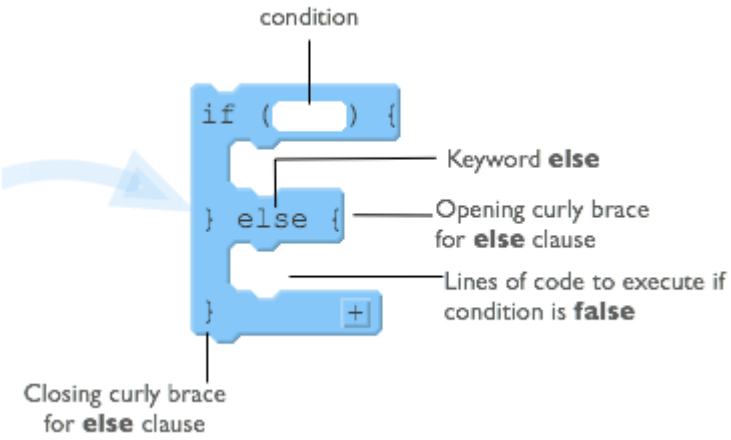
Set 30: If-Else Statements

Skill 30.01: Interpret If-Else statement pseudocode

Skill 30.01 Concepts

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, lets 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.

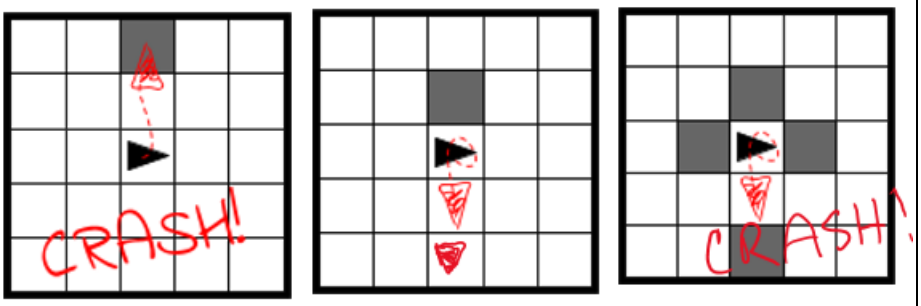
There are a few patterns to the ways if-statements are typically used. We explored several of these in previous lesson.

- 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. For each type **study, and make sure you understand, the example** and why each of the 3 scenarios ends up in the state shown.

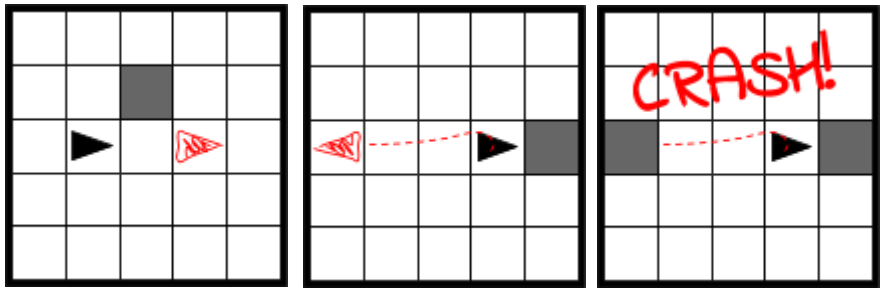
EXAMPLE: If-else Statement

```
ROTATE_LEFT ()
IF (CAN_MOVE (forward))
{
  MOVE_FORWARD ()
}
ELSE
{
  ROTATE_LEFT ()
  ROTATE_LEFT ()
}
MOVE_FORWARD ()
```



EXAMPLE: Nested Statements

```
IF (CAN_MOVE (forward))
{
  MOVE_FORWARD ()
}
ELSE
{
  IF(CAN_MOVE( backward ))
  {
    ROTATE_LEFT ()
    ROTATE_LEFT ()
    MOVE_FORWARD ()
  }
  MOVE_FORWARD ()
}
MOVE_FORWARD ()
```

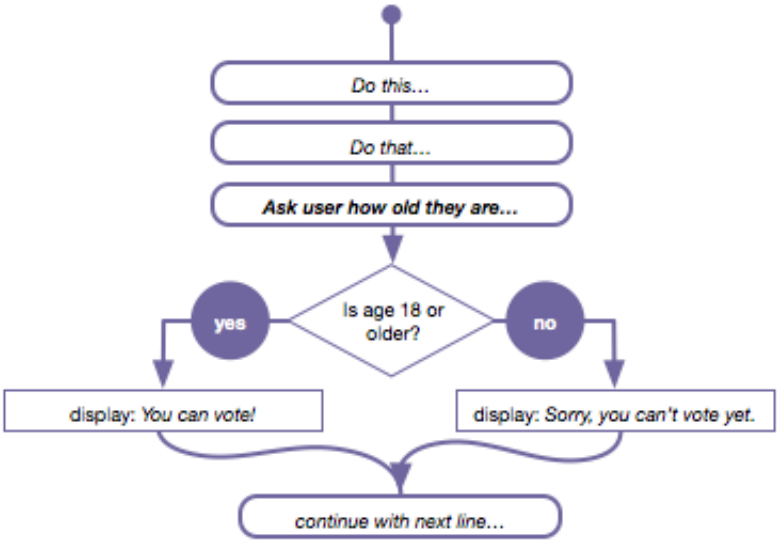


[Skill 30.01 Exercises 1 thru 3]

Skill 30.02: Write and If-Else Statement in JavaScript

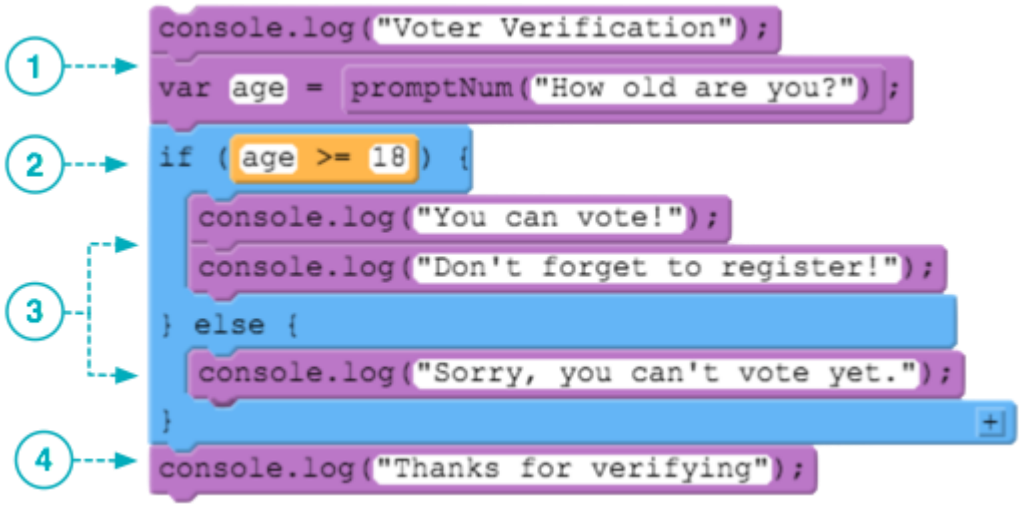
Skill 30.02 Concepts

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,

Consider our flow chart from before. Until now we haven't had a way to make the program do



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.

Skill 30.02 Exercises 1 and 2

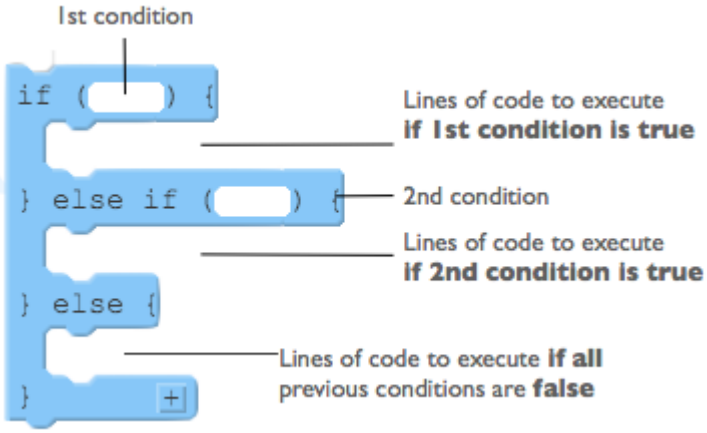
Skill 30.03: Write an If-Else-If Statement

Skill 30.03 Concepts

How it works

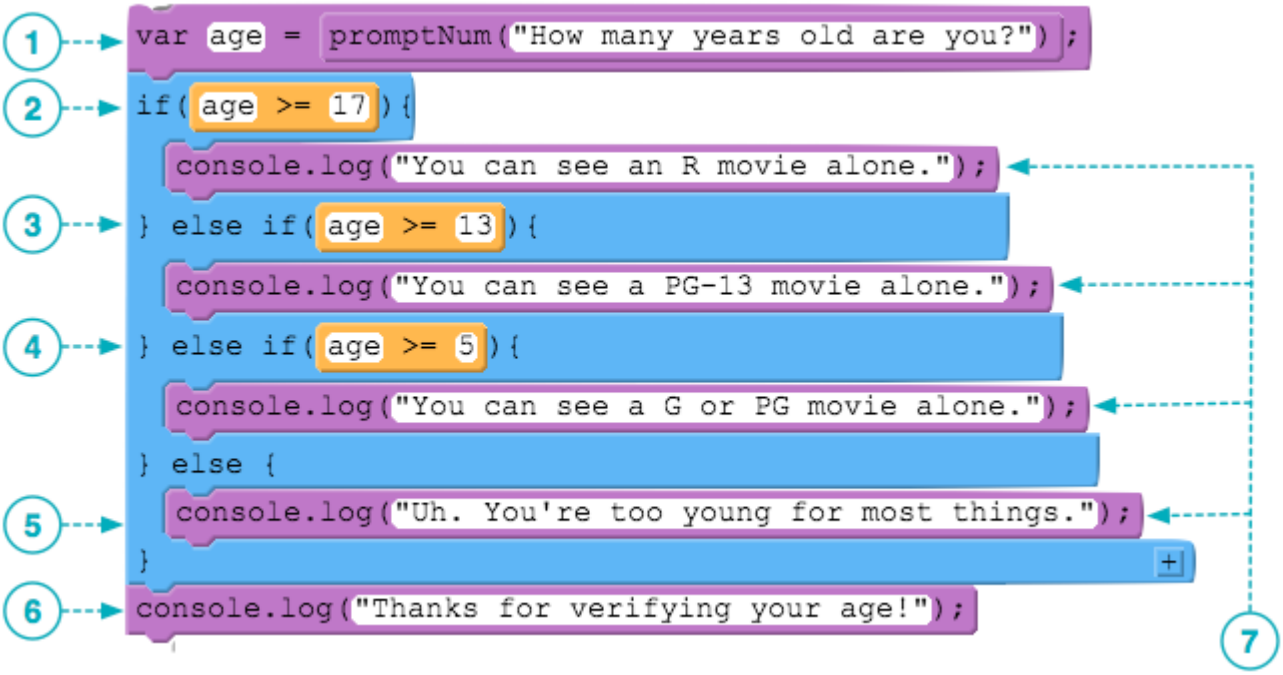
Not all conditions you want to check have only two possible outcomes. However a computer can only check *one true/false condition at a time*.

- 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.

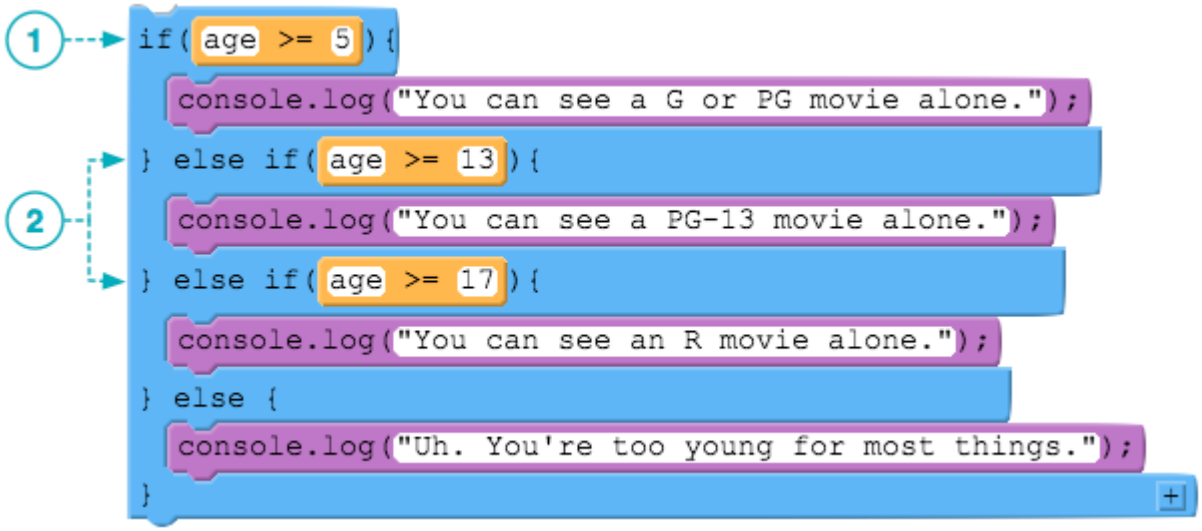
What can go wrong

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:



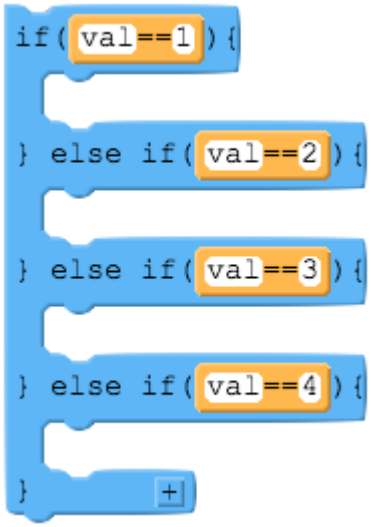
1. This condition is true for any age over 4, so this *console.log* will execute right away, and the rest of the statements will be ignored.

2. Notice that i*t'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* tha *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!"

Skill 30.03 Exercise 1