Unit 5 Assessment 4 Review

1-

Going through the code line by line:

- i is set to 1
- The code inside the loop runs until i equals 5 (the length of the data)
- Inside the loop, the data at index i is set to the value of the next index. Therefore, each number in the array is shifted down by one.
- i is then incremented by one each time, until it reaches 5
- Note that when i = 5, nothing is done to the value at index 5 because the loop has stopped by then
- Thus, shifting all values down one index except for the last one, the data = [3,1,8,4,4] at the end

2-

Looking at the code line by line:

- i is set to 1
- The code inside the loop will run to the end of the list (the length of the list)
- Inside the loop, if the data at index i equals n, the i (index) is displayed
- i is incremented by one until it equals the length

In summary, this code goes through the entire list and if the value equals n at that index, it displays it.

3-

Looking at the code, *count* is set to 0 and at the very end, *count* is returned. Therefore, we already know that a count of something will be returned. Going through the code:

- i is set to 1
- The code inside the loop is repeated until it reaches the end of the data length
- If the data at index i is less that the value at the next index, count is incremented by one
- i is incremented by 1 until it reaches the end of the list

Since count is returned, the number of adjacent items that are in ascending order is returned.

4-

The purpose of the procedure *countFives* is to see how many fives are in the deck. A variable in *countFives* that keeps track of this information is the variable *count*. To send this information to the playGame() function, it has to be returned. Therefore, the missing code is:

```
RETURN(count)
```

5-

Starting off with the robot at B3 and that data = ["F", "F", "R", "F", "L", "F", "R", "R", "F"], let's go through each index of the data:

- "F" - the robot can move forward and is at B2
- "F" - the robot can move forward and is at B1
- "R" - the robot turns right and is facing east
- "F" - the robot can move forward and is at C1
- "L" - the robot turns left and is facing north
- "F" - the robot cannot move forward, nothing happens
- "R" - the robot turns right and is facing east
- "R" - the robot turns right and is facing south
- "F" - the robot can move forward and ends up at C2

6-

Looking at the if statements:

- If a is >= (greater than or equal) to b, and a is >= to c, return a
- else if, b is >= (greater than or equal) to a, and b is >= to c, return b
- Otherwise, (implying that c is greater than both a and b) c is returned

This code will return the largest number of the three. It may be helpful to use concrete examples by assigning values to a, b, and c.

7-

Going through the code line by line as it repeats while keeping track of data[]:

- Currently, data = [3,10,4,9,16]
- val is set to 0, and i is set to 1
- Every time the loop repeats, val equals the current val plus the data at index i. i is incremented by one.

- First repetition: val = 0, i = 1, data[i] = 3. val = 0 + 3 = 3
- Second repetition: val = 3, i = 2, data[i] = 10. val = 3 + 10 = 13
- Third repetition: val = 13, i = 3, data[i] = 4. val = 13 + 4 = 17
- val is displayed where val = 17

8-

There are many valid answers to this question. Here is one example:

Cecilia could keep a separate list to track which of the items on her list she has actually found and placed in her shopping cart. So for example if she had a shopping list called list and list of true/false values called got that is initially filled with false to indicate she hasn't gotten anything yet. Then as she shops if list[i] is "eggs", and she got eggs, then the app would update got[i]=true.

Other uses:

- List of prices
- Date of last purchase
- List of categories

9-

Going through the code line by line as it repeats while keeping track of data[]:

- Currently, data = [2,3,1,2]
- i is set to 1, val is set to 0, and n is set to the length of data which n = 4
- The loop repeats 4 times. Every time it loops: val equals the current val plus the data at index i, the data at index i is set to val, and i is incremented by one.
- First repetition: current val = 0, i = 1, data[i] = 2. val = 0 + 2 = 2. data[1] = 2
- Current data = = [2,3,1,2]
- Second repetition: current val = 2, i = 2, data[i] = 3. val = 2 + 3 = 5. data[2] = 5
- Current data = = [2,5,1,2]
- Third repetition: current val = 5, i = 3, data[i] = 1. val = 5 + 1 = 6. data[3] = 6
- Current data = = [2,5,6,2]
- Fourth repetition: current val = 6, i = 4, data[i] = 2. val = 6 + 2 = 8
- Final data = = [2,5,6,8]

In summary: This code adds the current index value to the previous index value, resulting in the accumulative sum of the data.

10-

In a series of if-else-if statements, the order of the statements matters because the first statement that evaluates to true is the code that will be executed, and no other statements need to be checked. So if you got a score of 90, then the second condition: else if(score >= 60)evaluates true (since 90 > 60) and the function will return “D”.

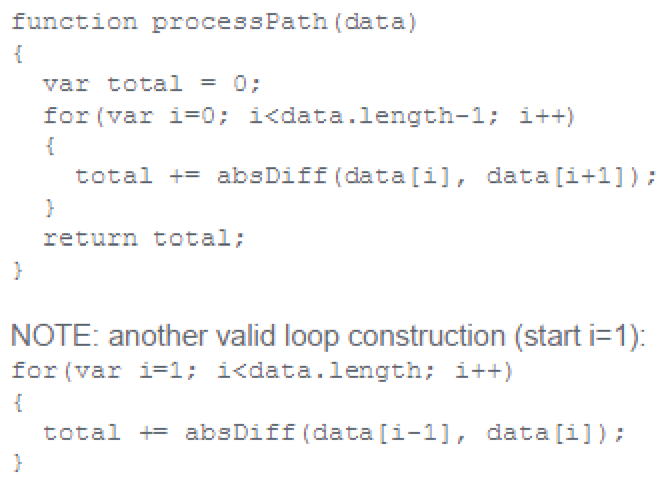
11-

Looking at the code, val is set to 0. Going through the data, if the current item in the data is greater than val, set val to the current item. At the very end, val is returned.

In other words, the largest number in data is set to val. In this case, 16 will be returned.

12-

Javascript:



```
function processPath(data)
{
  var total = 0;
  for(var i=0; i<data.length-1; i++)
  {
    total += absDiff(data[i], data[i+1]);
  }
  return total;
}
```

NOTE: another valid loop construction (start i=1):

```
for(var i=1; i<data.length; i++)
{
  total += absDiff(data[i-1], data[i]);
}
```

Pseudocode:

```
PROCEDURE processPath(data)
{
  total ← 0
  i ← 1
  REPEAT UNTIL (i = LENGTH(data))
  {
    total ← total + absDiff(data[i], data[i+1])
    i ← i+1
  }
  RETURN (total)
}
```