JavaScript

Introduction

- JavaScript provides the ability to create dynamic web pages
  - Adds behavior to otherwise static web content.
- The content of a web page can be dynamically changed with the scripting capability of the JavaScript interpreter.

JavaScript Primitives

- Primitives are values and have no properties or methods:
- ***number***
  - Integers (decimal, octal, hexadecimal)
    - For example: 16 (decimal), 020 (octal), 0x10 (hexadecimal)
  - Floating point (decimal followed by decimal point and the decimal digits and/or an exponent)
    - For example, 3.1412, 5e2
- ***string***
  - Enclosed by double quotes
  - For example, "Hello!," "" (an empty string)
- ***boolean***
  - True or false
- ***null***
  - Represented by null
- ***undefined***
  - A data type has not been assigned, or the variable does not exist.

Wrapper Objects

- Some of the primitive data types have corresponding wrapper objects
- Allows an object of the related type to be created
- Stores a primitive value and offers methods with which to process it
- Wrapper objects have the same name as the primitive type, but they begin with a capital letter

PRIMITIVE TYPE	WRAPPER OBJECT
boolean	Boolean
number	Number
string	String

Examples with Wrapper Objects

- The keyword **new** is used to create instances of the wrapper objects
- JavaScript readily converts (coerces) between wrapper objects and primitives

```
typeof "abc"; // "string" (primitive)
typeof String("abc"); // "string"
typeof new String("abc"); // "object"
typeof (new String("abc")).valueOf(); // "string"
```

Array Objects

- Contain methods and properties used to store data
  - Accessible by indexed keys
  - Use a zero-based indexing scheme
  - Grow or shrink dynamically by adding or removing elements

- Declaration of an array can use either an array literal or an array constructor
- Array constructors use the new array keyword and specify the array elements as parameters

```
numArray = new array(0, 1, 2, 3, 4, 5);
numArray.length // returns 6
```

- Array literals create an array and initialize the elements in the array

```
colors = ["red", "yellow", "green"];
```

Date Objects

- The Date object is a snapshot of an exact millisecond in time
  - Represented in the format YYYY-MM-DD 00:00:00 UT

- There are number of ways to provide parameters to the Date constructor
  - Example without a parameter:

```
var today = new Date(); // returns the local date and time
```

  - JavaScript automatically applies the toString() method if you attempt to display the date on a page or alert box

```
// Tues Jan 17 2012 13:15:00 GMT-8 (Pacific Standard Time)
```

Providing parameters to the Date constructor examples:

```
// create a new date from a string
var newDate = new Date("2012-1-17 13:15:30");

// create a new date instance representing
// 17 Jan 2012 00:00:00

// NOTE: The month number is zero-based
var newDate = new Date(2012, 0, 17);
```

Error Objects

- Error object instances are created when an exception is thrown
  - Properties of the error object instance contain information about the error
  - `message`: A description of the message
  - `name`: Identifies the type of error, such as `TypeError`, `RangeError`, or `URIError`

- Custom error objects can be created
- For example:

```
throw new Error("Only values 1 – 10 are permitted")
```

Summary

- JavaScript is a client-side scripting language
- Primitives are values, and have no properties or methods:
  - number, string, boolean, null, undefined
- Wrapper objects allow objects of primitive data types to be created
- Wrapper objects names begin with a capital letter to differentiate them from the primitive type
- Examples of wrapper objects are:
  - array, data, error

JavaScript Variables

Variables are:

- Containers for storing data values
- Used and modified throughout a program's execution
- Declared with the `var` keyword followed by variable name

```
var age;

var age = 54;
```

- Values can be assigned and reassigned
- Data type does not need to be declared
- Data type is assumed during assignment
- Variable type can change during program execution

```
var age;
var age = 54;
```

`var` keyword:

- If not assigned a value, it is undefined

Rules for variable names, or identifiers:

- Name must start with a letter, underscore (_), or a dollar sign ($)
- Subsequent characters can also be digits [0-9]
- Identifiers are case-sensitive

- Variable scope declared:
  - Within a function have local scope
  - Outside of a function have global scope
  - Without the var keyword have global scope and have a value of undefined

JavaScript Control (If-Else) Statements

```
var letter = "E";
if (letter != "") {
    if (letter == "A") {
        // Statements
    } else if (letter == "B") {
        // Statements
    } else if (letter == "C") {
        // Statements
    } else {
        // None of the above
    }
} else {
    // Letter is blank
}
```

- Indentation of statements is not required but helps in deciphering control statements
- JavaScript aware text editors automatically indent the control structure to make it more readable
- There is no block statement scope in JavaScript

  (Having no block statement scope: Variables declared inside one IF condition can be used outside the scope of that condition.)

JavaScript Switch Statement

```
switch (expr) {
    case LABEL1:
    // Statements
    break;
    case LABEL2:
    // Statements
break;
default:
// statements
}
```

- An alternative to the IF then ELSE control statements
- Expression parameter can evaluate any number or string value
- Program looks for a case clause with matching values and transfers control to that clause
- The **break** keyword is used to prevent the code from automatically falling into the next case clause.

JavaScript for Loop

Formal syntax definition:

```
for (initial expression; condition; update expression) {
    // Statements that execute each time the for() loop
// cycles, as long as the condition is satisfied
}
```

- Repeats a series of statements for any number of times

JavaScript while Loop

Formal syntax definition:

```
while (condition) {
    // Statements that execute each time the while() loop
// cycles, as long as the condition is satisfied
}
```

- The loop repeats while the condition remains true
- The condition evaluates to false and exits the loop

Recap

- Variables are declared using the keyword var followed by the variable name
- Variables can be initialized at the time of declaration or assigned a value later
- Variables take their data type from the value assigned and can change type during program execution
- Variables have different scopes:
  - Declared within a function have local scope
  - Declared outside of a function have global scope
  - Declared without the var keyword have global scope and have a value of undefined

JavaScript Functions

```
function add(n, m) {
    return n + m;
}
var x = add(1, 2); // returns 3
x = add(1.23, 3.45); // returns 4.68
x = add("hello", "world"); // returns "helloworld"
```

- Data types are determined by the values of the arguments that are being passed to the function
- If no specific return type is declared, then the function returns whatever type is required

JavaScripts Custom Objects

```
function Car(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year;
    this.getName = function () {
        return this.make + ' ' + this.model + ' ' + this.year;
    }
}
var c = new Car ("Meridian", "Saber GT", 2012);
alert(c.getName()); // displays "Meridian Saber GT 2012"
```

Local

Global variable associated to this instance of the Car object.

- Using the keyword `this` differentiates whether the reference is to the global or local instance of a variable

- If a prototype changes, all objects using it will automatically inherit the new properties and functions within that prototype.

JavaScript Prototypes

- Defines properties and methods for all instances of the object
- Exists for all JavaScript objects that can be constructed with the new keyword

```
function Car(make, model, year) {
    this.make = make;
    this.model = model;
    this.year = year;
}
```

- Objects inherit all properties and methods from their prototype
- Prototype properties and methods can be changed in one call

- Adding new properties/methods to objects requires modifying the object's code

```
function Car(make, model, year, color) {
this.make = make;
this.model = model;
this.year = year;
this.color = color;
}
```

```
Car.color = "Red"
```

- Adding new properties/methods to prototypes can be done with one call

```
Car.prototype.color = "Red"
```

Self Executing Fucntions

- Self-executing or auto-invocation functions:
  - Start running immediately after they have been declared
  - Functions and variables are isolated from the rest of the script
  - Can be an anonymous (unnamed) function

```
(function () {
// statements
})();
```

- Used to initialize data or to declare DOM elements on the page

Of self executing functions

Recap

- A function is a block of code which can be called from any point in the script
- Functions can take arguments and return results
- Prototypes allow you to define properties and methods for all instances of a specific object
- Prototypes exist for all objects that can be created with the `new` keyword
- New function can be added to an object by modifying the prototype for that object
- Self-executing functions start running immediately after they have been declared
- The functions and variables are isolated from the rest of the script

Introduction To APIs

- An API is a way for two applications to communicate with each other.
  - It delivers your request to another device, such as a database, and returns the response back to you.
- Restaurant Analogy
  - Imagine you are sitting in a restaurant and have selected your order from the menu. Your waiter communicates your order to the kitchen and returns the food back to you. if you order a food item which does not exist on the menu, the waiter will inform you that it is an invalid choice.
    - Waiter → API
      - As it communicates a request from one device to another
    - Menu → API documentation
      - Each API has documentation that outlines the requests you are allowed to make, and the type of response you should expect to recieve
- JavaScript API
  - Uses JavaScript scripting to dynamically access and modify content.

Introduction To APIs

The common APIs used in JavaScript applications are:

Document	Element	Window
document.getElementById(id)	element.innerHTML	window.open
document.getElementsByTagName(name)	element.style	window.onload
document.createElement(name)	element.setAttribute	window.scrollTo
parentNode.appendChild(node)	element.getAttribute	

Roadmap

- How to retrieve a reference node using document.getElementById and document.getElementsByTagName
- How to create an element using document.createElement and place it using insertBefore, appendChild, or replaceChild
- How to modify elements using element.innerHTML, element.style and element.setAttribute
- How to manage a window object using functions that include window.open and window.dump("message")

1. Retrieving a Node Reference

- Use `document.getElementById` to:
  - Pass the id value as an argument
  - Return one specific HTML or XML element that is based on the id attribute in the node

- document.getElementsByTagName(tagName):
  - Retrieves a NodeList of elements with a specified tag name
  - Nodelist contains an array of elements in the document
  - tagName parameter is the literal name of the HTML tag

Example
If you run the function getElementsByTagName with a “p” as a parameter argument, a NodeList of all the paragraphs in the document is returned

2. Creating Nodes

`document.createElement(tagName)`

- Creates a new element with the namespace of the current document
- Functions can be used to place the element in the appropriate location

Example
insertBefore, appendChild, or replaceChild function can be used to add the newly created element into the document

```
<head>
    <script>
        function addPara() {
            var newPara = document.createElement("p");
            var newText = document.createTextNode("Hello
World!");
            newPara.appendChild(newText);
            document.body.appendChild(newPara);
        }
    </script>
</head>
<body onload="addPara()">
</body>
```

- ‘new_para’ element is being created which includes a text node with the string “Hello world!”
- The text node is then appended as a child of the paragraph element.
- Finally, the entire paragraph with text is appended as a child node at the end of the body node of the HTML page.

3. Modifying an Element

`element.innerHTML`

- Retrieves or sets the contents within an HTML element
- Returns all child elements as a text string
- Allows content change of an HTML element
- Removes all current child elements by setting value to string
- Browser parses the string and sets the contents of the HTML element

- `element.style`
  - Retrieves or sets the inline CSS style for a particular element
  - Overrides setting from a CSS style sheet with one specific style

  Set the style in JavaScript is with the forma:
  `element.style.propertyName` = value

- For example, `element.style.color="blue";`

```
<div id="div1" style="color: blue">
</div>
<script>
    var div1 = document.getElementById("div1");
    div1.style.color = "red";
</script>
```

- Using the `element.setAttribute('style', ...)` wipes out all previously set inline CSS styles

3a. Modifying Attributes

Methods	
element.setAttribute(attrName, attrValue)	• Dynamically modifies the attribute of an element • Example: document.getElementById("theImage").setAttribute("src", "another.gif");
element.removeAttribute(attrName)	Removes an attribute from an element
element.getAttribute(attrName)	Retrieves the value of the specified attribute in the element

4. Window Object

```
window.open(url, name, [features, replace])
```

- Returns a reference to a new window object for the web browser

Parameters	
URL	Indicates the location of the web page to be displayed
Name	Specifies the name of the window
Features	Specifies the features of the window, such as its placement and dimensions
Replace	Optional Boolean value

Methods	
window.onload	Used to start a function after the page is loaded
window.dump("message")	Writes a string into the console for the web browser and is less intrusive
window.scrollTo(x-value, y-value)	Scrolls the web browser to a particular set of coordinates

REST Architecture

- REST : Representational State Transfer
- Most JavaScript APIs follow the REST architectural style. These are referred to as RESTful APIs, and follow the CRUD paradigm - four basic functionalities needed when communicating between services and with a database
    - Create
    - Read
    - Update
    - Delete
- In a REST environment, these CRUD operations are often aliased as follows:
    - Create → POST
    - Read → GET
    - Update → PUT
    - Delete → DELETE

- Example: Imagine an API which communicates with a banking service to process online payments

Method	URL	Description
POST	api/customer	• Create a new banking customer • Depending on the specifications of the API, this may include options to provide data for this customer, such as a name or credit card details. • It may also automatically generate new information upon creation, such as an id.
GET	api/customers/{id}	• Retrieve the information of a customer • The API assumes a unique id for each customer, that is used in the URL to specify which customer's information you are searching. • A response for this request can come in many different formats, such as JSON or XML, depending on the API.

PUT	api/customers/{id}	• Update information for a specific customer
		• This will overwrite the current data with new data.
		• Some APIs allow you to include a "body" in which you can specify a load of data to be sent with the request.
		• Example: `Update a customer's first and last name`
		{
		"first_name": "Thomas",
		"last_name": "Watson"
		}

DELETE	api/customers/{id}	• Delete a banking customer

Examples of JavaScript APIs

- **Document Object Model (DOM) API**
  - One of the most basic JavaScript APIs
  - It connects web pages to scripts by representing the **structure of a document** (e.g. an HTML web page) in memory, making it accessible for modfication as required.
- **XMLHttpRequest (XHR)**
  - Allows you to retrieve data without refreshing the entire page.
  - This is important when you to want to update only a part of a page without disrupting what a user is currently doing on the page.

Client-Side JavaScript : with HTML

- A client-side script is a program that:
  - Accompanies an HTML document or may be embedded directly in it
  - Runs when the document loads, a link is activated, or a button is clicked
- HTML support is independent of the scripting language

- Scripts offer authors a means to extend HTML documents in highly interactive ways, such as:
  - Running after an HTML document loads
  - Validating forms and processing input as it is typed
  - Being triggered by events that affect a document
  - Creating graphical and other document elements in an HTML page dynamically

<script> Tag Examples

- Example 1: Include a script directly inside the HTML document

```
<script>
    // JavaScript code
</script>
```

Used for short scripts

- Example 2: Use src attribute to point to an external script file

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

Used for importing JavaScript libraries for complex interactions or using the same script across several HTML documents

When Scripting Is Not Available

- The <noscript> tag is used to handle situations where scripts have been disabled or a certain browser doesn't support them

```
<script>
    // JavaScript code to retrieve data
</script>
<noscript>
    <p>Access the data from:</p>
    <a href="http://www.someplace.com/data"></a>
</noscript>
```

- The unsupported browser runs the section of code within the <noscript> tag

Scripts That are Tied To Intrinsic Events

- Scripts are executed every time a specific event occurs on a page
- Calling a function when an event occurs is called event binding

Event	Description
onload = script	Occurs when the user agent finishes loading a window
onclick = script	Occurs when the pointing device button is clicked over an element
onmouseover = script	Occurs when the pointing device is moved onto an element
onfocus = script	Occurs when an element receives focus
onkeyup = script	Occurs when a key is released over an element
onsubmit = script	Occurs when a form is submitted
onselect = script	Occurs when a user selects some text in a text field

Example

```
<button type="button" onclick="showAnswers()">Show Solution
    <script>
        function showAnswers() {
            // Code
            alert("A")
        }
    </script>
</button>
```

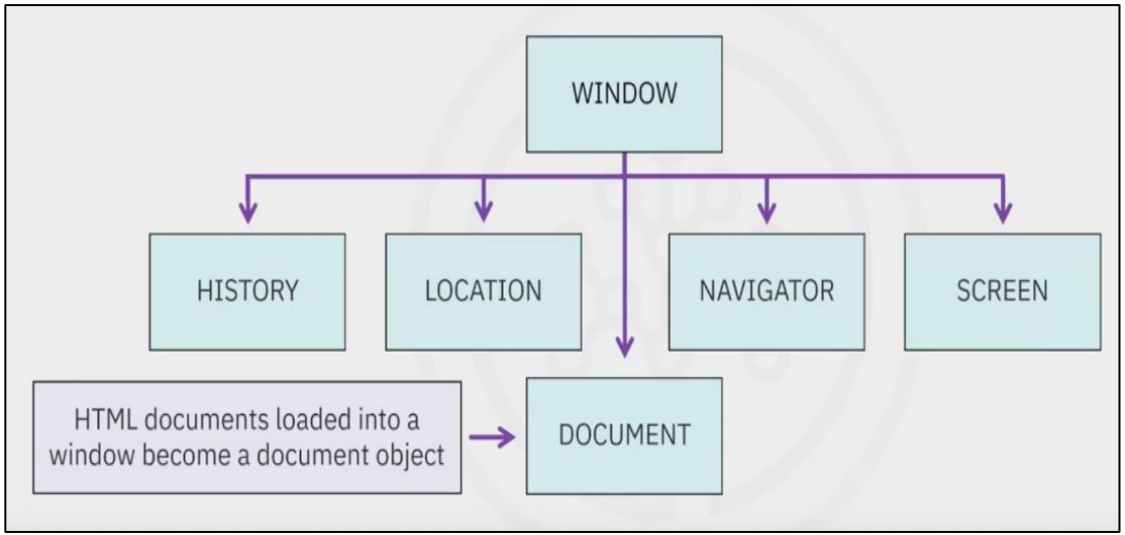
Recap

- A client-side script is a program that accompanies an HTML document
- Scripts enable developers to incorporate more interactive elements
- Script tag can include a script within an HTML document or call a script from an external file
- The <noscript> tag provides an alternative when scripting is disabled
- Scripts are bound to events so that they can run automatically

Client-Side JavaScript : with DOM

- The DOM Programming Interface
  - It is a programming interface between **HTML/XHTML and JavaScript**.
  - The Document Object Model (DOM) is a browser-based interface for applications and scripts to dynamically access and update the content, structure, and style of documents.

Basic DOM Models for Browsers



Contd...

- **Window object**
  - Present at the top of the DOM hierarchy and automatically created when the browser loads a page
  - Serves as the global object and everything in the DOM takes place in a window.
  - Can access the window object properties and functions from your JavaScript code.
- **History object**
  - Keeps internal details about the recent history of pages in the browser.
  - Has methods for letting you simulate clicking the back or forward buttons in a browser.
- **Location object**
  - Contains information about the URL of a page.
- **Navigator object**
  - Representation of the client browser
  - There is no standard that applies to the navigator object, so the property values returned when running queries on the navigator object are not consistent across browsers.
- **Screen object**
  - Used to derive information about a user's screen, such as the dimensions of the display screen.
- **Document object**
  - Provides access to all HTML elements within a page.
  - Each HTML document that gets loaded into a window becomes a document object.

Contd...

Client-side window object

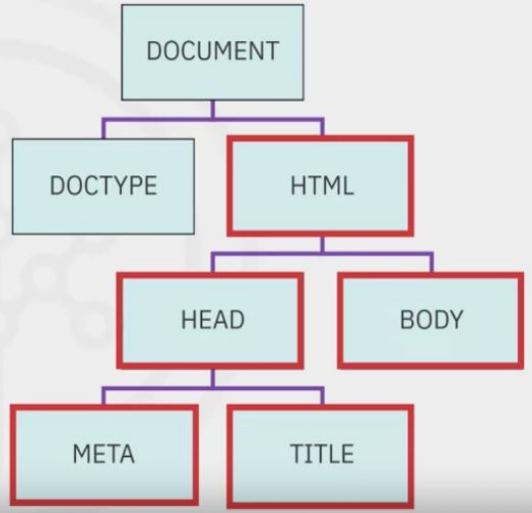
- Window object dialog boxes:

WINDOW OBJECTS	DISPLAY	CODE
`window.alert`	Plain alert box	alert("message");
`window.confirm`	Confirmation OK/Cancel dialog	confirm("message");
`window.prompt`	Text-entry prompt	prompt("message", "defaultReply");

- Can omit the `window` prefix from object references

HTML Document Object Diagram

```
<!DOCTYPE html>
<html>
    <head>
        <meta …>
        <title>Page Title</title>
    </head>
    <body>
    </body>
</html>
```

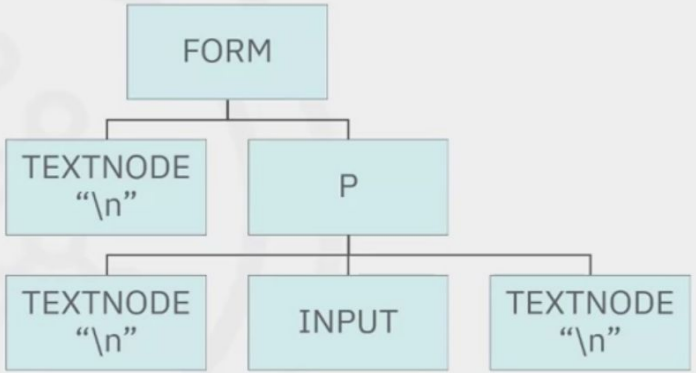


Contd...

- This figure shows the object model for a simple HTML document. Notice how the object hierarchy matches the HTML containment hierarchy on the left.
- The branches of the tree structure are termed nodes.
    - There are **two** types of nodes in the W3C DOM:
        - **Element nodes**
            - All HTML tags (html, head, meta, title, and body) are element nodes.
        - **Text nodes**
            - Nodes that contain actual text that go between an element start tag and end tag

DOM Level 2 Tree for a “form”

```
<!DOCTYPE html>
<html>
    <head>
    </head>
    <body>
        <form>
            <p>
            <input type=”button”
                onclick=”checkpoint1()”
                value=”Run checkpoint
                    question1”>
            </p>
        </form>
    </body>
</html>
```



Contd...

- The figure shows the DOM level 2 tree for the FORM portion of the document.
- The line feeds between elements are text nodes and are part of the DOM level 2 tree.
    - Includes a line feed text node before the paragraph and input elements.
- The input element includes a text node that contains all the text that follows the input tag.
    - An additional line feed text node follows the input element.
- The DOM level 0 for the form portion of the document would have only the form, p, and input boxes.

Recap

- DOM is the programming interface between HTML or XHTML and JavaScript
  - Each DOM levels provide a detailed set of features
  - Different browsers have different levels of compatibility with the DOM standard
- DOM for browsers is a hierarchy that includes objects, which perform different functions. For example, window, location, screen, and document objects
- DOM levels define object types that assists in building various documents

JavaScript DOM Objects

DOM level 2 nodes that are applicable to HTML documents:

NODE TYPE (TEXT)	INTEGER VALUE	NODE NAME	NODE VALUE	DESCRIPTION
Element	1	Tag name	null	Any HTML tag
Attribute	2	Attribute name	Attribute value	A name-value pair
Text	3	#text	Text content	Text that is contained by the element
Comment	8	#comment	Text comment	HTML comment
Document	9	#document	null	document object
Document Type	10	DOCTYPE	null	DTD specification
Fragment	11	#document fragment	null	Nodes outside the document

Exmaple

- element:

```
<div id="div123">This is DIV example</div>
```

  - Attribute name is “id” and value is “div123”

- <p> element:

```
<p>This is a paragraph example</p>
```

  - Node value is the text string

Accessing Document Elements

- When the document is loaded
  - The browser creates arrays for forms, images, anchors, links, applets, and embeds.
  - It then places all the objects of each type into these arrays.
    - The arrays are indexed as they occur in the source document.
      - The first index of each array starts at zero. For example document.images[0]
- Example

Element field1 can be referenced by:

- `document.forms[0].elements[0]`
- `document.forms["form1"]`
  `.elements["field1"]`
- `document.form1.field1`

Object Naming

The id attribute:

- Identifies an element in a document
- Refers to the element with a name that matches the value of the id attribute

To Assign a scriptable reference name to an object with the id attribute:

- The id must be a unique name in the document
- The name must appear in quotation marks when assigned to the id attribute
- The name must not start with a numeric digit

Function used to return a node object that matches the id value:

```
<div id="grid_container2">

var aDiv = document.getElementById("grid_container2")
if (aDiv) {
    // do something with the element
…
```

Tips: JavaScript

- Use IDEs to code: VS Code, Webstorm
- Use linters like XO
  - Linters are tools that analyze source code to flag programming errors, bugs, stylistic errors, and suspicious constructs.
  - **Uses**:
    - Helps developers catch errors and enforce coding standards early in the development process.
    - Identifies potential bugs, such as syntax errors, unused variables, or missing semicolons.
    - Enforce coding conventions, ensuring consistency across a codebase.
    - Helps in identifying security vulnerabilities and performance issues.
- Use Typescript
  - TypeScript is an open-source programming language developed by Microsoft.
  - It is a superset of JavaScript that adds static typing to the language.
    - Static typing is a programming language feature that requires variables, functions, and expressions to have their types explicitly declared at compile time.
  - **Uses**:
    - Allows developers to catch type-related errors during development.
- Avoid declaring global variables and functions as it can slow down your program since they are not deleted until the window is closed.
- Scripts that include same variable and run after your code will override your variables.
  - In order to reduce page load times, place your js at the bottom of your page, especially if the purpose of js is to add functionality after an event occurs.

Module Summary

- JavaScript is a scripting language that enables developers to add dynamic content to webpages.
- JavaScript variables are declared using the var keyword and take their type from the value assigned.
- Program execution is controlled by statements like If…Then…Else, Switch, For loops, and While loops.
- JavaScript uses blocks of code, called functions, that can be called from anywhere in the script.
- New methods and properties can be added to an object by modifying the prototype for that object.
- Prototypes allow you to define properties and methods for all instances of a specific object.
- Client-side scripts are programs that accompany HTML documents and are used by developers to incorporate more interactive elements.
- The script tag can incorporate a script within an HTML document or call a script from an external file.
- The Document Object Model (DOM) is the programming interface between HTML or XHTML and JavaScript.
- Developers can access HTML DOM elements from JavaScript scripts using the correct DOM notation.
- APIs are often used to access HTML DOM elements in web pages.

Browser Console Guide

- https://author-ide.skills.network/render?token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJtZF9pbnN0cnVjdGlvbnNfdXJsIjoiaHR0cHM6Ly9jZi1jb3Vyc2VzLWRhdGEuczMudXMuY2xvdWQtb2JqZWN0LXN0b3JhZ2UuYXBwZG9tYWluLmNsb3VkL0lCTURldmVsb3BlclNraWxsc05ldHdvcmstQ0QwMTAxRU4tU2tpbGxzTmV0d29yay9sYWJzL1RoZWlhJTIwTGFicy8wMyUyMC0lMjBKYXZhc2NyaXB0L0pTQmFzaWNzUHJhY3RpY2UubWQiLCJ0b29sX3R5cGUiOiJpbnN0cnVjdGlvbmFsLWxhYiIsImFkbWluIjpmYWxzZSwiaWF0IjoxNzAwNjcwMTE0fQ.oP_uAdRMWTpJtG0sGcT20V60ybz3z4qaPwR_FrE8TpU

Practice Problems

- What would the alert be, when the following code is executed?

```
var a = new String(“Hello”);
var b = “Hello”;
if (a ===b){
alert(“Same”);
}else{
alert(“Different”);
}
```

  A. Same
  B. Different
  C. It would not give any alert as it is an error
  D. None of the above

- Ans: B
  - In JavaScript, when comparing objects, such as strings created with the `new String()` constructor and string literals, using the strict equality operator (`===`), they are considered equal only if they reference the same object in memory.