Introduction

Introduction to Programming

Statement

```
a = b + 3;
```

- Variables - `a` , `b`
- Literal values - `3`
- Operators - `+`
- (Semicolon - `;`)

Expressions

The statement `a = b + 3;` consists of four expressions

- Literal value expression - `2`
- Variable expression - `b`
- Arithmetic expression - `b + 3`
- Assignment expression - `a = b + 3`

Expression Statement

```
c = 8;
```

Call Expression

```
console.log(c);
```

Running a Program

Interpreted versus Compiled

Interpreter

Compiler

The JavaScript engine compiles the program on the fly and then immediately runs the compiled code.

Declaring Variables

```
var a
```

Or with a assignment

```
var b = 6
```

Operators

Assignment

- `=`

As in

```
c = 8
```

Mathematical Operators

- `+` (addition)
- `-` (subtraction)
- `*` (multiplication)
- `/` (division)

As in

```
3 + 4
```

Compound Assignment Operators

- `+=`
- `-=`
- `*=`
- `/=`

As in

```
c += 2
```

Increment/Decrement Operators

- `++`
- `--`

As in

```
i++
```

Object Property Access Operator

- `.`

As in

```
foo.bar
```

Equality Operators

- `==` (loose-equals)
- `===` (strict-equals)
- `!=` (loose-not-equals)
- `!==` (strict-not-equals)

Comparison Operators

- `<` (less than)
- `>` (greater than)
- `<=` (less than or loose-equals)
- `>=` (greater than or loose-equals)

Logical Operators

- `&&` (and)
- `||` (or)

Comments

```
// Line comment

a = 4; // Rest of line comment
```

```
/*
    Block comment
*/

a = 4 * 0.5 /* inline comment */ + 3;
```

Variables

Static Typing versus Dynamic Typing

Static Typing

A variable is declared to hold values only of a given type.

Dynamic Typing

Variables can hold values of different types.

Blocks

```
{
    a = 4;
    b = 6;
    c = a / b;
}
```

Conditionals

if-Statements

```
if (index > 5) {
    // true stuff
}

if (index === 1) {
    // true stuff
} else {
    // false stuff
}
```

Loops

while-Statements

```
while (index < 10) {
    // zero or more repeating stuff
}
```

do-Statements

```
do {
    // one ore more repeating stuff
} while (index < 10)
```

for-Statements

```
for (var i = 0; i < 10; i++) {
    // 10 times repeating stuff
}
```

Functions

```
function foo() {
    // stuff
}
```

Parameters

```
function foo(a, b) {
    // stuff with a and b
}
```

Return Statements

```
function foo(a, b) {
    return a + b;
}
```

Scope

Lexical Scoping versus Dynamic Scoping

Lexical Scope

```
function bar() {
    console.log(i); // Error!
}

function foo() {
    var i = 5;
    bar();
}

foo();
```

Nested Scope

```
function foo() {
    var i = 5;

    function bar() {
        console.log(i);
    }

    bar();
}

foo();
```

Introduction to JavaScript

Types and Values

Build-In Types

- `string`
- `number`
- `boolean`
- `null`
- `undefined`
- `object`
- `symbol` (ES6)

String

Literals

```
var a = 'single';
var b = "double";
```

typeof

```
var a = 'foo';
typeof a; // 'string'
```

number

Literals

```
var a = 1;
var b = 2.3;
```

typeof

```
var a = 2;
typeof a; // 'number'
```

boolean

Literals

```
var a = true;
var b = false;
```

typeof

```
var a = false;
typeof a; // 'boolean'
```

null

Literals

```
var a = null;
```

typeof

```
var a = null;
typeof a; // 'object' - Bug!
```

undefined

Literals

```
var a = undefined;
```

typeof

```
var a;
typeof a; // 'undefined'
```

object

Literals

```
var a = {
    b: 5,
    c: 'foo'
};
```

typeof

```
var a;
typeof a; // 'object'
```

Property Accessors

```
var b = a.b;
var c = a['c'];
```

Sub-Types of Object

- Array
- Function

Array

Literals

```
var a = [1, 2, 4, 8];
var b = [null, false, 5, 'foo', undefined];
var c = [];
var d = [[1, 3], [2, 4]];
```

typeof

```
var a = [];
typeof a; // 'object'
```

Accessors

```
var i = a[2];
a[4] = 5;
```

Build-In Properties

```
var a = [1, 2, 4];
a.length; // 3
```

Function

Literals

```
function foo(a, b, c) {
    // stuff
}

var bar = function(a, b, c) {
    // stuff
}

var baz = function foo(a, b, c) {
    // stuff
};
```

typeof

```
function foo() {}
typeof foo; // 'object'
```

Build-In Properties

```
function foo(a, b, c) {
    return a + b + c;
}
a.length; // 3
```

Variables

Naming

An identifier must start with a-z, A-Z, $, or _. It can then contain any of those characters plus the numerals 0-9.

There is a list of reserved words that cannot be used as variable names.

Function Scope

JavaScript uses a kind of Lexical Scope called Function Scope where the function is the scope barriers.

Hoisting

Every variable declaration is hoisted to the top of the scope.

Conditionals

More on if-else-Statements

```
if (i < 5) i = 0;

if (i < 5) i = 0 else i = 10;

if (i < 5) {
    // stuff
} else if (i < 10) {
    // stuff
}
```

switch-Statements

```
switch (i) {
    case 1:
        // stuff
        break;
    case 2:
        // stuff
        break;
    case 3:
        // stuff
        break;
    default:
        // stuff
}
```

With fall through

```
switch (i) {
    case 1:
        // stuff
    case 2:
        // stuff
    case 3:
        // stuff
    default:
        // stuff
}
```

First-Order Functions

In computer science, a programming language is said to have first-class functions if it treats functions as first-class citizens. Specifically, this means the language supports passing functions as arguments to other functions, returning them as the values from other functions, and assigning them to variables or storing them in data structures.

Passing functions as arguments to other functions

```
function foo(bar) {
    bar();
}
```

The build-in setTimeout function

```
setTimeout(function() {
    // stuff
}, 1000);
```

Returning them as the values from other functions

```
function foo() {
    return function() {
        // stuff
    };
}
```

Assigning them to variables

```
var foo = function() {
    // stuff
};
```

Storing them in data structures

```
var foo = {
    bar: function() {
        // stuff
    }
};
```

Immediately Invoked Function Expressions (IIFEs)

```
(function() {
    //stuff
})();
```

Closure

```
function foo() {
    var a = 0;
    return function(b) {
        return a += b;
    };
}

var bar = foo();
bar(2); // => 2
bar(3); // => 5

var baz = foo();
baz(2); // => 2
```

this