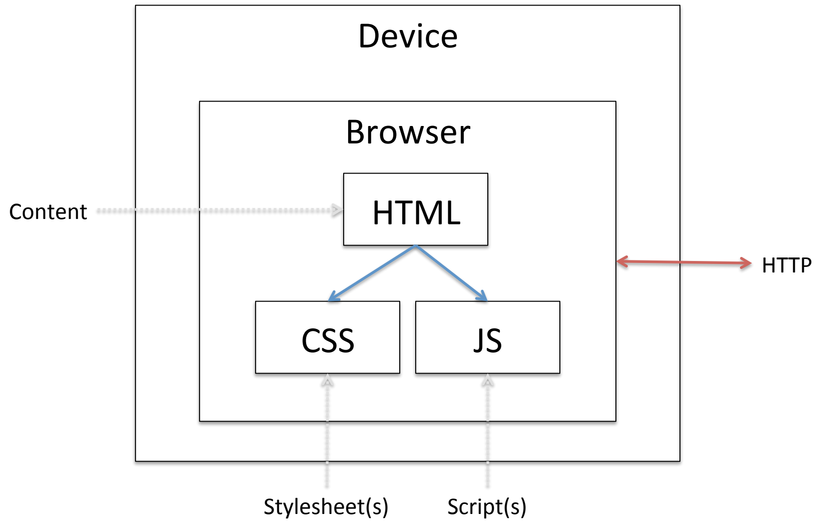
University of Glasgow

Client Side Scripting

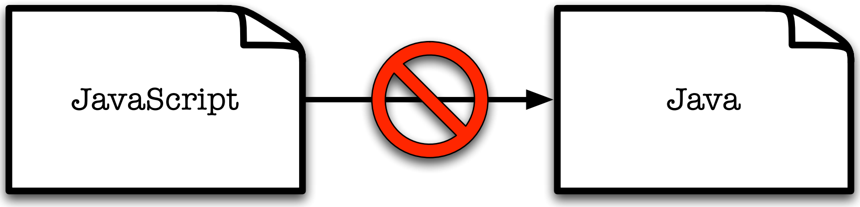
Web Application Development 2

Client-side Scripting



JAVASCRIPT

The Name



JavaScript

Java

- It is a completely different language
- Good/bad marketing idea
- Originally called “LiveScript”, but this was not confusing enough

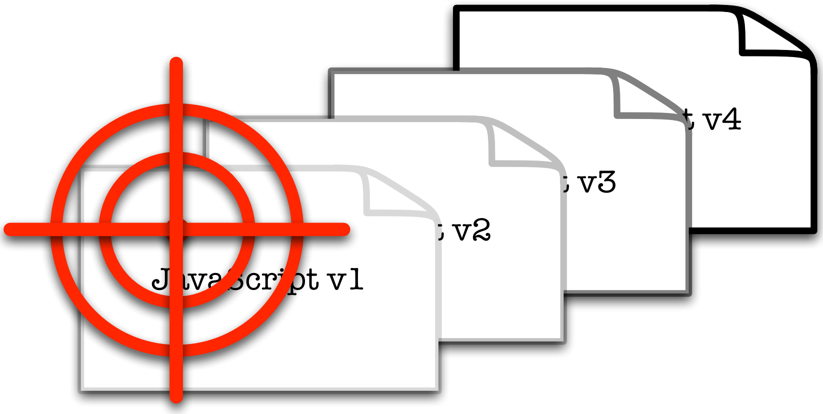
Typecasting

- Designed to run in Netscape Navigator
- Became standard in all browsers
- But useful for a wide range of programming tasks (*e.g. node.js*)

Procedural / Functional

- It looks like a procedural language, but is closer to functional
- Functions are first class
- Supports anonymous functions (*heavily used by jQuery*)

Moving Target



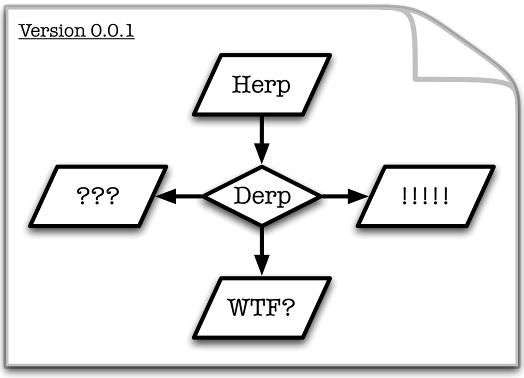
- Opinions formed on earlier versions
- Lacked object-orientation and exception handling
- There is a standard (ECMA)

The Standard

ecma INTERNATIONAL

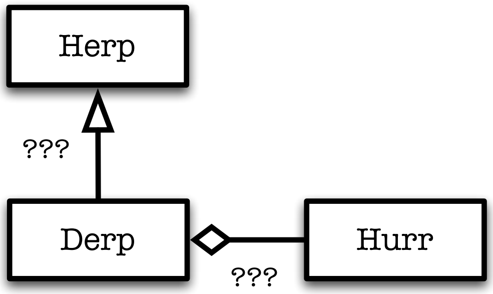
https://www.ecma-international.org/

Design Errors



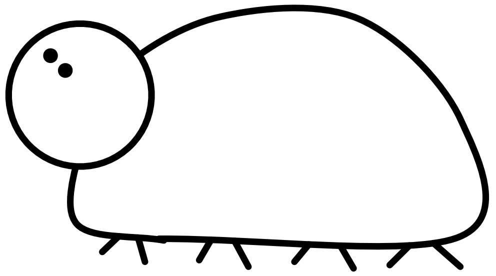
- No language is perfect (why there are so many)
- Small things annoy (semi-colon insertion, overloaded operators)
- Some problems can be avoided by using IDEs to check syntax, or JSLint (http://www.jslint.com/)

Object-Oriented



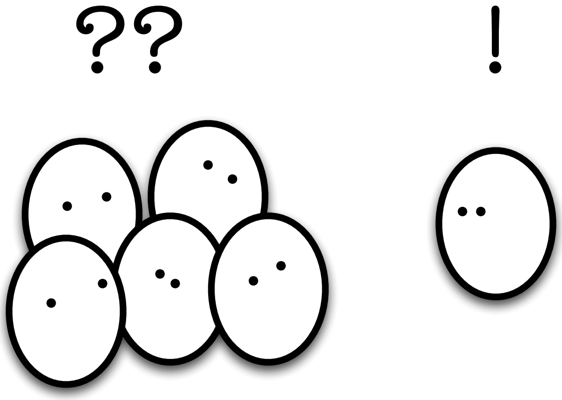
- Is it OO? Yes, has objects, which encapsulate data/methods
- Does it have classes? Where is inheritance?
- Different style of OOP

Lousy Implementations



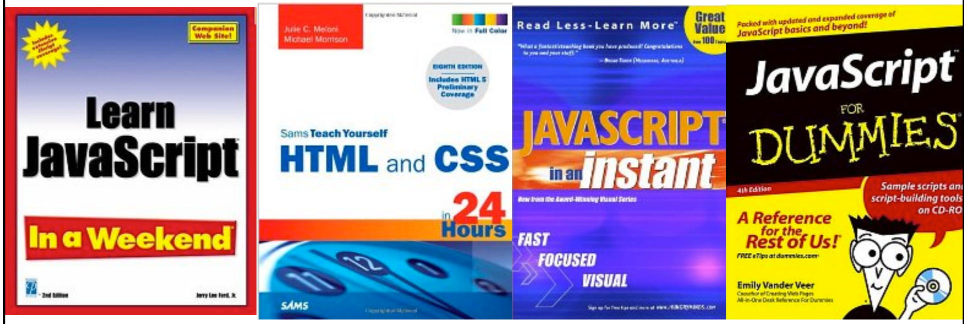
- The JavaScript engines of early browsers were buggy
- The browsers containing JavaScript engines were buggy
- JavaScript performance war has helped

Bunch of Amateurs



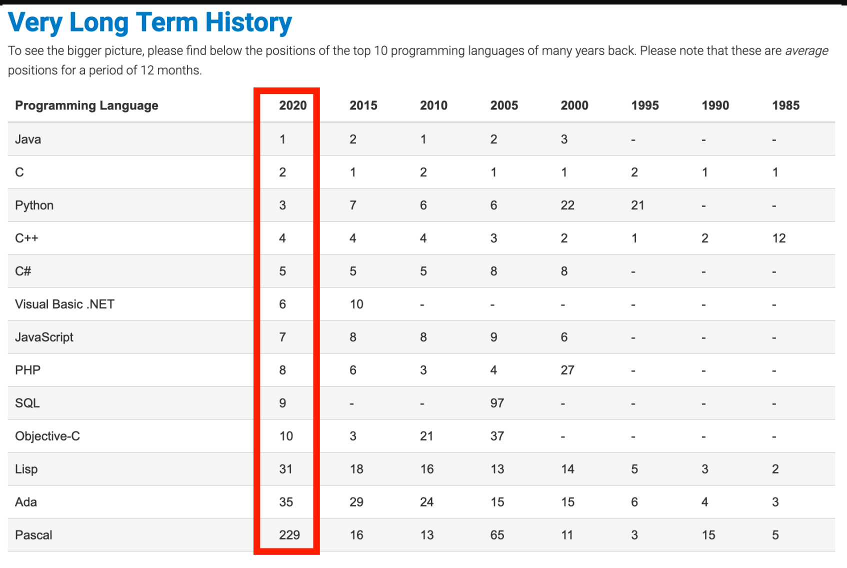
- Most people writing JavaScript are not programmers
- Lack of training, discipline, common sense
- Expressive language that is severely underutilised

Bad Books


Learn JavaScript In a Weekend
Sams Teach Yourself HTML and CSS in 24 Hours
JAVASCRIPT in an instant
FAST
FOCUSED
VISUAL
JavaScript FOR DUMMIES
A Reference for the Rest of Us!
Emily Vander Veer

THE MISUNDERSTOOD LANGUAGE

Language Popularity



Very Long Term History

To see the bigger picture, please find below the positions of the top 10 programming languages of many years back. Please note that these are *average* positions for a period of 12 months.

Programming Language	2020	2015	2010	2005	2000	1995	1990	1985
Java	1	2	1	2	3	-	-	-
C	2	1	2	1	1	2	1	1
Python	3	7	6	6	22	21	-	-
C++	4	4	4	3	2	1	2	12
C#	5	5	5	8	8	-	-	-
Visual Basic .NET	6	10	-	-	-	-	-	-
JavaScript	7	8	8	9	6	-	-	-
PHP	8	6	3	4	27	-	-	-
SQL	9	-	-	97	-	-	-	-
Objective-C	10	3	21	37	-	-	-	-
Lisp	31	18	16	13	14	5	3	2
Ada	35	29	24	15	15	6	4	3
Pascal	229	16	13	65	11	3	15	5

http://www.tiobe.com/tiobe-index

Core Features

- Syntactically similar to Java/C
  (if/else, while, for)
- Familiar primitive datatypes
  (numbers, strings, Booleans)
- Object-oriented
  (in its own way)

Core Features

- Interpreted Language
  (no compiling)

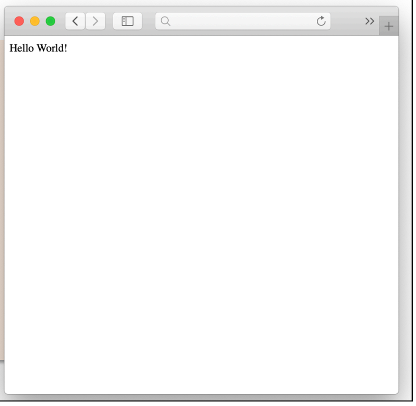
- Dynamic Typing
  (var x = 10; var y = “abc”;)

- Functions are first class
  (can also be anonymous and nested)

Inline JavaScript

- Scripts can be included inline with HTML code
- Good for experimentation
- Violates separation of concerns

```
<html>
  <head>
  </head>
  <body>
    <script type="text/javascript">
      document.writeln("Hello World!");
    </script>
  </body>
</html>
```



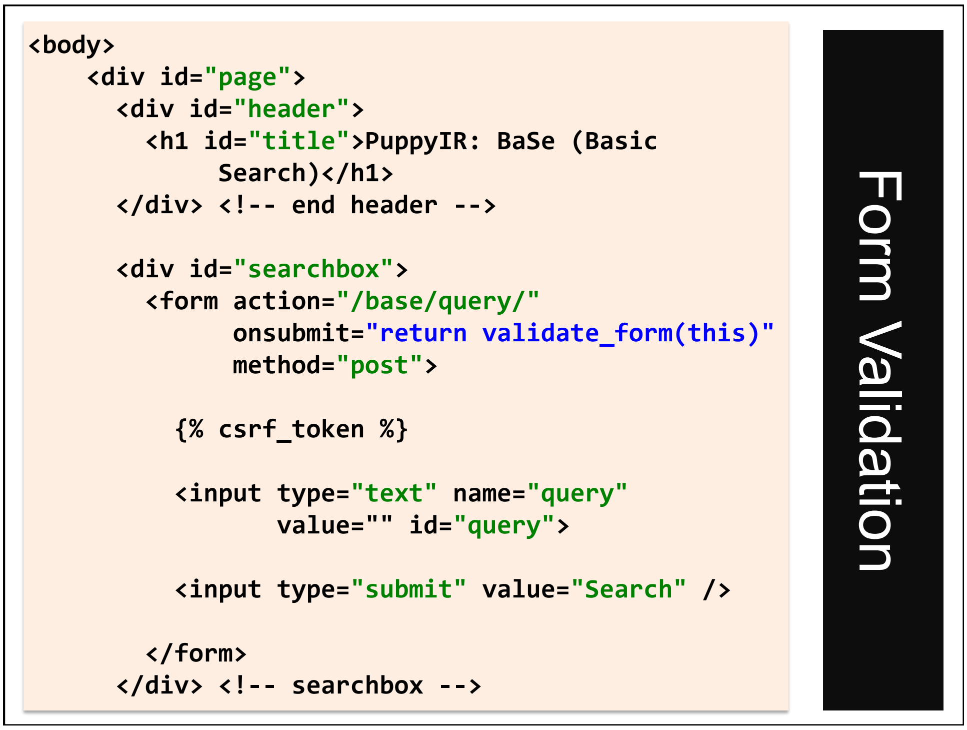
Hello World!

Embedding JavaScript

- Scripts can also be added in html head
  - Add JS to event handlers to e.g. call functions
- Fragile to maintain

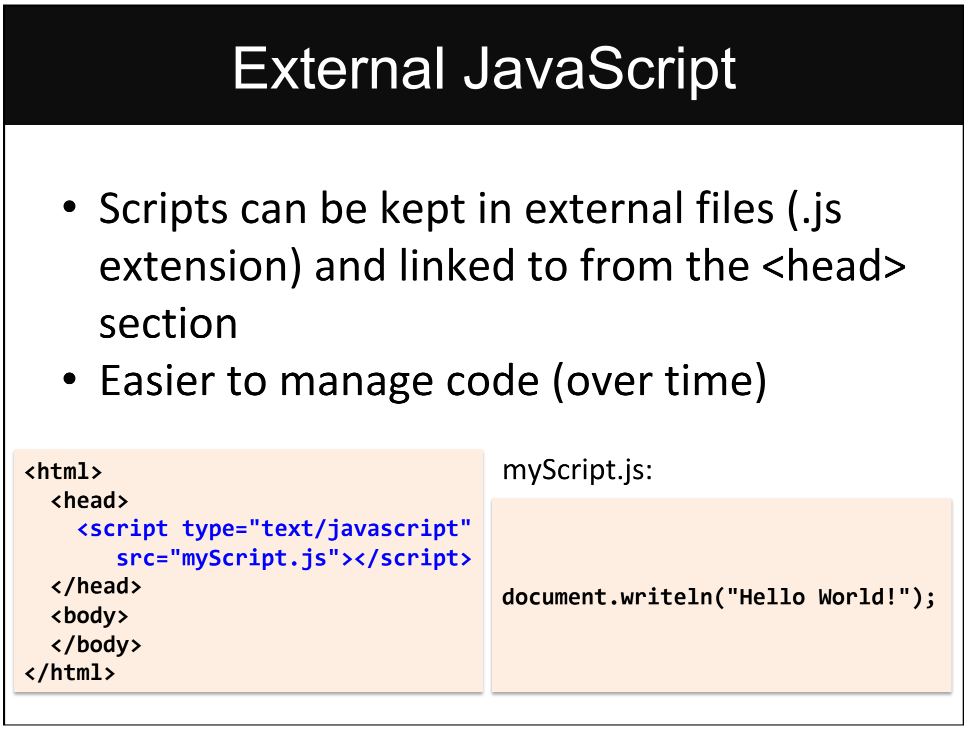
Form validation

```
<head>
   <script type="text/javascript">

      function validate_form(thisform) {
         with (thisform) {
            if (query.value==null ||
   query.value=="") {
               query.focus();
               return false;
            }
            else {
               return true;
            }
         }
      }

   </script>
</head>
```



Form Validation

```
<body>
    <div id="page">
      <div id="header">
        <h1 id="title">PuppyIR: BaSe (Basic
             Search)</h1>
      </div> <!-- end header -->

      <div id="searchbox">
        <form action="/base/query/"
              onsubmit="return validate_form(this)"
              method="post">

          {% csrf_token %}

          <input type="text" name="query"
                 value="" id="query">

          <input type="submit" value="Search" />

        </form>
      </div> <!-- searchbox -->
```



External JavaScript

- Scripts can be kept in external files (.js extension) and linked to from the <head> section
- Easier to manage code (over time)

```
<html>
  <head>
    <script type="text/javascript"
       src="myScript.js"></script>
  </head>
  <body>
  </body>
</html>
```

myScript.js:

```
document.writeln("Hello World!");
```

DOM Integration

- The intent behind JavaScript was to dynamically script/manipulate documents
- HTML documents are modelled using DOM
- DOM methods and properties can be accessed and altered using JavaScript

Finding Elements in the DOM

- Finding DOM elements to manipulate
- getElementsByTagName( )
- getElementById( )

```
// Find the number of tables in a document
var tables = document.getElementsByTagName("table");
alert("This document contains " + tables.length + " tables");

// Find a specific Table within a document and count its rows
var tableOfContents = document.getElementById("TOC");
var rows = tableOfContents.getElementsByTagName("tr");
var numrows = rows.length;
```

Modifying Elements in the DOM

- The real impact of JavaScript is changing the content of the DOM

```
// This function traverses the DOM tree and
// converts all Text node data to uppercase
function upcase(n) {
   if (n.nodeType == 3 /*Node.TEXT_NODE*/) {
      n.nodeValue = n.nodeValue.toUpperCase();
   } else {
      // If the node is not Text, loop through its children
      // and recursively call this function on each child.
      var kids = n.childNodes;
      for (var i = 0; i < kids.length; i++) {
         upcase(kids[i]);
      }
   }
}
```

Modifying Elements in the DOM

- nodeType returns the type of the node
  - 1 for an element node
  - 2 for an attribute node
  - 3 for a text node
  - 8 for a comment node
  - 9 for a document node
- Include a reference to script containing upcase in html head
- Call the function by putting the following at the **bottom** of the document body:

```
<script type="text/javascript">
    upcase(document.body)
</script>
```

Factorials (N!)

```
<html>
  <head>
    <title>Factorials</title>
  </head>
  <body>
    <h3>Table of Factorials</h3>
    <p id="demo"></p>
    <script type="text/javascript" src="factorial.js">
    </script>
  </body>
</html>
```

factorial.js

```
var fact = 1;
var text = "";
for (var i = 1; i < 10; i++) {
    fact *= i;
    text += (i + "! = " + fact + "<br />");
}
document.getElementById("demo").innerHTML = text;
```

Button Event

```
<html>
  <head>
    <title>Button Event</title>
  </head>
  <body>
    <h3>Button Event</h3>
    <button onClick="alert('You clicked the button');">
      Click Here
    </button>
  </body>
</html>
```

SYNTAX

The JavaScript Language

- A simple script
- Lexical structure
- Datatypes and values
- Variables
- Expressions and operators
- Statements
- Objects and arrays
- Functions
- Classes and constructors
- Pattern matching and regular expressions

Lexical Structure

- JavaScript is a case-sensitive language (keywords, identifiers, variables, functions etc must be consistent)
- Whitespace is ignored (spaces, tabs and newlines) but see below
- Semi-colons are optional – but it is good practice
  - JavaScript interpreters automatically add them – this is a very bad thing so it is better to be explicit!

```
return  ──────────→  return;
true;                true;
```

What happens is that **undefined** is returned instead of **true**

Lexical Structure

- **Comments** can be single // or multiline /* */ C-style
- **Literals** are data values that appear directly in the language: 12, 1.2, “hello”, true, false
- **Identifiers** are names for variables and functions
  - First character must be letter, underscore or dollar
  - Remaining characters can include above and numbers
- **Reserved word** set cannot be used as identifiers
  - Be careful, JavaScript has an unusually large set of reserved words that may become part of the language in the future

Datatypes

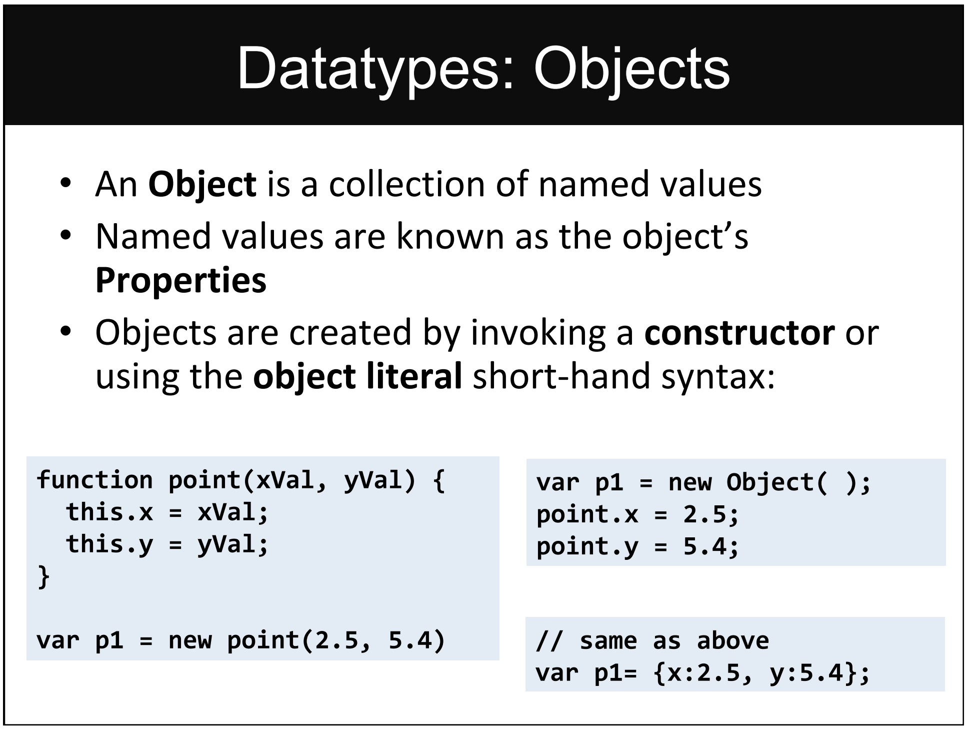
- Three primitive types:
  - **Number**: no distinction between integers (123) and decimal (3.14) and floating-point (6.02e23) values
  - **String**: sequence of unicode letters, digits and punctuation characters delimited by single or double quotes (“Hello!”)
  - **Boolean**: true or false
- Two trivial types:
  - **null**: an assignment value that can represent no value – null is (a placeholder for) an object
  - **undefined**: variable that has been declared but no value has been assigned to it, or an object property that does not exist

Datatypes: Functions

- A function is a piece of executable code that is defined once, but can be called multiple times
- In other languages, functions or methods are often just useful construct to gather related code
- In JavaScript, functions are first class objects in the language, and can be passed as datatypes
- No return type required in function signature

```
function square(x) {
  return x * x;
}

y = square(4);
```

```
var sq = function(x) {
  return x * x;
};

function applyOperator(op, x){
   return op(x)
}

y = applyOperator(sq,4);
```



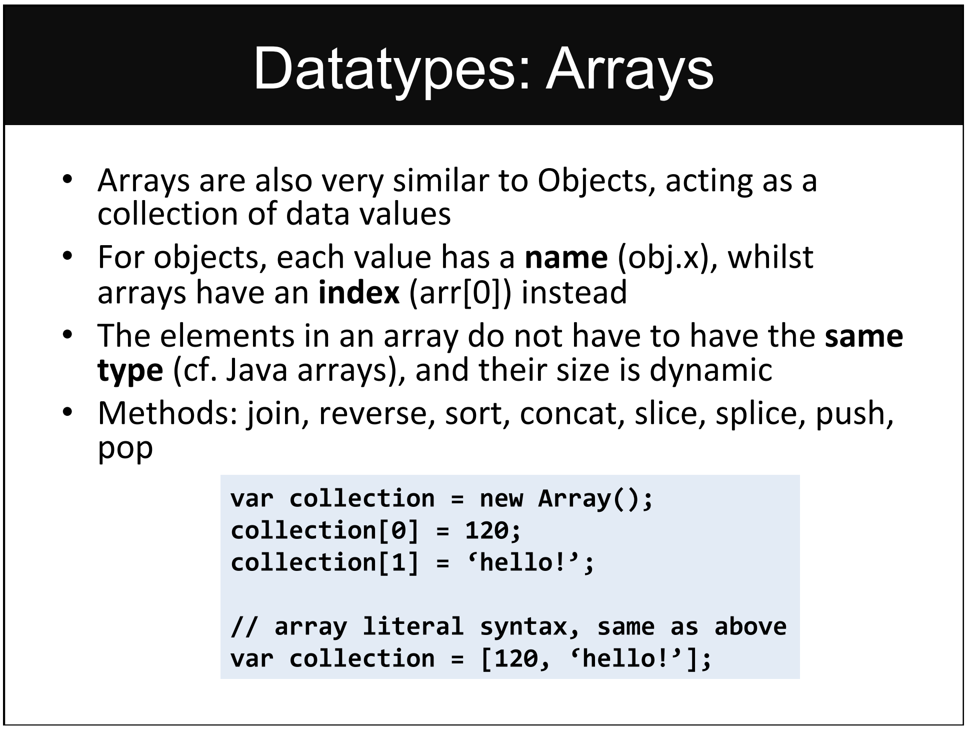
Datatypes: Objects

- An **Object** is a collection of named values
- Named values are known as the object’s **Properties**
- Objects are created by invoking a **constructor** or using the **object literal** short-hand syntax:

```
function point(xVal, yVal) {
  this.x = xVal;
  this.y = yVal;
}

var p1 = new point(2.5, 5.4)
```

```
var p1 = new Object( );
point.x = 2.5;
point.y = 5.4;
```

```
// same as above
var p1= {x:2.5, y:5.4};
```



Datatypes: Arrays

- Arrays are also very similar to Objects, acting as a collection of data values
- For objects, each value has a **name** (obj.x), whilst arrays have an **index** (arr[0]) instead
- The elements in an array do not have to have the **same type** (cf. Java arrays), and their size is dynamic
- Methods: join, reverse, sort, concat, slice, splice, push, pop

```
var collection = new Array();
collection[0] = 120;
collection[1] = ‘hello!’;

// array literal syntax, same as above
var collection = [120, ‘hello!’];
```

Variables

```
var i = 10;
i = “hello!”;
```

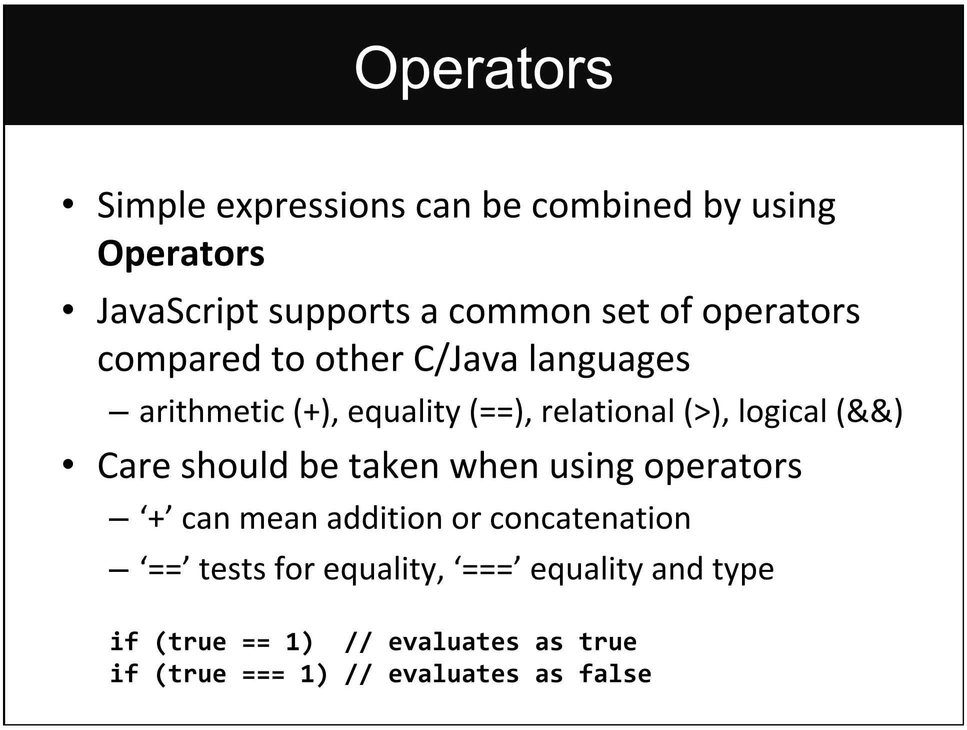
- An **identifier** associated with a **value**
- Used to store and manipulate values in a program
- All variables are **untyped** (weak or loose typing)
- Variables are **declared** using the **var** keyword
  - if this is missing, the variable is global – not recommended
- Scope of variables depends on where they are declared
  - global variables can be seen everywhere
  - variables declared in a function are only visible locally
  - omitting **var** in functions will use matching global variables
  - there is no **block** scope like C/Java languages (e.g. in **for** or **if/else** blocks

```
var i=10;
var j=10;
function scope() {
   i="hello";
   var j="hello";
}

scope();
// i is "hello”
// j is 10
```

Expressions

- An **expression** is a phrase of code that can be evaluated to produce a **value**

```
1.5                          // a numeric literal
“hello!”                     // a string literal
True                         // a boolean literal
/java/                       // a regular
                             // expression literal
{x:1.2, y:2}                 // an object literal
[1, 2, 3, 4, 5]              // an array literal
function(x) {return x*x;}    // function literal
sum                          // the variable sum
```



Operators

- Simple expressions can be combined by using **Operators**
- JavaScript supports a common set of operators compared to other C/Java languages
  - arithmetic (+), equality (==), relational (>), logical (&&)
- Care should be taken when using operators
  - '+' can mean addition or concatenation
  - '==' tests for equality, '===' equality and type

```
if (true == 1)  // evaluates as true
if (true === 1) // evaluates as false
```