

Create Performance Task

Row 6 - Testing

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 **1** in the bottom right corner if the response gets the mark for this row (*all criteria have been satisfied*), otherwise write a **0**.
5. When finished, save this presentation as a pdf with 2 slides to a page and submit this file.

Scoring Criteria

- The written response:
 - **describe two calls** to the selected **procedure identified in written response 3c**. Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
 - Notes:
 - For this to be true the procedure must require “at least one parameter that has an effect on the functionality of the procedure” (*as mentioned in Row 4 - Procedural Abstraction*).
 - The procedure does not necessarily have to be student developed but one would assume that the code must be a “student developed algorithm”.
 - The easiest way to do this is to use some form of *if* structure, where an *if* condition directly, or indirectly, involving at least 1 procedure parameter is *true* with one call but *false* with the other (*so that the if body code is only executed when the if condition is true but not with the other*).
 - A loop is possible, but the loop would have to iterate with one call but not at all with the other call (*so that the loop body code is executed with one call but not with the other*).
 - The 2 calls, the conditions being tested, and the identified results must be true in relation to the identified procedure.
 - **describes the condition(s) being tested** by each call to the procedure.
 - identifies the **result of each call**.

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.
- Do NOT award a point if any one or more of the following is true:
 - A procedure is not identified in written response 3c or the procedure does not have a parameter.
 - The written response for 3d does not apply to the procedure in 3c.
 - The two calls cause the same segment of code in the algorithm to execute even if the result is different.
 - The response describes conditions being tested that are implausible, inaccurate, or inconsistent with the program.
 - The identified results of either call are implausible, inaccurate, or inconsistent with the program.

My Recommendations:

1. **Calls:** What **specific** parameter values will you use?

- *e.g. Instead of stating something general like “a valid/invalid number in/outside of the range 0 – 20”, state actual specific values like **5** and **-1**.*

2. **Conditions:** What do you believe “**should**” happen?

3. **Results:** What happened?

- Use the past tense (*e.g. “-ed”*) and avoid using the future or present tense, or predicting (*e.g. avoid using the words “will”, “should” or “would”*).
 - *e.g. Instead of writing “... will get a card” or “... will definitely lose”, write “... **got a card**” and “... **loses**”.*
- **However, as you will see, not following these recommendations does not automatically lead to a point deduction but doing so will help you fit the requirements more comfortably.**

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 **First call:** The function is called after the first turn when the array "grid" has been updated so that all positions have the status " " except for one position which has status "X". "grid" is then passed into check_overall as argument "current_grid" and "X" is passed as argument "player."

Second call: In another instance where the function is called, player input has updated the array "grid" such that Player O has four "O" pieces in an uninterrupted vertical line. The status of all other pieces are determined by previous player inputs such that no other "connect-four's" exist. This new "grid" is then passed into check_overall and "O" is passed as argument "player."

Condition(s) tested by first call: The first call tests whether four positions in a row of any kind (horizontal, etc.) are occupied by "X."

Condition(s) tested by second call: The second call tests whether four positions in a row of any kind are occupied by "O."

Results of the first call: In the first call, due to Player X not having a four-in-a-row, the function returns 2 which does not satisfy the "if" statement in the main function and enters the "else" protocol, proceeding to the next player's turn.

Results of the second call: In the second call, the function identifies the existence of four-in-a-row for Player O and returns 1 which satisfies the "if" statement in the function. This sets the "turn" variable to 0, breaking the loop and ending the game.

```
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;
}
```

- **describe two calls** to the selected **procedure identified in written response 3c**. Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
- **describes the condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

?



First call: The first call is hard difficulty.

b

Second call: The second call is easy difficulty

Condition(s) tested by first call: If the user inputs the number 2, then the procedure will run the “then” portion of the command and not the “else” portion because the value is greater than 1. Hard Difficulty is enabled.

Condition(s) tested by second call: If the user inputs the number 0, then the procedure will run the “else” portion of the command and not the “then” portion because the number is less than 1. Easy Difficulty is enabled.

Results of the first call: Includes a smaller size for targets, a greater boundary to spawn since the sprites are smaller, and a smaller initial wait time for how long a target will stay on screen.

Results of the second call: Includes a larger size for targets, a smaller boundary to spawn since the sprites are bigger, and a bigger initial wait time for how long a target will stay on screen.

- **describe two calls to the selected procedure identified in written response 3c.** Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
- **describes the condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

?

C

First call: The two test cases are based on passing a different amount by which the red ball is moved each time the specific mouse keys are clicked, in order to create different speeds for the character. The first call is `move_character(event, "Player: " + player, 5)`.

Second call: `move_character(event, "Player: " + player, 15)`

Condition(s) tested by first call: The if-statement on line 385 which checks if the speed parameter is equal to 5 or if it equals 15, will execute the first part of this selection statement, which consists of lines 386-391.

Condition(s) tested by second call: The if-statement on line 385 which checks if the speed parameter is equal to 5 or if it equals 15, will execute the second part of this selection statement, which consists of lines 393-398.

Results of the first call: First call results in the points of each the stars and mystery blocks to be 50, and displays this at the top of the screen. It also displays that the difficulty of the game is medium at this level, and with this argument the red ball moves in increments of 5.

Results of the second call: Second call results in the points of each the stars and mystery blocks to be 100, and displays this at the top of the screen. It also displays that the difficulty of the game is hard at this level, and with this argument the red ball moves in increments of 15.

```
def move_character(event, name, speed):
    ...
    if speed == 5:
        final_score = (len(score_list))*50
        add(level_text)
        add(star_label)
        add(slash)
        add(box_label)
        add(equals)
    elif speed == 15:
        final_score = (len(score_list))*100
        add(level_text_two)
        add(star_label_two)
        add(slash_two)
        add(box_label_two)
        add(equals_two)
```

- **describe two calls** to the selected **procedure identified in written response 3c**. Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
- **describes the condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

?

d First call: When the user clicks on the word "Comedy" on the genreChoiceScreen, genre gets "Comedy." When this genre is chosen, the code will search through the comedy list (for the particular age given) in order to suggest movies: `getMovie("Comedy")`


Second call: When the user clicks on the word "Action" on the genreChoiceScreen, genre gets "Action." When this genre is chosen, the code will search through the action list (for the particular age given) in order to suggest movies: `getMovie("Action")`

Condition(s) tested by first call: Dependent on if the user clicks on the text, the code will iterate through that chosen genre and pull out random movies from the corresponding list: `genre=="Comedy"`

Condition(s) tested by second call: Dependent on if the user clicks on the text, the code will iterate through that chosen genre and pull out random movies from the corresponding list: `genre=="Action"`

Results of the first call: The user is suggested three movies from one of the three different Comedy lists (which is dependent on the user's age).

Results of the second call: The user is suggested three movies from one of the three different Action lists (which is dependent on the user's age).



```
function getMovie(genre) {
  if (genre == "Action") {
    for (var ya = 0; ya < 3; ya++) {
      appendItem(movie, youthAction [random]);
    }
    setScreen(▼ "movieOutputScreen");
    setText(▼ "movieOutputText", (movie));
  } else if (genre == "Comedy") {
    for (var yc = 0; yc < 3; yc++) {
      appendItem(movie, youthComedy [random]);
    }
  }
}
```

- **describe two calls** to the selected **procedure identified in written response 3c**. Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
- **describes the condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

?

e

First call: Color2 X=green

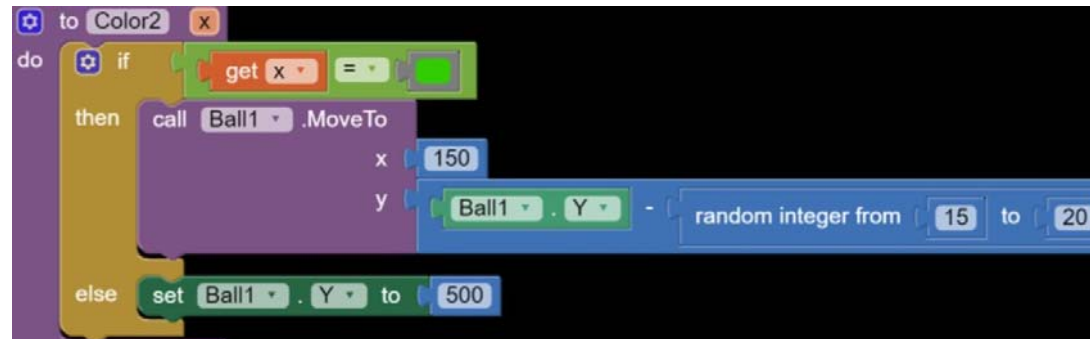
Second call: Color2 X=Red

Condition(s) tested by first call:

The condition tested in the call is the color of the screen.

Condition(s) tested by second call:

The condition tested in the call is the color of the screen.



Results of the first call: If it is green the ball moves forward.

Results of the second call: If it is red then the ball would be sent back to the bottom of the screen.

- **describe two calls** to the selected **procedure identified in written response 3c**. Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
- **describes the condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

?

f First call: The user adds the numbers 5,1,7,3,2,1 and 4 to represent the total hours they've spent listening to music.

Second call: The user adds the numbers 6,4,9,5,3,2, and 5 to represent the total hours they've spent listening to music.

Condition(s) tested by first call: For the first call, the function will calculate the numbers 5,1,7,3,2,1, and 4, and find that the total is 23 hours. The if statement then checks to see if the total is greater than 27. In this call, the number isn't greater than 27, so no text will appear other than the total.

Condition(s) tested by second call: For the second call, the function will calculate the numbers 6,4,9,5,3,2, and 5 and find that the total is 29 hours. The if statement then checks to see if the total is greater than 27. In this call, the number is greater than 27, so the total will appear as well as a new text that says "Wow! You've already listened to more music than the average person per week!"

Results of the first call: After the for loop finishes running, the user will see "You've listened to 23 hours of music this week" in the first text area.

Results of the second call: After the for loop finishes running, the user will see "You've listened to 29 hours of music this week" in the first text area and "Wow! You've already listened to more music than the average person per week!" in the second text area.

```
function findTotal(hours) {  
  var total = 0;  
  for (var i = 0; i < hours.length; i++) {  
    total = total + hours[i];  
    setText("totaloutput", "You've listen  
    if (total > 27) {  
      setText("iftotal>27", "Wow! You've
```

- **describe two calls** to the selected **procedure identified in written response 3c**. Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
- **describes the condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

?

g

First call: *filter*("movie line");

Second call: *filter*("song lyric");

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

Condition(s) tested by first call: The argument passed through the filter function in this case would be “movie line”. An if loop in my procedure tests to see whether the user choice calls for movie lines or song lyrics.

Condition(s) tested by second call: The argument passed through the filter function in this case would be “song lyric”. An if loop in my procedure tests to see whether the user choice calls for movie lines or song lyrics.

Results of the first call: Movie lines are selected and added to my empty list named filteredList. This list is then output into the text box on the screen.

Results of the second call: Song lyrics are selected and added to my empty list named filteredList. This list is then output into the text box of the screen.

- **describe two calls** to the selected **procedure identified in written response 3c**. Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
- **describes the condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

?

h First call: The user could select Arizona as the state in the dropdown see whether the House or the Senate had more female representation between 1981 and 2019.

Second call: To find whether the House or the Senate had more female representation between 1981 and 2019 in a particular state, the user could select Delaware as the state in the dropdown.

Condition(s) tested by first call: When the user selects Arizona as the state, the program will add up the numbers in the Female Senate Total and Female House Total columns and compare the two.

Condition(s) tested by second call: After the state, Delaware, is selected by the user, the program will find two columns: Female Senate Total and Female Total columns. Then, it will add the values in the two columns and see which one is higher/lower.

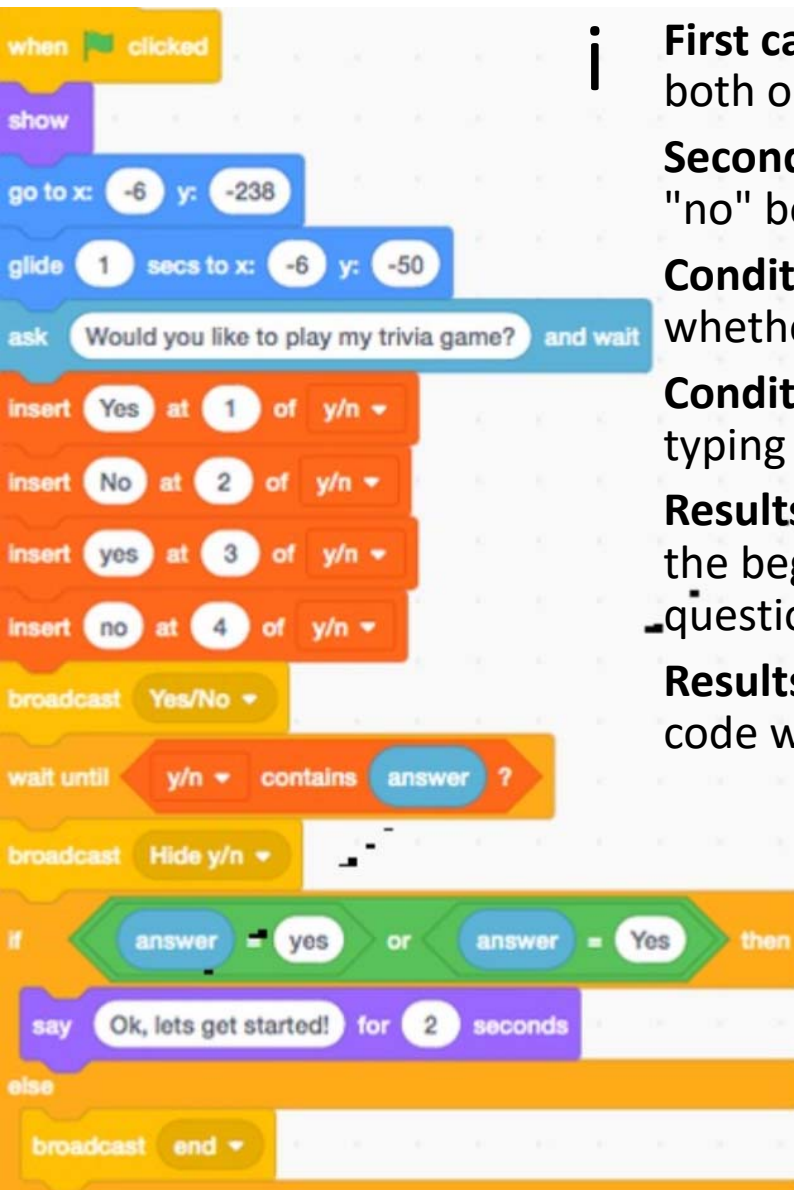
Results of the first call: After the program adds the two columns, it will realize the number in the Female House Total is larger than the Female Senate Total, so in the text area, the word House will display for the user to show that there is more female representation in the House versus the Senate for that particular state in between 1981 and 2019.

Results of the second call: When the program finds the sum of the two columns, it will come to the conclusion that both of the values in the columns are equal, and it will display Equal Representation for the user.

- **describe two calls** to the selected **procedure identified in written response 3c**. Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
- **describes the condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

```
function housevsenate(input) {  
  var stateNames = getColumn("Female State Legislators  
  var senateTotals = getColumn("Female State Legislato  
  var houseTotals = getColumn("Female State Legislator  
  var cumulativeSenate = 0;  
  var cumulativeHouse = 0;  
  for (var i = 0; i < stateNames.length; i++) {  
    if (stateNames[i] == input) {  
      cumulativeSenate = cumulativeSenate + senateTota  
      cumulativeHouse = cumulativeHouse + houseTotals[  
    }  
    if (cumulativeSenate > cumulativeHouse) {  
      setText("stateResults", "Senate");  
    } else if (cumulativeSenate < cumulativeHouse) {  
      setText("stateResults", "House");  
    } else {  
      setText("stateResults", "Equal Representation");  
    }  
  }  
}
```

?



First call: In the code, the first call is the option to type "Yes" or "yes" both options are accepted as answers.

Second call: In the code, the second call is the option to type "No" or "no" both options are accepted as answers.

Condition(s) tested by first call: By using a list I applied the condition of whether the user will type "yes" or "Yes"

Condition(s) tested by second call: Using this same list the condition of typing "no" or "No" is tested

Results of the first call: If the first call was acted upon it would trigger the beginning of the game for the user to answer the five preset questions.

Results of the second call: If the user typed either variation of no the code would have ended with a short message after their response.

- **describe two calls to the selected procedure identified in written response 3c.** Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm **to execute**.
- **describes the condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

?

```

def Mainprogram ():
    strikes = 0
    points = 0
    repeat1 = 0
    repeat2 = 0
    repeat3 = 0
    repeat4 = 0
    repeat5 = 0
    print( "Welcome To find the
chose a scenery. All scenery
print("1. This option will
be location beach
print("2. The villian was a
location space
print("3. The villian is h
location city
print("4. This location wou
boat
print("5. this will give yo
print("6. option 6 will clo
program
while True:
    if strikes == 3:
        print("sorry stone was
        break

```

First call: The first call of the procedure starts the program and will give the user multiple options to move forward within the program. This gives the user all options so that once the program is started the options will be the next step the user would need to use to end the program.

Second call: The second call will give the options a key on the keyboard which will be numerical and start the game part of the program. This will give all the numerical options that are on the screen of the program a value and will connect you to a scenery that the user will need to be able to play to move on to another scenery by wining.

Condition(s) tested by first call: The procedure will test that once at a certain point the program will due to the user wining, losing , or ending the program if the user would like to.

Condition(s) tested by second call: The conditions tested were if a element was not part of the five lists it would give you a point into strikes and will start giving the user points to either win or lose the game part of the program.

Results of the first call: the result of the first call will tell the user if you win or lose.

Results of the second call: the result of the second call will give the user its option to play and win or lose or to end the program early.

- describe two calls to the selected procedure identified in written response 3c. Each call must pass a **different argument(s)** that **causes a different segment of code** in the algorithm to execute.
- describes the **condition(s) being tested** by each call to the procedure.
- identifies the **result of each call**.

?