# PeopleCert
# Software Development Skills

JavaScript Stream

Lesson 13

Study Guide

## Copyright Details

- Document Version: PC-SDS_SG 1.0 | February 2020

PeopleCert

All talents, certified.

# PeopleCert: A Global Leader in Certification

- ✓ **Web** & **Paper based** exams in **25** languages
- ✓ **2,500 Accredited Training Organisations** worldwide
- ✓ Delivering exams across **200 countries every year**
- ✓ Comprehensive Portfolio of **500+ Exams** and Growing

AXELOS GLOBAL BEST PRACTICE · PeopleCert DevOps · digital marketing institute · IASSC · ISACA · Language Cert · LEAN IT ASSOCIATION · Quality Software Development · ECDL

PeopleCert
All talents, certified.

## How to Use This Document

This document is your **PeopleCert Software Developer Skills Study Guide** to help you prepare for the **PeopleCert Software Developer Skills Foundation & Advanced examination**.

It is meant to provide you with a clear outline of everything covered in the course presentation by your instructor that will be on the PeopleCert Software Developer Skills Foundation & Advanced exams.

Your exams will be closed book. You will be given 120 minutes to complete it. It contains 100 multiple choice questions and to pass the exam you must achieve a grade of 65% or higher, or a minimum of 65/100 correct responses. For further details on your exam, including more information on question types and learning objectives, please refer to your course syllabus.

As you follow along, you may see that some material here is not replicated in the trainer presentation. This study guide includes questions, activities, knowledge checks, or other material in the presentation that are facilitated verbally by the instructor. It also does not contain content that is not examinable, but instead is designed to reinforce learning or add value to your course experience.  It also provides valuable links and references, throughout the slides, which you can explore further to enhance your learning and understanding of the material provided in the study guide.

PeopleCert
All talents, certified.

## Coding Bootcamp

## Lesson 13

## Web Design and Development Fundamentals (Front End)

### Objectives

- JavaScript Functions
- Object Prototypes
- Exception Handling
- Document Object Model
- JavaScript Events
- JSON
- XML Syntax and Validation
- Data Validation

### Syllabus Items:

- 4.6.1 Code in JavaScript, variables, functions
- 4.6.2 Code using advance java script: if conditions, loops

## Syllabus

| Category | Topic | Task |
|---|---|---|
| **FSD_4 Web Design and Development Fundamentals (Front-End)** | **4.6 JavaScript/ jQuery** | 4.6.1 Code in JavaScript, variables, functions |
| | | 4.6.2 Code using advance java script: if conditions, loops |
| | | 4.6.3 Improve usability of forms, validate data from the user |

## Contents | Learning Objectives

- ✓ Understand the JavaScript functions and the different ways that they can be defined and called
- ✓ Understand the object prototypes
- ✓ Familiarize with the exception handling
- ✓ Understand the Document Object Model and the selection process
- ✓ Familiarize with the JavaScript events
- ✓ Understand the JSON and XML model
- ✓ Learn how to validate the form data

## Introduction to JS Functions

**Definition:**

*A **function** is a JavaScript procedure—a set of statements that performs a task or calculates a value.*

- Functions are one of the fundamental building blocks in JavaScript.
- To use a function, you must define it somewhere in the scope from which you wish to call it.
- A JavaScript function is a block of code designed to perform a particular task
  - This block can be reused within the code, and, if provided different arguments, can produce different results
  - Executed when "something“ invokes it (calls it – via an event call, direct call from code, or automatically)
  - A JavaScript function is defined with the function keyword, followed by a name, followed by parentheses ()
- Function names can contain letters, digits, underscores, and dollar signs - same rules as variables
  - In fact, functions can be used the same way as you use variables, in all types of formulas, assignments, and calculations
- The parentheses may include parameter names separated by commas: (parameter1, parameter2, ...)
- The code to be executed, by the function, is placed inside curly brackets: {}
- Inside a function:
  - When JavaScript reaches a return statement, the function will stop executing.
- If the function was invoked from a statement, JavaScript will "return" to execute the code after the invoking statement
- If no return statement is explicitly defined in a function, an implicit **return undefined;** takes place.
- Functions are the building block for modular code in JavaScript
- function subtotal(price,quantity) {

```
function subtotal(price,quantity) {
        return price * quantity;
}
```

- The above is formally called a function declaration, called or invoked by using the () operator

```
        var result = subtotal(10,2);
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions*

## Defining Functions

- A function definition (also called a function declaration, or function statement) consists of the function keyword, followed by:
  - The name of the function.
  - A list of parameters to the function, enclosed in parentheses and separated by commas.
  - The JavaScript statements that define the function, enclosed in curly brackets, { }.on Declarations
- In addition to defining functions as described below, you can also use the Function constructor to create functions from a string at runtime, much like eval().

**Example 1**

```
function square(number) {
  return number * number;
}
```

- Code above defines a simple function named **square**
- The function square takes **one parameter**, called **number**
- The function consists of one statement that says to return the parameter of the function (that is, number) multiplied by itself
- The return statement specifies the value returned by the function
- Primitive parameters (such as a number) are passed to functions **by value**; the value is passed to the function, but if the function changes the value of the parameter, this change is not reflected globally or in the calling function.

**Example 2**

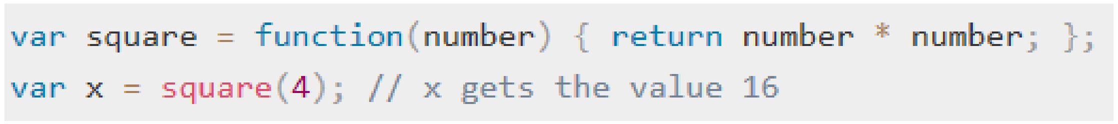
If you pass an object (i.e. a non-primitive value, such as **Array** or a user-defined object) as a parameter and the function changes the object's properties, that change is visible outside the function, as shown in the example below:

```
function myFunc(theObject) {
  theObject.make = 'Toyota';
}

var mycar = {make: 'Honda', model: 'Accord', year: 1998};
var x, y;

x = mycar.make; // x gets the value "Honda"

myFunc(mycar);
y = mycar.make; // y gets the value "Toyota"
                // (the make property was changed by the function)
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions*

## Function Expressions

- While the function declaration above is syntactically a statement, functions can also be created by a function expression.
- Such a function can be anonymous; it does not have to have a name.
- For example, the function square could have been defined as:



```
var square = function(number) { return number * number; };
var x = square(4); // x gets the value 16
```

```
// defines a function using a function expression
var sub = function subtotal(price, quantity) {
    return price * quantity;
};
// invokes the function
var result = sub(10,2);
```

- It is conventional to leave out the function name in function expressions

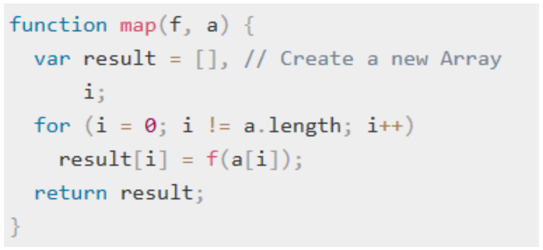
*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions*

## Anonymous Function Expression

```
// defines a function using an anonymous function expression
var calculateSubtotal = function (price,quantity) {
   return price * quantity;
};
// invokes the function
var result = calculateSubtotal(10,2);
```

## Passing Functions

- Function expressions are convenient when passing a function as an argument to another function
- The following example shows a map function that should receive a function as first argument and an array as second argument



```
function map(f, a) {
  var result = [], // Create a new Array
      i;
  for (i = 0; i != a.length; i++)
    result[i] = f(a[i]);
  return result;
}
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions*

## Nested Functions

- You can nest a function within a function.
- The nested (inner) function is private to its containing (outer) function. It also forms a closure. A closure is an expression (typically a function) that can have free variables together with an environment that binds those variables (that "closes" the expression).
- Since a nested function is a closure, this means that a nested function can "inherit" the arguments and variables of its containing function. In other words, the inner function contains the scope of the outer function.
- To summarize:
  - The inner function can be accessed only from statements in the outer function.
  - The inner function forms a closure: the inner function can use the arguments and variables of the outer function, while the outer function cannot use the arguments and variables of the inner function.

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions*

## Nested Functions – **Examples**

**Example 1**

```
function calculateTotal(price,quantity) {
        var subtotal = price * quantity;
        return subtotal + calculateTax(subtotal);
        // this function is nested
        function calculateTax(subtotal) {
                var taxRate = 0.05;
                var tax = subtotal * taxRate;
                return tax;
        }
}
```

**Example 2**

```
function addSquares(a, b) {
        function square(x) {
                return x * x;
        }
        return square(a) + square(b);
}
```

## Class Exercises

- Consider the following code:

```
function map(f, a) {
  var result = []; // Create a new Array
  var i; // Declare variable
  for (i = 0; i != a.length; i++)
    result[i] = f(a[i]);
      return result;
}
var f = function(x) {
   return x * x * x;
}
var numbers = [0,1, 2, 5,10];
var cube = map(f,numbers);
console.log(cube);
```

- What is happening here?
- What is the outcome of this code?

- Consider the following code:

```
function addSquares(a, b) {
  function square(x) {
    return x * x;
  }
  return square(a) + square(b);
}
a = addSquares(2, 3);
b = addSquares(3, 4);
c = addSquares(4, 5);
```

- What will the following lines return?

  a = addSquares(2, 3);

  b = addSquares(3, 4);

  c = addSquares(4, 5);

## Defining Functions

- In JavaScript, a function can be defined based on a condition.
- For example, the following function definition defines **myFunc** only if **num equals 0**:

```
var myFunc;
if (num === 0) {
  myFunc = function(theObject) {
    theObject.make = 'Toyota';
  }
}
```

## Calling Functions

- Defining a function does not execute it
- Defining the function simply names the function and specifies what to do when the function is called
- Calling the function actually performs the specified actions with the indicated parameters
- Functions must be in scope when they are called

**Example:** if you define the function square, you could call it as follows:
**square(5);**
*This statement calls the function with an argument of 5. The function executes its statements and returns the value 25.*

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions*

## Calling Functions

- A function can call itself.
- For example, here is a function that computes factorials **recursively**:

```
function factorial(n) {
  if ((n === 0) || (n === 1))
    return 1;
  else
    return (n * factorial(n - 1));
}
```

- You could then compute the factorials of one through five as follows:

```
var a, b, c, d, e;
a = factorial(1); // a gets the value 1
b = factorial(2); // b gets the value 2
c = factorial(3); // c gets the value 6
d = factorial(4); // d gets the value 24
e = factorial(5); // e gets the value 120
```

- There are other ways to call functions
  - There are often cases where a function needs to be called dynamically
  - Call a function where the number of arguments needs to vary
  - The context of the function call needs to be set to a specific object determined at runtime
- It turns out that functions are, themselves, objects, and these objects in turn have methods (see the Function object).
- One of these, the apply() method, can be used to achieve this goal.

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions*

## Calling Functions (2)

- The function .call() and .apply() are very similar in their usage except a little difference. .call() is used when the number of the function's arguments are known to the programmer, as they have to be mentioned as arguments in the call statement. On the other hand, .apply() is used when the number is not known. The function .apply() expects the argument to be an array.
- The basic difference between .call() and .apply() is in the way arguments are passed to the function. Their usage can be illustrated by the following example:

```
var someObject = {
myProperty : 'Foo',
        myMethod : function(prefix, postfix) {
            alert(prefix + this.myProperty + postfix);
        }
};
someObject.myMethod('<', '>'); // alerts '<Foo>'
var someOtherObject  = {
            myProperty : 'Bar'
};
someObject.myMethod.call(someOtherObject, '<', '>'); // alerts '<Bar>'

someObject.myMethod.apply(someOtherObject, ['<', '>']); // alerts '<Bar>'
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions*

# Callback Functions

1. 
```
var calculateTotal = function (price, quantity, tax) {
    var subtotal = price * quantity;
    return subtotal + tax(subtotal);
};

var calcTax = function (subtotal) {
    var taxRate = 0.05;
    var tax subtotal * taxRate;
    return tax;
};

var temp = calculateTotal(50,2,calcTax);
```

*The local parameter variable tax is a reference to the calcTax() function*

*Passing the calcTax() function object as a parameter*

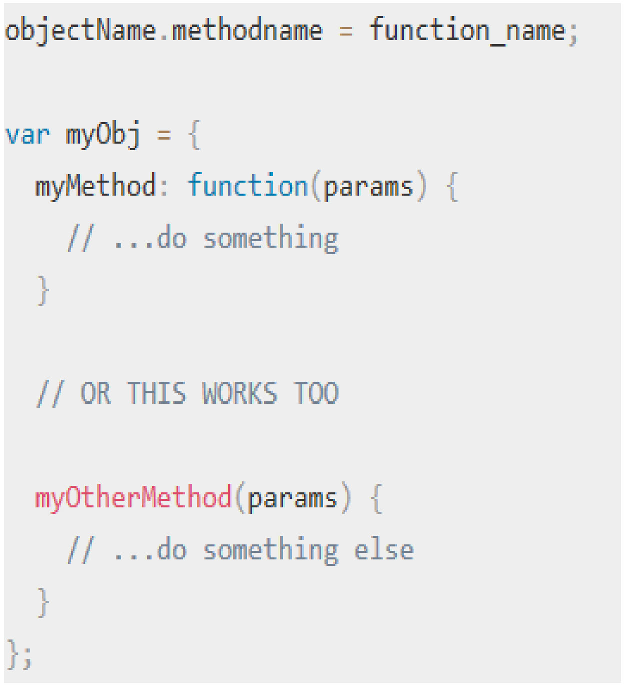
*We can say that calcTax variable here is a callback function*

2. In the previous example we can write the callback function inline:

```
var temp = calculateTotal( 50, 2, function (subtotal) {
                                   var taxRate = 0.05;
                                   var tax = subtotal * taxRate;
                                   return tax;
                       }
      );
```

## Methods

- A **method** is a function that is a property of an object
- A **method** is a **function** associated with an object, or, simply put, a method is a property of an object that is a function
- Methods are defined the way normal functions are defined, except that they have to be assigned as the property of an object



```
objectName.methodname = function_name;

var myObj = {
  myMethod: function(params) {
    // ...do something
  }

  // OR THIS WORKS TOO

  myOtherMethod(params) {
    // ...do something else
  }
};
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects*

## Using this for Object References

- JavaScript has a special keyword, **this**, that you can use within a method to refer to the current object
- **this** refers to the calling object in a method
- When combined with the **form** property, **this** can refer to the current object's parent form.

### Example

- In the following example, the form myForm contains a Text object and a button.
- When the user clicks the button, the value of the Text object is set to the form's name.
- The button's onclick event handler uses this.form to refer to the parent form, myForm.

```
<form name="myForm">
<p><label>Form name:<input type="text" name="text1" value="Beluga"></label>
<p><input name="button1" type="button" value="Show Form Name"
     onclick="this.form.text1.value = this.form.name">
</p>
</form>
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects*

## Objects and Functions Together

```
var order = {
      salesDate : “May, 5, 2017”,
      product : {
                  type: “laptop”,
                  price: 500.00,
                  output: function () {
                              return this.type + ‘ $’ + this.price;
                              }
      },
      customer : {
                  name: “Sue Smith”,
                  address: “123 Somewhere Str.”,
                  output: function () {
                              return this.name + ‘, ’ + this.address;
                  }
      },
      output: function () {
                  return ‘Date’ + this.salesDate;
      }
};
```

## Scope

- Scope determines the accessibility (visibility) of variables
- In JavaScript there are two types of scopes:
  - Local scope
  - Global scope
- JavaScript has function scope
  - Each function creates a new scope
- Variables defined inside a function are not accessible (visible) from outside the function
- Since local variables are only recognized inside their functions, variables with the same name can be used in different functions
  - Local variables are created when a function starts, and deleted when the function is completed
- JavaScript also has block scope, defined by the { curly brackets }. Only variables declared with **let** or **const** remain private inside block scope.
- A variable declared outside a function, becomes GLOBAL
  - All scripts and functions on a web page can access it
  - In HTML, the global scope is the window object
  - If you assign a value to a variable that has not been declared, it will automatically become a GLOBAL variable

- Do NOT create global variables unless you intend to
  - Your global variables (or functions) can overwrite window variables (or functions)
  - Any function, including the window object, can overwrite your global variables and functions

*Source: https://www.w3schools.com/js/js_scope.asp*

## Function Scope

- Variables defined inside a function cannot be accessed from anywhere outside the function, because the variables are defined only in the scope of the function
- However, a function can access all variables and functions defined inside the scope in which it is defined
- A function defined in the global scope can access all variables defined in the global scope
- A function defined inside another function can also access all variables defined in its parent function and any other variable to which the parent function has access

**Example**

```
// Function declaration
function myFunction() {
        carName = “Volvo”;
        var scopedCarName = “Saab”;

        // Code here can use globalCarName and carName (as they are global)
        // but also scopedCarName as local variable to myFunction
}

var globalCarName = “BMW”;

// Code her can use globalCarName as a global variable
// Code here can use carName as a global variable
// Code here can not use scopedCarName

// Function call
myFunction();
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects*

## Error Handling Tips

- Ensure your code is working generally
- Common JavaScript problems that you will want to be mindful of, such as:
  - Basic syntax and logic problems
  - Making sure variables, etc. are defined in the correct scope, and you are not running into conflicts between items declared in different places
  - Confusion about this, in terms of what scope it applies to, and therefore if its value is what you intended
  - Incorrectly using functions inside loops
  - Making sure asynchronous operations have returned before trying to use the values they return

## Alert

- The alert() function makes the browser show a pop-up to the user, with whatever is passed being the message displayed
- The following JavaScript code displays a simple hello world message in a pop-up:

```
alert ( "Good Morning" );
```

- Using alerts can get tedious fast:
  - When using debugger tools in your browser you can write output to a log with:
  - `console.log("Put Messages Here");`
  - And then use the debugger to access those logs

*Source: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/JavaScript*

## Tools

- You can ensure better quality, less error-prone JavaScript code using a linter, which points out errors and can also flag up warnings about bad practices, etc., and be customized to be stricter or more relaxed in their error/warning reporting.
- The JavaScript/ECMAScript linters recommended are:
  - JSHint
  - ESLint
- The Atom code editor has a JSHint plugin
  - Go to Atom's Preferences... dialog (e.g. by Choosing Atom > Preferences... on Mac, or File > Preferences... on Windows/Linux) and choose the Install option in the left-hand menu.
  - In the Search packages text field, type "jslint" and press Enter/Return to search for linting-related packages.
  - You should see a package called lint at the top of the list. Install this first (using the Install button), as other linters rely on it to work. After that, install the linter-jshint plugin.

*Source: https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/JavaScript*

## Error Handling Statements

- You can **throw exceptions** using the throw statement and handle them using the try...catch statements.
  - throw statement
  - try...catch statement
- Exception Types: Just about any object can be thrown in JavaScript. Nevertheless, not all thrown objects are created equal. While it is fairly common to throw numbers or strings as errors it is frequently more effective to use one of the exception types specifically created for this purpose:
  - ECMAScript exceptions
  - DOMException and DOMError

*Source: https://developer.mozilla.org/en-US/docs/docs/Web/JavaScript/Guide/Control_flow_and_error_handling*

try...catch statement

- The **try...catch statement** marks a block of statements to try, and specifies one or more responses should an exception be thrown.
- If an exception is thrown, the try...catch statement catches it.
- The try...catch statement consists of a try block, which contains one or more statements, and a catch block, containing statements that specify what to do if an exception is thrown in the try block. That is, you want the try block to succeed, and if it does not succeed, you want control to pass to the catch block.
- If any statement within the try block (or in a function called from within the try block) throws an exception, control immediately shifts to the catch block.
- If no exception is thrown in the try block, the catch block is skipped.
- The finally block executes after the try and catch blocks execute but before the statements following the try...catch statement.

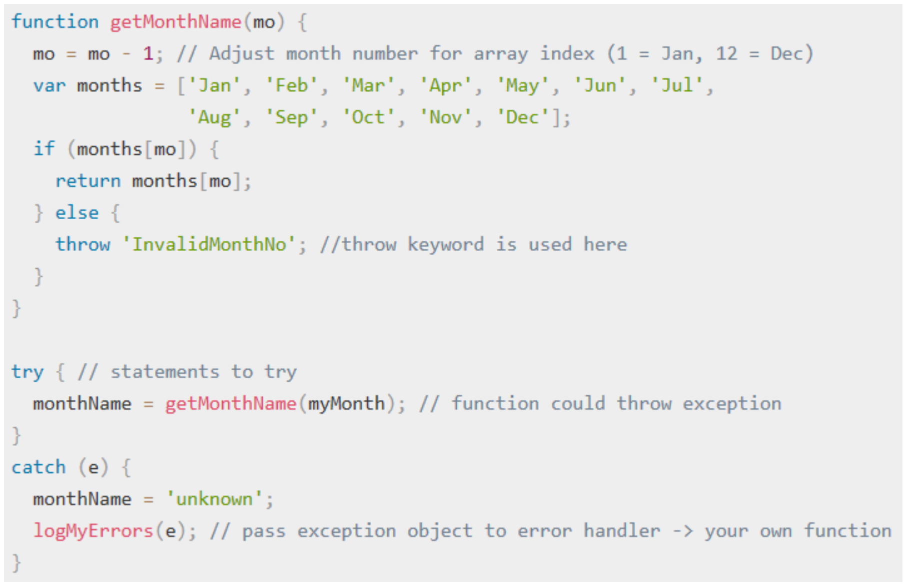
*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch*

## Errors Using Try and Catch

- When the browser's JavaScript engine encounters an error, it will throw an exception
- These exceptions interrupt the regular, sequential execution of the program and can stop the JavaScript engine altogether
- However, you can optionally catch these errors preventing disruption of the program using the try–catch block

```
try {
        nonexistantfunction("hello");
}
catch(err) {
        alert("An exception was caught: " + err);
}
```

**try…catch Example**

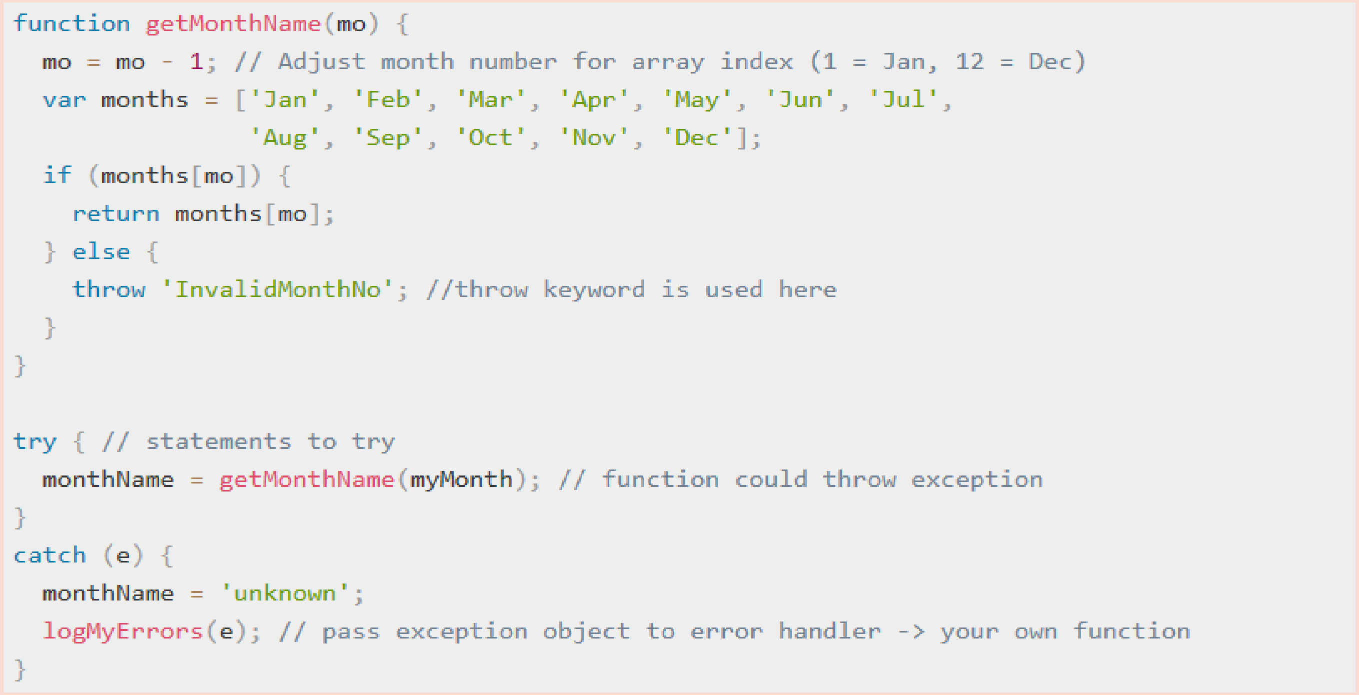


```
function getMonthName(mo) {
  mo = mo - 1; // Adjust month number for array index (1 = Jan, 12 = Dec)
  var months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
                'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
  if (months[mo]) {
    return months[mo];
  } else {
    throw 'InvalidMonthNo'; //throw keyword is used here
  }
}

try { // statements to try
  monthName = getMonthName(myMonth); // function could throw exception
}
catch (e) {
  monthName = 'unknown';
  logMyErrors(e); // pass exception object to error handler -> your own function
}
```

*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

## Class Discussion

✓ What does this function do?



```
function getMonthName(mo) {
  mo = mo - 1; // Adjust month number for array index (1 = Jan, 12 = Dec)
  var months = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul',
                'Aug', 'Sep', 'Oct', 'Nov', 'Dec'];
  if (months[mo]) {
    return months[mo];
  } else {
    throw 'InvalidMonthNo'; //throw keyword is used here
  }
}

try { // statements to try
  monthName = getMonthName(myMonth); // function could throw exception
}
catch (e) {
  monthName = 'unknown';
  logMyErrors(e); // pass exception object to error handler -> your own function
}
```

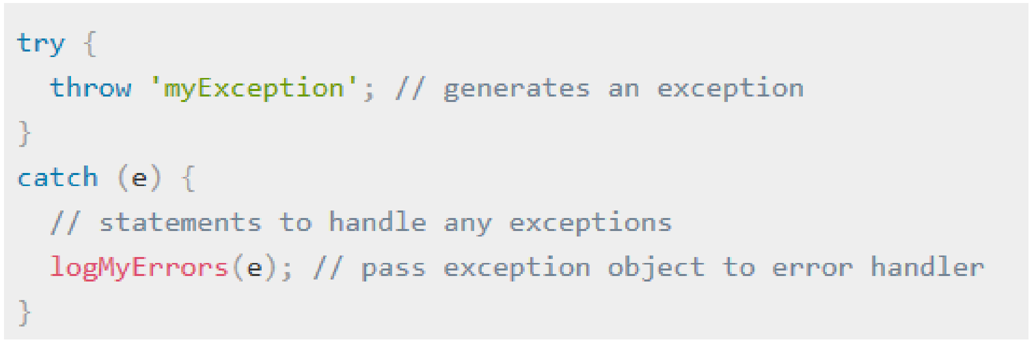
## The **catch block**

- You can use a catch block to handle all exceptions that may be generated in the try block.

  **catch (catchID) {**
   **statements**
  **}**
- The catch block specifies an identifier (catchID in the preceding syntax) that holds the value specified by the throw statement; you can use this identifier to get information about the exception that was thrown.
- JavaScript creates this identifier when the catch block is entered; the identifier lasts only for the duration of the catch block; after the catch block finishes executing, the identifier is no longer available.

**Example:**

- For example, the following code throws an exception. When the exception occurs, control transfers to the catch block.



```
try {
  throw 'myException'; // generates an exception
}
catch (e) {
  // statements to handle any exceptions
  logMyErrors(e); // pass exception object to error handler
}
```

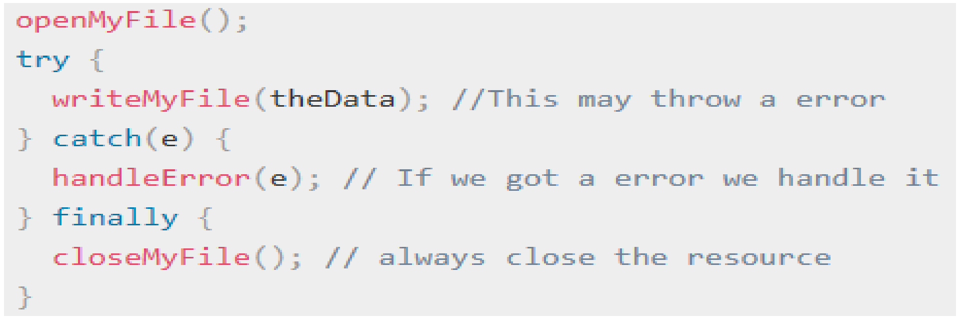
*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling*

## The **finally block**

- The finally block contains statements to execute after the try and catch blocks execute but before the statements following the try...catch statement.
- The finally block executes whether or not an exception is thrown. If an exception is thrown, the statements in the finally block execute even if no catch block handles the exception.
- You can use the finally block to make your script fail gracefully when an exception occurs; for example, you may need to release a resource that your script has tied up.

**Example 1**

- The following example opens a file and then executes statements that use the file (server-side JavaScript allows you to access files).
- If an exception is thrown while the file is open, the finally block closes the file before the script fails.



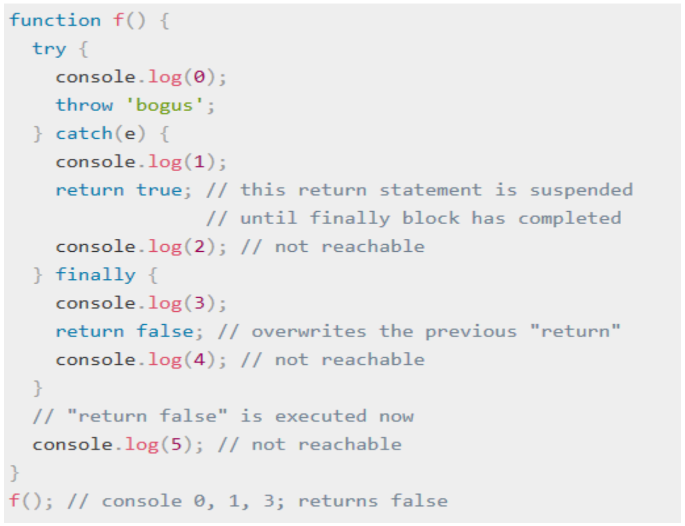
```
openMyFile();
try {
  writeMyFile(theData); //This may throw a error
} catch(e) {
  handleError(e); // If we got a error we handle it
} finally {
  closeMyFile(); // always close the resource
}
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling*

## The **finally block**

### Example 2

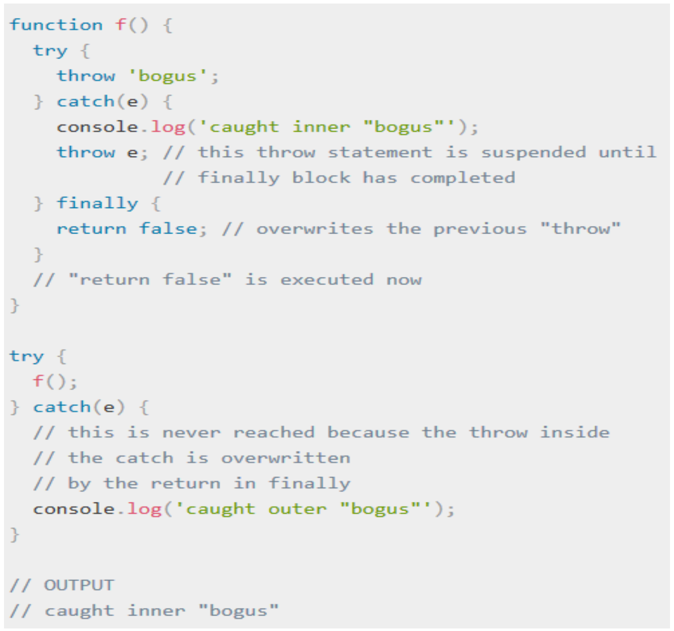
- If the finally block returns a value, this value becomes the return value of the entire try-catch-finally production,
  regardless of any return statements in the
  try and catch blocks:



```
function f() {
  try {
    console.log(0);
    throw 'bogus';
  } catch(e) {
    console.log(1);
    return true; // this return statement is suspended
                 // until finally block has completed
    console.log(2); // not reachable
  } finally {
    console.log(3);
    return false; // overwrites the previous "return"
    console.log(4); // not reachable
  }
  // "return false" is executed now
  console.log(5); // not reachable
}
f(); // console 0, 1, 3; returns false
```

### Example 3

- Overwriting of return values by the finally block also applies to exceptions thrown or re-thrown inside of the catch block:



```
function f() {
  try {
    throw 'bogus';
  } catch(e) {
    console.log('caught inner "bogus"');
    throw e; // this throw statement is suspended until
             // finally block has completed
  } finally {
    return false; // overwrites the previous "throw"
  }
  // "return false" is executed now
}

try {
  f();
} catch(e) {
  // this is never reached because the throw inside
  // the catch is overwritten
  // by the return in finally
  console.log('caught outer "bogus"');
}

// OUTPUT
// caught inner "bogus"
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling*

## Nesting try…catch Statements

- You can nest one or more try...catch statements.
- If an inner try...catch statement does not have a catch block, it needs to have a finally block and the enclosing try...catch statement's catch block is checked for a match.

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling*

## Throw your own

- Although try-catch can be used exclusively to catch built-in JavaScript errors, it can also be used by your programs, to throw your own messages
- The throw keyword stops normal sequential execution, just like the built-in exceptions

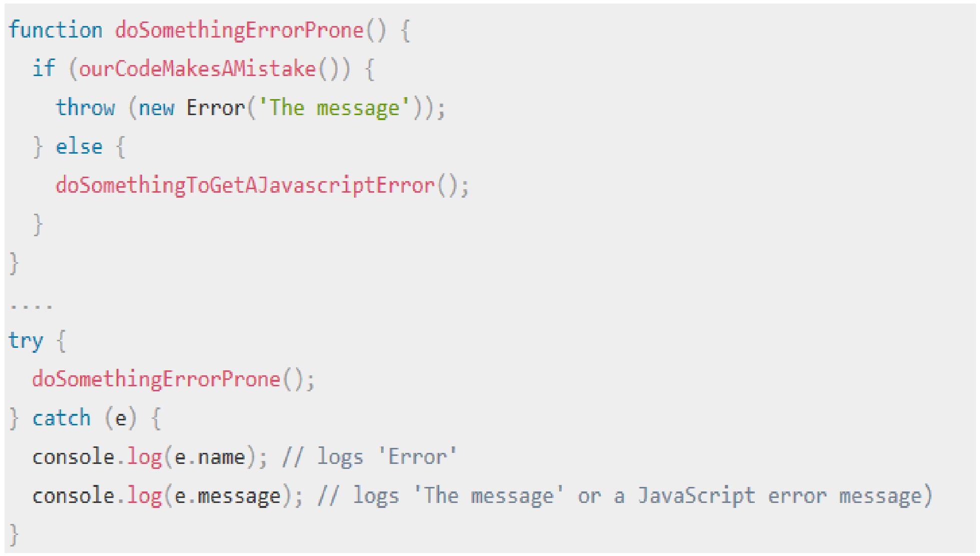
```
try {
        var x = -1;
        if (x<0)
          throw “smallerthanoError”;
}
catch(err) {
        alert(err + “was thrown”);
}
```

*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

## Utilizing Error objects

- Depending on the type of error, you may be able to use the 'name' and 'message' properties to get a more refined message. 'name' provides the general class of Error (e.g., 'DOMException' or 'Error'), while 'message' generally provides a more succinct message than one would get by converting the error object to a string.
- If you are throwing your own exceptions, in order to take advantage of these properties (such as if your catch block doesn't discriminate between your own exceptions and system ones), you can use the Error constructor.

**Example**



```
function doSomethingErrorProne() {
  if (ourCodeMakesAMistake()) {
    throw (new Error('The message'));
  } else {
    doSomethingToGetAJavascriptError();
  }
}
....
try {
  doSomethingErrorProne();
} catch (e) {
  console.log(e.name); // logs 'Error'
  console.log(e.message); // logs 'The message' or a JavaScript error message)
}
```

*Source: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling*

## The Document Object Model (DOM) – Part I

## DOM Introduction

**Definition:**

*The Document Object Model (DOM) is a programming interface for HTML.*

- It provides a structured representation of the document (e.g. a web page) as a tree.
- It defines methods that allow access to the tree, so that they can change the document structure, style and content.
- JavaScript is almost always used to interact with the HTML document in which it is contained
- This is accomplished through a programming interface (API) called the Document Object Model
- According to the W3C, the **DOM** is a:

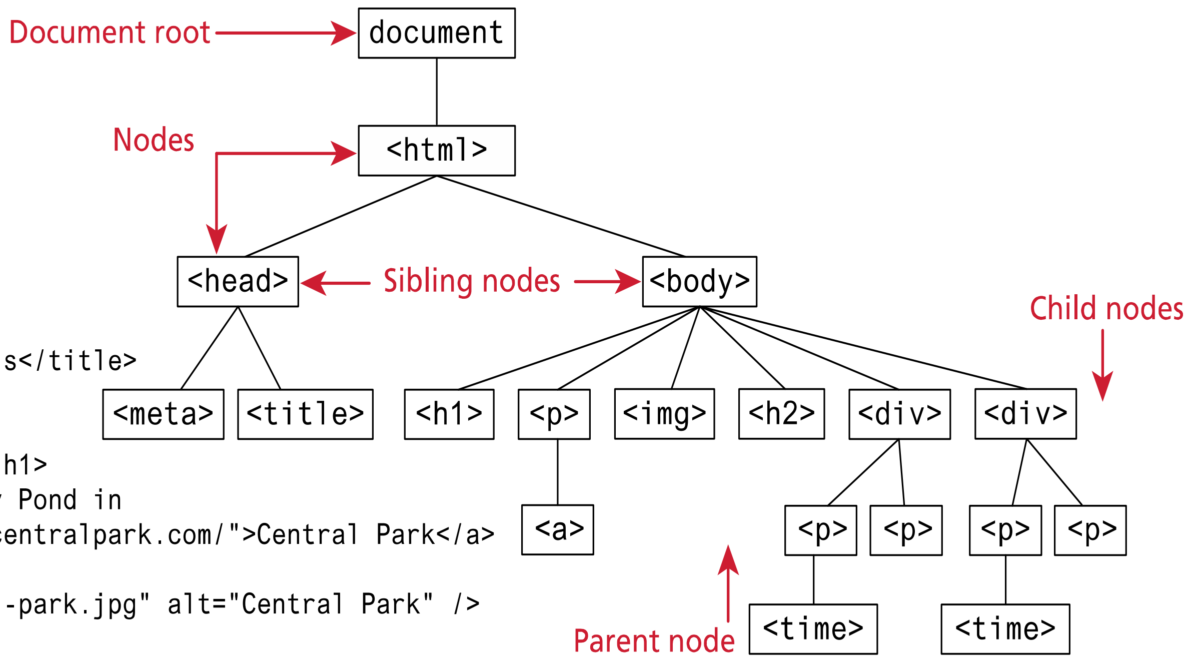
**Definition:**

*Platform- and language-neutral interface that will allow programs and scripts to dynamically access and update the content, structure and style of documents.*

*Source: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model*

## DOM Overview

```
<html>
<head>
   <meta charset="utf-8">
   <title>Share Your Travels</title>
</head>
<body>
   <h1>Share Your Travels</h1>
   <p>Photo of Conservatory Pond in
      <a href="http://www.centralpark.com/">Central Park</a>
   </p>
   <img src="images/central-park.jpg" alt="Central Park" />
   <h2>Reviews</h2>
   <div id="latestComment">
      <p>By Ricardo on <time>2016-05-23</time</p>
      <p>Easy on the HDR buddy.</p>
   </div>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p>I love Central Park.</p>
   </div>
</body>
</html>
```



*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

## DOM Nodes

- In the DOM, each element within the HTML document is called a node. If the DOM is a tree, then each node is an individual branch
- There are:
  - element nodes,
  - text nodes, and
  - attribute nodes
- All nodes in the DOM share a common set of properties and methods

## Element Node Object

- Element Node object represents an HTML element in the hierarchy, contained between the opening <> and closing </> tags for this element
- Every node has
  - classList
  - className
  - Id
  - innerHTML
  - Style
  - tagName

*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

## Essential Node Object properties

| Property | Description |
|---|---|
| attributes | Collection of node attributes |
| childNodes | A NodeList of child nodes for this node |
| firstChild | First child node of this node |
| lastChild | Last child of this node |
| nextSibling | Next sibling node for this node |
| nodeName | Name of the node |
| nodeType | Type of the node |
| nodeValue | Value of the node |
| parentNode | Parent node for this node |
| previousSibling | Previous sibling node for this node |

*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

## Other Common Attributes

- Other common attributes include:
  - href
  - name
  - src
  - value
- However these do not exist in all objects

## Document Object

- The DOM document object is the root JavaScript object representing the entire HTML document

```
// retrieve the URL of the current page
var a = document.URL;
// retrieve the page encoding, for example ISO-8859-1
var b = document.inputEncoding;
```

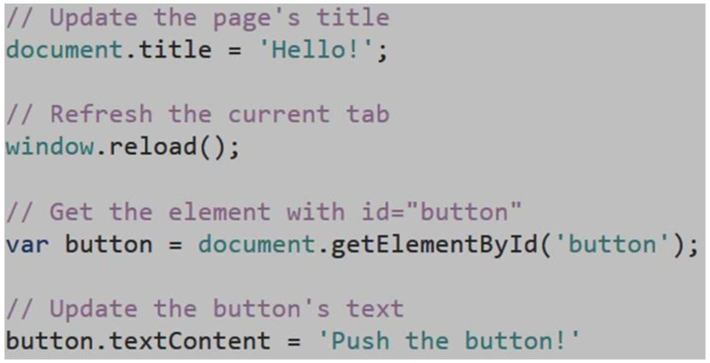
*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

## Selection Methods (1)

- Classic
  - getElementById()
  - getElementsByTagName()
  - getElementsByClassName()
- Newer
  - querySelector() and
  - querySelectorAll()

## Accessing the DOM

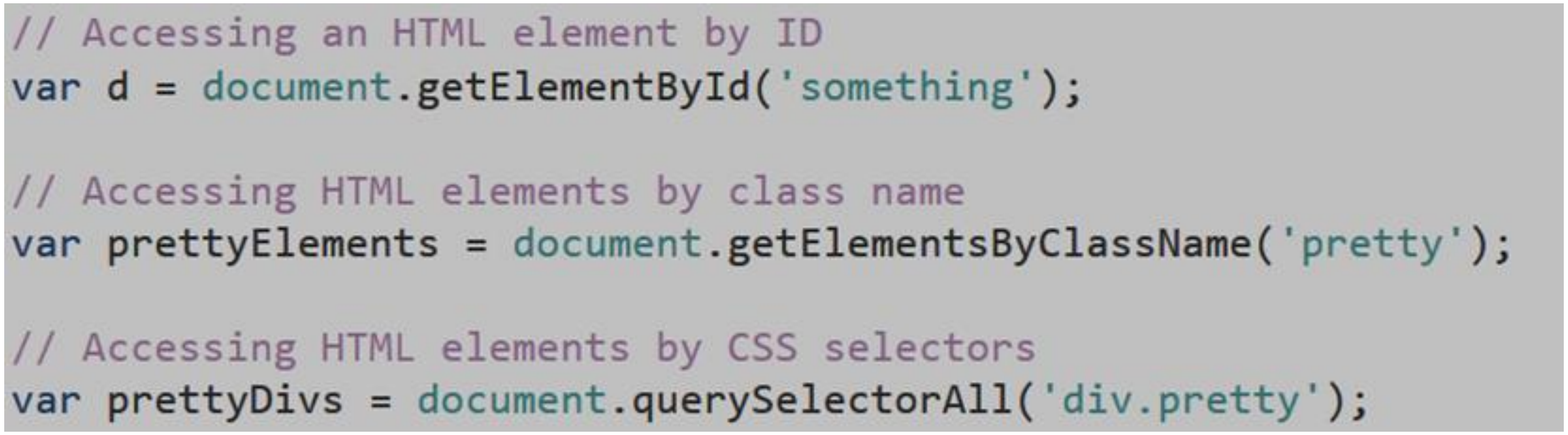
- The browser provides us with access to DOM objects



```
// Update the page's title
document.title = 'Hello!';

// Refresh the current tab
window.reload();

// Get the element with id="button"
var button = document.getElementById('button');

// Update the button's text
button.textContent = 'Push the button!'
```

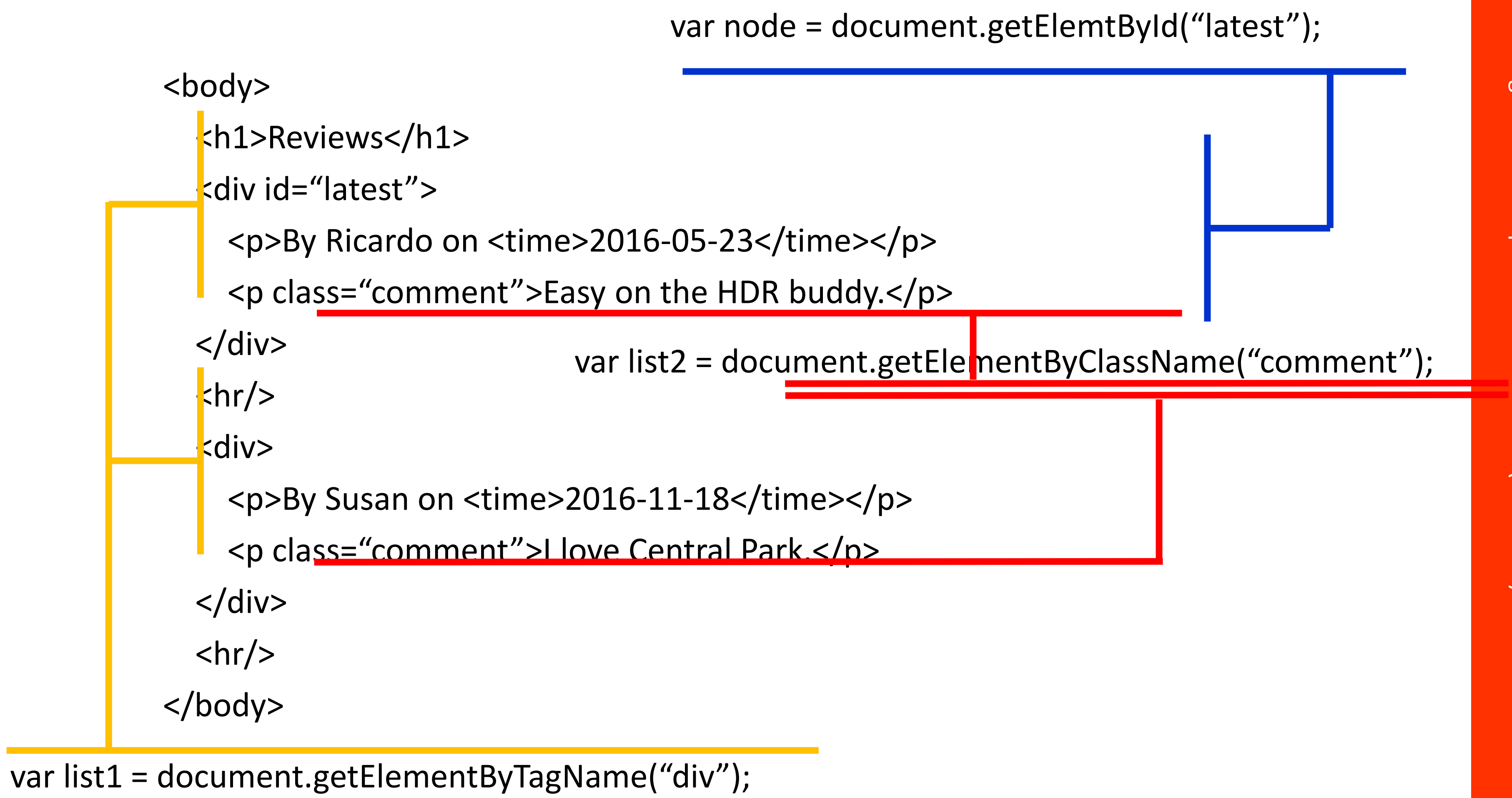
## Accessing HTML Elements

The browser provides us with access to HTML elements



```
// Accessing an HTML element by ID
var d = document.getElementById('something');

// Accessing HTML elements by class name
var prettyElements = document.getElementsByClassName('pretty');

// Accessing HTML elements by CSS selectors
var prettyDivs = document.querySelectorAll('div.pretty');
```

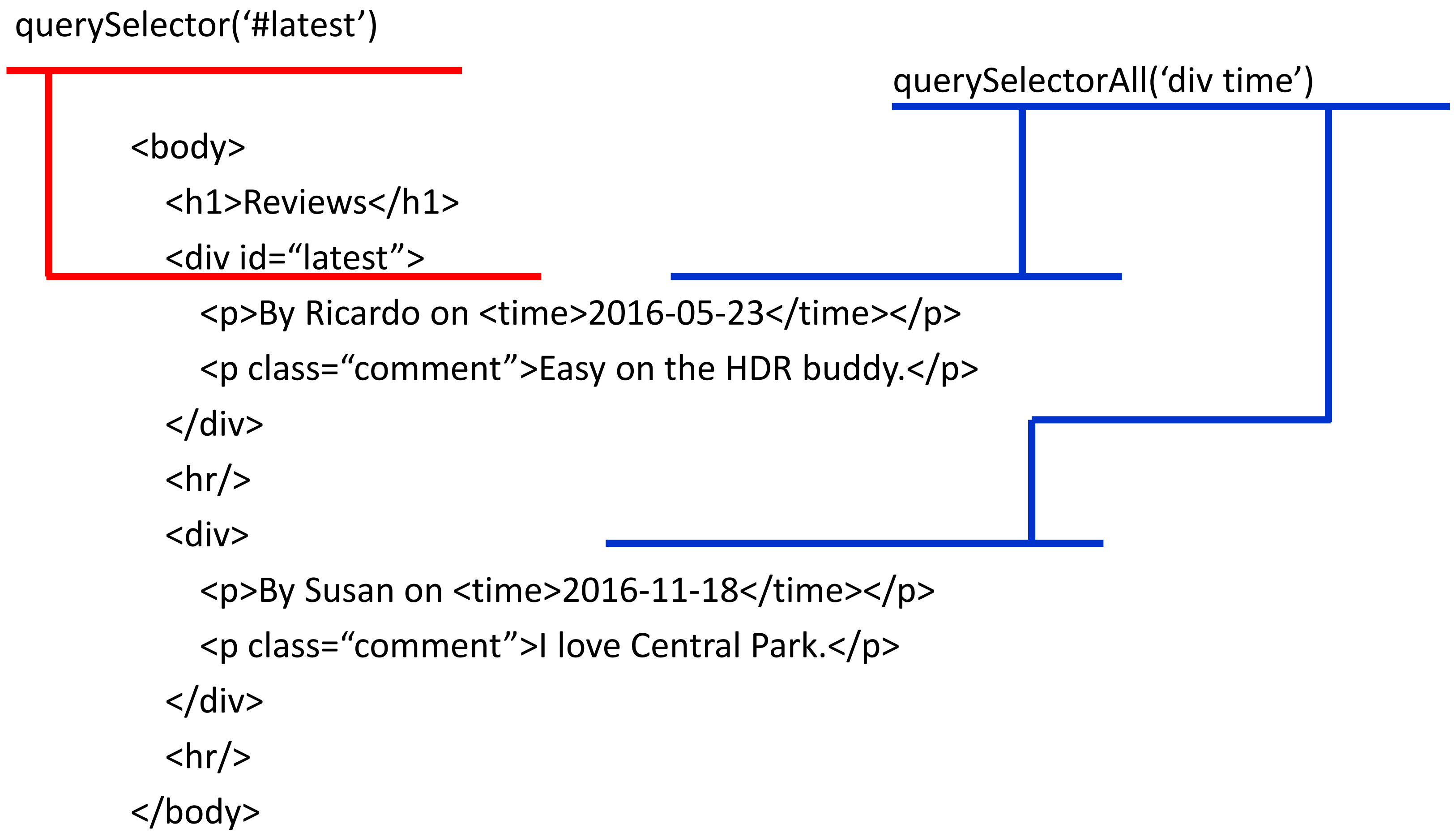
## Selection Methods



```
var node = document.getElemtById(“latest”);

<body>
   <h1>Reviews</h1>
   <div id=“latest”>
      <p>By Ricardo on <time>2016-05-23</time></p>
      <p class=“comment”>Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p class=“comment”>I love Central Park.</p>
   </div>
   <hr/>
</body>

var list2 = document.getElementByClassName(“comment”);

var list1 = document.getElementByTagName(“div”);
```

*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

## Query Selectors



querySelector(‘#latest’)

querySelectorAll(‘div time’)

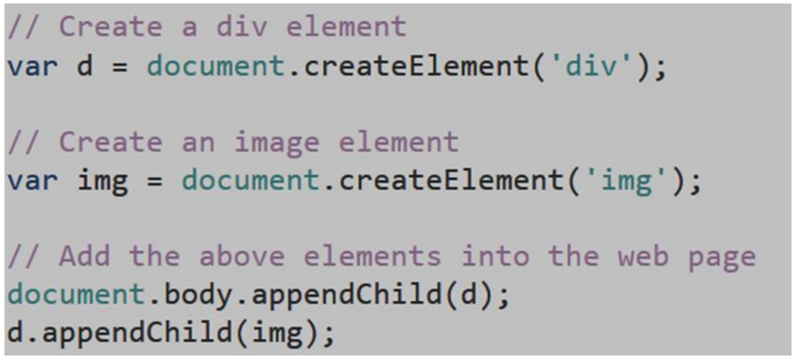
```
<body>
   <h1>Reviews</h1>
   <div id=“latest”>
      <p>By Ricardo on <time>2016-05-23</time></p>
      <p class=“comment”>Easy on the HDR buddy.</p>
   </div>
   <hr/>
   <div>
      <p>By Susan on <time>2016-11-18</time></p>
      <p class=“comment”>I love Central Park.</p>
   </div>
   <hr/>
</body>
```

*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

## Creating HTML Elements

- In an HTML document, the **Document.createElement() method** creates the HTML element specified by tagName, or an HTMLUnknownElement if tagName isn't recognized.
- In a XUL document, it creates the specified XUL element.
- In other documents, it creates an element with a null namespace URI.
- To explicitly specify the namespace URI for the element, use document.createElementNS().
- Syntax:
  **var element = document.createElement(tagName[, options]);**

**Example**



```
// Create a div element
var d = document.createElement('div');

// Create an image element
var img = document.createElement('img');

// Add the above elements into the web page
document.body.appendChild(d);
d.appendChild(img);
```

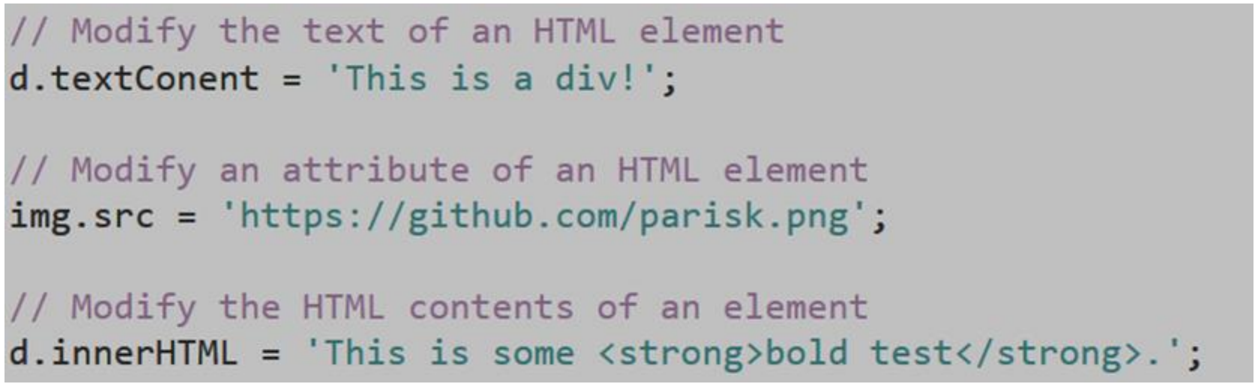
*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

## Class Exercise 3

- Create a <img></img> element and append it to the element with id parent

## Modifying HTML Elements

**Example**



```
// Modify the text of an HTML element
d.textConent = 'This is a div!';

// Modify an attribute of an HTML element
img.src = 'https://github.com/parisk.png';

// Modify the HTML contents of an element
d.innerHTML = 'This is some <strong>bold test</strong>.';
```

*Source: Randy Connolly and Ricardo Hoar: Fundamentals of Web Development*

**It is time for a Knowledge Check!**

## Sample Questions

1. A JavaScript __________ is a block of code designed to perform a particular task.

   A. variable

   B. error

   C. function

   D. Object

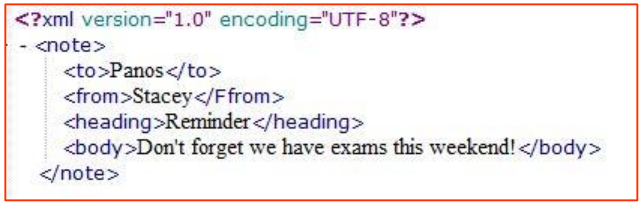
2. Consider the XML code to the right. This code:

   A. Has a syntax error

   B. Is error free

   C. Is not standard XML code

   D. Is JavaScript, not XML



```
<?xml version="1.0" encoding="UTF-8"?>
- <note>
    <to>Panos</to>
    <from>Stacey</Ffrom>
    <heading>Reminder</heading>
    <body>Don't forget we have exams this weekend!</body>
  </note>
```

3. What will the following JS code do?

```
<!DOCTYPE html>
<html>
<body>
<button
onclick="document.getElementById('demo').innerHTML
=Date()">The time is?</button>
<p id="demo"></p>
</body>
</html>
```

   A. Will create a button, which when clicked will take us to the InnerHTML page

   B. Will create a link named The time is?, which when clicked will take us to a page with the current date and time

   C. Will create a link named Demo, which when clicked internally navigate to an area where a preset date is displayed

   D. Will create a button, which when clicked will display the current date and time

## Class Review Exercises

- Write a JavaScript program that accept two integers and display the larger.
  Challenge: What if the user gives as input a non-numeric value?
- Write a JavaScript program that takes a positive number as argument and prints all the numbers in the console, along with either “Odd” or “Even” based on what they are.
- Describe the attributes of a cell phone in both XML and JSON format:
  - Brand: Samsung,
  - Model: Galaxy S8,
  - Cores: 8,
  - Storage in GB: 64
  - Screen Size in Inches: 5.8.
  - Features: SMS, MMS, Email, Push Mail, IM

## Exercises

1. Write a JavaScript program where the program takes a random integer from 1 to 10, the user is then prompted to input a guess number. If the user input matches with guess number, the program will display a message "Good Work" otherwise display a message "Not matched".
   **Tip 1**: Use Math.random() that returns a random number between 0 (inclusive) and 1 (exclusive).
   **Tip 2**: Use Math.ceil(number) that returns the smallest integer greater than or equal to a given number.

2. Write a JavaScript program to calculate multiplication and division of two numbers (input from user).
   **Challenge: Are there any exceptional cases? If yes, how can we handle them?**

3. Write a JavaScript program which iterates the integers from 1 to 100. But for multiples of three print "Fizz" instead of the number and for the multiples of five print "Buzz". For numbers which are multiples of both three and five print "FizzBuzz".
   **Challenge/Discussion: What if we wanted to do this for 1000 integers as easy as possible?**

4. Addition From Part I, Exercise 3:
   - Create a car object of a Ferrari F430. This should contain the following properties: brand, model, max speed (330 km/h), current speed, status of whether the car has started or not. (….)
   - Describe the car properties of Part 1 exercise 3 in both valid XML and JSON format. Additionally, add an array of its monthly max speeds reached this year:
     - 250km/h for January 2017
     - 300km/h for February 2017
     - 278.5km/h for March 2017.
     - Note: Consider year and month as attributes.

5. Create appropriate form validation in one of your HTML pages using:
   - Built in form validation and HTML5 features
   - JavaScript validation

## Any Questions?

## Self Study

- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#Nested_try-blocks
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch
- https://developer.mozilla.org/en-US/docs/Web/Events
- http://www.javascriptkit.com/jsref/events.shtml
- https://www.w3schools.com/js/js_htmldom_events.asp
- https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation

# References / Further Reading

- https://www.w3schools.com/js/
- https://www.w3schools.com/xml/default.asp
- https://www.w3schools.com/xml/xsl_intro.asp
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Functions
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Control_flow_and_error_handling
- https://www.tutorialspoint.com/javascript/javascript_events.htm
- https://www.w3schools.com/js/js_events.asp
- https://www.w3schools.com/js/js_json_syntax.asp
- https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Working_with_Objects
- https://docs.oracle.com/cd/E19957-01/816-6409-10/obj2.htm
- https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Cross_browser_testing/JavaScript
- Randy Connolly and Ricardo Hoar: Fundamentals of Web Development
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Details_of_the_Object_Model
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch#Nested_try-blocks
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/try...catch
- https://developer.mozilla.org/en-US/docs/Web/Events
- http://www.javascriptkit.com/jsref/events.shtml
- https://www.w3schools.com/js/js_htmldom_events.asp
- https://developer.mozilla.org/en-US/docs/Learn/HTML/Forms/Form_validation

## PeopleCert Values Your Feedback

**Like the course?**
**Have something to say?**

Send us your comments at:
**academic@peoplecert.org**

# Thank You!

For latest news and updates follow us.

**linkedin.com/company/peoplecert-group/**
**twitter.com/peoplecert**
**youtube.com/channel/UCLBidKZS9Xk08f_PM5Edtfg**
**facebook.com/peoplecert.org**