CS2011 Intermediate Programming and Problem Solving I 2022-2023

Final project report

Student Name: Rachel Lombard

Student Number: 121705031

Demonstrator's Name: Prof. Laura Maye

Assignment Number: 3

Submission Date: 2/12/2022

Word Count: 880

Escapebot Class

Escapebot class was previously created by Tim, who was in the midst of developing an escape room game. Included in the Escapebot class were basic interactions which were eventually built upon and modified to create a fully working escape room game using object orientated programming concepts. The code is organised into objects and interact with one another. By organising code into objects, it can support reuse, be extended and is easier to maintain. These objects have behaviours and attributes which are controlled in a class. The main objectives were to enable reusability and flexibility, to remove as much hardcoding as possible as well as inputting new necessary methods. Below is an example of making the code reusable and not hardcoded.

```
def __init__(self, name, lives):
    self.name = name #changed this t
    self.position = 1 #decided to ke
    #1 is a good starting number and
    self.goodbye = "Thank you for pl
    self.lives = lives #changed live
```

In this case the class variable is Escapebot, it is not unique and can be shared across all instances of a particular object and can be reinstantiated across all instances of the class. Contrast to this, an instance variable can be created from the class and be unique, this was done within the self.lives under the display_lives method.

```
def display_lives(self):
    print("You have %s lives remaining..." % self.lives)
```

The initial code is a nested dictionary format which includes the question, stimulus and answer that are going to be asked to the game player in the console. It contains key value pairs and can be changed in the future to be developed by future coders by adding a getter and setter which would entail different questions. These can then be called in the def

__init__() constructor if they choose which is used to initialize objects. Self is usually written within the __init__() constructor as it refers to a unique instance of the object created, in this case, name and lives were included so that they could be called further down into the Escapebot code.

```
def __init__(self, name, lives):
    self.name = name #changed this
    self.position = 1 #decided to
    #1 is a good starting number a
    self.goodbye = "Thank you for
    self.lives = lives #changed li
```

For example, self.name = name which allows for resusability, the name of the bot can be changed as its object is called in the main instance section of Escapebot, e.g escape_bot = Escapebot("Zoey"). This object is being instantiated. An instance needs to be created of the class in order to input data which is declared in __init__() and are unique to each object. These are instance variables.

```
# created an instance of Escapebot
# Bot can be renamed, number of lives can be changed
escape_bot = EscapeBot("Zoey", 3)
print("Information about object:", escape_bot)
escape_bot.draw("Bot") #calling the previous method a
escape_bot.display_name()
```

Private variables can also be created so that they are only accessible within the class that it is made. Its methods can also be called within the class. There are no private variables inside of Escapebot class. Getters and setters can be created so that the private variables can be accessed outside the class. Getters get values and setters can set new values. Properties are then set by accessing one attribute which is done by using the property() function. These are examples of encapsulation which bundles data into methods and hides data to avoid potential hacking or accidental deletion.

```
def get_name(self):
    return self.name

def set_name(self, new_name):
    if type(new_name) == str:
        self.name = new_name

names = property(get_name, set_name)
```

String representation was completed to return a string to the user which contains information about the object instance.

```
def __str__(self):
    desc = "Robot. Name: %s, Position: %s, Total lives: %s" % (self.name, self.position, self.lives)
    return desc
```

File reader and Question File Reader class

To further advance and develop the escape room, Filereader class and QuestionFileReader class were created along with a Main file. The classes were imported to the main file.

```
from filereader import QuestionFileReader
from bot import EscapeBot
```

Filereader acted as the parent class and QuestionFileReader is the child class. Inheritance was applied within this code, as the child class inherits public attributes and methods of the parent class. It also allows reusability of code. Method overriding occurred within the child class with the super() function.

```
#created child class
#added super() to method override
#self.__filename is a private instance variable
class QuestionFileReader(FileReader):
    def __init__(self, filename):
        super().__init__(filename)
        self.__filename = filename
        print("Instance of QuestionFileReader class created!")
```

Main.py

Main.py called all of the methods from each file, but also had an object instance of Escapebot class and Question file reader so that these could be called in the main and be able to function.

```
#Created an object instantiation
question_file_reader = QuestionFileReader("python-game-file.txt")
escape_bot = EscapeBot("Zoey", 3, question_file_reader.all_dictionary_questions())
```

Feature 1 and feature 2

Different actions were carried out when creating feature one and feature two. The first idea for feature one was during the game, the user will be prompted with a question on whether they would like to change the question from the dictionary. Zoey bot asks the user when they get a question wrong if they would like to change their questions, if yes, a menu will appear with different options for the user to choose from, they must enter in a number which correlates with the options given. They will then receive a question and the number of lives will stay the same. If the user chooses no, the current question that they were currently on will be asked again.

Feature two involved the game changing the number of lives in some way. When the user is on their last life, they are prompted with a question which asks if they would like to answer another question and gain back a life. They will gain back the life if they get the answer correct, if they get the answer wrong then the game ends as they will lose their only life that is remaining.

Inheritance, object interaction and code reuse including reusing different instance variables were applied. New methods were also created for these features which also include inheritance within its code. Examples of this are below –

Calling methods from Filereader.py into main

```
if choice == "1":
    input_list = []
    input_list = [int(item) for item in input("Enter the list items: ").split()]
    EscapeBot("Zoey", 3, question_file_reader.lines_as_dictionary(input_list))
    break
elif choice == "2":
    input_list = []
    input_list = [int(item) for item in input("Enter the 2 item list: ").split()]
    EscapeBot("Zoey", 3, question_file_reader.get_dictionary_range(input_list))
    break
elif choice == "3":
    EscapeBot("Zoey", 3, question_file_reader.random_dictionary_questions())
    break
```

Calling the decrement lives () method in the main which is in Bot.py

```
else:
    no_lives_left = escape_bot.decrement_lives()
    if no_lives_left == False:
        in_play = False
```

Inheriting read.all() into exclude_dictionary_range method

```
def exclude_dictionary_range(self, questions_range):
    try:
        question_dict = {}
        data = self.read_all()
```