JAVASCRIPT

HISTORY

- Originally in browsers
- Not Java
- It has some good parts
- Standardised by Ecma (once ECMA) as EcmaScript
- Current version is ES8 (2018)
- Most recent widely-supported version is ES6 (2015)
- Support varies

CLIENT- AND SERVER- SIDE

- Recently JS is also used server-side: nodejs
- Good JS engines in mobile browsers
- JS often used for cross-platform App dev Cordova
- Also for desktop applications with electron e.g. atom
- Interpreted, not compiled: errors only happen at run-time
- `console.log` is your friend

SYNTAX

- Mostly insensitive to white space (not python)
- Case sensitive (not php)
- Block structured, with braces (like Java)
- Semicolons at the end of lines can be inferred
- "use strict"; good practice
- Use `require` for modules in nodejs (many ways in browser)

VARIABLES AND SCOPE

- Variables must be declared (in strict mode)
- Can declare with
  - `var` (old-style function scope)
  - `let` (new-style block scope)
  - `const` (new-style block scope)
- `var` declarations are 'hoisted' to the top of the block
- In non-strict undeclared variables are global

TYPES

Six primitive types

- boolean (`true` and `false`)
- null
- undefined
- number (no separate int) see Number and Math
- string see String
- symbol (immutable)
- Also objects and functions (non-primitive)

USING TYPES

- Values have types
- No type for variable declarations: dynamic typing
- Function parameters do not have types
- Might choose to document e.g. parameters with comments
- `typeof` find the type of a value

CONTROL STRUCTURES

```
if (condition) {
  statement_1;
} else {
  statement_2;
}
```

See also `while`, `for`, `switch`, `do`, `throw`, `try`, `catch`, ternary

TRUE, FALSE, TRUTHY AND FALSY

These are all 'falsy':

- false
- undefined
- null
- 0
- NaN
- the empty string ("")

Can use for default values e.g

```
var x = x || 4;
```

Useful when

- optional parameters have not been provided
- object properties might not have been initialised

FUNCTIONS

- Are first-class objects and can be
  - assigned to variables
  - passed as parameters
- Often used for defining event callbacks
- Don't have to be associated with objects but can be

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

FUNCTIONS AS VALUES

Almost equivalent is

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

or

```
var sum2 = (a,b) => a+b;
```

ARRAYS

Square bracket notation: like python, Java, C

These are equivalent:

```
var arr = new Array(1,2,3);
```

```
var arr = Array(1,2,3);
```

```
var arr = [1,2,3];
```

Arrays can contain elements of different types

ARRAY ITERATION

```
for(var i=0; i < arr.length; i++){
   console.log(arr[i]);
}
```

See methods in Array e.g.

```
arr.push(4);
```

OBJECTS

- Objects have named properties
- Properties can have any type (including object, function)
- A bit like Java Map and python Dictionary
- Create with `Object` constructor or literal syntax
- Access with dot or bracket
- Inheritance through prototypes

```
var myCar = new Object();
myCar.make = 'VW';
myCar.model = 'Touran';

console.log(myCar.make);
console.log(myCar['make']);
```

SEE ALSO

- modules: require and module.exports
- regexps
- backticks (string expansion)
- Set, Map
- spread operator
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide
- https://www.theodinproject.com/courses/web-development-101/
- https://www.w3schools.com/js/default.asp