Ky Kartchner
A01847326
CS 1400-1
Assignment 10

Task 1: Chessboard

1. **Requirements Specification** – The program must adhere to the following requirements:
   - Must utilizes the "main-task1.py" file in its original form as basis of program
   - Must create a "chessboard.py" file that contains a Chessboard class.
   - Must draw the same as Assignment 8, but instead be drawn with the call of draw() as shown in main-task1.py

2. **System Analysis –** The program will have the following input/output:

   Outputs: 8x8 chessboard drawn on screen

   Inputs: x-y coordinates of board

   width and height of board

3. **System Design**

   Step 1. User is prompted to enter a starting point and the width and height of the board

   Step 2. An instance of Chessboard is created with the inputted coordinates and dimensions

   Step 3. The chessboard is drawn on the screen by invoking it's draw() method

   - In order to accomplish Step 3, the original draw functions will still be utilized but changed so they are inside the chessboard class and called by the draw() method.

4. **Implementation - (code)**

Implemented chessboard.py:

```
5.
    class Chessboard:
        def __init__(self, start_x, start_y, width=250, height=250):
            self.__start_x = start_x
            self.__start_y = start_y
            self.__width = width
            self.__height = height

            turtle.speed(0)

        # Draws the chess board outline then calls draw_all_squares():
        def draw(self):
            # Calculate dimension of an individual square
```

```
            square_width, square_height = self.__width / 8, self.__height / 8

            # Draw board outline
            Chessboard.__draw_square(self.__start_x, self.__start_y, self.__width,
    self.__height, pen_color="red")

            # Draw all black squares
            Chessboard.__draw_all_squares(self.__start_x, self.__start_y,
    square_width, square_height)
            Chessboard.__draw_all_squares(self.__start_x + square_width,
    self.__start_y + square_height,
                                          square_width, square_height)
            return

        # Draws all of the black squares by calling draw_square() many times:
        @staticmethod
        def __draw_all_squares(start_x, start_y, square_width, square_height):
            for y in range(4):
                offset_y = square_height * y * 2
                for x in range(4):
                    offset_x = square_width * x * 2
                    Chessboard.__draw_square(start_x + offset_x, start_y +
    offset_y,
                                             square_width, square_height,
    fill_color = "black")
            return

        # Draws a single black square:
        @staticmethod
        def __draw_square(pos_x, pos_y, width, height, fill_color="",
    pen_color="black"):
            turtle.penup()
            turtle.pencolor(pen_color)
            turtle.setposition(pos_x, pos_y)
            turtle.pendown()
            turtle.begin_fill()
            turtle.fillcolor(fill_color)

            for i in range(2):
                turtle.forward(width)
                turtle.left(90)
                turtle.forward(height)
                turtle.left(90)

            turtle.end_fill()
            turtle.penup()
            return
```

Provided main function:

```
import turtle
from chessboard import Chessboard


def main():
    start_x, start_y = eval(input("Enter a starting point: "))
    width = input("Input a width: ")
    height = input("Input a height: ")

    if width == "" and height == "":
        chessboard = Chessboard(start_x, start_y)
    elif height == "":
```

```
        chessboard = Chessboard(start_x, start_y, width=eval(width))
    elif width == "":
        chessboard = Chessboard(start_x, start_y, height=eval(height))
    else:
        chessboard = Chessboard(start_x, start_y, eval(width), eval(height))

    chessboard.draw()

    turtle.hideturtle()
    turtle.done()


main()
```

5. Testing

Test 1

	X-coord.	Y-coord.	Width	Height
Input	0	0	250	250
Expected Output	250 x 250-pixel Chessboard printed to screen with bottom-left corner at (0,0)			
Actual Output	Same as expected			

Test 2

	X-coord.	Y-coord.	Width	Height
Input	0	0	125	250
Expected Output	125 x 250-pixel Chessboard printed to screen with bottom-left corner at (0,0). Width is half as much as the height			
Actual Output	Same as expected			

Test 3

	X-coord.	Y-coord.	Width	Height
Input	-125	-62.5	250	125
Expected Output	250 x 125-pixel Chessboard printed centered on the screen. Height is half as much as the width.			
Actual Output	Same as expected			

Problems: NONE

Task 2: Password

1. **Requirements Specification** – The program must adhere to the following requirements:

   - Must define a class called "Password" in its own file.
   - Must only ever create one instance of password.
   - Must prompt user for a password and allow them to enter one.
   - Program must check password for adherence to the following rules:
     - Must have at least eight characters
     - Must consist of only letters and digits
     - Must contain at least two digits
     - Cannot contain the word "password"
     - Cannot end with "123"
   - Must have a private method for each password test.
   - Must display "valid password" if the rules are followed, "invalid password" if not.
   - Must ask user if they want to enter another password and allow them to do so.
   - "Password" class must have at least the following:
     - set_password() method
     - is_valid() method
       - This should return a Boolean
     - get_error_message() method
       - This should return a string that indicates the problems with the password
       - It should be called if is_valid() returns False

2. **System Analysis –** The program will have the following input/output:

   Outputs: Prints validity of password ("Password is valid" or "Password is invalid" with specific errors)

   Inputs: Password

3. **System Design**

   Step 1. User is prompted to enter a password

   Step 2. The validity of the password is printed

   Step 3. The user is asked if they would like to enter another password (if yes, steps repeat; if no, program exits)

4. **Implementation - (code)**

password.py:

```
class Password:
    def __init__(self, password):
        self.__value = ""
        self.__value_to_check = password
        self.__error_message = ""
```

```
        self.__is_valid = True

        self.set_value(password)

    def set_value(self, password):
        self.__value_to_check = password
        if self.is_valid(password):
            self.__value = password
            print("valid password\n")
        else:
            print("invalid password\n")
            print(self.get_error_message())

    def get_error_message(self):
        return self.__error_message

    def is_valid(self, password):
        self.__value_to_check = password
        self.__is_valid = True

        self.__check_length()
        self.__check_alpha_numeric()
        self.__check_number_of_digits()
        self.__check_if_contains_password()
        self.__check_if_ends_with_123()

        return self.__is_valid

    def __check_length(self):
        if len(self.__value_to_check) < 8:
            self.__error_message += "Password must have at least 8 characters\n"
            self.__is_valid = False

    def __check_alpha_numeric(self):
        if not self.__value_to_check.isalnum():
            self.__error_message += "Password must only consist of letters and
digits\n"
            self.__is_valid = False

    def __check_number_of_digits(self):
        number_of_digits = 0
        for char in self.__value_to_check:
            if str(char).isdigit():
                number_of_digits += 1

        if number_of_digits < 2:
            self.__error_message += "Password must contain at least two digits\n"
            self.__is_valid = False

    def __check_if_contains_password(self):
        if "password" in self.__value_to_check:
            self.__error_message += "Password cannot contain the word
\"password\"\n"
            self.__is_valid = False

    def __check_if_ends_with_123(self):
        value = self.__value_to_check
        last_char = len(self.__value_to_check) - 1
        if value[last_char - 2] == "1" and value[last_char - 1] == "2" and
value[last_char] == "3":
            self.__error_message += "Password cannot end with \"123\"\n"
            self.__is_valid = False
```

task2:

```
from password import Password


def main():
    print("NEW ACCOUNT:")
    password = Password(input("Enter account password: "))

    done = False
    while not done:
        command = input("Would you like to enter another password? (Y/N): ")
        if command == 'Y' or command == 'y':
            password.set_value(input("Enter account password: "))
        else:
            done = True
main()
```

5. Testing

Test 1

Input	password123
Expected Output	Password is invalid: Password cannot contain “password” Password cannot end with “123”
Actual Output	Same as expected

Test 2

Input	Bla!bla
Expected Output	Password is invalid: Password must have at least 8 characters Password must only consist of letters and digits Password must contain at least two digits
Actual Output	Same as expected, except it printed the error_messages from test 1 again because error_message had been neglected to be reset.

Test 3

Input	Th1s1sMyPass34word
Expected Output	Password is valid
Actual Output	Same as expected

Problems: When entering another password, the previous error messages stored in error_message did not get cleared. I fixed this by resetting error_message to an empty string. Other than that, things worked great.