Introduction to Javascript

A brief overview

Lecture Goals

- Understand (at a high level) how a computer interprets code.
- Have a ***vocabulary*** to talk about programming.
- An ability to write and understand some JavaScript code.
- Understand *what to ask* and *where to look* when you get stuck.

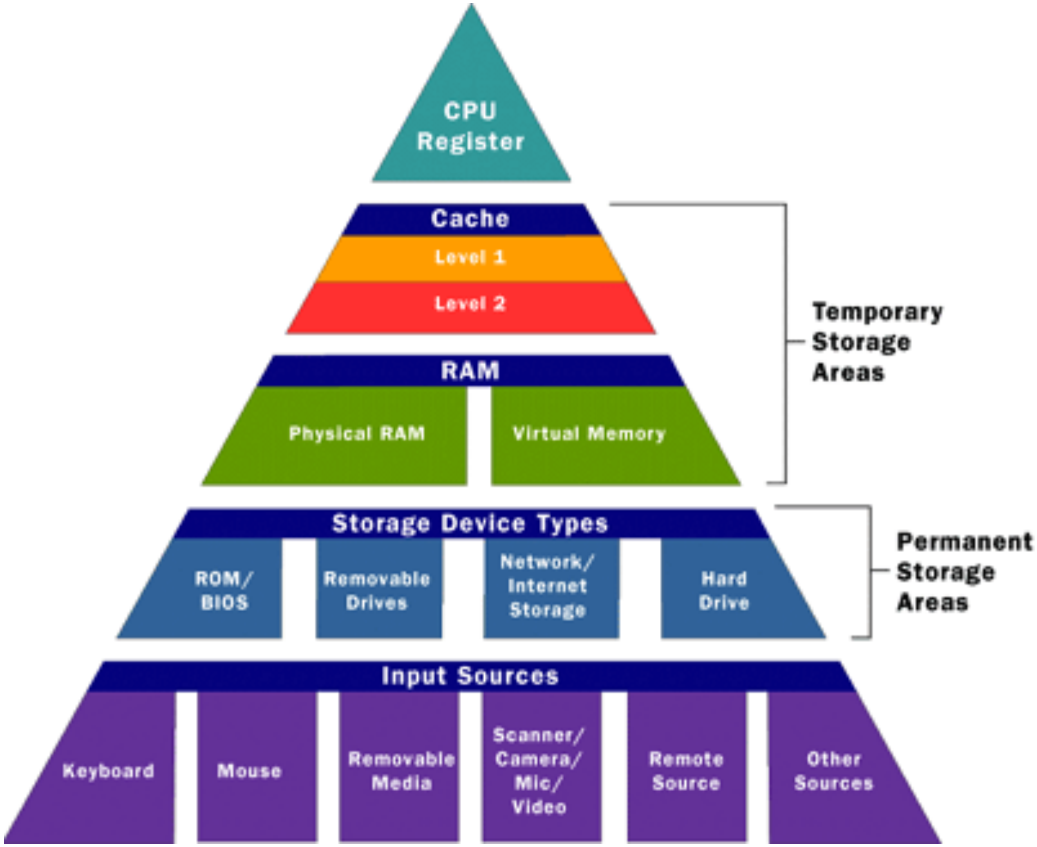
Computers (an overview)

- Computers have three fundamental parts:

    a. ***C***_entral_ ***P***_rocessing_ ***U***_nit_ (CPU)
    b. ***R***_andom_ ***A***_ccess_ ***M***_emory_ (RAM)
    c. ***Secondary Storage*** (often, a hard disk)

- These parts are used to run ***programs*** made up of a series of ***instructions*** to access and manipulate data.

Computers (an overview)



Taken from: http://computer.howstuffworks.com/computer-memory1.htm

Computers (an overview)

- The fundamental language of computers is ***binary***.

`010100101` = $2^7 + 2^5 + 2^2 + 2^0 = 165$

- A ***bit*** is one ***binary digit***.
- A ***byte*** is 8 bits.
- A ***megabyte*** is (roughly) 1 million bytes.
- A ***gigabyte*** is (roughly) 1 billion bytes.

Computers (an overview)

- In order to properly interpret a stream of 1's and 0's, some sort of ***data structure*** needs to be assumed.

e.g. every 4 bits is a number

`010100101` → `0`; `1010`; `0101`

- ***Data types*** are ways languages give structure to data.
- Examples of data types in JavaScript (more on this later):
    - `Number`, `String`, `Object`, `Array`, ...

Computers (an overview)

- Programs are written in a human-readable format.

- We need a mechanism to ***translate*** human-readable code into machine-readable numbers.

- ***Compilers*** provide one way to translate high-level (human-readable) code into low-level (machine-readable) code.

Computers (an overview)

```
function my_func(foo, bar) {
    var bud = foo;
    if (bar && typeof bar === "string") {
        bar = bar + "blah";
    }
    return bud + bar;
};
```

JavaScript Code

```
section .text
global _start, write
write:
        mov     al, 1 ; write syscall
        syscall
        ret
_start:
        mov     rax, 0x0a68732f6e69622f
        push    rax
        xor     rax, rax
        mov     rsi, rsp
        mov     rdi, 1
        mov     rdx, 8
        call    write

exit: ; just exit not a function
        xor     rax, rax
        mov     rax, 60
        syscall
```

Machine Code
(for an Intel CPU)

Compilation

→

Browsers

- All modern browsers contain a compiler for JavaScript.
- This is what makes JavaScript the "language of the web."
- Unlikely that a different language will have such broad support anytime soon.

Text Editors

- Can't use Microsoft Word to write code.

- Programmers use WYSIWYG editors, which allow them to see *exactly* what is inside a file.

- **W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et (WYSIWYG)

e.g. Notepad (Windows), TextEdit (OSX), *etc.*

NOTE: **make sure files are being saved as plaintext! You can look in Preferences for this option.**

JavaScript

- All languages have a ***grammar*** that defines valid ***syntax***.

- ***Programs*** can be thought of as a collection of ***statements*** made up of ***expressions*** written using the language's syntax.

- Each statement is executed in sequential order.

JavaScript

- In JavaScript, statements are separated by semicolons.

```
// statement 1: assign a variable
var my_name = "Samuel Messing";
// statement 2: launch a pop-up:
alert(my_name);
// statement 3: print to the console:
console.log(my_name);
```

- Lines beginning with `//` are ***comments***, and are ignored.

JavaScript

- Consoles give us a way to execute individual statements of JavaScript.

- A good way to make sure you understand what your code is doing.

- In Chrome:

  View->Developer->JavaScript Console

- Other browsers: http://webmasters.stackexchange.com/questions/8525/how-to-open-the-javascript-console-in-different-browsers

JavaScript

- The next statement isn't executed until the current statement is finished (`statment 3` didn't execute until we closed the alert window).

- This is called ***synchronous*** execution (as opposed to ***asynchronous***, which is an important technique in web programming).

Variables

Variables are named values.

```
var x = 1;
var y = 2;
var z = 0;
var point = {x: x, y: y, z: z};
var point_array = [point, point];
```

Data Types

Variables have specific data types.

```
// number
var x = 1;
var y = 2;
var z = 0;
// object
var point = {x: x, y: y, z: z};
// array
var point_array = [point, point];
```

Data Types (continued)

- Data types specify what operations are valid, and what those operations do.

```
var x = 1, y = 2;
x + y; // 3
var name = "sam";
x + name; // "1sam"
var foo = [];
foo.length; // 0
var bar = {};
bar.length; // undefined
```

JavaScript API

***A**pplication **P**rogramming **I**nterfaces* (APIs) give programmers a source of documentation for a language or collection of code.

API for JavaScript:

http://www.w3schools.com/jsref/default.asp

Arrays

- Arrays are ***lists*** of data.
- In JavaScript, these lists can have any kind of data.

```
var my_empty_array = [];
var my_array = [1, 2, 3, 4];
var my_mixed_array = ['foo', 1, []];
```

Arrays (continued)

- Arrays have special methods to help you manipulate them.

```
var my_array = [];
my_array.push(1); // [ 1 ]
my_array.push(2, 3); // [ 1, 2, 3]
// slice(index, delete_count);
my_array.splice(1, 1); // [ ????? ]
```

Objects

Objects in JavaScript are collections of ***key-value pairs***. Think of them like dictionaries: we give it a word (key) and get out a definition (value).

```
var obj = {
    'key_1': 'value_1',
    'key_2': 'value_2'
};
obj['key_1']; // 'value_1'
```

Objects (continued)

```
var user = {
  'name': 'Samuel Messing',
  // values can be any type:
  'age': 25,
  'email': 'sbm2158@columbia.edu'
};

user['name']; // 'Samuel Messing'
user['name'] = 'Bob';
```

Objects (continued)

Objects allow us to organize data.

```
var university = {
   'name': 'Columbia University',
    'founded': 1754
};

// Objects can be nested within others:
user['university'] = university;
delete user['university'];
```

Special Values

```
NaN // literally, not a number
Infinity
undefined // trying to access
          // something that
          // doesn't exist
null // represents absence
```

Special Values (continued)

```
1 / 0; // Infinity
10 / "seventeen"; // NaN
var boo = {};
boo.length // undefined
```

Boolean Algebra

Programming languages use ***boolean algebra*** to define different ***conditions***.

```
var true_expression = true;
if (true_expression) {
  console.log('expression was true');
} else {
  console.log('expression was false');
}
```

Boolean Values

There are two valid ***boolean values***: `true` and `false`. There are several ***boolean operations*** that enable us to combine values.

```
true && false; // "true AND false" (false)
true || false; // "true OR false" (true)
true && ! false; // "true AND NOT false" (true)
```

Iteration

Often we want to iterate over many values. ***for-loops*** and ***while-loops*** allow us to do this.

```
// for (initialization;
//      condition_to_test;
//      update)
for (var i = 1; i < 10; i = i + 1)
{
  console.log(i);
}
```

Iteration (continued)

```
// while (cond_is_true) {
//    do_stuff;
// }

var i = 1;
while (i < 11) {
  // something is wrong...
  console.log("i is: " + i);
}
```

Iteration (continued)

```
var i = 1;
while (i < 11) {
  console.log("i is: " + i);
  i = i + 1;
}
```

Functions

Functions take ***input values*** and transform them through a series of statements. They can also have a (single) ***output value***.

```
function add(x, y) {
   return x + y; // output x + y
};
add(2, 3); // 5
var foo = function (x, y) {
   return x - y; // output x - y
};
foo(3, 2); // 1
```

Functions (continued)

Functions can be assigned to variables. *This is a very significant and special property of JavaScript*.

```
var scare_me = function () {
  alert('BOO');
};
// setTimeout(function_to_call, ms_from_now)
setTimeout(scare_me, 1000);
```

The Sandbox

```
<!DOCTYPE html>
<html>
   <body>
      <script src="sandbox.js"></script>
   </body>
</html>
```

Have `sandbox.js` defined in the same folder, it will be loaded and run immediately.

Timing issues

`sandbox.js` may run before all of the HTML on a page is loaded!

A better ***pattern*** is to define a ***main function*** to run once a page is loaded. And to use `<body onload="main()">` to ***call*** the function once the whole page is loaded.

Timing issues (continued)

Example: `table_color_broken`

https://gist.github.com/smessing/5066406

shortened: http://goo.gl/YqwuD

Fix: `table_color` (using the main function pattern)

https://gist.github.com/smessing/5062870

shortened: http://goo.gl/UZwjB

Debugging

Chrome gives us ***Developer Tools*** in order to help debugging JavaScript.

First thing to do when something isn't working is to check the console for ***error messages***.

A good pattern is to find an error message, Google it, and then try some of the suggestions.

Appendix

A good general read on computers:

D is for Digital: What a well-informed person should know about computers and communications

by Brian Kernighan

Covers a lot of different aspects of Computer Science, Programming and Networking. Specifically written for a non-technical audience.