Contents

[Group Profile 2](#_Toc158461027)

[Question 1 3](#_Toc158461028)

[Code: 3](#_Toc158461029)

[Output: 5](#_Toc158461030)

[Question2: 6](#_Toc158461031)

[Code: 6](#_Toc158461032)

[Output: 11](#_Toc158461033)

[Question3: 12](#_Toc158461034)

[GitHub file screenshot: 12](#_Toc158461035)

[Team member participation: 13](#_Toc158461036)

# Group Profile

**Group Members:**  
[Sameer Basnet] - [S372941]  
[Samir Dhakal] - [S373048]  
[Susanti Djie] - [S375655]  
[Nishat Anjum] - [S374044]

**Github:**

<https://github.com/Sameer84/HIT137-Assignment-03.git>

# Question 1

## Code:

import tkinter as tk

#Define a custom Calculator class inheriting from tk.Tk

class Calculator(tk.Tk):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()  #Call the constructor of the parent class

        self.title("Calculator")  #Set the title of the calculator window

        self.geometry("300x380")  #Set the initial size of the window

        self.result\_var = tk.StringVar()  #Variable to store calculation result

        self.create\_widgets()  #Method to create GUI elements

    #Method to create GUI widgets

    def create\_widgets(self):

        #Create an Entry widget for displaying and entering calculations

        self.entry = tk.Entry(self, textvariable=self.result\_var, font=("Arial", 16), justify="right")

        self.entry.grid(row=0, column=0, columnspan=4, padx=10, pady=10, sticky="nsew")

        #Define buttons with their text, row, and column positions

        buttons = [

            ("7", 1, 0), ("8", 1, 1), ("9", 1, 2), ("/", 1, 3),

            ("4", 2, 0), ("5", 2, 1), ("6", 2, 2), ("\*", 2, 3),

            ("1", 3, 0), ("2", 3, 1), ("3", 3, 2), ("-", 3, 3),

            ("0", 4, 0), (".", 4, 1), ("=", 4, 2), ("+", 4, 3),

            ("C", 5, 0)  #Add a "Clear" button

        ]

        button\_width = 4

        button\_height = 1

        button\_padding = 8

        #Create buttons dynamically and place them in the grid layout

        for (text, row, column) in buttons:

            button = CalculatorButton(self, text=text, font=("Arial", 16), width=button\_width, height=button\_height, column=column)

            button.configure(command=lambda t=text: self.on\_button\_click(t))

            button.grid(row=row, column=column, padx=button\_padding, pady=button\_padding, sticky="nsew")

        #Configure grid layout to expand with window size

        self.grid\_rowconfigure(0, weight=1)

        self.grid\_columnconfigure(0, weight=1)

    #Method to handle button clicks

    def on\_button\_click(self, text):

        if text == "=":

            self.calculate()  #Call the calculate method when "=" button is clicked

        elif text == "C":

            self.clear()  #Call the clear method when "C" button is clicked

        else:

            current\_text = self.result\_var.get()

            self.result\_var.set(current\_text + text)

    #Method to evaluate and display calculation result

    def calculate(self):

        try:

            result = eval(self.result\_var.get())

            self.result\_var.set(str(result))

        except Exception as e:

            self.result\_var.set("Error")

    #Method to clear the entry widget

    def clear(self):

        self.result\_var.set("")  #Clear the entry widget when "C" button is clicked

#Define a custom CalculatorButton class inheriting from tk.Button

class CalculatorButton(tk.Button):

    def \_\_init\_\_(self, \*args, column, \*\*kwargs):

        super().\_\_init\_\_(\*args, \*\*kwargs)

        #Set different background colors based on the column index

        if column % 2 == 0:

            self.configure(bg="lightblue", fg="black")

        else:

            self.configure(bg="lightgreen", fg="black")

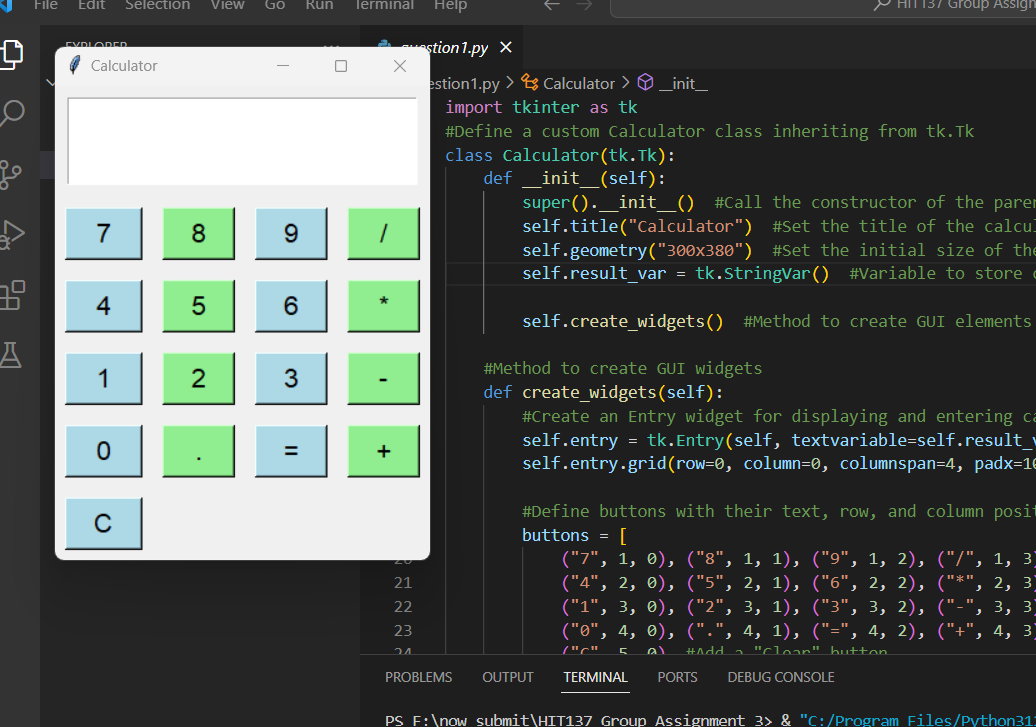
#Main entry point of the program

if \_\_name\_\_ == "\_\_main\_\_":

    app = Calculator()  #Create an instance of the Calculator class

    app.mainloop()  #Start the Tkinter event loop to display the GUI and handle events

## Output:



# Question2:

## Code:

import pygame

import sys

import random

#Initialize Pygame

pygame.init()

#Constants

WIDTH, HEIGHT = 800, 600

FPS = 60

#Colors

WHITE = (255, 255, 255)

BLACK = (0, 0, 0)

RED = (255, 0, 0)

#Create the game window

screen = pygame.display.set\_mode((WIDTH, HEIGHT))

pygame.display.set\_caption("Human-like Character Game")

#Clock for controlling the frame rate

clock = pygame.time.Clock()

#Player class

class Player(pygame.sprite.Sprite):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        # Create the player's image

        self.image = pygame.Surface((30, 90))

        self.image.fill(WHITE)

        self.rect = self.image.get\_rect(midleft=(100, HEIGHT - 150))  # Adjusted position for the character

        self.speed = 5

        self.jump\_height = -20

        self.gravity = 1

        self.vel\_y = 0

        self.health = 100

        self.lives = 3

    def update(self):

        #Get keyboard input for player movement and jumping

        keys = pygame.key.get\_pressed()

        self.vel\_y += self.gravity

        #Move left and right

        if keys[pygame.K\_LEFT] and self.rect.left > 0:

            self.rect.x -= self.speed

        if keys[pygame.K\_RIGHT] and self.rect.right < WIDTH:

            self.rect.x += self.speed

        #Jump when spacebar is pressed

        if keys[pygame.K\_SPACE] and self.rect.bottom >= HEIGHT:

            self.vel\_y = self.jump\_height

        #Apply gravity

        self.rect.y += self.vel\_y

        #Check if the player is on the ground

        if self.rect.bottom >= HEIGHT:

            self.rect.bottom = HEIGHT

            self.vel\_y = 0

#Projectile class

class Projectile(pygame.sprite.Sprite):

    def \_\_init\_\_(self, x, y):

        super().\_\_init\_\_()

        #Create the projectile's image

        self.image = pygame.Surface((10, 10))

        self.image.fill(WHITE)

        self.rect = self.image.get\_rect(midleft=(x, y))  #Adjusted position for the projectile

        self.speed = 8

    def update(self):

        #Move projectile horizontally

        self.rect.x += self.speed

        #Remove projectile if it goes out of the screen

        if self.rect.x > WIDTH:

            self.kill()

#Enemy class

class Enemy(pygame.sprite.Sprite):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        #Create the enemy's image

        self.image = pygame.Surface((30, 30))

        self.image.fill(RED)

        self.rect = self.image.get\_rect(midright=(WIDTH, random.randint(0, HEIGHT - 50)))  #Adjusted position for the enemy

        self.speed = random.randint(3, 6)

    def update(self):

        #Move enemy towards the left

        self.rect.x -= self.speed

        #Remove enemy if it goes out of the screen

        if self.rect.right < 0:

            self.kill()

#Collectible class

class Collectible(pygame.sprite.Sprite):

    def \_\_init\_\_(self):

        super().\_\_init\_\_()

        #Create the collectible's image

        self.image = pygame.Surface((20, 20))

        self.image.fill(WHITE)

        self.rect = self.image.get\_rect(midright=(WIDTH, random.randint(0, HEIGHT - 30)))  #Adjusted position for the collectible

        self.speed = 3

    def update(self):

        #Move collectible towards the left

        self.rect.x -= self.speed

        #Remove collectible if it goes out of the screen

        if self.rect.right < 0:

            self.kill()

#Create sprite groups

all\_sprites = pygame.sprite.Group()

projectiles = pygame.sprite.Group()

enemies = pygame.sprite.Group()

collectibles = pygame.sprite.Group()

#Create player

player = Player()

all\_sprites.add(player)

#Main game loop

running = True

while running:

    for event in pygame.event.get():

        if event.type == pygame.QUIT:

            running = False

        elif event.type == pygame.KEYDOWN:

            if event.key == pygame.K\_ESCAPE:

                running = False

            if event.key == pygame.K\_SPACE:

                #Create projectile when spacebar is pressed

                projectile = Projectile(player.rect.right, player.rect.centery - 5)

                projectiles.add(projectile)

                all\_sprites.add(projectile)

    #Spawn enemies randomly

    if random.randint(1, 100) < 4:

        enemy = Enemy()

        enemies.add(enemy)

        all\_sprites.add(enemy)

    #Spawn collectibles randomly

    if random.randint(1, 300) == 1:

        collectible = Collectible()

        collectibles.add(collectible)

        all\_sprites.add(collectible)

    #Update all sprites

    all\_sprites.update()

    #Check for collisions

    for enemy in pygame.sprite.spritecollide(player, enemies, True):

        #Decrease player's health when colliding with enemies

        player.health -= 10

        #Check for game over conditions

        if player.health <= 0:

            player.lives -= 1

            player.health = 100

            if player.lives == 0:

                running = False

    for enemy, projectile in pygame.sprite.groupcollide(enemies, projectiles, True, True).items():

        #Handle collisions between enemies and projectiles (e.g., increase score)

        pass

    for collectible in pygame.sprite.spritecollide(player, collectibles, True):

        #Handle collection of collectibles (e.g., health boost, extra life)

        pass

    #Draw all sprites

    screen.fill(BLACK)

    all\_sprites.draw(screen)

    #Draw health bar

    pygame.draw.rect(screen, RED, (10, 10, player.health \* 2, 20))

    #Draw remaining lives

    font = pygame.font.Font(None, 36)

    lives\_text = font.render(f"Lives: {player.lives}", True, WHITE)

    screen.blit(lives\_text, (WIDTH - 150, 10))

    #Update display

    pygame.display.flip()

    #Cap the frame rate

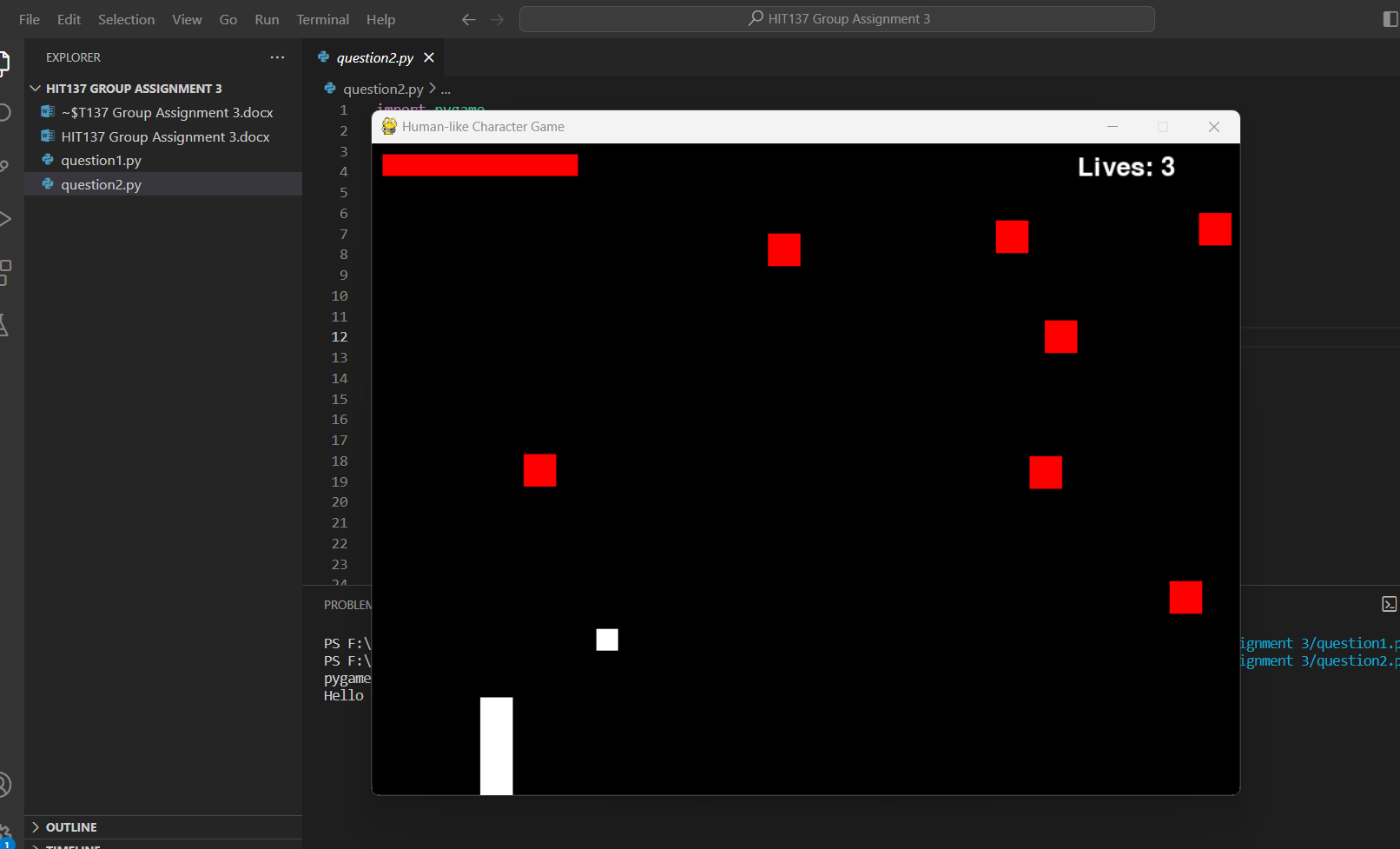
    clock.tick(FPS)

#Quit Pygame

pygame.quit()

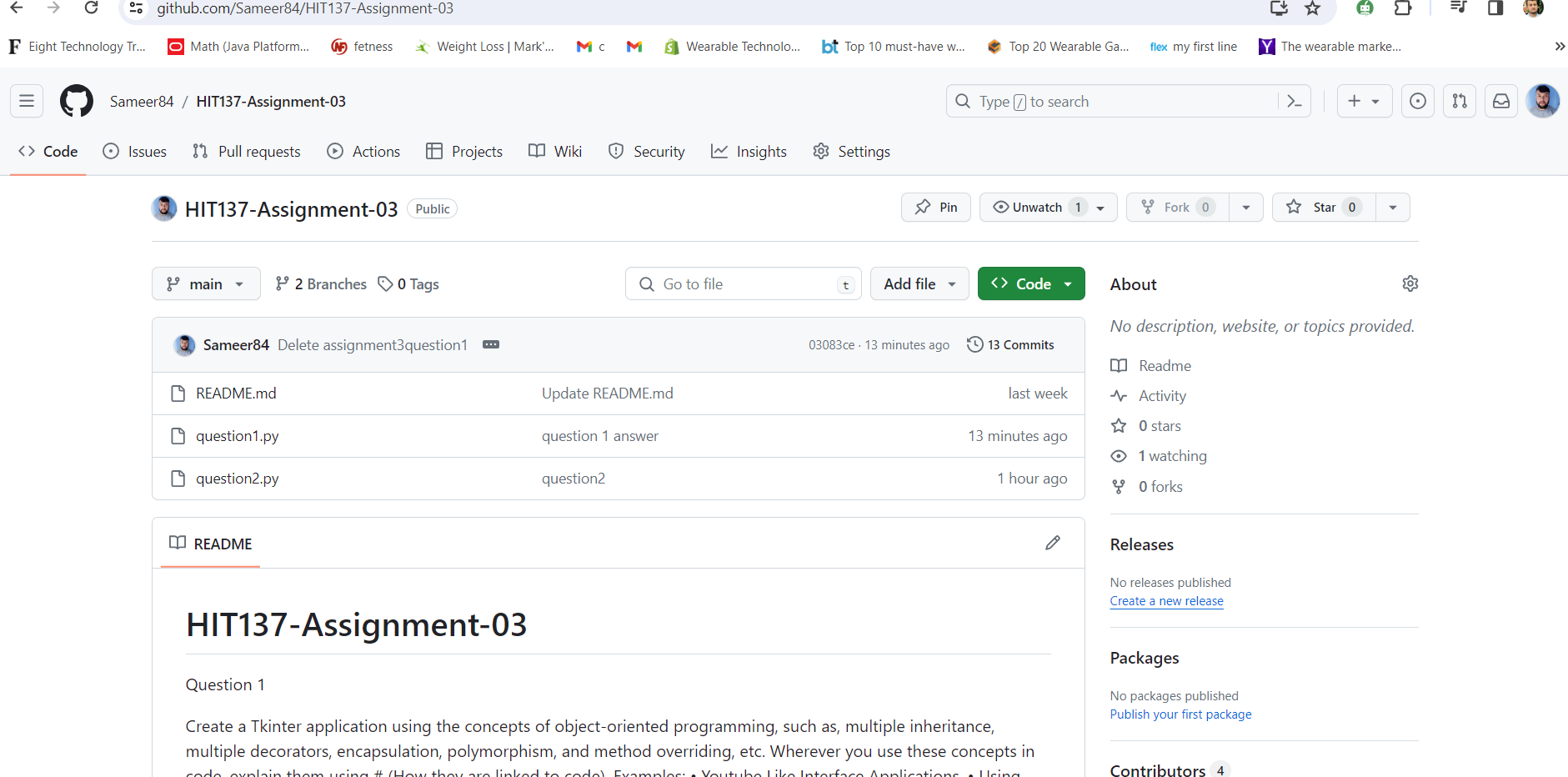
sys.exit()

## Output:



# Question3:

## GitHub file screenshot:



## Team member participation:

