Conditional Statements

- Very often when you write code, you want to perform different actions for different decisions.
- In JavaScript we have the following conditional statements:

1. Use if to specify a block of code to be executed, if a specified condition is true
2. Use else to specify a block of code to be executed, if the same condition is false
3. Use else if to specify a new condition to test, if the first condition is false
4. Use switch to specify many alternative blocks of code to be executed

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*) {
  // *block of code to be executed if the condition is true*
}

- if is in lowercase letters. Uppercase letters (If or IF) will generate a JavaScript error.

The if Statement: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript if</h2>

<p>Display "Good day!" if the hour is less than 18:00:</p>

<p id="demo">Good Evening!</p>

<script>
if (new Date().getHours() < 18) {
  document.getElementById("demo").innerHTML = "Good day!";
}
</script>

</body>
</html>
```

The else Statement

- Use the else statement to specify a block of code to be executed if the condition is false.
- if (*condition*) {
  // block of code to be executed if the condition is true
  } else {
  // block of code to be executed if the condition is false
  }

The else Statement:Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript if .. else</h2>

<p>A time-based greeting:</p>

<p id="demo"></p>

<script>
const hour = new Date().getHours();
let greeting;

if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}

document.getElementById("demo").innerHTML = greeting;
</script>

</body>
</html>
```

The else if Statement

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

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  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
}
```

The else if Statement: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript if .. else</h2>

<p>A time-based greeting:</p>

<p id="demo"></p>

<script>
const time = new Date().getHours();
let greeting;
if (time < 10) {
  greeting = "Good morning";
} else if (time < 20) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
document.getElementById("demo").innerHTML = greeting;
</script>

</body>
</html>
```

The Switch Statement

- Use the switch statement to select one of many code blocks to be executed.
- Syntax:

```
switch(expression) {
    case x:
      // code block
      break;
    case y:
      // code block
      break;
    default:
      // code block
  }
```

- This is how it works:

-The switch expression is evaluated once.

-The value of the expression is compared with the values of each case.

-If there is a match, the associated block of code is executed.

-If there is no match, the default code block is executed.

- If multiple cases matches a case value, the **first** case is selected. If no matching cases are found, the program continues to the **default** label. If no default label is found, the program continues to the statement(s) **after the switch**.

The Switch Statement: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
let day;
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case  6:
    day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>

</body>
</html>
```

The break Keyword

- When JavaScript reaches a break keyword, it breaks out of the switch block.
- This will stop the execution inside the switch block.
- It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.
- **Note:** If you omit the break statement, the next case will be executed even if the evaluation does not match the case.

The default Keyword

- The default keyword specifies the code to run if there is no case match.
- The default case does not have to be the last case in a switch block.

The default Keyword: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript switch</h2>

<p id="demo"></p>

<script>
let text;
switch (new Date().getDay()) {
  case 6:
    text = "Today is Saturday";
    break;
  case 0:
    text = "Today is Sunday";
    break;
  default:
    text = "Looking forward to the Weekend";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript Loops

- Loops are handy, if you want to run the same code over and over again, each time with a different value.
- JavaScript supports different kinds of loops:

1. for - loops through a block of code a number of times
2. for/in - loops through the properties of an object
3. for/of - loops through the values of an iterable object
4. while - loops through a block of code while a specified condition is true
5. do/while - also loops through a block of code while a specified condition is true

The For Loop

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For Loop</h2>

<p id="demo"></p>

<script>
let text = "";

for (let i = 0; i < 5; i++) {
  text += "The number is " + i + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

Loop Scope

- Using var in a loop.
- Example

```
var i = 5;

  for (var i = 0; i < 10; i++) {
    // some code
  }

  // Here i is 10
```

Loop Scope

- Using let in a loop.
- Example

```
let i = 5;

  for (let i = 0; i < 10; i++) {
    // some code
  }

  // Here i is 5
```

The For In Loop

- The JavaScript for in statement loops through the properties of an Object.
- Syntax

```
for (key in object) {
    // code block to be executed
  }
```

The For In Loop: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For In Loop</h2>
<p>The for in statement loops through the properties of an object:</p>

<p id="demo"></p>

<script>
const person = {fname:"John", lname:"Doe", age:25};

let txt = "";
for (let x in person) {
  txt += person[x] + " ";
}

document.getElementById("demo").innerHTML = txt;
</script>

</body>
</html>
```

The For Of Loop

- The JavaScript for of statement loops through the values of an iterable object.
- Syntax

```
for (variable of iterable) {
  // code block to be executed
}
```

The For Of Loop: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript For Of Loop</h2>

<p>The for of statement loops through the values of an iterable object.</p>

<p id="demo"></p>

<script>
let language = "JavaScript";

let text = "";
for (let x of language) {
  text += x + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

The While Loop

- The while loop loops through a block of code as long as a specified condition is true.
- Syntax
- while (*condition*) {
  // code block to be executed
  }

The While Loop: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript While Loop</h2>

<p id="demo"></p>

<script>
let text = "";
let i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

The Do While Loop

- The do while loop is a variant of the while loop. This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.
- Syntax
- do {
  // code block to be executed
  }
  while (*condition*);

The Do While Loop: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Do While Loop</h2>

<p id="demo"></p>

<script>
let text = ""
let i = 0;

do {
  text += "<br>The number is " + i;
  i++;
}
while (i < 10);

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

The Break Statement

- You have already seen the break statement used in an earlier chapter of this tutorial. It was used to "jump out" of
  a switch() statement.
- The break statement can also be used to jump out of a loop.

The Break Statement: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Loops</h2>

<p>A loop with a <b>break</b> statement.</p>

<p id="demo"></p>

<script>
let text = "";
for (let i = 0; i < 10; i++) {
  if (i === 3) { break; }
  text += "The number is " + i + "<br>";
}

document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

The Continue Statement

- The continue statement breaks one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop.

The Continue Statement: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Loops</h2>

<p>A loop with a <b>continue</b> statement.</p>

<p>A loop which will skip the step where i = 3.</p>

<p id="demo"></p>

<script>
let text = "";
for (let i = 0; i < 10; i++) {
  if (i === 3) { continue; }
  text += "The number is " + i + "<br>";
}
document.getElementById("demo").innerHTML = text;
</script>

</body>
</html>
```

JavaScript Popup Boxes

JavaScript has three kind of popup boxes:

1. Alert box
2. Confirm box
3. Prompt box.

Alert Box

- An alert box is often used if you want to make sure information comes through to the user.
- When an alert box pops up, the user will have to click "OK" to proceed.
- Syntax:- window.alert("*sometext*");
- The window.alert() method can be written without the window prefix.

Alert Box: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Alert</h2>

<button onclick="myFunction()">Try it</button>

<script>
function myFunction() {
  alert("I am an alert box!");
}
</script>

</body>
</html>
```

Confirm Box

- A confirm box is often used if you want the user to verify or accept something.
- When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.
- If the user clicks "OK", the box returns **true**. If the user clicks "Cancel", the box returns **false**.
- Syntax:- window.confirm("*sometext*");
- The window.confirm() method can be written without the window prefix.

Confirm Box: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Confirm Box</h2>


<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  var txt;
  if (confirm("Press a button!")) {
    txt = "You pressed OK!";
  } else {
    txt = "You pressed Cancel!";
  }
  document.getElementById("demo").innerHTML = txt;
}
</script>

</body>
</html>
```

Prompt Box

- A prompt box is often used if you want the user to input a value before entering a page.
- When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.
- If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.
- Syntax:- window.prompt("*sometext*","*defaultText*");
- The window.prompt() method can be written without the window prefix.

Prompt Box: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Prompt</h2>

<button onclick="myFunction()">Try it</button>

<p id="demo"></p>

<script>
function myFunction() {
  let text;
  let person = prompt("Please enter your name:", "Harry Potter");
  if (person == null || person == "") {
    text = "User cancelled the prompt.";
  } else {
    text = "Hello " + person + "! How are you today?";
  }
  document.getElementById("demo").innerHTML = text;
}
</script>

</body>
</html>
```

JavaScript Objects

- Objects are variables too. But objects can contain many values.
- Object values are written as **name : value** pairs (name and value separated by a colon).
- A JavaScript object is a collection of **named values.**

JavaScript Objects: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>
<p>Creating an object:</p>

<p id="demo"></p>

<script>
const person = {
  firstName : "John",
  lastName  : "Doe",
  age     : 50,
  eyeColor  : "blue"
};

document.getElementById("demo").innerHTML = person.firstName + " " + person.lastName;
</script>

</body>
</html>
```

JavaScript Properties

- Properties are the values associated with a JavaScript object.
- A JavaScript object is a collection of unordered properties.
- Properties can usually be changed, added, and deleted, but some are read only.
- Accessing JavaScript Properties
- The syntax for accessing the property of an object is:

objectName.property   // person.age

or

objectName["*property*"]   // person["age"]

JavaScript Properties: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Object Properties</h2>
<p>Access a property with .property:</p>

<p id="demo"></p>

<script>
const person = {
  firstname: "John",
  lastname: "Doe",
  age: 50,
  eyecolor: "blue"
};

document.getElementById("demo").innerHTML = person.firstname + " is " + person.age + " years old.";
</script>

</body>
</html>
```

JavaScript Properties: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Object Properties</h2>
<p>Access a property with ["property"]:</p>

<p id="demo"></p>

<script>
const person = {
  firstname: "John",
  lastname: "Doe",
  age: 50,
  eyecolor: "blue"
};

document.getElementById("demo").innerHTML = person["firstname"] + " is " + person["age"] + " years old.";
</script>

</body>
</html>
```

Adding New Properties

- You can add new properties to an existing object by simply giving it a value.
- Assume that the person object already exists - you can then give it new properties:
- Example

```
person.nationality = "English";
```

Adding New Properties: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Object Properties</h2>
<p>Add a new property to an existing object:</p>

<p id="demo"></p>

<script>
const person = {
  firstname: "John",
  lastname: "Doe",
  age: 50,
  eyecolor: "blue"
};

person.nationality = "English";
document.getElementById("demo").innerHTML =
person.firstname + " is " + person.nationality + ".";
</script>

</body>
</html>
```

Deleting Properties

- The delete keyword deletes a property from an object by any ways given below:-
- Example

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

delete person.age;
```

- Example

```
const person = {
  firstName: "John",
  lastName: "Doe",
  age: 50,
  eyeColor: "blue"
};

delete person["age"];
```

Deleting Properties: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Object Properties</h2>
<p>Deleting object properties.</p>

<p id="demo"></p>

<script>
const person = {
  firstname: "John",
  lastname: "Doe",
  age: 50,
  eyecolor: "blue"
};

delete person.age;

document.getElementById("demo").innerHTML =
person.firstname + " is " + person.age + " years old.";
</script>

</body>
</html>
```

JavaScript Object Methods

- JavaScript methods are actions that can be performed on objects.
- In JavaScript, the this keyword refers to an **object**.
- **Which** object depends on how this is being invoked (used or called).
- If you access the fullName **property**, without (), it will return the **function definition.**

JavaScript Object Methods: Example

```
<!DOCTYPE html>
<html>
<body>

<h1>The JavaScript <i>this</i> Keyword</h1>
<p>In this example, <b>this</b> refers to the <b>person</b> object.</p>
<p>Because <b>fullName</b> is a method of the person object.</p>

<p id="demo"></p>

<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};

// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>

</body>
</html>
```

Using Built-In Methods: Example

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript Objects</h2>
<p id="demo"></p>

<script>
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
};
person.name = function() {
  return (this.firstName + " " + this.lastName).toUpperCase();
};

document.getElementById("demo").innerHTML =
"My father is " + person.name();
</script>

</body>
</html>
```