UNIVERSITY of HOUSTON

DEPARTMENT OF COMPUTER SCIENCE

COSC 4351 Fall 2023

Software Engineering

M & W 4 to 5:30 PM

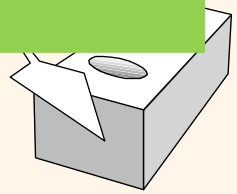
Prof. Victoria Hilford

PLEASE TURN your webcam ON

NO CHATTING during LECTURE

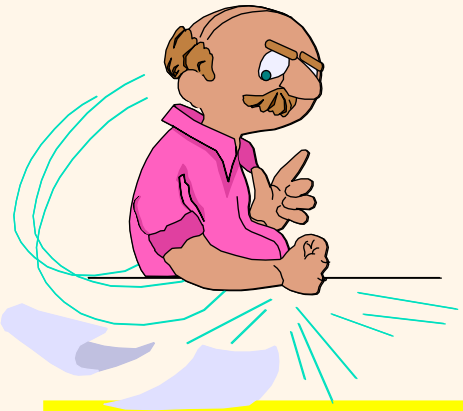
VH, HIDE ZyBook Sections 1 – 5.

From 4:00 to 4:10 PM – 10 minutes.



COSC 4351

4 to 5:30



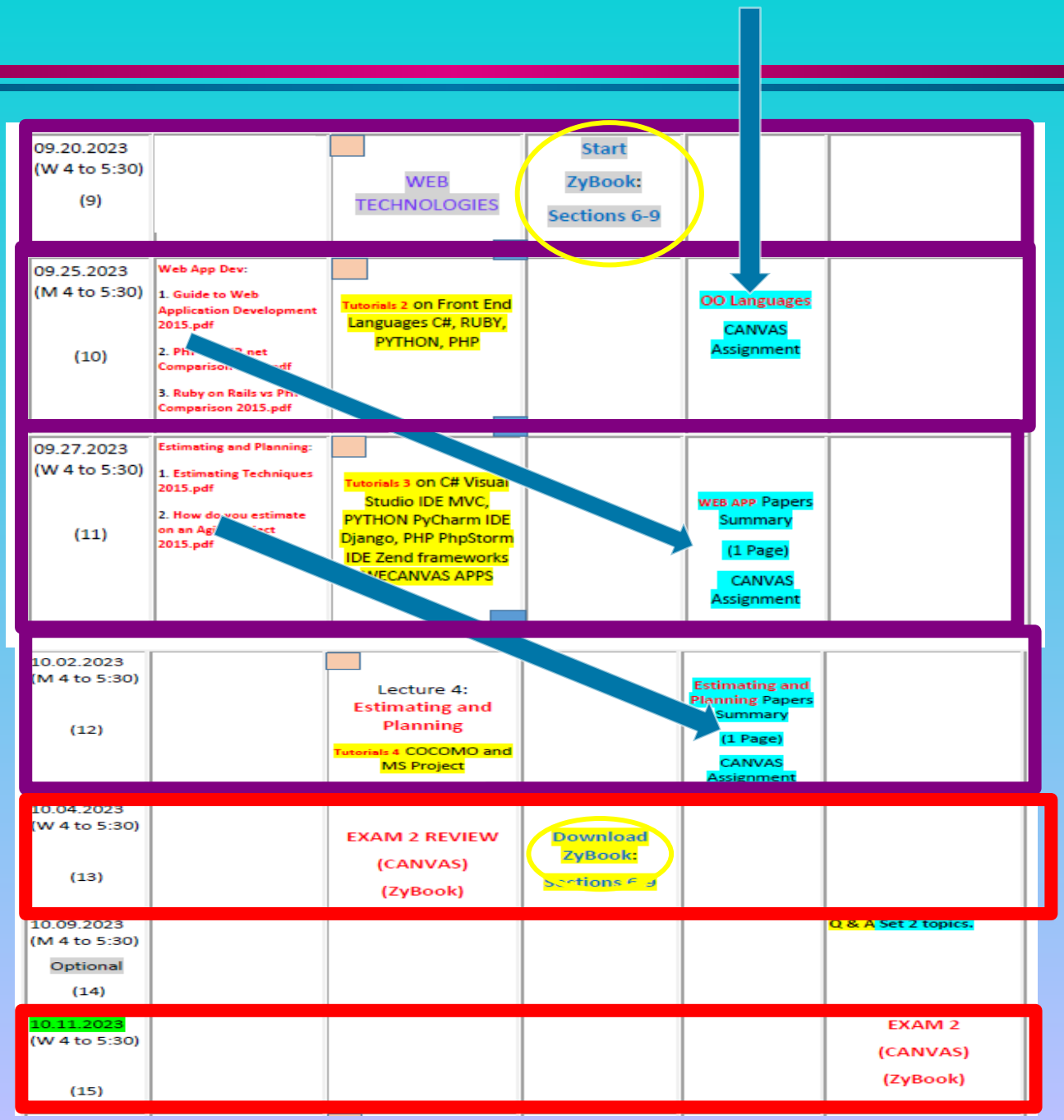
PLEASE

LOG IN

CANVAS

Youyi [A-L] Kevin [M-Z]

Please close all other windows.


09.20.2023
(W 4 to 5:30)
(9)
WEB TECHNOLOGIES
Start ZyBook:
Sections 6-9
09.25.2023
(M 4 to 5:30)
(10)
Web App Dev:
1. Guide to Web Application Development 2015.pdf
3. Ruby on Rails vs Ph... Comparison 2015.pdf
Tutorials 2 on Front End Languages C#, RUBY, PYTHON, PHP
OO Languages
CANVAS Assignment
09.27.2023
(W 4 to 5:30)
(11)
Estimating and Planning:
1. Estimating Techniques 2015.pdf
2. How do you estimate on an Ag... 2015.pdf
Tutorials 3 on C# Visual Studio IDE MVC, PYTHON PyCharm IDE Django, PHP PhpStorm IDE Zend frameworks ...ECANVAS APPS
WEB APP Papers Summary
(1 Page)
CANVAS Assignment
10.02.2023
(M 4 to 5:30)
(12)
Lecture 4:
Estimating and Planning
Tutorials 4 COCOMO and MS Project
Estimating and Planning Papers Summary
(1 Page)
CANVAS Assignment
10.04.2023
(W 4 to 5:30)
(13)
EXAM 2 REVIEW
(CANVAS)
(ZyBook)
Download ZyBook:
Sections 6-9
10.09.2023
(M 4 to 5:30)
Optional
(14)
Q & A Set 2 topics.
10.11.2023
(W 4 to 5:30)
(15)
EXAM 2
(CANVAS)
(ZyBook)

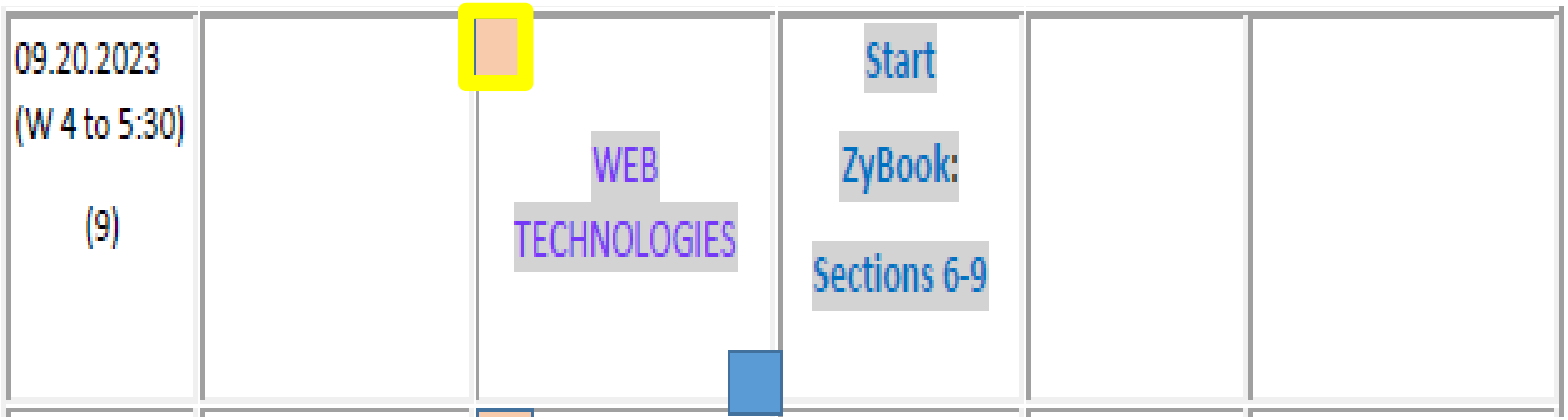
Class 9

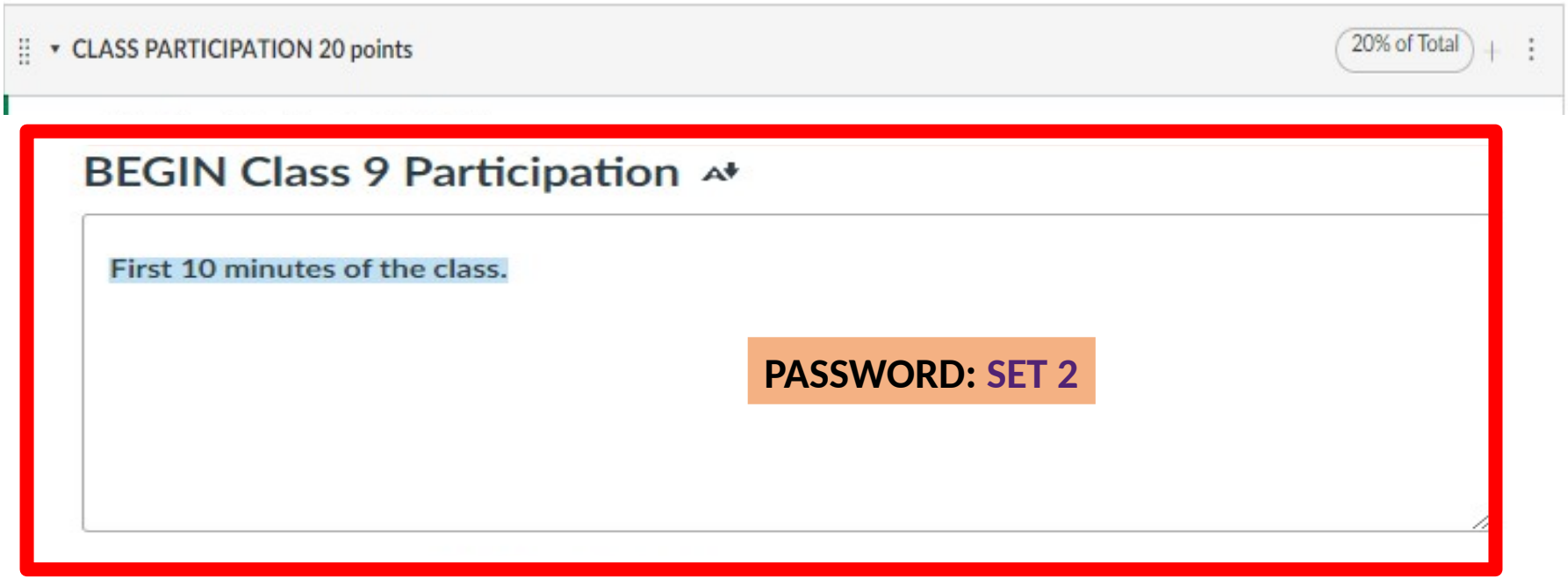
COSC 4351

Software engineering

09.20.2023 (W 4 to 5:30) (9)	Front End Languages: 1. How Do You Learn – Dzone Agile 2016.pdf	WEB TECHNOLOGIES	Start ZyBook: Sections 6-9		

From 4:00 to 4:10 PM – 10 minutes.

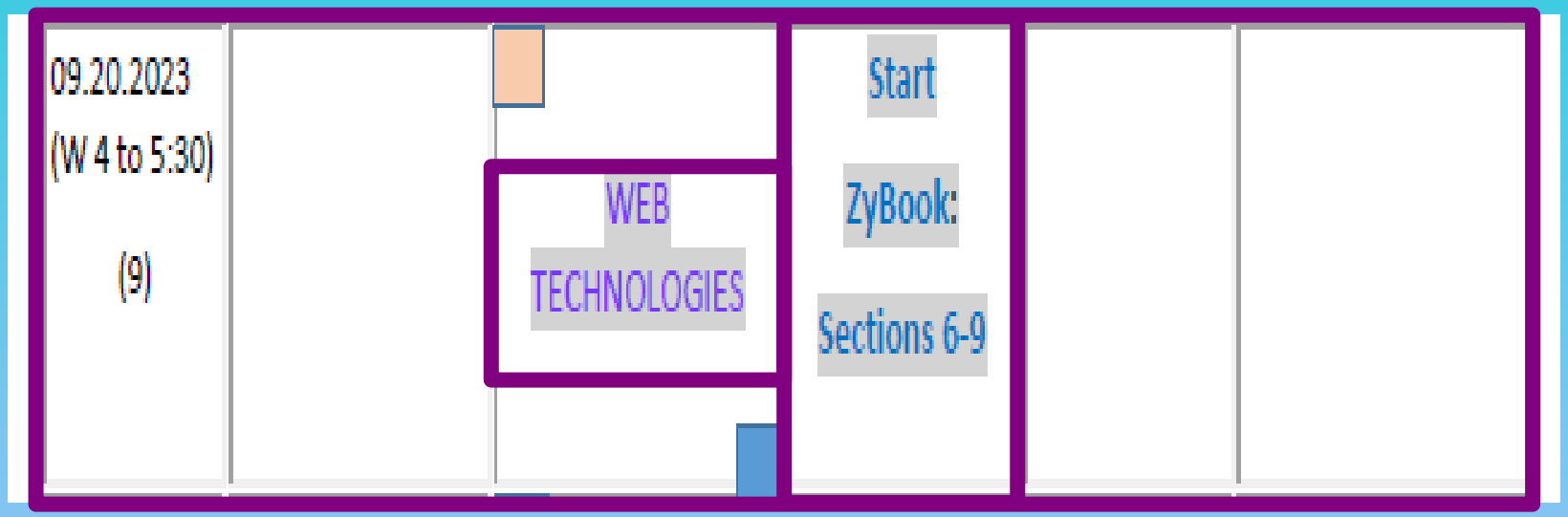




VH– Publish.

From 4:10 to 4:55 PM – 45 minutes.

Sections 6 - 9



Most websites and web applications require the development of client-side technologies that interact with server-side technologies. ***Client-side*** (or ***front-end***) refers to those technologies that run in the web browser like HTML, CSS, and JavaScript. ***Server-side*** (or ***back-end***) refers to those technologies that run on the web server like PHP, Python, Node.js, etc. and databases. Ex: Amazon uses server-side technologies to store information on millions of products and a client-side search interface that interacts with the web server so customers can find and purchase products.

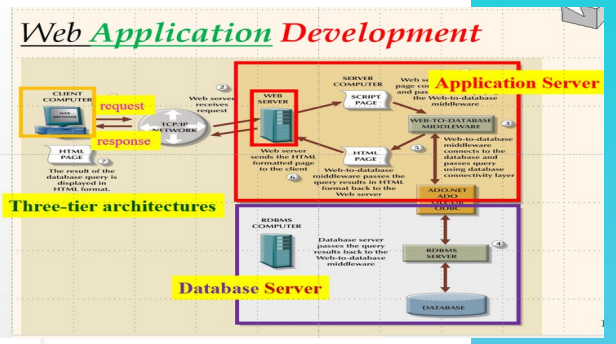
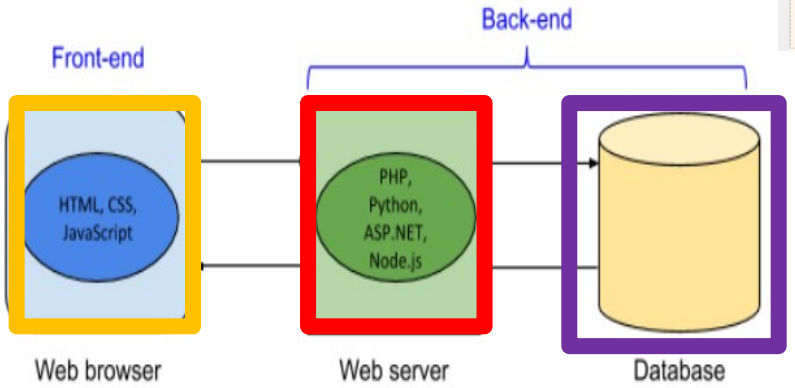


Figure 11.1.1: Front-end and back-end technologies.



A ***front-end developer*** is a developer that is proficient in client-side technologies. A ***back-end developer*** is a developer that is proficient in server-side technologies. Many developers strive to be proficient in both front-end and back-end technologies and how the two sides work together. A ***full-stack developer*** is a developer who has expertise in all aspects of a website or web application's development, including client technologies, server technologies, data modeling, and user interfaces. The "stack" in "full-stack" refers to the various layers that compose websites and web applications. Technology stacks have increased in complexity over the years, so even "full-stack" developers typically specialize in a few areas of the technology stack.

ZyBook – Web Programming

Table of contents
About this material

View full index

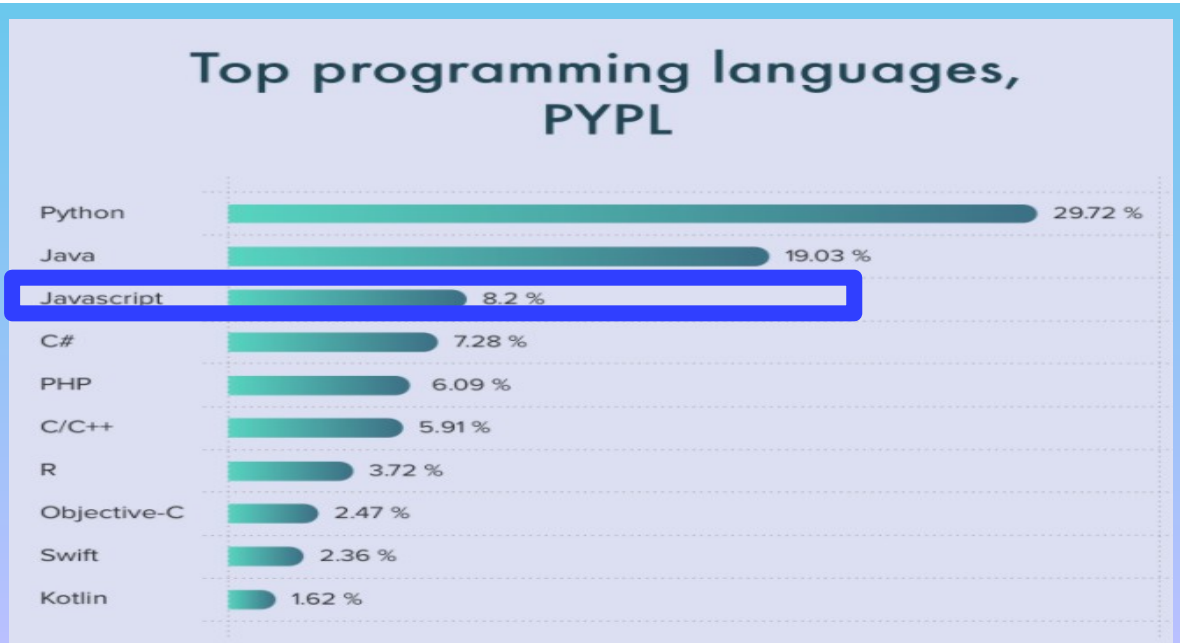
8 sections selected
Clear

- [ ] 1. Introduction to Web Programming
- [ ] 2. HTML
- [ ] 3. More HTML
- [ ] 4. Basic CSS
- [ ] 5. Advanced CSS
- [ ] 6. Basic JavaScript
- [ ] 7. JavaScript in the Browser
- [ ] 8. Advanced JavaScript
- [ ] 9. jQuery
- [ ] 10. Mobile Web Development
- [ ] 11. Node.js
- [ ] 12. PHP
- [ ] 13. Advanced PHP
- [ ] 14. Relational Databases and SQL

In 1995, Brendan Eich created JavaScript so the Netscape Navigator browser could dynamically respond to user events. Ex: The web page's content could change when the user clicked a button or hovered over an image. JavaScript was standardized by Ecma International in 1997 and called ECMAScript. The ***ECMAScript*** standard has been improved over the years, and JavaScript is an implementation of ECMAScript. The latest version of ECMAScript is version 11 (***ECMAScript 2020***).

Today, JavaScript is one of the most popular programming languages. JavaScript is supported by every major web browser and makes web applications like Gmail and Google Maps possible. JavaScript is also popular outside the web browser. Ex: Node.js, which runs JavaScript, is a popular technology for creating server-side web applications.

JavaScript is executed by an interpreter. An ***interpreter*** executes programming statements without first compiling the statements into machine language. Modern JavaScript interpreters (also called ***JavaScript engines***) use ***just-in-time (JIT) compilation*** to compile the JavaScript code at execution time into another format that can be executed quickly.



2022

- [ ] 6. Basic JavaScript 0% 0% 0%
  - [ ] 6.1 Syntax and variables 4% 4%
  - [ ] 6.2 Arithmetic 4% 4%
  - [ ] 6.3 Conditionals 0% 0%
  - [ ] 6.4 More conditionals 0%
  - [ ] 6.5 Loops 0% 0%
  - [ ] 6.6 Functions 0% 0%
  - [ ] 6.7 Scope and the global object 0%
  - [ ] 6.8 Arrays 0% 0%
  - [ ] 6.9 Objects 0% 0%
  - [ ] 6.10 Maps 0%
  - [ ] 6.11 String object 0% 0%
  - [ ] 6.12 Date object 0% 0%
  - [ ] 6.13 Math object 0%
  - [ ] 6.14 Exception handling 0% 0%

6. Basic JavaScript

6.1 Syntax and variables

Variables

A ***variable*** is a named container that stores a value in memory. A ***variable declaration*** is a statement that declares a new variable with the keyword ***let*** followed by the variable name. Ex: `let score` declares a variable named score.

A variable may be assigned a value after being declared. An ***assignment*** assigns a variable with a value, like `score = 2`. A variable may also be assigned a value on the same line when the variable is declared, which is called ***initializing*** the variable. Ex: `let maxValue = 5;` initializes `maxValue` to 5.

A variable may be assigned a value without first declaring the variable, but good practice is to always declare a variable before assigning a value to the variable.

PARTICIPATION ACTIVITY 6.1.2: Declaring variables and assigning values.

let

```
// Declaring a variable
let numSongs;

// Variable is assigned a number
numSongs = 5;

// Variable is declared and assigned a number (initialized)
let numAlbums = 20;

// Variable may be assigned a value without first being declared
hitCount = 10;
```

memory

numSongs	5
numAlbums	20
hitCount	10

6. Basic JavaScript

6.1 Syntax and variables

Data types

Variables are not explicitly assigned a data type. JavaScript uses ***dynamic typing***, which determines a variable's type at run-time. Every variable has a data type, such as one of the data types in the table below.

Table 6.1.1: Example JavaScript data types.

Data type	Description	Example
string	Group of characters delimited with 'single' or "double" quotes	`let name = "Sam";` `let quote = 'The computer asked, "Shall we play a game?"';`
number	Numbers with or without decimal places	`let highScore = 950;` `let pi = 3.14;`
boolean	true or false	`let hungry = true;` `let thirsty = false;`
array	List of items	`let teams = ["Broncos", "Cowboys", "49ers"];`
object	Collection of property and value pairs	`let movie = {title:"Life is Beautiful", rating:"PG-13"};`
undefined	Variable that has not been assigned a value	`let message;`
null	Intentionally absent of any object value	`let book = null;`

6. Basic JavaScript

6.1 Syntax and variables

Input and output

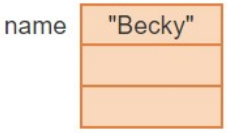
A JavaScript program may obtain text input from the user with the `prompt()` function. The ***prompt()*** function prompts the user with a dialog box that allows the user to type a single line of text and press OK or Cancel. The `prompt()` function returns the string the user typed or `null` if the user pressed Cancel.

Output may be produced using the function ***console.log()***, which displays text or numbers in the console. The ***console*** is a location where text output is displayed. Web browsers have a console (accessible from the brower's development tools) that displays output from code the browser executes. This chapter's activities display the console output in the web page.

PARTICIPATION ACTIVITY | 6.1.6: Prompting for input and displaying output.

```
// Display the prompt dialog box
let name = prompt("What is your name?");

// Output to the console
console.log("Hello, " + name + "!");
```



6. Basic JavaScript

6.2 Arithmetic

Arithmetic operators

An ***expression*** is a combination of items like variables, numbers, operators, and parentheses, that evaluates to a value like 2 * (x + 1). Expressions are commonly used on the right side of an assignment statement, as in `y = 2 * (x + 1)`.

An ***arithmetic operator*** is used in an expression to perform an arithmetic computation. Ex: The arithmetic operator for addition is +. JavaScript arithmetic operators are summarized in the table below.

Table 6.2.1: JavaScript arithmetic operators.

Arithmetic operator	Description	Example
+	Add	`// x = 3` `x = 1 + 2;`
-	Subtract	`// x = 1` `x = 2 - 1;`
*	Multiply	`// x = 6` `x = 2 * 3;`
/	Divide	`// x = 0.5` `x = 1 / 2;`
%	Modulus (remainder)	`// x = 0` `x = 4 % 2;`
**	Exponentiation	`// x = 2 * 2 * 2 = 8` `x = 2 ** 3;`
++	Increment	`// Same as x = x + 1` `x++;`
--	Decrement	`// Same as x = x - 1` `x--;`

- [ ] 6. Basic JavaScript
  - [ ] 6.2 Arithmetic

Arithmetic with numbers and strings

The + operator is also the string concatenation operator. ***String concatenation*** appends one string after the end of another string, forming a single string. Ex: `"back" + "pack"` is `"backpack"`.

The JavaScript interpreter determines if + means "add" or "concatenate" based on the operands on either side of the operator. An ***operand*** is the value or values that an operand works on, like the number 2 or variable x.

- If both operands are numbers, + performs addition. Ex: 2 + 3 = 5.
- If both operands are strings, + performs string concatenation. Ex: "2" + "3" = "23".
- If one operand is a number and the other a string, + performs string concatenation. The number is converted into a string, and the two strings are concatenated into a single string. Ex: "2" + 3 = "2" + "3" = "23".

For all other arithmetic operators, combining a number and a string in an arithmetic expression converts the string operand to a number and then performs the arithmetic operation. Ex: "2" * 3 = 2 * 3 = 6.

CUTE!

- [ ] 6. Basic JavaScript
- [ ] 6.3 Conditionals

If statement

An ***if statement*** executes a group of statements if a condition is true. Braces { } surround the group of statements. Good practice is to indent statements in braces using a consistent number of spaces. This material indents 3 spaces.

Construct 6.3.1: if statement.

```
if (condition) {
   // Statements to execute when condition is true
}
```

If-else statement

An ***if-else statement*** executes a block of statements if the statement's condition is true, and executes another block of statements if the condition is false.

Construct 6.3.2: if-else statement.

```
if (condition) {
   // statements to execute when condition is true
}
else {
   // statements to execute when condition is false
}
```

6. Basic JavaScript

6.3 Conditionals

Table 6.3.1: Comparison operators.

Comparison operator	Name	Example
==	Equality	`2 == 2` // true `"bat" == "bat"` // true
!=	Inequality	`2 != 3` // true `"bat" != "zoo"` // true
===	Identity	`2 === 2` // true `"2" === 2` // false
!==	Non-identity	`2 !== 2` // false `"2" !== 2` // true
		`2 < 3` // true

When the equality operator `==` and inequality `!=` operator compare a number and a string, the string is first converted to a number and then compared. Ex: `3 == "3"` is true because "3" is converted to 3 before the comparison, and 3 and 3 are the same.

The ***identity operator*** `===` performs ***strict equality***. Two operands are strictly equal if the operands' data types and values are equal. Ex: `3 === 3` is true because both operands are numbers and the same value, but `"3" === 3` is false because "3" is a string, and 3 is a number.

The ***non-identity operator*** `!==` is the opposite of the identity operator. Ex: `"3" !== "3"` is false because both operands are the same type and value, but `"3" !== 3` is true because "3" is a string, and 3 is a number.

Other comparison operators also convert a string to a number when comparing a string with a number. Ex: `2 < "12"` is true because 2 is less than the number 12. When comparing two strings, JavaScript uses Unicode values to compare characters. Ex: `"cat" <= "dog"` is true because "c" has a smaller Unicode value than "d".

A common error when comparing two values for equality is to use a single `=` instead of `==` or `===`. Ex: `if (name = "Sue")` assigns `name` with "Sue" instead of asking if `name` equals "Sue".

6. Basic JavaScript

6.4 More conditionals

Truthy and falsy

A ***truthy*** value is a non-Boolean value that evaluates to `true` in a Boolean context. Ex: `if (3)` evaluates to `true` because non-zero numbers are truthy values. A ***falsy*** value is a non-Boolean value that evaluates to `false` in a Boolean context. Ex: `if (null)` evaluates to `false` because `null` is a falsy value.

Table 6.4.1: Truthy values.

Example	Description
`if (32)`	Non-zero number
`if ("cat")`	Non-empty string
`if (myObject)`	Object variable
`if (myArray)`	Array variable

Feedback?

Table 6.4.2: Falsy values.

Example	Description
`if (0)`	Zero
`if ("")`	Empty string
`if (NaN)`	Not a number
`if (undefined)`	Variable that has not been assigned a value
`if (null)`	No object value

Feedback?

6.7 Scope and the global object

The var keyword and scope

In addition to declaring variables with `let`, a variable can be declared with the ***var*** keyword. Ex: `var x = 6;` declares the variable x with an initial value of 6. When JavaScript was first created, `var` was the only way to declare a variable. The `let` keyword was added to JavaScript in 2015.

Both `let` and `var` declare variables but with differing scope. A JavaScript variable's ***scope*** is the context in which the variable can be accessed.

A variable declared inside a function has ***local scope***, so only the function that defines the variable has access to the ***local variable***. A variable declared outside a function has ***global scope***, and all functions have access to a ***global variable***.

A variable declared inside a function with `var` is scoped to that function: the variable is accessible anywhere within the function, but not outside. A variable declared inside a function with `let` has ***block scope***: the variable is accessible only within the enclosing pair of braces.

A variable declared using `var` or `let` that is not inside a function creates a global variable that is accessible from anywhere in the code.

PARTICIPATION ACTIVITY 6.7.1: var vs. let scoping.

var

x GLOBAL BAD!

Function using var

```
var x = 17;

function numbers() {
 console.log(x);          → 17
 if (x > 0) {
    var y = x / 2;
    console.log(y);       → 8.5
 }
 if (x < 100) {
    var z = x * 2;
    console.log(z);       → 34
 }

 console.log(y);          → 8.5
 console.log(z);          → 34
}

numbers();
console.log(x);           → 17

console.log(y);
console.log(z);           → ReferenceError
```

let

y & z ARE LOCAL!

Function using let

```
let x = 17;

function numbers() {
 console.log(x);          → 17
 if (x > 0) {
    let y = x / 2;
    console.log(y);       → 8.5
 }
 if (x < 100) {
    let z = x * 2;
    console.log(z);       → 34
 }

 console.log(y);
 console.log(z);
}

numbers();
console.log(x);           → 17

console.log(y);
console.log(z);           → ReferenceError
```

- [ ] 6. Basic JavaScript
- [ ] 6.5 Loops
- [ ] 6.6 Functions
- [ ] 6.7 Scope and the global object
- [ ] 6.8 Arrays

Array introduction

An ***array*** is an ordered collection of values called ***elements***. Each array element is stored in a numeric location called an ***index***. An array is initialized by assigning an array variable with brackets `[]` containing comma-separated values.

Array elements may be of the same type or different types. Arrays increase in size as elements are added and decrease as elements are removed.

PARTICIPATION ACTIVITY | 6.8.1: Initializing and displaying array elements.

```
let scores = [];

scores[0] = 6;
scores[1] = 15;
scores[2] = 8;

console.log(scores[0]);
console.log(scores[1]);
console.log(scores[2]);
```

```
let teams = ["Tigers", "Bisons",
             "Eagles", "Cobras"];

console.log(teams);
```

	scores
0	6
1	15
2	8

	teams
0	Tigers
1	Bisons
2	Eagles
3	Cobras

```
6
15
8
Tigers,Bisons,Eagles,Cobras
```

- [ ] 6. Basic JavaScript
- [ ] 6.5 Loops
- [ ] 6.6 Functions
- [ ] 6.7 Scope and the global object
- [ ] 6.8 Arrays

Adding and removing array elements

An array is an ***Array object*** that defines numerous methods for manipulating arrays. A ***method*** is a function that is attached to an object and operates on data stored in the object. Methods are called by prefacing the method with the object. Ex: `myArray.method();`.

Table 6.8.1: Array methods for adding and removing array elements.

Method	Description	Example
`push(value)`	Adds a value to the end of the array	`let nums = [2, 4, 6];` `nums.push(8);  // nums = [2, 4, 6, 8]`
`pop()`	Removes the last array element and returns the element	`let nums = [2, 4, 6];` `let x = nums.pop();  // returns 6, nums = [2, 4]`
`unshift(value)`	Adds a value to the beginning of the array	`let nums = [2, 4, 6];` `nums.unshift(0);  // nums = [0, 2, 4, 6]`
`shift()`	Removes the first array element and returns the element	`let nums = [2, 4, 6];` `let x = nums.shift();  // returns 2, nums = [4, 6]`

`splice(startingIndex, totalElementsToDelete, valuesToAdd)`	Adds or removes elements from anywhere in the array and returns the deleted elements (if any)	`let nums = [2, 4, 6, 8, 10];`
		
		`// Deletes all elements from index 3 to the end`
		`nums.splice(3);  // nums = [2, 4, 6]`
		
		`// Deletes 2 elements starting at index 0`
		`nums.splice(0, 2);  // nums = [6]`
		
		`// Adds 3, 5 starting at index 0`
		`nums.splice(0, 0, 3, 5);  // nums = [3, 5, 6]`
		
		`// Adds 7, 9, 11 starting at index 2`
		`nums.splice(2, 0, 7, 9, 11);  // nums = [3, 5, 7, 9, 11, 6]`

6. Basic JavaScript

6.9 Objects

Objects and properties

An ***object*** is an unordered collection of properties. An object ***property*** is a name-value pair, where the name is a string and the value is any data type. Objects are often defined with an object literal. An ***object literal*** (also called an ***object initializer***) is a comma-separated list of property name and value pairs.

PARTICIPATION ACTIVITY | 6.9.1: Creating an object with an object literal.

```
let book = {};

book = {
   title: "Outliers",
   published: 2011,
   keywords: ["success", "high-achievers"]
};
```

```
console.log(book.title);
console.log(book.keywords[0]);
```

```
book = {
   title: "Outliers",
   published: 2011,
   keywords: ["success", "high-achievers"],
   author: {
      firstName: "Malcolm",
      lastName: "Gladwell"
   }
};
```

```
console.log(book.author.lastName);
```

Outliers
success
Gladwell

Accessor properties

An object property may need to be computed when retrieved, or setting a property may require executing some code to perform data validation. The `get` and `set` keywords define getters and setters for a property. A ***getter*** is a function that is called when an object's property is retrieved. Syntax to define a getter: `get property() { return someValue; }`. A ***setter*** is a function that is called when an object's property is set to a value. Syntax to define a setter: `set property(value) { ... }`. An ***accessor property*** is an object property that has a getter or a setter or both.

Figure 6.9.3: Defining an accessor property called 'area'.

```
let rectangle = {
   width: 5,
   height: 8,
   get area() {
      return this.width * this.height;
   }
   set area(value) {
      // Set width and height to the square root of the value
      this.width = Math.sqrt(value);
      this.height = this.width;
   }
};

let area = rectangle.area;     // Calling getter returns 40
rectangle.area = 100;          // Calling setter sets width and height to 10
console.log(rectangle.width);  // 10
```

6. Basic JavaScript

6.10 Maps

A ***map*** or ***associative array*** is a data structure that maps keys to values. Each key/value pair in a map is called an ***element***. JavaScript objects can be used as maps, in which the key is the object property and the value is the property's value.

The for-in loop is ideal for looping through an object map. The ***for-in loop*** iterates over an object's properties in arbitrary order.

PARTICIPATION ACTIVITY 6.10.1: State capitals in an object map.

```
let stateCapitals = {
   AR: "Little Rock",
   CO: "Denver",
   NM: "Sante Fe"
};

console.log("CO capital is " + stateCapitals["CO"]);

stateCapitals["TX"] = "Austin";

console.log("All capitals:");
for (let state in stateCapitals) {
   console.log(state + " is " + stateCapitals[state]);
}
```

stateCapitals

AR	Little Rock
CO	Denver
NM	Santa Fe
TX	Austin

```
CO capital is Denver
All capitals:
AR is Little Rock
CO is Denver
NM is Santa Fe
TX is Austin
```

Map object

The ***Map object*** is a newer alternative to using objects for storing key/value pairs. Common methods and properties of the `Map` object include:

- The ***set(key, value)*** method sets a key/value pair. If the key is new, a new element is added to the map. If the key already exists, the new value replaces the existing value.
- The ***get(key)*** method gets a key's associated value.
- The ***has(key)*** method returns true if a map contains a key, false otherwise.
- The ***delete(key)*** method removes a map element.
- The ***size*** property is the number of elements in the map.

The for-of loop, which is often used to loop through an array, is ideal for looping through a `Map`. Each of the map's key/value pairs are assigned to the `[key, value]` variables declared in the for-of loop, as illustrated in the animation below.

- [ ] 6. Basic JavaScript
  - [ ] 6.11 String object
  - [ ] 6.12 Date object
  - [ ] 6.13 Math object
  - [ ] 6.14 Exception handling

- [ ] 7. JavaScript in the Browser
  - [ ] 7.1 Using JavaScript with HTML
  - [ ] 7.2 Document Object Model (DOM)
  - [ ] 7.3 More DOM modification
  - [ ] 7.4 Event-driven programming
  - [ ] 7.5 Timers
  - [ ] 7.6 Modifying CSS with JavaScript
  - [ ] 7.7 Form validation
  - [ ] 7.8 JavaScript Object Notation (JSON) ~~XML~~
  - [ ] 7.9 XMLHttpRequest (Ajax)
  - [ ] 7.10 Using third-party web APIs (JavaScript)
  - [ ] 7.11 Browser differences: JavaScript

- [ ] 7. JavaScript in the Browser
  - [ ] 7.1 Using JavaScript with HTML
  - [ ] 7.2 Document Object Model (DOM)

7.2 Document Object Model (DOM)

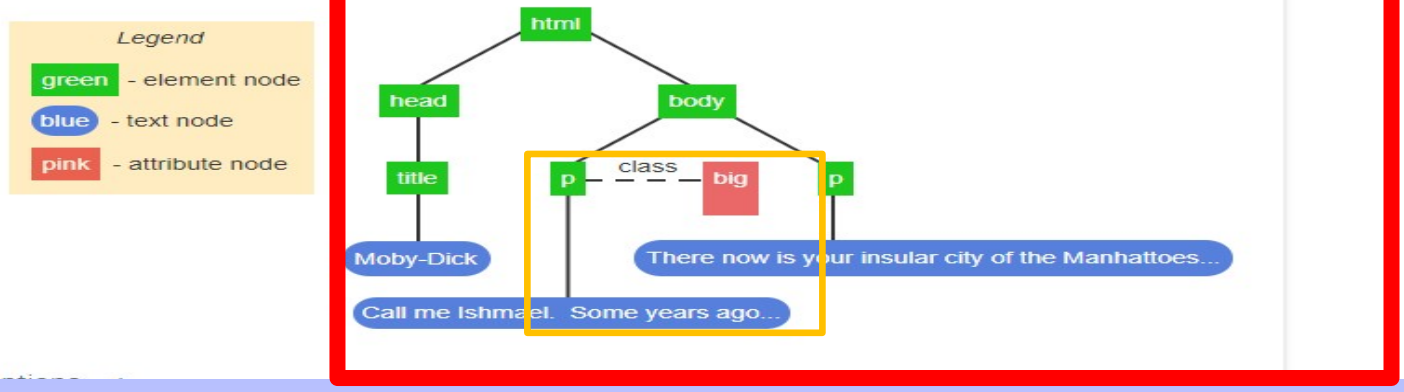
DOM structure

The Document Object Model (DOM) is a data structure corresponding to the HTML document displayed in a web browser. A DOM tree is a visualization of the DOM data structure. A ***node*** is an individual object in the DOM tree. Nodes are created for each element, the text between an element's tags, and the element's attributes.

- The ***root node*** is the node at the top of the DOM.
- A ***child node*** is the node directly under another node. A node can have zero, one, or more child nodes (children).
- A ***parent node*** is the node directly above another node. All nodes, except the root node, have one parent node.

PARTICIPATION ACTIVITY | 7.2.1: Creating the DOM from HTML.

```
<html>
  <title>Moby-Dick</title>
  <body>
    <p class="big">Call me Ishmael.  Some years ago...</p>
    <p>There now is your insular city of the Manhattoes...</p>
  </body>
</html>
```



7. JavaScript in the Browser

7.1 Using JavaScript with HTML

7.2 Document Object Model (DOM)

Searching the DOM

JavaScript is commonly used to search the DOM for a specific node or set of node
an email application, the user may click a Delete button to delete an email. The Jav
the email's contents and then change the contents to read "Email deleted".

PARTICIPATION ACTIVITY | 7.2.3: Searching the DOM by id and by element.

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <title>Early Programming Languages</title>
      <script src="dom.js" defer></script>
   </head>
   <body>
      <p id="early_languages">
         Early programming languages still in use:
      </p>

      <ol class="traditional">
         <li><a href="https://w.wiki/4ZAb">Fortran - 1954</a></li>
         <li><a href="https://w.wiki/4NzE">Lisp - 1958</a></li>
         <li>COBOL - 1959</li>
         <li>BASIC - 1964</li>
      </ol>
   </body>
</html>
```

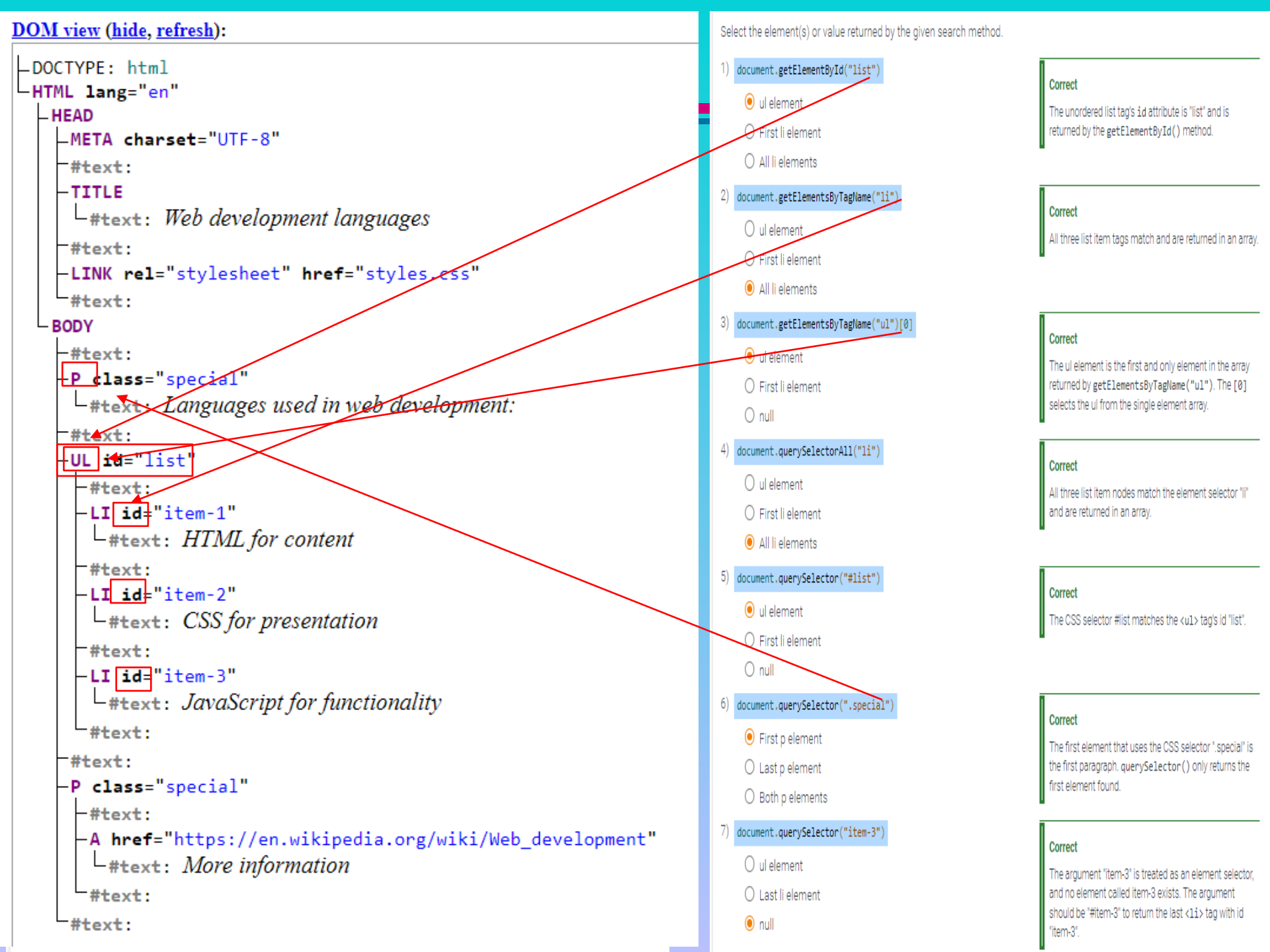
dom.js

```
let para = document.getElementById("early_languages");
let listItems = document.querySelectorAll("li");
```

The `document` object provides five primary methods that search the DOM for specific nodes:

1. The ***document.getElementById()*** method returns the DOM node whose `id` attribute is the same as the method's parameter.
   Ex: `document.getElementById("early_languages")` returns the p node in the HTML below.
2. The ***document.getElementsByTagName()*** method returns an array of all the DOM nodes whose type is the same as the method's parameter.
   Ex: `document.getElementsByTagName("li")` returns a list of the four li nodes from in the HTML below.
3. The ***document.getElementsByClassName()*** method returns an array containing all the DOM nodes whose `class` attribute matches the method's parameter.
   Ex: `document.getElementsByClassName("traditional")` returns an array containing the `ol` node with the `class` attribute matching the word traditional.
4. The ***document.querySelectorAll()*** method returns an array containing all the DOM nodes that match the CSS selector passed as the method's parameter.
   Ex: `document.querySelectorAll("li a")` returns an array containing the two anchor nodes in the HTML below.
5. The ***document.querySelector()*** method returns the first element found in the DOM that matches the CSS selector passed as the method's parameter. `querySelector()` expects the same types of parameters as `querySelectorAll()` but only returns the first element found while navigating the DOM tree in a depth-first traversal.
   Ex: `document.querySelector("li")` returns the li node about Fortran.

A DOM search method name indicates whether the method returns one node or an array of nodes. If the method name starts with "getElements" or ends in "All", then the method returns an array, even if the array contains one node or is empty. `getElementById()` and `querySelector()` either return a single node or null if no node matches the method arguments.


DOM view (hide, refresh):
DOCTYPE: html
HTML lang="en"
HEAD
META charset="UTF-8"
#text:
TITLE
#text: Web development languages
#text:
LINK rel="stylesheet" href="styles.css"
#text:
BODY
#text:
P class="special"
#text: Languages used in web development:
#text:
UL id="list"
#text:
LI id="item-1"
#text: HTML for content
#text:
LI id="item-2"
#text: CSS for presentation
#text:
LI id="item-3"
#text: JavaScript for functionality
#text:
#text:
P class="special"
#text:
A href="https://en.wikipedia.org/wiki/Web_development"
#text: More information
#text:
#text:
Select the element(s) or value returned by the given search method.
1) document.getElementById("list")
ul element
First li element
All li elements
2) document.getElementsByTagName("li")
ul element
First li element
All li elements
3) document.getElementsByTagName("ul")[0]
ul element
First li element
null
4) document.querySelectorAll("li")
ul element
First li element
All li elements
5) document.querySelector("#list")
ul element
First li element
null
6) document.querySelector(".special")
First p element
Last p element
Both p elements
7) document.querySelector("item-3")
ul element
Last li element
null
Correct
The unordered list tag's id attribute is "list" and is returned by the getElementById() method.
Correct
All three list item tags match and are returned in an array.
Correct
The ul element is the first and only element in the array returned by getElementsByTagName("ul"). The [0] selects the ul from the single element array.
Correct
All three list item nodes match the element selector "li" and are returned in an array.
Correct
The CSS selector #list matches the <ul> tag's id "list".
Correct
The first element that uses the CSS selector ".special" is the first paragraph. querySelector() only returns the first element found.
Correct
The argument "item-3" is treated as an element selector, and no element called item-3 exists. The argument should be "#item-3" to return the last <li> tag with id "item-3".

7. JavaScript in the Browser

7.4 Event-driven programming

The following are events for which web developers commonly write handlers.

- A ***change*** event is caused by an element value being modified. Ex: Selecting an item in a radio button group causes a change event.
- An ***input*** event is caused when the value of an input or textarea element is changed.
- A ***load*** event is caused when the browser completes loading a resource and dependent resources. Usually load is used with the body element to execute code once all the web page's CSS, JavaScript, images, etc. have finished loading.
- A ***DOMContentLoaded*** event is caused when the HTML file has been loaded and parsed, although other related resources such as CSS, JavaScript, and image files may not yet be loaded.
- A ***focus*** event is caused when an element becomes the current receiver of keyboard input. Ex: Clicking in an input field causes a focus event.
- A ***blur*** event is caused when an element loses focus and the element will no longer receive future keyboard input.
- A ***submit*** event is caused when the user submits a form to the web server.

☐ 7. JavaScript in the Browser

☐ 7.4 Event-driven programming

Registering event handlers

Handlers are written in three ways:

1. Embedding the handler as part of the HTML. Ex: `<button onclick="clickHandler()">Click Me</button>` sets the click event handler for the button element by using the **`onclick`** attribute. The attribute name used to register the handler adds the prefix "on" to the event name. Ex: The attribute for a mousemove event is **`onmousemove`**. Embedding a handler in HTML mixes content and functionality and thus should be avoided whenever possible.
2. Setting the DOM node event handler property directly using JavaScript. Ex: **`document.querySelector("#myButton").onclick = clickHandler`** sets the click event handler for the element with an id of "myButton" by overwriting the **`onclick`** JavaScript property. Using DOM node properties is better than embedding handlers within the HTML but has the disadvantage that setting the property only allows one handler for that element to be registered.
3. Using the JavaScript ***addEventListener()*** method to register an event handler for a DOM object. Ex: **`document.querySelector("#myButton").addEventListener("click", clickHandler)`** registers a click event handler for the element with the id "myButton". Good practice is to use the **`addEventListener()`** method whenever possible, rather than using element attributes or overwriting JavaScript properties. The **`addEventListener()`** method allows for separation of content and functionality and allows multiple handlers to be registered with an element for the same event.

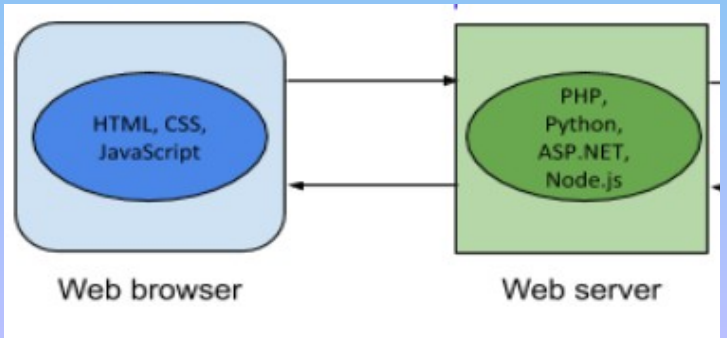
Every handler has an optional ***event object*** parameter that provides details of the event. Ex: For a keypress event, the event object indicates which key was pressed, or for a click event, which element was clicked.

In the animation below, **`keypressHandler()`** uses **`event.target`** to access the text box object where the keypress event occured. Inside an event handler, the **`this`** keyword refers to the element to which the handler is attached. So **`event.target`** and **`this`** both refer to the text box object in the event handler.

7. JavaScript in the Browser

7.8 JavaScript Object Notation (JSON)

Communicating data between the server and browser is a significant task for modern web applications. Initial attempts to do so included unstructured text documents and heavily structured XML documents, both of which required significant effort to convert to a usable format. **JavaScript Object Notation**, or **JSON**, is an efficient, structured format for data based on a subset of the JavaScript language. JSON (pronounced "Jason") is intended to be easily readable by humans and computers. Debugging communication that uses JSON is easy because humans can read JSON. Communication is efficient because computers can transmit and parse JSON quickly. As a result, JSON has rapidly become the dominant format of data transfer between web browsers and servers.



XML Vs JSON

```xml
<users>
    <user>
        <name>John</name>
        <age>22</age>
    </user>
    <user>
        <name>Mary</name>
        <age>21</age>
    </user>
</users>
```

```json
var users=[
        {
            name:"John",
            age:22
        },
        {
            name:"Mary",
            age:21
        }
    ];
```

- [ ] 8. Advanced JavaScript
  - [ ] 8.1 Regular expressions
  - [ ] 8.2 Classes
  - [ ] 8.3 Classes (ES6)
  - [ ] 8.4 Inner functions, outer functions, and function scope
  - [ ] 8.5 Closures
  - [ ] 8.6 Strict mode
  - [ ] 8.7 Web storage
  - [ ] 8.8 Canvas drawing
  - [ ] 8.9 Canvas transformations and animation
  - [ ] 8.10 WebSockets

8. Advanced JavaScript

8.2 Classes

Constructor functions

A JavaScript ***class*** is a special function, called a constructor function, that defines properties and methods from which an object may inherit. A ***constructor function*** is a function that initializes a new object when an object is instantiated with the `new` operator. The ***this*** keyword refers to the current object and is used to access properties inside the class.

PARTICIPATION ACTIVITY 8.2.1: Creating a Person class with a constructor function.

```
function Person(name, age) {
  this.name = name;
  this.age = age;

  this.sayHello = function() {
     console.log("Hello, " + this.name);
  };
};

let bob = new Person("Bob", 21);
let sue = new Person("Sue", 40);

bob.sayHello();
sue.sayHello();
```

bob	name: "Bob"	age: 21	sayHello: func
sue	name: "Sue"	age: 40	sayHello: func

```
Hello, Bob
Hello, Sue
```

- [ ] 9. jQuery
  - [ ] 9.1 Getting started
  - [ ] 9.2 Selectors
  - [ ] 9.3 Events
  - [ ] 9.4 Styles and animation
  - [ ] 9.5 DOM manipulation
  - [ ] 9.6 Ajax
  - [ ] 9.7 Using third-party web APIs (jQuery)
  - [ ] 9.8 Plugins

- [ ] 7. JavaScript in the Browser
  - [ ] 7.1 Using JavaScript with HTML
  - [ ] 7.2 Document Object Model (DOM)
  - [ ] 7.3 More DOM modification
  - [ ] 7.4 Event-driven programming
  - [ ] 7.5 Timers
  - [ ] 7.6 Modifying CSS with JavaScript
  - [ ] 7.7 Form validation
  - [ ] 7.8 JavaScript Object Notation (JSON) ~~XML~~
  - [ ] 7.9 XMLHttpRequest (Ajax)
  - [ ] 7.10 Using third-party web APIs (JavaScript)
  - [ ] 7.11 Browser differences: JavaScript

9. jQuery

JavaScript libraries

Writing JavaScript that creates dynamic, interactive web pages that work across all browsers can be very tedious, often requiring many lines of brittle code. Developers typically rely on libraries to ease the burden of writing such code. A ***library*** is a collection of functions that focus on a related set of tasks. ***jQuery*** is a popular JavaScript library that focuses on a broad range of tasks, many of them associated with the visual elements of a web page.

Library vs. framework

New JavaScript developers are sometimes confused about the difference between a "library" and a "framework". A ***framework*** *is a suite of libraries designed to offer a more comprehensive platform in which to program. When using a framework, the program's flow is dictated by the framework, not the programmer. Examples of popular JavaScript frameworks include AngularJS, Ember, and Backbone.js.*

Accessing the jQuery library

The jQuery library can be obtained from jquery.com. Version 3 is the latest version. After downloading the library, developers often place the library in a standard location on their web server. Web pages that use jQuery import the library as shown in the figure below. The filename contains a version number (3.5.1), and the ".min" means the code has been minified to download quicker.

Figure 9.1.1: Downloading jQuery library from the local web server.

```
<script src="jquery-3.5.1.min.js"></script>
```

Feedback?

Another option is to write web pages that download the jQuery library from a CDN. A ***Content Delivery Network (CDN)*** hosts popular web files around the globe and automatically routes requests to the closest server, thus speeding up the delivery of the files. The figure below shows how to import the jQuery library from a CDN. The ***integrity*** and ***crossorigin*** attributes are used for ***Subresource Integrity*** (***SRI***) checking, which allows web browsers to verify resources hosted on third-party servers have not been altered.

Figure 9.1.2: Downloading jQuery library from the code.jquery.com CDN.

```
<script
  src="https://code.jquery.com/jquery-3.5.1.min.js"
  integrity="sha256-9/aliU8dGd2tb6OSsuzixeV4y/faTqgFtohetphbbj0="
  crossorigin="anonymous"></script>
```

The `$()` function is the same as the `jQuery()` function, which developers often use to type less code.

Figure 9.1.4: Using the $() function.

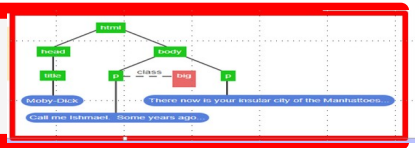
```
let $p = $(document.getElementById("hello"));

// same as
let $p = jQuery(document.getElementById("hello"));
```

Feedback?

Table 9.1.1: Common tasks performed by jQuery.

DOM manipulation	Find, alter, add, or remove DOM elements
User interaction	Respond to mouse clicks, mouse movement, or typing
Animation	Smoothly show, hide, or move web page elements
Widgets	Display and manage the interaction of complex GUI elements
Ajax	Issue asynchronous HTTP requests and handle responses
Browser quirks	Handle inconsistencies across different browsers



9. jQuery

jQuery provides various methods for different tasks e.g. manipulate DOM, events, ajax etc. The following table lists different categories of methods.

Category	Description	Imp Methods
DOM Manipulation	These methods manipulate DOM elements in some manner e.g. changing attribute, style attribute, adding and removing elements etc.	after(), append(), attr(), before(), more..
Traversing	These methods help in navigating from DOM element to another element in a parent child hierarchy e.g. finding ancestors, descendants or sibling element of a specified element.	children(), closest(), each(), first(), next(), filter(), parent(), siblings(), more..
CSS	These methods get and set css related properties of elements.	addClass(), css(), hasClass(), removeClass(), toggleClass() more..
Attributes	These methods get and set DOM attributes of elements.	attr(), html(), removeAttr(), prop(), val(), more..
Events	These methods are used to handle DOM or JavaScript events.	bind(), blur(), change(), click(), dblclick(), focus(), keyup(), keydown(), more..

The jQuery() function

The jQuery library defines a primary function called ***jQuery()***. The function behaves differently depending on what arguments are passed to `jQuery()`, but the function always returns a `jQuery` object. Good practice is to use variables that start with "$" to hold `jQuery` objects.

PARTICIPATION ACTIVITY | 9.1.3: Creating jQuery objects from DOM nodes.

```
<!DOCTYPE HTML>
<html lang="en">
   <title>jQuery Example</title>
   <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
   <body>
      <p id="hello">Hello, jQuery!</p>
      <p>This stuff is great!</p>
      <script>
         let helloPar = document.getElementById("hello");
         let $helloPar = jQuery(helloPar);
         let allParas = document.getElementsByTagName("p");
         let $allParas = jQuery(allParas);
      </script>
   </body>
</html>
```

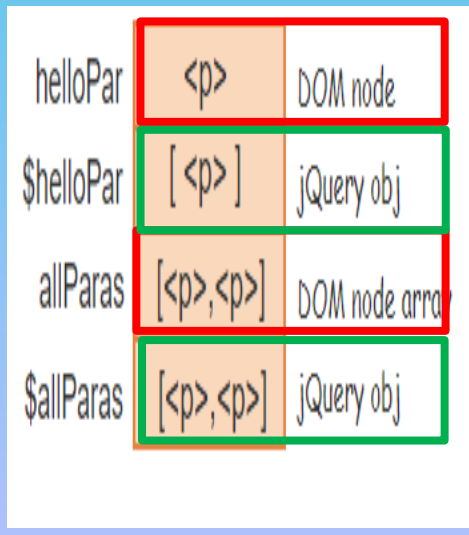
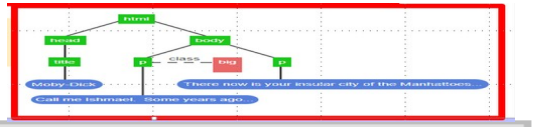


Table 9.2.1: Basic jQuery selectors.



Selector Type	Example	Explanation
Element	`$("p")`	Selects all `<p>` elements
ID	`$("#hello")`	Selects the element with `id="hello"`
Class	`$(".important")`	Selects all elements with `class="important"`

```
<!DOCTYPE html>
<html lang="en">
<title>jQuery Example</title>
<script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
<style>
   .important { color: red; }
</style>

<p id="hello" class="important">Hello, jQuery!</p>
<p class="important">This stuff is great!</p>

<script>
   let $allParas = $("p");
   $allParas.addClass("important");
</script>
</html>
```

Hello, jQuery!
This stuff is great!

$allParas [<p>,<p>]

JavaScript Essentials with jQuery

Course Overview

JavaScript Essentials with jQuery provides an introduction to and experience working with the JavaScript programming language in the environment it's used in the most: the browser.

Course Objectives

Working within in a dynamic, hands-on learning environment, guided by our expert team, attendees will:Become both familiar with the language and confident enough to work with it in any context Learn enough of the DOM API to bend it to your will Make communication between the browser and your server possible Understand and effectively leverage closures Understand how JavaScript's object model differs from the model classical object-oriented programming languages Learn what jQuery is and how to add it to your applications Use jQuery to select complex sets of elements from the DOM Develop rich web pages that respond to user interaction Interact with your server-side code using Ajax Explore a wide variety of plugins and learn how to write your own Test your applications to make sure your JavaScript is as solid as the rest of your code

Course Prerequisites

- HTML5 - Content Authoring Fundamentals
- HTML5 - Content Authoring with New and Advanced Features

Course Outline

Back to Course Search

Class Dates & Times

- **OLL Class times are listed Eastern time**
- **ILT Class times are local to that city**

This is a 5 day course

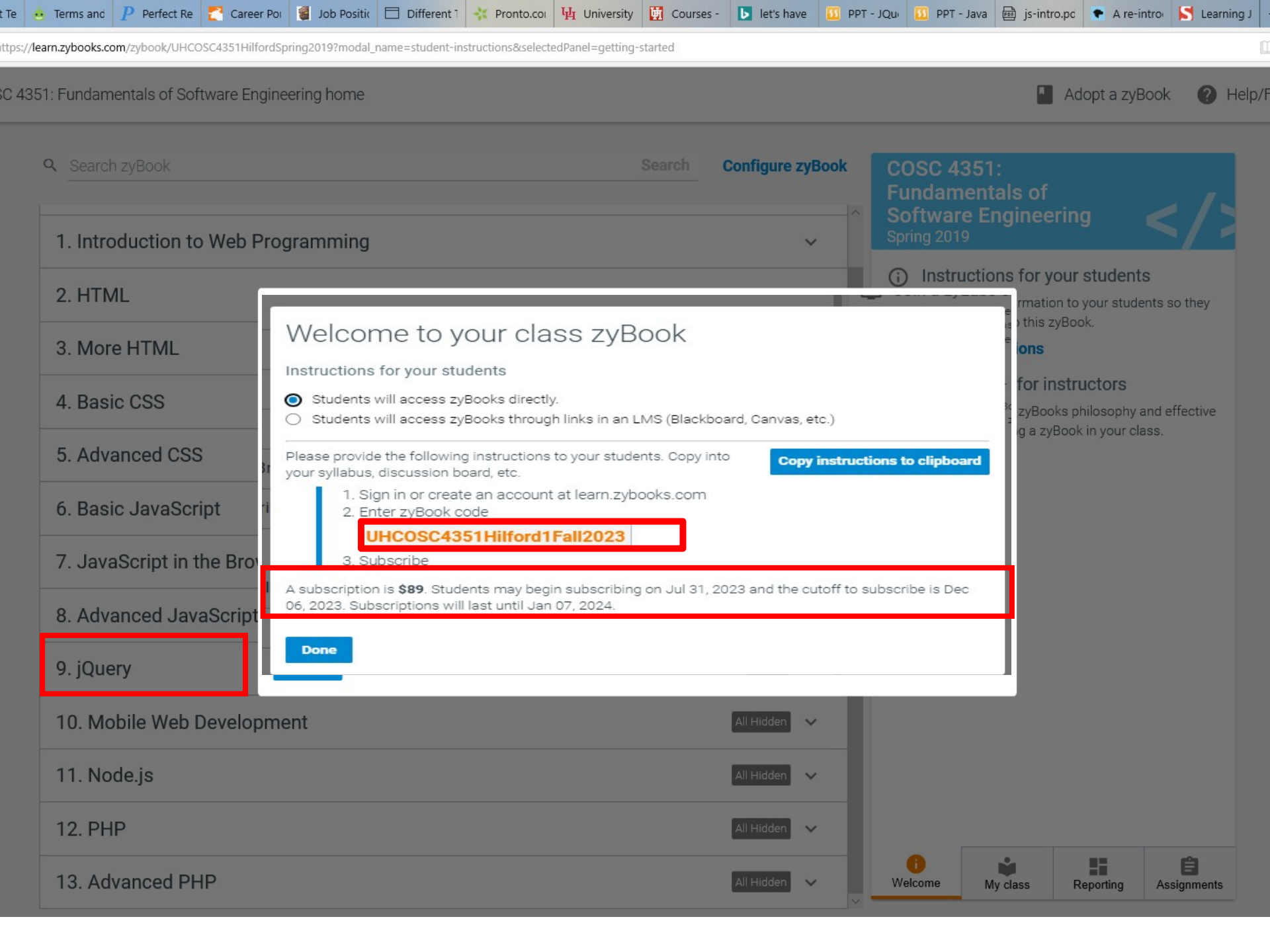
Full Price: ~~$ 3495.00~~

Trinity Health Price: $ 2040.00

Register for Classroom or Online LIVE

To sort by location or date, click the 'When' and 'Where' column headings.

Register	When	Time	Where	How
Register	02/11/2019	9:00AM - 5:00PM	Online LIVE	OLL
Register	04/22/2019	9:00AM - 5:00PM	Online LIVE	OLL
Register	06/03/2019	9:00AM - 5:00PM	Online LIVE	OLL


Terms and
Perfect Re
Career Poi
Job Positic
Different T
Pronto.co
University
Courses -
let's have
PPT - JQu
PPT - Java
js-intro.pc
A re-intro
Learning J
https://learn.zybooks.com/zybook/UHCOSC4351HilfordSpring2019?modal_name=student-instructions&selectedPanel=getting-started
SC 4351: Fundamentals of Software Engineering home
Adopt a zyBook
Help/F
Search zyBook
Search
Configure zyBook
1. Introduction to Web Programming
2. HTML
3. More HTML
4. Basic CSS
5. Advanced CSS
6. Basic JavaScript
7. JavaScript in the Bro
8. Advanced JavaScript
9. jQuery
10. Mobile Web Development
11. Node.js
12. PHP
13. Advanced PHP
All Hidden
COSC 4351:
Fundamentals of
Software Engineering
Spring 2019
Instructions for your students
rmation to your students so they
this zyBook.
ons
for instructors
zyBooks philosophy and effective
g a zyBook in your class.
Welcome
My class
Reporting
Assignments
Welcome to your class zyBook
Instructions for your students
Students will access zyBooks directly.
Students will access zyBooks through links in an LMS (Blackboard, Canvas, etc.)
Please provide the following instructions to your students. Copy into your syllabus, discussion board, etc.
Copy instructions to clipboard
1. Sign in or create an account at learn.zybooks.com
2. Enter zyBook code
UHCOSC4351Hilford1Fall2023
3. Subscribe
A subscription is $89. Students may begin subscribing on Jul 31, 2023 and the cutoff to subscribe is Dec 06, 2023. Subscriptions will last until Jan 07, 2024.
Done

From 4:45 to 4:55 PM – 10 minutes.

- [ ] 7. JavaScript in the Browser
  - [ ] 7.1 Using JavaScript with HTML
  - [ ] 7.2 Document Object Model (DOM)
  - [ ] 7.3 More DOM modification
  - [ ] 7.4 Event-driven programming
  - [ ] 7.5 Timers
  - [ ] 7.6 Modifying CSS with JavaScript
  - [ ] 7.7 Form validation
  - [ ] 7.8 JavaScript Object Notation (JSON)
  - [ ] 7.9 XMLHttpRequest (Ajax)
  - [ ] 7.10 Using third-party web APIs (JavaScript)
  - [ ] 7.11 Browser differences: JavaScript
  - [ ] 7.12 Example: Lights Out game
  - [ ] 7.13 Example: Weather Comparison (XMLHttpRequest)
  - [ ] 7.14 LAB: Grade distribution
  - [ ] 7.15 LAB: Grade distribution
  - [ ] 7.16 LAB: Temperature converter
  - [ ] 7.17 LAB: JavaScript Tic-Tac-Toe

Temperature Converter

Celsius:
12

Fahrenheit:
53.6

Convert

le is marked as read only Current file: index.html

```
<body>
    <h1>Temperature Converter</h1>
    <p>
        <label for="cInput">Celsius:</label>
        <input id="cInput" type="text">
    </p>
    <p>
        <label for="fInput">Fahrenheit:</label>
        <input id="fInput" type="text">
    </p>
    <input id="convertButton" type="button" value="Convert">
    <div id="errorMessage">
    </div>
    <p>
        <img id="weatherImage" src="warm.png" alt="Warm">
    </p>
</body>
```

1. The ***document.getElementById()*** method returns the DOM node whose `id` attribute is the same as the method's parameter. Ex: `document.getElementById("early_languages")` returns the p node in the HTML below.

- A ***DOMContentLoaded*** event is caused when the HTML file has been loaded and parsed, although other related resources such as CSS, JavaScript, and image files may not yet be

```
window.addEventListener("DOMContentLoaded", domLoaded);
```

3. Using the JavaScript ***addEventListener()*** method to register an event handler for a DOM object. Ex: `document.querySelector("#myButton").addEventListener("click", clickHandler)` registers a click event handler for the

```
var btn = document.getElementById("convertButton");
btn.addEventListener("click", convertButtonClicked);
```

```
window.addEventListener("DOMContentLoaded", domLoaded);

function domLoaded() {
    // TODO: Complete the function
}

function convertCtoF(degreesCelsius) {
    // TODO: Complete the function
}

function convertFtoC(degreesFahrenheit) {
    // TODO: Complete the function
}
```

1. The ***document.getElementById()*** method returns the DOM node whose `id` attribute is the same as the method's parameter. Ex: `document.getElementById("early_languages")` returns the p node in the HTML below.

7.7 Form validation

7.8 JavaScript Object Notation (JSON)

7.9 XMLHttpRequest (Ajax)

7.10 Using third-party web APIs (JavaScript)

7.11 Browser differences: JavaScript

7.12 Example: Lights Out game

7.13 Example: Weather Comparison

7.14 LAB: Grade distribution Lab

- A ***DOMContentLoaded*** event is caused when the HTML file has been loaded and parsed, although other related resources such as CSS, JavaScript, and image files may not yet be loaded.

3. Using the JavaScript ***addEventListener()*** method to register an event handler for a DOM object. Ex: `document.querySelector("#myButton").addEventListener("click", clickHandler)` registers a click event handler for the element with the id "myButton". Good practice is to use the `addEventListener()` method whenever possible, rather than using

```
1   window.addEventListener("DOMContentLoaded", domLoaded);
2
3   function domLoaded() {
4       // TODO: Complete the function
5   }
6
7   function convertCtoF(degreesCelsius) {
8       // TODO: Complete the function
9   }
10
11  function convertFtoC(degreesFahrenheit) {
12      // TODO: Complete the function
13  }
14
```

```
window.addEventListener("DOMContentLoaded", domLoaded);

function domLoaded() {
   // Input doms
   cBox = document.getElementById('cInput');
   fBox = document.getElementById('fInput');
   // Input Event Listeners
   cBox.addEventListener('input', function(){fBox.value = ""})
   fBox.addEventListener('input', function(){cBox.value = ""})
   // Convert button event listener
   document.getElementById('convertButton').addEventListener('click', function(){

      if(cBox.value !== ""){
         fBox.value = convertCtoF(parseFloat(cBox.value))
      }else{
         cBox.value = convertFtoC(parseFloat(fBox.value))
      }

      // Implment Weather image switch
   })
}

function convertCtoF(degreesCelsius) {
   return ((degreesCelsius*(9/5))+32)
}

function convertFtoC(degreesFahrenheit) {
   return ((degreesFahrenheit-32)*(5/9))
}
```

From 4:55 to 5:00 PM – 5 minutes.

Using jQuery, add class "standout" to the element with an id of "fifth". SHOW EXPECTED

HTML | CSS | JavaScript

```
1  <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
2
3  <p id="first">First paragraph</p>
4  <p id="second">Second paragraph</p>
5  <p id="third">Third paragraph</p>
6  <p id="fourth">Fourth paragraph</p>
7  <p id="fifth">Fifth paragraph</p>
8  <p id="sixth">Sixth paragraph</p>
```

Table 9.2.1: Basic jQuery selectors.

Selector Type	Example	Explanation
Element	`$("p")`	Selects all `<p>` elements
ID	`$("#hello")`	Selects the element with `id="hello"`
Class	`$(".important")`	Selects all elements with `class="importan`

CSS	These methods get and set css related properties of elements.	4	addClass(), css(), hasClass(), removeClass(), toggleClass() more..

```
$("#fifth").addClass("standout");
```

Using jQuery, remove class "subtle" from the second <p> tag SHOW EXPECTED

HTML | CSS | JavaScript

```
1 <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
2
3 <p class="subtle">First paragraph</p>
4 <p class="subtle">Second paragraph</p>
5 <p class="subtle">Third paragraph</p>
6 <p class="subtle">Fourth paragraph</p>
7 <p class="subtle">Fifth paragraph</p>
```

Table 9.2.1: Basic jQuery selectors.

Selector Type	Example	Explanation
Element	$("p")	Selects all <p> elements
ID	$("#hello")	Selects the element with id="hello"
Class	$(".important")	Selects all elements with class="important"

CSS	These methods get and set css related properties of elements.	4	addClass(), css(), hasClass(), removeClass(), toggleClass() more..

```
$("p:eq(1)").removeClass("subtle")
```

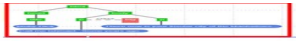
CHALLENGE ACTIVITY | 9.2.1: jQuery selectors.

Using jQuery, add class "highlight" to the element with an id of "second". SHOW EXPECTED

HTML | CSS | JavaScript

```
1 <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
2
3 <p id="first">First paragraph</p>
4 <p id="second">Second paragraph</p>
5 <p id="third">Third paragraph</p>
6 <p id="fourth">Fourth paragraph</p>
7 <p id="fifth">Fifth paragraph</p>
8 <p id="sixth">Sixth paragraph</p>
```

Table 9.2.1: Basic jQuery selectors.



Selector Type	Example	Explanation
Element	$("p")	Selects all <p> elements
ID	$("#hello")	Selects the element with id="hello"
Class	$(".important")	Selects all elements with class="important"

CSS

These methods get and set css related properties of elements.

4

addClass(),
css(),
hasClass(),
removeClass(),
toggleClass()
more..

```
$("#second").addClass("highlight");
```

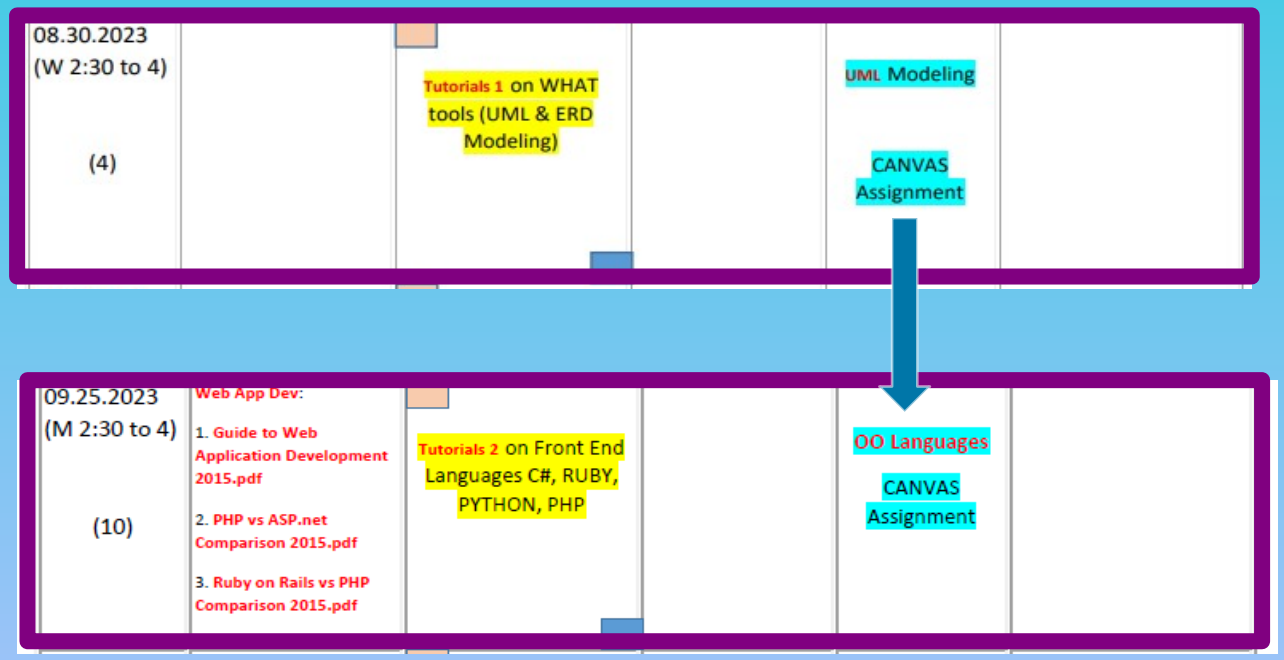
ZyBook – Web Programming

Have fun!
ZyBooks takes you step by step!

8 sections selected
Clear

- [ ] 1. Introduction to Web Programming
- [ ] 2. HTML
- [ ] 3. More HTML
- [ ] 4. Basic CSS
- [ ] 5. Advanced CSS
- [ ] 6. Basic JavaScript
- [ ] 7. JavaScript in the Browser
- [ ] 8. Advanced JavaScript
- [ ] 9. jQuery
- [ ] 10. Mobile Web Development
- [ ] 11. Node.js
- [ ] 12. PHP
- [ ] 13. Advanced PHP
- [ ] 14. Relational Databases and SQL

From 5:00 to 5:05 PM



09.25.2023 (M 2:30 to 4) (10)	Web App Dev: 1. Guide to Web Application Development 2015.pdf 2. PHP vs ASP.net Comparison 2015.pdf 3. Ruby on Rails vs PHP Comparison 2015.pdf	Tutorials 2 on Front End Languages C#, RUBY, PYTHON, PHP	OO Languages CANVAS Assignment

Complete your UML Model MVC Class Diagram to have the pseudocode for the methods you plan to implement
MUST attach it to this assignment.

OO Languages

Publish | Edit | ⋮

Complete your UML Model MVC Class Diagram to have the pseudocode for the methods you plan to implement (minimum restoreCatalog, saveCatalog, 1. addDVD, 4. display by Category).

MUST attach it to this assignment.

TA, please do not grade unless updated UML CLASS DIAGRAM is attached.

Please follow the instructions in the .doc for things that need to be turned in.

Please name the projects:

DVDApplication**CSharp**.zip
DVDApplication**Ruby**.zip
DVDApplication**Python**.zip
DVDApplication**Php**.zip

Attach these zips to this assignment.

Attachments

- UML to CSharp, Ruby, Python, Php5 Practice.docx
- Sample Screenshot.docx

For the Grader: if the student did not submit, please skip and do not assign a ZERO (MISSING).

VH– Publish.

Attachments

UML to CSharp, Ruby, Python, Php5 Practice.docx

Sample Screenshot.docx

Name:

800 points

UML MVC Class Diagram Model to C#, Ruby, Python, and Php5 *.

Enhance your **UML MVC Class Diagram** Model submitted for the UML Tutorial (add pseudocode for **main**, **showSelection**, UC**RestoreCatalog**, **restoreCatalog**, UC**AddDVD**, **addDVD**, UC**DisplayDVDByCategory**, **displayDVDByCategory**, UC**SaveCatalog**, **saveCatalog**) and convert it to the following Classes in **C#, Ruby, Python, and Php5***.

Please attach your new UML MVC Class Diagram to CANVAS Assignment.

You must use an IDE.

If not MVC Classes you will not get any points.

1. (200 points)

Given the above DVD Catalog Application above, convert the UML MVC Class Diagram Model to C# Classes and only implement the restore Catalog (using the given Catalog.txt), add DVD, display DVDs by category and save Catalog (using the given Catalog.txt).

(Must use an IDE, without it only half).

Answer:

Code here for each .cs file (one Class per file).

Full screen screenshot (output window, source window with first few lines of comments showing your name and Fall 2023)

Attachments

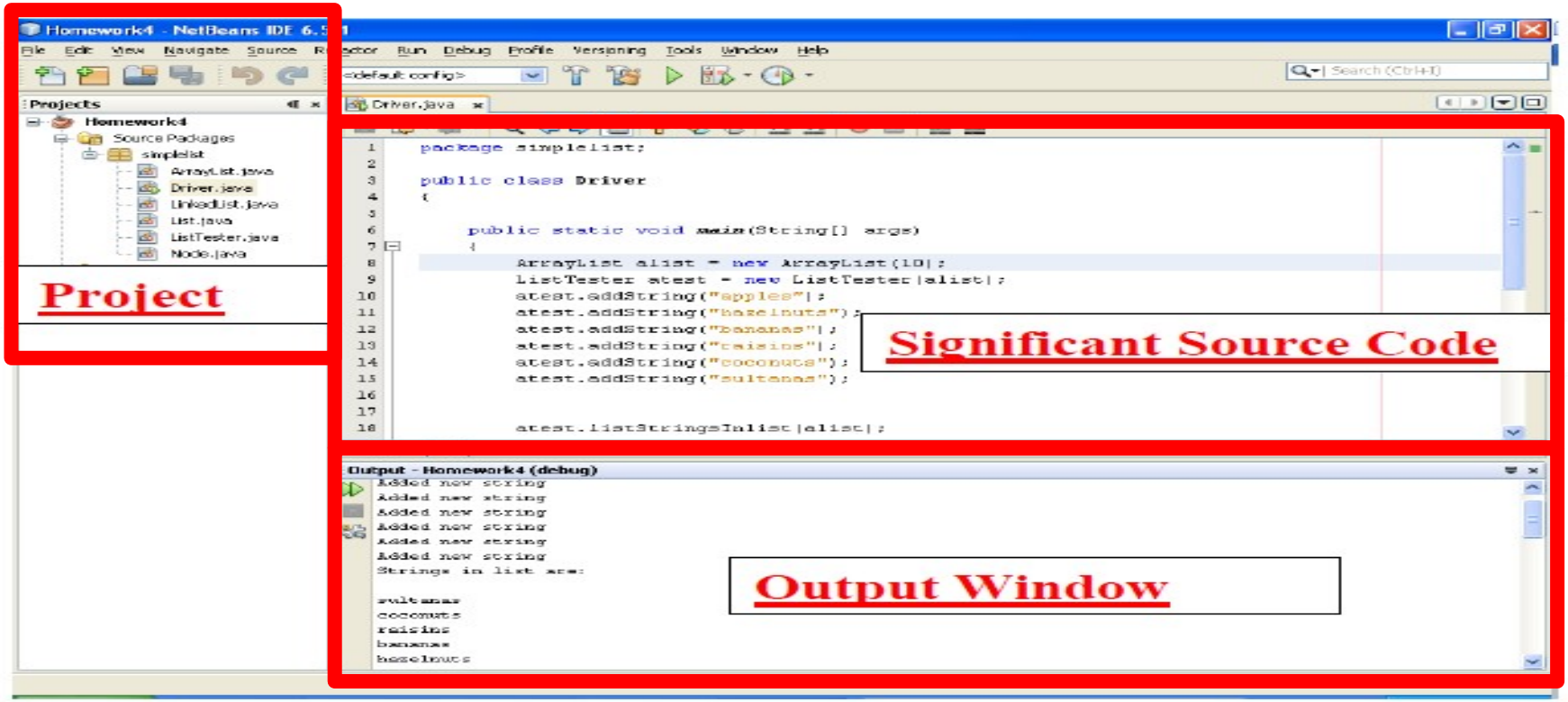
- UML to CSharp, Ruby, Python, Php5 Practice.docx

Sample Screenshot.docx

Step 1: Sample Screenshot (whole screen PRINT key on the keyboard will copy the screen with the following information:

Project, Output Window, Significant Source Code

Step 2: Open a word document and paste copy of the screen!



From 5:05 to 5:15 PM – 10 minutes.

09.20.2023 (W 4 to 5:30) (9)		WEB TECHNOLOGIES	Start ZyBook: Sections 6-9		

▾ CLASS PARTICIPATION 20 points — 20% of Total

END Class 9 Participation

Last 10 minutes of the class.
You can leave meeting when you are done.
PPT will be posted in CANVAS under CLASSES.

PASSWORD: SET 2

VH– Publish.

At 5:15 PM.

End Class 9

VH, Download Attendance Report
Rename it:
9.20.2023 Attendance Report FINAL

VH, upload Class 9 to CANVAS.