JavaScript – theory and tasks

Control Flow

Now that we've got the parts of JavaScript, let's see what they can do. In this unit, we'll explore how we can take those parts, and create decision trees, games, and much more.

In programming, making decisions with code is called control flow.

For instance, if we were making a choose-your-own-adventure game, we'd need to program a way for a user to choose which plot line they'd like to pursue. Control flow statements enable JavaScript to make those decisions by executing different code based on a condition.

In this lesson, we'll learn how to make decisions with JavaScript and how it can control the program's flow.

If you think about what we've been doing so far, we've been writing instructions for computers.

That's all programming really is: a list of instructions for computers.

The main difficulty of being a developer is translating our ideas in *human-speak* into ideas in *computer-speak*.

Let's learn how we can ask JavaScript to think like us and make decisions like us.

Let's start with human-speak. Many decisions we make everyday boil down to this sentence in some form:

*"If something is true, let's do option 1, or else, if it is false, let's do option 2."*

This sentence looks fairly similar when we write it with JavaScript. See for yourself:

var needCoffee = true;

if (needCoffee) {

console.log('Finding coffee');

} else {

console.log('Keep on keeping on!');

}

1. If the variable *needCoffee* is *true*, JavaScript will run one code block, and if a variable is *false*, it will run another.
2. *needCoffee* is the condition we are checking inside the *if'*s parentheses. Since it is equal to *true*, our program will run the code between the first opening curly brace { (line 2) and the first closing curly brace } (line 4). It will completely ignore the *else { ... }* part. In this case, we'd see *'Finding coffee'* log to the console.

Note: Code between curly braces are called blocks. *if/else* statements have two code blocks.

1. If we adjusted *needCoffee* to be *false*, only the *else*'s *console.log* will run.
2. *if*/*else* statements are how we can process yes/no questions programmatically.



***Instructions***

*Write your code in the ifElse.js file.*

1. Create a variable named harryPotterFan and set it equal to a boolean, based on your preference.
2. Then, write an if/else statement that uses harryPotterFan as its condition. If it is true, then log, 'Mischief managed.'. If it is false, then log, 'I lead a muggle\'s life.'.

***Note:*** Why is there a \ in 'I lead a muggle\'s life.'? Since the string is surrounded by single quotes, we can use a back slash to add a single quote within the string. This is called escaping a character.

*if/else* statements are made even more powerful with comparison operators.

There are two comparisons you might be familiar with:

* Less than: <
* Greater than: >

You may also recognize these:

* Less than or equal to: <=
* Greater than or equal to: >=

Comparisons need two things to compare and they will always return a boolean (*true* or *false*).

How can we use comparisons and an *if/else* statement to see if it's time to eat?

***Instructions***

*Write your code in the comparison.js file.*

1. Write a variable named hungerLevel and set it equal to 5. (comparison.js)
2. Write an if/else statement that checks if hungerLevel is greater than 7. If so, log 'Time to eat!', and if it is less than 7, log 'Let\'s eat later.'.
3. Notice that since hungerLevel is 5, the if statement evaluates to false, therefore the else's code block runs.

Now set the hungerLevel variable equal to 10 and run your code again.

1. Now it's 'Time to eat!'

We can use the <, >=, and <= to compare variables in an if/else statement, which is a powerful way to make decisions in JavaScript.

Play around with these operators and values to see how exactly they work.

There are two more extremely useful comparisons we can make. Often times, we might want to check if two things are equal, or if they are not.

1. To check if two things equal each other, we can use *===* (three equals in a row).
2. To check if two things do not equal each other, we can write *!==* (exclamation with two equals in a row).

Slow down here, this can be easily confused with creating variables. Variables use one equals sign = to assign a value to a variable. When you're comparing two variables, make sure to use three *===.*

How can we utilize *===* or *!==* and an *if*/*else* statement to write a program to see whether a werewolf will turn into its wolf form this evening?

***Instructions***

*Write your code in the werewolf.js file.*

1. Write a variable named moonPhase and set it equal to 'full'.
2. Write an if/else statement that checks if the moon is full. If the moonPhase is 'full', log 'Howwwwlll!!' to the console, and if it is anything else, log 'I swear I am not a werewolf...'.
3. Notice the code inside the first block of curly braces { } ran. That's because moonPhase equals 'full', and therefore the condition evaluates to true.

*if*/*else* statements are either this or that for us right now. They answer questions that are either yes or no.

What can we do if we have a question that has multiple yes conditions, or multiple no conditions?

We can add more conditions to our *if*/*else* statement with: *else if*. Check out how this fits into our current knowledge of *if*/*else* statements:

var stopLight = 'green';

if (stopLight === 'red') {

console.log('Stop');

} else if (stopLight === 'yellow') {

console.log('Slow down');

} else if (stopLight === 'green') {

console.log('Go!');

} else {

console.log('Caution, unknown!');

}

1. We created a variable named *stopLight* that is assigned to the String *'green'*.
2. Then, there's an *if*/*else* statement with multiple conditions, using *else if*. *else if* allows us to find multiple states of the *stopLight* variable, and output different things based on its color.
3. The whole thing ends with the singular *else* we know and love. The *else* is a catch-all for any other situation. For instance, if the *stopLight* was blinking blue, the last *else* would catch it and return a default message.

***Instructions***

*Write your code in the werewolf.js file.*

1. We all know that turning into a werewolf is not an instant thing. It happens in stages. So let's expand our program from before to accommodate that fact.

If the moon is 'mostly full', log 'Arms and legs are getting hairier' If the moon is 'mostly new', log 'Back on two feet'.

If someone enters in an invalid moon phase, make sure to the final else to log 'Invalid moon phase'.

1. Set moonPhase to 'mostly full' and run your code.

We expect 'Arms and legs are getting hairier' to log to the console.

1. Set moonPhase to 'mostly new' and run your code.

We expect 'Back on two feet' to log to the console.

1. Now set moonPhase to 'solar eclipse' and run your code.

Since there is not an else if condition for 'solar eclipse', we expect the default else code block to run. You should see 'Invalid moon phase' print to the console.

So far, we've been able to translate certain thoughts into JavaScript code, like, "Are these things equal?" with *===,* or, "Is one thing greater than another thing?" with >.

In English, sometimes we say "both of these things" or "either one of these things." Let's translate those phrases into JavaScript with some special operators called logical operators.

1. To say "both must be true," we can use &&.
2. To say "either can be true," we can use ||.
3. To say "I want to make sure this is the opposite of what it really is," we can use !.
4. To say "these should not be equal to each other," we can use !==.

*For example:*

if (stopLight === 'green' && pedestrians === false) {

console.log('Go!');

} else {

console.log('Stop');

}

1. In the example above, we make sure that the *stopLight* is *'green'* and *&&* there are no pedestrians before we log 'Go!'.
2. If either of those conditions are false, we log 'Stop'.

Just like the operators we learned previously, these logical operators will return either true or false.

These logical operators are helpful when writing if/else statements, since they let us make sure multiple variables are true or false.

***Instructions***

*Write your code in the werewolf.js file.*

1. Let's say the werewolf can only become its wolf form when there is a full moon and a it's a foggy night.

We already have a moonPhase variable, so let's start with making a foggyNight variable set equal to true.

1. Now, set moonPhase to 'full' again. Now that we have both conditions, let's write that in our if/else statement.

In the first condition of the if/else statement, make sure that moonPhase === 'full' and foggyNight are true, using &&.

***Note:*** if/else statements check the true-ness or false-ness of variables, so there's not need to write foggyNight === true. You only need to write foggyNight, and you'll get the same result.

1. Now, change the foggyNight variable to equal false and run it again.
2. Notice that the default else will print to the console. That's because && requires both moonPhase and foggyNight to be true to execute its code block.

Now, let's try out ||. Use || in place of the && that you wrote in the previous steps to make the if/else statement print 'Howwwlll!!!' if moonPhase is 'full' or if foggyNight is true.

Leave the moonPhase variable's value as 'full' and the foggyNight variable's value as false.

1. The first if statement's block was executed. That's because moonPhase is 'full', which is true. The || operator only needs one thing to be true in order to return true.

Before we move on, let's circle back to else if statements.

Using else if is a great tool for when we have a few different conditions we'd like to consider.

else if is limited however. If we want to write a program with 25 different conditions, like a JavaScript cash register, we'd have to write a lot of code, and it can be difficult to read and understand.

To deal with times when you need many else if conditions, we can turn to a switch statement to write more concise and readable code.

***Note:*** To a computer, a switch statement and an if/else statement are the same, but a switch statement can be easier for other humans to read. Part of being a good developer is writing code that both computers and other humans can read.

switch statements look like this:

var groceryItem = 'papaya';

switch (groceryItem) {

case 'tomato':

console.log('Tomatoes are $0.49');

break;

case 'lime':

console.log('Limes are $1.49');

break;

case 'papaya':

console.log('Papayas are $1.29');

break;

default:

console.log('Invalid item');

break;

}

1. The switch keyword initiates the statement, and is followed by ( ... ), which contains the condition that each case will compare to. In the example, the condition is groceryItem.
2. Inside the block, { ... }, there are cases. case is like the else if part of an if/else if/else statement. The word following the first case is *'tomato'*. If groceryItem equalled *'tomato'*, that case's console.log would run.
3. groceryItem equals *'papaya'*, so the first and second case statements are skipped. The third case runs since the case is *'papaya'* matches groceryItem's value. This particular program will log out: 'Papayas are $1.29'.
4. Then the program stops with the break keyword. This keyword will prevent the switch statement from executing any more of its code.
5. At the end of each switch statement, there is a default condition. If none of the cases are true, then this code will run.

***Instructions***

*Write your code in the werewolf.js file.*

1. Let's illustrate this by converting our werewolf program to a switch statement. For now, let's also delete the foggyNight variable so it doesn't fog up this concept.

moonPhase will become the condition of the switch statement. Then, each moon phase will become each case that the switch statement checks for.

Start by writing a switch statement with moonPhase as its condition.

1. Then, write each else if condition as a case.

If moonPhase is 'full', then use console.log to print 'Howwwwlll!!'.

If moonPhase is 'mostly full', then use console.log to print 'Arms and legs are getting hairier'.

If moonPhase is 'mostly new', then use console.log to print 'Back on two feet'.

Remember to add a break after each console.log, like in the example in the instructions.

1. Now, add a default at the end of the switch that uses console.log to print 'Invalid moon phase', in the case that moonPhase does not equal one of our cases.
2. switch statements can make your code easier to read and easier for others to understand.

Way to go! We just learned a lot of control flow concepts:

* if/else statements make binary decisions and execute separate code based on a condition.
* We can add extra conditions with to if/else statements with else if conditions.
* switch statements make complicated if/else statements easier to read, however they achieve the same result as if/else statements.
* Comparison operators, like <, >, <=, and >= can compare two variables. After they compare, they always return either true or false.
* Logical Operators, like &&, ||, !==, and !, can compare two variables to see if a certain condition exists:
  + && checks if both sides are true.
  + || checks if either side is true.
  + !== checks if both sides are not equal.
  + ! changes a variable that is true to false, and vice versa.

In the next lesson, we'll learn about functions, and how to write blocks of code that are reusable.