

Name _____ Period _____ Role (Circle one) Programmer/Driver

Name _____ Period _____ Role (Circle one) Programmer/Driver

If-Else Statements

Your Tasks (Mark these off as you go)

- ☐ Interpret if-else statement pseudocode
- ☐ Have Ms. Pluska check off the above tasks
- ☐ Write an if-else statement
- ☐ Write an if-else-if statement
- ☐ Brainstorm a program
- ☐ Receive credit for the group portion of this lab

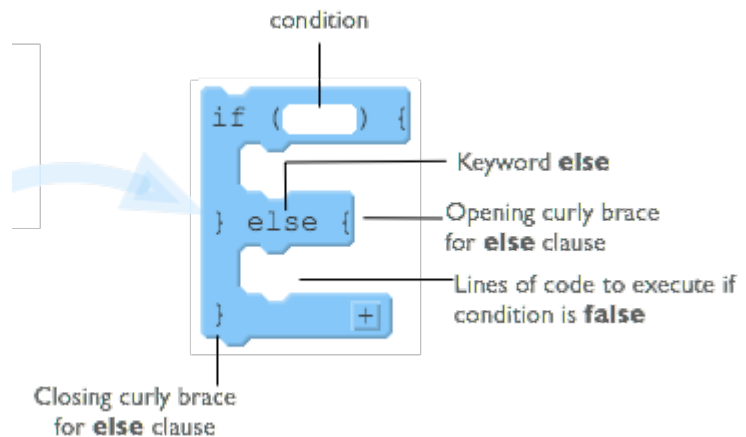
☐ Interpret if else statement pseudocode

To the right is a diagram showing the elements of a basic *if-else* statement in JavaScript.

With an *if-else* statement you are giving an either-or command:

- **either** the lines of code inside the *if* will execute
- **or** the lines inside the *else* will execute.

Inside the curly braces for the *else* clause you put lines of code that you want to run if the Boolean condition from the *if* statement is false.



Some important notes about the else clause:

- The *else* must come immediately after the closing curly brace of an if statement
- The *else* also has its own set of opening and closing curly braces to encapsulate lines of code

Before we get started writing *if-else* statements, let's practice some pseudocode scenarios.

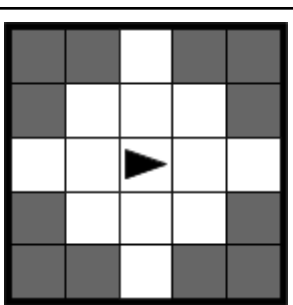
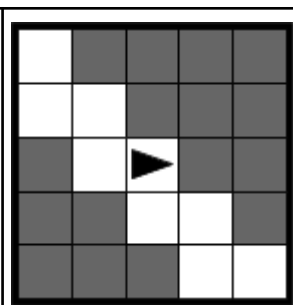
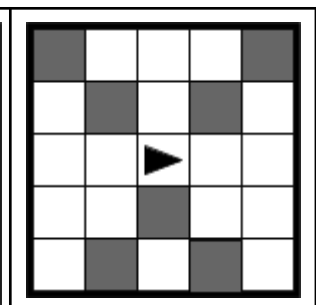
Each row in the table below presents a small program that uses if-else statements and robot commands. Trace the code and plot the movements of the robot for the 3 scenarios shown to the right of the code. If the robot is directed to move onto a black square, it "crashes" and the program ends. If the robot doesn't crash, then draw a triangle showing its ending location and direction.

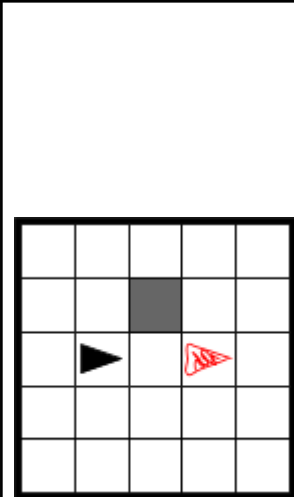
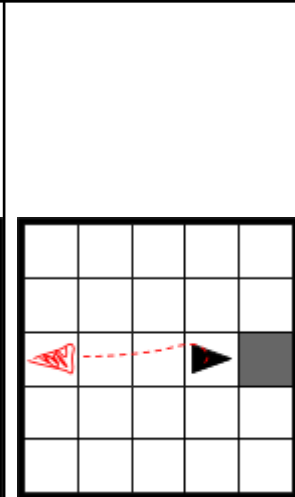
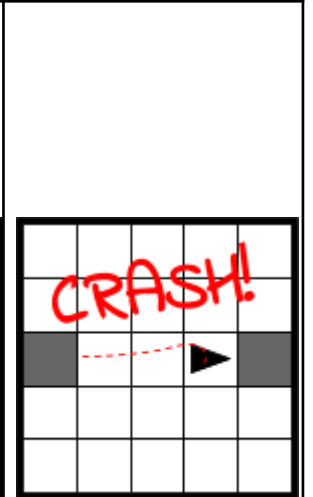
There are a few patterns to the ways if-statements are typically used. We explored several of these in previous lesson.

- ☐ Basic If-statements
- ☐ Sequential If-statements
- ☐ Basic if-else statements
- ☐ Nested If and if-else statements
- ☐ Combinations of all of the above

Each section below presents an example of one of the common patterns, followed by a few problems for you to try. For each type **study, and make sure you understand, the example** and why each of the 3 scenarios ends up in the state shown.

EXAMPLE: If-else Statement			
<p>The code in the if-block executes <i>ONLY</i> if the expression is true, otherwise the code in the else block will run. But one or the other <i>must</i> execute. An else statement can be attached to a single if-statement.</p>			
<pre> ROTATE_LEFT () IF (CAN_MOVE (forward)) { MOVE_FORWARD () } ELSE{ ROTATE_LEFT () ROTATE_LEFT () } MOVE_FORWARD () </pre>			
YOU TRY IT - Simple If-Else			
<p>Here is a block-based version of a very similar program.</p> <pre> MOVE_FORWARD IF CAN_MOVE forward MOVE_FORWARD ELSE ROTATE_LEFT MOVE_FORWARD ROTATE_RIGHT MOVE_FORWARD MOVE_FORWARD </pre>			

<pre> IF (CAN_MOVE (left)) { ROTATE_LEFT () MOVE_FORWARD () } ELSE { ROTATE_RIGHT() MOVE_FORWARD () } IF (CAN_MOVE (right)) { ROTATE_RIGHT () } ELSE { ROTATE_LEFT () } MOVE_FORWARD () </pre>			
--	--	---	--

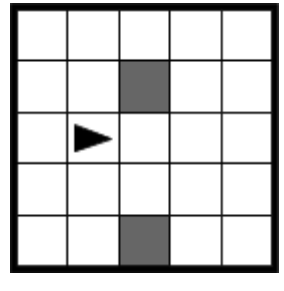
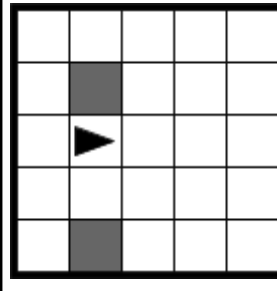
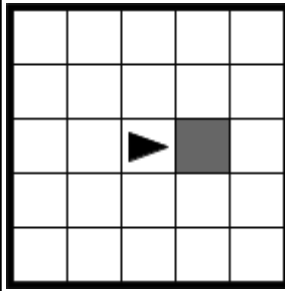
EXAMPLE: Nested Statements			
<p><i>You can put if- and if-else statements inside other if-statements. All previous rules apply, but tracing the code can be tricky.</i></p> <pre> IF (CAN_MOVE (forward)) { MOVE_FORWARD () } ELSE { IF(CAN_MOVE(backward)) { ROTATE_LEFT () ROTATE_LEFT () MOVE_FORWARD () } MOVE_FORWARD () } MOVE_FORWARD () </pre>			

YOU TRY IT - Nested If-statements

```

IF (CAN_MOVE (forward))
{
  IF (CAN_MOVE (left))
  {
    ROTATE_LEFT ()
  }
  ELSE{
    ROTATE_RIGHT ()
  }
}
MOVE_FORWARD ()
}
ELSE
{
  ROTATE_LEFT ()
  ROTATE_LEFT ()
}
MOVE_FORWARD ()

```



□ Have Ms. Pluska check off the above tasks



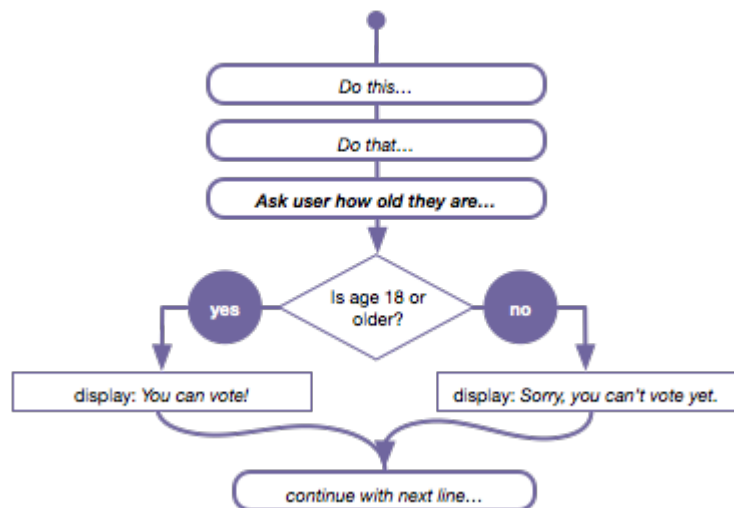
Before you continue have Ms. Pluska check off the above tasks

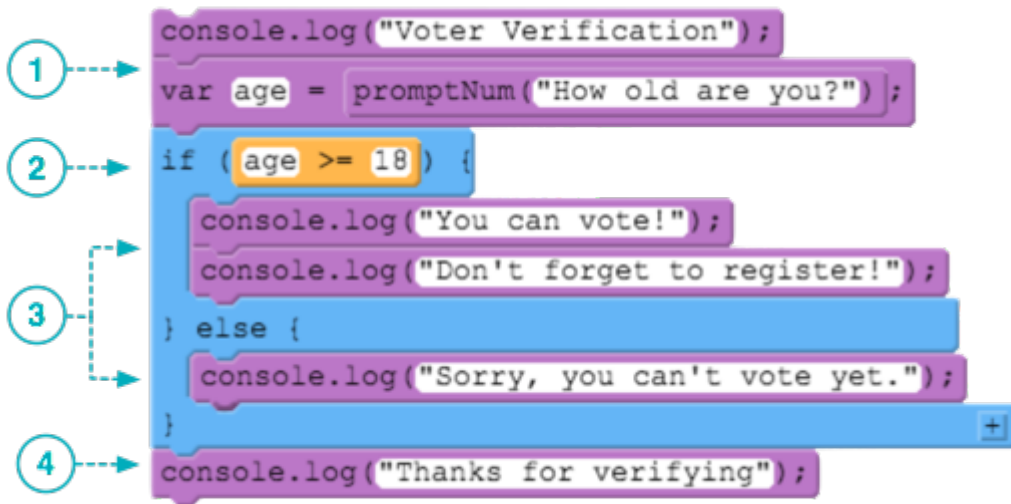
Do not continue until you have Ms. Pluska's (or her designated TA's) signature _____

□ Write an if-else statement

Consider our flow chart from before. Until now we haven't had a way to make the program do something different if the condition was false. With an *if-else* statement we do. We can now write a program that "branches" at a particular point, running one of two possible sections of code.

Below is a worked example for how the example to the right could be written in javascript,





1. Lines of code execute sequentially as usual. Prompt the user to enter their age.
2. The *if* statement and Boolean expression are also the same as before. The expression evaluates to either true or false.
3. With an *if-else* statement you are guaranteeing that exactly one of these two sections of code will execute. If the condition is true (age is 18 or greater) then the lines of code inside the *if*-statement's curly braces are executed. If the condition is false it jumps to the *else* clause and executes any lines of code it finds between the *else* clause's curly braces.
4. Finally the program picks up normal execution directly after the *if-else* block. At this point in the program, we know that either the code in the *if*-block or the *else* block has executed.

Below is the start of a program. Write an *if-else* statement that checks whether or not someone is old enough to drive. If they are 16 or older, indicate to the user that they are old enough to drive. Otherwise, indicate to the user how long they need to wait until they can drive.

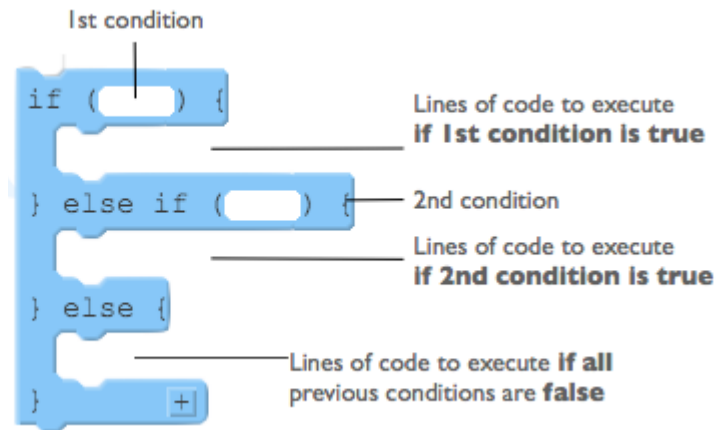
```
var age = prompt("What is your age");
```

□ Write an if-else-if statement

How it works

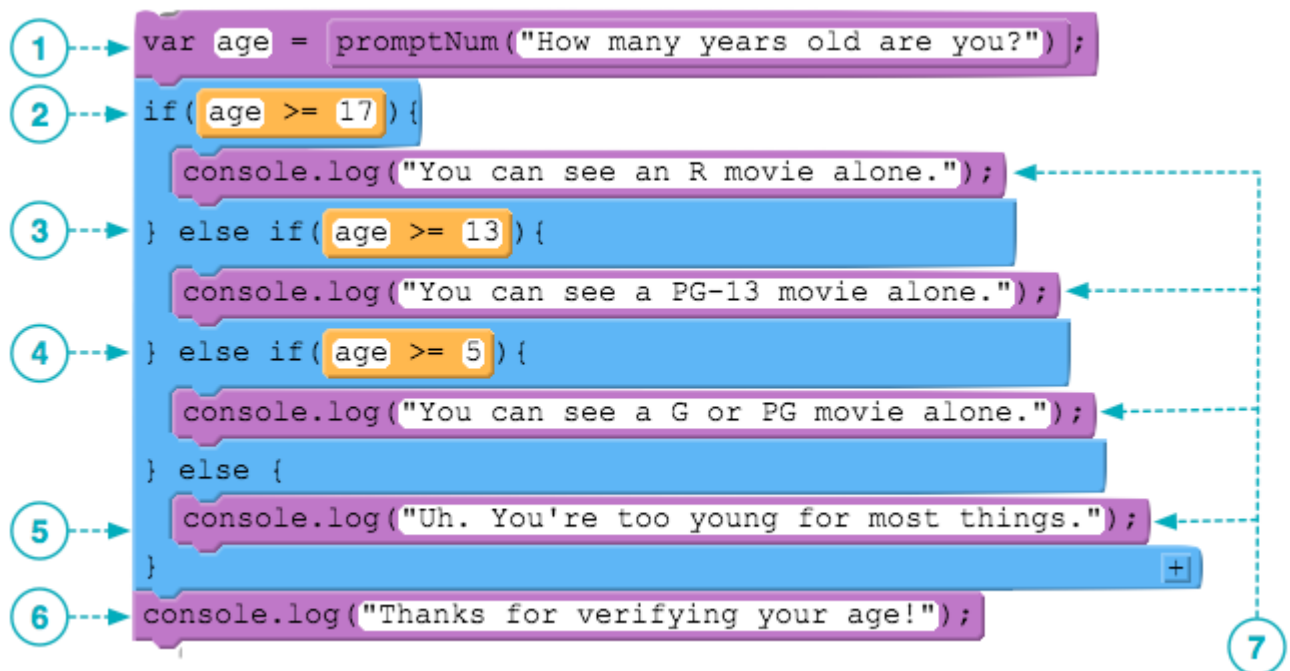
Not all conditions you want to check have only two possible outcomes. However a computer can only check one true/false condition at a time.

- You add an *else-if* clause to an *if* statement when you have another condition you want to check.
- You can add as many *else-if* clauses as you want.
- Each condition in an *if-else-if* is checked in order from top to bottom and the final *else* clause is executed if all the previous conditions evaluated to false.



An Example

Below is an example,



1. Ask the user to enter their age and save it in a variable.
2. First check to see if the age is 17 or over. If it is they can see an R-rated movie.
3. If we reach this condition it means that the previous condition was false. So now let's check if the age is 13 or over and display a message.
4. To reach this statement means that, so far, the previous conditions we've checked have come up false. So now let's check if the age is 5 or over and display a message.
5. If we reach the final *else* clause it means all the previous conditions came up false. So this is what gets displayed.

6. Execution picks up on the first line after the if-else-if block. This "thank you" message at the end will display no matter what.

7. Because there is a final else clause you are guaranteeing that exactly one of these statements will execute. It is possible to write a chain of else-if conditions without a final else, in which case it's possible that the whole structure will be skipped.

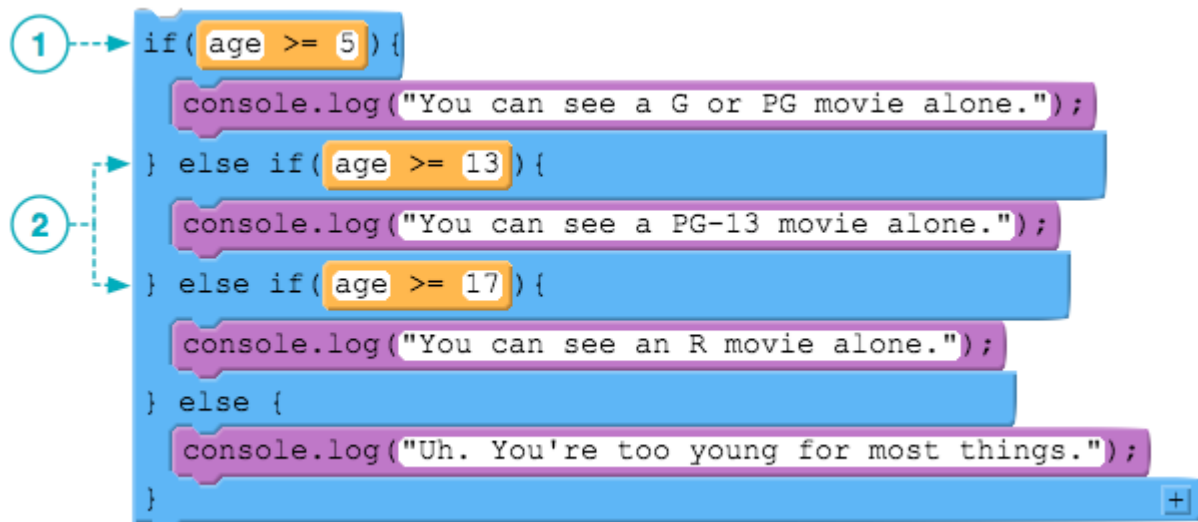
What can go wrong

When writing *if-else-if* there are two common mistakes and thus two things to pay attention to:

The order of the conditions matters!

- An *if-else-if* will execute the code for the first condition that comes up *true* - all other conditions will be ignored.
- An if-else-if statement is like saying, "FIRST check this condition, THEN check this one, THEN check this one and so on."

The common mistake is to accidentally have a condition early in the sequence that makes the other statements impossible to reach. For example, if you changed the order of the conditions in our movie ratings example:



1. This condition is true for any age over 4, so this console.log will execute right away, and the rest of the statements will be ignored.

2. Notice that it's impossible for either of these two conditions to even be reached. We can't check if the age is over 13 because it would have been caught by the first condition.

The misconception that leads to this mistake is to think that the *if-else-if* statement is considered in its entirety before execution - it's not. The conditions are checked sequentially, from top-to-bottom just like everything else.

Without a final else clause it's possible that the whole structure can be skipped

Just like an *if* statement without an *else* is skipped over if the condition is *false*, an *if-else-if* without a final *else* clause is skipped over entirely if none of the conditions is *true*.

Here is a generic example - forgetting what the code inside the *if* statements is supposed to do, just consider the *if-else-if* structure here.

What happens if *val* is 0? (or anything that's not 1, 2, 3, or 4?)

Answer: nothing. If *val* is 0 then each of the conditions will be checked in order, and since none of them is true, nothing will be executed.

This might seem obvious here, but the most common way this happens is that in your program you think the *val* can only be 1, 2, 3, or 4, but something happens to make it be some other value you weren't expecting. It's particularly hard to catch because there are no syntax errors here, nothing will crash due to an illegal operation.

A way to protect yourself is to add a final *else* clause - even if you think it can't be reached - that prints a statement to yourself like "uh oh!"

```
if (val==1) {  
  // ...  
} else if (val==2) {  
  // ...  
} else if (val==3) {  
  // ...  
} else if (val==4) {  
  // ...  
}
```

Consider the prompt below. Write code that could be used to indicate the grade received on the quiz.

```
var score = promptNum("What was your quiz score (0-100)?");
```

```
console.log("Your quiz grade is an A");  
console.log("Your quiz grade is a B");  
console.log("Your quiz grade is a C");  
console.log("Your quiz grade is a D");  
console.log("Your quiz grade is an F");
```


□ Have Ms. Pluska check off the above tasks



Before you continue have Ms. Pluska check off the above tasks

Do not continue until you have Ms. Pluska's (or her designated TA's) signature _____

□ Brainstorm a program

Challenge 1

Consider two cards A and B as defined below. Each card represents a card in a standard 52 deck of cards.

```
var a = jackOfClubs;  
var b = sevenOfHearts;
```

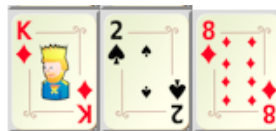


Brainstorm a program that can swap the values of the cards. That is var a becomes the sevenOfHearts and card b becomes the jackOfClubs.

Challenge 2

Now consider three cards with the following values.

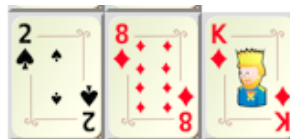
```
var a = kingOfDiamonds;  
var b = twoOfSpades;  
var c = eightOfDiamonds;
```



```
a.value = 13;  
b.value = 2;  
c.value = 8;
```

Brainstorm a program that could be used to sort the cards, as shown below. Your program must reassign the variables a, b, and c to the correct cards.

```
var a = twoOfSpades;  
var b = eightOfDiamonds;  
var c = kingOfDiamonds;
```



□ Receive Credit for the group portion of this lab



- Indicate the names of all group members.
- Make sure both you and your partner have completed the above tasks
- Have Ms. Pluska check off the group tasks
- Submit your lab to the needs to be graded folder to receive credit for the group portion of this lab.
- Do not submit your lab until you have Ms. Pluska's (or her designated TA's) signature
