Chapter #1: What is Programming

A program is a set of instructions that tells your computer what to do.

Algorithms: The steps that a computer needs to take are expressed in the form of an algorithm. Each of these steps may be represented by a single machine instruction.

Programming Languages

1. New languages that would be easier for programmers to understand and write programs in were necessary. The first of such languages that appeared were called assembly languages.
2. There was still a need for more powerful languages that could express complex algorithms in a form that both the computer and the human programmer could understand. The result was the birth of "high-level" programming languages.
3. The first programming language to achieve widespread use was called FORTRAN, which stands for FORmula TRANslation. This language was developed by IBM to write programs that performed complex numerical calculations.

Language Translation

1. The traditional translator program for programming languages is called a compiler. a compiler reads a program written in a programming language (usually from a file), then outputs the program in a form that the computer understands. The original program is called the source program, and the compiler output is called the executable program
2. a compiler usually translates the source program into an intermediate form, sometimes called the object program, which is later processed by a different translator to produce the executable.
3. Programmers often develop programs using an Integrated Development Environment, or IDE. An IDE takes care of many of the translation steps automatically, so you may not even be aware of what is happening
4. A different translation process has become popular for some recent languages. This process uses an interpreter instead of a compiler. A language like this is sometimes called a scripting language.

Chapter #2: What is JavaScript?

Background

1. Web pages are sent to a client computer (e.g., your laptop) over the internet from a remote server. These pages contain data expressed in the HyperText Markup Language (HTML).
2. HTML is read and interpreted by a program on the client computer called a browser. You are probably familiar with browsers such as Firefox, Chrome, Safari, or Internet Explorer.

Interacting with web pages

1. These ideas were first explored by Netscape Communications, developers of the first commercial web browser, Netscape Navigator. Netscape worked with Sun Microsystems, developers of the Java programming language, to create a method for packaging small Java programs, termed applets. These mini programs were delivered from a web server to a client along with a web page.

2. An applet could provide things such as a small animation or an interactive calculator.
3. An applet could theoretically access data, files and other resources that it was not authorized to use. It was difficult to secure these applets to ensure that they could do only what was intended.
4. To help resolve this problem, and perhaps improve performance, Netscape began work on a purely interpreted or scripting language designed specifically to bring interaction to web pages.

A First Example

Other languages need a compiler to translate programs to native code, or a run-time interpreter to read the program and make it work. JavaScript requires only an interpreter, which is built into your browser.

Editors and IDEs

Many programs in other languages are primarily developed using Integrated Development Environments (IDEs) such as Visual Studio or Eclipse. These environments combine editing with compiling or interpreting, testing and debugging, as well as various analysis tools.

JavaScript Versions

JavaScript was submitted to the European Computer Manufacturers Association (ECMA) for consideration as a standard. This effort led to the publication of the standard ECMA262, known as ECMAScript, in 1997. Since then, JavaScript has been viewed as an implementation of ECMAScript.

Using the Language

1. The original purpose for JavaScript was to write programs that could be sent from a web server to a client computer, along with web pages to be run on the client's browser. This is called client-side programming.
2. Client-side JavaScript programs interact with HTML elements on a web page such as text, images, and buttons. JavaScript can locate, examine, and change HTML elements using the HTML Document Object Model, or the DOM
3. More recently, the potential of JavaScript for other programming applications has been recognized. One important new use is for server-side programming.

<u>Chapter #3: Rules of the Road</u>

Statements

1. First, a program is composed of statements.
2. The if-else statement takes certain actions depending on a condition.
3. A statement block (also called a code block) is a sequence of one or more statements enclosed in curly braces.

Literals

1. Each literal has a specific data type. JavaScript supports three primitive data types: numbers, strings, and booleans.
2. Numbers are written as a sequence of digits, with or without a decimal point.
3. Strings are sequences of text enclosed by either double quotes or single quotes.

4. Strings enclosed by double quotes may contain single quotes, and vice versa. A string with no characters is the empty string or null string.
5. Boolean literals consist of one of the words: true or false .
6. If a calculation or input that should produce a number results in an invalid number, the result is given the special value NaN (Not-a-Number). This is a number literal, not a string!
7. JavaScript actually supports five primitive datatypes, not three. The others are undefined and null.

Identifiers

1. These provide names for things like variables and functions
2. There is also a list of names called reserved words that cannot be used as identifiers because they have a special use in the language.
3. Most JavaScript programmers follow the convention that identifiers composed of multiple words should be written in camel case. This means that the words should be directly adjoined, and that all except the first should start with an upper-case letter.

Functions

1. JavaScript subprograms are called functions. A function is a block of code which is created to perform a particular task.
2. The arguments are a set of values to be assigned to the various parameters that have been defined for this particular function.
3. A JavaScript function is also called a method, and methods are associated with objects.

Comments

1. They provide some explanation of what is being done. Any line that starts with // (two slashes) is ignored by the JavaScript interpreter and is considered to be a comment.
2. A line that begins with /* (A slash followed by an asterisk) introduces a comment that does not end until the closing sequence */ is encountered.

Input and Output

The primary purpose of JavaScript is to interact with web pages, so its effect is most often seen indirectly by updating a web display.

<u>Chapter #4: Computing Results</u>

Variables: A variable may be thought of as a container that holds a value.

Assignment Statements:

1. The most common way to store a value in a variable is to use an assignment statement.
2. When a variable is declared, if it is not immediately assigned a value, JavaScript gives it the special value undefined.

Operators and Expressions

1. Expressions in JavaScript represent values.
2. Operators accept one or more operands to produce a new value.

3. The operators we will consider are either binary operators (which take two operands), or unary operators (which take only one).

Arithmetic Operators

1. +: Add
2. -: Subtract
3. *: Multiply
4. /: Divide
5. The modulus or remainder operator gives the remainder from a division operation:
6. The exponentiation operator raises the first number to the power of the second number:
7. ++: Increment
8. –: Decrement
9. total = count++: postfix operator. It comes after the operand and changes the operand after using its value.
10. total = ++count: prefix operator. It comes before the operand and changes the operand before using its value.

Comparison and Boolean Operators

1. A large group of binary operators compare two values, giving a boolean (or logical) value as a result.
2. These are called comparison operators, and they are also often known as relational operators.
3. ==: equal
4. !=: not equal
5. >: greater than
6. <: less than
7. >=: greater than or equal
8. <=: less than or equal
9. &&: and
10. ||: or
11. !: not

String Operator: The addition operator (+) may be applied to strings. In this case, it performs concatenation.

Numbers in Strings: Changing the data type of a value in this way is called casting.

Assignment Operators:

1. Assignment, represented by the symbol (=), may also be considered an operator.
2. +=: add and assign
3. -=: subtract and assign
4. *=: multiply and assign
5. /=: divide and assign

Grouping and Precedence

1. The JavaScript operators that we have seen use at most two operands at a time, but expressions may include many operators and operands.
2. In these cases, rules are needed to determine the order in which the operators are applied. This is the problem of operator precedence.
3. ++, -- (postfix) ++, -- (prefix),! ** *, /, % +, <, <=, >, >= ==, != &&, | =, +=, -=, *=, /=

Chapter #5: Making Choices

Conditions: A conditional statement takes actions depending on conditions. A condition is an expression with a boolean value. Conditions must evaluate to either true or false .

If Statement

1. Allows you to execute a statement or a group of statements only if a given condition is true.
2. if (condition) {statement block}

If-Else Statement: if (condition) {statement block 1} else {statement block 2}

Switch Statement

1. The switch statement consists of the keyword switch followed by an expression and a sequence of case clauses enclosed in curly braces.
2. switch (expression) { case clause ... case clause }
3. The switch statement is especially useful if there are a large number of cases.
4. The switch statement compares the value of the expression with the value given by each case clause. If the values match, the following statements up to the next case clause are executed.
5. The break statement indicates that the flow of control should exit the block in which it occurs.

More About Conditions

1. JavaScript provides another comparison operator we haven't previously mentioned, composed of three equals signs (===). This is the strictly equal operator.
2. We can write this more compactly using the conditional assignment operator (?). This is a ternary operator; it uses three different values to compute a result:

Repetition

Another very important use of conditions is to control the repetition of statements. A conditional statement may be used to repeat a statement block as long as a particular condition is true .

Chapter #6: Repeating Yourself

While Statement

1. The while statement, or while loop, specifies that a statement block is to be executed repeatedly as long as a given condition is true .
2. while (condition) { sequence of statements }

3. Due to the while loop, the actions are repeated until a value of 0 is seen. Then the program exits and displays the total. Each repetition is called an iteration.

Do While Statement

1. With this statement, the statement block will always run at least once.
2. do { sequence of statements } while (condition);

For Statement

1. The for statement can simplify three important steps that are generally needed when repeating a loop a fixed number of times: 1. initializing the count variable 2. testing the termination condition 3. updating the count variable
2. for (statement 1 ; statement 2 ; statement 3) {sequence of statements}
3. A count variable is generally needed to count the iterations. This variable is then compared to a limit value in the termination condition.

Break and Continue

1. The break statement may be used within the statement block of any of the loop statements discussed here.
2. The continue statement is only used with loops. It is best used with for loops, but technically you could use it with a while or a do while loop.
3. Labels may only be placed on loop statements (while , do while , or for) or on a statement block.
4. Many early languages allowed labels on any statement and provided a goto statement to transfer to any label from almost anywhere in the program.

<u>Chapter #7: Input, Output, and the DOM</u>

Document Object Model

1. To achieve more flexible input and output of text in the context of a web page, we can make use of the HTML Document Object Model (DOM). This model represents any web page in such a way that individual elements on the page can be referred to, examined and modified, by program statements in a language such as JavaScript.
2. The JavaScript file contains a function definition for the function findSquare() , which is called by the HTML when the button is clicked.
3. The input element is a void element; it has attributes but does not have content or an end tag.

Making it Pretty

One thing we may do is add a little color, which we could do purely with the use of CSS, which stands for Cascading Style Sheets.

Manipulating HTML Elements

The DOM gives us a method to add, remove, or modify any of the elements that may appear on a web page.

Chapter #8: Functions

1. Functions are subprograms that can be called from a JavaScript program to perform an action, and usually return a result.
2. A function provides a list of arguments enclosed in parentheses. Each argument is matched to a parameter that is specified in the function definition.

Function Definitions: Many functions return a value. This is done by a return statement such as: return result;

Function Invocation: A function that is defined within an HTML document may be placed anywhere within that document using the <script> tag.

Anonymous Functions: These functions have no actual name, so they are referred to as anonymous functions.

Arrow Functions: JavaScript offers an alternate syntax for some functions that is shorter, but more cryptic. This syntax, called arrow functions, applies to simple anonymous functions.

Variable Scope

1. If a JavaScript program is written top to bottom with no functions, any variable declared in the program may be used anywhere within that program. We say these variables have global scope.
2. Variables declared outside of any function still have global scope, but it is also possible to declare variables inside the function. These variables cannot be used outside the function; in fact, they only exist while the function is being executed. They are said to have local scope.

Let and Const

The big difference is that variables declared with either of these keywords may have block scope.

Hoisting: JavaScript provides a feature called hoisting. Because of this, all variable declarations within a scope are moved to the beginning

Chapter #9: Objects and Methods

Objects:

1. our examples have been based on the traditional programming style, called procedural programming. The principal alternative to procedural programming is object-oriented programming. With this approach, data is viewed as consisting of objects. Objects have properties (associated values) and methods (associated functions).
2. Objects now join the very short list of JavaScript data types. Primitive types, which have been introduced earlier, include number, string, boolean, undefined and null
3. One important JavaScript object that we have already seen is the document object. This object represents the HTML document that is currently being processed, and provides the methods for accessing the DOM. These methods are called using dot notation;

4. Another very important object in web programming with JavaScript is the window object. This object represents the browser window in which the current page is displayed. This is the default object, also called the global object, for all JavaScript programs running in a web browser.

Properties

1. Properties of an object may be thought of as variables, and they are described by name:value pairs.
2. This code is an object literal. It defines an object representing a library, with four properties: its name, the count of fiction and nonfiction books, and a boolean indicating whether the library is open.

Iteration using for ... in: This can be done by a statement similar to a for loop called the for ... in statement.

Methods:

In JavaScript, a function associated with an object is called a method. In JavaScript, all functions are methods. Methods that are called without naming an object are assumed to be methods of the global object.

Some Predefined Objects

1. 2 important ones: Date and Math . The Date object contains a date. Math is a global object that exists by default in your JavaScript environment and does not need to be created.
2. Math.sqrt(num) returns the square root of the number num. Math.abs(num) returns the absolute value of num . Math.sin(num) returns the sine of num . The argument should be given in radians. Math.exp(num) return the value of E (Euler's constant) to the power num . Math.log(num) returns the natural logarithm of num . Math.min() and Math.max() return the minimum and maximum, respectively, from a list of values.
3. Math.E gives Euler's constant. Math.PI gives the value of Pi. Math.SQRT2 gives the square root of 2. Math.SQRT1_2 gives the square root of ½. Math.LN2 gives the natural logarithm of 2. Math.LN10 gives the natural logarithm of 10. Math.LOG2E gives the base 2 logarithm of E. Math.LOG10E gives the base 10 logarithm of E.

<u>Chapter #10: Arrays, Maps and Sets</u>

What are Arrays?: An array is a variable that may contain a list of values, not just a single value.

Reading and Writing Array Elements:

Managing Tables: Arrays are especially useful for storing tables of values.

Adding and Removing Elements

1. push() is a function, and like most functions, it returns a value. The value is the length of the array after the new element was added. To remove the last element, there is a corresponding pop() method. This method deletes the last element.

2. This need can be met with the methods shift() (to remove an element) and unshift() (to add elements).

Multidimensional Arrays

Each element in this array is referred to by the index of the subarray followed by the index within the subarray.

Array Iteration

1. JavaScript offers two simpler ways to do this for arrays, as well as other special objects that consist of collections of values. Both ways are variations on the for loop. First is the for ... of loop, which iterates over all the elements of a collection
2. The second option is the forEach method. This statement calls a specified function once for every element in the array

Break and Continue

1. If the desired condition is met during processing, a break statement can be used to exit the for statement, but this would not exit both loops.
2. The continue statement causes the program to go on to the next row as soon as the sum gets too large

Maps and Sets

1. Maps may be viewed essentially as collections of name value (or key-value) pairs similar to object literals.
2. Sets are essentially arrays that are guaranteed to contain only unique values.

Chapter #11: Events, Timers, and Errors

Web pages are expected to respond in many ways when something happens. That something is called an event.

HTML Events: Events can be handled in a web page only if they are first triggered or detected by the HTML code.

Handling Events

Event Listeners: To avoid even this slight aberration, we can define event listeners in the JavaScript code. Using listeners, we eliminate the on ... attributes in the HTML tags that contain JavaScript

Timers

1. JavaScript provides two functions to generate a time delay. These functions are methods of the global window object. They will call a specified function when the time is up.
2. Stopping requires two additional functions, clearTimeout(retval) and clearInterval(retval) . These can be called with the value returned by setTimeout() or setInterval() to stop those two timers, respectively.

Errors

1. When JavaScript code encounters a problem it cannot handle, it signals an error.

2. JavaScript actually has a robust error handling mechanism similar to that found in other modern languages, where these errors are generally called exceptions. This includes a try ... catch ... finally statement plus the ability to throw exceptions.
3. It turns out that when JavaScript detects an error, it creates an error object with all the information it can provide to help you understand the problem.
4. The error object has two properties: message and name . The message property provides details about the error. The name property classifies the error into a few distinct categories. The name for the error we have just seen is ReferenceError .

Chapter #12: More about Data and Expressions

Number Literals

1. JavaScript represents all numbers as floating points. Specifically, the standard "IEEE format" is used with 64 bits.
2. Numbers written in this way are usually meant to be thought of as sequences of bits or bit strings.

Bitwise Operators

1. & bitwise and: The bitwise and sets each output bit to 1 only if both input bits are 1, otherwise 0.
2. | bitwise or: The bitwise or sets each output bit to 1 if either input bit is 1.
3. ^ bitwise or: The bitwise exclusive or sets each output bit to 1 only if the input bits are different
4. ~ bitwise not

Type Conversion: On many occasions, a function or an operator expects arguments of a certain type, while the values available are of a different type.

The typeof Operator: typeof is a unary operator (not a function) that takes a value of any type as an argument and returns a string identifying the type of that value. For example: typeof "Hello" -- returns "string"

Methods for Searching Strings: If there is more than one instance of the substring, search() finds only the first. The replace() method is designed for this. if we write:

Regular Expressions:

1. A regular expression is something different: a type of expression that can represent many different values at once.
2. We need more than just the wild card character; we really need a complete notation for patterns. This notation is called a regular expression.
3. There are also a variety of special codes called metacharacters that have special meaning, such as: ^ represents the beginning of the string $ represents the end of the string \s represents any whitespace, such as spaces and tabs \d matches any digit (this is the same as [0-9]) \w matches and letter, digit, or underscore (same as [a zA-Z0-9_]) \b matches the beginning or end of a word (i.e. a word boundary) And there are some codes called quantifiers that indicate how many times a preceding regular expression should occur

Chapter #13: Working with Objects

Properties that are functions are called methods of the object.

Constructors and Prototypes

1. The answer is that Object() is a special kind of function called a constructor. The Object() function creates a new, empty object.
2. Constructor functions can be useful, but they do not support the classical model of object construction and inheritance. In that model, an object is an instance of a class, which defines the object's properties (or methods). Further, the class is a subclass of a parent, called the superclass.
3. Every superclass has its own superclass, forming a chain until the top-level class is reached, which is generally called the object class.
4. It turns out that JavaScript objects have a built-in property called a prototype. The prototype of an object points to another object (possibly null) from which it inherits properties.

More about "new": You know that one way to create an object is to use the keyword new.

JavaScript Classes: A class in JavaScript is a type of function. In fact, the class requires one function, a constructor, which must have the name constructor

Accessors: Most object-oriented languages offer special methods called getters and setters to provide access to object properties. Getters return the value of a property, while setters initialize or modify that value.

Chapter #14: The DOM and the BOM

Browser Object Model (BOM) that represents more specifically the content appearing in the browser window.

Document Object Model

1. A web page, as we know, is defined by an HTML document.
2. Most elements have their own start tag and end tag, and all HTML elements are nested inside other elements, forming a tree. This tree is called the DOM tree.
3. All HTML elements in a document are nodes of the DOM tree. The purpose of the DOM is to enable programs running in the browser (typically in JavaScript, possibly in other languages) with the ability to identify and manipulate specific elements or groups of elements within a document
4. The DOM is defined by standards originally managed by the World Wide Web Consortium (W3C), but currently maintained by the Web Hypertext Application Technology Working Group (WHATWG).

JavaScript and the DOM

JavaScript programs work with the DOM using two main objects: 1. The document object, representing the entire document 2. The element object, representing individual elements

The Document Object

1. Its principal use is to provide a reference to locate specific elements on a page

2. item(index) : returns the element at the specified index position, if any.
3. namedItem(id) : returns the item whose id attribute matches the given id .
4. createElement(tag) creates a new element with the specified tag type.
5. createAttribute(attname) can be used to add an attribute to an element we have created, or to any other existing element.
6. addEventListener(event, function) defines a function to be executed when a specified event occurs.

Element Object: The element object represents a single element.

Browser Object Model: The Browser Object Model (BOM) is a representation of some specific properties of the window displayed by a browser.

Chapter #15: Related Technologies

HTML and CSS

XML: The eXtensible Markup Language (XML) has become extremely widely used in recent years. XML is syntactically very similar to HTML

XHTML: XHTML documents still represent web pages, but they can be parsed (and error checked) more reliably by system-independent tools.

JSON: JSON is not a markup language; it is actually a JavaScript data structure.

AJAX: Asynchronous JavaScript And XML (AJAX) obtains its name from a method for processing XML data asynchronously using JavaScript. AJAX also helps enable JavaScript to update a portion of the page, such as a table, without the need to recreate the entire page.

jQuery: jQuery (not an acronym) is a package of JavaScript code that can simplify a lot of common tasks.

Chapter #16: Beyond the Browser

JavaScript was developed for one central reason: to create interactive web pages that operate within a web browser.

Web Server Architecture

1. The most widely used web servers include Apache, Microsoft IIS, and nginx.
2. A web browser acts independently, usually controlled by a user sitting at the computer or working with a mobile device. The user initiates a request for a web page, causing the browser to send a message to a specific IP address. The message is sent using a standard web protocol, generally HTTP or the more secure HTTPS. This request also specifies a port number, and generally has various other parameters as well. This is a lot like calling a function, but on a different computer.

Breaking Loose of the Browser: Each browser contains a JavaScript Engine that provides the necessary support for JavaScript running within that browser.

Node.js

1. Using Node.js, JavaScript can be used to write server applications, or even some types of applications that are completely unrelated to the web.
2. Node.js gives a developer access to the full JavaScript language – current versions support ES6 or higher – but some things are a little different.
3. This includes a standard collection of library modules, including modules that provide: File operations Event handling Server creation (HTTP or HTTPS) Timer methods

Chapter #3 Hw

1. Write a statement to display your full name.

```
const firstName = "Your";
const lastName = "Name";
console.log(`Full Name: ${firstName} ${lastName}`);
```

2. Which of the following literals are written correctly? What is wrong with the incorrect ones?
   a. 35: *
   b. 5,284: comma
   c. 'Happy birthday' : *
   d. "The boy said "Hello"": *
   e. FALSE: *

3. Write the number 256,000,000,000 as a literal in exponential form: $2.56 * 10^{10}$

4. Which of the following are valid identifiers? What is wrong with the invalid ones?
   a. piece of cake: spaces not allowed
   b. $100: *
   c. 2ndOne: cannot start with a number
   d. else: reserved keyword
   e. thisIsAVeryLongIdentifier: *

5. Write the phrase "time of day" as a camel case identifier.: timeOfDay

6. What will the following program print? /* This program prints "Hello" and "Goodbye" document.write("Hello"); */ document.write("Goodbye")

```
Goodbye
```

7. Write a program that prompts for a line of input and then prints that line as its output.

```
const userInput = prompt("Enter a line of text:");
```

```
console.log("You entered: " + userInput);
```

Chapter #4: HW

Question 1: Suppose the regular price of a concert ticket is $7.50, but there may be a variable discount. Write a program that reads the discount as a percentage and displays the new price. If the discount is 10(%), the price should be $6.75.

```
// Function to calculate the new price after discount
function calculateDiscountedPrice(originalPrice, discountPercentage) {
    let discountAmount = (originalPrice * discountPercentage) / 100;
    let newPrice = originalPrice - discountAmount;
    return newPrice.toFixed(2); // Return the new price rounded to 2 decimal places
}

// Regular price of the concert ticket
let regularPrice = 7.50;

// Example discount percentage
let discountPercentage = 10; // You can change this value to test with different discounts

// Calculate the new price
let newPrice = calculateDiscountedPrice(regularPrice, discountPercentage);

// Display the new price
console.log("The new price after a discount of " + discountPercentage + "% is $" + newPrice);
```

Question2: Suppose in a restaurant numberOfTables is 50, seatsPerTable is 4, and dirtyTables is 10. Give the value of the following expressions: a. numberOfTables * seatsPerTable b. numberOfTables - dirtyTables * seatsPerTable c. (numberOfTables - dirtyTables) * seatsPerTable

```
// Variables
let numberOfTables = 50;
let seatsPerTable = 4;
let dirtyTables = 10;

// Calculations
let a = numberOfTables * seatsPerTable;
let b = numberOfTables - dirtyTables * seatsPerTable;
let c = (numberOfTables - dirtyTables) * seatsPerTable;
```

```
// Display the results
console.log("a. numberOfTables * seatsPerTable = " + a);
console.log("b. numberOfTables - dirtyTables * seatsPerTable = " + b);
console.log("c. (numberOfTables - dirtyTables) * seatsPerTable = " + c);
```

Question 3: In the following program, what will be the value of x? var a = 30; var b = 7; var x = a / b - a % b;
In the following program, what will be the value of x? var a = 30; var b = 7; var x = a / b - a % b;
2.2857

Question 4: Suppose ageSarah , ageRobert , and ageMiguel give the ages of three people. Write an expression that sets robertIsOldest to true only if ageRobert is the highest.

```
// Variables for ages
let ageSarah = 25; // Example age for Sarah
let ageRobert = 30; // Example age for Robert
let ageMiguel = 28; // Example age for Miguel

// Expression to check if Robert is the oldest
let robertIsOldest = (ageRobert > ageSarah) && (ageRobert > ageMiguel);

// Display the result
console.log("Is Robert the oldest? " + robertIsOldest);
```

Question 5: What will be printed by the following program? What will be the value of total after these statements are executed? count = 10; total = --count * 3; document.write("Total count is " + total++);
28

Chapter #5

Question 1

Suppose color is "green" and day is 17. Which of the following conditions are true ? a. day <= 20 b. color == "red" c. color == "green" | day > 20 d. color == "red" && day < 20

Question 2: What, if anything, is wrong with the following statement? if day > 20 {color = "blue";}

```
Syntax and Semicolon: if (day > 20) {
    color = "blue";
}
```

Question 3: What is z after the following statements? var x=15; var y=20; var z = y; if (x > 5) { z += x; }

35

Question 4: Give the value of cheese after the following: var cheese = "swiss"; var color = "yellow"; if (color == "brown") { cheese = "gouda"; } else if (color != "yellow") { cheese = "parmesan"; } else { cheese = "cheddar"; }

Cheddar

Question 5: Give the value of stateName after the following: var stateCode = 2; var stateName; switch (stateCode) { case 1: stateName = "New York"; break; case 2: stateName = "New Jersey"; case 3: stateName = "Pennsylvania"; break; default: stateName = "invalid state"; }

Pennsylvania

Question 6: Suppose highTemp is 60 and lowTemp is 50. What is the value of tempDiff after the following statement? var tempDiff = ((highTemp - lowTemp) > 15) ? "big" : "small";

Small

Chapter #6

Question 1: How many times will the body of this while loop run? var num = 100; while (num > 1) { num /= 3; }

5 times

Question 2: Write a while loop that will read and then print a series of strings until two identical strings are typed.

```
const prompt = require('prompt-sync')(); // This line is for Node.js environment to read input
let previousString = '';
let currentString = '';
while (true) {
    currentString = prompt('Enter a string: ');
    console.log(currentString);
    if (currentString === previousString) {
        break;
    }
    previousString = currentString;
}
console.log('Two identical strings were typed. Loop terminated.');
```

Question 3: Rewrite your code from Question 2 as a do while loop.

```
const prompt = require('prompt-sync')(); // This line is for Node.js environment to read input
let previousString = '';
let currentString = '';
do {
    currentString = prompt('Enter a string: ');
    console.log(currentString);
    if (currentString === previousString) {
        break;
    }
    previousString = currentString;
} while (true);
console.log('Two identical strings were typed. Loop terminated.');
```

Question 4: What is wrong with the following for statement? for (var count = 1, count < 6, count += 2) { statement block }

```
for (var count = 1; count < 6; count += 2) {
    // statement block
}
```

Question 5: Write a for loop that writes the cumulative sum of the first 25 integers as follows: 1 3 6 10 ...

```
let cumulativeSum = 0;
for (let i = 1; i <= 25; i++) {
    cumulativeSum += i;
    console.log(cumulativeSum);
}
```

Question 6: Rewrite your answer to Question 5, using a break statement to exit this loop when the sum would exceed 100.

```
let cumulativeSum = 0;
for (let i = 1; i <= 25; i++) {
    cumulativeSum += i;
    if (cumulativeSum > 100) {
        break;
    }
    console.log(cumulativeSum);
}
```

```
console.log('Loop terminated because the cumulative sum exceeded 100.');
```

Chapter #7

Question 1: Given the HTML code at the beginning of the chapter, why will the following reference not work? value = document.getElementById("textin").innerHTML;

```
document.addEventListener("DOMContentLoaded", function() {
    value = document.getElementById("textin").innerHTML;
});
```

Question 2:

```
function swapPars() {
    // Get the paragraphs by their IDs
    var p1 = document.getElementById("p1");
    var p2 = document.getElementById("p2");
    // Swap the innerHTML of the paragraphs
    var temp = p1.innerHTML;
    p1.innerHTML = p2.innerHTML;
    p2.innerHTML = temp;}
```

Question 3: In the "Find Squares" example, why can't we simply define the function as findSquare() , with no parameters, and get the input value with the statement: var inputNum = inform.textin.value;

```
function findSquare(num) {
    return num * num;
}
var inputNum = document.getElementById("textin").value;
var square = findSquare(inputNum);
```

Question 4: The following statement sets the color of the identified element to red: document.getElementById("elem1").style.color = "red"; In addition, the following HTML button is given an initial color of green: Push Me Using this information, write code to display a single button that changes color each time it is pressed in the sequence "green", "yellow", "red" . Show the body of the HTML page, along with the JavaScript function.

```
document.getElementById("colorButton").addEventListener("click", function() {
    var button = document.getElementById("colorButton");
    var currentColor = button.style.color;
    switch (currentColor) {
        case "green":
            button.style.color = "yellow";
```

```
            break;
        case "yellow":
            button.style.color = "red";
            break;
        case "red":
            button.style.color = "green";
            break;
        default:
            button.style.color = "green";
    }
});
```

Appendix D describes the HTML tags used to define simple tables. Set up a page to display a table of your favorite book titles along with their authors. The table should have two columns and five rows. Each row will contain a book title in the first column and the author's name in the second column. Fill in the book titles in the first column but leave the second column blank. Write a JavaScript function that will be called when a button is pushed. This function should step through each row and prompt for the author's name. When the name is typed it should be entered in its proper place in the table. A second button should be provided to allow the user to clear the table and start over.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Favorite Books</title>
    <style>
        table {
            width: 50%;
            margin: 20px auto;
            border-collapse: collapse;
        }
        th, td {
            border: 1px solid black;
            padding: 8px;
            text-align: left;
        }
        th {
```

```
            background-color: #f2f2f2;
        }
        button {
            display: block;
            margin: 20px auto;
            padding: 10px 20px;
            font-size: 16px;
        }
    </style>
</head>
<body>
    <h1 style="text-align: center;">My Favorite Books</h1>
    <table id="booksTable">
        <tr>
            <th>Book Title</th>
            <th>Author</th>
        </tr>
        <tr>
            <td>The Great Gatsby</td>
            <td></td>
        </tr>
        <tr>
            <td>To Kill a Mockingbird</td>
            <td></td>
        </tr>
        <tr>
            <td>1984</td>
            <td></td>
        </tr>
        <tr>
            <td>Pride and Prejudice</td>
            <td></td>
        </tr>
        <tr>
            <td>The Catcher in the Rye</td>
            <td></td>
        </tr>
```

```
    </table>
    <button onclick="fillAuthors()">Fill Authors</button>
    <button onclick="clearTable()">Clear Table</button>
    <script>
        function fillAuthors() {
            const table = document.getElementById('booksTable');
            for (let i = 1; i < table.rows.length; i++) {
                const author = prompt(`Enter the author for "${table.rows[i].cells[0].innerText}":`);
                table.rows[i].cells[1].innerText = author;
            }
        }
        function clearTable() {
            const table = document.getElementById('booksTable');
            for (let i = 1; i < table.rows.length; i++) {
                table.rows[i].cells[1].innerText = '';
            }
        }
    </script>
</body>
</html>
```

Chapter #8

Question 1: Write a function that accepts three numbers as arguments and returns the minimum

```
function findMinimum(a, b, c) {
    return Math.min(a, b, c);
}

// Example usage:
let result = findMinimum(5, 3, 9);
console.log("The minimum value is:", result);
```

Question 2: What, if anything, is wrong with the following code: function getBlue() { var color = "blue"; return color; } var myColor = getBlue;

```
function getBlue() {
    var color = "blue";
    return color;
}
```

var myColor = getBlue(); // Call the function
console.log(myColor); // This will output: "blue"

Question 3: What, if anything, is wrong with the following code: function getBlue() { var color = "blue"; } var myColor = color;
var color;

```
function getBlue() {
    color = "blue";
}

getBlue();
var myColor = color;
```

Question 4: Write an arrow function that compares two strings and returns the shortest (you should be able to do this in one statement so you can use the short form!)
const getShortestString = (str1, str2) => str1.length <= str2.length ? str1 : str2;

Question 5: Consider the following code sequence: var redBalls = 20; let whiteBalls = 50; { let redBalls = 30; } allBalls = redBalls + whiteBalls; What is the value assigned to allBalls ?
allBalls = 20 + 50; // 70

Question 6: What, if anything, is wrong with each of the following: a. b. c. x = 5; var y = x; var x = 10; x = 5; var y = x; let x = 10; const x = 5; var y = x; x = 10;

- ***a:*** *No error, but hoisting might cause unexpected behavior.*
- ***b:*** *`ReferenceError` due to the temporal dead zone of `let`.*
- ***c:*** *`TypeError` because `const` variables cannot be reassigned.*

Chapter #9

Question 1: The following object is designed to help plan a vegetable garden: var garden = { length: 20, width: 10, organic: true, fertilizer: "manure", plants: { lettuce: 20, tomatoes: 6, carrots: 30 } } Write a statement to add a method that returns the total number of plants planned for this garden. use a for ... in statement to iterate over the properties of plants

```
var garden = {
  length: 20,
```

```
  width: 10,
  organic: true,
  fertilizer: "manure",
  plants: {
    lettuce: 20,
    tomatoes: 6,
    carrots: 30
  },
  totalPlants: function() {
    let total = 0;
    for (let plant in this.plants) {
      total += this.plants[plant];
    }
    return total;
  }
};

// Example usage:
console.log(garden.totalPlants()); // Output: 56
```

Question 2: Write code to replace the lettuce plants listed in the garden object with 10 spinach plants.

```
var garden = {
  length: 20,
  width: 10,
  organic: true,
  fertilizer: "manure",
  plants: {
    lettuce: 20,
    tomatoes: 6,
    carrots: 30
  },
  totalPlants: function() {
    let total = 0;
    for (let plant in this.plants) {
      total += this.plants[plant];
    }
    return total;
  }
```

```
};

// Replace lettuce with spinach
garden.plants.spinach = 10;
delete garden.plants.lettuce;

// Example usage:
console.log(garden.plants); // Output: { tomatoes: 6, carrots: 30, spinach: 10 }
console.log(garden.totalPlants()); // Output: 46
```

Question 3: What is output by the following code (assuming the given order is maintained)? var result = ""; for (var prop in garden) { result = result + prop + ": " + garden[prop] + "
"; } document.write(result); Note that the last line may be surprising.

Question 4: Show code to create a Date object for the date and time June 5, 2020, 11:15 AM.

```
var date = new Date(2020, 5, 5, 11, 15, 0);
console.log(date);
```

Question 5: Write an object literal for a circle object which includes (at least) two properties: its radius, and a method to compute its area using the rule that area equals PI times the radius squared.

```
var circle = {
  radius: 5,
  area: function() {
    return Math.PI * Math.pow(this.radius, 2);
  }
};

// Example usage:
console.log("Radius: " + circle.radius); // Output: Radius: 5
console.log("Area: " + circle.area());   // Output: Area: 78.53981633974483
```

Chapter #10

Question 1: Consider the following code: var numList = [22, 35, 17, 40, 83]; numList[10] = 99; what would be the value of each of the following: a. numList[3] b. numList[7] c. numList.length

numList[3]: This refers to the 4th element in the array (since arrays are zero-indexed). The value is 40.

numList[7]: This index was not explicitly assigned a value. In JavaScript, unassigned indices in an array are undefined. So, numList[7] is undefined.

numList.length: When you assign a value to numList[10], the array's length automatically adjusts to accommodate the highest index. Therefore, numList.length becomes 11

Question 2: Suppose an array colors is defined by var colors = ["orange", "yellow", "green", "blue", "indigo"] Write two statements to complete the spectrum by adding " red " at the beginning and " violet " at the end

```
colors.unshift("red");   // Adds "red" at the beginning
colors.push("violet");   // Adds "violet" at the end
```

Question 3: Use forEach to write a loop that iterates over the elements of the colors array in Question 2 and constructs a new array colorLengths containing the length of each color name string.

```
var colors = ["red", "orange", "yellow", "green", "blue", "indigo", "violet"];
var colorLengths = [];

colors.forEach(function(color) {
    colorLengths.push(color.length);
});

console.log(colorLengths); // Output: [3, 6, 6, 5, 4, 6, 6]
```

Question 4: onsider the "2-dimensional" array composers in the text. Write code using nested for loops to search this array for the value 1840, using a break statement with a label to exit the search if the value is found. Assume this value may be found in any row and column. At the end, print the row index and column index where the value lies.

```
var composers = [
    [1770, 1827, 1756],
    [1685, 1750, 1840],
    [1813, 1883, 1862]
];

var found = false;
var rowIndex = -1;
var colIndex = -1;

search:
for (var i = 0; i < composers.length; i++) {
    for (var j = 0; j < composers[i].length; j++) {
```

```
        if (composers[i][j] === 1840) {
            rowIndex = i;
            colIndex = j;
            found = true;
            break search; // Exit both loops
        }
    }
}

if (found) {
    console.log("Value 1840 found at row index: " + rowIndex + ", column index: " + colIndex);
} else {
    console.log("Value 1840 not found in the array.");
}
```

Question 5: Write a function that will accept a Date object as an argument and return the day of the week for that date as a string (e.g. "Sunday"). Use an array to store all of the possible strings.

```
function getDayOfWeek(date) {
    var daysOfWeek = ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"];
    return daysOfWeek[date.getDay()];
}

// Example usage:
var date = new Date("2024-09-12");
console.log(getDayOfWeek(date)); // Output: "Thursday"
```

Chapter #11

Question 1: The HTML framework for our demo pages, with colors provided via CSS, was introduced in Chapter 7. Add the onmouseover attribute to the button element in this framework to change the color of the button to red when the mouse or pointer is over it. It is not necessary to change back again when the mouse moves away. Use any JavaScript example to show that your framework works

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Button Color Change</title>
```

```
    <style>
        .my-button {
            background-color: blue;
            color: white;
            padding: 10px 20px;
            border: none;
            cursor: pointer;
        }
    </style>
</head>
<body>
    <button class="my-button" onmouseover="changeColor(this)">Hover over me!</button>

    <script>
        function changeColor(element) {
            element.style.backgroundColor = 'red';
        }
    </script>
</body>
</html>
```

Question 2: Repeat Question 1 using an event listener. The name of the event is mouseover

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Button Color Change with Event Listener</title>
    <style>
        .my-button {
            background-color: blue;
            color: white;
            padding: 10px 20px;
            border: none;
            cursor: pointer;
        }
    </style>
```

```
</head>
<body>
    <button class="my-button" id="colorButton">Hover over me!</button>

    <script>
        document.getElementById('colorButton').addEventListener('mouseover', function() {
            this.style.backgroundColor = 'red';
        });
    </script>
</body>
</html>
```

Question 3: What is printed by the following code sequence? var running = true; function func1(running = false); function func2(document.write("Running
"); var ret1 = setTimeout (func1, 5000); var ret2 = setInterval (func2, 2000); while (running) {} clearInterval(ret2); document.write("Done
");

Infinite Loop

Question 4: Rewrite the factTable function from Chapter 10, using try ... catch and throw to create an exception, then catch that exception if the limit is out of range.

```
function factTable(limit) {
    try {
        if (limit < 0 || limit > 10) {
            throw new RangeError("Limit out of range");
        }
        for (let i = 1; i <= limit; i++) {
            console.log(` ${i}! = ${factorial(i)}`);
        }
    } catch (error) {
        console.error(error.message);
    }
}

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

```
}
```

Project 2

```
const vegetables = [
    ["Tomato", 10, 2.5],
    ["Carrot", 5, 1.2],
    ["Lettuce", 8, 1.0]
];

function populateTable() {
    const table = document.getElementById("vegetableTable");
    vegetables.forEach(veg => {
        let row = table.insertRow();
        veg.forEach(cell => {
            let cellElement = row.insertCell();
            cellElement.textContent = cell;
        });
    });
}

function calculateTotal() {
    try {
        const budget = parseFloat(document.getElementById("budget").value);
        if (isNaN(budget)) throw "Invalid budget input";

        let totalCost = 0;
        vegetables.forEach(veg => {
            totalCost += veg[1] * veg[2];
        });

        const result = document.getElementById("result");
        if (totalCost > budget) {
            result.textContent = `Total cost ($${totalCost.toFixed(2)}) exceeds your budget!`;
            result.style.color = "red";
        } else {
            result.textContent = `Total cost is $${totalCost.toFixed(2)}.`;
            result.style.color = "green";
```

```
        }
    } catch (error) {
        alert(`Error: ${error}`);
    }
}

window.onload = populateTable;
```

Final Project

Login Page

```
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
</head>
<body>
    <h2>Login Page</h2>
    <form id="loginForm">
        <label for="username">Username:</label>
        <input type="text" id="username" name="username"><br><br>
        <label for="password">Password:</label>
        <input type="password" id="password" name="password"><br><br>
        <input type="button" value="Login" onclick="validateLogin()">
    </form>
    <script>
        function validateLogin() {
            var username = document.getElementById('username').value;
            var password = document.getElementById('password').value;
            if (username === 'user' && password === 'pass') {
                window.location.href = 'main.html';
            } else {
                alert('Invalid username or password');
            }
        }
    </script>
</body>
</html>
```

Main Page

```
<!DOCTYPE html>
<html>
<head>
    <title>Main Page</title>
    <style>
        table {
            width: 100%;
            border-collapse: collapse;
        }
        table, th, td {
            border: 1px solid black;
        }
        th, td {
            padding: 15px;
            text-align: left;
        }
    </style>
</head>
<body>
    <h2>Main Page</h2>
    <h3>To-Do List</h3>
    <table id="todoTable">
        <tr>
            <th>Task</th>
            <th>Duration</th>
            <th>Due Date</th>
            <th>Action</th>
        </tr>
    </table>
    <input type="text" id="task" placeholder="Task">
    <input type="text" id="duration" placeholder="Duration">
    <input type="date" id="dueDate">
    <button onclick="addTask()">Add Task</button>

    <h3>Countdown Clock</h3>
    <input type="datetime-local" id="targetTime">
```

```
<button onclick="startCountdown()">Start</button>
<button onclick="stopCountdown()">Stop</button>
<div id="countdown"></div>

<script>
    var tasks = [];
    var countdownInterval;

    function addTask() {
        var task = document.getElementById('task').value;
        var duration = document.getElementById('duration').value;
        var dueDate = document.getElementById('dueDate').value;
        tasks.push({task: task, duration: duration, dueDate: dueDate});
        renderTasks();
    }

    function renderTasks() {
        var table = document.getElementById('todoTable');
        table.innerHTML = '<tr><th>Task</th><th>Duration</th><th>Due Date</th><th>Action</th></tr>';
        tasks.forEach((task, index) => {
            var row = table.insertRow();
            row.insertCell(0).innerHTML = task.task;
            row.insertCell(1).innerHTML = task.duration;
            row.insertCell(2).innerHTML = task.dueDate;
            row.insertCell(3).innerHTML = '<button onclick="deleteTask(' + index + ')">Delete</button>';
        });
    }

    function deleteTask(index) {
        tasks.splice(index, 1);
        renderTasks();
    }

    function startCountdown() {
        var targetTime = new Date(document.getElementById('targetTime').value).getTime();
        countdownInterval = setInterval(function() {
            var now = new Date().getTime();
```

```
            var distance = targetTime - now;
            var days = Math.floor(distance / (1000 * 60 * 60 * 24));
            var hours = Math.floor((distance % (1000 * 60 * 60 * 24)) / (1000 * 60 * 60));
            var minutes = Math.floor((distance % (1000 * 60 * 60)) / (1000 * 60));
            var seconds = Math.floor((distance % (1000 * 60)) / 1000);
            document.getElementById('countdown').innerHTML = days + "d " + hours + "h " + minutes + "m " +
seconds + "s ";
            if (distance < 0) {
                clearInterval(countdownInterval);
                document.getElementById('countdown').innerHTML = "EXPIRED";
            }
        }, 1000);
    }

    function stopCountdown() {
        clearInterval(countdownInterval);
    }
  </script>
</body>
</html>
```