CS 2340 Objects and Design

Javascript!

Matt Hughes
matthugs@gatech.edu

What is Javascript?

- Extremely popular for web front-end programming
- *Not* Java . . . at all
- Tragically misunderstood (but maybe not)
- Functional, object-oriented, prototype-based

What is Javascript?

- Extremely popular for web front-end programming
- *Not* Java . . . at all
- Tragically misunderstood (but maybe not)
- Functional, object-oriented, prototype-based

Variables do *not* have static type

This means that you can use a variable to hold anything you like:

```
var magicallyTransformingVariable = {}; // now it's an object!
magicallyTransformingVariable = []; // now it's an array!
magicallyTransformingVariable = 5; // now it's a number!
```

Coming from a language with strict type checking (like Java), this can be freeing. . . but also dangerous.

Let's unpack that first line

The `var` keyword is mandatory: you should always declare variables with `var`.

```
var exampleVariable = 5;
```

If you forget to use `var`, you'll make a global variable (which you don't want).

Let's unpack that first line

The `var` keyword is mandatory: you should always declare variables with `var`.

```
var exampleVariable = 5;
```

If you forget to use `var`, you'll make a global variable (which you don't want).

Let's unpack that first line

The `var` keyword is mandatory: you should always declare variables with `var`.

```
var exampleVariable = 5;
```

If you forget to use `var`, you'll make a global variable (which you don't want).

Built-in types

Not having static type-checking doesn't mean there are no types:

- `Number`
- `String`
- `Boolean`
- `Array`
- `Object`
- `Null`
- `Undefined`

You can ask the type a variable holds with `typeof`.

The Number type

Unlike in many languages, there is only one variety of number in Javascript. It is analogous to Java's `double`.

```
var sanity;
if(0.1 + 0.2 == 0.3) {
    sanity = "Verily, we have made contact!";
};
```

After this executes, what is the value of `sanity`?

Number madness

```
var sanity;
if(0.1 + 0.2 == 0.3) {
    sanity = "Verily, we have made contact!";
};
```

After this executes, `sanity` has the value `undefined`, the default value for unassigned variables.

To restore sanity, we have to do silly things:

```
if(10 * 0.1 + 10 * 0.2 == 10 * 0.3) {
```

(which works because `Numbers` which are integers behave as such)

Number madness

```
var sanity;
if(0.1 + 0.2 == 0.3) {
    sanity = "Verily, we have made contact!";
};
```

After this executes, `sanity` has the value `undefined`, the default value for unassigned variables.

To restore sanity, we have to do silly things:

```
if(10 * 0.1 + 10 * 0.2 == 10 * 0.3) {
```

(which works because `Numbers` which are integers behave as such)

Objects

```
var anObject = {};
anObject.name = "value";
```

This creates an empty object and adds into it the pair `name : value`. We could've done this all in one go:

```
var anObject = {name : "value"};
```

Javascript objects are at their heart an *associative array* (or a map, a dictionary).

Objects

```
var anObject = {};
anObject.name = "value";
```

This creates an empty object and adds into it the pair `name : value`.
We could've done this all in one go:

```
var anObject = {name : "value"};
```

Javascript objects are at their heart an *associative array*
(or a map, a dictionary).

Objects

You can do anything you like with objects. . . anything at all. . .
The only limit is yourself.

```
var anObject = {inner : {}};
```

You can assign anything to any position in objects you have access to.

Functions as Objects

Javascript considers functions to be *first-class objects*.

```
var aFunction = function whee() {
    return "Gettin' pumped about Javascript!";
};
```

This means that functions can be passed around wherever an object could be (which is everywhere).

Notably, you can also attach functions as members of objects:

```
var anObject = {method : aFunction};
```

Such functions are called *methods*.

Scope

Function scope is the only notion of scope we have in Javascript. Blocks (like those of Java's `for` and `if`) do not have separate namespaces.

```
function aFunction() {
    var secretPassword = "tacos";
}
```

After executing this, `secretPassword` is still `undefined`.

Functionception

Naturally you can put functions inside other functions.
Inner functions retain access to variables of their enclosing scope.

```
function aFunction() {
    var secretPassword = "tacos";
    function informationThief() {
        return secretPassword;
    }
}
```

Here the `informationThief` has access to `secretPassword`.

Closures

In the last example, our `informationThief` couldn't smuggle the password out of the enclosing scope.
But the `informationThief` is an *object*, so we can pass it out of that function's scope by returning it.

```
function aFunction() {
    var secretPassword = "tacos";
    return function informationThief() {
        return secretPassword;
    }
}
```

What happens if I execute `aFunction()();` ?

Encapsulation using closures

Functions used. . . that way. . . are called *closures*.
(There's a technical definition. I'll spare you.)
The reason why I'm telling you about closures is because they help us achieve *encapsulation* in Javascript. It is the only way.

```
function aFunctionYouMightActuallyDefine() {
    // all of your secrets
    return function publicFunction() {
        // whatever you want to let other people do
        // with your variables
    }
}
```

More Fun: Inheritance

I mentioned that Javascript uses something called "prototyping."

Each object has an internal reference to another object, its "prototype".

An object inherits members from its prototype if those members are not explicitly overridden.

Inheritance

```
var speaker = "cheese";
String.prototype.speakThineName = function() {
    return this;
};
```

Now `speaker.speakThineName()` returns `"cheese"`.
We could also do

```
speaker.speakThineName = function() {
    return "Heisenberg";
};
```

which would change the behavior of `speaker.speakThineName()`, but not anything else.

The End

Questions? Thoughts? Lingering feelings of disgust?