32. JS Conditions, Switch, Loops

Overview and purpose

In this lesson you will be learning about the following:

- JS Conditions
- JS Switch
- JS Loop For
- JS Loop While
- JS Break
- JS Type Conversion

JavaScript if else and else if

Conditional statements are used to perform different actions based on different conditions.

Conditional Statements

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
- Use switch to specify many alternative blocks of code to be executed

The switch statement is described later.

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
}
```

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

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

Activity 32A01 copy the following:

```
<html>
    <head>
        <title>32A01</title>
    </head>
    <body>
        <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 (condition1) {
    //  block of code to be executed if condition is true
} else {
    //  block of code to be executed if the condition is false
}
if (hour < 18) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

Activity 32A02 copy the following:

```
<html>
    <head>
        <title>32A02</title>
    </head>
    <body>
        <p>Click the button to display a time-based greeting:</p>
        <button onclick="myFunction()">Try it</button>
        <p id="demo"></p>
        <script>
            function myFunction() {
                var hour = new Date().getHours();
                var 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
}

if (time < 10) {
    greeting = "Good morning";
} else if (time < 20) {
    greeting = "Good day";
} else {
    greeting = "Good evening";
}
```

Activity 32A03 copy the following:

```
<html>
    <head>
        <title>32A03</title>
    </head>
    <body>
        <p>Click the button to display a time-based greeting:</p>
        <button onclick="myFunction()">Try it</button>
        <p id="demo"></p>
        <script>
            function myFunction() {
                var greeting;
                var time = new Date().getHours();
                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>
```

JavaScript Switch Statement - The switch statement is used to perform different actions based on different conditions.

The JavaScript 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.

Example

The getDay() method returns the weekday as a number between 0 and 6.
(Sunday=0, Monday=1, Tuesday=2 ..)
This example uses the weekday number to calculate the weekday name:

```
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";
}
```

The result of day will be: Sunday (Sunday is the day this was written on)

Activity 32A04 copy the following:

```
<html>
    <head>
        <title>32A04</title>
    </head>
    <body>
        <p id="demo"></p>
        <script>
            var 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 of inside the 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:

```
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";
}
```

(Written on a Sunday)

The default case does not have to be the last case in a switch block:

```
switch (new Date().getDay()) {
    default:
        text = "Looking forward to the Weekend";
        break;
    case 6:
        text = "Today is Saturday";
        break;
    case 0:
        text = "Today is Sunday";
}
```

If default is not the last case in the switch block, remember to end the default case with a break.

Activity 32A05 copy the following:

```
<html>
    <head>
        <title>32A05</title>
    </head>
    <body>
        <h2>JavaScript switch</h2>
        <p id="demo"></p>
        <script>
            var text;
            switch (new Date().getDay()) {
                default:
                    text = "Looking forward to the Weekend";
                    break;
                case 6:
                    text = "Saturday";
                    break;
                case 0:
                    text = "Sunday";
            }
            document.getElementById("demo").innerHTML = text;
        </script>
    </body>
</html>
```

Common Code Blocks - Sometimes you will want different switch cases to use the same code. In this example case 4 and 5 share the same code block, and 0 and 6 share another code block:

```
switch (new Date().getDay()) {
    case 4:
    case 5:
        text = "Soon it is Weekend";
        break;
    case 0:
    case 6:
        text = "It is Weekend";
        break;
    default:
        text = "Looking forward to the Weekend";
}
```

Activity 32A06 copy the following:

```
<html>
    <head>
        <title>32A06</title>
    </head>
    <body>
        <h2>JavaScript switch</h2>
        <p id="demo"></p>
        <script>
            var text;
            switch (new Date().getDay()) {
                case 4:
                case 5:
                    text = "Soon it is Weekend";
                    break;
                case 0:
                case 6:
                    text = "It is Weekend";
                    break;
                default:
                    text = "Looking forward to the Weekend";
            }
            document.getElementById("demo").innerHTML = text;
        </script>
    </body>
</html>
```

Switching Details - 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.

Strict Comparison - Switch cases use strict comparison (===). The values must be of the same type to match. A strict comparison can only be true if the operands are of the same type.
In this example there will be no match for x:

```
var x = "0";
switch (x) {
    case 0:
        text = "Off";
        break;
    case 1:
        text = "On";
        break;
    default:
        text = "No value found";
}
```

Activity 32A07 copy the following:

```
<html>
    <head>
       <title>32A07</title>
    </head>
    <body>
        <h2>JavaScript switch</h2>
        <p id="demo"></p>
        <script>
            var x = "0";
            switch (x) {
                case 0:
                    text = "Off";
                    break;
                case 1:
                    text = "On";
                    break;
                default:
                    text = "No value found";
            }
            document.getElementById("demo").innerHTML = text;
        </script>
    </body>
</html>
```

JavaScript For Loop - Loops can execute a block of code a number of times.

JavaScript Loops - Loops are handy, if you want to run the same code over and over again, each time with a different value.
Often this is the case when working with arrays:

```
text += cars[0] + "<br>";
text += cars[1] + "<br>";
text += cars[2] + "<br>";
text += cars[3] + "<br>";
text += cars[4] + "<br>";
text += cars[5] + "<br>";
```

You can write:

```
var i;
for (i = 0; i < cars.length; i++) {
    text += cars[i] + "<br>";
}
```

Different Kinds of Loops - JavaScript supports different kinds of loops:

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

The For Loop - The for loop has the following syntax:

```
for (statement 1; statement 2; statement 3) {
    // code block to be executed
}
```

Statement 1 is executed (one time) before the execution of the code block.
Statement 2 defines the condition for executing the code block.
Statement 3 is executed (every time) after the code block has been executed

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

From the example above, you can read:
Statement 1 sets a variable before the loop starts (var i = 0).
Statement 2 defines the condition for the loop to run (i must be less than 5).
Statement 3 increases a value (i++) each time the code block in the loop has been executed.

Activity 32A08 copy the following:

```
<html>
    <head>
        <title>32A08</title>
    </head>
    <body>
        <h2>JavaScript For Loop</h2>
        <p id="demo"></p>
        <script>
            var text = "";
            var i;
            for (i = 0; i < 5; i++) {
                text += "The number is " + i + "<br>";
            }
            document.getElementById("demo").innerHTML = text;
        </script>
    </body>
</html>
```

Statement 1
Normally you will use statement 1 to initialize the variable used in the loop (i = 0). This is not always the case, JavaScript doesn't care. Statement 1 is optional. You can initiate many values in statement 1 (separated by comma):

```
for (i = 0, len = cars.length, text = ""; i < len; i++) {
    text += cars[i] + "<br>";
}
```

And you can omit statement 1 (like when your values are set before the loop starts):

```
var i = 2;
var len = cars.length;
var text = "";
for (; i < len; i++) {
    text += cars[i] + "<br>";
}
```

Statement 2 - Often statement 2 is used to evaluate the condition of the initial variable. This is not always the case, JavaScript doesn't care. Statement 2 is also optional. If statement 2 returns true, the loop will start over again, if it returns false, the loop will end.

If you omit statement 2, you must provide a break inside the loop. Otherwise the loop will never end. This will crash your browser. Read about breaks in a later chapter of this tutorial.

Statement 3 - Often statement 3 increments the value of the initial variable. This is not always the case, JavaScript doesn't care, and statement 3 is optional. Statement 3 can do anything like negative increment (i--), positive increment (i = i + 15), or anything else. Statement 3 can also be omitted (like when you increment your values inside the loop):

```
var i = 0;
var len = cars.length;
for (; i < len; ) {
    text += cars[i] + "<br>";
    i++;
}
```

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

```
var person = {fname:"John", lname:"Doe", age:25};
var text = "";
var x;
for (x in person) {
    text += person[x];
}
```

Activity 32A09 copy the following:

```
<html>
    <head>
        <title>32A09</title>
    </head>
    <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>
            var txt = "";
            var person = {fname:"John", lname:"Doe", age:25};
            var x;
            for (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 objects
for/of lets you loop over data structures that are iterable such as Arrays, Strings, Maps, NodeLists, and more.
The for/of loop has the following syntax:

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

variable - For every iteration the value of the next property is assigned to the variable.
Variable can be declared with const, let, or var.
iterable - An object that has iterable properties.

Looping over an Array

```
var cars = ['BMW', 'Volvo', 'Mini'];
var x;
for (x of cars) {
    document.write(x + "<br >");
}
```

Looping over a String

```
var txt = 'JavaScript';
var x;
for (x of txt) {
    document.write(x + "<br >");
}
```

The While Loop - The while loop and the do/while loop will be explained in the next chapter.

JavaScript While Loop - Loops can execute a block of code as long as a specified condition is true.

The While Loop - The while loop loops through a block of code as long as a specified condition is true.

In the following example, the code in the loop will run, over and over again, as long as a variable (i) is less than 10:

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

If you forget to increase the variable used in the condition, the loop will never end. This will crash your browser.

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.

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

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

Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Comparing For and While - If you have read the previous chapter, about the for loop, you will discover that a while loop is much the same as a for loop, with statement 1 and statement 3 omitted. The loop in this example uses a for loop to collect the car names from the cars array:

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";
for (;cars[i];) {
    text += cars[i] + "<br>";
    i++;
}
```

Activity 32A10 copy the following:

```
<html>
    <head>
        <title>32A10</title>
    </head>
    <body>
        <p id="demo"></p>
        <script>
            var cars = ["BMW", "Volvo", "Saab", "Ford"];
            var i = 0;
            var text = "";
            for (;cars[i];) {
                text += cars[i] + "<br>";
                i++;
            }
            document.getElementById("demo").innerHTML = text;
        </script>
    </body>
</html>
```

The loop in this example uses a while loop to collect the car names from the cars array:

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
var i = 0;
var text = "";
while (cars[i]) {
    text += cars[i] + "<br>";
    i++;
}
```

JavaScript Break and Continue - The break statement "jumps out" of a loop. The continue statement "jumps over" one iteration in the loop.

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 breaks the loop and continues executing the code after the loop (if any):

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

Activity 32A11 copy the following:

```
<html>
    <head>
        <title>32A11</title>
    </head>
    <body>
        <h2>JavaScript Loops</h2>
        <p>A loop with a <b>break</b> statement.</p>
        <p id="demo"></p>
        <script>
            var text = "";
            var i;
            for (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 skips one iteration (in the loop), if a specified condition occurs, and continues with the next iteration in the loop. This example skips the value of 3:

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

JavaScript Labels

To label JavaScript statements you precede the statements with a label name and a colon:

```
label:
statement
```

The break and the continue statements are the only JavaScript statements that can "jump out of" a code block.
Syntax:

```
break labelName;

continue labelName;
```

The continue statement (with or without a label reference) can only be used to skip one loop iteration. The break statement, without a label reference, can only be used to jump out of a loop or a switch. With a label reference, the break statement can be used to jump out of any code block:

```
var cars = ["BMW", "Volvo", "Saab", "Ford"];
list: {
    text += cars[0] + "<br>";
    text += cars[1] + "<br>";
    break list;
    text += cars[2] + "<br>";
    text += cars[3] + "<br>";
}
```

Activity 32A12 copy the following:

```
<html>
    <head>
        <title>32A12</title>
    </head>
    <body>
        <h2>JavaScript break</h2>
        <p id="demo"></p>
        <script>
            var cars = ["BMW", "Volvo", "Saab", "Ford"];
            var text = "";
            list : {
                text += cars[0] + "<br>";
                text += cars[1] + "<br>";
                break list;
                text += cars[2] + "<br>";
                text += cars[3] + "<br>";
            }
            document.getElementById("demo").innerHTML = text;
        </script>
    </body>
</html>
```

A code block is a block of code between { and }.

Make the loop stop when i is 5.

```
for (i = 0; i < 10; i++) {
  console.log(i);
  if (i == 5) {
     ;
  }
}
```

JavaScript Type Conversion - Number() converts to a Number, String() converts to a String, Boolean() converts to a Boolean.

JavaScript Data Types - In JavaScript there are 5 different data types that can contain values:	• `string` • `number` • `boolean` • `object` • `function`
There are 6 types of objects:	• `Object` • `Date` • `Array` • `String` • `Number` • `Boolean`
And 2 data types that cannot contain values:	• `null` • `undefined`

The typeof Operator - You can use the typeof operator to find the data type of a JavaScript variable.

```
typeof "John"                 // Returns "string"
typeof 3.14                   // Returns "number"
typeof NaN                    // Returns "number"
typeof false                  // Returns "boolean"
typeof [1,2,3,4]              // Returns "object"
typeof {name:'John', age:34}  // Returns "object"
typeof new Date()             // Returns "object"
typeof function () {}         // Returns "function"
typeof myCar                  // Returns "undefined" *
typeof null                   // Returns "object"
```

Please observe:

- The data type of NaN is number
- The data type of an array is object
- The data type of a date is object
- The data type of null is object
- The data type of an undefined variable is **undefined** *
- The data type of a variable that has not been assigned a value is also **undefined** *

You cannot use `typeof` to determine if a JavaScript object is an array (or a date).

The Data Type of typeof - The typeof operator is not a variable. It is an operator. Operators (+ - * /) do not have any data type. But, the typeof operator always returns a string (containing the type of the operand).

The constructor Property - The constructor property returns the constructor function for all JavaScript variables.

```
"John".constructor                 // Returns function String()  {[native code]}
(3.14).constructor                 // Returns function Number()  {[native code]}
false.constructor                  // Returns function Boolean() {[native code]}
[1,2,3,4].constructor              // Returns function Array()   {[native code]}
{name:'John',age:34}.constructor   // Returns function Object()  {[native code]}
new Date().constructor             // Returns function Date()    {[native code]}
function () {}.constructor         // Returns function Function(){[native code]}
```

You can check the constructor property to find out if an object is an Array (contains the word "Array"):

```
function isArray(myArray) {
    return myArray.constructor.toString().indexOf("Array") > -1;
}
```

Activity 32A13 copy the following:

```
<html>
    <head>
        <title>32A13</title>
    </head>
    <body>
        <h2>JavaScript Arrays</h2>
        <p>This "home made" isArray() function returns true when used on an array:</p>
        <p id="demo"></p>
        <script>
            var fruits = ["Banana", "Orange", "Apple", "Mango"];
            document.getElementById("demo").innerHTML = isArray(fruits);
            function isArray(myArray) {
                return myArray.constructor.toString().indexOf("Array") > -1;
            }
        </script>
    </body>
</html>
```

Or even simpler, you can check if the object is an Array function:

```
function isArray(myArray) {
    return myArray.constructor === Array;
}
```

You can check the constructor property to find out if an object is a Date (contains the word "Date"):

```
function isDate(myDate) {
    return myDate.constructor.toString().indexOf("Date") > -1;
}
```

Or even simpler, you can check if the object is a Date function:

```
function isDate(myDate) {
    return myDate.constructor === Date;
}
```

JavaScript Type Conversion

JavaScript variables can be converted to a new variable and another data type:

- By the use of a JavaScript function
- **Automatically** by JavaScript itself

Converting Numbers to Strings

The global method String() can convert numbers to strings. It can be used on any type of numbers, literals, variables, or expressions:

```
String(x)         // returns a string from a number variable x
String(123)       // returns a string from a number literal 123
String(100 + 23)  // returns a string from a number from an expression
```

The Number method toString() does the same.

```
x.toString()
(123).toString()
(100 + 23).toString()
```

Method	Description
toExponential()	Returns a string, with a number rounded and written using exponential notation.
toFixed()	Returns a string, with a number rounded and written with a specified number of decimals.
toPrecision()	Returns a string, with a number written with a specified length

Converting Booleans to Strings - The global method String() can convert booleans to strings.

```
String(false)       // returns "false"
String(true)        // returns "true"
```

The Boolean method toString() does the same.

```
false.toString()    // returns "false"
true.toString()     // returns "true"
```

Converting Dates to Strings - The global method String() can convert dates to strings.

```
String(Date())   // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)"
```

The Date method toString() does the same.

```
Date().toString()   // returns "Thu Jul 17 2014 15:38:19 GMT+0200 (W. Europe Daylight Time)"
```

Method	Description
getDate()	Get the day as a number (1-31)
getDay()	Get the weekday a number (0-6)
getFullYear()	Get the four digit year (yyyy)
getHours()	Get the hour (0-23)
getMilliseconds()	Get the milliseconds (0-999)
getMinutes()	Get the minutes (0-59)
getMonth()	Get the month (0-11)
getSeconds()	Get the seconds (0-59)
getTime()	Get the time (milliseconds since January 1, 1970)

Converting Strings to Numbers - The global method Number() can convert strings to numbers. Strings containing numbers (like "3.14") convert to numbers (like 3.14). Empty strings convert to 0. Anything else converts to NaN (Not a Number).

```
Number("3.14")    // returns 3.14
Number(" ")       // returns 0
Number("")        // returns 0
Number("99 88")   // returns NaN
```

Method	Description
parseFloat()	Parses a string and returns a floating point number
parseInt()	Parses a string and returns an integer

The Unary + Operator - The unary + operator can be used to convert a variable to a number:

```
var y = "5";      // y is a string
var x = + y;      // x is a number
```

Activity 32A14 copy the following:

```
<html>
    <head>
        <title>32A14</title>
    </head>
    <body>
        <h2>The JavaScript typeof Operator</h2>
        <p>The typeof operator returns the type of a variable or expression.</p>
        <button onclick="myFunction()">Try it</button>
        <p id="demo"></p>
        <script>
            function myFunction() {
                var y = "5";
                var x = + y;
                document.getElementById("demo").innerHTML = typeof y + "<br>" + typeof x;
            }
        </script>
    </body>
</html>
```

If the variable cannot be converted, it will still become a number, but with the value NaN (Not a Number):

```
var y = "John";   // y is a string
var x = + y;      // x is a number (NaN)
```

Converting Booleans to Numbers - The global method Number() can also convert booleans to numbers.

```
Number(false)     // returns 0
Number(true)      // returns 1
```

Converting Dates to Numbers - The global method Number() can be used to convert dates to numbers.

```
d = new Date();
Number(d)          // returns 1404568027739
```

The date method getTime() does the same.

```
d = new Date();
d.getTime()        // returns 1404568027739
```

Automatic Type Conversion - When JavaScript tries to operate on a "wrong" data type, it will try to convert the value to a "right" type. The result is not always what you expect:

```
5 + null    // returns 5         because null is converted to 0
"5" + null  // returns "5null"   because null is converted to "null"
"5" + 2     // returns "52"      because 2 is converted to "2"
"5" - 2     // returns 3         because "5" is converted to 5
"5" * "2"   // returns 10        because "5" and "2" are converted to 5 and 2
```

Automatic String Conversion - JavaScript automatically calls the variable's toString() function when you try to "output" an object or a variable:

```
document.getElementById("demo").innerHTML = myVar;
```

```
// if myVar = {name:"Fjohn"}  // toString converts to "[object Object]"
// if myVar = [1,2,3,4]       // toString converts to "1,2,3,4"
// if myVar = new Date()      // toString converts to "Fri Jul 18 2014 09:08:55 GMT+0200"
```

Numbers and booleans are also converted, but this is not very visible:

```
// if myVar = 123             // toString converts to "123"
// if myVar = true            // toString converts to "true"
// if myVar = false           // toString converts to "false"
```

JavaScript Type Conversion Table - This table shows the result of converting different JavaScript values to Number, String, and Boolean:

Original Value	Converted to Number	Converted to String	Converted to Boolean
false	0	"false"	false
true	1	"true"	true
0	0	"0"	false
1	1	"1"	true
"0"	0	"0"	**true**
"000"	0	"000"	**true**
"1"	1	"1"	true
NaN	NaN	"NaN"	false
Infinity	Infinity	"Infinity"	true
-Infinity	-Infinity	"-Infinity"	true
""	**0**	""	**false**
"20"	20	"20"	true
"twenty"	NaN	"twenty"	true
[ ]	**0**	""	true
[20]	**20**	"20"	true
[10,20]	NaN	"10,20"	true
["twenty"]	NaN	"twenty"	true
["ten","twenty"]	NaN	"ten,twenty"	true
function(){}	NaN	"function(){}"	true
{ }	NaN	"[object Object]"	true
null	**0**	"null"	false
undefined	NaN	"undefined"	false

Values in quotes indicate string values. **Red values** indicate values (some) programmers might not expect.

JavaScript Object.keys() - Object.keys() returns the keys (properties) of any object type.

Syntax

```
Object.keys(object)
```

Activity 32A15 copy the following:

```
<html>
    <head>
        <title>32A15</title>
    </head>
    <body>
        <h1>JavaScript Objects</h1>
        <h2>The Object.keys() Method</h2>

        <p>Object.keys() returns an enumerable array of the keys of an object:</p>

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

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

        // Get the Keys
        const keys = Object.keys(object);
        document.getElementById("demo").innerHTML = keys;
        </script>
    </body>
</html>
```

- The Object.keys() method returns an array with the keys of an object.
- The Object.keys() method does not change the original object.

JavaScript Object.values() - Object.values() returns the values of all object keys(properties).

Syntax

```
Object.values(object)
```

Activity 32A16 copy the following:

```
<html>
    <head>
        <title>32A16</title>
    </head>
    <body>
        <h1>JavaScript Objects</h1>
        <h2>The Object.values() Method</h2>

        <p>Object.values() returns an array of the values of an object.</p>

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

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

        let text = Object.values(person);
        document.getElementById("demo").innerHTML = text;
        </script>
    </body>
</html>
```

- The Object.values() method returns an array of the property values of an object.
- The Object.values() method does not change the original object.

JavaScript Object.entries() – Object.entries() returns the keys and values of any object types.

Syntax

```
Object.entries(object)
```

Activity 32A17 copy the following:

```
<html>
    <head>
        <title>32A17</title>
    </head>
    <body>
        <h1>JavaScript Objects</h1>
        <h2>Object.entries() Method</h2>

        <p>Object.entries() returns an array of the key/value pairs of an object.</p>

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

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

        let text = Object.entries(person);
        document.getElementById("demo").innerHTML = text;
        </script>
    </body>
</html>
```

- The Object.entries() method returns an array of the key/value pairs of an object.
- The Object.entries() method does not change the original object.

The methods above return an **Iterable** (enumerable array).

Iterables makes it simpler to use objects in loops and to convert objects into maps.

JavaScript Object.assign() – The Object.assign() method copies properties from one or more source objects to a target object.

Syntax

```
Object.assign(target, source(s))
```

Activity 32A18 copy the following:

```
<html>
    <head>
        <title>32A18</title>
    </head>
    <body>
        <h1>JavaScript Object</h1>
        <h2>The Object.assign() Method</h2>

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

        <script>
        // Create Target Object
        const person1 = {
          firstName: "John",
          lastName: "Doe",
          age: 50,
          eyeColor: "blue"
        };

        // Create Source Object
        const person2 = {firstName: "Anne",lastName: "Smith"};

        // Assign Source to Target
        Object.assign(person1, person2);

        // Display Target
        let text = Object.entries(person1);
        document.getElementById("demo").innerHTML = text;
        </script>

    </body>
</html>
```

The JavaScript Spread Operator(…) – The JavaScript spread operator (...) expands an iterable (like an array) into more elements.
This allows us to quickly copy all or parts of an existing array into another array.

```
const numbersOne = [1, 2, 3];
const numbersTwo = [4, 5, 6];
const numbersCombined = [...numbersOne, ...numbersTwo];
```

We can use the spread operator with objects too:

```
const myVehicle = {
    brand: 'Ford',
    model: 'Mustang',
    color: 'red'
};

const updateMyVehicle = {
    type: 'car',
    year: 2021,
    color: 'yellow'
};

const myUpdatedVehicle = {...myVehicle, ...updateMyVehicle};
```

Activity 32A19 copy the following:

```
<html>
    <head>
        <title>32A19</title>
    </head>
    <body>
        <script>
            const myVehicle = {
              brand: 'Ford',
              model: 'Mustang',
              color: 'red'
            };

            const updateMyVehicle = {
              type: 'car',
              year: 2021,
              color: 'yellow'
            };

            const myUpdatedVehicle = {...myVehicle, ...updateMyVehicle};

            // Check the result object in the console:
            console.log(myUpdatedVehicle);
        </script>
        <p>Press F12 and see the result object in the console view.</p>
    </body>
</html>
```

Notice the properties that did not match were combined, but the property that did match, `color`, was overwritten by the last object that was passed, `updateMyVehicle`. The resulting color is now yellow.

Exercises:

32E01

Exercise:

Fix the if statement to alert "Hello World" if `x` is greater than `y`.

```
if  x > y
  alert("Hello World");
```

32E02

Exercise:

Fix the if statement to alert "Hello World" if `x` is greater than `y`, otherwise alert "Goodbye".

```
  (x  y) {
  alert("Hello World");
}    {
  alert("Goodbye");
}
```

32E03

Exercise:

Create a `switch` statement that will alert "Hello" if `fruits` is "banana", and "Welcome" if `fruits` is "apple".

```
    (fruits) {
     "Banana":
    alert("Hello")
    break;
     "Apple":
    alert("Welcome")
    break;
}
```

32E04

Exercise:

Add a section that will alert("Neither") if `fruits` is neither "banana" nor "apple".

```
switch(fruits) {
  case "Banana":
    alert("Hello")
    break;
  case "Apple":
    alert("Welcome")
    break;

    alert("Neither");
}
```

32E05

Exercise:

Create a loop that runs from 0 to 9.

```
var i;
    (  =  ;  <  ;  ) {
  console.log(i);
}
```

32E06

Exercise:

Create a loop that runs through each item in the `fruits` array.

```
var fruits = ["Apple", "Banana", "Orange"];
for (x      ) {
  console.log(x);
}
```

32E07

Create a loop that runs as long as i is less than 10.

```
var i = 0;
     (i  10) {
  console.log(i);
  i++
}
```

32E08

Create a loop that runs as long as i is less than 10, but increase i with 2 each time.

```
var i = 0;
while (i < 10) {
  console.log(i);
   =    ;
}
```

32E09

Make the loop stop when i is 5.

```
for (i = 0; i < 10; i++) {
  console.log(i);
  if (i == 5) {
     ;
  }
}
```

32E10

Make the loop jump to the next iteration when i is 5.

```
for (i = 0; i < 10; i++) {
  if (i == 5) {
      ;
  }
  console.log(i);
}
```

32E11

In this exercise, you will collect the fruit names from the fruit array: "Mango", "Strawberry", "Kiwi", "Lemon". Use activity 10 as your reference.

32E12

Write a function that will console.log positive, negative, or zero depending if the number inputed is a positive number, a negative number or zero.

32E13

Create a function that will receive two parameters **id#** and **password**. If **id#** is 123 and **password** is 'happybirthday' console.log 'accepted'. Otherwise 'denied'.

32E14

Write a function that will accept a number parameter between the range of 0 to 100. Console.log out F (If the number is less than 50), C (If its between 50 to 72), B (If its between 73 to 86) and A (If its greater that 86) based on the grade.

32E15

Write a function that takes as a parameter the number of wheels a bike has and prints out the following depending on the number of wheels:

1.unicycle
2.bicycle
3.tricycle
4.quadracycle

for any other input, print "Hmm idk what to call that" using console.log.

32E16
Create an array numbers [1, 2, 3, 4, 5, 6], use a loop to double all values in the array.

32E17
Create a variable “thisisalongstring” use a loop to convert it into an array with a letter in each index.

32E18
console.log the numbers 1 to 10 using a **for** loop, a **while** loop, and a **do while** loop.

32E19
Write a loop that prints out all the numbers from 1 to 10 except number 4.

32E20
Create an array [1, 3, 2, 4, 6] loop through it, print out the index of the number 4 and **break** out of the loop.

32E21
Create a webpage and a function in the page that displays the **keys** and **values** of an inputed object on the page.
For example, running the function with the parameter {a: 1, b: true, cde: "a", f: null, gh: undefined}
Should make the following shown on the page:

- **a**: 1
- **b**: true
- **cde**: a
- **f**: null
- **gh**: undefined

32E22
Create a function called **objTypes** that takes an object and returns a **new object** with the same **keys** but the **values** should be the types of the **original values**.
For example:
objTypes({ a: 1, b: true, cde: "a", f: null, gh: undefined }) should return **{ a: "number", b: "Boolean", cde: "string", f: "object", gh: "undefined"}**.

32E23
Create a function that takes an **array of objects** and a **key.** This function will sort all the objects in the array by the inputed **key**.
For example, calling the function with the **array:**
[{ a: 1, c: "a" },
{ a: 1, c: "ab" },
{ a: 2, c: "c" },
{ a: 3, c: "de" },
{ a: 3, c: "3", c: [] }]
and the **key** "a" should return
{
"1": [{ a: 1, c: "a" }, { a: 1, c: "ab" }],
"2": [{ a: 2, c: "c" }],
"3": [{ a: 3, c: "de" }, { a: 3, c: "3", c: [] }]
}

32E24
Create a function called **objInfo** that takes an object and returns an object with the properties **keys** and **values**. These properties should have an array and inside that array, the **keys** and **values** of the inputed object respectively. However, if the parameter **is** an **array** instead of a normal object, it should return the string "Error: Input was an array" instead.

For example,
objInfo({ a: 1, b: 2, c: "3" }) should return
{ keys: ["a", "b", "c"], values: [1, 2, "3"] }
And
objInfo([1,2,3,4]) should return
"Error: Input was an array"

32E25
Create a function called **superAdd**. This function should take two parameters and return them added. However, make the function have the following behavior:
If the two inputs are numbers, return them added as a number: **superAdd(1, 2) == 3**
If the two inputs are strings, return them **added** (not concatenated) as a string: **superAdd("1", "2") == "3"**
If the two inputs are booleans, return false if they're both the same, and true if they're different:
superAdd(true, false) == true
superAdd(false, false) == false
superAdd(true, true) == false
If the two inputs are undefined, return undefined.
If the two inputs are null, return null.
If the two inputs are arrays, return them combined together: **superAdd([1,2], [3, true])** // **[1, 2, 3, true]**
If the two inputs are objects, return them combined together, but don't change the original objects.
superAdd({a:1,b:2,c:""}, {d: false, e: [], a: 10})
returns: **{ a: 10, b: 2, c: "", d: false, e: [] }**
If the two inputs are dates, return a new date with the original milliseconds added together.
superAdd(new Date(10000), new Date(2000000))
should be the same Date as **new Date(2010000)**.
If the two inputs are anything **else** (such as a string and a number), return the string "error".

32E26
Create a function called **entries**. This function should take an object parameter and return an array with arrays that are the object's keys and values.
For example, **entries({a: 1, b: true, c: "abc"})** should return
[["a", 1], ["b", true], ["c", "abc"]]

32E27
Create a function called **oneLetterKeys**. Make it take an object parameter and make it return an object with the same content but excluding keys with more than one letter.
For example,
oneLetterKeys({a: 2, b: 3, cd: 4, de: 5, ef: 6, fg: 7, h: 8, i: 9})
should return **{a: 2, b: 3, h: 8, i: 9}**

32E28
Create a function called **flipObject**. Make it take an object parameter and return an object with the original object's values as keys and the keys as values.
For example:
flipObject({a: 2, b: 4, c: "t"})
should return **{"2": "a", "4": "b", t: "c"}**
and
flipObject({a: 2, b: undefined, c: null, d: true, e: 2})
should return **{"2": "e", undefined: "b", null: "c", true: "d"}**
(Notice that the first key of the return, "2" is the same input value for both key "a" and "e". When passing it through the function, they will overlap, this is fine.)