**JavaScript**

**JavaScript Overview and History**

* Java Script is in the syntax family of Java
  + It uses semicolons to end statements and left and right curly braces to start and end code blocks
* Despite its name, Java Script has more in common with C programming language than with Java
* JavaScript runs in the client browser and manipulates the Document Object Model
  + JavaScript has the capability to code your entire web application from front end to back end
* JavaScript can be used to:
  1. Augment HTML
  2. Write web application’s view and controller functions in the browser client
  3. Write web application’s database interaction functions as server code

**JavaScript in the Browser**

* JavaScript can be used to augment HTML code by writing to the Document Object Model – writing to the DOM is effectively writing to the browser client’s screen.
* There are three patterns by which you can insert JavaScript into a webpage::

1. Inline within the document
2. As part of an event in an HTML tag
3. From a file

* Insert JavaScript inline within the document:
* You can write to the DOM with the write() method of the document variable.
* Format:

<script type="text/javascript">

document.write("<p>Hello World</p>")

</script>

* Example:

<html>

<head>

<title>Hello World</title>

<meta charset="utf-8"/>

</head>

<body>

<p>One Paragraph</p>

<script type="text/javascript">

document.write("<p>Hello World</p>")

</script>

<noscript>

Your browser doesn't support or has disabled JavaScript.

</noscript>

<p>Second Paragraph</p>

</body>

</html>

* In this example, JavaScript writes “Hello World” as a paragraph tag to the screen
* You can use the JavaScript alert() function as a debugging print statement
  + alert() will pop an alert window with the text string that is inside the parentheses
  + The browser client will pause presenting the DOM until the user clicks the OK dialogue button
  + Format:

<script type="text/javascript">

alert("Here I am");

document.write("<p>Hello World</p>");

</script>

* + Example:

<html>

<head>

<title>Hello World</title>

<meta charset="utf-8"/>

</head>

<body>

<p>One Paragraph</p>

<script type="text/javascript">

alert("Here I am");

document.write("<p>Hello World</p>");

</script>

<p>Second Paragraph</p>

</body>

</html>

* + In this example, The HTML file will be presented/parsed until the <script> tag is reached, a dialog box will appear, and the page presentation will pause until the user closes the dialog box, and the page presentation/parsing will resume
  + It’s not a good idea to overuse the JavaScript alert() function – e.g., within a loop block – the browser may object and throw an error
* Insert JavaScript as part of an event in an HTML tag
* Examples of events are “onclick” and “hover”
* Format:

<a href="somePageName.htm"

onclick="*JavaScript inserted here*;">Click Me</a>

* Example:

<html>

<head>

<title>Hello World</title>

<meta charset="utf-8"/>

</head>

<body>

<p>One Paragraph</p>

<p><a href="js-01.htm"

onclick="alert('Hi'); return false;">Click Me</a></p>

<p>Third Paragraph</p>

</body>

</html>

* + In the example above, alert('Hi'); return false; is a two-line JavaScript event-based programming with a semicolon at the end of each line.
    - (The second line, return false;, tells the browser client not to follow the link in the <a> tag.)
* Insert JavaScript from a file
* format:
  + <script type="text/javascript" src="script.js"></script>
  + Usually, the loaded source file will have multiple functions defined that will be called at various locations in the HTML file, but in the example below there is a simple, one-line function defined in the .js file.
  + script.js:

document.write("<p>Hello World</p>");

* + js-04.htm:

<html>

<head>

<title>Hello World</title>

<meta charset="utf-8"/>

</head>

<body>

<p>One Paragraph</p>

<script type="text/javascript" src="script.js"></script>

<p>Third Paragraph</p>

</body>

</html>

* Debugging for Syntax Errors
* When parsing the JavaScript code, e.g., in a .js file, as the browser client executes the JavaScript code block (the code that is parsed between the <script> tags, and if it runs into a syntax error, it will stop execution of that code block and throw an error message to the console.
* Best practice: therefore, as HTML and JavaScript code is being developed, the web developer tools should be enabled and the console view should be enabled
* You can debug your JavaScript code with the console.log(“*string*”) function
  + When the console.log(“*string*”) line of code is reached and parsed, a message with the text “string” will be sent to the console.
  + Unlike an alert, with the console.log(“*string*”)mechanism, the parsing is not halted. console.log(“*string*”) is appropriate for debugging in For-loops and other iterables.

**The JavaScript Language**

* Comments:
  + single line: //….
  + multiline:

/\* …

…

…. \*/

* Statements:
  + White space and newlines do not matter.
    - The statement will be read as if they were not there – so you can break up your statement with a new line, tab, and space and it will be parsed and computed as if it was all on a single line where words are separated by a single space
  + Statements end with a semicolon
    - In some situations, the semicolon is not needed at the end of a statement, but you’re better off ending statements with semicolons
* Variable names
  + Valid character a-z, A-Z, 0-9, \_ and $
  + Variables can’t start with a number
  + Variables are case sensitive
  + By convention variables don’t begin with a $, but it is technically allowable
* Double or Single Quotes – Typically, single quotes are used in JavaScript and double quotes are used in HTML as a convention to reduce confusion.
* Character Escaping - done using the backslash character, e.g.,\n for a new line
* JavaScript has a single number type – number – no integers, floats, doubles
* Math Operators:
  + Most common: +, -, \*, /, ++, +=
  + Truncate: Math.trunc(j)
* Comparison Operators
  + Most common: =, ==, !=, <, >
  + === : is the type and value the same
* Logical operators
  + And: &&
  + Or: ||
  + Not: !
    - ex.: ! (j==k)
* String concatenation:
  + JavaScript string concatenation does implicit conversion and will convert an integer to string if you concatenate a string to a number with the “+” operator
  + e.g., suppose x = 12, then this expression: y = 'Hello ' + x + ' people'
  + will return: "Hello 12 people"
  + Because it is implicit that we are trying to create a string
* Variable typing:
  + JavaScript is a loosely typed language and does automatic type conversion when evaluating expressions. It does not fail and trace back when arithmetic is confusing.
  + e.g., the expression: x = "123" + 10
  + evaluates to a string: "12310"
  + e.g., the expression: x = ("123" \* 1) + 10
  + evaluates to a number: 133
  + e.g., the expression: x = ("fred" \* 1) + 10
  + evaluates to “Not a Number: NaN
* NaN: If a string cannot be converted to a number, you end up with “Not a Number” or “NaN”. It is a value, but it is sticky - all operations with NaN as a operand end up with NaN.
* JavaScript provides a unary typeof operator that returns the type of a variable or constant as a string.

**JavaScript Functions and Arrays**

Declaring a function in JavaScript

* Form:

<script type="text/javascript">

function functionName(arg1, arg2, ….) {

…

return value;

}

…

</script>

* Global and local variables
  + All variables, even in a function, are global by default. This is a departure from most programming languages where variables declared in a function are local.
  + To make a variable local, use the var keyword.

<script type="text/javascript">

function functionName(arg1, arg2, ….) {

var num1 = 16

…

return value;

}

…

</script>

* Arrays
  + JavaScript has arrays and associative structures. These associative structures are like Python dictionaries and are objects.
  + There are multiple ways to declare an array:
    - arr = Array()
      * arr.push('first')
        + adds the string “first” to the end of the array which is a list. This is similar to the Python append() function.
      * arr[0] = ‘first’
    - arr = Array('first', 'second')
  + Declaring an associative structure:
    - b = {"name":"john", "class":"calculus"}
    - b['name'] will return “john”

**JavaScript Control Structures**

* Conditional if statements

if ( *condition1* ) {

*codeblock1*;

} else if ( c*ondition2* ) {

*codeblock2*;

}

…

else {

*codeblockN*;

}

* Loops
  + The Definite Loop with a for statement is used to iterate through associative structures

assocStruct = {"*key1*": "*value1*",

"*key2*": "*value2*",

"*key3*": "*value3*"};

for (*keyN* in *assocStruct*) {

*codeblock*;

}

* Counted loop

for(var count=1; count<=6; count++ ) {

*codeblock*;

}

* + count<=6; is the top test
  + count++ is performed after each iteration
* Use break to completely break out of a loop and go to the next line of code
* Use continue to end the current iteration and go to the top of the loop and begin the next iterations
* When you are iterating through an array which is a linear list, you are using a counted loop. When you are iterating through an associative structure , you are using a definite loop.

**Object Oriented JavaScript**

**JavaScript Object Oriented Concepts**

* Class - a template
* Method - A defined capability of a class – AKA message
* Attribute - A defined data item in a class – AKA fields, properties
* Object - A particular instance of a class – AKA instance
* State – state of an object, the set of values of an object’s attributes
* In JavaScript, the function keyword returns a value which is the function itself

**JavaScript Object Oriented Classes**

* In JavaScript, the function keyword is used to define a class

function ClassName(arg1, …, argN) {

// Declare the class attributes

this.attributeName0 = 0;

// ”this” refers to the instance variable that is

// constructing this class object

this.attributeName1 = arg1;

…

this.attributeNameN = argN;

…

codeBlock;

// Define the class methods

this.methodName = function () {

codeBlock;

}

}

* + - Notice that the method definition invokes the keyword function, but unlike the use of function for the class name, this invocation of function is unnamed.
  + For example:

function PartyAnimal(nam) {

this.x = 0;

this.name = nam;

console.log("Built "+nam);

this.party = function () {

this.x = this.x + 1;

console.log(nam+"="+this.x);

}

}

s = new PartyAnimal("Sally");

s.party();

j = new PartyAnimal("Jim");

j.party();

s.party();

* + - the keyword “new” is used to create a new instance of a class – i.e., create an object
* You may see JavaScript code that has different syntax from the form and example above – be aware that the code may have been written with ES6
  + Among other things, JavaScript ES6 provides new syntax and features with the goal of making JavaScript code more modern and more readable