JavaScript I

Introduction

- JavaScript traditionally runs in an interpreter that is part of a browsers
  - Often called a JavaScript engine
- Was originally designed to add interactive elements to HTML pages
  - First released in 1995 as part of Netscape Navigator
- Forms the backbone of modern web development
- Uses multiple paradigms, including object oriented and functional aspects

Background

Javascript is not related to Java in anyway other than trying to get some free publicity

Background (cont'd)

- Now Javascript is a standarized language that is overseen by European Computer Manufacturers Association
  - The official term for the language is ECMAScript
  - The 6th version of the standard was finalized in June, 2015
  - The 500+ page standard is available for browsing
- Javascript has moved beyond the web for use in databases and desktop programs.

JavaScript Capabilities

- Add text dynamically to an HTML page
- React to events that occur on an HTML page
- Basic validation (but not the only validation!)
- Detect browser and other data about the environment
- Asynchronous communication

JavaScript Restrictions

(On the Web)

- You cannot read or write to the file system in general
- You cannot interact with other processes on the system
- You can't keep your code private

JavaScript Engines

- Each Major browser uses their own JavaScript engine (interpreter)
    - There are differences not only in what is supported, but the speed of various functions
- The Engines
    - Spider Monkey
        - The original engine, tracing all the way back to Netscape Navigator
        - Now developed by Mozilla
        - Used in all Mozilla products, as well as Adobe Acrobat and Mongo DB

The Engines (Continued)

- V8
  - Developed by the Chromium team at Google
  - Used in Chrome, Opera, and Vivaldi browsers
  - Also used in Couchbase and node.js
- JavaScriptCore
  - Developed by the WebKit team
  - Used in Safari (known as Nitro)
- Chakra(Core)
  - Developed by Microsoft
  - Used in Internet Explorer and Edge
    - Slightly different engines in each

Placing JavaScript in a Web Page

- JavaScript is embedded in a web page using the `<script>` tags
  - You used to have to specify the `type` attribute, but HTML5 assumes JavaScript
- The JavaScript can be
  - Placed between the `script` tags
  - Stored in an external file, and specified using the `src` attribute

```
<script>
    JS GOES HERE
</script>

<script src="file_name.js"></script>
```

The `noscript` Tag

- Not all browsers will run JavaScript
  - The user may have disabled it
  - May be text-based or a screen reader
    - Although screen readers are getting more advanced
- To handle these situtations gracefully, use the `noscript` tag
  - Content in `noscript` is only displayed if running JavaScript is not possible

Developing Javascript Code

- NodeJS
- Web Browser
  - Web Console (Firefox)
  - Javascript Console (Chrome)
  - Show Error Console (Safari, after enabling developer menu)
  - Console Tool (IE 11+, Edge)

Variables & Scope

- Variables should always be declared using the keywords **var**, **let**, or **const**
  - Not necessary always, but easier than trying to remember when to use it and when not to
  - In *strict mode* is always necessary
- The scope of a variable is the function it was declared in
  - Making a new function is used to be only way to make a new scope
  - We can use the keyword **let** to define the scope inside a block (ES2015)

In [ ]:

```
var a = 1
var _a = 1
var $a = 1
//var 1a = 1
```

In [ ]:
```
var a = 5
if(true)
    {
        var b = 6
    }
a + b
```

```
In [ ]: var c = 6
        function f() {
            var d = 11
        }
        c + d
```

In [ ]:

```
var e = 5
if(true)
    {
        let g = 6
    }
e + g
```

Data Types

- Javascript is a dynamically typed language
- Javascript provides 7 data types
    - Undefined
    - Null
    - Boolean
    - String
    - Symbol
    - Number
    - Object

Number

- Only one type for both integers and floats
- Also can hold one of 3 special values
    - -Infinity
    - Infinity
    - NaN

Operators

```
In [ ]: 4 + 1
```

```
In [ ]: 4 - 1
```

```
In [ ]: 4 * 1
```

```
In [ ]: 4 / -2
In [ ]: 4 % -2
```

```
In [ ]: 4/0
```

```
In [ ]: Infinity / Infinity
```

```
In [ ]: Math.sqrt(-1)
```

```
In [ ]: 0/0
```

In [ ]:
```
var a = 1
a += 1
a
```

In [ ]:
```
var b = 20
b++
b
```

String

- Strings in Javascript can be delimited by either single or dobule quotes
- A specific character at position *i* in a string can be access through bracket notation [i]
- The concatenation operator is **+**

```
In [ ]: "abc"[0]
```

```
In [ ]: "abc" + "abc"
```

```
In [ ]: 'Someone said "this" '
```

String Methods

- `charAt(i)` allows you to index using a method rather than `[]`
- `concat(s1,s2...)` allows multiple strings to be concatenated in one call
- `indexOf(string)`/`lastIndexOf(string)` finds the first or last occurrence of the argument in the string
- `split(sep)` returns an array, the result of splitting the string on the separator passed in
- `length` is a property that holds the number of characters in a string

```
In [ ]: "abc".charAt(0)
```

```
In [ ]: "abc".concat("def","ghi",'jkl')
```

```
In [ ]: "abba".indexOf('b')
```

```
In [ ]: "abba".lastIndexOf('b')
In [ ]: "a,b,c,d".split(",")
```

```
In [ ]: var x = "a,,b,c,,d".split(",,")
        x
```

```
In [ ]: "abc".length
```

Boolean

- The boolean values in Javascript are **true** and **false**
  - 0, NaN and "" are coerced to **false**
- The operators are
  - && (and)
  - || (or)
  - ! (not)
- To test equality there are two operators
  - == Tests the value only
  - === Tests the value and the type

In [ ]:
```
(1 > 0 ) && (1 < 10)
```

In [ ]:
```
'1' == 1
```

In [ ]:
```
'1' === 1
```

In [ ]:
```
'1' != 1
```

In [ ]:
```
'1' !== 1
```

Undefined & Null

- A variable in Javascript that is uninitialized has a value of **undefined**
- **null** is used in similar situations
- Testing a variable equal to (==) **null** actually test **null** or **undefined**

In [ ]:

```
var undeclared
console.log(undeclared === undefined)
console.log(undeclared === null)
console.log(undeclared == null)
console.log(null == false)
console.log("abc"[200])
```

Arrays

- Arrays in Javascript are a special type of object
- They can be initialized by
  - listing the elements between square brackets
  - Calling the array constructor `Array()` with
    - The length of the array
    - The elements of the array
- They are indexed using []

In [ ]:
```
var arr = [1,2,3,4,5,6]
console.log(arr)
console.log(arr[0])

var arr2 = Array(10)
console.log(arr2)
console.log(arr2[0])

var arr3 = Array(10,9,8,7,6)
console.log(arr3)
console.log(arr3[0])
console.log(arr3[-1])
```

Array Methods

- `concat(a1,a2,a3)` Appends several arrays together into one array
- `join(string)` Returns a string, with each element joined by a string
- `pop/push(el)` Remove or add an element at the end of the array
- `shift/unshift(el)` Remove or add an element at the front of the array
- `reverse()` Returns array in reverse order
- `sort(function)` Returns the array, sorted by a function

In [ ]:
```
var my_array = Array(1,2,3,4)
my_array.concat([1,2,3,4],[1,2,3,4])
```

In [ ]:
```
my_array.join(",")
```

In [ ]:
```
my_array.join("...")
```

In [ ]:

```
var my_array2 = Array(1,2,3,4,5)
my_array2.pop()
console.log(my_array2)

my_array2.push("Elephant")
console.log(my_array2)
```

In [ ]:

```
var my_array3 = Array(10,9,8,7,6,5)
console.log(my_array3.shift())
console.log(my_array3)

my_array3.unshift("T-minus")
console.log(my_array3)
```

Type Coercion

- When dealing with two different data types, Javascript will prefer to attempt to cast one of the types rather than throw an error
  - This is known as type coercion
- If type coercion fails, rather throw an error, **NaN** or **undefined** are usually returned

```
In [ ]: 2 - '20'
```

```
In [ ]: 5 + Number('1')
```

```
In [ ]: 5 + '1'
In [ ]: '1' + 2
```

```
In [ ]: '5' * 20
```

```
In [ ]: '5' * '5'
In [ ]: 't' * 5
```

```
In [ ]: 't' / null
```

```
In [ ]: null == 0
```

```
In [ ]: '8' / null
```

Conditionals & Looping

- Javascript provides the following conditional statements
  - if
  - switch
- And the following looping mechanisms
  - for
  - while
  - do-while
  - for-in
  - for-of

If

- The **if** statement in Javascript is pretty straightforward

```
if (condition) {
  doSomething
}
```

- The parentheses are not necesary for a single line, but should always be used
- **if else** looks like this:

```
if (condition1){
}
else if(condition2){
}
else if(condition3){
}
else{
}
```

In [ ]:

```
var x = '0'
if(x < 0){
    console.log("Negative");
    }
//
/*
Note the triple equals

*/
else if(x === 0){
    console.log("Zero");
}
else{
    console.log("Positive");
}
```

Switch Statement

- The syntax and mechanics of the **swtich** statement borrow heavily from other languages
- Cases are marked with **case** and **default** provides a catch all case

```
switch(toTest){
  case 1:
  case 2:
      doSomething
      break
  case "A":
  case "B":
      somethingElse
      break
  case "D":
      other
      break
  default:
      final
      break
  }
```

In [ ]:

```
switch('0'){
    case -1:
        console.log("Negative")
        break
    case 0:
    //case '0':
        console.log("Zero")
        break
    default:
        console.log("Positive")
        break
}
```

Looping

- The for loop construct is similar to other languages you know

```
for(var i = 0; i < 10; i++){
}
```

- The while and do while syntax is also similar

```
var i = 0
while(i < 10){
  i++
}
```

In [ ]:
```
for(let z = 0; z < 10; z++){
    console.log(z * z)
}
console.log(z)
```

In [ ]:

```
var q = 1
while(q < 10){
    q++
}
```

For-In and For-Of

- The `for in` loop will loop over an objects keys
  - Order is not guaranteed to be maintained
- The `for of` loop is new, and iterates directly over the values of an object
  - Order is maintained

In [ ]:

```
let to_loop = ['a','b','c',1,2,3]
for (i in to_loop){
    console.log(i, to_loop[i])
}

for (j of to_loop){
    console.log(j)
}
```

Functions

- Functions in JavaScript are first class objects
- They can be passed into and returned from other functions
    - This means closures are possible
- To declare a function in JavaScript, the keyword is **function**

```
function name(param1, param2, ...){
}
```

Function Examples

In [ ]:

```
square(10)

function square(x){
    return x*x
}
```

In [ ]:

```
function counter(){
    var count = 0;
    return function(){
        count++
        return count
    }
}
```

In [ ]:

```
var c = counter()
```

In [ ]:

```
c() + 1
```