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)

My program was written using JavaScript in the App Lab on Code.org. The purpose of my program is to provide a simple way to test people's knowledge of art history. The first event that occurs is the program will ask the user for a number, in this case how many points they need to win. Then it will direct them to the directions screen, which tells them how the quiz will be scored. Afterwards the quiz will start at question one and end at question five. The game is won when the user meets or exceeds their goal. In my video, I show the title screen and the instructions. Then I show each question and the right answer to each but, purposefully get some questions wrong to show the win and lose screen.

2b)

I made this game by first picking my topic, planning out each question and each screen. Afterwards, I made a simple click through quiz that I later expanded on. The programming for this project was done independently. The main feature I knew was going to take a lot longer to develop was the scoring system and it became my main priority. One problem I encountered through testing, that the value of the variables holding the points the player could lose and could gain were not updating. To solve this problem, I looked over the section that called on the function, found that it was within an if statement and simply moved it to the beginning of teh function holding the statement. The second problem I ran into while testing was that the part of the app that was displaying the values of the points the player could gain or lose was not updating correctly. Soon, I found that the code was calling on the global version of the variables. I simply put the new code that would update the display in a new function and had the function updating the values call on it.

2c)

```
// parent function,
function scoreUpdate (){
   updateValue();
   winloseCondition(mustGet);
if (mystery == 1 ) {
  score = score + correct;
 return score;
}
else if( (mystery != 1)){
  score = score - incorrect;
 return score;
}
}
//child 1
function updateValue() {
 correct = correct + randomNumber(3,6);
  incorrect = incorrect + randomNumber(3,6) ;
}
//child 2 and abstraction
function winloseCondition(value){
  if(score >= value){
    setText("winLabel","Congratulations, you met your goal at" +score+ "points");
    hideElement("loseLabel");
    hideElement("loseImage");
   showElement("winLabel");
   showElement("winImage");  }
```

```
 else if ( score <= value){

   setText("loseLabel","Oh no! You did not meet your goal at " +score+ "points");
hideElement("loseImage");

  hideElement("winLabel");

   hideElement("winImage");

  showElement("loseLabel");

  showElement("loseImage");

}

 }
```

The main algorithm is inside the “scoreUpdate” function”. This algorithm updates the score. To do this, it checks if the variable “mystery” equals one. If so, then the number of points the player earns is added on, if not then the number of points the player loses is subtracted. The first sub-algorithm is called “updateValue”. Once the player answers the question, we need to update the number of points the player is supposed to lose or gain, which should increase as the questions get harder. This algorithm simply adds a random value to the number of points the player is supposed to lose or gain. The second sub-algorithm is called “ winloseCondition” and it checks if the players gained enough points to win the game. If the player gains enough points then the player gets taken to the “win” screen at the end, if not then they get taken to the lose screen. The algorithm checks for this using an if-statement. Together these algorithms not only provide their user with their score, and the number of points they could gain or lose but also shows the user the end result.
This helps the purpose of creating a useful study tool by being user friendly.

2d)

```
//child 2 and abstraction
function winloseCondition(value){
  if(score >= value){
    setText("winLabel","Congratulations, you met your goal at" +score+ "points");
    hideElement("loseLabel");
    hideElement("loseImage");
   showElement("winLabel");
   showElement("winImage"):
  } else if ( score <= value){
    setText("loseLabel","Oh no! You did not meet your goal at " +score+ "points");
    hideElement("loseImage");
   hideElement("winLabel");
    hideElement("winImage");
   showElement("loseLabel");
   showElement("loseImage”);
}
```

My abstraction is the “winloseCondition” that I programmed myself. This function checks whether or not the player has met the point goal. If they met the point goal then the win screen is shown, if not then the lose screen is shown. This is apart of the main algorithm but it also is something that needs to be done every time the player finishes and restarts the quiz since they may want to select a new goal. This abstraction helps manage complexity since it allows the task to not only be called in multiple places in a nice orderly fashion that allows the code to be more clearly written but also allows the user to not have to worry about how that section of code works to win, just that they need to get that amount of points because of abstracting all the details away to that extent.