Untitled-1

Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 1: Variables & Data

Just like we follow certain rules in English grammar, we also have rules for writing a JavaScript program. The set of these rules is called the syntax in JavaScript.

What is a Variable?

A variable is a container that stores a value. This is very similar to the containers used to store rice, water, and oats (treat this as an analogy!).

The value of a variable in JavaScript can be changed during the execution of a program.

```
var a = 7; // Using var
let b = 7; // Using let
```

- **Identifier:** a, b
- **Assignment operator:** =
- **Literal:** 7

Declaring Variables

Rules for choosing variable names:

- Letters, digits, underscores, and the $ sign are allowed.
- Must begin with a letter or a $.
- JavaScript is case-sensitive: 'Harry' and 'harry' are different variables.
- Cannot use reserved words as variable names.

Var vs Let in JavaScript

1. `var` is functionally scoped while `let` and `const` are block-scoped.
2. `var` can be updated and re-declared within its scope.
3. `let` can be updated but not re-declared.
4. `const` can neither be updated nor re-declared.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Page: [Insert Page Number Here]

5. `var` variables are initialized with `undefined`, whereas `let` and `const` variables are not initialized.

6. `const` must be initialized during declaration, unlike `let` and `var`.

Primitive Data Types & Objects

Primitive data types are a set of basic data types in JavaScript. An object is a non-primitive data type in JavaScript.

These are the 7 primitive data types in JavaScript:

- `null`
- `number`
- `string`
- `symbol`
- `undefined`
- `boolean`
- `BigInt`

Object

An object in JavaScript can be created as follows:

```
const item = {
  name: "LED Bulb", // Key: value
  price: 150, // Key: value
};
```

Quick Quiz: Write a JavaScript program to store the name, phone number, and marks of a student using objects.

Note: Ensure to replace "[Insert Page Number Here]" with the appropriate page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 1: Practice Set

1. Create a variable of type string and try to add a number to it.

2. Use the `typeof` operator to find the datatype of the result in the last question.

3. Create a `const` object in JavaScript. Can you change it to hold a number later?

4. Try to add a new key to the `const` object in Problem 3. Were you able to do it?

5. Write a JavaScript program to create a word-meaning dictionary of 5 words.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 2: Expressions & Conditionals

A fragment of code that produces a value is called an expression. Every value written literally is an expression. For example: `77`, `"Harry"`.

Operators in JavaScript

1. **Arithmetic Operators**
    - `+` Addition
    - `-` Subtraction
    - `*` Multiplication
    - `**` Exponentiation
    - `/` Division
    - `%` Modulus
    - `++` Increment
    - `--` Decrement
2. **Assignment Operators**
    - `=` Assign
    - `+=` Add and assign
    - `-=` Subtract and assign
    - `*=` Multiply and assign
    - `/=` Divide and assign
    - `%=` Modulus and assign
    - `**=` Exponentiate and assign

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

3. **Comparison Operators**
    - `==` Equal to
    - `!=` Not equal
    - `===` Equal value and type
    - `!==` Not equal value or not equal type
    - `>` Greater than
    - `<` Less than
    - `>=` Greater than or equal to
    - `<=` Less than or equal to

- `?` Ternary operator

4. **Logical Operators**

   - `&&` Logical AND
   - `||` Logical OR
   - `!` Logical NOT

Apart from these, we also have type and bitwise operators. Bitwise operators perform bit-by-bit operations on numerical operands.

Example:

```
7 + 8 = 15 // Result
```

Comments in JavaScript

Sometimes we want our programs to contain text that is not executed by the JS Engine. Such text is called a comment in JavaScript.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

A comment in JavaScript can be written as follows:

```
let a = 2; // This is a single-line comment

/* This is a
   multi-line comment */
```

Sometimes comments are used to prevent the execution of some lines of code:

```
let switch = true;
// switch = false;  // Commented line won't execute
```

Conditional Statements

Sometimes we might have to execute a block of code based on some condition. For example, if the age of a user is greater than 18, we might display a special message.

In JavaScript, we have three forms of conditional statements:

1. `if` statement
2. `if...else` statement
3. `if...else if...else` statement

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting

for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

If Statement

The `if` statement in JavaScript looks like this:

```
if (condition) {
  // Execute this code
}
```

If the condition inside the parentheses evaluates to true, the code inside the block is executed; otherwise, it is not.

If-Else Statement

The `if` statement can have an optional `else` clause. The syntax looks something like this:

```
if (condition) {
  // Block of code if condition is true
} else {
  // Block of code if condition is false
}
```

If the condition is true, the code inside the `if` block is executed; if the condition is false, the code inside the `else` block is executed.

If-Else If Statement

Sometimes we might want to keep rechecking a set of conditions one by one until one matches. We use `if...else if` for achieving this.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Syntax of `if...else if`

The syntax of `if...else if` looks like this:

```
if (age >= 70) {
  console.log("A valid age");
} else if (age > 10 && age < 15) {
  console.log("But you are a kid");
```

```
} else if (age >= 18) {
  console.log("Not a kid");
} else {
  console.log("Invalid age");
}
```

JavaScript Ternary Operator

Evaluates a condition and executes a block of code based on the condition.

Syntax: `condition ? expression1 : expression2`

Example syntax of the ternary operator looks like this:

```
marks > 10 ? "Yes" : "No";
```

If marks are greater than 10, the result is 'Yes'; otherwise, it is 'No'.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 2: Practice Set

1. Use logical operators to determine whether a person's age lies between 10 and 20.
2. Demonstrate the use of switch case statements in JavaScript.
3. Write a JavaScript program to find whether a number is divisible by 2 and 3.
4. Write a JavaScript program to determine whether a person can drive or not based on their age using the ternary operator. Print "Youth" if the age is greater than 18; otherwise, print "Cannot Drive."

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 3: Loops & Functions

We use loops to perform repeated actions. For example, to print numbers from 1 to 100, it would be very hectic to do it manually. Loops help us automate such tasks.

Types of Loops in JavaScript

- **For Loop:** Executes a block of code a number of times.
- **For...in Loop:** Loops through the keys of an object.
- **For...of Loop:** Loops through the values of an iterable object.
- **While Loop:** Executes a block of code based on a specific condition.
- **Do-While Loop:** A variant of the while loop which runs at least once.

The For Loop

The syntax of a for loop looks something like this:

```
for (statement1; statement2; statement3) {
  // Code to be executed
}
```

- **Statement 1** is executed one time.
- **Statement 2** is the condition based on which the loop runs (loop body is executed).
- **Statement 3** is executed every time the loop body is executed.

Quick Quiz: Write a sample for loop of your choice.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

The For-In Loop

The syntax of a for-in loop looks like this:

```
for (key in object) {
  // Code to be executed
}
```

Quick Quiz: Write a sample program demonstrating a for-in loop.

Note: For-in loops also work with arrays, which will be discussed in later videos.

The For-Of Loop

The syntax of a for-of loop looks like this:

```
for (variable of iterable) {
  // Code to be executed
}
```

Iterable data structures include arrays, strings, etc.

Quick Quiz: Write a sample program demonstrating a for-of loop.

The While Loop

The syntax of a while loop looks like this:

```
while (condition) {
  // Code to be executed
}
```

Note: If the condition never becomes false, the loop will never end and this might crash the runtime.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Quick Quiz: Write a sample program demonstrating a while loop.

The Do-While Loop

The syntax of a do-while loop looks like this:

```
do {
  // Code to be executed
} while (condition);
```

The code is executed at least once.

Quick Quiz: Write a sample program demonstrating a do-while loop.

Functions in JavaScript

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

The syntax of a function looks something like this:

```
function myFunc() {
  // Code
}

function binodFunc(parameter1, parameter2) {
  // Code
}
```

A function with parameters: Here, the parameters behave as local variables.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Function Invocation

Function invocation is a way to use the code inside the function. A function can also return a value. The value is "returned" back to the caller.

Example:

```
binodFunc(7, 8); // Function invocation

const sum = (a, b) => {
  let c = a + b;
  return c; // Returns the sum
};

let y = sum(1, 3);
console.log(y); // Prints 4
```

Another way to create and use a function.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 3: Practice Set

1. Write a JavaScript program to print the marks of students in an object using a for loop.

```
const obj = { harry: 98, rohan: 70, aakash: 73 };
for (let key in obj) {
  console.log(key + ": " + obj[key]);
}
```

2. Write the program in question 1 using a for-in loop.

```
const obj = { harry: 98, rohan: 70, aakash: 73 };
for (let key in obj) {
  console.log(key + ": " + obj[key]);
}
```

3. Write a JavaScript program to print "Try again" until the user enters the correct number.

```
let correctNumber = 7;
let guess;
```

```
do {
  guess = prompt("Guess the number:");
  if (guess != correctNumber) {
    console.log("Try again");
  }
} while (guess != correctNumber);
console.log("Correct!");
```

4. Write a function to find the mean of 5 numbers.

```
function findMean(a, b, c, d, e) {
  return (a + b + c + d + e) / 5;
}
console.log(findMean(1, 2, 3, 4, 5)); // Example usage
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 4: Strings

Strings are used to store and manipulate text. A string can be created using the following syntax:

```
let name = "Harry"; // Creates a string
console.log(name.length); // This property prints the length of the string
```

Strings can also be created using single quotes:

```
let name = "Harry";
```

Template Literals

Template literals use backticks instead of quotes to define a string:

```
let name = `Harry`;
```

With template literals, it is possible to use both single and double quotes inside a string:

```
let sentence = `The name "is" Harry's.`; // Backtick, double quote, single quote
```

We can insert variables directly in template literals. This is called string interpolation:

```
let a = `This is ${name}`; // Prints "This is Harry"
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Escape Sequence Characters

When you try to print the following string in JavaScript, you will encounter an issue:

```
let name = 'Adam D'Angelo';  // Incorrect due to the single quote in the name
```

We can use the single quote escape sequence to solve the problem:

```
let name = "Adam D'Angelo"; // Correct usage of escape sequence
```

Similarly, we can use `"` inside a string with double quotes. Other escape sequence characters are as follows:

- `\n` → Newline
- `\t` → Tab
- `\r` → Carriage Return

String Properties and Methods

1. To get the length of a string:

```
let name = "Harry";
console.log(name.length); // Prints 5
```

2. To convert a string to uppercase:

```
let name = "Harry";
console.log(name.toUpperCase()); // Prints "HARRY"
```

3. To convert a string to lowercase:

```
let name = "Harry";
console.log(name.toLowerCase()); // Prints "harry"
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

4. Using `slice()` to extract a substring:

```
let name = "Harry";
console.log(name.slice(2, 4)); // Prints "rr" (from index 2 to 4, 4 not included
```

5. Using `slice()` to extract a substring from a specified position to the end:

```
let name = "Harry";
console.log(name.slice(2)); // Prints "rry" (from index 2 to end)
```

6. Using `replace()` to replace a substring:

```
let name = "Harry Bhai";
let newName = name.replace("Bhai", "Bhau");
```

7. Using `concat()` to concatenate strings:

```
let name1 = "Harry";
let name2 = "Naman";
let name3 = name1.concat(name2, "Yes");
// We can even use the + operator
```

8. Using `trim()` to remove whitespaces:

```
let name = " Harry ";
let newName = name.trim(); // Removes whitespaces
```

Accessing Characters in Strings

Strings are immutable. To access a character, we use the following syntax:

```
let name = "Harry";
console.log(name[0]); // Prints "H"
console.log(name[1]); // Prints "a"
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 4: Practice Set

1. What will the following print in JavaScript? `console.log("har".length)`
2. Explore the `includes()`, `startsWith()`, and `endsWith()` functions of a string.
3. Write a program to convert a given string to uppercase.
4. Extract the amount out of this string: "Please give Rs 1000".
5. Try to change the 4th character of a given string. Were you able to do it?

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 5: Arrays

Arrays are variables that can hold more than one value.

```
const fruits = ["banana", "apple", "grapes"];
const mixedArray = [7, "Harry", false]; // Can be of different types
```

Accessing Values

```
let numbers = [1, 2, 7, 9];
console.log(numbers[0]); // 1
console.log(numbers[1]); // 2
```

Finding the Length

```
let numbers = [1, 2, 7, 9];
console.log(numbers.length); // 4
```

Changing the Values

```
let numbers = [7, 2, 40, 9];
numbers[2] = 8; // 'numbers' now becomes [7, 2, 8, 9]
```

Arrays are mutable, meaning they can be changed.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

In JavaScript, arrays are objects. Using the `typeof` operator on arrays returns "object":

```
const n = [1, 7, 9];
console.log(typeof n); // Returns "object"
```

Arrays can hold many values under a single name.

Array Methods

There are some important JavaScript array methods. Some of them are as follows:

1. **`toString()`** - Converts an array to a comma-separated string:

```
let n = [1, 7, 9];
console.log(n.toString()); // "1,7,9"
```

2. **`join()`** - Joins all the array elements using a separator:

```
let n = [7, 9, 13];
console.log(n.join("-")); // "7-9-13"
```

3. **`pop()`** - Removes the last element from the array and returns the popped value:

```
let n = [1, 2, 4];
console.log(n.pop()); // Updates the original array, returns the popped value (4
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

4. **`push()`** - Adds a new element at the end of the array:

```
let a = [7, 1, 2, 8];
a.push(9); // Modifies the original array and returns the new array length
```

5. **`shift()`** - Removes the first element and returns it.

6. **`unshift()`** - Adds an element to the beginning. Returns the new array length.

7. **`delete`** - Array elements can be deleted using the `delete` operator:

```
let d = [7, 8, 9, 10];
delete d[1]; // `delete` is an operator
```

8. **`concat()`** - Used to join arrays to the given array:

```
let a1 = [1, 2, 3];
let a2 = [4, 5, 6];
let a3 = [9, 8, 7];
let result = a1.concat(a2, a3); // Returns [1, 2, 3, 4, 5, 6, 9, 8, 7]
// Returns a new array. Does not change existing arrays.
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

9. **`sort()`** - The `sort()` method is used to sort an array alphabetically:

```
let a = [7, 9, 8];
a.sort(); // Changes to [7, 8, 9], modifies the original array
```

`sort()` takes an optional compare function. If provided as the first argument, the `sort()` function will consider these values (the values returned from the compare function) as the basis of sorting.

10. **`splice()`** - `splice()` can be used to add new items to an array:

```
const numbers = [1, 2, 3, 4, 5];
numbers.splice(2, 1, 23, 24); // Returns deleted items, modifies the array
// Position: 2, No of elements to remove: 1, Elements to be added: 23, 24
```

`slice()` - Slices out a piece from an array. It creates a new array:

```
const num = [1, 2, 3, 4];
console.log(num.slice(2)); // [3, 4]
console.log(num.slice(1, 3)); // [2, 3]
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

12. `reverse()` - Reverses the elements in the source array.

Looping through Arrays

Arrays can be looped through using the classical JavaScript for loop or through some of the methods discussed below:

1. **`forEach` loop** - Calls a function once for each array element:

```
const a = [1, 2, 3];
a.forEach((value, index, array) => {
  // Function logic
});
```

2. **`map()`** - Creates a new array by performing some operation on each array element:

```
const a = [1, 2, 3];
const newArray = a.map((value, index, array) => {
  return value * 2; // Example operation
});
```

3. **`filter()`** - Filters an array with values that pass a test. Creates a new array:

```
const a = [1, 2, 3, 4, 5];
const filteredArray = a.filter((value) => value > 3); // Example test
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

4. **`reduce()` method** - Reduces an array to a single value:

```
const numbers = [1, 8, 7, 11];
let sum = numbers.reduce((accumulator, currentValue) => {
  return accumulator + currentValue;
});
// 1 + 8 + 7 + 11
```

5. **`Array.from()`** - Used to create an array from any other object:

```
Array.from("Harry"); // Creates an array from the string "Harry"
```

6. **`for...of`** - The `for...of` loop can be used to get the values from an array:

```
for (let value of array) {
  // Access each value
}
```

7. **`for...in`** - The `for...in` loop can be used to get the keys from an array:

```
for (let index in array) {
  // Access each index
}
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 5: Practice Set

1. Create an array of numbers and take input from the user to add numbers to this array.
2. Keep adding numbers to the array in question 1 until 0 is added to the array.
3. Filter for numbers divisible by 10 from a given array.
4. Create an array of squares of given numbers.
5. Use `reduce()` to calculate the factorial of a number from an array of natural numbers. (n being the number whose factorial needs to be calculated)

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 6: JavaScript in the Browser

JavaScript was initially created to make web pages come alive. It can be written right in a web page's HTML to make it interactive.

The browser has an embedded engine called the JavaScript engine or the JavaScript runtime.

Security is very important. For example, a webpage on `http://example.com` cannot access information from `http://somesite.com`.

Developer Tools

Every browser has some developer tools which make a developer's life a lot easier.

Pressing F12 on Chrome opens the Developer Tools.

Console

All the errors can be seen in the Console tab. We can also write JavaScript commands in the Console.

Network

All network requests can be monitored in the Network tab.

HTML Elements

The Elements tab allows you to inspect and modify HTML elements.

The `<script>` Tag

The `<script>` tag is used to insert JavaScript into an HTML page.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

The `<script>` tag can be used to insert external or internal scripts.

For internal scripts:

```
<script>
  alert("Hello");
</script>
```

For external scripts:

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

The benefit of using a separate JavaScript file is that the browser will store it in its cache.

Console Object Methods

The console object has several methods, with `log` being one of them. Some of them are as follows:

- `assert()` - Used to assert a condition.
- `clear()` - Clears the console.
- `log()` - Outputs a message to the console.
- `table()` - Displays tabular data.
- `warn()` - Used for warnings.
- `error()` - Used for errors.
- `info()` - Used for special information.

You will naturally remember some or all of these with time. A comprehensive list can be looked up on MDN.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Interaction: alert, prompt, and confirm

- `alert`: Used to invoke a mini window with a message.

  ```
  alert("Hello");
  ```

- `prompt`: Used to take user input as a string.

  ```
  let input = prompt("Hi", "No"); // "No" is an optional default value
  ```

- `confirm`: Shows a message and waits for the user to press OK or Cancel. Returns true for OK and false for Cancel.

The exact location and look of these dialogs are determined by the browser, which is a limitation.

Window Object, BOM & DOM

When JavaScript runs in a browser, we have the following:

- **Window Object**: Represents the browser window and provides methods to control it. It is a global object.
- **DOM (Document Object Model)**: Represents the structure of the webpage.
- **BOM (Browser Object Model)**: Provides objects to interact with the browser.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Document Object Model (DOM)

The DOM represents the page content as objects that can be manipulated by JavaScript.

```
document.body; // Accesses the page body as an object
document.body.style.background = "green"; // Changes the page background to gree
```

Browser Object Model (BOM)

The Browser Object Model (BOM) represents additional objects provided by the browser (host environment) for working with everything except the document. Functions like `alert()`, `confirm()`, and `prompt()` are also part of the BOM.

```
location.href = "https://codewithharry.com"; // Redirects to another URL
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 6: Practice Set

1. Write a program using the `prompt()` function to take age as input from the user and use `alert()` to tell them if they can drive.
2. In the previous question, use `confirm()` to ask the user if they want to see the prompt again.
3. In the previous question, use `console.error()` to log an error if the age entered is negative.
4. Write a program to change the website to `google.com` (Redirection) if the user enters a number greater than 4.
5. Change the background color of the page to yellow, red, or any other color based on user input through `prompt()`.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 7: Walking the DOM

The DOM tree refers to the structure of the HTML page, where all the elements are represented as nodes. There are 3 main types of nodes in the DOM tree:

1. Element nodes
2. Text nodes
3. Comment nodes

In an HTML page, `<html>` is at the root, with `<head>` and `<body>` as its children, etc. A text node is always a leaf of the tree.

Auto Correction

If the browser encounters erroneous HTML, it tends to correct it automatically. For example, if a tag is placed after the body, it is automatically moved inside the body. Another example is the `<table>` tag, which must contain a `<tbody>`, but if omitted, the browser will insert it.

Walking the DOM

```
<html>
  <head>
    <title>Hello</title>
  </head>
  <body>
    <!-- Page body -->
  </body>
</html>
```

- `document.body` refers to the `<body>` tag.
- `document.documentElement` refers to the entire HTML page.
- `document.head` refers to the `<head>` tag.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Note: `document.body` can sometimes be `null` if the JavaScript is written before the body tag.

Children of an Element

Elements that are directly nested within an element are called its children.

- **Child Nodes:** Elements that are direct children. For example, `<head>` and `<body>` are children of `<html>`.
- **Descendant Nodes:** All nested elements, including children, their children, and so on.

`firstChild`, `lastChild` & `childNodes`

- `element.firstChild` - Refers to the first child element.
- `element.lastChild` - Refers to the last child element.
- `element.childNodes` - Refers to all child nodes.

The following is always true:

```
elem.childNodes[0] === elem.firstChild;
elem.childNodes[elem.childNodes.length - 1] === elem.lastChild;
```

There is also a method `elem.hasChildNodes()` to check whether there are any child nodes.

Note: `childNodes` looks like an array, but it's not actually an array; it's a collection. We can use `Array.from(collection)` to convert it into an array. Array methods won't work directly on a collection.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Notes on DOM Collections

- They are read-only. A variable (reference) to `elem.childNodes` will automatically update if the childNodes of `elem` are changed.
- They are iterable using the `for...of` loop.

Siblings and the Parent

Siblings are nodes that are children of the same parent. For example, `<head>` and `<body>` are siblings because they share the same parent. In the mentioned example, `<body>` is considered the "next" or "right" sibling of `<head>`, and `<head>` is considered the "previous" or "left" sibling of `<body>`.

- The next sibling can be accessed through the `nextSibling` property, and the previous one through the `previousSibling` property. The parent is available as `parentNode`.

```
alert(document.documentElement.parentNode); // document
alert(document.documentElement.parentElement); // null
```

Element-only Navigation

Sometimes, we don't want text or comment nodes. Some properties only take element nodes into account, for example, `previousElementSibling` refers to the previous sibling which is an element.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting

for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

```
document.nextElementSibling; // Next sibling (Element)
document.firstElementChild; // First Element child
document.lastElementChild; // Last Element child.
```

Table Links

Certain DOM elements may provide additional properties specific to their type for convenience. The table element supports the following properties:

- `table.rows`: Collection of `<tr>` elements.
- `table.caption`: Reference to a caption.
- `table.thead`: Reference to `<thead>`.
- `table.tfoot`: Reference to `<tfoot>`.
- `table.tBodies`: Collection of `<tbody>` elements.

Inside a `<tbody>`:

- `tbody.rows`: Collection of `<tr>` inside.

Inside a `<tr>`:

- `tr.cells`: Collection of `<td>` and `<th>`.
- `tr.sectionRowIndex`: Index of `<tr>` inside the enclosing element.
- `tr.rowIndex`: Row number starting from 0.

Inside a `<td>`:

- `td.cellIndex`: Number of cells inside the enclosing `<tr>`.

Quick Quiz: Print `typeof document` and `typeof window` in the Console and see what it prints.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Searching the DOM

DOM navigation properties are helpful when elements are close to each other. If they are not, we have some more methods to search the DOM:

- **`document.getElementById`**: This method is used to get the element with a given "id" attribute.

```
let span = document.getElementById("span");
span.style.color = "red";
```

- **`document.querySelectorAll`**: Returns all elements inside the document matching the given CSS selector.
- **`document.querySelector`**: Returns the first element for the given CSS selector. An efficient version of `document.querySelectorAll(css)[0]`.
- **`document.getElementsByTagName`**: Returns elements with the given tag name.
- **`document.getElementsByClassName`**: Returns elements that have the given CSS class.

  Don't forget the "s" in "Elements".
- **`document.getElementsByName`**: Searches elements by the "name" attribute.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Matches, Closest & Contains Methods

There are three important methods to search the DOM:

1. **`elem.matches(css)`**: Checks if the element matches the given CSS selector.
2. **`elem.closest(css)`**: Looks for the nearest ancestor that matches the given CSS selector. The element itself is also checked.
3. **`elemA.contains(elemB)`**: Returns true if `elemB` is inside `elemA` (a descendant of `elemA`), or when `elemA === elemB`.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 7: Practice Set

1. Create a navbar and change the color of its first element to red.
2. Create a table without `<tbody>`. Now use the "View Page Source" button to check whether it has `<tbody>` or not.

3. Create an element with 3 children. Now change the color of the first and last element to green.
4. Write JavaScript code to change the background color of `<h1>` tags to cyan.
5. Which of the following is used to look for the farthest ancestor that matches a given CSS selector?
    - (a) matches
    - (b) closest
    - (c) contains
    - (d) none of these

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 8: Events & Other DOM Properties

Console.dir Function

- `console.log` shows the element's DOM tree.
- `console.dir` shows the element as an object with its properties.

tagName/nodeName

- Used to read the tag name of an element.
- `tagName` only exists for element nodes.
- `nodeName` is defined for any node (text, comment, etc.).

innerHTML and outerHTML

- The `innerHTML` property allows you to get the HTML inside the element as a string (for element nodes only).
- The `outerHTML` property contains the full HTML (`innerHTML` + the element itself).

textContent

- Provides access to the text inside the element, only text minus all tags.

The hidden Property

- The "hidden" attribute and the DOM property specify whether the element is visible or not.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

```
<div hidden>I am hidden</div>

<div id="element">I Can be hidden</div>

<script>
  document.getElementById("element").hidden = true;
</script>
```

Attribute Methods

1. **`elem.hasAttribute(name)`** - Method to check for the existence of an attribute.
2. **`elem.getAttribute(name)`** - Method used to get the value of an attribute.
3. **`elem.setAttribute(name, value)`** - Method used to set the value of an attribute.
4. **`elem.removeAttribute(name)`** - Method to remove the attribute from the element.
5. **`elem.attributes`** - Method to get the collection of all attributes.

Data-* Attributes

We can always create custom attributes, but the ones starting with "data-" are reserved for programmers' use. They are available in a property named `dataset`.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

If an element has an attribute named "data-one", it's available as `element.dataset.one`.

Insertion Methods

We looked at some ways to insert elements into the DOM. Here is another way:

```
let div = document.createElement("div"); // Create
div.className = "alert"; // Set class
div.innerHTML = "<span>Hello</span>";
document.body.append(div);
```

Here are some more insertion methods:

1. `node.append(c)` - Appends at the end of the node.
2. `node.prepend(e)` - Inserts at the beginning of the node.
3. `node.before(e)` - Inserts before the node.
4. `node.after(e)` - Inserts after the node.
5. `node.replaceWith(e)` - Replaces the node with the given node.

Quick Quiz: Try out all these methods with your own webpage.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting

for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Node Removal

To remove a node, there's a method `node.remove()`:

```
let id1 = document.getElementById("id1");
id1.remove();
```

ClassName and ClassList

Assigning something to `elem.className` replaces the entire string of classes. Often, we want to add, remove, or toggle a single class.

1. **`elem.classList.add("class")`** / **`elem.classList.remove("class")`** - Adds/removes a class.
2. **`elem.classList.toggle("class")`** - Adds the class if it doesn't exist, otherwise removes it.
3. **`elem.classList.contains("class")`** - Checks for the given class, returns true/false.

setTimeout and setInterval

`setTimeout` allows us to run a function once after the interval of time.

The syntax of `setTimeout` is as follows:

```
let timerId = setTimeout(function, delay, arg1, arg2);
// Returns a timer ID
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

`clearTimeout` is used to cancel the execution (in case we change our mind). For example:

```
let timerId = setTimeout(() => alert("never"), 1000);
clearTimeout(timerId); // Cancels the execution
```

The `setInterval` method has a similar syntax as `setTimeout`:

```
let timerId = setInterval(function, delay, arg1, arg2);
```

All arguments have the same meaning. But unlike `setTimeout`, it runs the function not only once, but regularly after the given interval of time. To stop further calls, we can use `clearInterval(timerId)`.

Browser Events

An event is a signal that something has happened. All the DOM nodes generate such signals.

Some important DOM events are:

- Mouse events: `click`, `contextmenu` (right click), `mouseover`/`mouseout`, `mousedown`/`mouseup`, `mousemove`
- Keyboard events: `keydown` and `keyup`

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

- Form element events: `submit`, `focus`, etc.
- Document events: `DOMContentLoaded`

Handling Events

Events can be handled through HTML attributes:

```
<input value="Hey" onclick="alert('hey')" type="button" />
```

Events can also be handled through the `onclick` property:

```
elem.onclick = function () {
  alert("yes");
};
```

Note: Adding a handler with JavaScript overwrites the existing handler.

addEventListener and removeEventListener

`addEventListener` is used to assign multiple handlers to an event:

```
element.addEventListener(event, handler);
```

`removeEventListener` is used to remove a handler:

```
element.removeEventListener(event, handler);
```

The handler must be the same function object for this to work.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

The Event Object

When an event occurs, the browser creates an event object, puts details into it, and passes it as an argument to the handler:

```
elem.onclick = function (event) {
  // event.type: Event type
  // event.currentTarget: Element that handled the event
  // event.clientX / event.clientY: Coordinates of the cursor
};
```

This event object contains all the information about the event, including its type, the target element, and, for mouse events, the position of the cursor.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 8: Practice Set

1. Write a program to show different alerts when different buttons are clicked.
2. Create a website capable of storing bookmarks of your favorite websites using `href`.
3. Repeat question 2 using event listeners.
4. Write a JavaScript program to keep fetching the contents of a website every 5 seconds.
5. Create a "glowing bulb" effect using the `classList.toggle` method.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 9: Callbacks, Promises, & Async/Await

Asynchronous actions are actions that we initiate now and they finish later, e.g., `setTimeout`. Synchronous actions are actions that initiate and finish one by one.

Callback

A callback function is a function passed into another function as an argument, which is then invoked inside the outer function to complete an action.

Here is an example of a callback:

```
function loadScript(src, callback) {
  let script = document.createElement("script");
  script.src = src;
  script.onload = () => callback(script);
  document.head.append(script);
}

// Now we can do something like this:
loadScript("https://cdn.example.com", (script) => {
  alert("Script is loaded");
  alert(script.src);
});
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

This is called programming in a "callback-based style" of async. A function that does something asynchronously should provide a callback argument where we put the function to run after it's complete.

Handling Errors

We can handle callback errors by supplying an error argument like this:

```
function loadScript(src, callback) {
  let script = document.createElement("script");
  script.onload = () => callback(null, script);
  script.onerror = () => callback(new Error("Script load failed"));
  document.head.append(script);
}

// Then inside the loadScript call:
loadScript("https://cdn.example.com/harry.js", function (error, script) {
  if (error) {
    // Handle error
  } else {
    // Script loaded successfully
  }
});
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting

for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Pyramid of Doom

When we have callbacks inside callbacks, the code becomes difficult to manage. This is sometimes called "callback hell" or the "pyramid of doom".

As calls become more nested, the code becomes deeper and increasingly more difficult to manage, especially with real code instead of placeholders. The "pyramid" grows to the right and spirals out of control, making asynchronous control towards the right increasingly cumbersome. So, this way of coding is not very manageable.

Introduction to Promises

The solution to the callback hell is promises. A promise is essentially a "promise of code execution". Whether the code executes successfully or fails, the subscriber will be notified in both cases.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

The syntax of a Promise looks like this:

```
let promise = new Promise(function (resolve, reject) {
  // Promise executor
});
```

`resolve` and `reject` are two callbacks provided by JavaScript itself. They are called like this:

- `resolve(value)` - If the job is finished successfully.
- `reject(error)` - If the job fails.

The promise object returned by the `new Promise` constructor has these properties:

1. **State:** Initially "pending", then changes to either "fulfilled" when `resolve` is called, or "rejected" when `reject` is called.
2. **Result:** Initially `undefined`, then changes to `value` if resolved (`resolve(value)`) or `error` when rejected (`reject(error)`).

Consumers: then & catch

The consuming code can receive the final result or promise through `then` & `catch`.

The most fundamental one is `then`:

```
promise.then(
  function (result) {
    /* handle success */
  },
  function (error) {
    /* handle error */
  }
);
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

If we are interested only in successful completions, we can provide only one function argument to `then()`:

```
let promise = new Promise((resolve) => {
  setTimeout(() => resolve("done"), 1000);
});

promise.then(alert);
```

If we are interested only in errors, we can use `null` as the first argument, or we can use `catch`:

```
promise.then(null, alert);
// or
promise.catch(alert);
```

`promise.finally(() => {})` is used to perform general cleanups.

Quick Quiz: Rewrite the `loadScript` function we wrote at the beginning of this chapter using promises.

Promise Chaining

We can chain promises and make them pass the resolved values to one another like this:

```
a.then(function (result) {
  alert(result);
  return 2;
})
  .then
  // ...
  ();
// 'a' is a promise
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting

for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

The idea is to pass the result through the chain of `then` handlers.

Here is the flow of execution:

1. The initial promise resolves in 1 second (assumption).
2. The next `then()` handler is then called, which returns a new promise.
3. The next promise gets the result of the previous one and this keeps on going.

Every call to `then()` returns a new promise, whose value is passed on to the next one. We can even create custom promises inside `then()`.

Attaching Multiple Handlers

We can attach multiple handlers to one promise. They do not pass the result to each other; instead, they process it independently.

Let `p` be a promise:

```
p.then(handler1); // Handler 1
p.then(handler2); // Runs independently, handler 2
p.then(handler3); // Runs independently, handler 3
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Promise API

There are 6 static methods of the Promise class:

1. **`Promise.all(promises)`**: Waits for all promises to resolve and returns an array of their results. If any one fails, it results in an error & all other results are ignored.

2. **`Promise.allSettled(promises)`**: Waits for all the promises to settle and returns their results as an array of objects with status and value.

3. **`Promise.race(promises)`**: Waits for the first promise to settle, and its result/error becomes the outcome.

4. **`Promise.any(promises)`**: Waits for the first promise to settle (and not be rejected), and its result becomes the outcome. Throws an AggregateError if all the promises are rejected.

5. `Promise.resolve(value)`: Creates a resolved promise with the given value.

6. `Promise.reject(error)`: Creates a rejected promise with the given error.

Quick Quiz: Try all these promise APIs with custom promises on your own.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Async/Await

There is a special syntax to work with promises in JavaScript.

A function can be made asynchronous by using the `async` keyword like this:

```
async function harry() {
  return 7;
}
```

An async function always returns a promise. Other values are wrapped in a promise automatically.

We can do something like this:

```
harry().then(alert);
```

So, `async` ensures that the function returns a promise and wraps non-promises in it.

The await Keyword

There is another keyword called `await` that works only inside async functions:

```
let value = await promise;
```

The `await` keyword makes JavaScript wait until the promise settles and returns its value.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

It's just a more elegant syntax of getting the promise result, similar to the `then` method of promises, but it's easier.

Error Handling

We all make mistakes, and sometimes our scripts can have errors. Usually, a program halts when an error occurs. The `try...catch` syntax allows us to catch errors so that the script, instead of terminating, can do something more reasonable.

The `try...catch` Syntax

The `try...catch` system has two main blocks: `try` and `catch`.

```
try {
  // Try the code
} catch (err) {
  // Error handling
}
```

The `err` variable contains an error object.

It works like this:

1. First, the code in `try` is executed.
2. If there is no error, `catch` is ignored.
3. Else, `catch` is executed.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

`try...catch` works synchronously. If an exception happens in scheduled code, like in `setTimeout`, then `try...catch` won't catch it:

```
try {
  setTimeout(function () {
    // Error code
  }, 1000);
} catch (e) {
  // This won't work
}
```

That's because the function itself is executed later when the JavaScript engine has already left the `try...catch` construct.

The Error Object

For all built-in errors, the error object has two main properties:

```
try {
  hey; // Error: Variable not defined
} catch (err) {
```

```
  alert(err.name); // Name of the error
  alert(err.message); // Error message
  alert(err.stack); // Stack trace
}
```

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Throwing Custom Errors

We can throw our own errors by using the `throw` syntax:

```
if (age > 18 && age < 80) {
  throw new Error("Invalid Age");
}
```

We can also throw a particular built-in error by using the constructor for standard errors:

```
let error = new SyntaxError(message);
let error = new ReferenceError(message);
```

The `finally` Clause

The `try...catch` construct may have one more clause: `finally`. If it exists, it runs in all cases:

- After `try` if there were no errors.
- After `catch` if there were errors.

If there is a `return` in `try`, `finally` is executed just before the control returns to the outer code.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number. Here's the proofread version of the text with corrected grammar, punctuation, and formatting for clarity:

JavaScript Essentials

Date: [Insert Date Here]

Page: [Insert Page Number Here]

Chapter 9: Practice Set

1. Write a program to load a JavaScript file in a browser using Promises. Use `then()` to display an alert when the load is complete.

2. Write the same program and use `catch()` to handle errors.

3. Create a promise that rejects after 3 seconds. Use `async/await` to get its value. Use `try...catch` to handle its error.

4. Write a program using `Promise.all()` inside an `async/await` to await 3 promises. Compare its results with the case where we await these promises one by one.

Note: Ensure to replace "[Insert Date Here]" and "[Insert Page Number Here]" with the appropriate date and page number.