# Create – Applications From Ideas
# Written Response Submission Template

Please see Assessment Overview and Performance Task Directions for Student for the task directions and recommended word counts.

**Program Purpose and Development**

2a)

In my CREATE Project, I decided to write my program in python. The purpose of this program is for the user to utilize the keys on their keypad to navigate their on-screen avatar throughout a maze and collect each letter on each letter to beat the maze game. In my CREATE demonstration video, one can see how the introduction of the game, read the beginning dialogue, and see the on-screen avatar go through the first level of the game. In the video, the significant feature that is to be highlighted is that the on-screen avatar is unable to move any place that an X is. This not only creates the border for the game but also dictates and limits the moves that the player is able to make.

2b)

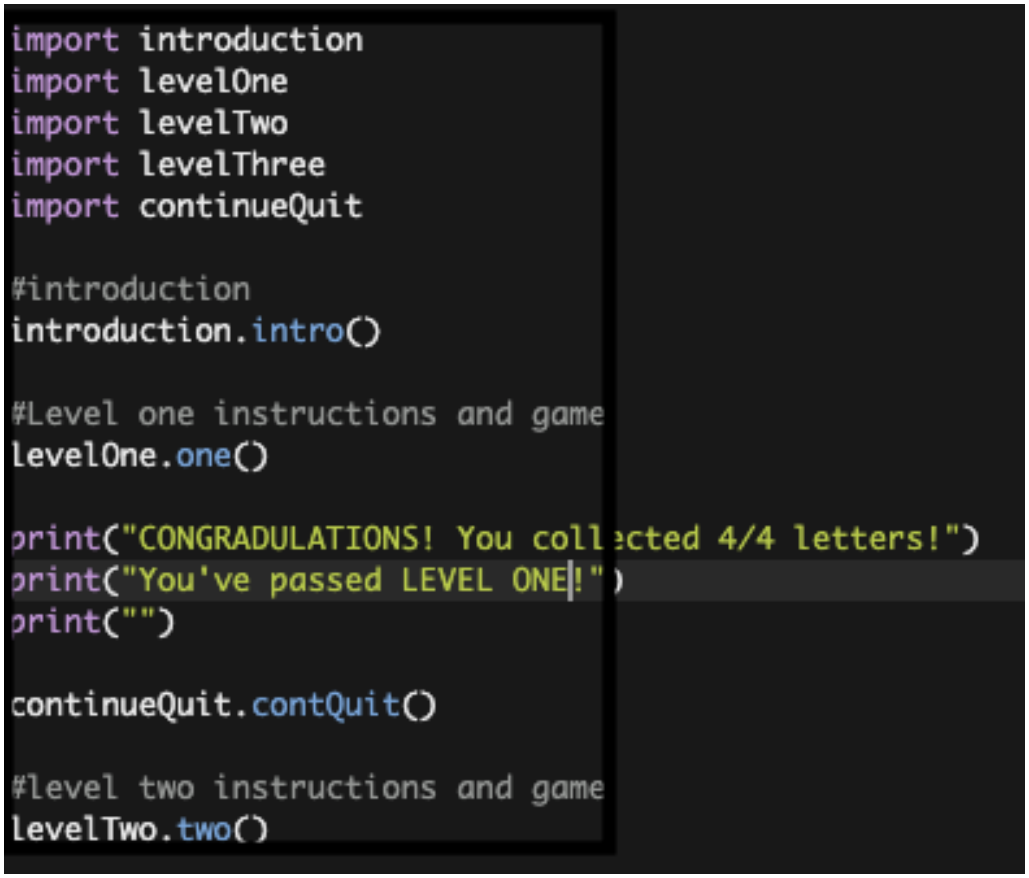
While writing my python-based CREATE project, I initially believe it would be difficult, yet a very doable challenge. The reason for my confidence was due to the fact that I have learned Java previously and knew about ncurses. Nevertheless, while progressing through my code, I was interrupted by unforeseen challenges. The initial challenge was after writing the first level of the CREATE maze game. I wanted to add a condition for continuing to the next level, and also between the text information and the actual maze. To overcome this challenge, I created a function, because of its frequent use, that allows the user to either quit or continue. I thought of this after I noticed how often the function would need to be used. The second challenge that I confronted was how to create multiple levels using ncurses in python. Originally, I had coded every level into the main function, but I had not proficiently advanced my knowledge on ncurses to know how to create a new map without having to create a new window. What I found out was that I could put each level into separate functions and call them into the main function.

2c)

```
while user_input != ord('q'):

    #move down
    if user_input == ord('s') and chr(stdscr.inch(oy + 1, ox)) != 'X' and oy + 1 < 35:
        stdscr.addch(oy, ox, ' ')
        oy += 1
        stdscr.addch(oy, ox, 'o')
        stdscr.refresh()

    #move up
    if user_input == ord('w') and chr(stdscr.inch(oy - 1, ox)) != 'X' and oy - 1 < 35:
        stdscr.addch(oy, ox, ' ')
        oy -= 1
        stdscr.addch(oy, ox, 'o')
        stdscr.refresh()

    #move left
    if user_input == ord('a') and chr(stdscr.inch(oy, ox - 1)) != 'X' and ox - 1 < 35:
        stdscr.addch(oy, ox, ' ')
        ox -= 1
        stdscr.addch(oy, ox, 'o')
        stdscr.refresh()

    #move right
    if user_input == ord('d') and chr(stdscr.inch(oy, ox + 1)) != 'X' and ox + 1 < 35:
        stdscr.addch(oy, ox, ' ')
        ox += 1
        stdscr.addch(oy, ox, 'o')
        stdscr.refresh()
```

For my double algorithm, I am highlighting my code that controls the movement fo the on-screen avatar. The first algorithm that is implemented is a logic algorithm, which tests to see if the user input receive from the user is not equal to the char 'q'. If the user input is equal to the char 'q', then the whole program will end. If 'q' is not equal to the user input, the program will enter the while loop and move directly to the second algorithm. In the second algorithm, the program is testing to see if the user input received is equal to one of the designated movement keys, 'w', 's', 'a', or 'd'. If the user input is equal to one of these four keys, then the on-screen avatar will move accordingly in that key's appointed direction.

2d)



```
import introduction
import levelOne
import levelTwo
import levelThree
import continueQuit

#introduction
introduction.intro()

#Level one instructions and game
levelOne.one()

print("CONGRADULATIONS! You collected 4/4 letters!")
print("You've passed LEVEL ONE!")
print("")

continueQuit.contQuit()

#level two instructions and game
levelTwo.two()
```

In the picture above, there are are a plethora of abstractions highlighted in the text above. Firstly, one such abstraction is represented by the beginning lines of the program. This demonstrates an abstraction because it uses the statement 'import' to make code found outside the main function usable. The second example of an abstraction is found on line 8 in the picture above. The program calls the function intro() from the introduction file found outside the main function. By calling this, it displays in the console the instructions for the game. This helps simplify the main program as all a reader needs to know is that it contains the instructions for the program without reading each line. Another example is line 11 where the program calls the one() function found in the levelOne file. This helps decomplexify the main program as there are 3 similar but different levels within the game, the reader of the program knows what the function is calling and does not have to read dozens of lines of code with some similar lines. If there was a change that one wanted to make, they only need to go to the specific folder containing the code.