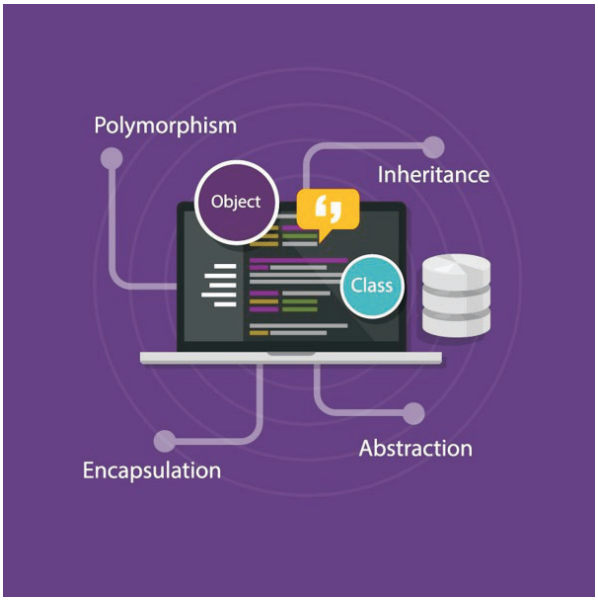
JavaScript OOP and DOM

JAVASCRIPT is a complex, powerful language that supports many of the features of object-oriented programming (OOP). It allows us to create custom objects, along with providing a rich library of built-in objects. Additionally, it exposes the content of a webpage as a set of objects referred to as the document object model (DOM). Working with the DOM allows for the modification of webpage content and styles using JavaScript code.



Objective:

Create and work with built-in JavaScript objects and understand the role of the DOM in creating dynamic webpages.

Key Terms:

attributes/properties
black box
composition relationship
concatenates
constructor function
document object model (DOM)
encapsulation
functional/procedural programming
instantiation
methods
new
NodeList
object
object composition
object-oriented programming (OOP)
string
this
Unified Modeling Language™ (UML)
UML class diagram

Understanding JavaScript OOP and DOM

Program code development is a structured process with a set of predetermined steps. As the number of programs increases, code reuse is a way to decrease the time required to code and develop programs. The most common technique for sharing code is to place commonly used functions in a ".js" file that can be used by multiple Web documents. **Functional/procedural programming** is a method of code reuse that depends on reusing functions. User-written

functions can become complex if they take in multiple parameters. Using complex functions written by others can be a daunting experience.

CREATING OBJECTS IN JAVASCRIPT

You need to know various steps to create objects in JavaScript.

Object-Oriented Programming (OOP)

Object-oriented programming (OOP) is a type of software design that views data as the most important aspect of a programming solution and focuses on data rather than upon the functions. It is a programming approach to software problems from a different perspective than functional/procedural programming. In OOP, an object is used as a repository of data, and functions manipulate the data maintained inside the object. Currently, OOP is viewed as being more logical and efficient for application development than functional programming.

An **object** is the basic entity created in an OOP environment. An object maintains data or **attributes/properties**. An object also contains **methods** (functions) that work upon the class attributes. **Encapsulation** is the notion of creating a packaged entity that holds data and the procedures that work on it; encapsulation is the hallmark (trademark) of an OOP system.

Since an object maintains data and functions that work with the data, an object is considered a "black box." A **black box** is an object in which the user feeds in input and then receives output through a return value. Once an object definition has been created, it can be used multiple times with the assurance that the data and functions work correctly with one another (see the example below).

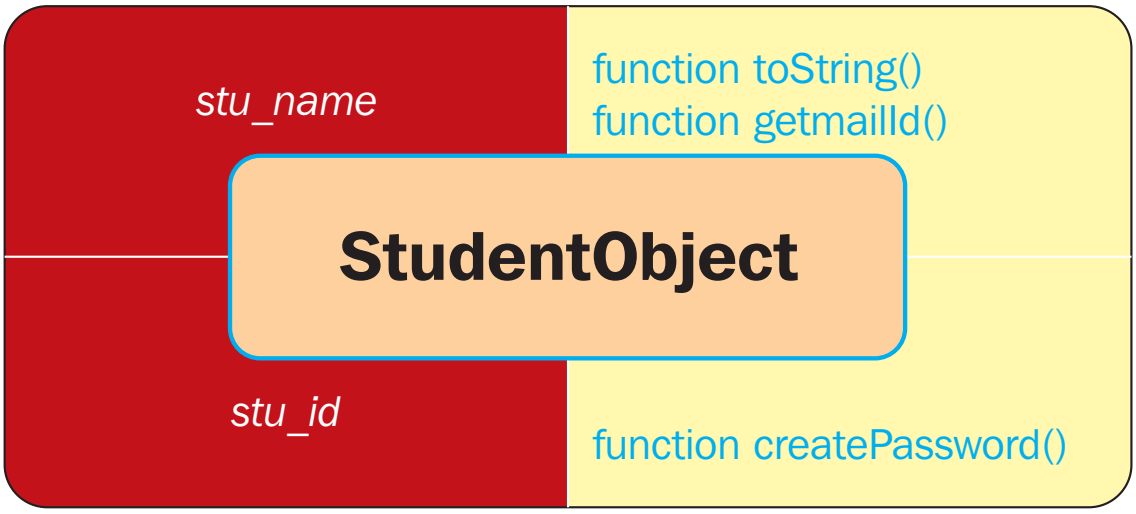


FIGURE 1. Here is an object definition for an object called StudentObject that maintains two class attributes: stu_name and stu_id. The object also contains three functions. None of these functions take in parameters.

Unified Modeling Language (UML)

Developing programming code is a complex process that starts with designing the code. Once a design is finalized, it is converted into programming statements. Once code has been written, it is difficult to implement changes. Therefore, it is imperative that the design process be robust—a process where all aspects of the application are taken into account before writing programming code.

A rigorous program design process requires adequate tools. Currently, **Unified Modeling Language™ (UML)** is the standard tool for software design processes based on OOP methodology. It is based upon the work of three computer scientists: James Rumbaugh, Grady Booch, and Ivar Jacobsen.

UML 2, the current version of UML specifications, contains two categories of diagrams: diagrams that depict the structure of a system and those that depict the behavior of the system. This lesson discusses only one diagram: the **UML class diagram**, which is an illustration that depicts the object structure.

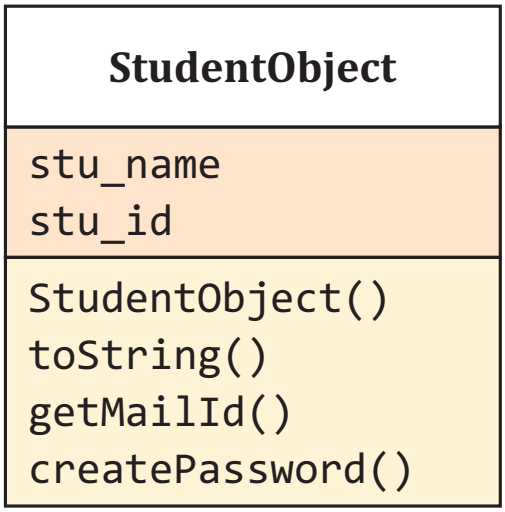


FIGURE 2. Here is an UML diagram for the StudentObject class. The first row contains the text "StudentObject" or the name of the object. The second row lists the object attributes, and the third row contains function names, including the constructor function. It is typical to list the class constructor as the first function in a class UML diagram.

A class diagram represents an object's definition in a table with three rows. The first row holds the name of the class. The second row displays the object's attributes. The third and last row in the class diagram lists all the methods of the object.

A class usually contains a special function called the **constructor function**, which is a task (function) that automatically executes when an object variable is created in a program. The name of this function is the same as the name of the class. The process of creating an object variable is **instantiation**.

Languages, such as C++ and Java, use a reserved word called "class" to create an object definition. This reserved word is not used in JavaScript. However, it is common practice to use the words "object" and "class" interchangeably since that is the case in Java and C++. (NOTE: This notion of using "object" and "class" to mean "object" in an OOP environment exists in JavaScript environments and in literature.)

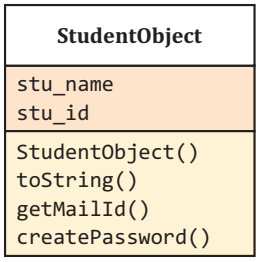
Creating an Object in JavaScript

After an object definition is created, object variables can be created or instantiated. There are multiple methods for instantiating object variables in JavaScript. We will discuss three methods: Constructor, literal syntax, and factory functions.

Method 1

Constructor function: In this method, the class constructor function is developed. Inside the class constructor, values are assigned to class private variables, and class functions are coded. Within the constructor function, the keyword "**this**" is used as a prefix to denote class variables. This is the method that will be used in this lesson to create object definitions. One class function coded for most objects is a function called a toString function. This function returns all the class private variables as a single string value.

Class UML Diagram



Class Code

```
1  <script>
2     function StudentObject()
3     {
4        this.stu_name = "-default-"
5        this.stu_id = 99
6
7        this.toString = function ()
8        {
9           return "name=" + this.stu_name + ", id=" + this.stu_id
10       }
11       this.getMailId = function()
12       {
13          return this.stu_name + this.stu_id
14       }
15
16       this.createPassword = function()
17       {
18          return "default_password"
19       }
20    }
21
22    var myStudent = new StudentObject()
23    alert(myStudent.toString())
24    myStudent.stu_name = "John"
25    myStudent.stu_id = 1234
26    alert(myStudent.toString())
27
28    alert("password=" + myStudent.createPassword() +
29       "\nemail id=" + myStudent.getMailId())
30
31 </script>
```

Output Produced by Code

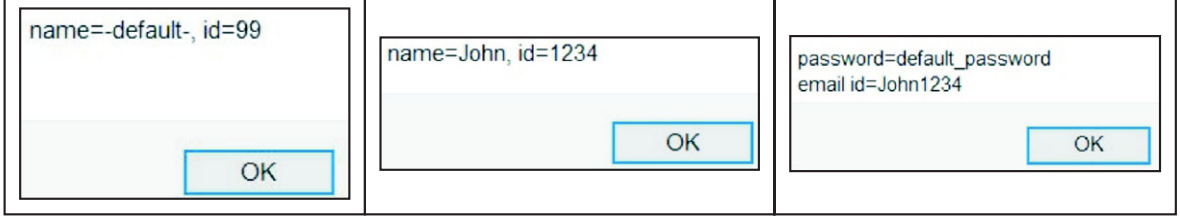


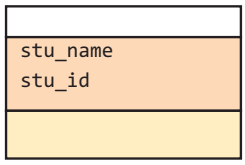
FIGURE 3. This code shows the class UML diagram for the StudentObject class, along with the code for the class coded inside the class constructor function. Also, note the statements to instantiate the class definition and the invocation of class methods. Code highlighted in light green constitutes the class definition. The definition starts with a function called StudentObject. Inside the function, the keyword "this" is used to indicate that it contains a class definition. The "this" keyword is used to reference class data. Lines 4 and 5 place values in class attributes called stu_name and stu_id. Line 7 creates a class function called toString. It is a class function since the name of the function is prefixed by the "this" keyword. There are function definitions for the class functions getMailId and createPassword.

FIGURE 3—Interpretation continued: Code not included in the green box is considered to be "outside" the class. In line 22, an object variable is instantiated using the "new" keyword. The **new** keyword is a keyword that creates a copy of the object in main memory and assigns it the name "myStudent." Line 23 invokes the toString function inside the object variable, myStudent. The "." operator is used to retrieve a function or attribute of an object variable. The "." operator is used again in lines 24 and 25 to place values in class attributes. Remaining lines in the code invoke various class functions. The object definition can be used to instantiate other variables of the same object type.

Method 2

Literal syntax: In this method of defining objects, the object variable is created using the "var" keyword. Instead of assigning a single value to the variable, a set of key-value-pairs is assigned to the variable. The key value represents the name of the property, and the value is the literal placed in that property. This method does not make use of the "new" keyword. Additionally, the object is not given a name; only the instance is given a name.

Class UML Diagram	Class Code



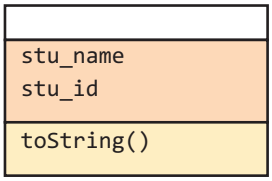

```
1  var myStudent = {
2     stu_name: "--default--",
3     stu_id: 99
4     }
5  alert(myStudent.stu_name + ", " + myStudent.stu_id)
6
7  myStudent.stu_name = "John"
8  myStudent.stu_id = 1234
9  alert(myStudent.stu_name + ", " + myStudent.stu_id)
```

FIGURE 4. Here is an UML diagram for a class, along with the code for the object. In this technique, the name of the object is not specified. A variable is set up in line 1 with the name myStudent. In lines 2 and 3, attributes and values for the attributes are specified. The class UML diagram shown indicates that the class is set up without a name and without any functions. Lines 5 through 9 show that the "." operator may be used across the myStudent variable, indicating that it is an object and that it supports properties.

Variations of the literal syntax allow for definitions of functions.

When using literal syntax to create objects, the object definition cannot be used to instantiate other object variables. The only object variable being defined is the one whose name is specified after the "var" keyword.

Class UML Diagram	Class Code




```
1  var myStudent = {
2     stu_name:"--default--",
3     stu_id:99,
4
5     toString: function()
6     {
7        return this.stu_name + ", " +
8        this.stu_id
9     }
10 }
11
12 alert(myStudent.stu_name + ", " + myStudent.stu_id)
13
14 myStudent.stu_name = "John"
15 myStudent.stu_id = 1234
16 alert(myStudent.toString())
```

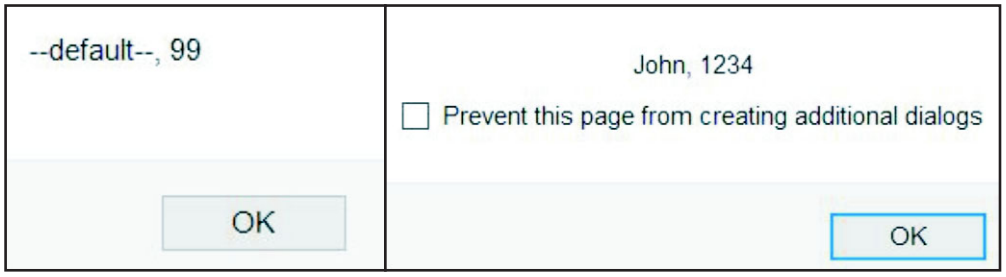
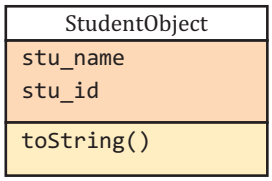


FIGURE 5. In this variation of the literal syntax, the toString function has been defined inside the var statement. It may now be accessed via the "." operator in line 16.

Method 3

Factory functions: In this method, the principles of the constructor method and the literal method are combined. Object definitions are set up using the "var" keyword, but the identifier after the "var" keyword denotes an object name, not an object instance.

Class UML Diagram	Class Code




```
1   var StudentObject = function()
2      {
3         var aStudent = new Object()
4         aStudent.stu_name = "--default--"
5         aStudent.stu_id = 99
6
7         aStudent.toString =  function()
8         {
9            return this.stu_name + ", " +
10              this.stu_id
11        }
12        return aStudent
13     }
14  myStudent = new StudentObject()
15  alert(myStudent.stu_name + ", " + myStudent.stu_id)
16
17  myStudent.stu_name = "John"
18  myStudent.stu_id = 1234
19  alert(myStudent.toString())
```

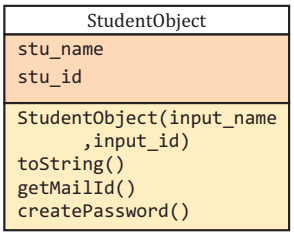
FIGURE 6. The "var" keyword is used in line 1 to create an object definition for StudentObject by creating a function definition that serves as the constructor function for the class. Inside the constructor function, an object variable of type "Object" is declared in line 3. "Object" is a built-in object in JavaScript and can be used to create an object variable. This variable's attributes are set up in lines 4 and 5, and an object function is set up in lines 8 through 11. After the object definition has been created, the "Object" variable is returned from the function. Lines 14 through 19 create an object variable of type StudentObject and use the "." operator to access the object variable's attributes and functions.

Constructors That Take In Parameters

The constructor function and factory function techniques for creating object definitions use constructor functions that are automatically invoked when an object is instantiated. In the examples used to illustrate the two techniques, the constructor function was coded with no parameters.

However, a constructor function may take in parameters used to update the class's attributes. If the constructor is coded to take in parameters, it must be invoked with the corresponding number of arguments. The constructor function technique is used in an example to illustrate a constructor function that can take in parameters.

Class UML Diagram



Class Code

```
1  function StudentObject(input_name, input_id)
2  {
3     this.stu_name = input_name
4     this.stu_id = input_id
5
6     this.toString = function ()
7     {
8        return "name=" + this.stu_name + ", id=" + this.stu_id
9     }
10 }
11
12 var myStudent = new StudentObject("Jane", 2345)
13 alert(myStudent.toString())
14
15 myStudent.stu_name = "John"
16 myStudent.stu_id = 1234
17 alert(myStudent.toString())
```

Output

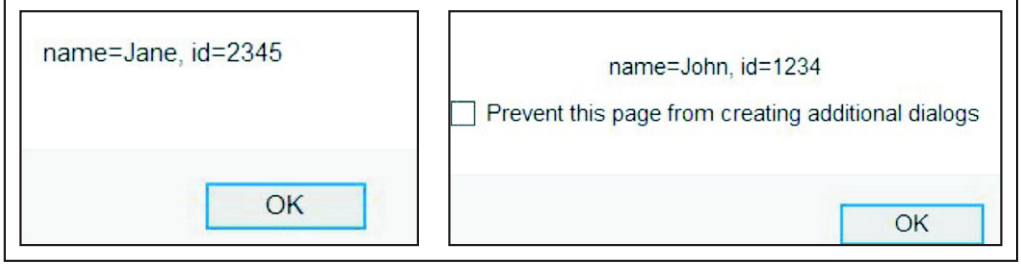


FIGURE 7. Here is the class definition for StudentObject where the constructor function takes in two parameters, as shown in line 1. The name of the parameter variables is prefixed with "input" to indicate that they are being sent into the function. The values in the parameter variables are placed in the class attributes in lines 3 and 4. Since the constructor requires two parameters, the statement that creates an object variable in line 12 provides two arguments to the object instantiation statement.

Creating constructor functions that take in parameters can speed up the process of populating class variables. Multiple class variables can have values assigned to them using parameterized constructor functions, thereby alleviating the need to code assignment statements for each and every class attribute.

USE OBJECTS IN JAVASCRIPT

The Object Definition Process

The process of creating an object definition involves multiple steps. When developing an object definition, it must be at the most elemental level. For example, rather than create an object that maintains student and school information within the same class, it is more appropriate to develop a separate class to hold student information and a separate class to hold school information.

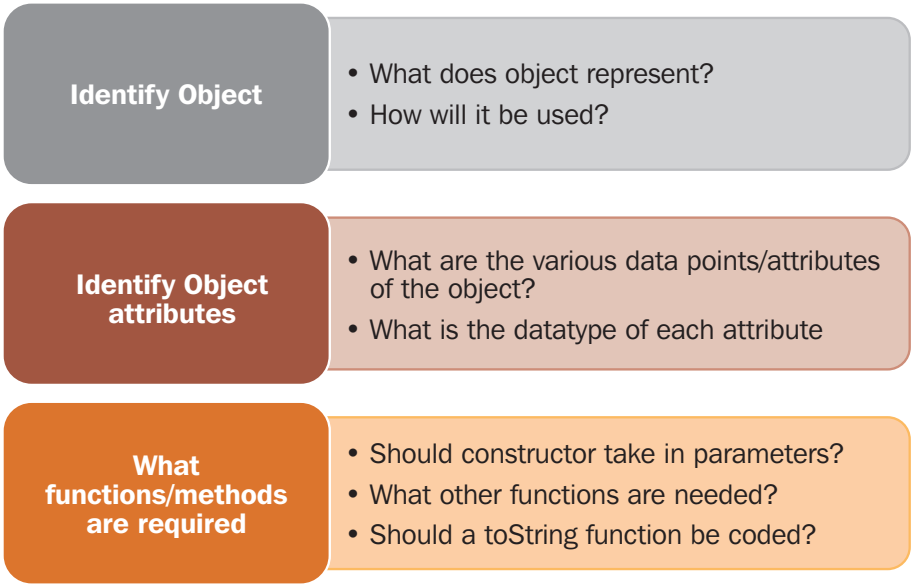


FIGURE 8. At this stage of the design process, all attributes that will be used in the application system for this specific entity must be identified, along with the appropriate data type. Even though JavaScript is a loosely typed language, where variables are not tied to a specific data type, good object design dictates that the datatype of the attribute be taken into consideration when an object is designed.

Object composition is the creation of objects from other objects. Object composition is shown in a class diagram as a line connecting the object with the object that contains it to represent a **composition relationship**. The line ends with a filled diamond across the object that holds an instance of the composed object.

The design decision to place address-related information in a separate object instead of maintaining address information in the StudentObject and the TeacherObject is made so code that converts address fields into a mailing address format is written in the AddressObject object definition and used in all classes that need to hold address information.

Composition is one way by which object relationships can be built. Aside from composition, objects may implement other relationships (e.g., prototypal inheritance, association, and interfaces). [NOTE: This lesson does not address prototypal, inheritance, association, or interface relationships.]

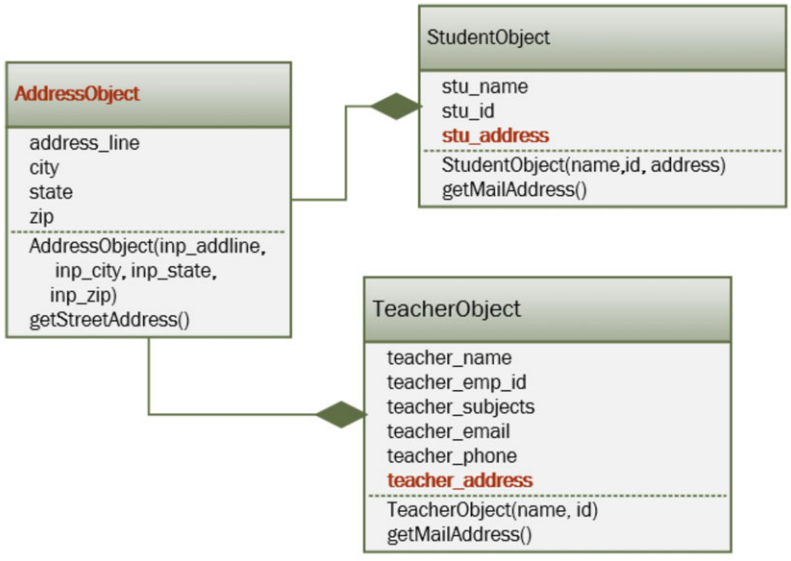
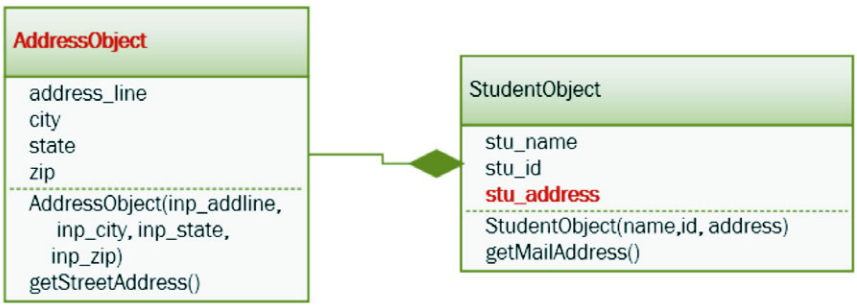


FIGURE 9. This illustration shows the UML class diagrams for three objects. The AddressObject contains attributes that hold address information. The other two objects—StudentObject and TeacherObject—contain an attribute of the AddressObject type. As these classes are related, the filled diamond points to the StudentObject and TeacherObject class diagrams since each contains an instance of the AddressObject as a class attribute.

Using Object Methods and Properties with Object Composition

Once an object has been instantiated, its attributes can be modified and class functions can be invoked. These principles are true even when a composition relationship exists between two classes.




```
1  function AddressObject(inp_addline, inp_city, inp_state, inp_zip)
2  {
3     this.address_line = inp_addline
4     this.city = inp_city
5     this.state = inp_state
6     this.zip = inp_zip
7
8     this.getStreetAddress = function()
9     {
10       return this.address_line + "\n" +
11          this.city + ", " + this.state +
12          this.zip
13    }
14 }
15
16 function StudentObject(name, id, address)
17 {
18    this.stu_name = name
19    this.stu_id = id
20    this.stu_address = address
21
22    this.getMailAddress = function()
23    {
24       return this.stu_name + "\n" +
25          this.stu_address.getStreetAddress()
26    }
27 }
28
29 var stAddress = new AddressObject("12 Hill Street", "Chicago", "IL", 66554)
30 var myStudent = new StudentObject("Jane", 2345, stAddress)
31 alert("Student name is " + myStudent.stu_name)
32 alert(myStudent.getMailAddress())
```

Output

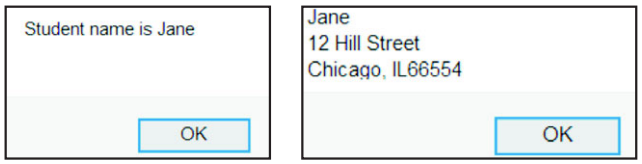


FIGURE 10. Here are the AddressObject and StudentObject class diagrams, with an instance of AddressObject maintained inside the StudentObject definition. To keep the code simple, the TeacherObject class definition is omitted.

FIGURE 10—Interpretation: In the code, lines 1 through 14 implement the AddressObject class definition, and lines 16 through 27 implement the class definition for the StudentObject object. The constructor function of the StudentObject class takes in a parameter called "address" that is placed in the class attribute stu_address. This statement looks no different from lines 18 and 19 that populate the stu_name and stu_id attributes from parameter values.

Line 29 creates an instance of the AddressObject class and gives the instance the name stAddress. This variable is sent in as the third parameter to the StudentObject constructor in line 30 when creating an instance of the StudentObject class in an object variable called myStudent.

In the StudentObject class definition, the getMailAddress class function defined in lines 22 through 26 **concatenates** (brings together two or more separately located objects in a new location) the values in the stu_name class variable with the value returned by the stu_address's getStreetAddress class function. This statement executes correctly since the stu_address class variable was populated using the third parameter in the constructor invocation, and the parameter was of type AddressObject. As can be observed, object relationships can be used in code to implement composition, but it requires an understanding of the class definitions and the nature of the relationship.

PRE-DEFINED OBJECTS IN JAVASCRIPT

In addition to objects that can be created in JavaScript, many built-in objects are available for use. These objects can be instantiated, and their attributes and methods can be used in code.

The String Object

The string object is one of the most commonly used JavaScript objects. This lesson contains examples of variables that hold string data, such as the stu_name class variable maintained in the StudentObject class. However, string variables were treated as "regular" variables and were used without using the "new" keyword. Nevertheless, string is an object in JavaScript, and string variable may be created using object variable syntax.

The string class's constructor may be invoked without any arguments or with one constructor.

```
1  var name_1 = "John"
2
3  var name_2 = new String()
4  name_2 = "Jane"
5
6  var name_3 = new String("Jill")
7
9  alert(name_1 + "\n" + name_2 + "\n" + name_3)
```

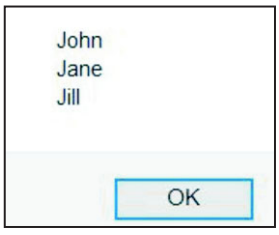


FIGURE 11. Here are the different ways to create string variables. Line 1 creates name_1 as a "regular" variable and assigns a string literal to it. Line 2 uses the "new" keyword to set up an object variable called name_2. This statement clearly shows that string is an object definition. Line 6 also uses the "new" keyword to create a string object variable and passes in a string literal as an argument to the constructor function.

Regardless of how a string variable has been set up, the value maintained in it can be displayed by coding in the name of the variable. **String** is an object that supports many different functions. It also has a property called "length" that returns the number of characters in the string. The value in this property may not be modified.

Function / Attribute	String Function Usage	Output

`length`
 	
Attribute: Returns the number of characters in the string
	`var string_length = myStr.length`
	
	`alert(myStr + "\nlength=" + string_length)`
		JavaScript is fun!
		length=18

`toLowerCase` **Function:** Returns the string with all characters in lower case	`var lc_myStr = myStr.toLowerCase()` `alert("lower case version = \n" + lc_myStr)`	lower case version = javascript is fun!
`toUpperCase` **Function:** Returns the string with all characters in upper case	`var uc_myStr = myStr.toUpperCase()` `alert("upper case version = \n" + uc_myStr)`	upper case version = JAVASCRIPT IS FUN!

`indexOf`
 	
Function: Returns the position of the argument character in the string, with 0 denoting the first character; A-1 is returned if it is not found in the string.
	`var indexof_X = myStr.indexOf("X")`
	`alert("indexOf('X') = " + indexof_X)`
	
	`var indexof_J = myStr.indexOf("J")`
	`alert("indexOf('J') = " + indexof_J)`
		indexOf('X') = -1
		
		indexOf('J') = 0

`charAt` **Function:** Returns the character at a specific position within the string	`var charAt_3 = myStr.charAt(3)` `alert("charAt(3) = " + charAt_3)`	charAt(3) = a

`split`
 	
Function: Splits up a string based upon a delimiter and places the output in an array
	`var words = myStr.split(" ")`
	`var msg=""`
	`for(i=0; i < words.length;i++)`
	`  msg = msg + words[i] + "\n"`
	`alert(msg)`
		JavaScript
		is
		fun!

FIGURE 12. The first entry demonstrates the use of the length property. The toLowerCase and toUpperCase functions take in no parameters and return a version of the string with all characters lowercase or uppercase, respectively. The indexOf function takes in one character or a string as a parameter and locates it within the string. If it is found, the start position is returned with 0 used as the position number for the first position. If the parameter string is not found in the string, the indexOf function returns a –1. The charAt function takes in a number. The character at that position is returned. As with the indexOf function, the position number of the first character in the string is 0. The split function takes in a separator as an argument. In the example shown in FIGURE 11, the space is used as the separator. This action causes the myStr variable to be split into three elements, which are placed in the words array. A "for" loop is used to retrieve individual elements from the array and place in a variable called "msg," which is then displayed out.

The Math Object

JavaScript contains a built-in object called "math" that provides access to mathematical functions, such as random number generation and trigonometric functions. The math class is a static class, so a math object variable cannot be instantiated in code. Instead, the name of the class, "math," is used instead of an instance variable.

INCORRECT Use of Math Class	CORRECT Use of Math Class

`var myMathObj = new Math()`
`var myPI = myMathObg.PI`
`alert("PI=" + myPI)`
	`var myPI = Math.PI`
	`alert("PI=" + myPI)`
	This page says:
	PI=3.141592653589793

FIGURE 13. In the left column, the math object is instantiated, and the name of the object variable is myMathObj. However, since the math class is a static class, instance variables of its type cannot be created. The right side column shows the correct use of the math class. Instead of creating an instance of the class, its methods are used directly by prefixing them with the name of the class instance of the name of the instance variable. The PI attribute of the math class is retrieved and displayed.

The math class provides a function that rounds up a number to the nearest integer. However, the math class does not provide a function to round up a number to a specific number of places after the decimal. To do so, the toFixed method of the number class is used. All variables that contain numeric data are considered instances of the number class.

The math class provides methods that return the maximum and minimum from a series of numbers. If the series of numbers is maintained in an array, the spread operator denoted by three "." symbols is used. This informs the function that an array is specified as an argument instead of a set of numbers.

The math class provides a function called "random" that returns a number between 0 and 1. If an integer value is needed, the value returned by the random function is multiplied by a larger number and rounded up. [See FIGURE 14.]

Function / Attribute	Function Usage	Output
PI **Attribute:** Returns the value of PI	`alert("PI=" +` `Math.PI)`	PI=3.14159265358979

round
 	
Returns the argument number rounded to an integer
	`var number = 123.4567`
	`alert("Math.round(123.4567)=" +`
	`Math.round(number))`
		Math.round(123.4567)=123

toFixed
 	
This is a method of a float number which returns a number rounded to a certain number of places after the decimal point
	`var number_2_dec_places = number.toFixed(2)`
	`alert("number.toFixed(2) = " +`
	`number_2_dec_places)`
		number.toFixed(2)=123.46

max
 	
Function: Returns the maximum value in an array provided the spread operator, "...", is used
	`var numbers = [23, 56, 78, 99,90, 4, 1];`
	`var max_num = Math.max(...numbers);`
	`alert("Max = " + max_num)`
		Max = 99

min **Function:** Returns the minimum value of the arguments sent in	`var min_num = Math.min(45, 67, 89,23, 5, 23)` `alert("Min = " + min_num)`	Min = 5

random
 	
Function: Returns a random number between 0 and 1; To obtain an integer value, multiply the value returned by random and round it
	`var random_num = Math.random()`
	`alert("Math.random=" + random_num)`
	
	`var random_num_0_100 =`
	`Math.round(Math.random() * 101)`
	`alert("Random # 0-100 =" + random_num_0_100)`
		Math.random=0.460681274827
		
		Random # 0-100 =59

FIGURE 14. Math class functions along with examples of their usage. NOTE: The toFixed method is shown in yellow since it is not a math class function.

The Date Object

The date object is another built-in JavaScript object. Unlike the math object, the date object is not a static object. It needs to be instantiated before it can be used. When it is instantiated without any parameters, it returns the current system date from the machine. The date object contains properties that return day of the month, month number, year, and day of the week, along with time-related properties.

The date object provides methods that return day of the month, day of the week, month, and year. These are returned as numbers. The day number is returned as a number in the range 1 to 31. The getDay function returns day of the week with 0 for Sunday and 6 for Saturday. The getMonth returns the month as a number with 0 for January and 11 for December.

Property/Function	Description
`getDate()`	Returns the day of the month as a number 1-31
`getMonth()`	Returns the month as a 0-based number. For example, January is returned as 0, while December is returned as 11.
`getDay()`	Returns the day of the week as a 0-based number. For example, Sunday is returned as 0 and Saturday is returned as 6
`getYear()`	Returns the last 2 digits of year
`getFullYear()`	Returns year in yyyy format

Code that demonstrates use of the date object:

```
var today = new Date()
var today_dd = today.getDate()
var today_mm = today.getMonth()
var today_yyyy = today.getFullYear()

alert((today_mm + 1) + "/" + today_dd + "/" + today_yyyy)
```

Code Output

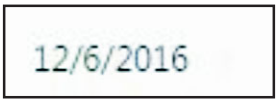


FIGURE 15. Here are some of the methods of the date object, along with code that makes use of it. Output is shown below the code. Notice that the alert statement adds 1 to the variable that holds the value returned by getMonth() since it returns a number that is –0-based instead of being 1-based.

The date object has no methods/functions that return a text representation of the day of the week or the name of the month maintained in a date object. If there is a need to display the day of the week in literals, such as "Sunday," "Monday," code must be written to examine the value returned by the getDay function.

JavaScript contains functions to work with current time. When the date object is instantiated, it holds information about the current date and the current time at the instant it was created. Time-related information may be retrieved using date class functions. The minutes returned by the date object do not contain leading zeros. Therefore, 11:01 will return minutes as 1. Code needs to be written to pad a 0 in front of it. If time is to be displayed in AM/PM format, code needs to be written to examine the value in the hours-variable and display hours accordingly.

```
1   var arr_day=["Sun","Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]
2   var today = new Date()
3   var today_dd = today.getDate()
4   var today_mm = today.getMonth()
5   var today_yyyy = today.getFullYear()
6
7   var today_dow = today.getDay()
8   var today_dow_str = arr_day[today_dow]
9
10  alert(" today_date=" + today_dd +
11          "\ntoday_dow=" + today_dow +
12          "\ntoday_dow_str=" + today_dow_str
13          )
```

Code Output

```
today_date=6
today_dow=2
today_dow_str=Tue
```

FIGURE 16. Here is code that displays the name of the week as a string literal. The first line of code declares an array that holds string literals corresponding to weekday names, starting with "Sunday" since the getDay function returns a 0 for "Sunday." In line 8, the value returned by the getDay function is used as a subscript across the arr_days array defined in line 1. This retrieves the literal that corresponds to the day of the week and is displayed in the alert statement.

Property/Function	Description
getMinutes()	Returns the number of minutes since the last hour 1 to 59.
getHours()	Returns the hours from the start of day as a number 0 to 23.
getSeconds()	Returns the number of seconds from the start of the last minute 1 to 59.

Code demonstrating use of the date object:

```
var now_time = new Date()
var hh = now_time.getHours()
var minm = now_time.getMinutes()
if (minm < 10)
   minm = "0" + minm

alert(hh + ":" + minm)
```

Code Output

```
15:37
```

FIGURE 17. This illustration is of a time-related function maintained in the date object, along with code that displays the current time. [NOTE: A decision statement is used to display 1 minute as "01" instead of "1."]

OOP can be used to create a user-defined object that encapsulates date and time information and returns user-friendly date and time information. In the class, two class variables are set up to hold string representation of date and time in the class constructor. Once the class has been instantiated, these class variables can be used to display date and time.

Class Code

```
1   function DisplayDateTime()
2   {
3         var arr_day=["Sun","Mon", "Tue", "Wed", "Thu", "Fri", "Sat"]
4         var arr_month=["Jan","Feb", "Mar", "Apr", "May", "Jun",
5          "Jul","Aug", "Sep", "Oct", "Nov", "Dec"]
6         var now = new Date()
7         var now_dd = now.getDate()
8         var now_dow_num = now.getDay()
9         var now_mm = now.getMonth()
10        var now_yyyy = now.getFullYear()
11        var hh = now.getHours()
12        var minutes = now.getMinutes()
13
14        var now_mon_str = arr_month[now_mm]
15        var now_dow_str = arr_day[now_dow_num]
16
17        if (minutes < 10)
18              minutes = "0" + minutes
19
20        this.fulldate = now_dow_str + ", " +
21              now_mon_str + " " + now_dd + " " + now_yyyy
22              + " (" + (now_mm + 1) + "/" + now_dd + "/" +
23              now_yyyy + ")"
24
25        this.fulltime = hh + ":" + minutes
26  }
27
28  var now_date_time = new DisplayDateTime()
29  alert("Todays date is " + now_date_time.fulldate +
30        "\nThe current time is " +now_date_time.fulltime)
```

Output

FIGURE 18. Class UML and code is shown, along with code that instantiate and use the class. The class contains two properties and no methods. Lines 1 through 26 contain the class definition. There are many variables used in the class constructor function. However, only the highlighted variables are class properties since they are prefixed with the "this" keyword. Arrays are set up in lines 3 through 5 to hold string literals to represent day of week and month names. The current date is retrieved in line 6. Date class functions are used in lines 7 through 12. Some of these values are used as subscripts across the arrays defined in lines 3 through 5. Lines 20 through 25 populate the class properties. Line 28 instantiates the DisplayDateTime object and the alert statement in lines 29 and 30 retrieves the values in the class private variables and displays it back.

EXPLORING OUR WORLD...

SCIENCE CONNECTION: Built-in Objects

Any numbers used in JavaScript are instances of the number object. Because "number" is an object, it contains properties and methods. Learning to use number object's methods allows for fine precision when handling numeric data in JavaScript code, such as rounding up numbers to a set number of decimal places or formatting it based on the locale of the browser software. Learn more about JavaScript—The Number Object and its methods at

https://www.tutorialspoint.com/javascript/javascript_number_object.htm

In addition to objects that can be created in JavaScript, many built-in objects are available for use. These objects can be instantiated, and their attributes and methods can be used in code.

This object also demonstrates the power of OOP technology. If the class constructor definition is placed in a .js file, it may be used across an entire website to retrieve and display system date and time.

The Array Object Properties and Methods

Arrays have been discussed in-depth in prior lessons. An array is a "compound" variable that stores multiple elements inside it, and each individual element within an array is accessed using a numeric position number known as a subscript. However, an array is actually an object of type Array, where Array is a built-in object in JavaScript. Since it is an object, it supports methods and properties. The length property is a read-only property and returns the number of elements in the array.

The sort function arranges the elements of the array in ascending order. Once the sort has been completed, the original order of elements is lost. The reverse function reorders the array from back to front. The indexOf function is similar to the indexOf function for the string class. It returns the position of the function argument in the array, and a –1 if the argument element is not present as an array element.

The array object supports other functions, such as:

- The pop function removes the last element in the array.
- The push function serves as an array append function.
- The shift function removes the first element in the array.

- The unshift function, which takes in an argument, places the argument value at the start of the array. So shift and unshift work at the beginning of the array object, while pop and push work at the end of the array object.
- Some array functions allow for addition and deletion of elements in the middle of the array.

Function/ Attribute	String Function Usage	Usage, Output

length	Number of elements in the array	`var names = ["John", "Jane", "Tim", "Pia", "Amelia"]`
		`alert(names + "\n# of elements="+ names.length)`
		John,Jane,Tim,Pia,Amelia
		# of elements=5

sort	Sorts array elements	`var names = ["John", "Jane", "Tim", "Pia", "Amelia"]`
		`names.sort()`
		`alert(names)`
		Amelia,Jane,John,Pia,Tim

reverse	Reverses the order of elements in the array	`names.reverse()`
		`alert(names)`
		Amelia,Pia,Tim,Jane,John

pop	Removes the last element in the array	`names.pop()`
		`alert(names)`
		John,Jane,Tim,Pia

push	Appends an element to the end of the array	`names.push("Xander")`
		`alert(names)`
		John,Jane,Tim,Pia,Amelia,Xander

indexOf	Returns the position of an element in the array, and returns a −1 if the element is not in the array	`var names = ["John", "Jane", "Tim", "Pia", "Amelia"]`
		`alert("indexOf('John')=" + index_of_john)`
		`var index_of_greene = names.indexOf("Greene")`
		`alert("indexOf('Greene')=" + index_of_greene)`
		indexOf('John')=0
		indexOf('Greene')=-1

FIGURE 19. The uses of array class functions.

Approaches to Iterate an Array

"Traditional For Loop" approach: In the prior examples, a "for" loop was used to iterate through the elements of an array object using a numeric subscript that starts at 0 and ends at 1 less than the array size. However, other methods may be used to iterate an Array object.

"Callback" approach: "Callback" is the second approach. It uses the "forEach" reserved word. In this approach, a function is defined without a name. It takes in 1 parameter that is a placeholder for a single array element. The "callback" function is called once for each element in the array.v

"Object-Properties" approach: The "object-properties" approach iterates once for each element in the array. Unlike the "callback" approach, this technique returns the numeric subscript for each element in the array instead of the element itself.

Method Type	Code

Traditional "For" Loop	`var names = ["John", "Jane", "Tim", "Pia", "Amelia"]`
	
	`var names_msg = ""`
	`for(var i = 0; i < names.length; i++)`
	`  names_msg += names[i] + "\n"`
	
	`alert(names_msg)`

Callback Function	`names_msg = ""`
	`names.forEach(function (one_name)`
	`  {`
	`  names_msg += one_name + "\n"`
	`  }`
	`)`
	`alert(names_msg)`

For-in "Properties of an Object" Syntax	`names_msg = ""`
	`for(one_subs in names)`
	`  names_msg += names[one_subs] + "\n"`
	
	`alert(names_msg)`

Output produced by each of the alert statements:

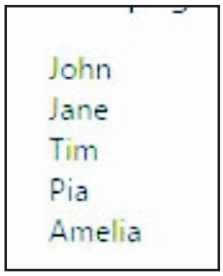


FIGURE 20. This illustration shows various methods of iterating an array.

Other JavaScript Objects

Aside from the built-in objects, JavaScript maintains objects that deal with the use of JavaScript in a webpage. For example:

- Window: The window object maintains browser window related properties.
- Navigator: The navigator object keeps track of the browser being used (e.g., Internet Explorer, Firefox, or Chrome).
- Screen: The screen object is closely related to the window object and is related to the actual monitor being used.
- History: The history object keeps a list of URLs visited by the browser in the current session.
- Location: The location object maintains the URL of the current page.
- DOM: The document object model (DOM) maintains information about the content in a webpage.

Object Type	Description
Window	It represents the browser window. It maintains properties that keep track of size and position of the browser window.
Navigator	It maintains information about the type of browser being used.
Screen	It maintains information about the screen portion of the browser window, such as color resolution.
History	It tracks the pages visited in the current browser session.
Location	It maintains the current URL of the browser window.
Document object model	It maintains information about the HTML content of a webpage.

FIGURE 21. These are examples of JavaScript objects that deal with browsers and with webpages.

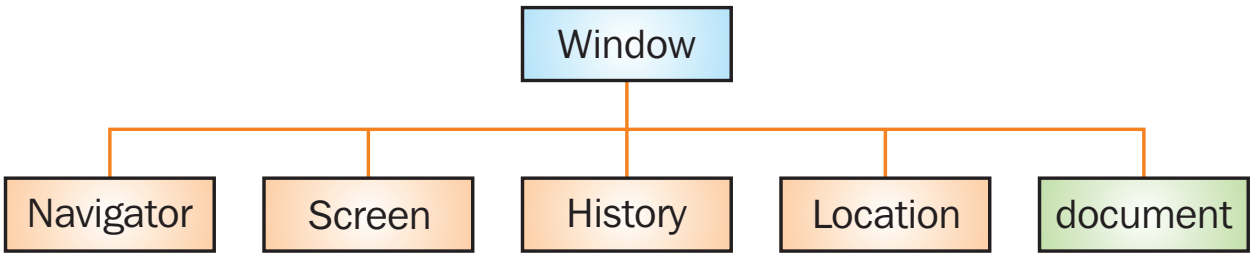


FIGURE 22. This visual illustrates the hierarchy of browser-related objects. All of these objects are global objects and do not need to be instantiated. They are available for use as soon as the webpage is loaded into a browser window.

DOM: DOCUMENT OBJECT MODEL

DOM Objects

A webpage is a document that maintains content using HTML elements. The visual aspect of content is maintained using CSS styles. The **document object model (DOM)** is an entity that provides an OOP visualization of webpage content. Since DOM is an object, it supports properties and methods. These properties and methods can be used to modify webpage content and styles using JavaScript code.

When a webpage is loaded into the browser, an instance of the DOM is automatically created and given the name "document." This object's write method was used to display content in a webpage. All the HTML elements in the page are contained inside this object.

The document object visualizes the page as an inverted tree with nodes. All nodes in the DOM hierarchy have a parent node—except for the HTML node, which is the root of the hierarchy. Some of the elements contain text content maintained in a text node. Aside from the text node that holds data, elements may have attributes.

Both Google Chrome and Firefox have tools that display the DOM view of a webpage. It is displayed in text format and not as a graphic. It displays the HTML in the document in an indented format.

HTML File

```
<!DOCTYPE html>
<head>
   <meta charset="utf-8" />
   <title>Myfirst HTML5 page</title>

</head>
<body>
   <header>
      <h1>DOM-I</h1>
   </header>
   <section>
      <h2>Elements in DOM</h2>
      <p>Here is a paragraph</p>
      <img src="my_logo.gif" />
   </section>
</body>
</html>
```

DOM View of the File

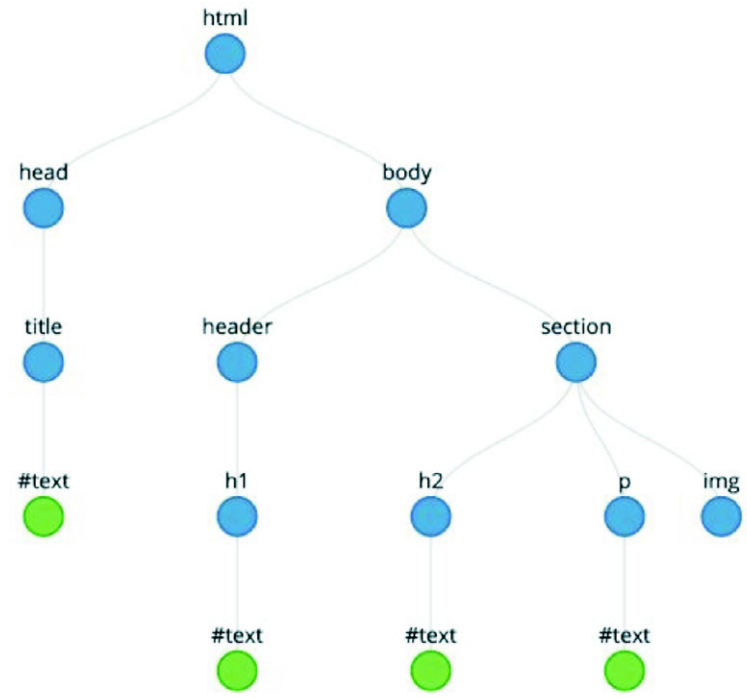


FIGURE 23. The DOM inverted tree structure is shown in this simple illustration. Nodes that pertain to the <title>, <h1>, <h2>, and <p> nodes contain text inside them via a text node.

Accessing DOM Nodes

Nodes can be retrieved using several methods. Three methods of the document object can be used to retrieve one or multiple elements.

Method 1

The getElementsByTagName takes in one argument that is the name of an element. For example, document.getElementsByTagName("p") retrieves all the <p> elements in the document. Since there may be multiple elements with the same tag name, these are returned by the method in an array-like object called a **NodeList**. Elements are retrieved from a NodeList using subscripts, with 0 used to retrieve the first element in the NodeList.

Method 2

The getElementsByName retrieves elements by their name attribute. Again, this method returns a NodeList, with 0 to many elements. The number of elements in the NodeList is maintained in a property called "length."

Method 3

The getElementById takes in the value in the id attribute as an argument and returns a single element. This is because no two elements in a document may have the same value for the id property. It is used as a unique identifier for each element in a webpage.

Name Attribute vs id Attribute: Elements in a form are required to have a name attribute because the program that receives form data (the name of the server program that receives form data is specified in the action attribute of the form element) identifies form elements by their name attribute. However, the id attribute is used by JavaScript to identify an individual element in a webpage. Therefore, it is a good design principle to provide both a name and an id attribute for each element in a webpage that will be accessed via JavaScript or be sent to a server program.

Method	Description
getElementsByTagName	It returns a collection of nodes, such as the collection of all <h2> elements in the document. This collection is maintained in an object called a NodeList. Elements in a NodeList are retrieved like elements in an array using a numeric subscript, with 0 used as the subscript for the first element. The NodeList object contains a length property that returns the number of elements in the NodeList object.
getElementById	It returns a single node (or element) from the document since the id attribute cannot be duplicated across multiple elements in a document.
getElementsByName	It returns a collection of nodes, such as the collection of all <h2> elements in the document. This collection is maintained in an object called a NodeList. Elements in a NodeList are retrieved like elements in an array using a numeric subscript, with 0 used as the subscript for the first element. The NodeList object contains a length property that returns the number of elements in the NodeList object.

FIGURE 24. The methods to retrieve DOM nodes are described here.

Manipulating DOM Nodes

Once an element has been retrieved from the webpage, its properties can be changed. For example, the color-related or font-related properties of an element can be changed. Many ways exist to change the text content of an element or DOM node. By modifying an element's textContent or innerHTML property, the text content of an element is changed. The textContent modifies only the text in the element, while innerHTML allows for the specification of HTML, along with content. Good programming practice dictates that the textContent property be used to modify the text, and other methods should be used to modify style-related information.

Property	Code
`textContent`	Modifies the text content of a node. Accepts only text
`innerHTML`	Modifies the text content of a node and accepts HTML code

Code that demonstrates these properties:

```
<!DOCTYPE html>
<head>
   <meta charset="utf-8" />
   <title>DOM Related</title>
   <script>
      function btn_click_handler()
      {
         var h2_elements = document.getElementsByTagName("h2")

         h2_elements[0].textContent = "???"
         h2_elements[1].innerHTML = "<a href='www.google.com'>Google!</a>"
      }
   </script>
</head>
<body>
   <header>
      <h2>DOM-I</h2> <!-- 1st h2 element [0]-->
   </header>
   <section>
      <h2>Elements in DOM</h2> <!-- 2nd h2 element [1]-->
      <p>Here is a paragraph</p>
      <img src="Bird-Logo.png" /><br/>

      <input type="button" value = "Check DOM" onclick="btn_click_handler()" />
   </section>
</body>
</html>
```

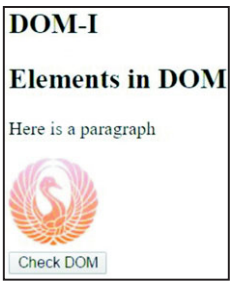
When Page Loads	After Button is Clicked
	

FIGURE 25. Here is an example of a webpage that contains a button. The btn_click_handler function executes when the button is clicked. In this function, the h2_elements variables are a NodeList variable that holds all the h2 elements in the webpage. Since there are two <h2> elements, the first is retrieved using the subscript value of 0 and the second one is retrieved using the subscript value of 1. The first element's textContent property is used to modify the <h2> element's text to "???" For the second element, the innerHTML property is used to introduce an <a> element in the text of the h2 element. Since the new content includes an HTML element, innerHTML was used instead of textContent.

Nodes in a DOM hierarchy have a property called style. This is an object with properties. The properties of the style object are closely linked to the properties of the CSS style attribute. The names of the style object's properties are closely related to the corresponding CSS property. The main difference is that some CSS properties have a "-" in them, such as background-color, and the corresponding style object property does not have the "-." The "-" is considered an invalid symbol in an identifier in JavaScript, where the "-" is used to denote a subtraction operator.

Style Object Property	Description
background	Set background-color, background-image, and other properties in a single statement.
backgroundColor	Set background color.
border, borderColor, boxShadow	Set all border-related properties in a single statement, or set each border-related property in separate statements.
font, fontFamily, fontStyle	Set font-related properties in a single statement or in separate statements.
textShadow	Set text shadow styles.
transform, transition	Set transform- and transition-related properties.

Code

```
<!DOCTYPE html>
<head>
   <script>
      function btn_click_handler()
      {
         var header_h2 = document.getElementById("header_h2_1")
         header_h2.style.textShadow = "-3px 3px 1px grey"
      }
   </script>
</head>
<body>
      <h2 id = "header_h2_1">DOM-I</h2>
      <input type="button" value = "Check DOM" onclick="btn_click_handler()" />
</body>
</html>
```

When Page Loads	After Button is Clicked
	

FIGURE 26. Here are the properties of the style object and a brief description of their functionality, along with code that demonstrates their usage. In the button object's event handler, the <h2> element is retrieved using the document's getElementById method. Since the <h2> element has been set up with an id attribute, it can be retrieved using the getElementById method. Once it has been retrieved, its style object's textShadow property is set. This is shown in the table that displays the element at webpage load time and after the button has been clicked.

Learning the main principles of the DOM object hierarchy, properties, and methods of objects in this hierarchy is essential to creating applications that modify content and styles dynamically.

Consider an HTML file that displays a dropdown list box by using a <select> element with various nested <option>elements. The first <option> element has an empty string as

```
<!DOCTYPE html>
<head>
   <script>
      function change_color()
      {
         var bus_list = document.getElementById("BUS")
         bus_list.style.color = "black"

         var cis_list = document.getElementById("CIS")
         cis_list.style.color = "black"

         var selected_maj = document.getElementById("lstmajors")
         var selected_val = selected_maj.value

         if (selected_val != "")
         {
            var list_to_change = document.getElementById(selected_val)
            list_to_change.style.color = "blue"
         }
      }
   </script>
</head>
<body>
   <p>Select a major:</p>
   <select id = "lstmajors" onchange = "change_color()"
         onfocus = "this.selectedindex = -1;">
      <option value="">--select major--</option>
      <option value = "BUS">Business</option>
      <option value = "CIS">Computer Information Systems</option>
   </select>

   <ol>
      <li>Business courses
         <ol id="BUS" >
            <li>bus100</li>
            <li>bus200</li>
         </ol>
      </li>
      <li>CIS courses

         <ol id="CIS">
            <li>cis100</li>
            <li>cis200</li>
         </ol>
      </li>
   </ol>
</body>
</html>
```

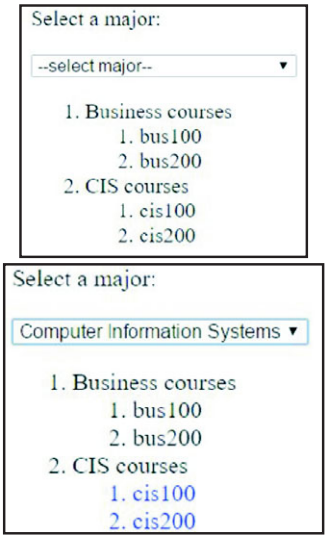


FIGURE 27. Here is code to display a dropdown list and its event handlers.

the value attribute, while the other 2 <option> elements have three character string literals for their value attribute.

- The dropdown list has an id value of "lstmajors" and two event handlers. The first event handler is for the change event triggered when a user selects an item in the list. This causes the change_color function to execute.
- In this event, the color style property for all <ol> list elements are set to black. Then the value attribute of the selected <option> element is retrieved. Next, the <ol> list that corresponds to the selected element is retrieved, and the color of the list is set to blue.

Comingling HTML and JavaScript Code

Consider code that wishes to change the content of an HTML element based on the day of the week. For example, on weekends, the contents of a <h2> element must say "Happy Weekend" and "Off to Work" on days other than the weekend.

```
<!DOCTYPE html>
<head>
   <script>
      var today = new Date()
      var dow = today.getDay()
      if (dow == 0 || dow == 6) //0=sunday, 6=saturday
         message = "Happy Weekend!"
      else
         message = "Off to Work :-( "

      //document.getElementById("h2").textContent = message

   </script>
</head>
<body>
   <h2 id = "h2">
      <script>
         document.write(message)
      </script>
   </h2>
</body>
</html>
```

FIGURE 28. Here in the <script> element, the day of the week is retrieved. Based on the value in that variable, a literal is placed in the message variable. In the HTML portion of the document, within the <h2> element, a mini-script section is created with a document.write statement that writes out the message variable. The value placed in the message variable is displayed out.

FURTHER EXPLORATION...

ONLINE CONNECTION: Coding Standards

Websites contain HTML code, CSS code, and JavaScript code. Developing, updating, and maintaining all of these different code blocks can be a daunting task. There are sites that discuss the best practices to be used when developing code. Setting up coding standards creates consistency and can lead to cleaner code with fewer errors. Additionally, if everyone in a development follows the same standards, collaborative efforts are possible. To learn more about JavaScript Best Practices, go to:

http://www.w3schools.com/js/js_best_practices.asp

Websites contain HTML code, CSS code, and JavaScript code.

FIGURE 28—Interpretation: Note the commented out line that changes the textContent of the <h2> element. The line has been commented out because it causes a runtime error. The reason is that the code within the <script> element executes even before the webpage has loaded into the browser window. So when that statement executes, there may not be an element with the id attribute “h2" in the page. Therefore, the content of the <h2> element needs to be set inside the HTML portion of the document where the system can be sure that the element exists within the browser window.

Summary:

JavaScript is a mature and object-oriented programming environment. Additionally, webpage elements are exposed to JavaScript as objects and may be manipulated using object-oriented JavaScript code. Learning JavaScript and the DOM is crucial to creating dynamic websites.

Checking Your Knowledge:

1. How can you use the math object to round up numbers?
2. How can the string object be used to convert strings to uppercase alphabets?
3. How can JavaScript be used to change the border property of a webpage element?
4. What method can be used to retrieve all <p> elements in a webpage?

5. Can there be multiple math objects in a webpage? What explains your response?

Expanding Your Knowledge:

Use the information discussed to create a function that changes the background image of a webpage based on the month of the year.

Web Links:

DOM Visualizer
http://bioub.github.io/d3.DOMVisualizer/

HTML DOM Style Object
http://www.w3schools.com/jsref/dom_obj_style.asp

JavaScript String Reference
http://www.w3schools.com/jsref/jsref_obj_string.asp

JavaScript Math Reference
http://www.w3schools.com/jsref/jsref_obj_math.asp

UML Tutorial
http://www.tutorialspoint.com/uml/

Unified Modeling Language
https://en.wikipedia.org/wiki/Unified_Modeling_Language