Booleans, And Conditionals

Software Development Bootcamp

Topic

Boolean

What are Boolean Values?

In JavaScript a **Boolean** is a data type that can only have two possible values: `true` or `false`

Booleans are often used to:

- Check if something is true or not
- Make decisions in code (“if this is true, do that”)
- Store yes/no information

Truthy and Falsy

- Truthy values are values that evaluate to `true` in conditional statements or boolean operations
- All values are considered truthy except the following falsy values
  - **false**
  - **0 (zero)**
  - **‘ ‘ “ “ (empty string)**
  - **null**
  - **undefined**
  - **NaN (not a number)**
- Everything else including empty arrays and objects are considered truthy.

Why are Boolean Values Important?

- **Decision making:** Help control the flow of a program, like deciding whether to stop a process or keep going
- **Comparing values:** Booleans are the result of comparisons, letting us check if things are equal, greater than, or less than each other.
- **Clarity:** Booleans make code easier to read and understand.

Topic

Conditionals

What Are Conditionals?

Conditionals are statements that only run code if something is true or false. They help your program choose different actions based on different conditions.

Types of Conditionals

- `if` **Statement**: Checks if a condition is true, then runs some code.
- `else` **Statement**: Runs some code if the `if` condition is false.
- `else if` **Statement**: Checks another condition if the first `if` condition is false.

Using If Statements to Set Up Conditions

- `if (expression) { codeBlock }`
- The expression is evaluated, and if it's truthy, the code block runs. If it's falsy, the code block is skipped.

```
if(age >= 18) {
   // this block runs if the
condition is true
   console.log('you can vote!')
} else {
   // this block runs if the
condition is false
   console.log('too young to vote.')
}
```

Using Else Statements

- The **else** keyword allows you to run a code block when the conditional expression is falsy
- In this example which code block will execute?

```
if (false) {
    console.log('🍕')
} else {
    console.log('🍔')
}
```

Not Using Else Statements

- Compare the behavior of these two snippets:
- When we change the condition to true what differences do we see in the two examples?

```
// Example 1
if (false) {
   console.log('🍕')
} else {
   console.log('🍔')
}

// Example 2
if (false) {
    console.log('🍕');
}
console.log('🍔');
```

Conjunctions (AND, OR, NOT) make complex conditions.

- You can create more complex logical expressions using the logical operators (AND, OR, NOT)

AND &&, OR ||, NOT ! Examples

```
let a = true;
let b = true;
// The AND operator checks if both a and b
are true
if (a && b) {
   console.log("Both are true!");
} // Output: "Both are true!"

a = true
b = false
// The OR operator checks if at least one
thing is true
if (a || b) {
   console.log('One is true!')
} // Output: "One is true!"
```

```
a = true
// The NOT operator flips true to false
and false to true
if (!a) {
   console.log("a is false!")
} else {
   console.log('a is true!')
} // Output: "a is true!"
```

Combining Logical Operators

- You can combine logical operators to check more complicated conditions.
- Logical operators use **short-circuit evaluation**, which means that the second part is not checked if the first part already gives the answer.

```
let a = true;
let b = false;
let c = true;
// using both the && (AND) and ||
(OR) operators
if ((a && b) || c) {
   console.log("Complex condition
is true!");
} // Output: "Complex condition is
true!"
```

Why Use Conditionals?

Conditionals help make our programs smarter by allowing them to make decisions. We use them in games, apps, websites, and more to react to different situations.

Exercise

Divisible