

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/>

JSON

- JSON – JavaScript Object Notation
 - Syntax for serializing objects, arrays, numbers, booleans, and null
 - Based on JavaScript syntax, but distinct from it
 - Some JavaScript is not JSON and some JSON is not JavaScript
- Lightweight data-interchange format
- Alternative to XML
- Derived from JavaScript but it is language independent
- JSON Example: <http://json.org/example.html>

JSON

- **Methods**

- **JSON.parse()** → parse a string as JSON (returns the Object corresponding to the JSON text)
- **JSON.stringify()** → returns a string corresponding to the specified value

- Examples and information:

https://www.w3schools.com/js/js_json_intro.asp

- References:

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON

- **Example:** JSONExample.html

Objects

- **Property** – association between a name and a value
 - When the value is a function the property is referred to as a method
 - Name can be any valid JavaScript string or anything that can be converted to a String (that includes empty string)
 - Any invalid property name can only be accessed using square bracket notation
- **Object** – Collection of properties
 - You can define your own; browser predefines a set of objects
 - A property can be seen as a variable associated with a value
 - Approaches to access and add properties
 - Using dot-notation
 - Using square brackets

Objects

- **How to create objects**

- Using Object constructor (e.g., `new Object()`)
 - Object constructor creates an object wrapper for the given value
 - `var x = new Object(true);`
 - If the provided value is null or undefined an empty object will be created
- Using object initializer/literal notation
 - An initializer is a list of zero or more property names/values in { }
 - Example: `var x = {}, y = { radius: 20 }`
- Using `Object.create`

- **Example:** `Objects.html`

Objects as 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

Object Type

- All objects in JavaScript are descended from **Object**
- In JavaScript objects have a property called **prototype**
- The prototype property points to an object from which properties are inherited
- Objects inherit methods and properties from `Object.prototype`
- **Prototype chain** – Set of objects defined by the prototype property
 - The end of the chain is a prototype property with the **null** value

Object.prototype

- **Methods:**
 - **Object.prototype.hasOwnProperty(prop)**
 - prop is a direct property (not inherited through the prototype chain)
 - **Object.prototype.isPrototypeOf(obj)**
 - **Object.prototype.toString()**
 - Returns a string representation of the object
 - **Object.prototype.valueOf()**
 - Returns the primitive value of the specified object

Enumerating Properties

- Three native ways to list/traverse object properties
 - **for...in loops**
 - Traverses all enumerable properties of the object and its prototype chain
 - **Object.keys()**
 - Returns array with all the own (not in prototype chain) enumerable properties
 - **Object.getOwnPropertyNames()**
 - Returns array with all own properties' names (whether enumerable or not)

Creating Objects Using Object.create

- You can also create object using Object.create()
 - It allows the specification of a prototype object for the object you want to create
- **Example:** ObjectCreate.html

Function Properties and Methods

- In JavaScript every function is a Function object
- The Function constructor creates a new Function object
- Length property
 - **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 Using Constructor Functions

- To create a custom object you can:
 - Create a function referred to as **constructor function**
 - Convention is to use an uppercase initial letter
 - Instantiate and initialize an object using new and the constructor function
 - Any function called with the new operator behaves as a constructor; without it the function behaves as a normal function
- **Example:** ConstructorFunction.html
 - Notice that in this example object creation is not efficient as we duplicate the code for the functions (we will see a better alternative later on)

Custom Type Definition

- ECMAScript 5 does not provide a way to define classes as in Java
 - ECMAScript 6 does!
- Different approaches has been developed to address the creation of objects associated with a particular abstraction
 - **Constructor Pattern**
 - **Prototype Pattern**
 - **Constructor/Prototype Pattern**
- **Example:** ConstructorPattern.html
 - Using constructor functions
 - Disadvantage: duplicating info object

Prototype Pattern

- The Constructor pattern for custom type definition has some disadvantages
 - Each instance has its own copy of methods
- The Prototype pattern addresses this situation
- **Example:** PrototypePattern.html
 - Notice that sharing is a problem for certain properties using the Prototype Pattern
- How sharing of prototype takes place when a constructor function is used to create an object using new:
 1. JavaScript creates a new empty object and calls the function with “this” referring to the new object
 2. The prototype property of the new object is initialized to point to the object referred to by the prototype property of the constructor function
 3. The new object is returned

Default Pattern for Custom Types

- 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

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
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working with Objects](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects)
- [https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance and the prototype chain](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Inheritance_and_the_prototype_chain)