8.8 Lesson Overview

- Including Scripts
- Strings
- Operators
- Value comparisons
- Control structures

9 External Script

- Script tag with **src** attribute.
- Almost always added in the body.
- Added after all content.

```
<!DOCTYPE html>
<html>
  <head>
    <title>My Page</title>
  </head>
  <body>
        <h1>Hello SEDC!</h1>
        <script src="myScript.js"></script>
  </body>
</html>
```

10 Strings

10.1 Combining Strings "Concatenation"

- Using the '**+**' operator.

  Ex. `"SEDC" + " " + "2018"`

- Works with string and number as well, Known as automatic type conversion.

  Ex. `"SEDC" + " " + 2018`
  Ex. `"2" + 4 = ?`

- Also variables

  Ex.
  ```
  var sedc = "SEDC";
      sedc + " " + 2018
  ```
  Ex. `` `Hello ${sedc} Academy 2018` ``

10.2 Quotes within Strings

- "It's really nice to be a programmer" ✓

- 'It's really nice to be a programmer' ✗
- "It's really nice to be a programmer" ✓
- 'It\'s really nice to be a programmer' ✓

II COMPARISON OPERATORS

- The result is always a Boolean value.
- 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	*Description*	*Comparing*	*Returns*
==	equal to	x == 8	false
		x == 5	true
		x == "5"	true
===	equal value and equal type	x === 5	true
		x === "5"	false
!=	not equal	x != 8	true
!==	not equal value or not equal type	x !== 5	false
		x !== "5"	true
		x !== 8	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

11.1 LOGICAL OPERATORS

- Logical operators are typically used with Boolean (logical) values, and they return a Boolean value.
- When used with non-Boolean values, they may return a non-Boolean value
- 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	Description	Example
&&	and	(x < 10 && y > 1) is true
||	or	(x == 5 || y == 5) is false
!	not	!(x == y) is true

11.2 CONDITIONAL (TERNARY) OPERATOR

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

```
variablename = (condition) ? value1:value2
```

Example:

```
var voteable = (age < 18) ? "Too young":"Old enough";
```

If the variable age is a value below 18, the value of the variable **voteable** will be "Too young", otherwise the value of **voteable** will be "Old enough".

11.3 COMPARING DIFFERENT TYPES

- Comparing data of different types may give unexpected results.
- When comparing a string with a number, JavaScript will convert the string to a number when doing the comparison.
- An empty string converts to 0.
- A non-numeric string converts to NaN which is always false.

Case	Value
2 < 12	true
2 < "12"	true
2 < "John"	false
2 > "John"	false
2 == "John"	false
"2" < "12"	false
"2" > "12"	true
"2" == "12"	false

- When comparing two strings, "2" will be greater than "12", because (alphabetically) 1 is less than 2.
- To secure a proper result, variables should be converted to the proper type before comparison:

```
age = Number(age);
if (isNaN(age)) {
    voteable = "Input is not a number";
} else {
    voteable = (age < 18) ? "Too young" : "Old enough";
}
```

11.3.1 Structure

- [operand] [comparison operator] [operand]

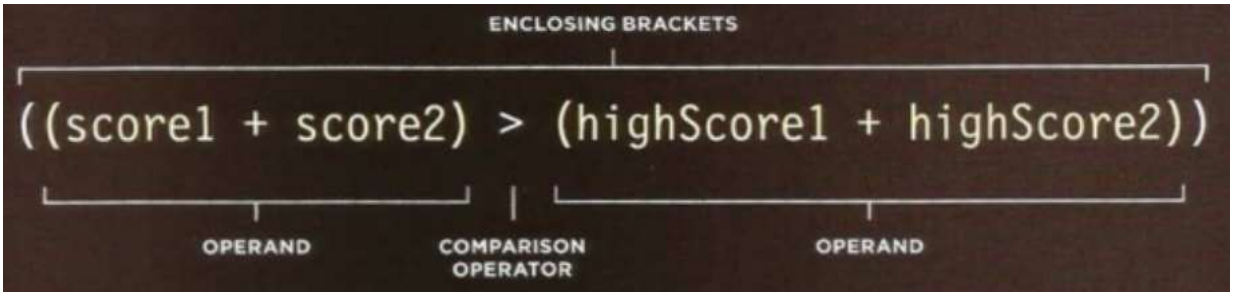
```
var pass = 50;  //pass mark
var score = 90; //score
var hasPassed = score >= pass; //true
```

- Comparison operators return Boolean values

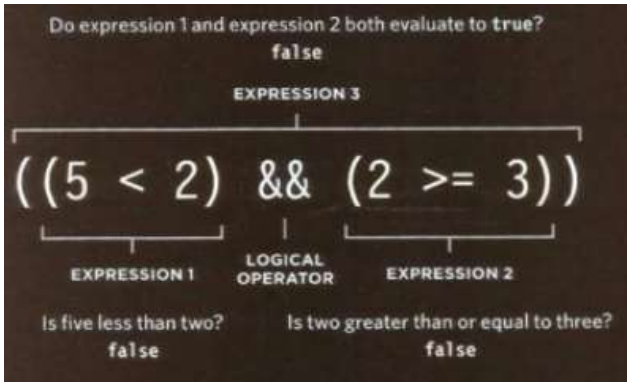
```
//An operand can be expression
( (score1 + score2) > (hightScore1 + heightScore2) )
//operand    //operator  //operand
```

```
//An operand can be expression
( (5 < 2) && (2 >= 3) ) // reuslt: false
//expression1   //expression2
```

- The operand does not have to be a single value or variable name.
- An operand can be an expression (because each expression evaluates into a single value).



- Comparison operators usually return single values of true or false.
- Logical operators allow you to compare the results of more than one comparison operator.



- In this one of code are three expressions, each of which will resolve to the value true or false.
- The expression on the left and the right both use comparison operators, and both return false.
- The third expression uses a logical operator (rather than a comparison operator). The logical AND operator checks to see whether both expressions on either side of it return true (in this case they do not, so it evaluates to false).

12 RULES ON TRUE/FALSE

- FALSE/TRUE - NOT AS YOU KNOW IT

12.1 FALSY VALUES:

- false
- 0, -0 (as number), "0"
- "" (empty string)
- null
- undefined
- NaN (invalid number)

12.2 TRUTHY VALUES:

- true
- "hello"
- 25
- [ ], [1, "2", 3] (arrays)
- { }, { a: 42 } (objects)
- function foo() { .. } (functions)
- evrything else :)

12.3 LOGICAL AND (&&)

```
a1 = true  && true        // t && t returns true
a2 = true  && false       // t && f returns false
a3 = false && true        // f && t returns false
a4 = false && (3 == 4)    // f && f returns false
a5 = 'Cat' && 'Dog'       // t && t returns "Dog"
a6 = false && 'Cat'       // f && t returns false
a7 = 'Cat' && false       // t && f returns false
a8 = ''    && false       // f && f returns ""
a9 = false && ''          // f && f returns false
```

12.4 LOGICAL OR (||)

```
o1 = true  || true        // t || t returns true
o2 = false || true        // f || t returns true
o3 = true  || false       // t || f returns true
o4 = false || (3 == 4)    // f || f returns false
o5 = 'Cat' || 'Dog'       // t || t returns "Cat"
o6 = false || 'Cat'       // f || t returns "Cat"
o7 = 'Cat' || false       // t || f returns "Cat"
o8 = ''    || false       // f || f returns false
o9 = false || ''          // f || f returns ""
```

12.5 LOGICAL NOT (!)

```
n1 = !true                // !t returns false
n2 = !false               // !f returns true
n3 = !'Cat'               // !t returns false
```

13 NUMBERS

- JavaScript has only one type of number. Numbers can be written with or without decimals.
- Extra large or extra small numbers can be written with scientific (exponent) notation:

Example

```
var x = 123e5;    // 12300000
var y = 123e-5;   // 0.00123
```

13.1 JAVASCRIPT NUMBERS ARE ALWAYS 64-BIT FLOATING POINT

- Unlike many other programming languages, JavaScript does not define different types of numbers, like integers, short, long, floating-point etc.
- JavaScript numbers are always stored as double precision floating point numbers, following the international IEEE 754 standard. This format stores numbers in 64 bits, where the number (the fraction) is stored in bits 0 to 51, the exponent in bits 52 to 62, and the sign in bit 63:

Value (aka Fraction/Mantissa)	*Exponent*
52 bits (0 - 51)	11 bits (52 - 62)

13.2 PRECISION

- Integers (numbers without a period or exponent notation) are accurate up to 15 digits.

```
var x = 999999999999999;   // x will be 999999999999999
var y = 9999999999999999;  // y will be 10000000000000000
```

- The maximum number of decimals is 17, but floating point arithmetic is not always 100% accurate:

```
var x = 0.2 + 0.1;         // x will be 0.30000000000000004
```

- To solve the problem above, it helps to multiply and divide:

```
var x = (0.2 * 10 + 0.1 * 10) / 10;       // x will be 0.3
```

13.3 ADDING NUMBERS AND STRINGS

❖ *WARNING !!*

➢ *JavaScript uses the + operator for both addition and concatenation.*

➢ *Numbers are added. Strings are concatenated.*

- If you add two numbers, the result will be a number:

```
var x = 10;
var y = 20;
var z = x + y;            // z will be 30 (a number)
```

- If you add two strings, the result will be a string concatenation:

```
var x = "10";
var y = "20";
var z = x + y;            // z will be 1020 (a string)
```

- If you add a number and a string, the result will be a string concatenation:

```
var x = 10;
var y = "20";
var z = x + y;            // z will be 1020 (a string)
```

- If you add a string and a number, the result will be a string concatenation:

```
var x = "10";
var y = 20;
var z = x + y;            // z will be 1020 (a string)
```

- A common mistake is to expect this result to be 30:

```
var x = 10;
var y = 20;
var z = "The result is: " + x + y;
```

- A common mistake is to expect this result to be 102030:

```
var x = 10;
var y = 20;
var z = "30";
var result = x + y + z;
```

❖ *The JavaScript compiler works from left to right. First 10 + 20 is added because x and y are both numbers. Then 30 + "30" is concatenated because z is a string.*

13.4 NUMERIC STRINGS

- JavaScript strings can have numeric content:

```
var x = 100;          // x is a number
var y = "100";        // y is a string
```

- JavaScript will try to convert strings to numbers in all numeric operations. This will work:

```
var x = "100";
var y = "10";
var z = x / y;        // z will be 10
```

- This will also work:

```
var x = "100";
var y = "10";
var z = x * y;        // z will be 1000
```

- And this will work:

```
var x = "100";
var y = "10";
var z = x - y;        // z will be 90
```

- But this will not work:

```
var x = "100";
var y = "10";
var z = x + y;        // z will not be 110 (It will be 10010)
```

In the last example JavaScript uses the + operator to concatenate the strings.

13.5 NAN - NOT A NUMBER

- NaN is a JavaScript reserved word indicating that a number is not a legal number.
- Trying to do arithmetic with a non-numeric string will result in NaN (Not a Number):

```
var x = 100 / "Apple";  // x will be NaN (Not a Number)
```

- However, if the string contains a numeric value, the result will be a number:

```
var x = 100 / "10";     // x will be 10
```

- You can use the global JavaScript function isNaN() to find out if a value is a number:

```
var x = 100 / "Apple";
isNaN(x);               // returns true because x is Not a Number
```

- Watch out for NaN. If you use NaN in a mathematical operation, the result will also be NaN:

```
var x = NaN;
var y = 5;
var z = x + y;          // z will be NaN
```

- Or the result might be a concatenation:

```
var x = NaN;
var y = "5";
var z = x + y;          // z will be NaN5
```

❖ *NaN is a number: typeof NaN returns number.*

```
typeof NaN;              // returns "number"
```

- Any mathematic operation you perform without both operands being numbers (or values that can be interpreted as regular numbers) will result in producing "not a number" or NaN.
- **The type of not-a-number is 'number'**! Hooray for confusing names and semantics!

```
var a = 2 / "foo";          // NaN

typeof a === "number";      // true

//Beware
a == NaN;     // false
a === NaN;     // false
isNaN( a ); // true
//buuut
var b = "foo";
isNaN( b ); // true -- ouch!
//ES6 to the rescue
//finally
Number.isNaN(..) // typeof n === "number" && window.isNaN( n )
```

13.6 INFINITY

Infinity (or -Infinity) is the value JavaScript will return if you calculate a number outside the largest possible number.

```
var myNumber = 2;
while (myNumber != Infinity) {          // Execute until Infinity
    myNumber = myNumber * myNumber;
}
```

Division by 0 (zero) also generates Infinity:

```
var x =  2 / 0;          // x will be Infinity
var y = -2 / 0;          // y will be -Infinity
```

❖ *Infinity is a number: **typeof** Infinity returns number.*

```
typeof Infinity;        // returns "number"
```

```
var a = 1 / 0;     // Infinity
var b = Infinity; // Infinity
var c = -1 / 0;     // -Infinity
var d = -Infinity; // -Infinity
```

13.7 HEXADECIMAL

- JavaScript interprets numeric constants as hexadecimal if they are preceded by 0x.

```
var x = 0xFF;            // x will be 255
```

- *Never write a number with a leading zero (like 07).*
- *Some JavaScript versions interpret numbers as octal if they are written with a leading zero.*

- *By default, JavaScript displays numbers as base 10 decimals.*
- *But you can use the **toString()** method to output numbers from base 2 to base 36.*
- *Hexadecimal is base 16. Decimal is base 10. Octal is base 8. Binary is base 2.*

```
var myNumber = 32;
myNumber.toString(10);  // returns 32
myNumber.toString(32);  // returns 10
myNumber.toString(16);  // returns 20
myNumber.toString(8);   // returns 40
myNumber.toString(2);   // returns 100000
```

13.8 NUMBERS CAN BE OBJECTS

- Normally JavaScript numbers are primitive values created from literals:

```
var x = 123;
```

- But numbers can also be defined as objects with the keyword new:

```
var y = new Number(123);
```

Example

```
var x = 123;                // typeof x returns number
var y = new Number(123);    // typeof y returns object
```

- *Do not create Number objects. It slows down execution speed.*
- *The **new** keyword complicates the code. This can produce some unexpected results:*

- When using the == operator, equal numbers are equal:

```
var x = 500;
var y = new Number(500);
// (x == y) is true because x and y have equal values
```

- When using the === operator, equal numbers are not equal, because the === operator expects equality in both type and value.

```
var x = 500;
var y = new Number(500);
// (x === y) is false because x and y have different types
```

- Or even worse. Objects cannot be compared:

```
var x = new Number(500);
var y = new Number(500);
// (x == y) is false because objects cannot be compared
```

- *Note the difference between (x==y) and (x===y).*
- *Comparing two JavaScript objects will always return false.*

13.9 == OR === DILEMMA

- **If either value** (aka side) in a comparison could be the **true** or **false** value, **avoid == and use ===**.
- **If either value** in a comparison could be of these specific values (**0**, **""**, or **[]** empty array), **avoid == and use ===**.
- **In all other cases**, you're **safe to use ==**. Not only is it safe, but in many cases it simplifies your code in a way that improves readability.

14 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.

14.1 THE IF STATEMENT

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

```
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.*

Example

Make a "Good day" greeting if the hour is less than 18:00

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

```
Good day
```

14.1.1 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
}
```

Example

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

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

```
Good day
```

14.2 THE ELSE IF STATEMENT

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

```
if (condition01) {
    block of code to be executed if condition01 is true
} else if (condition02) {
    block of code to be executed if the condition01 is false and condition02 is true
} else {
    block of code to be executed if the condition01 is false and condition02 is false
}
```

Example

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":

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

```
Good day
```

14.2.1 The Switch Statement

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

```
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";
}
```

14·2·2 The break Keyword

When JavaScript reaches a **break** keyword, it breaks out of the switch block. This will stop the execution of more code and case testing inside the block. When a match is found, and the job is done, it's time for a break. There is no need for more testing.

❖ *A break can save a lot of execution time because it "ignores" the execution of all the rest of the code in the switch block.*

It is not necessary to break the last case in a switch block. The block breaks (ends) there anyway.

14·2·3 The default Keyword

- The **`default`** keyword specifies the code to run if there is no case match:

Example

The `getDay()` method returns the weekday as a number between 0 and 6. If today is neither Saturday (6) nor Sunday (0), write a default message:

```
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";
}
```

Looking forward to the Weekend

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

Example

```
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.*

14·2·4 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:

Example

```
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";
}
```

14·2·5 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**.

14·2·6 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.