JS

Pull from upstream!

Commit any changes first!

Slides created in conjunction with: Ilchul Yoon, Nelson Padua Perez

Agenda

- Higher order functions with arrays
- Symbols
- Maps/Sets
- Objects

Three More Notable Array Methods

- reduce
  - Executes a reducer function (callback) on each element of the array, resulting in a single output value
  - First argument of the callback function is “accumulator”
  - Passes the result of callback (the accumulator) from one array element to the other
- filter
  - Creates a new array with all elements that pass the test implemented by the provided function

Three More Notable Array Methods

- Map
    - The **map()** method **creates a new array** populated with the results of calling a provided function on every element in the calling array.
    - Syntax (how we will use it anyway):

```
let new_array = arr.map(function callback(
currentValue[, index]) {

    // return element for new_array

})
```

Symbols

- New primitive type in ES6
- Tokens that serve as unique ids
    - Create via the factory function Symbol()
    - “new” keyword does not work

```
Let x = Symbol(“Description”);
```

Symbols

- Can be used as special property keys
- Every symbol is unique
  - Symbol() === Symbol() is false
- Symbols can be used as property keys
  - Computed property key
  - Allows you to specify key of a property via an expression, by putting it in square brackets
- String value parameter is optional

Using Symbols

- Following operations ignore symbols
  - for-in loop
  - Object.keys()
  - Object.getOwnPropertyNames()
- Conversion of Symbol to Boolean returns true
- Can be used to represent concepts
  - const RED_COLOR = Symbol('red color');

Sets

- Collection of keys
- Keys can be primitive or references
- The Set constructor has zero or more arguments. With no arguments an empty Set will be created
- If an argument is specified, it needs to be iterable (e.g., array)
- When iterating over sets, elements will be processed in the order they were inserted

Maps

- Collection of keys
- Keys can be primitive or references
- The Set constructor has zero or more arguments. With no arguments an empty Set will be created
- If an argument is specified, it needs to be iterable (e.g., array)
- When iterating over sets, elements will be processed in the order they were inserted

Creating Maps and Sets

Map:

- let m = new Map();
- m.set(key, value);

Set:

- let s = new Set();
- s.add(value);

Immediately Invoked Function Expression (IIFE)

- A JS function that runs as soon as it is defined
- A design pattern known as a Self-Executing Anonymous Function
- Two parts
  - anonymous function with lexical scope enclosed within the Grouping Operator ().
  - Prevents accessing variables within the IIFE idiom as well as polluting the global scope.
- Emulating block-scoped variables
- Not needed, if “let” is used instead of “var”

Objects

- Just a 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

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

How do we create Objects?

- Using Object Constructor
- Using Object Initializer/literal notation
- Using Object.create

Objects as Maps

- We can also view an object as an entity that associates values with strings.
  - Use the [] operator

Ex: myObj.value == myObject[“value”]

Object Type

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

Object Prototypes

- 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
  - In ES6, Symbol.toPrimitive is a symbol that specifies a function valued property that is called to convert an object to a corresponding primitive value.

Object Constructors

- Rather than handwriting all values in an object, Javascript allows for Object Constructors

Ex:

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
```

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

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

Default Pattern for Custom Types

- The **default pattern** for custom type definition (“class definition”) combines the constructor and protoype 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 create 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

WTWAW

After today make sure you know how to:

- Create a symbol (and know it’s use)
- Use and manipulate maps and sets
- Create Objects all 3 ways
- Create an object constructor
- 3 different ways to create custom types