CS2011

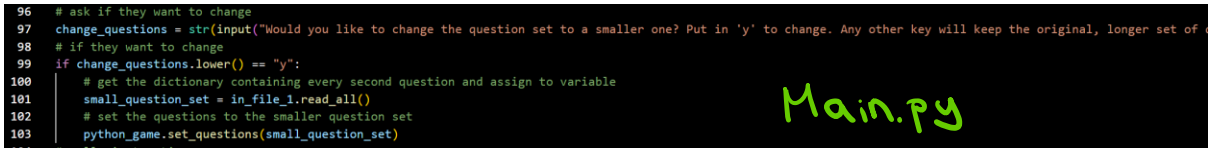
Rebecca Bronczyk

121722211

Feature 1

The first feature that was added, lets the user choose between a longer and a shorter set of questions. This is done, before the game starts so that based on whether the user wants to play the short or full version of the game. For instance they might want to play the short version if they are not very experienced or short on time. The program probes the user to use input and enter 'y' if they want to change the questions to the shorter version. Typing in any other key will keep the long version.

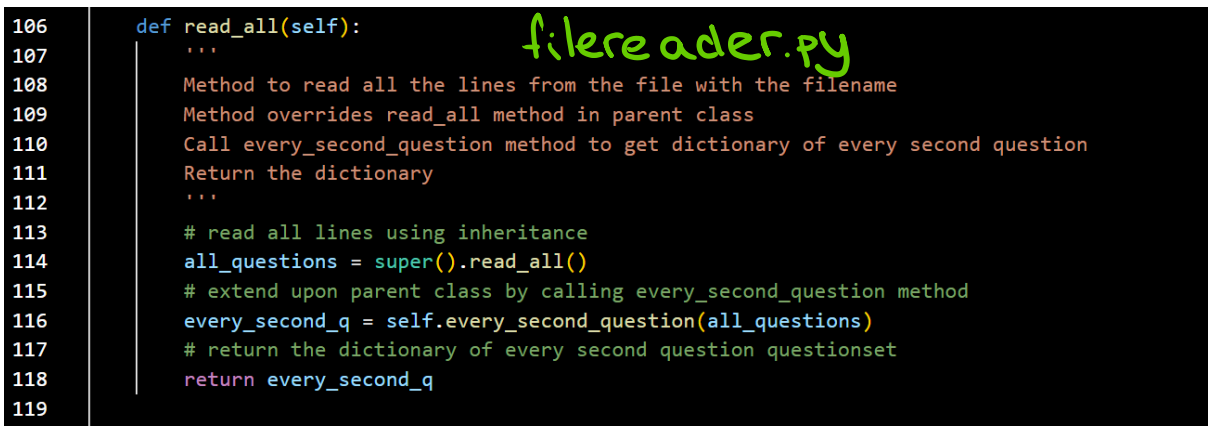
Picture 1




```
96  # ask if they want to change
97  change_questions = str(input("Would you like to change the question set to a smaller one? Put in 'y' to change. Any other key will keep the original, longer set of
98  # if they want to change
99  if change_questions.lower() == "y":
100     # get the dictionary containing every second question and assign to variable
101     small_question_set = in_file_1.read_all()
102     # set the questions to the smaller question set
103     python_game.set_questions(small_question_set)
```

If "y" is input, the instance created for the filereader class calls the method "read_all()" in the filereader file in the class QuestionFileReader(FileReader) (Picture 1, line 99). Calling super().read_all(), this method uses inheritance, that is that the child class inherits all public variables and methods from the parent class (Picture 2, Line 122). Furthermore, since it has the same name in the same case (i.e. lower case with the lower dash between read and all) as the method in the parent class. Method overriding is used by the "read_all()" method in the child class since calling read_all() for an instance of the child class (as done in main.py in Picture 1) now will call not the method in the parent but the one in the child class. The "read_all()" method in the child class therefore reads all lines from the file like the parent method but builds upon it.

Picture 2




```
106     def read_all(self):
107         '''
108         Method to read all the lines from the file with the filename
109         Method overrides read_all method in parent class
110         Call every_second_question method to get dictionary of every second question
111         Return the dictionary
112         '''
113         # read all lines using inheritance
114         all_questions = super().read_all()
115         # extend upon parent class by calling every_second_question method
116         every_second_q = self.every_second_question(all_questions)
117         # return the dictionary of every second question questionset
118         return every_second_q
119
```

The read_all() method in the child class builds upon the method in the parent class, in that it calls the every_second_question() method (Picture 2, Line 116). This method extracts every second question (Picture 3) from the lines read from the file and calls another method creating a dictionary out of these questions (Picture 3, Line 142). The to_dictionary() method (Picture 4) was created, since adding the code to change the list to a dictionary in the every_second_question() method would have been redundant because it is also used in the every_dictionary_question() method used for getting all questions. Therefore both methods now use the output by the to_dictionary() method.

Picture 3

filereader.py

```
120     def every_second_question(self, all_questions:list):
121         '''
122         Method to:
123         - get every second question
124         - Call a method to turn it into a dictionary
125         - Return the dictionary of questions, stimuli and answers
126         '''
127         # remove the first line and keep every second line in the list
128         all_questions = all_questions[1::2]
129         # create new list for all second questions
130         new_all_questions = []
131         # loop over the items in the list
132         for item in all_questions:
133             # remove line breaks
134             item = item.strip("\n")
135             # split the line contents at the comma
136             item = item.split(",")
137             # append the question to the list
138             new_all_questions.append(item)
139         # assign the new_all_questions containing every second question to all_questions
140         all_questions = new_all_questions
141         # call the to_dictionary method to turn the list to a dictionary and store it in a variab
142         dictionary_questions = self.to_dictionary(all_questions)
143         # return the dictionary questions
144         return dictionary_questions
```

Picture 4

filereader.py

```
83      def to_dictionary(self, wanted_questions:list):
84          '''
85          Method to turn a list (wanted_questions) into the dictionary format
86          '''
87          # create readall_dict dictionary to hold the questions in a nested dictionary format (this is the outer dictionary)
88          readall_dict = {}
89          # loop through the readall list keeping the index and package
90          for index, package in enumerate(wanted_questions):
91              # create subdict for creating nested dictionary
92              subdict = {}
93              # key is index+1 because first question is 1 not 0
94              key = index+1
95              #  question is 0 index in current index with subkey "question"
96              subdict["question"] = package[0]
97              # stimulus is 1 index in current index with subkey "stimulus"
98              subdict["stimulus"] = package[1]
99              # answer is rest of the line with subkey "answers"
100             subdict["answers"] = package[2::]
101             # add subdict with key to readall_dict
102             readall_dict[key] = subdict
103         # return the created dictionary
104         return readall_dict
```

Object interaction is used, since the dictionary of questions generated in the QuestionFileReader(FileReader) class in the filereader.py file are then handed to the set_questions() method in the EscapeBot() class in the bot.py file (Picture 5) so that it can use the questions for the game. This interaction is created in the main.py file (Picture 1, Line 103).

Picture 5

```
267     def set_questions(self, new_questions):
268         '''
269         Method to set new questions -> overwrites the class variable
270         left like it was because it is flexible and correct
271         '''
272         # check if the parameter is a dictionary
273         if type(new_questions) != dict:
274             # if not print error message
275             print("Questions must be of type dictionary (nested). Questions not reset.")
276             # return to exit the set_questions method
277             return
278         # if type of the parameter is dictionary, check for the nested dictionaries
279         for key in new_questions:
280             # if the key of the quetions are not digits
281             if str(key).isdigit() == False:
282                 # print error message
283                 print("Questions are not in the correct format. Questions not reset")
284                 # return to exit the set_questions method
285                 return
286             # if the questions are numbered with digits
287             else:
288                 # assign the each question package to keys
289                 keys = new_questions[key]
290                 # if the type of the nested dictionary is not dictionary
291                 if type(keys) != dict:
292                     # print error message
293                     print("Questions must be of type dictionary (nested). Questions not reset.")
294                     # return to exit the set_questions method
295                     return
296                 # if each question does not contain 3 keys (question, stimulus, answer)
297                 if len(keys) != 3:
298                     # print error message
299                     print("Questions are not in the correct format. Questions not reset")
300                     # return to exit the set_questions method
301                     return -1
302                 # if the keys of each question are not called question, answers and stimulus
303                 if "question" not in keys and "answers" not in keys and "stimulus" not in keys:
304                     # print error message
305                     print("Questions are not in the correct format. Questions not reset")
306                     # return to exit the set_questions method
307                     return
308         # only if types of parameter and nested dictionaries and their contents are according to the format reset the questions
309         EscapeBot.questions = new_questions
310         # if questions reset print success message
311         print("questions reset!")
```

bot.py

Feature 2

When the user enters a wrong question and loses a life, the user will be asked to input whether they would like to randomly either gain or lose a life. This is only done if the user still has a life, so if the lives were 0 after they had lost it the question, the game will terminate and not give the opportunity to gamble. This is because if they lose another life, lives would be down to -1 and that does not make sense. If the user has one life left and looses it by gambling, the python game is terminated as well.

Picture 6

```
69      python_game.display_incorrect()
70      # call the decrement_lives() function inside the EscapeBot class to decrement the lives of the player
71      python_game.decrement_lives()
72      # get the remaining lives of the player from inside the EscapeBot classes get_lives() method & save the remaining_lives to the variable
73      remaining_lives = python_game.get_lives()
74      # print how many lives remaining
75      print(python_game.display_lives())
76      # if the player is not out of lives
77      if remaining_lives > 0:
78          # ask if they want to gamble to gain or lose a life
79          another_life = input("Do you take the random risk of gaining your life back or losing another one? If so type in 'y' any other key will continue the game with your current liv
80          # if they do
81          if another_life.lower() == "y":
82              # call the method randomly_gain_or_lose() in bot.py
83              python_game.randomly_gain_or_lose()
84          # the function calls itself again to display the question again
85          question_and_answer()
86      # if the answer was false and the player is out of lives go back to inplay after question_and_answer was called and check wether all questions have been played & display the termi
```

main.py

When the user loses a life but still has at least one life left, the program will ask them for input on whether they would like to gamble (Picture 6, Line 79). If the user inputs “y”, the randomly_lose_or_gain() method in the EscapeBot class in the bot.py file is called (Picture 6, Line 83). This method randomly creates a Boolean (Picture 7, Line 119). If it is False, the

decrement_lives() method (Picture 8a) is called again (Picture 7, Line 129). This supports code reuse since the decrementing of lives is not part of the randomly_lose_or_gain() method and thus can be called from the latter but also when the user loses a life due to a wrong answer. If the Boolean is True, the increase_lives() method (Picture 8b) is called (Picture 7, Line 123). It was created for this function but it could also be used for other functionality if it was to be added in the future, an therefore also supports reusability.

Picture 7

bot.py

```
114     def randomly_gain_or_lose(self):
115         '''
116         Method to randomly decide wether the user loses or gains a life
117         '''
118         # create random booolean
119         random_number = bool(random.randint(0,1))
120         # if 1 = True
121         if random_number:
122             # increment lives
123             self.increase_lives()
124             # print winner message
125             print("Congrats, you gained a life! \n You now have: " + str(self.lives) + " lives.")
126         # if 0 = False
127         else:
128             # take another life
129             self.decrement_lives()
130             # if lives left after losing one again
131             if self.lives > 0:
132                 # print loser message but tell them that they can continue playing
133                 print("Loser! You lost another one, but life goes on! \n You now have: " + str(self.lives) + " lives")
134             # if now has 0 lives
135             else:
136                 # print loser message and tell them that the game is over
137                 print("Loser! You lost another one, life doesn't go on! \n You're out of lives.")
```

Picture 8

bot.py

```
92      a) def decrement_lives(self):
93             '''
94             Method to decrement lives if the answer was wrong
95             Returns whether lives left
96             '''
97             # decrement self.lives
98             self.lives -= 1
99             # if lives left, assuming the game ends when 0 lives are left
100            if self.lives > 0:
101                # return True that lives left
102                return True
103            # if no lives left (lives = 0)
104            else:
105                # return lives left is False
106                return False
107
108     b) def increase_lives(self):
109            '''
110            Method to increase lives if randomly gains a life
111            '''
112            self.lives += 1
113
```

Object oriented programming is used for the methods used for this functionality, since they are part of the EscapeBot class. A class for an object holds all attributes (in EscapeBot only public instance variables and no private or class variables) and behaviours (methods) about their object. So in this case, the EscapeBot class holds all methods and attributes of the bot created for playing the game. Methods are all behaviours the bot has for example decreasing lives of the bot. Getters and setters that get the value of a variable or set it to a new value are common as well.

Local variables have a certain scope for example can only be accessed in one method, therefore the same variable can be assigned to a new value in another method without affecting it in the other method. They are used for all coding not only object oriented programming.

Private variables can only be used within the class they are created. Else a getter or setter has to be used to get or set its value. While class variables are shared by all instances of a class, instance variables are unique to each instance.

The public instance variable “self.lives” (Picture 9, Line 32) which is an attribute of the bot and holds the lives of the player is kept in the EscapeBot class and is assigned in __init__(). The first reason that it is a public instance variable is, that the lives of each instance of the bot may vary, for example depending on the number of questions or type of bot used. Secondly, the variable has to be usable in all methods of the class and there is no need to make it private. Private instance variables can only be called in the class they were assigned. Since there is only one class (EscapeBot class) in the bot.py file, which does not have a child class and the variable does not contain sensitive information, it is made public. This furthermore enables the programmer to easily assign a new value to “self.lives” in the main.py file, without necessarily having to use a getter or setter.

Picture 9