Section 5 - Conditionals and More Loops

Learning Outcomes

Decision making and branching logic in applications is important in data driven and dynamic applications. This section will cover `if` / `else` statements as well as conditional operators that are used to make decisions in applications.

- **Conditional Statements** - `if` / `else` syntax is discussed
- **Comparison Operators** - these operators are used with `if` / `else` to make decisions
- **Nested Loops** - Loops inside loops (nested loops) may been needed to process information or make decisions.

Resources

- Conditionals (if/else) - https://www.w3schools.com/js/js_if_else.asp
- Boolean Expressions - https://www.w3schools.com/js/js_booleans.asp
- Comparisons (see Section 3) - https://www.w3schools.com/js/js_comparisons.asp
- While loop - https://www.w3schools.com/js/js_loop_while.asp
- Loops break and continue - https://www.w3schools.com/js/js_break.asp

Conditional Expressions (if/else)

Very often when you write code, you want to perform different actions for different decisions.

You can use conditional statements in your code to do this.

In JavaScript, we have the following conditional statements:

- Use `if` to specify a block of code to be executed, if a specified condition is `true` .
- Use `else` to specify a block of code to be executed, if the same condition is `false` .
- Use `else if` to specify a new condition to test, if the first condition is `false` .

The `if` Statement

Use the `if` statement to specify a block of JavaScript code to be executed if a condition is `true` .

```
Syntax:
if (condition(s) that evaluate to true or false) {
  //  block of code to be executed if the condition is true
}
```

Make a "Good day" greeting if the hour is less than 18:00 using the 24hr time clock:

```
Example:
let hour = 4;
let greeting = '';
if (hour < 18) {
    greeting = "Good day";
}
// using multiple conditions
if (hour > 6 && hour < 18) {
    greeting = "Good day";
}
```

The `else` Statement

Use the `else` statement to specify a block of code to be executed if the condition is `false`.

```
Syntax:
if (condition(s) that evaluate to true or false) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

If the hour is less than 18, create a "Good day" greeting, otherwise "Good evening":

```
Example:
let hour = 4;
let greeting = '';
if (hour < 12) {
    greeting = "Good morning";
} else {
    greeting = "Good afternoon";
}
```

The `else if` Statement

Use the `else if` statement to specify a new condition if the first condition is `false`.

```
Syntax:
if (condition(s) that evaluate to true or false) {
  //  block of code to be executed if condition1 is true
} else if (condition(s) that evaluate to true or false) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

If time is less than 10:00, create a "Good morning" greeting, if not, but time is less than 20:00, create a "Good day" greeting, otherwise a "Good evening":

```
Example:
let hour = 9;
let greeting = '';
if (hour < 6) {
    greeting = "Why are you up so early?";
} else if (hour >= 6 && hour < 12) {
    greeting = "Good morning!";
} else if (hour >= 12 && hour < 18) {
    greeting = "Good afternoon!";
} else {
    greeting = "Good evening";
}
```

Boolean Expressions (`TRUE` or `FALSE`)

Very often, in programming, you will need a data type that can only have one of two values, like

- `TRUE` or `FALSE`

The `Boolean` value of an expression is the basis for all JavaScript comparisons and conditions.

Refer back to `03 - Comparison and Logical Operators` the examples below use comparison operators in `Boolean` expressions. The result of the expressions drive which statements in the program are executed.

Operator	Description	Example
`==`	equal to	`if (day == "Monday")`

Operator	Description	Example
`>`	greater than	`if (salary > 9000)`
`<`	less than	`if (age < 18)`

Comparisons and Logical Operators

Comparison and Logical operators are used to test for `true` or `false`.

Comparison Operators

Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that `x = 5`, the table below explains the comparison operators:

Operator	Descrption	Example	Result

`==`	equal to	`x == 8`
		`x == 5`
		`x == "5"`
			`false`
			`true`
			`true`

`===`	equal value and equal type	`x === 5`
		`x === "5"`
			`true`
			`false`

`!=`	not equal	`x != 8`	`true`

`!==`	not equal value or not equal type	`x !== 5`
		`x !== "5"`
		`x !== 8`
			`false`
			`true`
			`true`

`>`	greater than	`x > 8`	`false`
`<`	less than	`x < 8`	`true`
`>=`	greater than or equal to	`x >= 8`	`false`
`<=`	less than or equal to	`x <= 8`	`true`

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that `x = 6` and `y = 3`, the table below explains the logical operators:

Operator	Descrption	Example	Result
`&&`	and	`(x < 10 && y > 1)`	`true`
||	or	`(x == 5 || y == 5)`	`false`
`!`	not	`!(x == y)`	`true`

Nested Loops and `break`, `continue` with loops

Nested loops

Loops can be nested (one loop inside another) as needed to process input or determine some business logic. Consider the following string of fruits (delimited with semicolons `;`) with counts (delimited with commas `,`) in the following pattern:

```
fruit,count;fruit,count;fruit,count
```

A nested `for` loop can be used to `split()` the string multiple times.

```
Example:
let input = "Apples,2;Bananas,12;Cherries,30";
const array1 = input.split(";");
for (let i=0; i<array1.length; i++) {
    const array2 = array1[i].split(",");
    for (let j=0; j<array2.length; j++) {
        document.write(`${i}{j}: ${array2[0]} = ${array2[1]}`);
    }
}
```

`break` and `continue`

The `break` statement terminates the current loop and transfers program control to the statement following the terminated statement.

```
// break with for loop
let text = '';
for (let i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is: " + i;
}
console.log(text);
// expected output:
// The number is: 0
// The number is: 1
// The number is: 2


// break in while loop
let i = 0;
while (i < 6) {
    if (i === 3) {
        break;
    }
    i = i + 1;
}
console.log(i);
// expected output: 3
```

The `continue` statement terminates execution of the statements in the current iteration of the curent loop, and continues execution of the loop with the next iteration.

```
let text = '';
for (let i = 0; i < 10; i++) {
    if (i === 3) {
        continue;
    }
    text = text + i;
}
console.log(text);
// expected output: "012456789"
```