A Quick Tour of JS

LECTURE 1.4: LANGUAGE BASICS

Not this Week

- Arrays
- Modules
- Inheritance
- Functional JS
- Modules
- (Objects are sort of covered this time).

Overall Structure and Syntax

- Syntax and look and feel are much like C or C# or Java
- Beware of scope issues in old code
- Expressions are essentially the same
- Comparison operators differ (soon)
- Grouping of statements is the same

Overall Structure and Syntax

- Block conditionals have the usual C-like syntax
- Looping constructs are very similar
- Style: *always* group statements using braces
- Even (especially) when you don't think you need to.

Overall Structure and Syntax (Continued)

- Zero-based indexing, square bracket syntax
- Immutable strings (see later)
- Functions rather than methods – methods attached to objects as function values as we shall see.
- Functions crucial in traditional scope rules
- Optional arguments are covered in Chapter 3:
  - http://eloquentjavascript.net/03_functions.html

Operators and Assignment Statements

- All the usual stuff you get in a modern language
  - https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference
  - https://www.w3schools.com/jsref/jsref_operators.asp
- Mostly common across languages, but some variants
  - C-like assignment statements, increments and decrements
- Standard comparison operators
  - `>, >=, <, <=, ==, !=`
  - And some that aren't – see strict equality later
- Usual bitwise operations
  - Not treated here – look these up as you need them
- `typeof` operator (next time)

Arithmetic Operators

```
let a = 3 + 4; //addition 7
console.log("a: " + a);
let b = 7 - 4; //subtraction 3
console.log("b: " + b);

let c = b++;   //post-increment 3
console.log("c: " + c);
let d = c--;   //post-decrement 3
console.log("d: " + d);

let e = ++b;   //pre-increment 5
console.log("e: " + e);
let f = --e;   //pre-decrement 4
console.log("f: " + f);
```

```
node v10.15.2 linux/amd64
a: 7
b: 3


c: 3
d: 3




e: 5
f: 4
```

https://www.w3schools.com/jsref/jsref_operators.asp

```
// Initial values
let a = 10, b = 5;

a+=b;    // a = a + b
console.log("a+=b: " + a);

// Restored
a-=b;    // a = a - b
console.log("a-=b: " + a);

a*=b;    // a = a * b
console.log("a*=b: " + a);

// Restored
a/=b;    // a = a * b
console.log("a/=b: " + a);

a%=b;    // a = a % b
console.log("a%=b: " + a);
```

```
node v10.15.2 linux/amd64


a+=b: 15


a-=b: 10


a*=b: 50


a/=b: 10


a%=b: 0
```

Similarly for strings…

```
// Initial values
let hello = "Hello World";
let again = "Hello Cruel World";

let text = hello + ", " + again;
console.log(text);

hello += ", " + again;
console.log(hello);
```

```
node v10.15.2 linux/amd64
Hello World, Hello Cruel World
Hello World, Hello Cruel World
```

But see later…

There are other things we need to know about strings…

Equality Comparison

```
// Initial values
let x = 10;

//Loose equality
x == 5;       // false
x == "5";     // false
x == "10";    //true

//Loose or inequality
x != 5;       // true
x != "5";     // true
x != "10";    //false
```

```
// Initial values
let x = 10;


//Strict equality
x === 5;       // false
x === "10";    // false
x === 10;      //true

//Strict inequality
x !== 5;       // true
x !== "10";    // true
x !== 10;      //false
```

See also: https://medium.freecodecamp.org/js-type-coercion-explained-27ba3d9a2839

Pitfall: Equality Comparisons in JS

- Equality comparison operators cause many bugs in JS
- Here, follow the advice of Douglas Crockford:

==

JavaScript has two sets of equality operators: === and !==, and their evil twins == and !=. The good ones work the way you would expect. If the two operands are of the same type and have the same value, then === produces `true` and !== produces `false`. The evil twins do the right thing when the operands are of the same type, but if they are of different types, they attempt to coerce the values. The rules by which they do that are complicated and unmemorable. These are some of the interesting cases:

```
'' == '0'           // false
0 == ''             // true
0 == '0'            // true

false == 'false'    // false
false == '0'        // true

false == undefined  // false
false == null       // false
null == undefined   // true

' \t\r\n ' == 0     // true
```

The lack of transitivity is alarming. My advice is to never use the evil twins. Instead, always use === and !==. All of the comparisons just shown produce `false` with the === operator.

http://archive.oreilly.com/pub/a/javascript/excerpts/javascript-good-parts/bad-parts.html

Logical Operators

```
// Initial values
let x = 10; y = 20;

//&& - Logical AND
(x < y) && (y == 20);     // true
(x < y) && (y > 20);      // false
(x >= 10) && !(y == 200); //true



//|| - Logical OR
(x < y) || (y == 20);      // true
(x < y) || (y > 20);       // true
(x < 10) || !(y == 200);   //true
```

Selection and Iteration

- Control structures similar to other C-like languages
- Block if statements, switch statements, ternary operator
- For loops, while loops, do-while loops
- We show some simple examples

Block if-else statements

```
let myCredits = 156;

const graduate  = 288;
const second = 192;
const first = 96;

if (myCredits >= graduate) {
    console.log("Bachelor of IT Graduate");
} else if (myCredits >= second) {
    console.log("Third Year Student");
} else if (myCredits >= first) {
    console.log("Second Year Student");
} else {
    console.log("First Year Student");
}
```

Switch Statements

```
let intDay = 3;
let strWeekDay = "";
switch (intDay) {
    case 0 :
    case 6 :
        strWeekDay = "Weekend";
        break;
    case 1 :
    case 2 :
    case 3 :
    case 4 :
    case 5 :
        strWeekDay = "Week Day";
        break;
    default :
        strWeekDay = "Invalid day";
}
```

For Loop

```
// Counter controlled loop
for (let count=1; count<=100; count++) {
    console.log(count);
}
```

```
node v10.15.2 linux/amd64
1
2
3
4
5
…
96
97
98
99
100
```

Equivalent While Loops

```
// Pre-tested Loop
let count = 1;
while (count <= 100) {
    console.log(count);
    count++;
}



// Post-tested Loop
let count = 0;
do {
    count++;
    console.log(count);
} while (count < 100);
```