Announcements

- http://lesscss.org/
- http://learnlayout.com/
- Google Fonts API - https://developers.google.com/fonts/
- Open Source Web Design (Free web design templates)
  - http://www.oswd.org/
- Color Palette Generator
  - http://www.degraeve.com/color-palette/

Objects

- **Object**
  - Type provided by JavaScript
  - Objects are different than in class-based systems (at least for now ☺)
  - Objects are created based on another object, either native or user-defined one
  - You can dynamically add properties and functions (methods) to objects
  - Creation (using new)

  var obj = new Object(); // You can omit () but don't do it
  - Creating (using literal notation)
- **Example:** DynamicPropFunc.html

Objects

- **Object Properties and Methods**
  - toLocaleString – string representation for current locale
  - toString – string representation
  - valueOf – Returns a number, string or boolean that corresponds to the object
  - Constructor – Function used to create the object
    - For previous example constructor was the Object() function
  - isPrototypeOf
  - hasOwnProperty – object has a property itself (not inherited)
  - propertyIsEnumerable
- ECMAScript objects inherit the above properties
- Host Objects
  - **Examples:** window, document
  - Provided by the host implementation; may not inherit from Object
- You can determine type using
  - typeof
  - Instance of
- **Example:** Type.html

Global Object

- ECMAScript defines a global object
  - In JavaScript **window** implements the global object (among other things)
  - All functions and variables defined globally become part of the Global object
- Some functions that are part of the Global object
  - isNaN
  - parseInt
  - parserFloat
- Some properties that are part of the Global object
  - NaN
  - undefined
  - Object → Constructor for Object
  - Array → Constructor for Array
  - Function → Constructor for Function
  - Number → Constructor for Number
  - String → Construct or for String
  - Date → Constructor
  - Error → Constructor
  - RegExp → Constructor
- ECMAScript also defines the Math object

Additional Objects

- ECMAScript defines Wrapper objects for primitives:
  - Boolean, Number, String
  - An object is created when primitive type treated as an object
- At the beginning of code execution two built-in objects are created:
  - **Global** and **Math**

Functions

- They are objects
- Each function is an instance of the **Function** type
- The name of the function is a reference
- Functions can be passed and returned from other functions
- Functions can be defined inside of other functions
- Three approaches to define functions
  - **Function declaration**
    - Read and available before any code is executed
      - Function declaration hoisting
  - **Function expression**
  - **Using Function constructor**
- **Example:** DefiningFunctions.html
- Function overloading is not possible
  - Second function redefines the first one

Execution Context

- Execution context – defines what data a function or variable can access
- Each context has a **variable object** where variables and functions exist
- Two primary execution contexts:
  - **Global execution context** – outermost one
    - window object in JS
  - **Function execution context**
    - When a function is call a new context is created
- When code is executed, a scope chain of variable objects is defined which provides access to all the functions and variables a context has access to
- Front of the scope chain is the variable object of the context whose code is being executed
- Last element in the chain is the global context's variable object
- **this** - Reference to the context object the function is operating on

Execution Context

- Resolution of identifiers is done by traversing the chain
- Inner contexts can access outer contexts, but not vice versa
- Example

```
var music = “classic”;
function addMusic(y) {
    var popularSinger = “TaylorSwift”;
     function singSong() {
        …
     }
}
```

- Keep in mind that objects have a prototype that defines variables and functions. Searching for a function or variable may require searching the prototype chain of an object in the scope chain
- Notice that the “scope” behavior is similar to what you already have seen in Java

Functions Properties and Methods

- Properties
  - length – number of arguments expected
  - prototype – where instance methods can be found (e.g., toString(), valueOf(), etc.
  - **Example:** FuncLength.html
- Inside of a function two object exists
  - **Argument** - Has all the arguments passed into the function
  - **Example:** FuncArguments.html
  - **this**
    - Reference to the context object the function is operating on
    - **Allows associating functions to object until runtime**
    - You can set the this value using apply(), call() or bind
    - **Example:** FuncThis.html, FuncApplyCallBind.html

Creating Objects

- ECMAScript 5 does not provide a way to define classes
  - ECMAScript 2015 does!
- Different approaches has been developed to address the creation of objects associated with a particular abstraction.
  - **Constructor Pattern**
  - **Prototype Pattern**
  - **Constructor/Prototype Pattern**
- **Constructor function**
  - It is not a special function
  - Any function called with the new operator behaves as a constructor. Without it the function behaves as a normal function
- **Example:** ConstructorPattern.html
  - Disadvantage: duplicating info object

Prototypes

- Prototype property – object containing properties and methods that are available to instances of a particular type
- Instances has a pointer to the prototype
- Prototype chaining – primary method for inheritance
- When a property/function is searched the search starts with the instance; if not found there the prototype is searched
- The hasOwnProperty allow us to tell if the property exists on the instance or the prototype
- All reference types inherit from Object (Object.prototype)
  - Object.prototype is the default prototype
- Object.prototype has
  - toString()
  - valueOf()
  - hasOwnProperty
  - …

Default Pattern for Custom Types

- The Constructor pattern for custom type definition has some disadvantages
  - Each instance has its own copy of methods
- The Prototype pattern helps with this situation
- **Example:** PrototypePattern.html
  - Notice that sharing is a problem for certain properties using the Prototype Pattern
- The **default pattern** for custom type definition ("class definition") combines the constructor and prototype pattern
  - Constructor pattern defines instance variables
  - Prototype pattern defines common methods and properties
- **Example:** DefaultPattern.html
- **Note:** Notice that even if instances for an object has been created adding a property/method to the prototype will make it immediately available

Inheritance

- Prototype chaining – primary method for inheritance
- We can assign a particular object to the prototype property
- **Example:** Inheritance.html

for/in

- General form

  for (property in object)

  statement

- for/in does not specify the order in which properties of an object are visited
- **Example:** ObjectEx.java
- The for/in does not loop through all the possible properties as some properties are considered non-enumerable
- User-defined properties are enumerable
- Let's see the properties of the window and document objects

Objects as a Maps

- We can also view an object as an entity that associates values with strings. How? Let's first see how we can use the [ ] operator to access properties
- You can use [ ] operator instead of **.** (period) operator

myObj.created → myObj["created"]

- IMPORTANT: Notice that we have a string on the right side ("created") whereas on the left side it is a property (variable)
- Using [ ] operator can provide a nice alternative to add properties to an object dynamically (when the program is executing)
- **Example:** AddingProperties.html

Traditional Server/Client Interaction

- Nothing happens until we submit data
- We must wait until the server request is processed (can do anything with the page)
- A page must be completed loaded even if most of the content identical to previous page
- Compare with a desktop application
- Can we do better? Can the page be updated without requiring a page load?
- AJAX is the answer
- EXAMPLE: **TypicalASynchronous folder**
  - directoryLookup.html, directory.php, processMemo.php

AJAX

- AJAX → Asynchronous JavaScript and XML
- Combination of technologies
- Adds a layer between the browser and the web server, handling server requests and processing the results
  - Layer Name → Ajax Framework/Ajax Engine
- The requests are not synchronized with user actions (e.g., clicking on links, buttons, etc.). User can continue interacting with the browser while request is being processed

AJAX

- In the traditional client/server model we submit server requests by clicking on a link or via submit (this generates the HTTP request for us)
  - Notice we get a new web page as a result
- XMLHTTPRequest
  - JavaScript object that will issue the HTTP request
  - No page load is generated as a result of the request
  - Can only issue request to URLs within the same domain
  - Cannot directly access a remote server
- There is nothing the server needs to do just because the request is associated with AJAX. The server is just receiving an HTTP request
- AJAX application just care about receiving an HTTP response

AJAX

- When/how to get the results
  - We need to add code to detect when the request has been completed
  - We need to add code to process the results (e.g., update page components)
- Creating the object (JavaScript code)

```
var requestObj = new XMLHttpRequest();
```

XMLHttpRequest Properties

- readyState → Request's status
  - 0 → ***uninitialized***, assumed when initial server request is submitted
  - 1 → ***loading***, placing data in XMLHTTPRequest object
  - 2 → ***loaded***, loading completed
  - 3 → ***interactive***, object interaction is possible
  - 4 → ***completed***
- onreadystatechange →event handler called when readyState property changes
- responseText → results in text form
- responseXML → results in XML
- status → HTTP status returned by the server
- statusText → HTTP reason phrase

XMLHttpRequest Functions

- open → Initializes the XMLHTTPRequest object
  - open(httpMethod, targetURL, requestMode)
    - httpMethod → GET or POST
    - requestMode → true for asynchronous, false for synchronous.
  - If GET is used targetURL must include any necessary parameters
- send(data) → sends the request
  - For GET request data is set to null
- **Example:** directoryLookup.html, directory.php, processMemo.php

Submitting a Synchronous Request

- We can also submit a synchronous request using Ajax
  - You want to get results from the server but cannot proceed without them
- EXAMPLE: **Synchronous folder**
  - directoryLookup.html,directory.php, processMemo.php

ECMAScript 2015

- ECMAScript 2015 Language Specification
  - http://www.ecma-international.org/ecma-262/6.0/ECMA-262.pdf
- 5 Great Features in ECMAScript 2015
  - http://www.wintellect.com/devcenter/nstieglitz/5-great-features-in-es6-harmony
- Compatibility table
  - http://kangax.github.io/compat-table/es6/
- Site:
  - http://www.es6fiddle.net/hrut24r0/
  - Select item from drop-down list

References

- Professional JavaScript for Web Developers, Third Edition, Nicholas C. Zakas
  - ISBN-13: **978-1118026694**
- Programming with JavaScript: Algorithms and Applications for Desktop and Mobile Browsers
  - ISBN-13: **978-0763780609**
- Ajax in 10 Minutes" by Phil Ballard, ISBN 0-672-32868-2