Introduction to JavaScript: Control Flow

codecademy

`if` Statement

An `if` statement accepts an expression with a set of parentheses `( )`:

- If the expression evaluates to a truthy value, then the code within its code body executes.
- If the expression evaluates to a falsy value, its code body will not execute.

```
const isMailSent = true;

if (isMailSent) {
  // This code block will be executed
  console.log('Mail sent to recipient');
}
```

`else` Statement

An `else` block is added to an `if` block or series of `if`-`else if` blocks. The `else` block will be executed only if the `if` condition fails.

```
const isTaskCompleted = false;

if (isTaskCompleted) {
  console.log('Task completed');
} else {
  console.log('Task incomplete');
}
```

`if` `else` `else if` Statement

After an initial `if` block, `else if` blocks can each check an additional condition. An optional `else` block can be added after the `else if` block(s) to run by default if none of the conditionals evaluated to truthy.

```
const size = 10;
if (size > 100) {
  console.log('Big!'):
} else if (size > 20) {
  console.log('Medium');
} else if (size > 4) {
  console.log('Small');
} else {
  console.log ('Tiny');
}
// 'Small'
```

`switch` Statement

JavaScript's `switch` statements provide a means of checking an expression against multiple values. It first evaluates an expression. The result of the expression is matched against values in one or more `case` clauses. If a case matches, the code inside that clause is executed.

The `case` clause should finish with a `break` keyword. If no case matches but a `default` clause is included, the code inside `default` will be executed. If `break` is omitted from the block of a `case` (or the execution is not broken otherwise, such as returning from a function with a switch), the `switch` statement will continue to check against `case` values until a break is encountered or the flow is broken.

```
const food = 'pizza';

switch (food) {
  case 'oyster':
    console.log('Enjoy the taste of the sea');
    break;
  case 'pizza':
    console.log('Enjoy a delicious pie');
    break;
  default:
    console.log('Enjoy your meal');
}

// Output: 'Enjoy a delicious pie'
// If food = 'Cheese', Output: 'Enjoy your meal'
```

JavaScript Strict Comparisons

The strict equality operator (`===`) checks if two values are the same and returns `true` if they are the same. The inequality comparison operator (`!==`) check if two values are different and return `true` if they aren't the same.

```
console.log(1 === 1);    // true
console.log('1' === 1);  // false
console.log(8 !== 9);    // true
```

JavaScript Comparison Operators

JavaScript *comparison operators* are used to compare two values and return `true` or `false` depending on the validity of the comparison. Comparison operators include:

- strict equal (`===`)
- strict not equal (`!==`)
- greater than (`>`)
- less than (`<`)
- greater than or equal (`>=`)
- less than or equal (`<=`)

```
1 > 3 // false
3 > 1 // true
250 >= 250 // true
1 === 1 // true
1 === 2 // false
1 === '1' // false
```

AND `&&` Operator

The logical AND operator `&&` checks two values and returns a boolean. If *both* values are truthy, then it returns `true`. If one, or both, of the values is falsy, then it returns `false`.

```
true && true;      // true
1 > 2 && 2 > 1;    // false
true && false;     // false
4 === 4 && 3 > 1;  // true
```

OR `||` Operator

The logical OR operator `||` checks two values and returns a boolean. If one or both values are truthy, it returns `true`. If both values are falsy, it returns `false`.

```
true || false;        // true
10 > 5 || 10 > 20;    // true
false || false;       // false
10 > 100 || 10 > 20;  // false
```

NOT `!` Operator

The `!` operator can be used to do one of the following:

- Invert a Boolean value.
- Invert the truthiness of non-Boolean values.

```
// example 1
let value = true;
let oppositeValue = !value;
console.log(oppositeValue); // false

// example 2
const emptyString = '';
!emptyString; // true
const truthyNumber = 1;
!truthyNumber // false
```

JavaScript Ternary Operator

The ternary operator allows for a compact syntax in the case of binary (choosing between two choices) decisions. It accepts a condition followed by a `?` operator, and then two expressions separated by a `;`. If the condition evaluates to truthy, the first expression is executed, otherwise, the second expression is executed. It can be read as “IF condition THEN expression1 ELSE expression2”.

```
let price = 10.5;
let day = "Monday";

// The following examples produce the same result:

// A: if/else
if (day === "Monday") {
  price -= 1.5;
} else {
  price += 1.5;
}

// B: ternary operator
day === "Monday" ? price -= 1.5 : price += 1.5;
```