Lecture 3

JavaScript Conditionals

Topics

- Conditional Programming
- If else statements
- Else if statements
- Logical Expression & Operators
- Ternary Operator
- Switch Statements

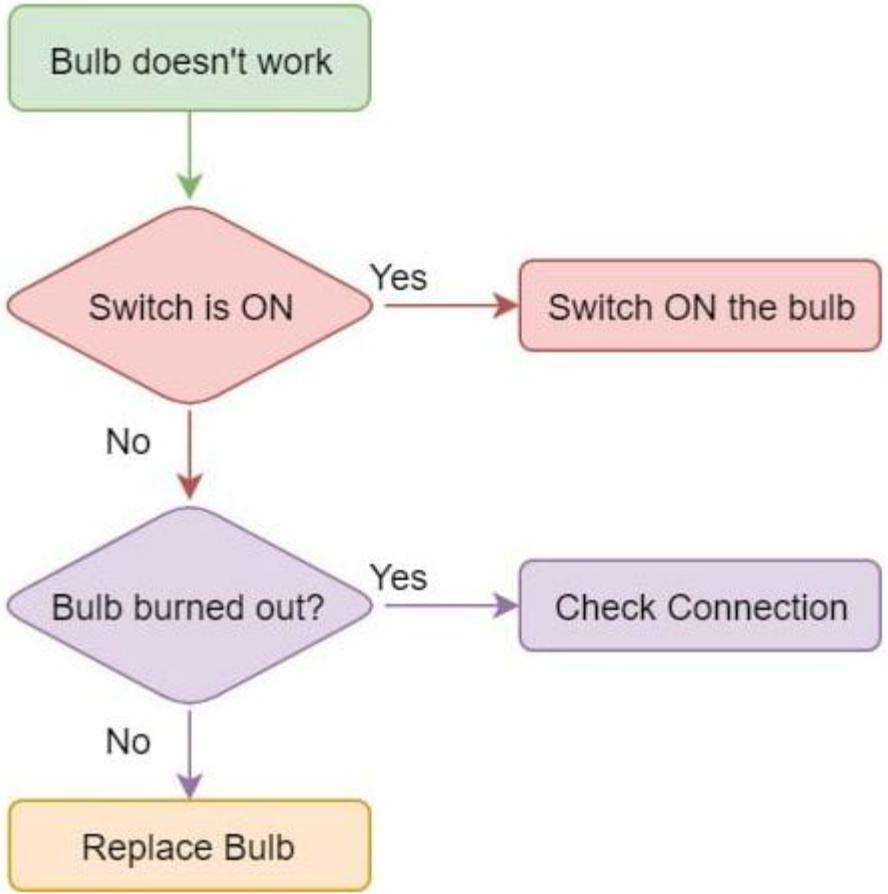
What is Conditional Programming?

What is Conditional Programming?

- Conditional statements check whether a programmer-specified Boolean condition is true or false.
- They make it possible to test a variable against a value/compare a variable with another variable and make the program act in one way if the condition is met, and another if it isn't.
- They make the program very powerful and be able to be used for a vast variety of purposes, from creating simple calculators to controlling robots.
- A program that has conditional statements is called a *Conditional Program*, and the process is known as *Conditional Programming*.

Flowcharts

- Flowcharts are used while designing and documenting simple processes or programs.
- They help visualise what is going on and thereby help understand a process, and perhaps also find flaws, bottlenecks, and other less-obvious features within it.



If...else statements

- **If...else statements** allow you to execute certain pieces of code based on a condition, or set of conditions, being met.

- This is extremely helpful because it allows you to choose which piece of code you want to run based on the result of an expression.

```
if (/* this expression is true */) {
  // run this code
} else {
  // run this code
}
```

If...else statements cont.

- Our first look into structural programming, and probably the most common control structure is our if / else if / else statements.
  These statements use some form of conditional to determine what to do next in the shape of if **(condition) { actions }**

```
if (true) {
    console.log("The statement was true.");
}
```

If...else statements - important points

The value inside the `if` statement is always *converted* to true or false. Depending on the value, the code inside the `if` statement is run or the code inside the `else` statement is run, but not both. The code inside the `if` and `else` statements are surrounded by **curly braces {...}** to separate the conditions and indicate which code should be run.

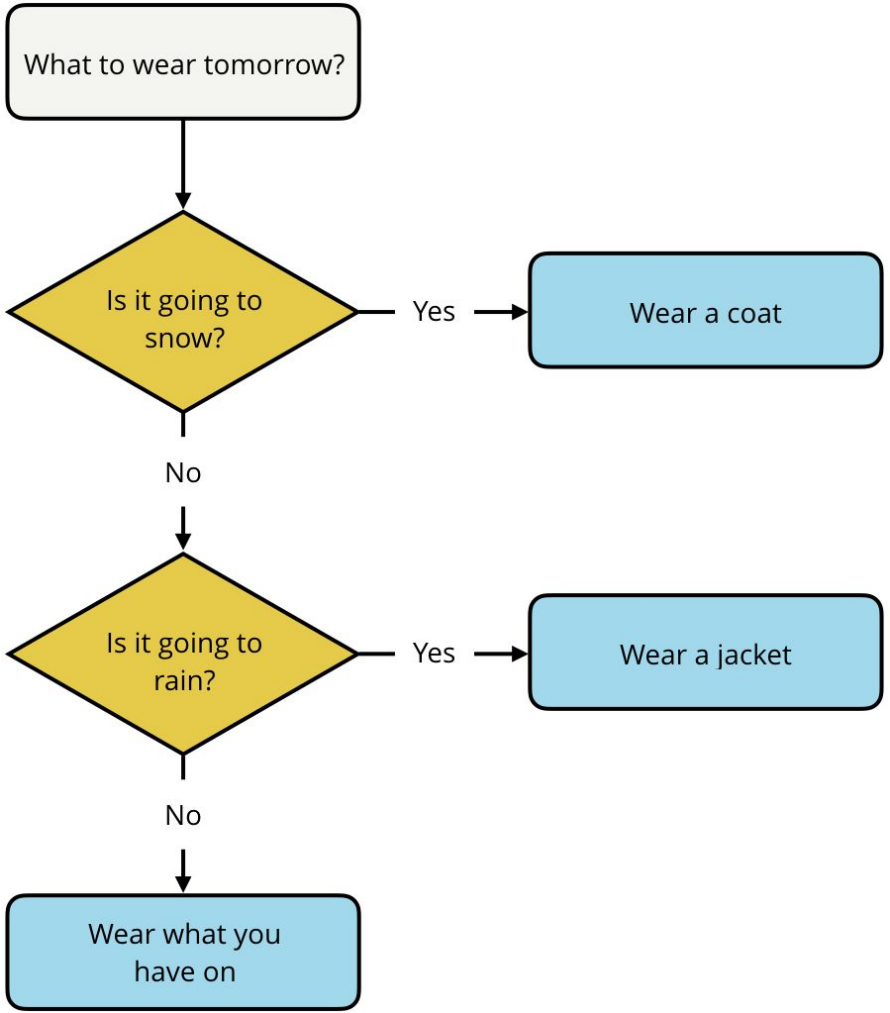
```
var a = 1;
var b = 2;

if (a > b) {
  console.log("a is greater than b");
} else {
  console.log("a is less than or equal to b");
}
```

else...if statements

In some situations, two conditionals aren't enough. Consider the following situation.

You're trying to decide what to wear tomorrow. If it is going to snow, then you'll want to wear a coat. If it's not going to snow and it's going to rain, then you'll want to wear a jacket. And if it's not going to snow or rain, then you'll just wear what you have on.



else...if statements cont..

- In JavaScript, you can represent this secondary check by using an extra if statement called an **else if statement**.
- The `else` statement essentially acts as the "default" condition in case all the other `if` statements are false.

```
var weather = "sunny";

if (weather === "snow") {
  console.log("Bring a coat.");
} else if (weather === "rain") {
  console.log("Bring a rain jacket.");
} else {
  console.log("Wear what you have on.");
}
```

```
const MAX_DISCOUNT = .15;
const MIN_DISCOUNT = .10;
let amount = 570;

if (amount >= 500) {
    console.log("Your discount is: " + (MAX_DISCOUNT * 100) + "%");
    console.log("Your total: $" + (amount * (1 - MAX_DISCOUNT)));
} else {
    console.log("Your discount is: " + (MIN_DISCOUNT * 100) + "%");
    console.log("Your total: $" + (amount * (1 - MIN_DISCOUNT)));
}
```

Discount Determiner - if/else

```
const rl = require('readline-sync');
const MAX_DISCOUNT = .15;
const MID_DISCOUNT = .12;
const MIN_DISCOUNT = .10;

let amount = rl.question("Please enter how much was spent: $");

if (amount >= 500) {
    console.log("Your discount is: " + (MAX_DISCOUNT * 100) + "%");
    console.log("Your total: $" + (amount * (1 - MAX_DISCOUNT)));
} else if (amount >= 250) {
    console.log("Your discount is: " + (MID_DISCOUNT * 100) + "%");
    console.log("Your total: $" + (amount * (1 - MID_DISCOUNT)));
} else {
    console.log("Your discount is: " + (MIN_DISCOUNT * 100) + "%");
    console.log("Your total: $" + (amount * (1 - MIN_DISCOUNT)));
}
```

Discount Determiner - if/else if/else

```
if (amount >= 500 && items >= 5) {
    //The user spent at least $500 and purchased more than 5 items.
    console.log("Your discount is: " + (MAX_DISCOUNT * 100) + "%");
    console.log("Your total: $" + (amount * (1 - MAX_DISCOUNT)));
} else if (amount >= 250) {
    //If the user spent at least $250.
    if (items >= 10) {
        //If the user purchased at least 10 items.
        console.log("Your discount is: " + (MAX_DISCOUNT * 100) + "%");
        console.log("Your total: $" + (amount * (1 - MAX_DISCOUNT)));
    } else {
        //If the user purchased less than 10 items.
        console.log("Your discount is: " + (MID_DISCOUNT * 100) + "%");
        console.log("Your total: $" + (amount * (1 - MID_DISCOUNT)));
    }
} else {
    //The user spent less than $250.
    console.log("Your discount is: " + (MIN_DISCOUNT * 100) + "%");
    console.log("Your total: $" + (amount * (1 - MIN_DISCOUNT)));
}
```

Nested Multi Conditional - if/else if/else

Logical Expression & Operators

Logical Operators & Expressions

Here is the logic expression used to represent a person's weekend plans.:

```
var person = "not busy";
var weather = "nice";

if (person === "not busy" && weather === "nice") {
  console.log("go to the park");
}
```

The `&&` symbol is the logical AND operator, and it is used to combine two logical expressions into one larger logical expression. If **both** smaller expressions are *true*, then the entire expression evaluates to *true*. If **either one** of the smaller expressions is *false*, then the whole logical expression is *false*.

Another way to think about it is when the `&&` operator is placed between the two statements, the code literally reads, "if person is not busy *AND* the weather is nice, then go to the park".

Logical Expressions

Logical expressions are similar to mathematical expressions, except logical expressions evaluate to either *true* or *false*.

```
11 != 12
```

> Returns True

You've already seen logical expressions when you write comparisons. A comparison is just a simple logical expression.

Similar to mathematical expressions that use `+`, `-`, `*`, `/` and `%`, there are logical operators `&&`, `||` and `!` that you can use to create more complex logical expressions.

Logical Operators

Logical operators can be used in conjunction with boolean values (`true` and `false`) to create complex logical expressions.

By combining two boolean values together with a logical operator, you create a *logical expression* that returns another boolean value.

Here's a table describing the different logical operators:

Operator	Meaning	Example	How it works
`&&`	Logical AND	`value1 && value2`	Returns `true` if **both** `value1` **and** `value2` evaluate to `true`.
`||`	Logical OR	`value1 || value2`	Returns `true` if **either** `value1` **or** `value2` (**or even both!**) evaluates to `true`.
`!`	Logical NOT	`!value1`	Returns the **opposite** of `value1`. If `value1` is `true`, then `!value1` is `false`.

Logical AND and OR

Truth tables are used to represent the result of all the possible combinations of inputs in a logical expression. `A` represents the boolean value on the left-side of the expression and `B` represents the boolean value on the right-side of the expression.

Truth tables can be helpful for visualizing the different outcomes from a logical expression. However, do you notice anything peculiar about the truth tables for logical AND and OR?

&& (AND)

A	B	A && B
`true`	`true`	`true`
`true`	`false`	`false`
`false`	`true`	`false`
`false`	`false`	`false`

|| (OR)

A	B	A || B
`true`	`true`	`true`
`true`	`false`	`true`
`false`	`true`	`true`
`false`	`false`	`false`

Logical Short-circuiting

&& (AND)

A	B	A && B
true	true	true
true	false	false
false	true	false
false	false	false

|| (OR)

A	B	A || B
true	true	true
true	false	true
false	true	true
false	false	false

Logical Short-circuiting cont..

In both tables, there are specific scenarios where regardless of the value of B, the value of A is enough to satisfy the condition.

For example, if you look at A AND B, if A is *false*, then regardless of the value B, the total expression will always evaluate to false because both A *and* B must be true in order for the entire expression to be true.

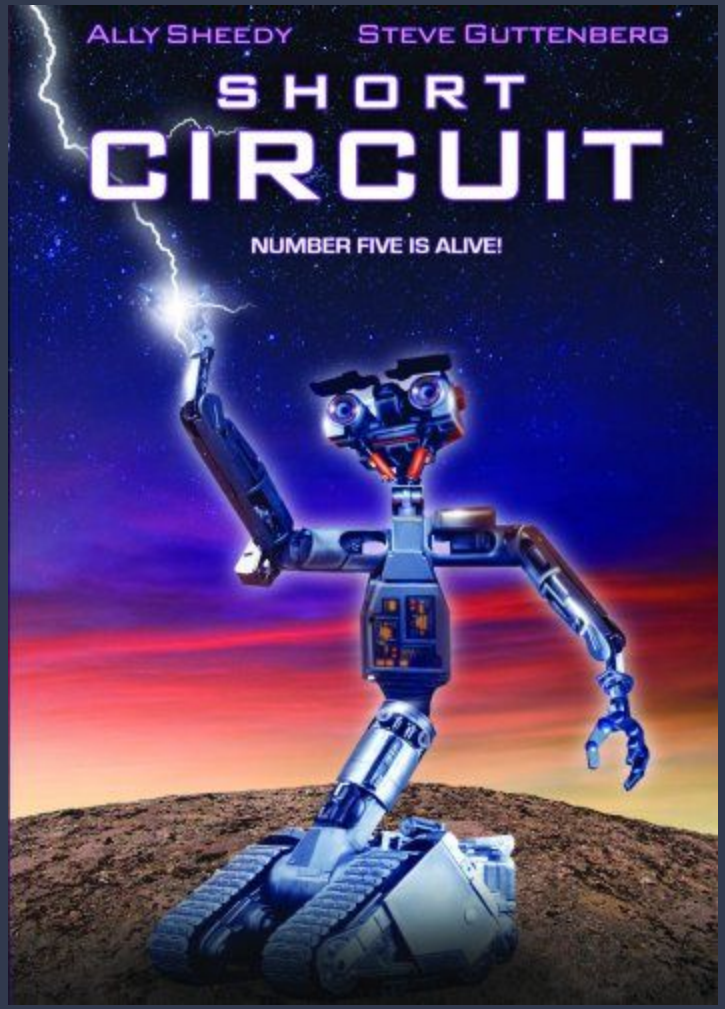
This behavior is called **short-circuiting** because it describes the event when later arguments in a logical expression are not considered because the first argument already satisfies the condition.

Below are listed several logical operator examples. Now if I were to chain many of them together you would be able to better see just how short-circuiting works.

```
console.log("hello" && "world" && "JavaScript"); //JavaScript
console.log(3 || 4 || 5 || 6); //3
```

In the first output, all 3 strings equate to true and are evaluated left to right, returning the last value. In the second output, all 4 numbers equate to true and the first instance of a true value is returned.

Short Circuiting Example



Depending on the scenario, you may come across a time where you need to run a complex function within your conditional, or possibly even many complex conditionals.

With understanding short-circuiting you can see that for any combination of logical AND conditions, starting with the value that is most likely to be false could prevent excess calculations.

Logical Expression & Operators

- Truthy & Falsy
- Ternary Operator

Truthy & Falsy

Every value in JavaScript has an inherent boolean value. When that value is evaluated in the context of a boolean expression, the value will be transformed into that inherent boolean value. (dense statement - re-read it!)

Falsy values

A value is **falsy** if it converts to `false` when evaluated in a boolean context. For example, an empty String `""` is falsy because, `""` evaluates to `false`. You already know if...else statements, so let's use them to test the truthy-ness of `""`.

```
if ("") {
    console.log("the value is truthy");
} else {
    console.log("the value is falsy");
}
```

Falsy Values

Here is a list of all the falsy values:

1. the Boolean value `false`
2. the `null` type
3. the `undefined` type
4. the number `0`
5. the empty string `""`
6. the odd value `NaN` (stands for "not a number", check out the `NaN` MDN article)

That's right, there are only *six* falsy values in all of JavaScript!

Truthy Values

A value is **truthy** if it converts to `true` when evaluated in a boolean context. For example, the number `1` is truthy because, `1` evaluates to `true`. Let's use an if...else statement again to test this out:

```
if (1) {
    console.log("the value is truthy");
} else {
    console.log("the value is falsy");
}
```

Truth Values

Here is a list of all the truthy values:

1. `true`
2. `42`
3. `"pizza"`
4. `"0"`
5. `"null"`
6. `"undefined"`
7. `{}`
8. `[]`

Essentially, if it's not in the list of falsy values, then it's truthy!

Ternary Operator

Sometimes, you might find yourself with the following type of conditional. This code works, but it's a rather lengthy way for assigning a value to a variable. Thankfully, in JavaScript there's another way.

```
var isGoing = true;
var color;

if (isGoing) {
  color = "green";
} else {
  color = "red";
}

console.log(color);
```

Ternary Operator cont..

The **ternary operator** provides you with a shortcut alternative for writing lengthy if...else statements.

```
conditional ? (if condition is true) : (if condition is false)
```

To use the ternary operator, first provide a conditional statement on the left-side of the `?`. Then, between the `?` and `:` write the code that would run if the condition is `true` and on the right-hand side of the `:` write the code that would run if the condition is `false`.

Ternary Operator cont..

We can rewrite the previous code using ternary operators

```
var isGoing = true;
var color = isGoing ? "green" : "red";
console.log(color);
```

This code not only replaces the conditional, but it also handles the variable assignment for `color`.

If you breakdown the code, the condition `isGoing` is placed on the left side of the `?`. Then, the first expression, after the `?`, is what will be run if the condition is *true* and the second expression after the, `:`, is what will be run if the condition is *false*.

Ternary Operator recap

One of the core uses of the ternary if is to quickly assign a value to a variable with as little code possible and are written in the form of **(condition) ? true statement : false statement**;

```
let x = 7;
let y = x >= 5 ? "x is larger than 5." : "x is smaller than 5.";
```

Switch Statements

```
if (option === 1) {
  console.log("You selected option 1.");
} else if (option === 2) {
  console.log("You selected option 2.");
} else if (option === 3) {
  console.log("You selected option 3.");
} else if (option === 4) {
  console.log("You selected option 4.");
} else if (option === 5) {
  console.log("You selected option 5.");
} else if (option === 6) {
  console.log("You selected option 6.");
}
```

If you find yourself repeating else if statements in your code, where each condition is based on the same value, then it might be time to use a switch statement.

Switch Statements

A **switch statement** is an another way to chain multiple else if statements that are based on the same value **without using conditional statements**. Instead, you just *switch* which piece of code is executed based on a value.

Here, each else if statement (option === [value]) has been replaced with a case clause (case [value]:) and those clauses have been wrapped inside the switch statement.

```
switch (option) {
  case 1:
    console.log("You selected option 1.");
  case 2:
    console.log("You selected option 2.");
  case 3:
    console.log("You selected option 3.");
  case 4:
    console.log("You selected option 4.");
  case 5:
    console.log("You selected option 5.");
  case 6:
    console.log("You selected option 6.");
}
```

Switch Statements

When the switch statement first evaluates, it looks for the first `case` clause whose expression evaluates to the same value as the result of the expression passed to the switch statement. Then, it transfers control to that `case` clause, executing the associated statements.

So, if you set `option` equal to `3`...

```
var option = 3;

switch (option) {
  ...
}
```

Break Statement

The **break statement** can be used to terminate a switch statement and transfer control to the code following the terminated statement. By adding a break to each case clause, you fix the issue of the switch statement *falling-through* to other case clauses.

```
var option = 3;

switch (option) {
  case 1:
    console.log("You selected option 1.");
    break;
  case 2:
    console.log("You selected option 2.");
    break;
  case 3:
    console.log("You selected option 3.");
    break;
  case 4:
    console.log("You selected option 4.");
    break;
  case 5:
    console.log("You selected option 5.");
    break;
  case 6:
    console.log("You selected option 6.");
    break; // technically, not needed
```

Falling-through

In some situations, you might want to leverage the "falling-through" behavior of switch statements to your advantage.

You can add a `default` case to a switch statement and it will be executed when none of the values match the value of the switch expression.

```
var tier = "none";
var output = "You'll receive ";

switch (tier) {
  ...
  default:
    output += "one copy of the Exploding Kittens card game.";
}

console.log(output);
```

```
const rl = require('readline-sync');

let showSomething = rl.question("Please enter a number between 1-3
inclusive: ");
showSomething = parseInt(showSomething);

switch(showSomething) {
    case 1:
        console.log("You chose what's behind door #1!");
        break;
    case 2:
        console.log("You chose what's behind door #2!");
        break;
    case 3:
        console.log("You chose what's behind door #3!");
        break;
    default:
        console.log("You didn't follow the instructions.");
}
```

Sample Switch Statement

```
let showSomething = rl.question("Please enter a number between 1-3
inclusive: ");
showSomething = parseInt(showSomething);

if (showSomething === 1) {
    console.log("You chose what's behind door #1!");
} else if (showSomething === 2) {
    console.log("You chose what's behind door #2!");
} else if (showSomething === 3) {
    console.log("You chose what's behind door #3!");
} else {
    console.log("You didn't follow the instructions.");
}
```

Sample Switch Statement rewritten as if else statements