JavaScript

Module 2 - JavaScript Basics

Chapter 6 – Control Structures

Aitrich

What are Control Structures

Control structures are programming constructs that manage the flow of execution in a program. They determine the order in which statements are executed based on certain conditions.

Conditional Statements:

- If Statement: Allows the program to execute a block of code if a specified condition is true.
- Else Statement: Provides an alternative block of code to execute if the condition in the "if" statement is false.
- Else-If Statement: Enables the program to evaluate multiple conditions sequentially.

Aitrich

Looping Statements:

- For Loop: Repeats a block of code for a specified number of times.
- While Loop: Executes a block of code repeatedly as long as a specified condition is true.
- Do-While Loop: Similar to the "while" loop, but it always executes the block of code at least once.

Switch Statement:

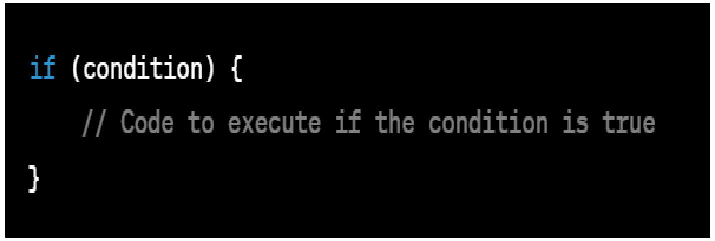
- Switch: Offers an efficient way to handle multiple conditions by selecting the appropriate code block based on the value of an expression.

Conditional Statements

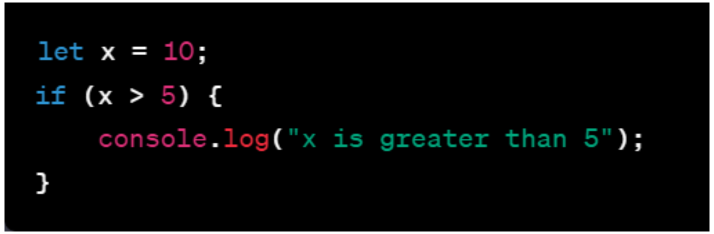
.

Types of Conditional Statements:

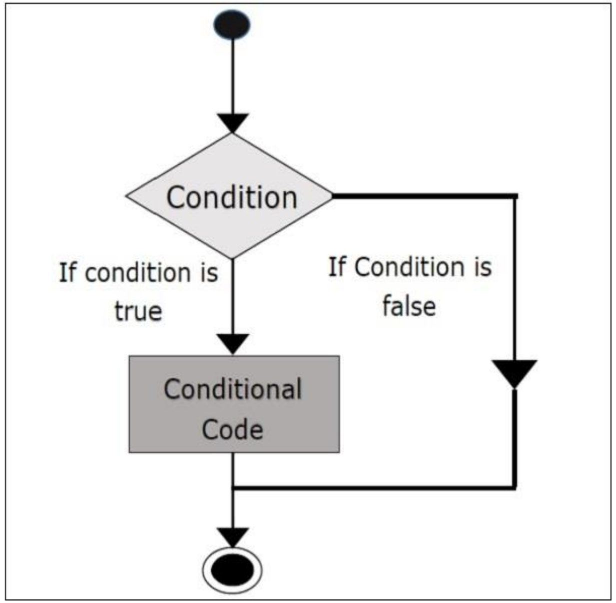
If Statement



```
if (condition) {
    // Code to execute if the condition is true
}
```



```
let x = 10;
if (x > 5) {
    console.log("x is greater than 5");
}
```



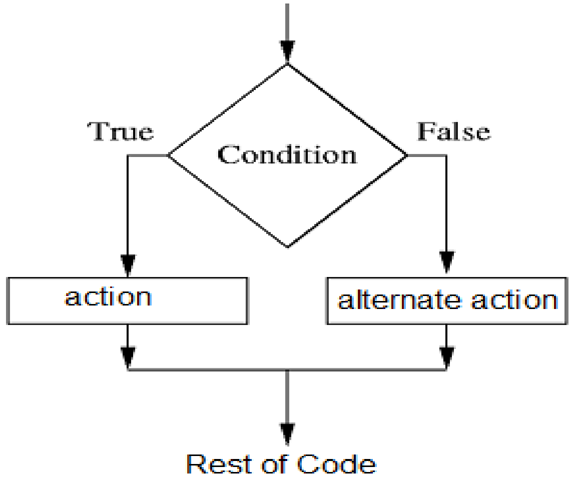
Aitrich

if-Else Statement:

- Used in conjunction with the "if" statement.
- Code to execute if the condition in "if" is false.

Example :

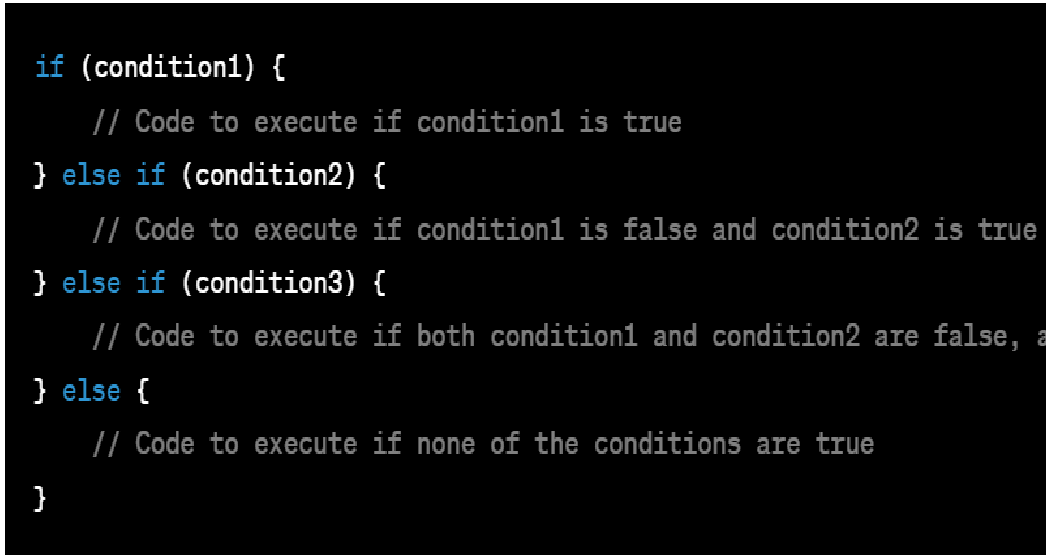
```
let y = 3;
if (y > 5) {
    console.log("y is greater than 5");
} else {
    console.log("y is not greater than 5");
}
```



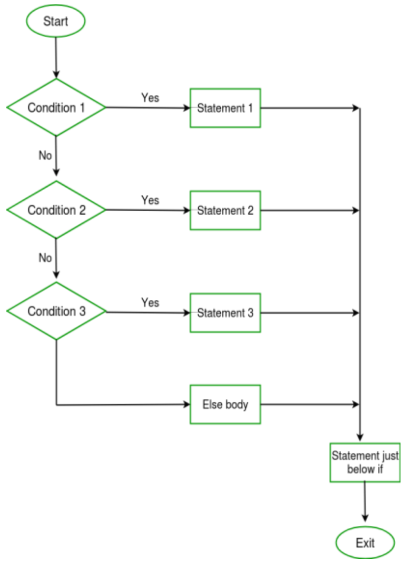
Else If Statement:

- The "else if" statement allows for multiple conditions to be checked in a sequence.
- It is an extension of the "if" and "else" statements, providing a more nuanced decision-making structure.

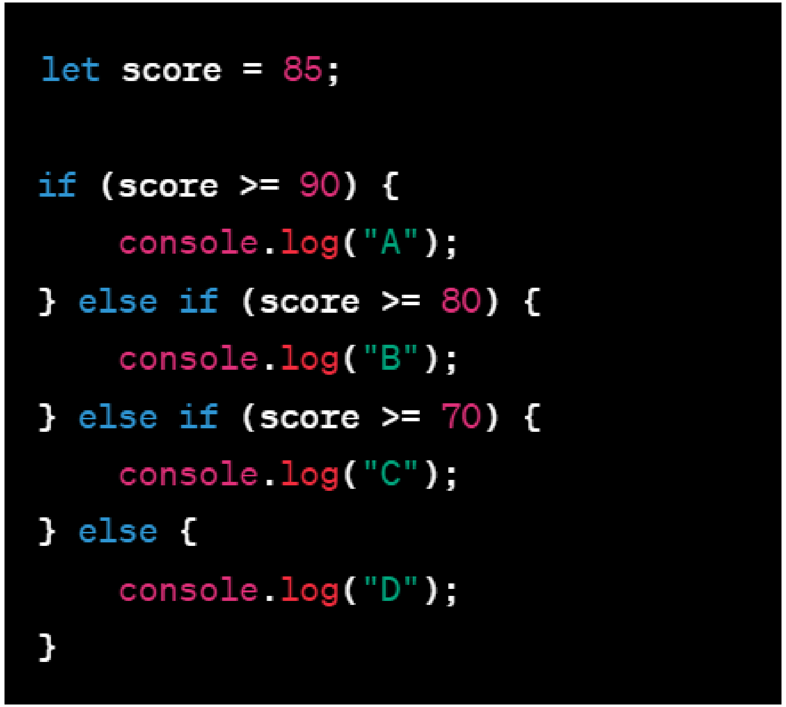
Syntax :



```
if (condition1) {
    // Code to execute if condition1 is true
} else if (condition2) {
    // Code to execute if condition1 is false and condition2 is true
} else if (condition3) {
    // Code to execute if both condition1 and condition2 are false, a
} else {
    // Code to execute if none of the conditions are true
}
```



Example



```
let score = 85;

if (score >= 90) {
    console.log("A");
} else if (score >= 80) {
    console.log("B");
} else if (score >= 70) {
    console.log("C");
} else {
    console.log("D");
}
```

Explanation:

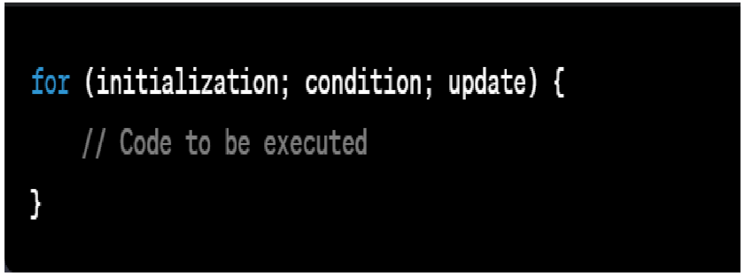
- The conditions are checked sequentially.
- If the first condition is true, the corresponding code block is executed, and the rest are skipped.
- If none of the conditions are true, the code inside the "else" block is executed.

Looping With For

Introduction to For Loops:

- A for loop is a control structure in JavaScript that allows you to repeatedly execute a block of code a certain number of times.
- **Initialization:** Executes once before the loop starts.
- **Condition:** Checked before each iteration. If false, the loop stops.
- **Update:** Executed after each iteration.

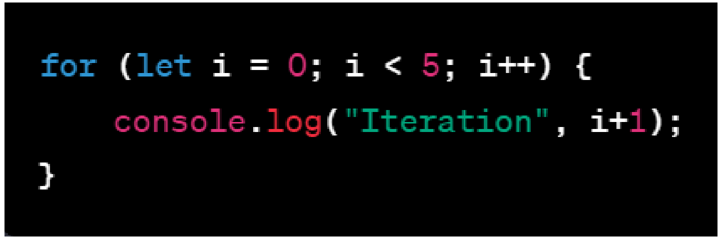
Basic Syntax:



```
for (initialization; condition; update) {
    // Code to be executed
}
```

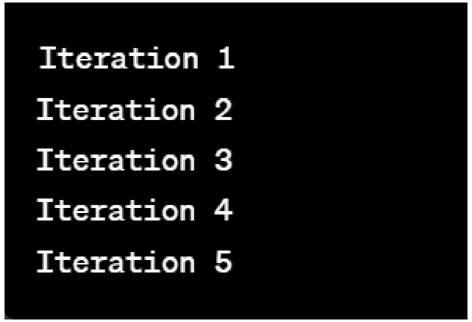
Looping With For

Example: Printing Numbers 1 to 5:



```
for (let i = 0; i < 5; i++) {
    console.log("Iteration", i+1);
}
```

Output



```
Iteration 1
Iteration 2
Iteration 3
Iteration 4
Iteration 5
```

Breakdown:

- **Initialization (let i = 0):** Sets the initial value of the loop variable.
- **Condition (i < 5):** Loop continues as long as this condition is true.
- **Update (i++):** Increments the loop variable after each iteration.

Looping With For

Use Cases:

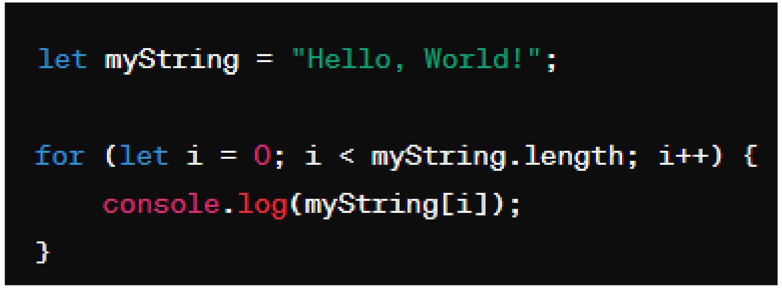
- Iterating over arrays.
- Executing a block of code a specific number of times.
- Any scenario where repetitive execution is required.

Iterate Over a String

Why Iterate Over a String?

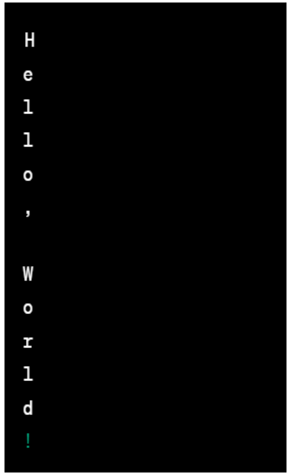
- Iterating over a string allows you to process each character individually, enabling various operations and manipulation.

Basic Looping Structure:



```
let myString = "Hello, World!";

for (let i = 0; i < myString.length; i++) {
    console.log(myString[i]);
}
```



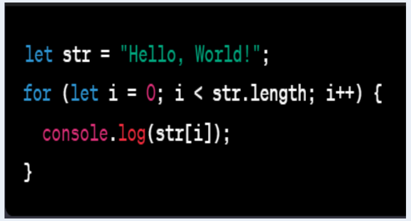
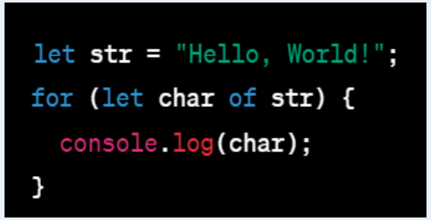
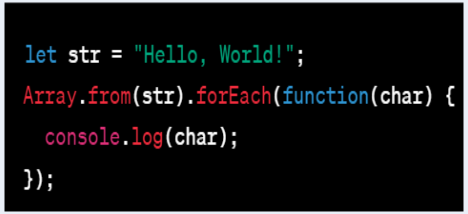
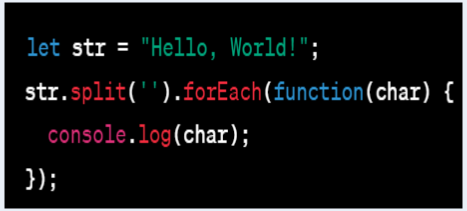
```
H
e
l
l
o
,

W
o
r
l
d
!
```

Explanation:

- str.length: Gets the length of the string.
- str[i]: Accesses the character at index i.

Iterate Over a String

For Loop	For...of Loop	forEach() Method	Split and Join
Traditional for loop is commonly used for iterating over strings.	Introduced in ECMAScript 2015, this loop simplifies iteration over iterable objects, including strings.	The forEach method is applicable to arrays but can be used on strings after converting them to arrays.	Another approach involves splitting the string into an array, iterating, and then joining back.
 `let str = "Hello, World!";` `for (let i = 0; i < str.length; i++) {` `console.log(str[i]);` `}`	 `let str = "Hello, World!";` `for (let char of str) {` `console.log(char);` `}`	 `let str = "Hello, World!";` `Array.from(str).forEach(function(char) {` `console.log(char);` `});`	 `let str = "Hello, World!";` `str.split('').forEach(function(char) {` `console.log(char);` `});`
Explanation: The loop runs from 0 to str. length - 1, accessing each character using the index.	**Explanation:** The loop iterates directly over each character in the string, making the code cleaner.	**Explanation:** Array. from() converts the string into an array and forEach iterates over each element.	**Explanation:** split('') splits the string into an array of characters, and forEach iterates over them.

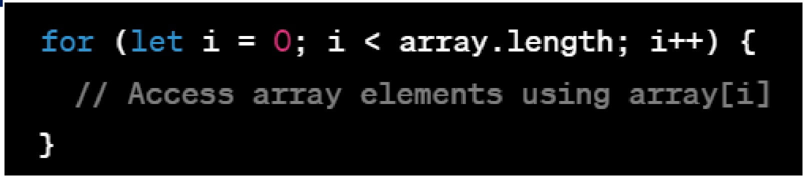
Iterate Over an Array

Introduction to Array Iteration:

- Arrays are fundamental data structures in JavaScript that allow you to store and organize multiple values in a single variable.
- Iterating over arrays is a crucial aspect of programming, enabling you to access and manipulate each element within the array.

➢ **For Loop:**

- The traditional for loop is a common choice for iterating over arrays.
- **Syntax :**



```
for (let i = 0; i < array.length; i++) {
  // Access array elements using array[i]
}
```

- This loop structure allows you to control the iteration process explicitly.

Iterate Over an Array

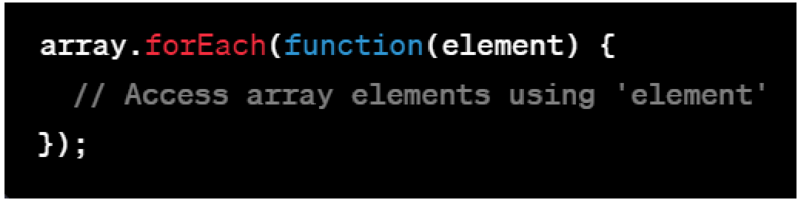
- **For...of Loop:**
  - Introduced in ECMAScript 2015 (ES6), the for...of loop provides a cleaner syntax for iterating over array elements.
  - It simplifies the code and enhances readability.
  - **Syntax:**

```
for (let element of array) {
  // Access array elements directly using 'element'
}
```

Iterate Over an Array

➢ ForEach Method:

- The forEach method is a built-in array method that executes a provided function once for each array element.
- It is particularly useful for applying a function to each element without the need for an explicit loop.
- **Syntax**:

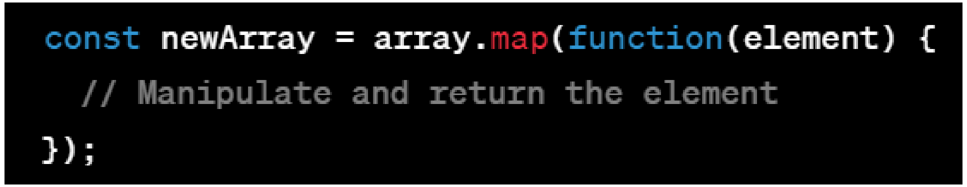


```
array.forEach(function(element) {
  // Access array elements using 'element'
});
```

Iterate Over an Array

➢ **Map Method:**

- The map method creates a new array by applying a function to each element in the existing array.
- This method is excellent for transforming array elements.
- **Syntax:**

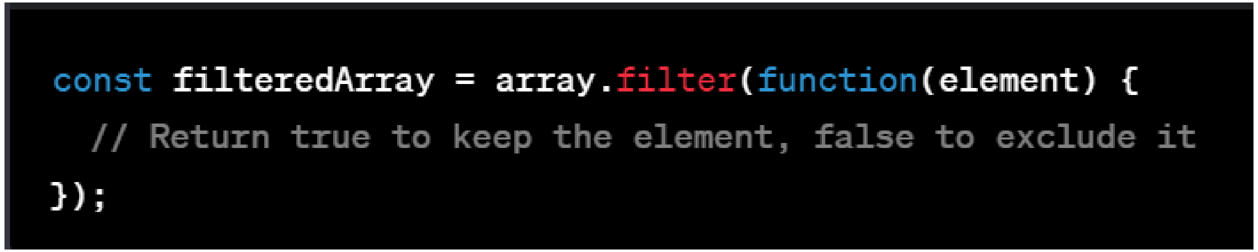


```
const newArray = array.map(function(element) {
  // Manipulate and return the element
});
```

Iterate Over an Array

➢ **Filter Method:**

- The filter method creates a new array containing elements that satisfy a given condition.
- Useful for creating a subset of elements based on specific criteria.
- **Syntax:**

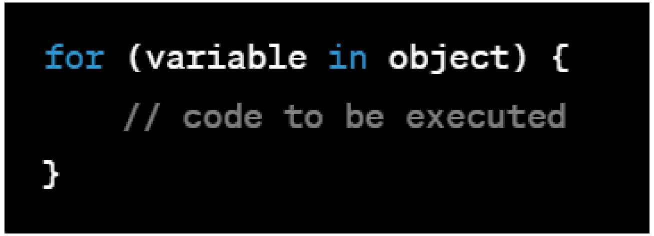


```
const filteredArray = array.filter(function(element) {
  // Return true to keep the element, false to exclude it
});
```

Looping With For...in

What is the for...in Loop?

- The for...in loop is a JavaScript control structure designed for iterating over the properties of an object. It provides an easy way to access and manipulate each property within an object.
- **Syntax**:



```
for (variable in object) {
    // code to be executed
}
```

  - **variable:** A variable that will be assigned the property name on each iteration.
  - **object:** The object whose properties are being iterated.

Looping With For...in

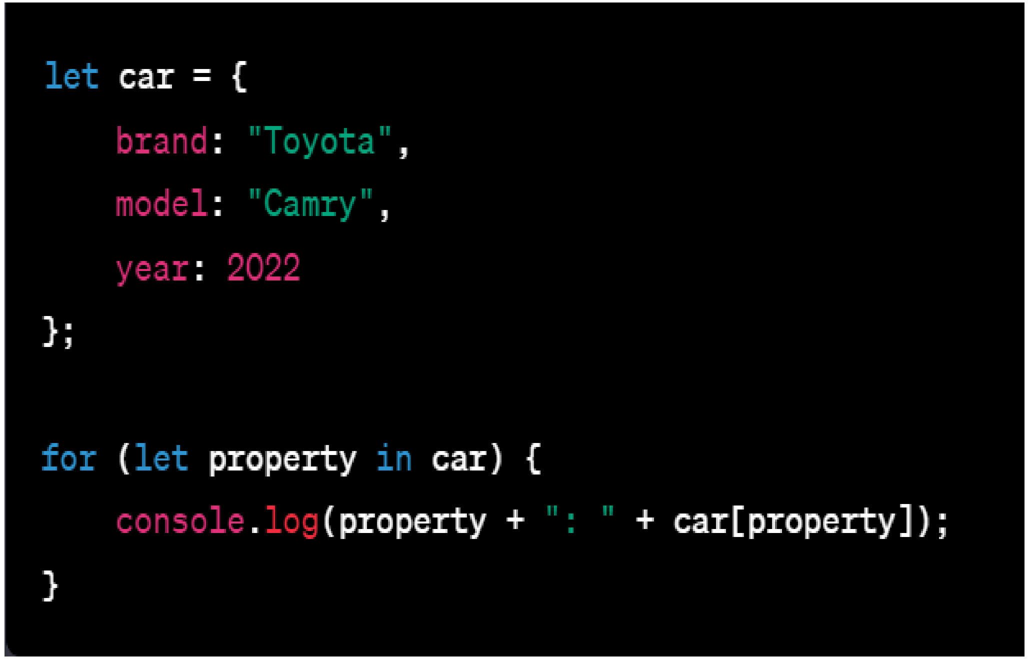
How It Works

- **Initialization:** The loop begins by initializing the variable with the name of the first property in the object.
- **Condition:** The loop continues as long as there are more properties in the object.
- **Iteration:** On each iteration, the variable holds the name of the current property, allowing you to perform actions based on it.
- **Exit:** The loop exits when all properties have been processed.

Important Considerations

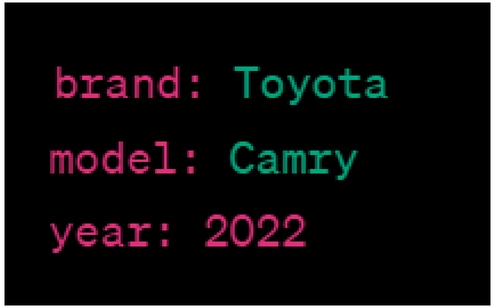
- **Order of Iteration:** The order of properties in the for...in loop is not guaranteed to be in a specific order. It is typically based on the internal order in which properties were added to the object.
- **Inherited Properties:** The loop iterates over all enumerable properties, including those inherited from the object's prototype chain.

Example



```
let car = {
    brand: "Toyota",
    model: "Camry",
    year: 2022
};

for (let property in car) {
    console.log(property + ": " + car[property]);
}
```



```
brand: Toyota
model: Camry
year: 2022
```

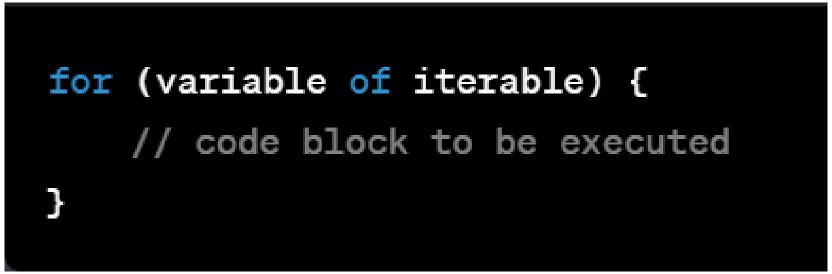
Output

Iterating Over Object Properties

Looping With For...of

Understanding the for...of Loop:

- Introduced in ECMAScript 2015 (ES6), the for...of loop simplifies the process of iterating over iterable objects.
- Iterable objects include arrays, strings, maps, sets, and more.
- The loop iterates over each element in the iterable, assigning it to the variable.
- **Syntax:**



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

Looping With For...of

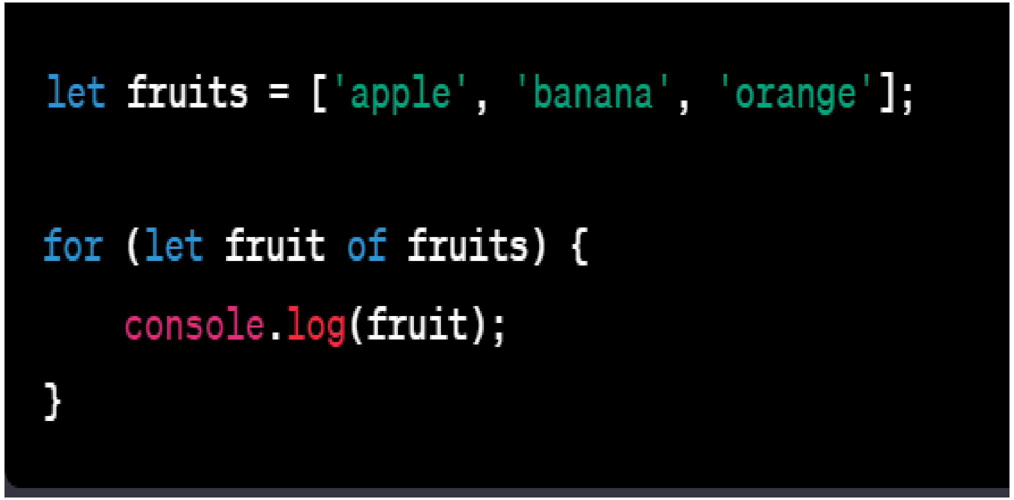
Benefits of for...of:

- **Simplicity:** Cleaner and more readable syntax compared to traditional for loops.
- **Iterable Objects**: Works seamlessly with a variety of iterable objects.
- **No Index Handling:** Eliminates the need for manual index handling, making the code less error-prone.

Limitations:

- While powerful, the for...of loop is not suitable for all scenarios, especially when you need access to the index or need to iterate over non-iterable objects.

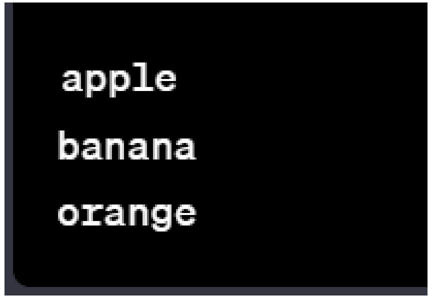
Example



```
let fruits = ['apple', 'banana', 'orange'];

for (let fruit of fruits) {
    console.log(fruit);
}
```

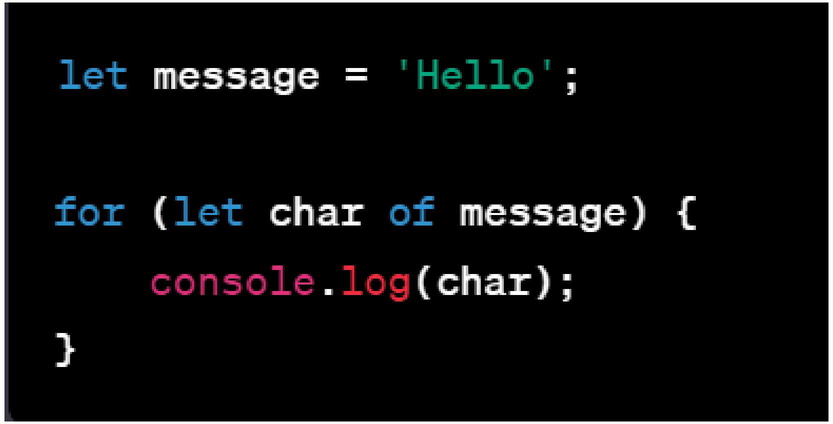
Iterating Over an Array



```
apple
banana
orange
```

Output

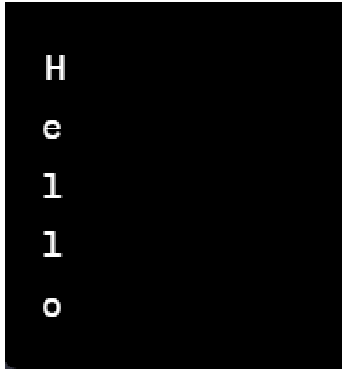
Example



```
let message = 'Hello';

for (let char of message) {
     console.log(char);
}
```

Iterating Over a String

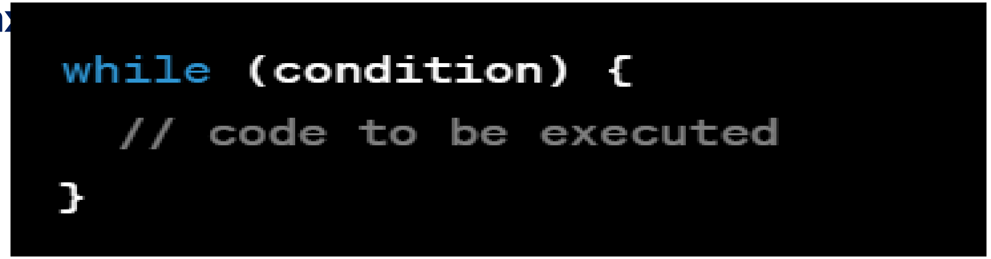


```
H
e
l
l
o
```

Output

Looping With while

- JavaScript offers several loop structures, and one of the basic ones is the "while" loop.
- The while loop is a foundational control flow structure in JavaScript that facilitates the execution of a block of code repeatedly as long as a specified condition remains true.
- **Synta**



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

Looping With While

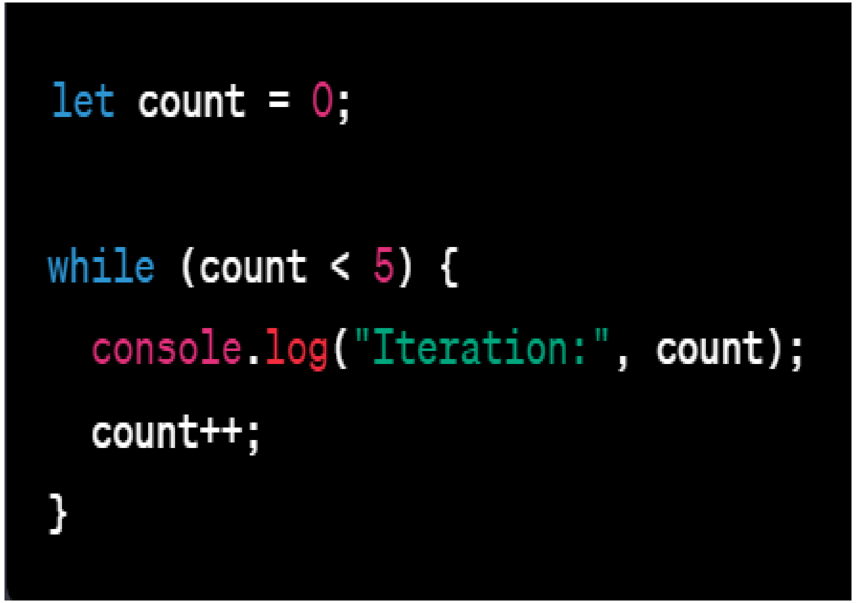
How it Works:

- The loop continues executing the block of code as long as the specified condition evaluates to true.
- Once the condition becomes false, the loop terminates, and the program moves to the next statement.

Key Points

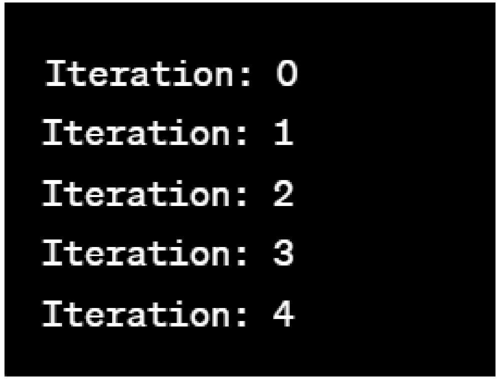
- **Initialization:** Ensure that any variables used in the condition are initialized before entering the loop.
- **Updating the Condition:** Inside the loop, make sure to modify the condition to eventually become false; otherwise, you might end up with an infinite loop.
- **Avoiding Infinite Loops**: Infinite loops can lead to program crashes; always ensure a proper exit condition.

Example



```
let count = 0;

while (count < 5) {
  console.log("Iteration:", count);
  count++;
}
```



```
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
```

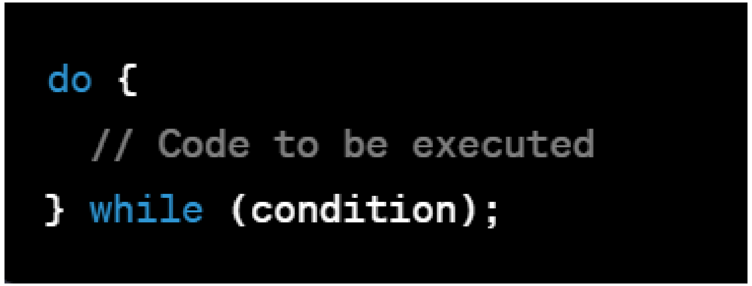
Output

Looping With do while

Introduction to Loops:

- JavaScript supports several loop types, and one of them is the "do-while" loop.
- The block of code inside the "do" statement is executed at least once, and then the condition is checked. If the condition is true, the loop continues; otherwise, it exits.

- **Syntax of the Do-While Loop:**



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

Looping With do while

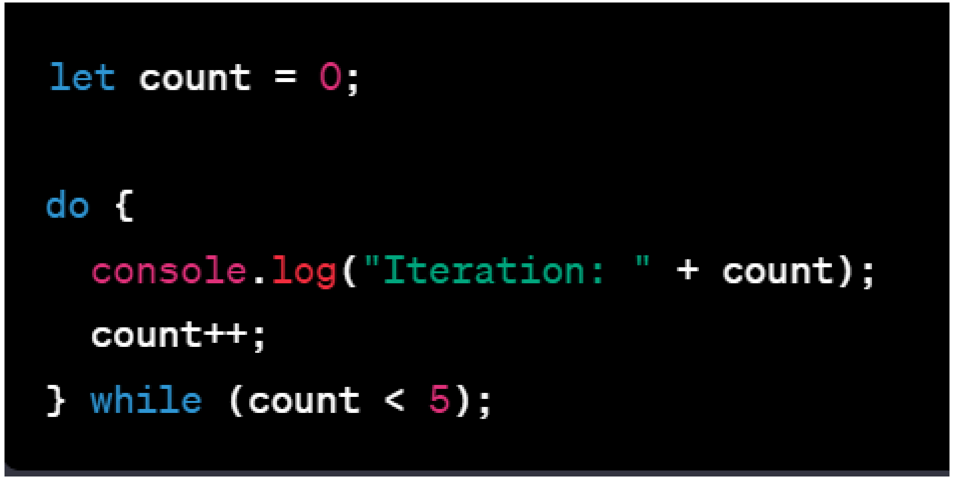
Execution Flow:

- The key feature of the do-while loop is that the code block is executed before checking the condition.
- This ensures that the block of code will be executed at least once, regardless of whether the condition is initially true or false.

Use Cases:

- Do-while loops are particularly useful when you need to execute a block of code at least once, and then repeat it based on a certain condition.
- Example scenarios include user input validation, menu-driven programs, and iterative processes where the number of iterations is not known in advance.

Example



```
let count = 0;

do {
  console.log("Iteration: " + count);
  count++;
} while (count < 5);
```

- This example prints "Iteration: 0" to "Iteration: 4" as the loop iterates five times.

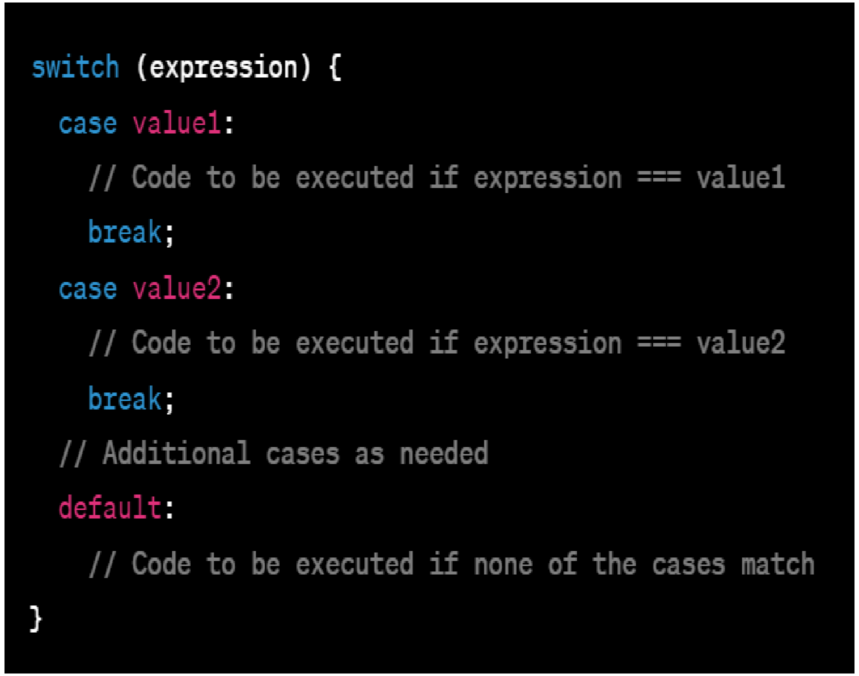
Switch Statements for Multi-Case Conditions in JavaScript

In JavaScript, switch statements provide a concise way to handle multiple possible conditions. Unlike if-else statements, which evaluate conditions one by one, switch statements are particularly useful when dealing with a variable that can have multiple values.

Benefits of Switch Statements

- **Readability**: Switch statements make the code more readable, especially when dealing with multiple conditions.
- **Performance**: In some cases, switch statements can be more efficient than equivalent if-else chains.

Syntax

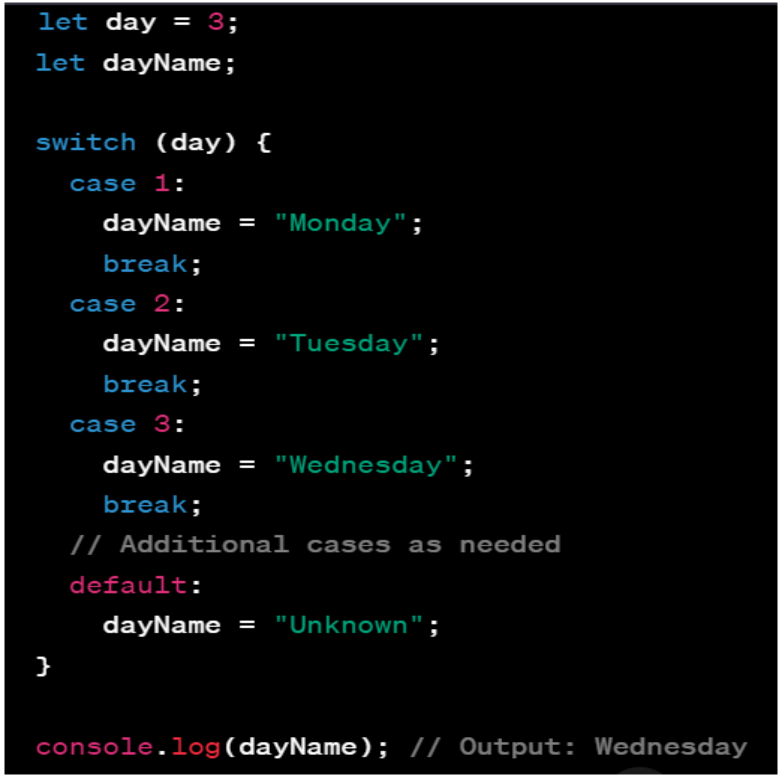


```
switch (expression) {
  case value1:
    // Code to be executed if expression === value1
    break;
  case value2:
    // Code to be executed if expression === value2
    break;
  // Additional cases as needed
  default:
    // Code to be executed if none of the cases match
}
```

How It Works

- The expression is evaluated once, and its value is compared with each case.
- If a match is found, the corresponding block of code is executed.
- The break statement terminates the switch, preventing fall-through to subsequent cases

Example



```
let day = 3;
let dayName;

switch (day) {
  case 1:
    dayName = "Monday";
    break;
  case 2:
    dayName = "Tuesday";
    break;
  case 3:
    dayName = "Wednesday";
    break;
  // Additional cases as needed
  default:
    dayName = "Unknown";
}

console.log(dayName); // Output: Wednesday
```

Considerations

Strict Comparison: Switch statements use strict comparison (===), ensuring both value and type match.

Thank You