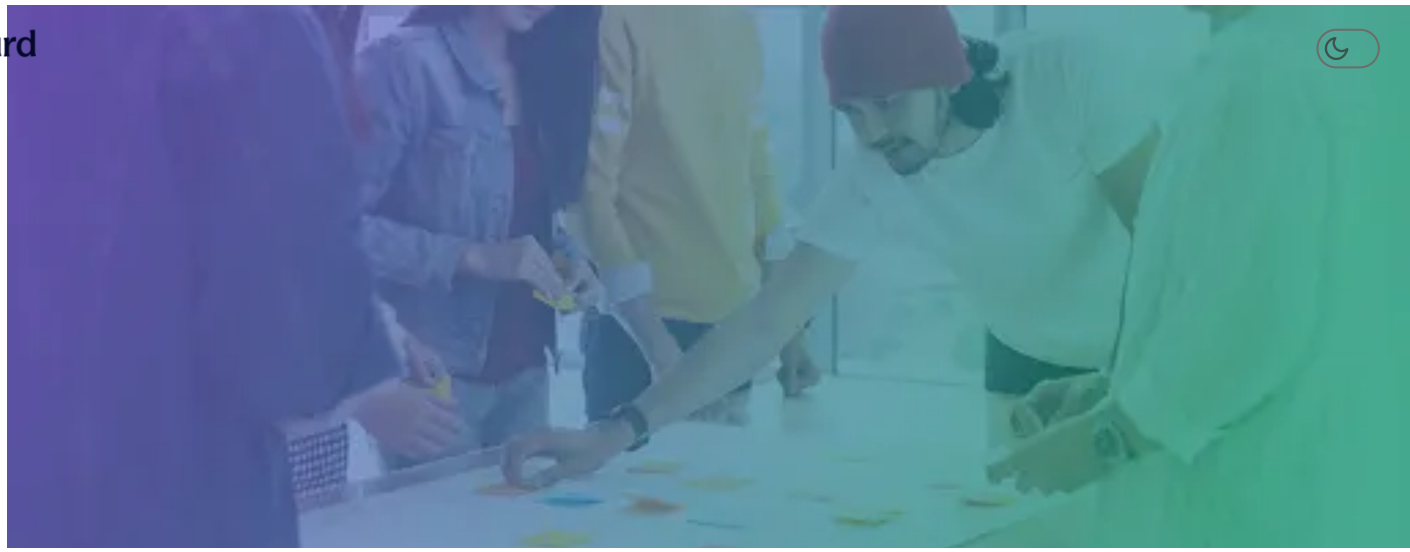
Springboard



Conditional Logic

Goals

- Explain the basic structure of a conditional statement
- Explain how expressions evaluate to either true or false for controlling conditional statements
- Construct ternaries and switch statements to control the flow of our programs

Comparison operators

A JavaScript expression can evaluate to either true or false through the use of comparison operators

Comparison operators are useful when comparing multiple values

```
let userAge = 20; userAge > 10; // -> true userAge <= 20; // ->
true userAge === 30; // -> false
```

A list of comparison operators

- === strict equality
- == equality

- *!*== strict inequality
- *!*= not equal
- < less than
- <= less than or equal to
- > greater than
- >= greater than or equal to

Be careful when using relational operators (i.e. less than, greater than, etc) on things that aren't numbers!

== vs ===

- Checks for equality of value, but not equality of type.
- It coerces both values to the same type and then compares them.
- This can lead to some unexpected results!
- Use === and *!*==

Conditional statements

- In javascript every expression returns a value
- All values can be evaluated as true or false
- This allows us to control the flow of our programs through conditional statements
- We will explore expression as boolean values in detail but first let's look at conditional statements

What Are Conditional Statements?

- When an expression is passed to a conditional statement, the expression is converted to true or false
- This allows us to determine if a code branch runs or not
- In other words, the code runs only if certain conditions are met

Basic conditional statements

```
if (true) { console.log("This was true") } else {
console.log("This was false") } // this would print true
```

- ***if*** x is true, run the first block of code
- ***else*** in any other case, run another block of code
- We can place any expression in the parentheses and it will be converted to true or false

Why Do We Need Conditionals?

```
if (true) { // print it was true } else { // print it was false
}
```

- In this example we hardcoded true so the first block will always execute
- But in programming real applications, it's all about responding to different user inputs
- Typically we won't know what the value of a variable is going to be and use conditional logic to handle different scenarios

Example :

```
let inputPassword = 'secret1'; if (secret1 === dbPassword) { //
authenticate the user } else { // print your password was
incorrect }
```

- User enters password
- Does it match the database?
  - Yes, proceed
  - No, try again

```
// let inputPassword = 'secret1'; if (secret1 === databasePassword) { // authenticate the user } else { // print your password was incorrect }
```

- Notice here we used '===' to compare the user input with the stored value in our database
- This is our first example of a comparison operator
- Comparison operators are one approach for converting an expression to a boolean value

Else if

- In addition to ***'if'*** and ***'else'*** statements
- There is a third condition ***'else if'***
- ***'else if'*** is unique because it can be used multiple times
- `if` x is true, run the first block of code
- `else if` y is true, run a different block of code
- `else` in any other case, run another block of code

```
let x = 10; // only one of the branches below will run if (x > 10) { console.log('x is greater than 10'); } else if (x === 10) { console.log('x is equal to 10'); } else { console.log('x is less than 10'); }
```

- It is important to note only one code branch will run

```
let x = 11; if (x > 10) { console.log('x is greater than 10'); } else if (x === 11) { console.log('x is equal to 11'); } else { console.log('x is less than 10'); }
```

Even if two conditions are true, only the code in the first true condition runs

Conditional statements

What will this code do?

```
let x = 11; if (x > 10) { console.log('x is greater than 10');
} if (x === 11) { console.log('x is equal to 11'); } else {
console.log('x is less than 10'); }
```

- Since there are multiple *'if'* statements multiple blocks of code can run
- If the second *'if'* condition was replaced with 'else if' only the first true statement will run

Boolean expressions

If you're wondering where the name Boolean comes from, it's George Boole. He invented Boolean Algebra / Logic which is the basis of modern computing.

Boolean logic is actually simple to understand! It's just the system of logic governing true or false values and the relationships formed by combining them together into logical expressions.

- When we pass an expression to a conditional statement the return value is converted to true or false
- This allows us to control the flow of the program

Evaluating expressions as true or false

To understand how expressions become true or false we must know the following concepts

- Truthy or falsy values
- Comparison operators
- Logical operators

Truthy or falsy values

- Expressions in the parentheses of a conditional statement resolve to true or false

- Values that are not explicitly true or false are still innately evaluated "truthy" or "falsey".

There are only six falsy values

- false
- null
- undefined
- 0
- ''
- NaN

Everything else evaluates to true

Determining Truthiness

If you are ever unsure if something is truthy or falsey, you can simply convert it to a boolean

- One way of doing this by wrapping it in a Boolean constructor

```
let favoriteNumber = 0; Boolean('hi'); // true Boolean(''); //
false Boolean(favoriteNumber); // false Boolean(null); // false
Boolean(1); // true Boolean(undefined); // false
Boolean('cat'); // true Boolean(-0); // false Boolean(-1); //
true
```

Logical Operators

Logical Operators are the final approach for evaluating expressions as either true or false

There are three essential logical operators in JavaScript:

- NOT !
- OR ||
- AND &&

NOT !

"!" flips the value of the expression

```
let a = true; console.log(!a); // => false
```

You can think of "!a" reading like the following

```
if (not a) { // -> run this code }
```

- The bang "!" operator can also be used to evaluate an expression as a boolean value
- We already saw how we can do this with the Boolean constructor

```
Boolean('hi'); // true
```

- Another approach is to use use "!!" (bang bang)

```
let favoriteNumber = 0; !!'hi'; // true !!''; // false
!!favoriteNumber; // false !!null; // false !!1; // true
!!undefined; // false !!'cat'; // true
```

- This flips the boolean value twice to give the correct "truthy / falsy" value

OR ||

The easiest way to think about '||'

- If the left value is true or the right value is true the entire expression is true
- If both the left value and the right value are false, the entire expression is false

```
let a = true; let b = false; console.log(a || b); // true
console.log(a || a); // true console.log(b || b); // false
```

- But we should look at this more precisely, the "||" logical operator checks two values
- If the value on the left evaluates to true, the left value is returned
- The left value then gets converted to a truthy / falsy value

```
'hi' || 5 // => 'hi' (truthy) 1000 || '' // => 1000 (truthy)
```

- If the value to the left is false then the value to the right will be returned regardless of its value

```
'' || 5 // => 5 (truthy) '' || 0 // => 0 (falsey)
```

The return value is what ultimately gets converted to a truthy falsy value

- The "||" logical operator is often used with conditional statements

```
'' || 5 // => 5
```

- Since 5 evaluates to true the entire expression evaluates to true

```
Boolean('' || 5) // => true
```

```
if ('' || 5) { // this code will execute } else { // this code
will not }
```

AND &&

- The “&&” logical operator also checks two values
- With “||” the left side `or` the right side need to be true to return a truthy value
- With “&&” the left side `and` the right side both need to be true for the expression to return a truthy value

```
let a = true; let b = false; console.log(a && b); // false
console.log(a && a); // true console.log(b && b); // false
```

Example

```
if (a && b) { // a and b are both true } else if (!a && b) { //
a is false but b is true } else if (a && !b) { // a is true but
b is false } else { // a is false and b is false }
```

Short Circuiting

- Note that in an expression with “&&” or “||”, sometimes the right-hand side never actually runs!
- Specifically there are two cases where the expression is “short-circuited”:

1. false expression && anything
    - (*If the left expression is false we immediately move on*)
2. true expression || anything
    - (*If the left expression is true we immediately move on*)

```
2 + 2 === 5 && console.log('this will never run!'); 5 > 1 ||
console.log('this won\'t either!');
```

Operator Precedence

- NOT (!) has higher precedence than && and ||
- && has higher precedence than ||
- You can alter this using parenthesis ()

The Ternary Operator

- A more advanced application of conditional statements is the ternary operator
- A ternary operator is the shorthand for an "if / else" statement
  - (*ternary operators can only have two branches and do not work with "else if"*)
- The name ternary comes from the fact that there are three operands. An expression followed by two statements
- They allow us to write conditionals on one line

```
( expression to check ) ? statement if true : statement if false
```

```
// guess a number between 1 to 20 let guess = 15; guess > answer ? console.log('too high') : console.log('too low');
```

Often you will capture the return value of a ternary operator in a variable

```
let guess = 15; let wasCorrect = (guess === 15) ? 'Your correct!' : 'Incorrect'; console.log(wasCorrect) // => 'Your correct!'
```