2a. Provide a written response or audio narration in your video that:

- identifies the programming language;
- identifies the purpose of your program; and
- explains what the video illustrates.

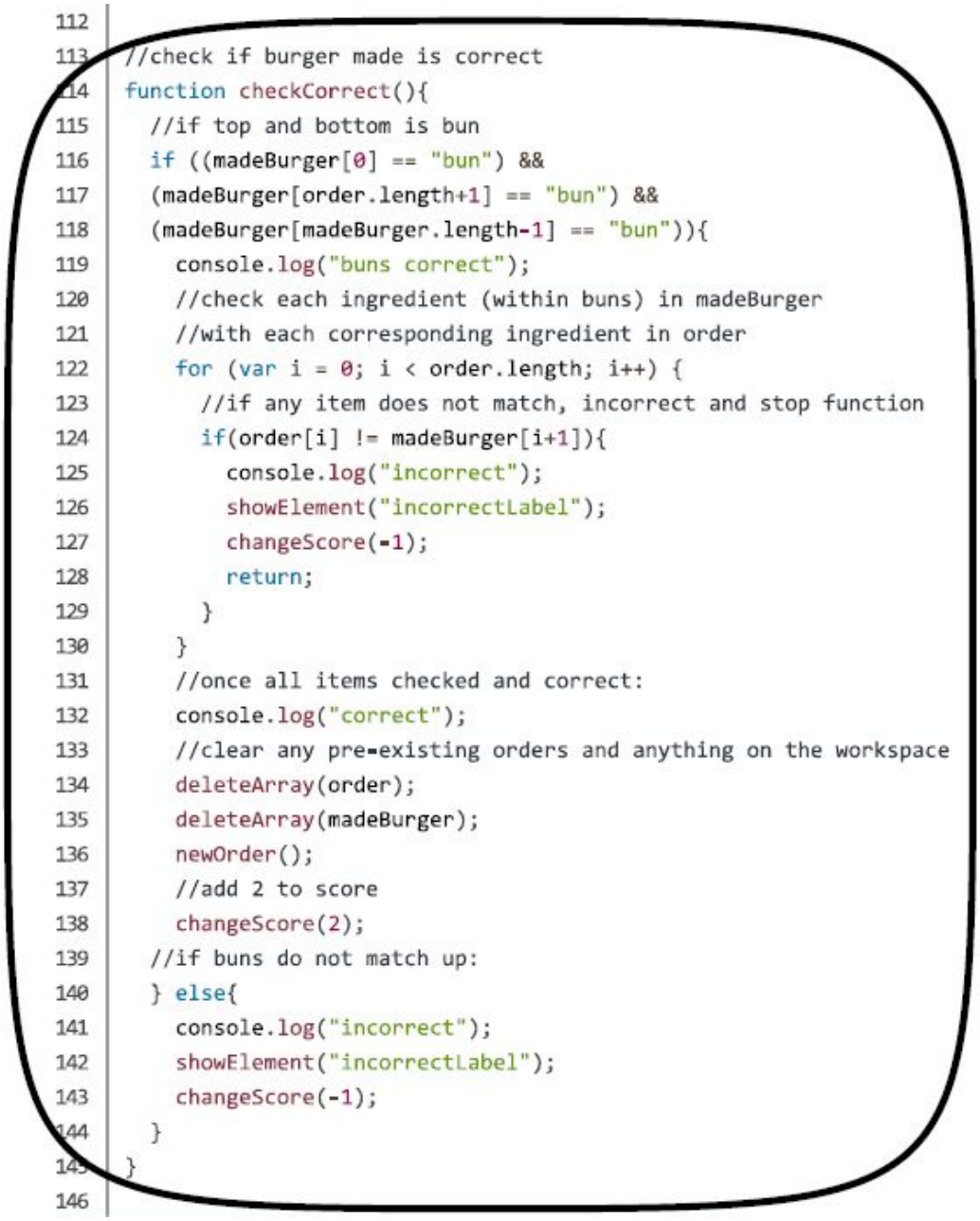
(Must not exceed 150 words)

JavaScript is the programming language used in the program. The purpose of this program is to provide entertainment by allowing users to create as many ordered burgers as possible in 45 seconds. The video illustrates typical user interaction. The user views the instructions and starts to play. An order is given in a speech bubble. The user clicks on the corresponding ingredients to make the burger and clicks the check mark to submit the burger. If it is correct, a new order is given, the board is cleared, and the score increases by 2 points. If it is incorrect, an incorrect message shows up, the score decreases by 1 point, and the user is given a chance to add or start over the burger by clicking the trash button. Once 45 seconds run out, the screen changes to a game over screen that displays the score.

2b. Describe the incremental and iterative development process of your program, focusing on two distinct points in that process. Describe the difficulties and/or opportunities you encountered and how they were resolved or incorporated. In your description clearly indicate whether the development described was collaborative or independent. At least one of these points must refer to independent program development. (Must not exceed 200 words)

First, I developed a user interface and used pseudocode to plan out the code (incremental process). Then, I started coding and testing my code. If something did not work, I used an iterative process, going back and fixing issues. I had a problem with keeping score. Every time the score function was called, it set the score to the parameter value of `changeScore` instead of adding that value to the previous score. Because the score variable was local, causing a new score variable to be created every time `changeScore` was called, I made the score variable global instead. Another problem I encountered was when checking if the top bun was at the correct index in the array `madeBurger`. I referred to `madeBurger[order.length]` as the bun location. However, since `order` only contained the correct items randomly selected to be within the buns and not the 2 buns, 2 needed to be added to `order.length` to get the correct length of `madeBurger`. Also, since indices in arrays start at 0, not 1, 1 needed to be subtracted from the array length to obtain the index. So, I added 1 to `order.length`. Both developments were independent.

2c. Capture and paste a program code segment that implements an algorithm (marked with an oval in section 3 below) and that is fundamental for your program to achieve its intended purpose. This code segment must be an algorithm you developed individually on your own, must include two or more algorithms, and must integrate mathematical and/or logical concepts. Describe how each algorithm within your selected algorithm functions independently, as well as in combination with others, to form a new algorithm that helps to achieve the intended purpose of the program. (Must not exceed 200 words)



```
112
113 //check if burger made is correct
114 function checkCorrect(){
115   //if top and bottom is bun
116   if ((madeBurger[0] == "bun") &&
117   (madeBurger[order.length+1] == "bun") &&
118   (madeBurger[madeBurger.length-1] == "bun")){
119     console.log("buns correct");
120     //check each ingredient (within buns) in madeBurger
121     //with each corresponding ingredient in order
122     for (var i = 0; i < order.length; i++) {
123       //if any item does not match, incorrect and stop function
124       if(order[i] != madeBurger[i+1]){
125         console.log("incorrect");
126         showElement("incorrectLabel");
127         changeScore(-1);
128         return;
129       }
130     }
131     //once all items checked and correct:
132     console.log("correct");
133     //clear any pre-existing orders and anything on the workspace
134     deleteArray(order);
135     deleteArray(madeBurger);
136     newOrder();
137     //add 2 to score
138     changeScore(2);
139   //if buns do not match up:
140   } else{
141     console.log("incorrect");
142     showElement("incorrectLabel");
143     changeScore(-1);
144   }
145 }
146
```

This algorithm checks if the user-made burger matches the order. It uses an if statement first to check if the top and bottom buns are in the correct indices in `madeBurger`. If not, the score is decreased by 1. Then, a for loop that repeats for the length of the array for the correct burger is used to compare each item in the burger made array with the corresponding item in the correct burger array. If any item does not match, the score is decreased by 1 and the function is returned. If the whole loop runs without any items that do not match, the burger is correct, the score is increased by 2, the arrays are cleared, and a new order is created. `changeScore` is an embedded function that adjusts the score by adding the parameter value and updates the score text to reflect the changes. `deleteArray` uses a for loop that runs for the length of the array, removing each item in the array starting at the last index. It also clears the user-made burger on the UI. `newOrder` randomly selects 3-5 ingredients to store in the

order array.

2d. Capture and paste a program code segment that contains an abstraction you developed individually on your own (marked with a rectangle in section 3 below). This abstraction must integrate mathematical and logical concepts. Explain how your abstraction helped manage the complexity of your program. (Must not exceed 200 words)

```
147  //updates score
148  function changeScore(amt){
149    //change score by amount
150    score += amt;
151    console.log("score: " + score);
152    setText("scoreText", "Score: " + score);
153    //score text becomes green if at least five
154    if(score >= 5){
155      setStyle("scoreText", "color: green");
156    //score text becomes red if lower than 0
157    }else if(score < 0 ){
158      setStyle("scoreText", "color: red");
159    //score text becomes black if 0-4
160    }else{
161      setStyle("scoreText", "color: black");
162    }
163  }
164
```

This function was used whenever the score needed to be changed. In the program, 2 was added to the score for a correct burger and 1 was deducted from the score for an incorrect burger. Because the function includes the amount to be changed as a parameter and adds the amount to the score (mathematical concept), it can be called whenever the score needs to be changed, regardless of the amount to be changed. This manages the complexity of the program by simplifying the score changing process and creating a generalization for it. Also, the function adjusts the color of the score text according to the value. If the score is equal to or higher than 5, it is green, if it is less than 0, it is red, and any value in between is black. Again, this function is a generalization that can be used in any of these cases because it implements conditional statements and if-else statements (logical concepts) to check the value of the score and assign the appropriate text color.

Reference

AppLab. (n.d.). *Code.org*. Retrieved from

https://code.org/educate/applab