Learning JavaScript

- Developed by Brendan Eich as LiveScript.

JavaScript Where To

- In HTML, JavaScript code is inserted between **<script></script>** tags:

```
<!DOCTYPE html>
<html>
    <body>
        <script src=""></script>
    </body>
</html>
```

- Scripts can be placed in the **body** or in the **head**.

JavaScript Output

- JavaScript can "display" data in different ways:
    1. Writing into an HTML element: **innerHTML (needs an ID)**
    2. Writing into the HTML output: **document.write() (does not need an ID)**
    3. Writing into an alert box: **alert()**
    4. Writing into the browser console: **console.log()**

JavaScript Variables

- There are 5 primitive types: Number, String, Boolean, Undefined, or Null.
    - Strings are immutable.
    - 0, -0, "", false, undefined, NaN are **always false.**
    - If we have Boolean("false"), it is true, since it is not an empty string.
- There are 3 wrapper objects: Number, String, and Boolean.
- Variables are declared using different keywords:
    1. nothing.
    2. var: global, re-declarable, can be updated.
    3. let: local, not re-declarable, can be updated.
    4. const: local, not re-declarable, cannot be updated.

This will not work:
let carName = "Volvo";
ler carName;

JavaScript Let

- Block scope:

```
{
  let x = 2;
}
// x can NOT be used here
```

- Must be declared before use.
- Cannot be redeclared in the same scope:

```
let x = "John Doe";

let x = 0;
// this cannot be done
```

JavaScript Const

- Must be assigned a value when they are declared:

```
const PI = 3.14159265359;
```

- Block scope.
- Cannot be redeclared:

```
{
const x = 2;   // Allowed
const x = 2;   // Not allowed
}
```

- Cannot be reassigned:

```
const PI = 3.141592653589793;
PI = 3.14;          // This will give an error
PI = PI + 10;   // This will also give an error
```

JavaScript Operators

ADDITION	COMPARISON	MULTIPLICATION
+ is addition if and only if both operands are numbers.	1. If one operand is a number, and the other can be converted to a number, < is a number comparison. 2. If one operand is number, and the other cannot be converted to a number, false all the time. 3. If two operands are string, < is a string comparison (check ASCII table).	1. If operands are numbers, or all can be converted to a number, * is a number multiplication. 2. If one operand is a number, and the other cannot be converted to a number, NaN all the time.
1+2 = 3 "1"+2="12" 1+"2"="12" "1"+"2"="12" 1+"bird"="1bird" 1+2+"birds"="3birds"	11 < 2: false "11" < 2: false 11 < "2": false "11" < "2": true (check ASCII table) 11 < "bird": false	11 * 2 = 22 "11" * 2 = 22 11 * "2bird" = NaN

JavaScript Logical Assignment Operators

&&

- Evaluates expressions from left to right and returns the first falsy value encountered or the value of the last expression if all are truthy. It stops evaluating as soon as it hits a falsy value.

```
console.log(true && false); // false, because the second operand is false
console.log(true && "hello"); // "hello", because both operands are truthy, returns the last value
console.log("hi" && 123); // 123, because both operands are truthy, returns the last value
console.log(0 && "nope"); // 0, because the first operand is falsy, returns the first falsy value
```

||

- Evaluates expressions from left to right and returns the first truthy value encountered or the value of the last expression if all are falsy. It stops evaluating as soon as it hits a truthy value.

```
console.log(false || true); // true, because the second operand is true
console.log(false || ""); // "", because all operands are falsy, returns the last value
console.log("hello" || 0); // "hello", because the first operand is truthy, stops and returns the first truthy value console.log(null || "default"); // "default", because the first operand is falsy, continues and returns the next truthy value
```

!: logical NOT

```
console.log(!true); // false
console.log(!false); // true
console.log(!0); // true, because 0 is falsy
console.log(!""); // true, because an empty string is falsy
console.log(!"hello"); // false, because non-empty strings are truthy
console.log(!null); // true, because null is falsy
```

!!: DOUBLE NOT

```
console.log(!!true); // true
console.log(!!false); // false
console.log(!!0); // false, because 0 is falsy
console.log(!!""); // false, because an empty string is falsy
console.log(!!"hello"); // true, because non-empty strings are truthy
console.log(!!null); // false, because null is falsy
console.log(!!123); // true, because numbers other than 0 and -0 are truthy
```

JavaScript Equal and Not Equal

- == and != can do type coercion.
- === and !== cannot do type coercion.

```
"3" == 3: true
"3" === 3: false
```

```
var a = "123";
var b = "123";
if(a==b) {document.write('yes');}
      else {document.write('no');}
```

- What are the outputs?
  A. yes
  B. no
  C. error
  D. nothing

Answer:A

```
if(3!== "3") {document.write('yes');}
      else {document.write('no');}
```

- What are the outputs?
  A. yes
  B. no
  C. error
  D. nothing

Answer:A

```
var a = new String("123");
var b = new String("123");
if(a==b) {document.write('yes');}
      else {document.write('no');}
```

not equal because we are comparing the reference of each object

- What are the outputs?
  A. yes
  B. no
  C. error
  D. nothing

Answer:B

```
var a = String("123");
var b = new String("123");
if(a==b) {document.write('yes');}
      else {document.write('no');}
```

we are using cohersion because variables are not of the same type

- What are the outputs?
  A. yes
  B. no
  C. error
  D. nothing

Answer:A

JavaScript Switch

- Used to perform different actions based on different conditions.

```
switch (expression) {
      case value_1:
         // value_1 statements
      case value_2:
         // value_2 statements
      …
      [default:
         // default statements]
  }
```

JavaScript Control Statements

while: creates a loop that is executed while a condition is true

```
while (condition) {
  code block to be executed
}
```

for: defines a code block that is executed as long as a condition is true

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

do…while: used when you want to run a code block at least one time

```
do {
  code block to be executed
}
while (condition);
```