





Create Performance Task

Row 5 - Algorithm Implementation

Instructions

1. You should only use this version if you cannot write on a pdf.
2. Read the criteria for this row on slides 3 - 4.
3. Note the points on slide 5.
4. For each response (*slides 6 – 15*):
 - a. Underline any code, phrases or sentences that meet any of the criteria.
 - b. Tick () any criteria that are satisfied and cross () any criteria that are not satisfied.
 - c. Write a  in the bottom right corner if the response gets the mark for this row (*all criteria have been satisfied*), otherwise write a .
5. When finished, save this presentation as a pdf with 2 slides to a page and submit this file.

Scoring Criteria

- The written response:
 - includes a program code segment of a **student-developed** algorithm that includes
 - sequencing
 - selection
 - iteration
 - explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

Note: the word "algorithm" so the procedure does NOT have to be student developed, only the code. This means that any code is acceptable, as long as the student did not copy and paste the code from somewhere and actually wrote the code themselves.

Any code segment of more than 2 lines will show sequencing so I suggest that you do not focus on this requirement in this presentation.

Decision Rules

- Consider ONLY written response 3c when scoring this point.
- Responses that do not earn the point in row 4 may still earn the point in this row.
- Requirements for program code segments:
 - The algorithm being described can utilize existing language functionality or library calls.
 - An algorithm that contains selection and iteration, also contains sequencing.
 - An algorithm containing sequencing, selection, and iteration that is not contained in a procedure can earn this point.
 - Use the first code segment, as well as any included code for procedures called within this first code segment, to determine whether the point is earned.
 - If this code segment calls other student-developed procedures, the procedures called from within the main procedure can be considered when evaluating whether the elements of sequencing, selection, and iteration are present as long as the code for the called procedures is included.
- Do NOT award a point if any one or more of the following is true:
 - The response only describes what the selected algorithm does without explaining how it does it.
 - The description of the algorithm does not match the included program code.
 - The code segment consisting of the selected algorithm is not included in the written response.
 - The algorithm is not explicitly identified (i.e., the entire program is selected as an algorithm without explicitly identifying the code segment containing the algorithm).
 - The use of either the selection or the iteration is trivial and does not affect the outcome of the program.

Important Note

- On the slides in this presentation, I have often cropped screenshots so that I can easily fit everything on one slide
- However, the first screenshot should show ALL the code in the function and second should show at least some of the surrounding code.

a

```
int check_overall(char current_grid[6][7], char player)
{
    for (int i = 0; i < 6; i++)
    {
        for (int j = 0; j < 7; j++)
        {
            if (current_grid[i][j] == player)
            {
                if (check_individual(i, j, current_grid) == 1)
                {
                    return 1;
                }
            }
        }
    }
    return 2;
}
```

- includes a program code segment of a **student-developed algorithm** that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

3.c.iv: The function (check_overall) is given an argument "current_grid" which is the updated array "grid" after each players turn and the argument "player" which is the player whose turn it was. It then works by using a "for loop" that runs six times, then nesting another "for loop" inside that runs seven times. The outer loop will initialize the variable "i" as 0 and increment by +1 each time it runs, while the inner loop will initialize the variable "j" as 0 and increment by +1 each time it runs. In the inner loop, If the piece belongs to "player", check_overall will call the check_individual function and pass in "i," "j," and "current_grid" as arguments. This function will access the item current_grid[i][j] and will determine whether a vertical, horizontal, or diagonal sequence of four pieces of the same owner stems from that position. If it does determine such a sequence, check_individual will return 1 and save it into the "status" variable. Otherwise, check_individual will return 2 and save into "status." As check_overall iterates through all forty-two positions, if "status" is ever 1, the function will return 1 and exit the check_overall function. If "status" never equals 1 after iterating through every position, check_overall will return 2.

?



- includes a program code segment of a **student-developed algorithm** that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

b

3.c.iv: The procedure's parameter accepts values greater or less than 1 when called based on a ask/answer block. The procedure has selection where a value greater than 1 enables the harder difficulty. Both outcomes are similar with some numbers adjusted. There is a timer paired with a repeat until loop and an operator block for the game to continue running for a set amount of time. Sequencing begins when size is included to increase difficulty. Then, the x & y coordinate blocks have a random operator block with a fixed value for the targets to spawn in various locations and not the edge. Finally, once the target has moved, it will appear on screen for a set amount of seconds so it can disappear. After that, it hides and waits while the second wait block gets enabled to control the spawn rates of the targets. If the second wait time is lower than the first, the targets may disappear faster than normal.

?

```
def move_character(event, name, speed):
    global final_score
    name_box = Text(name)
    name_box.set_position(3,15)
    name_box.set_font("10pt Arial")
    add(name_box)
    if event.key == "ArrowLeft":
        character.move((speed*-1), 0)
        for i in range(len(block_list)):
            ....
```

C

- includes a program code segment of a **student-developed algorithm** that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

3.c.iv: It moves the red ball depending on which mouse key the user inputted through the use of selection. If the user pushes the right key the ball moves to the right a certain amount, if user pushes the left key it moves to the left, if user enters the up key the ball moves up, and if user pushes the down key it moves down. After each time the ball moves, the procedure uses a for loop and if statements to check if the ball is on a coordinate of the walls, or the score items such as the stars, by accessing the lists that these coordinates are located in and by checking if even one of those coordinates matches the balls current coordinates. If the coordinates of the ball matches a coordinate of the wall, the procedure lets the user know that they bumped into a wall and makes them restart with zero points, however if the ball's coordinates match one of the coordinates of the score items, then it lets the user know that they got some points. It keeps track of these points by appending a number to a list each time the user gets a point. It then multiplies the length of the list by the amount of points of each item, and stores that in a variable which is the user's final score.

?


```
function getMovie(genre) {
  if (age <= 12) {
    if (genre == "Action") {
      for (var ya = 0; ya < 3; ya++) {
        ...
      }
    }
  }
}
```

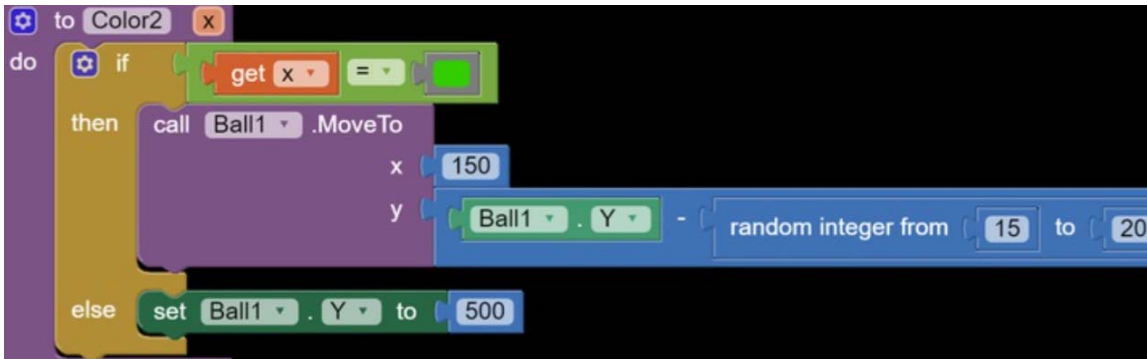
d

- includes a program code segment of a **student-developed algorithm** that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

3.c.iv. Using an if statement, it looks at the user's age. If the user is less than or equal to 12 years old, regardless of their genre, they will be suggested movies from either of the youth lists. If the user is NOT younger than 12 and they are between the ages of 13 and 19, they will be suggested movies from either of the teen lists. And if they are still not in either of those age categories, their movies will be pulled from either of the adult lists. After implementing one's age, the program then looks to see what genre the user chose and from there, using an if statement (selection), will decide if it is pulling movies from the age-appropriate action or comedy list. From there it uses an iteration to randomly select and append three different movies from that appropriate age and genre list to the empty list movie. And from there, using sequencing, the user is sent to the movieOutputScreen and their suggestions are displayed.

?

e



- includes a program code segment of a **student-developed algorithm** that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

3.c.iv.

If the screen is green then it will call ball 1 to move by any number 15-20, if the screen is red then it will set the balls Y value to 500.

?

f

```
function findTotal(hours) {  
  var total = 0;  
  for (var i = 0; i < hours.length; i++) {  
    total = total + hours[i];  
    setText("totaloutput", "You've listened to " + (total + " hours of music this week"));  
    if (total > 27) {  
      setText("iftotal>27", "Wow! You've already listened to more music than the average person per  
week! ");  
    }  
  }  
}
```

3.c.iv.

- includes a program code segment of a **student-developed** algorithm that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

First, when the function is called, it creates a variable called “total” and sets it to 0. The function also includes the parameter hours to allow an input to be used in the function. Then, using a for loop to go through every item in the empty list hours, if the inputted hour number is equal to the hours at the specific index it becomes the new total. The text is then set to explain the number of hours the user has inputted. An If statement is then used, if the total is larger than 27 then another set text is used in order to display that the user has listened to more than the average amount of music listened to per week.

?

```
function filter(userChoice) {
  var filteredList = [];
  var output = "";
  for (var i = 0; i < quoteList.length; i++) {
    if (quoteType[i] == userChoice) {
      appendItem(filteredList, quoteList[i]);
      output = output + quoteList[i] + "\n";
    }
  }
  setText("outputText", output);
}
```

g

- includes a program code segment of a **student-developed algorithm** that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

3.c.iv. The task my function accomplishes is filtering my quotes, sorting them into an empty list, and then outputting that list. My function begins by creating an empty list called `filteredList` (line 14) which will be used to collect the desired quote type. I also created a blank variable called `output` (line 15) which is used in tandem with `filteredList` to create a neat display when the list is output. In order to filter the list, a for loop is used (line 16) which traverses list `quoteType`. The if statement on the next line (line 17) checks to see if the quote type matches the user selection. If they do match, the item is appended to our empty list, `filteredList` (line 18). Next, on line 19, my variable `output` is defined, which includes adding a line break for better display on the user interface. The output is then displayed onto the screen by using `setText` on line 22.

?

```
function housevssenate(input) {
  var stateNames = getColumn("Female Senate Total");
  var senateTotals = getColumn("Female Senate Total");
  var houseTotals = getColumn("Female House Total");
  var cumulativeSenate = 0;
  var cumulativeHouse = 0;
  for (var i = 0; i < stateNames.length; i++) {
    if (stateNames[i] == input) {
```

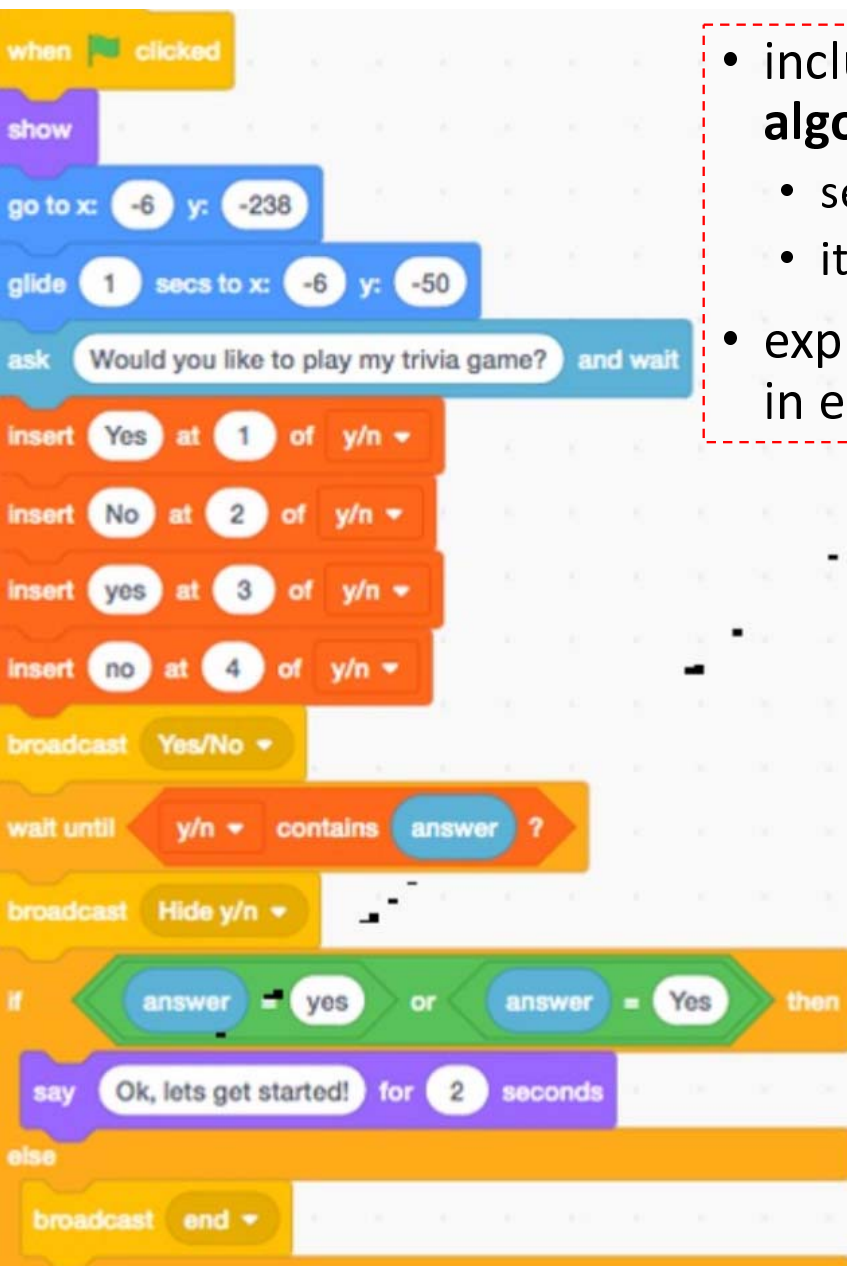
....

h

- includes a program code segment of a **student-developed algorithm** that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

3.c.iv. For the program output that the House or the Senate has more female representation from 1981 to 2019 in the list for a particular state, the program first has to choose three columns in the list: State, Female Senate Total, and Female House Total; these three columns started the function with a parameter named input because the user has to input a state. Two variables were created that stated the value for the Female Senate Total and Female House Total were starting out as zero. The program then includes a for loop. This ensures that the name of the state chosen by the user fits the program's appropriate length. Then, the if statement which is in the for loop adds up the Female Senate Total and Female House Total columns separately. After this, the if else-if else statement decides whether the sum of the Female Senate Total column is greater than the Female House Total column and vice versa. Based on this, the app tells the user that the Senate or House has more female representation or that there is equal representation.

?



- includes a program code segment of a **student-developed algorithm** that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

- 3.c.iv: Within the code, I used the different variables of yes and no to determine the user's participation in the game. I set 4 different versions of yes and no including the capitalized versions of the words so they will still be accepted if typed. Then with these variables, I was able to dictate through code whether or not the user wanted to play by only allowing them to move on if they typed, "Yes" or "yes". Alternatively, if the user were to answer "No or "no" the code would broadcast to end the procedure with the message "Ok, have a great day :P".

```

def Mainprogram ():
    strikes = 0
    points = 0
    repeat1 = 0
    repeat2 = 0
    repeat3 = 0
    repeat4 = 0
    repeat5 = 0
    print( "Welcome To find the Villian" )
    print( "Choose a scenery. All scenery will be location beach" )
    print("1. This option will start the program at the menu screen" )
    print("2. The villian was able to escape the location space" )
    print("3. The villian is hiding in the location city" )
    print("4. This location would be the location of the boat" )
    print("5. this will give you a chance to win the program" )
    print("6. option 6 will close the program" )
    while True:
        if strikes == 3:
            print("sorry stone was able to escape" )
            break

```

- includes a program code segment of a **student-developed algorithm** that includes:
 - selection
 - iteration
- explains in detailed steps how the identified algorithm works in enough detail that someone else could recreate it.

3.c.iv: The algorithm works by using a point system that once reached will end the game. There are three ways to end the program which would be winning the program or losing the program, or ending the program at the menu screen. There are five options that will each have a list connected to it and there are five elements in each list. The program has a while true loop to make sure that the points or strikes will end the game and also the program shall end once option to end the program if needed was used. Under the procedure if any element is typed in and it is not in one of elements within the five list the program will start to add points into strikes and once gaining enough end the program and giving you line about the fail. The menu is also what needs to be printed to give everyone their options about the program.

j

?