JavaScript II

Functions

- Functions in JavaScript are declared using the `function` keyword

```
function function_name(args){
}
```

- Functions are first class objects, they are stored in variables like everything else
  - Can be passed as arguments
  - Can be returned from functions

```
var function_name = function(args){
                                  }
```

In [ ]:

```
%%script node
function hello(){
    console.log("Hello World!")
}
hello()
```

In [ ]:

```
%%script node
function getDayOfWeek(date) {
        var days = ["Sunday", "Monday", "Tuesday", "Wednesday",
                    "Thursday", "Friday", "Saturday"];
        return days[date.getDay()];
      }
console.log(
    getDayOfWeek(new Date())
)
```

Custom Sorting

- Now that we know how to define a function, we can provide `Array.sort` with a custom function
  - The default sort is based on string representation
- The comparator should be a function that
  - Takes exactly two parameters, eg. `a` and `b`
  - Returns a negative number if `a < b`
  - Returns a 0 if `a = b`
  - Returns a positive number if `a > b`

In [ ]:

```
%%script node
var some_numbers = [45,63,22,9,13,2,1,64,"40"];
some_numbers.sort();
console.log(some_numbers)
```

In [ ]:

```
%%script node
var some_numbers = [45,63,22,9,13,2,1,64,'40'];
some_numbers.sort(function(a,b)
                  {
                      return a - b;
                  });
console.log(some_numbers)
```

Hoisting

- When a function is called, JavaScript looks for the definition **anywhere**
  - If the definition is below the call, it is said to be *hoisted* up the page
  - This only works if the name is included in the definition, not if it is assigned later

In [ ]:

```
%%script node
callMe();

function callMe(){
    console.log("I've been hoisted!")
}
```

In [ ]:

```
%%script node
callMe2();

var callMe2 = function(){
    console.log("I'll never be hoisted 😒")
}
```

Variable Number of Arguments

- JavaScript does not require any change to the declaration of a function meant to handle an unknown number of parameters
  - Declare it as you would any other function, only listing parameters that are **required**
- Inside the body of the function, the special variable `arguments` is populated with any extra arguments
  - Can be accessed like an array
    - Isn't really an array
  - `.length` property returns the number of extra arguments

In [ ]:

```
%%script node
function myJoin(separator){
    if (arguments.length < 2){
        return ""
    }

    var return_string = arguments[1]

    for(var i =2; i < arguments.length; i++ )
    {
        return_string += separator + arguments[i]
    }
    return return_string
}

console.log(myJoin(';','a','b','c'))
console.log(myJoin(';'))
```

Function Practice

- Write a function that takes in a function that is just a wrapper around a mathematical operator, and then applies that operator iteratively over the rest of the arguments

```
practice(function(a,b){return a + b;}, 1, 2, 3, 4, 5, 6, 7)
```

- should return 28

In [ ]:

```
%%script node
```

Closures

- A closure is a function that is permantly bound to the local variables that were in scope when it was created

```
function wrapper_function()
{
  var my_private_var = 10;
  return function closure()
  {
      my_private_var++
  }
}
```

Closures

- Because the local variables are bound to a function, the are "remembered" between calls to a function
  - Can be updated
- Closures provide encapsulation, but only of a single method at a time
  - Good for making event handlers

In [ ]:

```
%%script node
function my_incrementer()
{
    var counter = 0;
    return function()
    {
        counter++;
        return counter;
    }
}
var my_counter_function = my_incrementer()
console.log(my_counter_function());
console.log(my_counter_function());
console.log(counter);
```

Objects

- JavaScript is an object-oriented language, but has a relatively simply object system
- An object in JavaScript is essentially just a collection of key-value pairs
- Objects are indexed into using
  - Dot notation: `object.key`
  - Array index notation: `object['key']`
- New properties can be created through assignment

```
object.new_property = new_value
```

Creating Objects

- One of the common ways to create an object is to enumerate the key-value pairs using the following syntax

```
var my_object = {
  key1: value1,
  key2: value2,
  ...
              }
```

- Another very common way is to use `new Object()` to create an empty object, and then assign keys and values

```
var next_object = new Object();
next_object.key1 = value1;
next_object.key2 = value2;
```

In [ ]:

```
%%script node
var obj = {name:'Bryan Wilkinson','alma mater':'UMBC',
           graduation: [2009,2017]}
console.log(obj.name)
console.log(obj['alma mater'])
console.log(obj.graduation[0])
```

In [ ]:

```
%%script node
var obj = {name:'Bryan Wilkinson','alma mater':'UMBC',
           graduation: {bs:2009, phd:2017}
          };
console.log(obj.name)
console.log(obj['alma mater'])
console.log(obj.graduation.bs)
```

Custom Object Types

- JavaScript doesn't have classes in the traditional sense
- To define a new type of object, we make a constructor function and then call that function with the `new` keyword

```
function Person(name,dob,birthplace)
{
  this.name = name;
  this.dob = dob;
  this.brithplace = birthplace;
}
var person1 = new Person("Simon Bolivar","July 24 1783","Caracas")
```

- These are still just regular JavaScript Objects

In [ ]:

```
%%script node
function Car(make,model,year){
    this.make = make
    this.model = model
    this.year = year
}
var first_car = new Car("Toyota",'Camry',1998)
console.log(first_car.make)

first_car.color = "Beige"

console.log(first_car['color'])
console.log(typeof(first_car))
console.log(first_car instanceof Car)
console.log(first_car instanceof Object)
```

Object Creation Practice

- Write a constructor for a Book object that has
  - a title
  - an author
  - number of copies available

Methods

- Methods in JavaScript are keys with function values
  - Inside these functions, the keyword `this` can be used to refer to the object

In [ ]:

```
%%script node
function Car(make,model,year){
    this.make = make
    this.model = model
    this.year = year
    this.print = function(){
        console.log("I am a " + this.year + " "
                    + this.make + " " + this.model)
    }
}
var first_car = new Car("Toyota",'Camry',1998);
first_car.print();
```

Methods Practice

- Augment your Book object with a `sell` method that reduces the number of books by one
  - If there is 0 books, return an errors string "SOLD OUT"

Comparing Objects

- Objects in JavaScript are compared by reference
  - The only way two objects are equal is if they 'point' to the same object
- To compare objects based on content, a custom function must be written

In [ ]:

```
%%script node
function simpleObject(){
    this.value = "constant"
}
var obj1 = new simpleObject();
var obj2 = new simpleObject();
console.log(obj1 == obj2)
```

In [ ]:

```
%%script node
function compare_objects(obj_a,obj_b){

    for(key in obj_a){
        if(obj_a[key] != obj_b[key])
        {
            return false
        }
    }
    if(Object.keys(obj_a).length != Object.keys(obj_b).length)
    { return false;}
    return true;
}
function simpleObject(){
    this.value = "constant"
}
var obj1 = new simpleObject();
var obj2 = new simpleObject();
console.log(compare_objects(obj1,obj2));
```

Prototype Object Oriented Design

- JavaScript uses an object system known as prototyping
  - Another language with a similar system is Lua
- The prototype of an object is used when a particular key doesn't exist in the object itself
  - This allows us to do a kind of inheritance
  - If no prototype is set, the prototype is just Object

In [ ]:

```
%%script node
function Graph() {
  this.vertices = [];
  this.edges = [];
}

Graph.prototype = {
  addVertex: function(v) {
    this.vertices.push(v);
  }
};

var g = new Graph();
console.log(g);
g.addVertex('Node_1');
console.log(g)
console.log(g.prototype)
console.log(g.__proto__)
```

Object.create

- If we want to reuse a prototype over and over, we can do it manually ,as before
- The function `Object.create` takes an object, `o`, and returns a new object, `a`
  - `o` is set to be the prototype of `a`

```
parent = {key1:value1,key2:value2}
child = Object.create(parent)
```

In [ ]:

```
%%script node
var Animal = {print: function(){
    console.log(this.name + " says " + this.sound)}
        }

var Horse = Object.create(Animal)
Horse.sound = "neighhh"

var Homeboykris = Object.create(Horse)
var AmericanPharoah = Object.create(Horse)

Homeboykris.name = "Homeboykris"
AmericanPharoah.name = "American Pharoah"

Homeboykris.print();
AmericanPharoah.print();
```

ES 6 Classes

- Most developers of JavaScript come from Java/C++, etc. backgrounds
- Writing code with prototype objects can be annoying, and is error-prone if you don't take the time to really learn it
- ES6 introduces a new way to declare classes, using the `class` keyword
  - This is just syntactic sugar, its still prototypes behind the scenes
  - Uses `extends` to perform inheritance

```
class className{
       constructor(arg1,arg2){
            this.arg1 = arg1;
            this.arg2 = arg2;
                                }
       get arg1(){
                 }
       set arg1(){
                 }
 }
```

In [ ]:

```
%%script node
class Circle{
    constructor(radius){
        this.r = radius;
    }
    area() {
        return 3.14 * this.r * this.r;
    }

}

var circ1 = new Circle(2.0)
console.log(circ1.area());
circ1.r = 10;
console.log(circ1.area());
```

Document Object Model

- The document object model (DOM) is the abstraction of an HTML page's content for programmatic access
  - Developed by the W3C, not ECMA
- The DOM allows us to access part's of the web page, as well as interact with their properies
- JavScript in web browsers includes a DOM API, this may not be present in other environments
- The DOM can be viewed a tree structure, with tags nested in the HTML becoming children of their parent node in the tree

DOM DataTypes

- The DOM uses a number of objects to represent the parts of the webpage
  - document - A global object that refers to the root of the document, and can be used to modify the entire page
  - element - An HTML element, that is a tag and all its properties and contents
  - node - Anything in the Tree, including both elements and things like text and comments
  - nodeList - An array-like object, consisting of a list of nodes

Accessing the DOM

- Most functions for getting parts of the DOM are methods of the `document` object
  - `document.getElementById(id)` returns a single element object
  - `document.getElementsByTagName(name)` returns a nodeList of all elements of that tag
  - `document.getElementsByClassName(name)` returns a nodeList of all elements with that class
- Newer (more useful) methods
  - `NODE.querySelector(selector)` returns the first element matching the CSS selector
  - `NODE.querySelectorAll(selector)` returns all elements matching the CSS selector(s)

In [ ]:

```
%%html
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <div id="alone">
            <div class="inner">
                <p>
                Test
                </p>
            </div>
        </div>
        <p class="inner">
        Inner
        </p>
        <script>
            console.log("A: " + document.getElementById('alone'));
            console.log("B: " + document.getElementsByTagName('div'));
            console.log("C: " + document.getElementsByClassName('inner'));
            console.log("D: " + document.querySelector('.inner'));
            console.log("E: " + document.querySelectorAll('.inner'));
        </script>
    </body>
</html>
```

Common DOM Properties

- These properties are some of the most commonly used on DOM objects, especially elements
  - `NODE.innerHTML` - set or get the HTML contained in a node (doesn't include the node itself)
  - `NODE.outerHTML` - sets or get the HTML of a node, including the node itself
  - `NODE.id`,`NODE.className` - set or gets the id and class attributes of the node
  - `NODE.attributes` - A read only object that allows all attributes to be inspected
  - `NODE.childNodes` - A nodeList of the nodes under this one
  - `NODE.parentNodes` - The parent node of the current node
  - `NODE.textContent` - The text under this node, and all its children, with no HTML

In [ ]:

```
%%html
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <div id="an_id">

            <div class="a_class">
                <p>
                Test
                </p>
            </div>
        </div>
        <p class="a_class">
        Inner
        </p>
        <p id="my_results">

        </p>
        <script>
            var results = document.getElementById('my_results');
            results.innerHTML = document.getElementById('an_id').textContent;
        </script>
    </body>
</html>
```

In [ ]:

```
%%html
<!DOCTYPE html>
<html>
    <head>
    <style>
        #an_id1{border:2px solid black;}
    </style>
    </head>
    <body>
        <div id="an_id0">
            <div class="a_class0">
                <p>
                Test
                </p>
            </div>
        </div>
        <p class="a_class0">
        Inner
        </p>
        <p id="my_results0">
        </p>
        <script>
            var results = document.getElementById('my_results0');
            results.innerHTML = document.getElementById('an_id0').innerHTML;
        </script>
    </body>
</html>
```

In [ ]:

```
%%html
<!DOCTYPE html>
<html>
    <head>
    <style>
        #an_id1{border:2px solid black;}
    </style>
    </head>
    <body>
        <div id="an_id1">

            <div class="a_class1">
                <p>
                Test
                </p>
            </div>
        </div>
        <p class="a_class1">
        Inner
        </p>
        <p id="my_results1">
        </p>
        <script>
            var results = document.getElementById('my_results1');
            results.innerHTML = document.getElementById('an_id1').outerHTML;
        </script>
    </body>
</html>
```

In [ ]:

```
%%html
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <div id="an_id2">

            <div class="a_class2">
                <p>
                Test
                </p>
            </div>
        </div>
        <p class="a_class2">
        Inner
        </p>
        <p id="my_results2">
        </p>
        <script>
            var results = document.getElementById('my_results2');
            results.innerHTML = document.getElementById('an_id2').id;
        </script>
    </body>
</html>
```

DOM Practice

- Write the JavaScript necessary to turn all `<td>` cells in the table below with the class 'change' to be green

In [ ]:

```
%%html
<!DOCTYPE html>
<html>
    <head>
        <style>
            .green{background: green}
        </style>
    </head>
    <body>
        <p class="change">Careful</p>
        <table>
            <tr class="change">
                <td>A</td>
                <td>B</td>
                <td>C</td>
            </tr>
            <tr>
                <td class="change">1</td>
                <td>2</td>
                <td class="change">3</td>
            </tr>
            <tr>
                <td>$</td>
                <td class="change">?</td>
                <td>!</td>
            </tr>
```

Common DOM Methods

- There are many common methods that allow us to do greater manipulation of the DOM
  - `NODE.getAttribute(name)` returns the value of a specific attribute
  - `NODE.setAttribute(name,value)` sets the value of a specific attribute
  - `NODE.removeChild(node)` removes a child node when passed a *Node* object
  - `NODE.replaceChild(newNode,oldNode)` removes the `oldNode` and inserts the `newNode` in itsplace
  - `NODE.appendChild(newNode)` adds a new node as the last child of the calling node

In [ ]:

```
%%html
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <div id="an_id3">

            <div class="a_class3">
                <p>
                Test
                </p>
            </div>
        </div>
        <p class="a_class3">
        Inner
        </p>
        <p id="my_results3">
        </p>
        <script>
            document.getElementById("an_id3").removeChild(
                document.querySelector('div.a_class3'));
        </script>
    </body>
</html>
```

In [ ]:

```
%%html
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <div id="an_id4">

            <div class="a_class4">
                <p>
                Test
                </p>
            </div>
        </div>
        <p class="a_class4">
        Inner
        </p>
        <p id="outer4">
            I am outside
        </p>
        <script>
            document.getElementById("an_id4").replaceChild(
                    document.querySelector('#outer4'),
                    document.querySelector('div.a_class4'));
        </script>
    </body>
</html>
```

In [ ]:

```
%%html
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <div id="an_id5">

            <div class="a_class5">
                <p>
                Test
                </p>
            </div>
        </div>
        <p class="a_class5">
        Inner
        </p>
        <p id="outer5">
            I am outside
        </p>
        <script>
            document.getElementById("an_id5").appendChild(
                      document.createElement("div"));
        </script>
    </body>
</html>
```