GENERAL ASSEMBLY

INTERMEDIATE JAVASCRIPT & JQUERY

Evan Johnson, Developer

EVAN JOHNSON

- Front-end Engineer, Amazon
- Self-taught
- Loves JavaScript

johnson.evan1@gmail.com

evblurbs

@ev_blurbs

AGENDA

- Prerequisites
- Tools
- Scope
- Closure
- for statement
- Arrays
- Closure Revisted
- jQuery Events
- jQuery AJAX
- Hosting Exercise
- Resources
- What’s next?

PREREQUISITES

PREREQUISITES

VARIABLES

```
var x = 18;
x = 18;
```

"You use variables as **symbolic names** for values in your application. The names of variables, called **identifiers**, conform to certain rules."

- Mozilla Developer Network (MDN)

ARITHMETIC OPERATORS

```
var x = 3;     // assigns the value 3 to x
x += 2;        // assigns the value 5 to x
x -= 3;        // assigns the value 2 to x (5-3)
x *= 12;       // assigns the value 24 to x (2*12)
x /= 2;        // assigns the value 12 to x (24/2)
x %= 5;        // assigns the value 2 to x (12%5)
x = x++;       // assigns the value 3 to x (2+1)
x = x--;       // assigns the value 2 to x (3-1)
```

++	Increment
—	Decrement

COMPARISON OPERATORS

```
var x = '10'; // assigns the string '10' to x
var y = 2; // assigns the number 2 to y
var z = 10; // assigns the number 10 to z
x > y; // returns true
x < y; // returns false
x > z; // returns false
x >= z; // returns true
x <= z; // returns true
```

OBJECTS

```
var person = {
  firstName: "Jane",
  lastName: "Doe",
  age: 28
};

person.firstName; // returns "Jane"

person['firstName']; // returns "Jane"
```

FUNCTIONS

```
(function(myParameter) {
  // this is an anonymous self invoking function
  // your code block goes in betwen the brackets {}
  // you can access the parameter by it's name:
  // myParameter in this case

  // the block is ran immediatly
  // no need to invoke this function
})(myParameter);
```

PREREQUISITES

JQUERY

```
var hoverIn = function() {
  // hover in statement
};

var hoverOut = function() {
  // statement for hover out
};

$("mySelector").hover( hoverIn, hoverOut );
  // hover out statement
});
```

TOOLS

SUBLIME TEXT

- A text editor for code
- Can extend with packages (i.e. syntax highlighting, linting, etc.). Requires installing Package Control (https://packagecontrol.io/)
- http://www.sublimetext.com/

CHROME

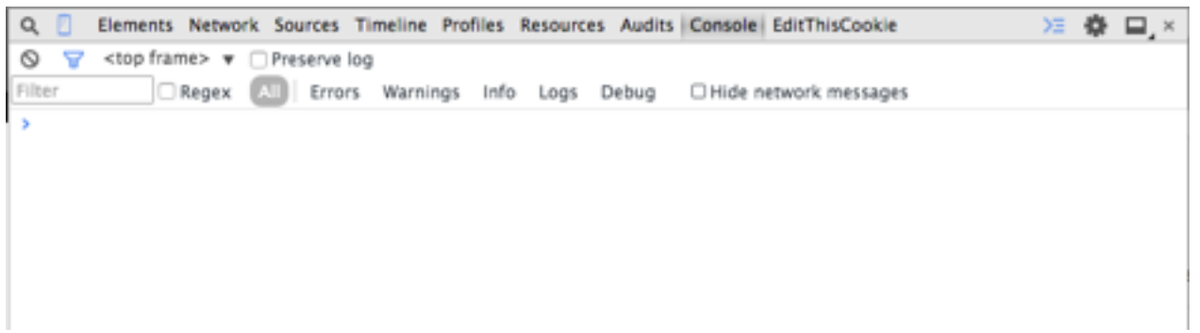
- Webbrowser
- Advanced developer tools
- https://www.google.com/chrome/browser/desktop/

EXERCISE FILES

- https://github.com/evblurbs/intermediate-javascript
- Download files using git clone or the ‘Download Zip’ button
- After unzipping the files, copy them to a directory you want to work from (i.e. ~/Document, ~/Desktop)

CONSOLE

- Chrome Menu (top right hamburger button) -> Tools -> Developer Tools
- Mac shortcut: Cmd + Opt + i
- PC shortcut: F12, Ctrl + Shift + i



SCOPE

SCOPE

> "Scope is the set of rules that determines where and how a variable (identifier) can be looked-up."
>
> – Kyle Simpson, You Don't Know JS: Scope & Closure

To understand scope, let's briefly review the JavaScript Engine:

- *Engine*: Responsible for the execution of our JavaScript program.
- *Compiler*: Engine's assistant; handles parsing and code generation.

SCOPE

- *Scope*: Another assistant to the Engine;

“[Scope] collects and maintains a look-up list of all the declared identifiers (variables), and enforces a strict set of rules as to how these are accessible to currently executing code.”

- Kyle Simpson, You Don’t Know JS: Scope & Closure

FUNCTION SCOPE

Every variable defined in a function, is available for the entirety of that function.

```
var myFunction = function() {
  var workshop = "Intermediate JavaScript and jQuery";
  console.log(workshop);    // logs "Intermediate JavaScript.."
};

console.log(workshop);      // ReferenceError: workshop is not defined
```

EXERCISE

File: 001.1-scope.hmtl

Let me know if you get stuck, or have any questions!

GLOBAL SCOPE

Everything not defined inside a scope (i.e. function scope) will become a global variable.

```
var location = null;          // global variable

var myFunction = function() {
  var location = null;        // local variable to the myFunction scope
};
```

GLOBAL SCOPE

If you do not use the var keyword, you will create a global variable regardless of where you define that variable.

```
var location = null;          // global variable

var myFunction = function() {
  location = null;            // global variable
};
```

EXERCISE

File: 001.2-scope.hmtl

Let me know if you get stuck, or have any questions!

GLOBAL SCOPE

Why should we avoid using the global scope for our data/object references?

- You variables, functions, and objects might clash with the global variables JavaScript provides.
- We might overwrite some of the default JavaScript behaviors on accident (i.e. *location*)

EXERCISE

File: 001.3-scope.hmtl

Let me know if you get stuck, or have any questions!

GLOBAL SCOPE

Why should we avoid using the global scope for our data/object references?

- You variables, functions, and objects might clash with the global variables JavaScript provides.
- We might overwrite some of the default JavaScript behaviors on accident (i.e. *location*)
- Our variables can clash with themselves!

PRINCIPLE OF LEAST PRIVILEGE

> “This principle states that in design of software, such as the API for a module/ object, you should only expose only what is minimally necessary, and ‘hide’ everything else.”
>
> – Kyle Simpson, You Don’t Know JS: Scope & Closure

In other words, our *student* object doesn’t need to know the *name* of our *workshop* object, and vice versa.

IIFE TO THE RESCUE

- Immediately-Invoked Function Expression
  (aka self-invoked anonymous function)
- By wrapping our code in an IIFE, we remove our variable declarations from the Global Scope
- We are able to use Function Scope to avoid name clashing, manipulating the global objects, and more.

IIFE TO THE RESCUE

```
(function IIFE() {
  var workshop = "Intermediate JavaScript & jQuery";      // local variable to the IIFE function scope
})();

var workshop = "Intermediate JavaScript & jQuery";        // global variable set to the window object
```

- We turn a function into an expression by wrapping it in a pair of ()
- The second pair of () executes the function.
- The same concept is used with anonymous functions.

EXERCISE

File: 001.4-scope.hmtl

Let me know if you get stuck, or have any questions!

BREAK

LAB

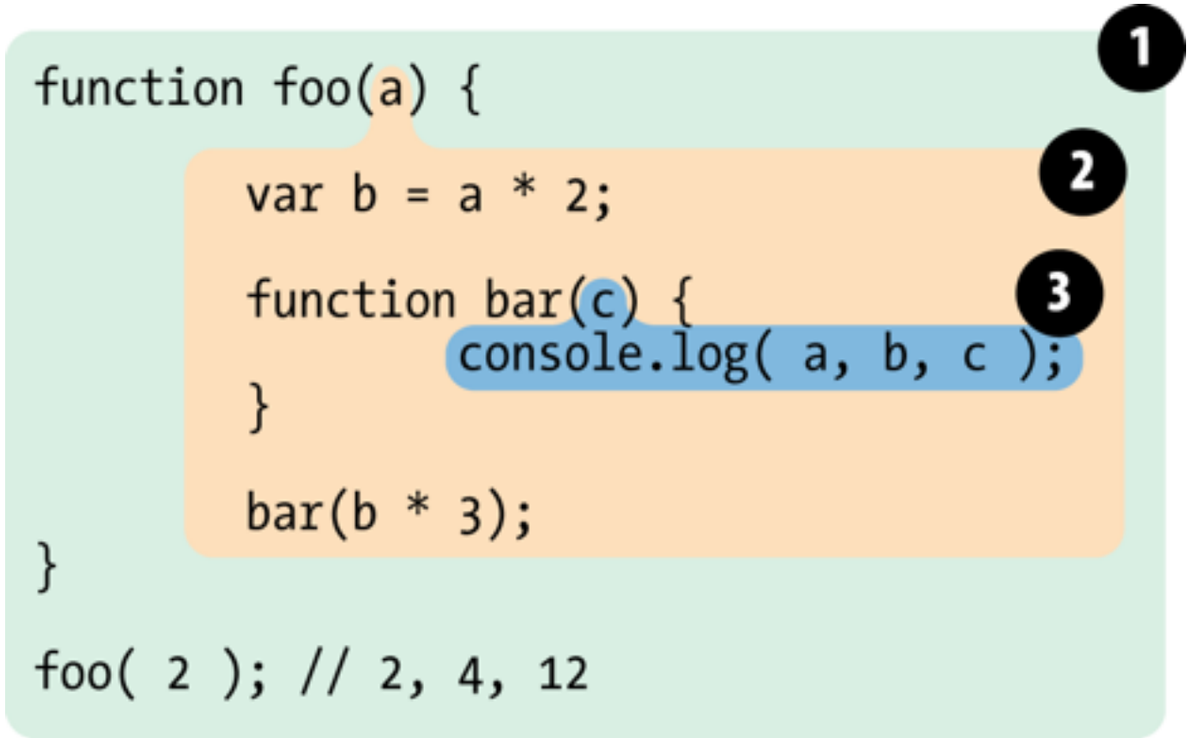
Files: project/instasearch.html, project/js/instasearch.view.js

CLOSURE

CLOSURE

"Closures are functions that refer to independent (free) variables. In other words, the function defined in the closure 'remembers' the environment in which it was created. "

- MDN



* Image from You Don’t Know JS: Scope & Closure

SCOPE CLOSURE

- If one function is declared inside another, the nested function has access to all variables declared in the parent function.
- Function nesting can go multiple levels. A nested function has access to all the variables declared in its parent functions - i.e, parent function, grandparent function, great grandparent function, and so on.
- The Global Scope is an example of this - all functions have access to global variables, or variables declared at the Global Scope.

EXERCISE

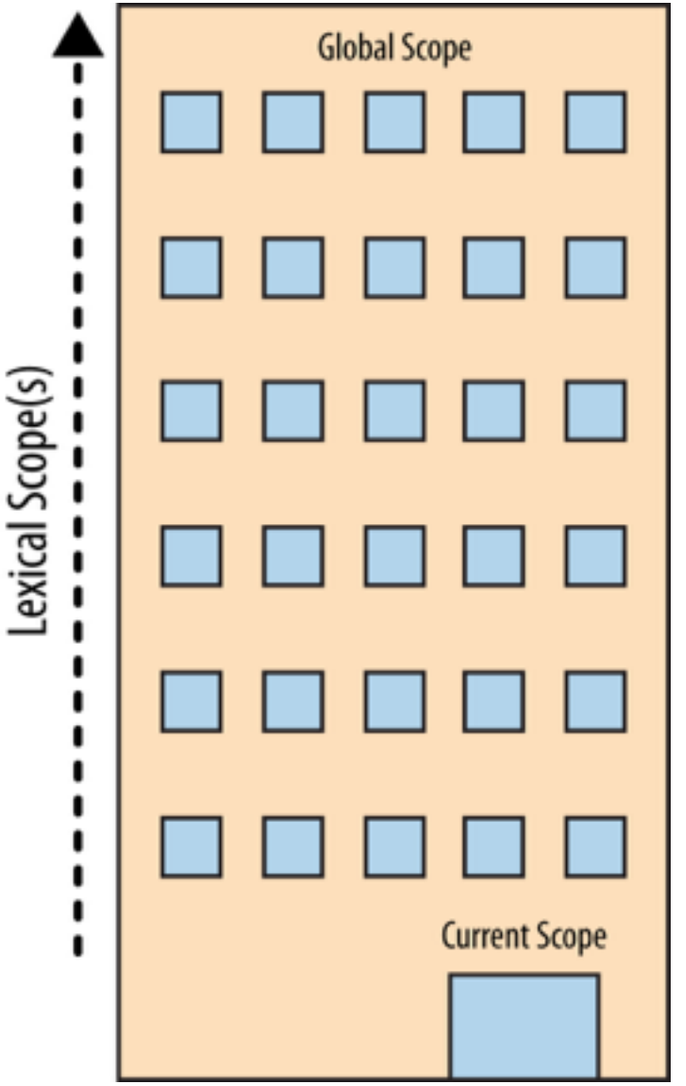
File: 003.1-closure.hmtl

Let me know if you get stuck, or have any questions!

LEXICAL SCOPE

Scope Closure in JavaScript an example of Lexical Scope - a common convention in programming languages that sets the scope of a variable.

- Assistant to the JavaScript Engine: helps the Engine look up variables.
- Starts at the Current Scope, and works it’s way up to the Global Scope.



VARIABLE SHADOWING

Since the Scope will stop looking for a variable once it finds it, you can override variables declared in a parent scope by creating another local variable.

```
function foo() {
  var a = 1;                // local variable set to the scope of foo()
  function bar() {
    var a = 2;              // local variable set to the scope of bar()
  }
  bar();
}
foo();
```

REASSIGN VARIABLES

Exactly how we reassigned the global variable *location* earlier, you can reassign local variables inside a nested scope.

```
function foo() {
  var a = 1;                // local variable set to the scope of foo()
  function bar() {
    a = 2;                  // local variable set to the scope of foo()
  }
  bar();
}
foo();
```

EXERCISE

File: 003.2-closure.hmtl

Let me know if you get stuck, or have any questions!

FOR STATEMENT

"The **for statement** creates a loop that consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a statement or a set of statements executed in the loop."

- MDN

```
for ([initialization]; [condition]; [final-expression])
   statement
```

FOR STATEMENT

```
for ([initialization]; [condition]; [final-expression])
   statement
```

initialization: “An expression (including assignment expressions) or variable declaration. Typically used to initialize a counter variable. This expression may optionally declare new variables with the var keyword. These variables are not local to the loop, i.e. they are in the same scope the for loop is in. The result of this expression is discarded.” - MDN

FOR STATEMENT

```
for ([initialization]; [condition]; [final-expression])
   statement
```

condition: “An expression to be evaluated before each loop iteration. If this expression evaluates to true, statement is executed. This conditional test is optional. If omitted, the condition always evaluates to true. If the expression evaluates to false, execution skips to the first expression following the for construct.” - MDN

FOR STATEMENT

```
for ([initialization]; [condition]; [final-expression])
   statement
```

final-expression: “An expression to be evaluated at the end of each loop iteration. This occurs before the next evaluation of condition. Generally used to update or increment the counter variable.” - MDN

FOR STATEMENT

```
for ([initialization]; [condition]; [final-expression])
   statement
```

statement: “A statement that is executed as long as the condition evaluates to true. To execute multiple statements within the loop, use a block statement ({ ... }) to group those statements.” - MDN

FOR STATEMENT

```
for ([initialization]; [condition]; [final-expression])
   statement
```

```
1 for (var i = 0; i < 9; i++) {
2    console.log(i);
3    // more statements
4 }
```

EXERCISE

File: 004-for-statement.hmtl

Let me know if you get stuck, or have any questions!

ARRAYS

JavaScript Arrays allow you to store multiple values in a single variable. Think of it as a list for JavaScript.

- Can be accessed by their position in the array. The *key* to access them is an integer based on their position.
- They use a zero-based index - meaning that the first item is in position *0*, second in *1*, third in *2*, and so on.
- Can find the number of items in an Array by appending *.length* to an Array variable.

ARRAYS

```
var fruit = ["apple", "banana", "grape"];

fruit[0];          // "apple"
fruit[2];          // "grape"
fruit[1];          // "banana"
fruit[3];          // undefined

fruit.length;      // 3
```

EXERCISE

File: 005-arrays.hmtl

Let me know if you get stuck, or have any questions!

LAB

Files: project/instasearch.html, project/js/instasearch.view.js

AJAX

AJAX

- AJAX stands for Asynchronous JavaScript and XML
- Classic webpages would have to reload pages to fetch new content. AJAX allows us to fetch content via JavaScript.
- While the X stands for XML, most applications use JSON to fetch data.

AJAX

JSON

```
{
  "key": "value",
  "objectKey": {
    "key1": "objectValue1",
    "key2": "objectValue2"
  },
  "arrayKey": [
    {
      "arrayKey": "arrayValue1"
    },
    "arrayValue2"
  ]
}

data.key;                    // "value"
data.objectKey.key2;         // "objectValue2"
data.arrayKey[0].arrayKey;   // "arrayValue1"
data.arrayKey[1];            // "arrayValue2"
```

AJAX

JQUERY AJAX

jQuery.ajax(url [, settings]) — version added: 1.5

url
Type: String
A string containing the URL to which the request is sent.

settings
Type: PlainObject
A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup(). See jQuery.ajax(settings) below for a complete list of all settings.

jQuery.ajax([settings]) — version added: 1.0

settings
Type: PlainObject
A set of key/value pairs that configure the Ajax request. All settings are optional. A default can be set for any option with $.ajaxSetup().

AJAX

JQUERY GET

jQuery.get([settings]) — version added: 3.0

settings
Type: PlainObject
A set of key/value pairs that configure the Ajax request. All properties except for `url` are optional. A default can be set for any option with $.ajaxSetup(). See jQuery.ajax(settings) for a complete list of all settings. The type option will automatically be set to `GET`.

jQuery.get(url [, data] [, success] [, dataType]) — version added: 1.0

url
Type: String
A string containing the URL to which the request is sent.

data
Type: PlainObject or String
A plain object or string that is sent to the server with the request.

success
Type: Function(PlainObject data, String textStatus, jqXHR jqXHR)
A callback function that is executed if the request succeeds. Required if `dataType` is provided, but you can use `null` or `jQuery.noop` as a placeholder.

dataType
Type: String
The type of data expected from the server. Default: Intelligent Guess (xml, json, script, or html).

AJAX

JQUERY GET

```
$.get(url, function(jsonData) {
  // do something with our
  // our jsonData here
}, "jsonp");
```

LAB

Files: project/instasearch.html, project/js/instasearch.model.js, project/js/instasearch.controller.js

CLOSURE

ESCAPING CLOSURE

```
function handleResults(dataArray) {
  // hook it up to our view function
  instasearch.view(dataArray);
}
```

```
if(searchTerm.length) {
  instasearch.model(searchTerm);
}
```

Q&A

THANKS!

NAME

- Optional Information: Want more assignments? Follow the Github repository. I'll add some homework there when I get a chance.
- Email: johnson.evan1@gmail.com (send me any questions you may have)
- Twitter: @ev_blurbs