
        gameClock = pygame.time.Clock()
        gameClock.tick(30)

        # Score Text:
        scoreText = textClass("Score: " + str(score), red, 15)
        scoreDisplayText = scoreText.textDisplay()
        highScoreText = textClass("High Score: " + str(playerHighScore),
red, 15)
        highScoreDisplayText = highScoreText.textDisplay()

        gameWindow.blit(background, (0, 0))
        spaceship.drawSpaceShip()
        for bullet in bulletList:
            bullet.drawBullet()
        for fireball in fireballList:
            fireball.drawFireball()
        gameWindow.blit(scoreDisplayText, (430, 20))
        gameWindow.blit(highScoreDisplayText, (392, 35))
        pygame.display.update()
    pygame.quit()
```

2d.

My abstraction is `bulletClass`. This class stores the attributes of the bullet objects along with the functions to work with. These functions perform tasks such as drawing the bullet objects onto the game window or controlling the movements of the bullet objects using mathematical operators. I proceeded to call the functions within `bulletClass` for every bullet object within the display. I also used `bulletClass` to create all of the bullet objects in my game. I did this by storing each bullet object in a separate list named `bulletList`. Then, I added a separate conditional statement that tracks the location of each bullet object and removes the object from `bulletList` if it was not within the display or if it collided with a fireball object. This was beneficial in managing the complexity of my program as I did not have to manually create every bullet object in my game, and delete the object when it collided with a fireball object.

```
class bulletClass():
    def __init__(self, x, y):
        self.bulletx = x
        self.bullety = y
        self.bulletWidth = 3
        self.bulletHeight = 7
        self.bulletSpeed = 8
        self.bulletColor = (255, 0, 0)
    def bulletCollision(self, list):
        list.pop(list.index(self))
    def moveBullet(self):
        self.bullety -= self.bulletSpeed
    def drawBullet(self):
        pygame.draw.rect(gameWindow, self.bulletColor, (self.bulletx,
self.bullety, self.bulletWidth, self.bulletHeight))
```