Web Applications

CSE183

Fall 2020

JavaScript I

Baskin Engineering UC SANTA CRUZ

Notices

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 2** due 23:59 **Thursday October 15**

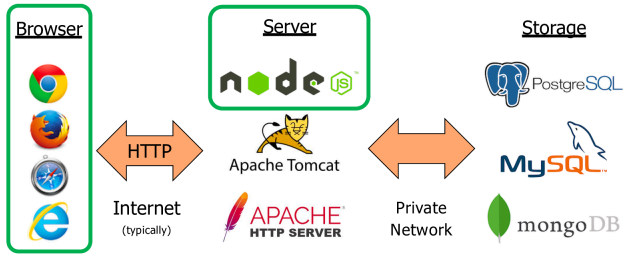
- **Quiz 1** during lecture **Friday October 16**

Today's Lecture

- JavaScript Overview
- Dynamic Types
- Scoping & Hoisting
- Basic Types
- Classes, Objects, Properties
- Arrays, Dates, Regular Expressions
- Exceptions
- Including JavaScript in HTML

- Questions

Full Stack Web Applications



JavaScript - Overview

- High-level
  - Heavily abstracted from hardware details
- Interpreted
  - Not compiled, executed by a platform-dependent run-time environment
- Dynamic
  - Undertakes compiler-like operations at runtime
- Untyped / Dynamically Typed
  - Any variable can hold any type of data
- Prototype-based
  - Object-oriented behaviors are re-used (inherited) from existing objects (prototypes)
- Has first-class functions
  - Functions are objects and can be manipulated as such

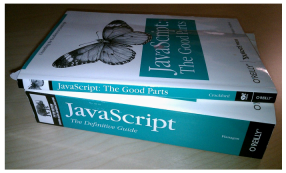
JavaScript - Overview Cont.

- Programming Models
  - Object-oriented
    - Encapsulation, abstraction, inheritance, polymorphism,
  - Imperative
    - Instructions executed sequentially; code is easy to understand
  - Functional
    - Declarative composition of value returning functions creating a call tree rather than manipulating a global state
- Not particularly like Java
  - Both heavily influenced by C, but took different paths
- ECMAScript
  - JavaScript Standard supported by most browsers
  - New version every year
    - Current is the 11th edition, "EMCAScript 2020"

https://www.ecma-international.org/ecma-262/11.0/index.html

JavaScript - Overview Cont.

- Early versions earned a bad reputation
  - Invented at Netscape in 1995
  - Appeared to have been designed in a rush
  - Grew rapidly with little consideration for cross-browser compatibility
  - ECMAScript initiative calmed things down
- The upside
  - It's feature rich ☺
- The downside
  - It's huge and complicated ☹
  - Code quality checkers are essential
    - Eg. `jshint`, `jslint`
- The real upside
  - We can use the same language in the browser and the server ☺ ☺ ☺



Heavily Based on C

```
int dat = 3;
int sum = 0;
int i = 0;

int bar(int p) {
  return p * 0.4;
}

void foo() {
  dat = dat * 6 - 2 + (dat / 10);
  while (dat >= 0) {
    sum += dat * dat * 1.2; // line comment
    dat--;
  }
  for (i = 0; i < 4; i++) {
    /* block comment */
  }
  if (dat < 2) {
    dat = bar(dat);
  } else {
    dat = dat * 0.04;
  }
}

int main(int argc,  char *argv[]) {
  foo();
  printf("dat: %d\nsum: %d\n", dat, sum);
}
```

```
var dat = 3;
var sum = 0;
var i = 0;

function bar(p) {
  return p * 0.4;
}

function foo() {
  dat = dat * 6 - 2 + (dat / 10);
  while (dat >= 0) {
    sum += dat * dat * 1.2; // line comment
    dat--;
  }
  for (i = 0; i < 4; i++) {
    /* block comment */
  }
  if (dat < 2) {
    dat = bar(dat);
  } else {
    dat = dat * 0.04;
  }
}

foo();
console.log(`dat: ${dat}\nsum: ${sum}`);
```

```
$ gcc -o SameAsC SameAsC.c
$./SameAsC
dat: 0
sum: 1789
```

```
$ node SameAsC.js
dat: -0.27999999999999975
sum: 1894.9560000000004
```

Why the difference?

Dynamic Typing

```
var i;          // typeof i == "undefined"
i = 128;        // typeof i == "number"
i = "hello";    // typeof i == "string"
i = true;       // typeof i == "boolean"
```

- Variables have the type of their most recent assignment
- Primitive types:
  `undefined`, `number`, `string`, `boolean` , `bigint`
- Structural types:
  `function`, `object`

Scoping & Hoisting

- Global and function scopes
- All `var` statements hoisted to function start

```
var gVar;

function() {
  // lVar2 is hoisted here
  var lVar;
  if (gVar > 0) {
    var lVar2 = 2;
  }
  // lVar2 is valid here - it has function scope
}

function foo() {
  // x is hoisted here
  x = 2
  var x;
  ...
}
```


Scoping & Non-Hoisting

- `var` has non-explicit scopes

```
function() {
  // str is hoisted here
  console.log('Str is:', str);
  ...
  for(let i = 0; i < 2; i++) {
    var str = "whatever";
    ...
  }
```

- `let` and `const` have explicit scopes

```
function() {
  console.log('Str is:', str);    // syntax error!
  ...
  for(let i = 0; i < 2; i++) {
    let str = "whatever";
    ...
  }
```


`Number` type

- Stored as 64-bit floating point number - like `double` in C
  `Number.MAX_SAFE_VALUE = 2^53 -1`
- Do NOT check for equality
  `0.1 + 0.2 != 0.3` (it's 0.30000000000000004)
- Peculiarities:
  - "not a number" and "infinity" are numbers
    `1/0 == Number.POISITIVE_INFINITY;`
    `Math.sqrt(-1) == Number.NaN;`
  - Bitwise operators (`~`, `&`, `|`, `^`, `>>`, `<<`, `>>>`) are 32-bit

`String` type

- Variable length (no equivalent of C `char` type)

```
var str = 'Hello World';
str.length == 11;
```

- Concatenation operator: `+`

```
str += '!';
str == 'Hello World!'
```

- Utility methods:

```
indexOf(), charAt(),
match(), search(), replace(), slice()
toUpperCAse(), toLowerCase()
```

etc. etc.

`Boolean` type

- `true` or `false`
- Plus `truthy` and `falsy` for conversions to Boolean
  - `falsy`

    `false`, `0`, `""`, `null`, `undefined`, `NaN`
  - `truthy`

    Implicitly `!falsey`

    All objects (including functions)

    non-empty strings

    non-zero numbers

`undefined` & `null`

- `undefined` : does not have a value assigned

```
var i;           // no initial value
i = undefined;   // explicit removal of value ( if any )
// i == undefined;
```

- `null` : A “special” value representing whatever you like
- Both are `falsy`, but *not* equal

```
null == undefined
null !== undefined
```

- Self Study:

  == & != abstract comparison operators

  === & !== strict comparison operators

`function` type

```
function foo(x) {
  return x > 1 ? 1 : x * foo(x-1);
}

// typeof foo == 'function';
// foo.name == 'foo';
```

- Same as:

```
var foo = function bar(x) {
  return x > 1 ? 1 : x * foo(x-1);
}
```

- Definitions are hoisted
- Implicitly support variable arguments
- All return a value

First class `function`

```
var vfunc = function (x) {
  console.log('Called with number', x);
  return x + 1;
};

function func(f) { // passed as a param
  console.log('Called with', f.toString());
  var ret = f(10);
  console.log('Return value', ret);
  return ret;
}

func(vfunc);
```

```
Called with function (x) {
  console.log('Called with number', x);
  return x+1;
}
Called with number 10
Return value 11
```


`Object` Type

- Unordered collection of name-value pairs

```
var foo = {};
var bar = {name: "Alice", age: 23, state: "California"};
```

- Name can be any string:

```
var x = { "": "empty", "---": "dashes"}
```

- Referenced like a structure

```
bar.name
// foo.nonExistent == undefined
```

- Or a hash table with string keys:

```
bar["name"]
```

- Global scope is an object in browser (i.e. `window[prop]`)

Dynamic Types - Properties

- Add:
  - Assign the property

```
var foo = { };
foo.name = "Foo's Name";
```

- Remove:

```
var foo = { name: "Foo's Name" };
delete foo.name;
```

- Enumerate:

```
var user = { name: "Alice", age: 23 };
console.log(Object.keys(user));
```

```
[ 'name', 'age' ]
```


Arrays

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

- Special objects: `typeof arr == 'object'`
- Indexed by non-negative integers: `arr[0] == 1`
- Can be sparse and polymorphic:
  `arr[5]='FooBar';` (`arr` is now `[1,2,3,,,'FooBar']`)
- Have many methods:
  - `push`, `pop`, `shift`, `unshift`, `sort`, `reverse`, `splice`, ...
- Can store object properties (e.g. `arr.name = 'Foo'`)
  - But some properties have special uses: (e.g. `arr.length = 0;`)
    - Self study:
      - Try to delete the `length` property from an array

`Date` Objects

```
var date = new Date();
```

- Special objects: `typeof date == 'object'`
  - The number of milliseconds since midnight January 1, 1970 UTC
  - Timezone needed to convert
  - Not good for fixed dates (e.g. birthdays)
- Many methods for getting and setting:

```
date.valueOf() == 1602463885467
date.toISOString() == '2020-10-12T00:51:25.467Z'
date.toLocaleString() == '10/11/2020, 5:51:25 PM'
```


Regular Expressions

- `var re = /ab+c/;` or `var re2 = new RegExp("ab+c");`
- Defines a pattern that can be searched for in a string
  - String: `search()`, `match()`, `replace()`, and `split()`
  - RegExp: `exec()` and `test()`
- `test`

	Returns
`/SS/.test(str);`	True if `str` has the substring `SS`
`/ss/i.test(str);`	Same but case insensitive
`/[Cc]se [0-9]/.test(str);`	True if str either “Cse N” or “cse N”

- `search`

`'XXX abbbbbbc'.search(/ab+c/);`	4 (position of ‘a’)
`'XXX ac'.search(/ab+c/);`	-1, no match
`'12e34'.search(/[^\d]/);`	2

Regular Expressions

```
var str = "This has 'quoted' words like 'this'";
var re = /'[^']*'/g;
var match = null;

while ((match = re.exec(str)) !== null)
  console.log(match);
```

```
[
  "'quoted'",
  index: 9,
  input: "This has 'quoted' words like 'this'",
  groups: undefined
]
[
  "'this'",
  index: 29,
  input: "This has 'quoted' words like 'this'",
  groups: undefined
]
```


Exceptions

```
try {
  undefinedFunction();
} catch (err) {
  console.error("Error:", err.name, ":", err.message);
} finally {
  console.log("Finally...");
}

function throwsString() {
  throw "Whoah Nelly!";
}

try {
  throwsString();
} catch (err) {
  console.error("Error", err.toString());
}

function throwsError() {
  throw new Error("Whoah Nelly!");   // better than throwing a string
}
```

JavaScript & HTML

- Include a separate file: (recommended approach)

```
<script type="text/javascript" src="foo.js"></script>
```

- Embed in the HTML:

```
<script type="text/javascript">
  console.log("Hello World!");
</script>
```


Upcoming Lectures

- Wednesday: JavaScript II
- Friday: Quiz 1 & Assignment 3
- Monday: JavaScript III

Tasks

- **Administration 1 & 2** due 23:59 **Thursday October 15**
- **Assignment 2** due 23:59 **Thursday October 15**