JavaScript Basics

Both familiar and alien simultaneously.

1. **Data Types**
2. **Operators**
3. **Miscellany**

Let’s keep it simple.

JavaScript Data Types

String, Number, Boolean, Array, Object, Null, Undefined

Data Types

```
var x;
```

Dynamic Types

```
var x;                          // x is: undefined
```

Dynamic Types

```
var x;                    // x is: undefined
x = 5;                    // x is set to a Number: 5
```

Dynamic Types

```
var x;                  // x is: undefined
x = 5;                  // x is set to a Number: 5
x = "Bond";             // x is changed to a String: "Bond"
```

Strings

```
var car = "duck-billed platypus";
var car2 = 'duck-billed platypus';
```

Numbers

```
var x = 42;          // Written without decimals
var y = 13.37;       // Written with decimals
var z = 10e3;        // Exponential notation
```

Booleans

```
var x = true;
var y = false;
```

Empty Values

```
var x = null;          // Explicit empty
var y = undefined;     // Implicit empty
```

Empty Values: null

- Useful for intentially communicating an empty value (i.e., “this value is empty intentionally!”

```
var x = null;
x === null;              // Checking for null, true
x.foo                    // throws “TypeError: Cannot read
                            property 'foo' of null”
```

Empty Values: undefined

```
var x;                    // Default value is undefined
x === undefined;          // Checking for undefined, true
x.foo                     // throws “TypeError: Cannot read
                             property 'foo' of undefined”
```

Undeclared Variables

```
asdf; // throws "ReferenceError: asdf is  not defined"
typeof asdf === "undefined" // true
```

Array

```
var platy = ["bill", "tail", "fur"];
```

```
["bill", "tail", "fur"]
```

Array (wrong way)

```
var platy = new Array();
platy[0] = "bill";
platy[1] = "tail";
platy[2] = "fur";

["bill", "tail", "fur"]
```

Array (wrong way)

```
var frukt = new Array("bill", "tail", "fur");
```

```
["bill", "tail", "fur"]
```

Array Methods

```
var frukt = ["bill", "tail", "fur"]


frukt.pop();              // ["bill", "tail"]
frukt.push("fur");        // ["bill", "tail", "fur"]
frukt.shift();            // ["tail", "fur"]
frukt.unshift("fur");     // ["fur", "tail", "fur"]
```

Objects

- **Everything** is an Object

Objects

- **Everything** is an Object
- Booleans can be objects or primitive data treated as objects
- Numbers can be objects or primitive data treated as objects
- Strings are also objects or primitive data treated as objects
- Dates, Maths, Regular expressions, Arrays and functions are always objects

Object Literals

An object is just a special kind of data, with **properties** & **methods**.

```
var person = {
    firstName: "George",
    lastName: "Orwell",
    id: 5
};
```

Object Literals

An object is just a special kind of data, with **properties** & **methods**.

```
var person = {
    firstName: "George",
    lastName: "Orwell",
    id: 5
};
person.id; // 5
```

Object Literals

An object is just a special kind of data, with **properties** & **methods**.

```
var person = {
    firstName: "George",
    lastName: "Orwell",
    address: {
        street: "El Camino Real",
        number: "27F"
    }
};
person.address.street; // "El Camino Real"
```

Object (wrong way)

```
var randomPerson = new Object();

randomPerson.firstName = "George";

randomPerson.lastName = "Orwell";
```

Functions

A block of code that can be executed, “called” or “invoked” later.

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

var add = function(a, b) {
    return a + b;
}

add(1,2) // 3
```

Function Constructor

```
function Person(firstName, lastName) {
    this.firstName = firstName;
    this.lastName = lastName;
}

var randomPerson = new Person("George", "Orwell");
```

JavaScript Operators

Conditionals, type coercion, equality, typeof

Boolean Expressions

```
if (a == 2) {//if this is true
    //do this...
}
```

Type Coercion

- JavaScript will attempt to convert to a compatible type when necessary

```
if ('false') { console.log("true"); }
// Convert from String to Boolean, coerce non-empty string to true
```

Truthy Values

- `true` // Boolean true
- `'false'` // Non-empty Strings
- `5` // Non-zero numbers
- `[]` // Any Array
- `{}` // Any Object
- `function() {}` // Any Function

Falsey Values

- `false` // Boolean false
- `''` // Empty Strings
- `0` // Non-zero numbers
- `null` // Empty references
- `undefined` // Empty references

More Type Coercion

```
1 == "1"

true == "1"

false == "0"
```

More Type Coercion

```
1 == "1"
    true
true == "1"

false == "0"
```

More Type Coercion

```
1 == "1"
    true
true == "1"
    true
false == "0"
```

More Type Coercion

```
1 == "1"
    true
true == "1"
    true
false == "0"
    true
```

More Type Coercion

```
1 == "1"
    true
true == "1"
    true
false == "0"
    true
false == "0" == 1 == true == [] == ""
```

More Type Coercion

```
1 == "1"
    true
true == "1"
    true
false == "0"
    true
false == "0" == 1 == true == [] == ""
    true
```

Avoid Type Coercion

- By using: ===, !==

===, !==

```
1 == true
   true
1 === true
   false
1 == "1"
   true
1 === "1"
   false
```

typeof

```
typeof 1 === "number"
    true
typeof "1" === "string"
    true
typeof undeclaredVariable === "undefined"
    true
typeof true === "boolean"
    true
typeof function(){} === "function"
    true
```

typeof: bad parts

```
typeof null === "object"
    true
typeof [] === "object"
    true
typeof {} === "object"
    true
```

JavaScript Miscellany

Scope, globals, “namespaces”, “methods”, IIFE

Scope & Global Variables

- C++/Java: anything inside curly brackets, {} , defines a scope

```
if (true)  {
        var scopeVariable = "Test";
}
scopeVariable = "Test2"; // variable not defined
```

Scope & Global Variables

- Javascript: only functions define a new scope (except in ES6)

```
if (true) {
    var scopeVariable = "Test";
}
scopeVariable; // value: "Test";
```

Scope & Global Variables

```
function scope1() {
    var member; //is a member of the scope defined by the function example

    //this function is also part of the scope of the function example
    var innerScope = function() {
       member= 12; // traverses scope and assigns member in scope1 to 12
    };
};
```

Scope & Global Variables

```
function scope1() {
    var member; //is a member of the scope defined by the function example

    //this function is also part of the scope of the function example
    var innerScope = function() {
      var member= 12; // defines member in this scope and do not traverse
    };
};
```

Scope & Global Variables

```
function scope1() {
    var member;    //is a member of the scope defined by the function example

    //this function is also part of the scope of the function example
    var bar = function() {
        member= 12; // traverses scope and assigns scope1member.foo to 12
    };
};

function scope2() {
    member = 15; // traverses scope and assigns global.member to 15
}
```

JavaScript Namespaces?

- Not built into JavaScript
- Problem?

Ad-hoc JavaScript Namespaces

```
var Peanuts = {};  // Object used as namespace
```

Ad-hoc JavaScript Namespaces

```
var Peanuts = Peanuts || {}; // in case it already
exists
```

Object Methods

```
var Peanuts = Peanuts || {};
Peanuts.Calculator = {
    add: function (a,b) {
        return a + b;
    },
    subtract: function () {
        return a - b;
    }
};
Peanuts.Calculator.add(1, 2); // 3
```

Immediately Invoked Function Expression (IIFE)

```
// Create a new closure/scope
(function () {
    // logic/code here
})();
```

IIFE: Why?

- Immediately executes code
- Avoids implicit globals
- Privacy: Avoids potential for variable name collision.

IIFE: How?

- JavaScript, parenthesis can't contain statements.
- When the parser encounters the function keyword, it knows to parse it as a function expression and not a function declaration.

IIFE: Cont'd

```
(function () {
    // logic/code here
})();
```

- The key thing about JavaScript expressions is that they return values.
- To invoke the function expression right away we just need to tackle a couple of parentheses on the end(like above).

IIFE: Arguments

```
(function (innerValue) {
    // logic/code here

})(outerValue);
```