Front-End Web Development

Conditional Statements

GENERAL ASSEMBLY

Today's Learning Objectives

In this lesson, you will:

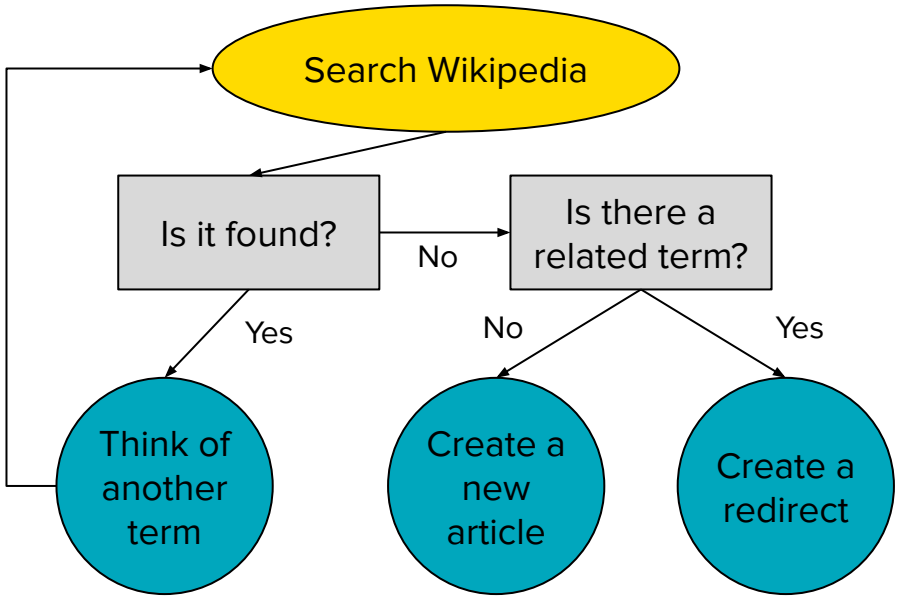
- Define conditional statements in JavaScript to create logic-driven programs.
- Use **`switch`** statements to encapsulate complex logical chains.
- Choose the correct logical operators to enhance conditional statements.

Front-End Web Development

Comparison Operators

Why Use Conditional Statements?

Conditional logic allows you to create **vastly more complex programs**. Think of the decision-making process of giant flowcharts...



Comparison Operators

First, we'll need **comparison operators** — a set of operators that give you the ability to compare values and return a Boolean result (true or false).

Comparison Operator	Meaning
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to
===	Strict equality (value and data types)
!==	Strict inequality

Double vs. Triple Equals

== vs ===

!= vs !==

Double equals doesn't check for type, so it won't care if the data types are the same. This can lead to some unpredictable behavior!

Triple equals — the strict equality operator — will compare both the value and data type, creating much more predictable results. **Just use triple equals!**

Conditional Statements

Conditionals are function-like statements that take Booleans as inputs:

```
if (valueOne === valueTwo) {
  console.log(“Valid”);
}
```

Note the **code block** defined in curly braces (`{ }`). This code block executes if the Boolean provided is **true**.

`else` Statements

You will often want to have an **`else`** statement immediately after the **`if`** statement. This will trigger when the **`if`** comparison turns out to be **`false`**.

```
if (valueOne === valueTwo) {
  console.log(“Valid”);
}
else {
  console.log(“Invalid”);
}
```


Multiple Conditions

```
if (test1 === test2) {
  console.log('test1 and test2 are equal');
}
else if (test1 > test2) {
  console.log('test1 is greater than test2');
}
else {
  console.log('none of the conditions were met');
}
```

Careful! Equals Aren't All Equal...

When you use **=**, that is an **assignment operator**.

If you try to use **=** instead of **===** in a comparison statement, you will get strange results — it will always evaluate as **true**!

```
// This won't work the way you think it will!
let temperature = 0;
if (temperature = 100) {
  console.log("Always going to happen!");
}
```

Guided Walk-Through: Conditionals in Action

For examples of conditionals affecting the page, check out this CodePen:

Reference code:

https://codepen.io/GAmarketing/pen/BayWLgR

Front-End Web Development

Logic Operators

Logic Operators

Logic operators allow us to combine multiple conditions together. For very complex conditionals, you can put several conditions in parentheses to evaluate them as a single expression.

Operator	Description
&& (AND)	Evaluates to **`true`** only if all combined values are true.
|| (OR)	Evaluates to **`true`** if any of the combined values are true.
! (NOT)	Reverses the Boolean result of whatever follows it.

Multiple Conditions, One Statement

You can check to see if two conditions are met in one statement with a logic operator:

```
let clickDetected = true;
let clickTwoDetected = false;

if (clickDetected === true && clickTwoDetected === false){
  // do something
}
```

Using !

The **not** operator, `!`, is a great way to check if something exists in the JS memory system.:

```
let whatever; // would be undefined, no memory assignment

if( !whatever ) {
  console.log(“Turns out whatever doesn’t exist!”);
}
```

If there isn’t a comparison inside a conditional statement, JS will interpret the argument as a Boolean. The only values that would equate to false are `0`, `-0`, `null`, `NaN`, `undefined`, or `“”`, so nearly anything that exists will pass the test.

Pro Tip: Condensing Conditionals

Like we saw in the previous slide, not all conditionals need a comparison statement — especially if the values being tested are already Booleans!

Thus, you will rarely see a comparison with `=== true` or `!== false`.

Instead, developers will use the following pattern:

```
if (clickDetected && !clickTwoDetected) {
  // do something
}
```

Front-End Web Development

`switch` Statements

`switch` Statements

`switch` is an implicit equality comparison (`===`). It's comparing the argument of **`switch`** with each **`case`**. The **`switch`** looks for every matching **`case`**, then performs instructions until it hits **`break`**, which exits the chain.

container → `switch (transportOption) {`
condition → `case 'car':`
instruction → `document.querySelector('#dest').classList.add('car');`
stop → `break;`

```
let transportOption = 'car';

// Add class to <div id="dest"> depending on variable
switch (transportOption) {
  case 'car':
    document.querySelector('#dest').classList.add('car');
    break;
  case 'bus':
    document.querySelector('#dest').classList.add('bus');
    break;
  case 'metro':
    document.querySelector('#dest').classList.add('metro');
    break;
}
```


Multi-Case `switch`

`switch` can be used to funnel multiple cases to the same instruction. Just list cases together before the instruction/break combo, and you can cleanly handle similar cases:

```
switch(seasonCheck) {
  case 'autumn':
  case 'fall':
    console.log("It's fall now!");
    break;

  case 'spring':
  case 'getting warmer':
    console.log("Spring time is near");
    break;
}
```

Making Conditionals Easier

Conditionals make programs harder to debug because more operations can take place in any given program.

- Use functions to encapsulate code as necessary.
    - If the code inside a conditional takes up five lines or more, it will become hard to keep track of which layer of conditional we're in. Make it a function!
- **`console.log()`** is your friend.
    - Leave yourself messages to figure out what path a conditional took.
    - If a conditional gives you mysterious results, log the values and data types.
- Use **`switch`** statements for situations with four or more cases.
    - The setup for a **`switch`** only pays off in lengthier situations.

Partner Exercise:

Channel Changer

60 minutes

This challenge will involve using event listeners, functions, and conditionals to create an interactive TV with channels the user can change.

Remember: Write small pieces at a time and test them early and often!

Starter code:

https://codepen.io/GAmarketing/pen/eYmvBMq

Solution code:

https://codepen.io/GAmarketing/full/dyPvOjN

Key Takeaways

Conditionals Allow Decisions

- Use conditionals to allow JavaScript to make logic-based decisions.
- Conditionals are one tool in the **control flow** toolbox.
- Use logical operators and conditional chains for more complex reasoning.

For Next Time

JavaScript Loops and Arrays

- Loops will allow you to repeat a code block until a condition is met.
- Loops are another example of **control flow** determining how and when code should execute.
- Arrays are collections of data that allow us to store multiple things together.


GA