AIMERZ.ai
Aim.Act.Achieve

NOTES

Conditionals & Loops

Conditionals

Introduction of Conditionals

Conditional statements in programming allow your code to make decisions and perform different actions based on certain conditions. Think of it as telling your program, **"If this happens, do this; otherwise, do something else."**

Here's an easy way to understand:

Everyday Example:

- **If it's raining**, take an umbrella.
- **If it's sunny**, wear sunglasses.

Conditional statements control behavior in JavaScript and determine whether or not pieces of code can run.

There are multiple different types of conditionals in JavaScript

- **if:** Executes a block of code only if a specified condition is true.
- **if else:** Executes one block of code if the condition is true, and another block if it's false.
- **nested if:** Places one if statement inside another if statement to check more specific conditions.
- **if else if ladder:** Checks multiple conditions in sequence, executing the first true condition.

Lets's go through each and every type of conditional in JavaScript.

If Statement

The **if statement** allows your program to run a specific block of code only when a certain condition is true.

Syntax:

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

Explanation:

- **condition**: This is a test that can be either true or false (like comparing numbers or checking if something exists).
- If the condition is true, the code inside the curly braces {} runs.
- If the condition is false, the code inside the braces does not run.

Example 1:
We want to check if a person is old enough to vote.

```javascript
var age = 20;

if (age >= 18) {
    console.log("You are eligible to vote.");
}

//Output: You are eligible to vote.
```

Example 2:
Check, the given number is Odd or Even.

```javascript
var num = 20;

if (num % 2 === 0) {
    console.log("Given number is even number.");
}

if (num % 2 !== 0) {
    console.log("Given number is odd number.");
};
//Output:Given number is even number.
```

Example 3: Grade Calculator

Problem Statement: Let's build a system that gives students their grades! What do we want to do?

1. Check if a student passes based on their test score and attendance.
2. If a student scores 90 or higher *and* has at least 75% attendance, they get **Grade A**.
3. If a student scores between 80 and 89 *and* has at least 75% attendance, they get **Grade B**.
4. If the student's score is below 80 or their attendance is below 75%, they need to improve.

```javascript
let score = 85;
let attendance = 80;

if (score >= 90 && attendance >= 75) {
    console.log("Grade: A - Excellent performance!");
}
if (score >= 80 && score < 90 && attendance >= 75) {
    console.log("Grade: B - Good job!");
}
if (score < 80 || attendance < 75) {
    console.log("Need improvement - Please see the teacher.");
}
// Output: Grade: B - Good job!
```

Example 4: Movie Theater Ticket System Problem Statement: Problem Statement: Help a movie theater set ticket prices! We need to check:

What this system should do:

1. Check the **age** of the person buying the ticket.
2. Check if they are a **student**.
3. Check if it's a **weekend**.

Ticket Prices Based on These Rules:

- **Child (Under 12 years):** $8
- **Student (12 to 18 years and is a student):** $10
- **Senior Citizen (Over 60 years):** $9
- **Adult (18 to 60 years and not a student):**
  -Weekend price: $15
  -Weekday price: $12

JavaScript
```javascript
let age = 15;
let isStudent = true;
let isWeekend = false;

if (age < 12) {
    console.log("Child Ticket Price: $8");
}
if (age >= 12 && age <= 18 && isStudent) {
    console.log("Student Ticket Price: $10");
}
if (age > 60) {
    console.log("Senior Citizen Price: $9");
}
if (age >= 18 && age <= 60 && !isStudent) {
    if (isWeekend) {
        console.log("Weekend Adult Price: $15");
    }
    if (!isWeekend) {
        console.log("Weekday Adult Price: $12");
    }
}
// Output: Student Ticket Price: $10
```

If else Statement

The **if else** statement allows your program to choose between two different actions based on whether a condition is true or false.

Syntax:

```javascript
if (condition) {
    // code to execute if the condition is true
} else {
    // code to execute if the condition is false
}
```

Explanation:

- **condition**: This is the test that you want to check (it can be true or false).
- If the condition is true, the code inside the first block (the if block) runs.
- If the condition is false, the code inside the else block runs.

Example 1:
Let's create a simple program to check if a person is old enough to drive.

```javascript
var age = 16; // You can change this value to test different
ages

if (age >= 18) {
    console.log("You are old enough to drive.");
} else {
    console.log("You are not old enough to drive.");
}

//Output:You are not old enough to drive.
```

Example 2:
Checking if a Number is Even or Odd

```
JavaScript
var number = 7; // You can change this value to test different
numbers

if (number % 2 === 0) {
    console.log("The number is even.");
} else {
    console.log("The number is odd.");
}

//Output:The number is odd.
```

Example 3:
Ticket Price Discounts

A movie theater wants to set ticket prices based on the age of the customer. Write a program to determine the correct ticket price for each age group.

Ticket Pricing Rules:

1. **Children** (under 12 years) get a ticket for **$8**.
2. **Adults** (12 to 64 years) pay **$12**.
3. **Seniors** (65 years and older) get a ticket for **$10**.

```
JavaScript
let customerAge = 12;
let ticketPrice;

if (customerAge < 12) {
    ticketPrice = 8;
    console.log("Child ticket price: $" + ticketPrice);
} else if (customerAge >= 12 && customerAge <= 64) {
    ticketPrice = 12;
    console.log("Adult ticket price: $" + ticketPrice);
} else {
    ticketPrice = 10;
    console.log("Senior ticket price: $" + ticketPrice);
}
```

Example 4:
Vacation Planner

What this planner should do:

1. Check the **weather forecast**.
2. Check what **day of the week** it is.
3. Suggest activities based on these factors.

Activity Suggestions:

- If it's **sunny** and it's **Saturday**: Go to the beach!
- If it's **sunny** but not Saturday: Go for a hike in the park.
- If it's **rainy**: Visit the museum or watch a movie.
- If it's **snowy**: Go skiing or snowboarding!
- If none of these conditions apply: Stay indoors and relax.

JavaScript
```
let weatherForecast = "sunny";
let dayOfWeek = "Saturday";

if (weatherForecast === "sunny" && dayOfWeek === "Saturday") {
    console.log("It's a perfect day to go to the beach!");
} else if (weatherForecast === "sunny" && dayOfWeek !==
"Saturday") {
    console.log("Go for a hike in the park!");
} else if (weatherForecast === "rainy") {
    console.log("Visit the museum or catch a movie.");
} else if (weatherForecast === "snowy") {
    console.log("Hit the slopes for some skiing or
snowboarding!");
} else {
    console.log("Stay indoors and relax - it's not the best
day for outdoor activities.");
}
```

Nested if Statement

A **nested** if statement is an if statement inside another if statement. This allows you to check multiple conditions in a structured way.

Syntax:

```
JavaScript
if (condition1) {
    // code to execute if condition1 is true
    if (condition2) {
        // code to execute if condition2 is true
    }
}
```

Explanation:

- **condition1**: The outer condition that is checked first.
- If **condition1** is true, the program will then check **condition2**.
- If both conditions are true, the code inside the inner if block runs.

Example 1:

Let's create a program that checks a person's age to determine their eligibility for a driver's license, and then checks if they have passed the driving test.

JavaScript
```
var age = 20; // Change this value to test different ages
var passedTest = true; // Change this value to test whether
the person passed the test

if (age >= 18) { // Check if the person is 18 or older
    console.log("You are eligible to apply for a driver's
license.");

    if (passedTest) { // Check if the person passed the
driving test
        console.log("You can get your driver's license.");
    } else {
        console.log("You need to pass the driving test to get
your license.");
    }
} else {
    console.log("You are not eligible to apply for a driver's
license.");
}

//Output: You are eligible to apply for a driver's license.
//         You can get your driver's license.
```

Example 2:

Let’s create a another simple example using a nested `if` statement to determine whether a person qualifies for a discount based on their age and membership status.

JavaScript
```
var age = 25; // Change this value to test different ages

var isMember = true; // Change this value to test membership
status
```

```
if (age >= 60) { // Check if the person is a senior citizen
    console.log("You qualify for a senior discount.");
} else { // If not a senior citizen
    if (isMember) { // Check if the person is a member
        console.log("You qualify for a member discount.");
    } else {
        console.log("You do not qualify for any discounts.");
    }
}

//Output: You qualify for a member discount.
```

if else if Ladder Statement

The **if else if ladder** allows you to check multiple conditions in sequence. It helps you choose one action from many possible actions based on different conditions.

Syntax:

```
JavaScript
if (condition1) {
    // code to execute if condition1 is true
} else if (condition2) {
    // code to execute if condition2 is true
} else if (condition3) {
    // code to execute if condition3 is true
} else {
    // code to execute if none of the conditions are true
}
```

Explanation:

- **condition1, condition2, condition3**: These are the tests you want to check in order.

- The program checks each condition one by one:
  - If **condition1** is true, the code inside its block runs, and the rest are skipped.
  - If **condition1** is false, it checks condition2. If **condition2** is true, it runs that block and skips the rest.
  - This continues until it finds a true condition or reaches the final else block if none are true.

Example 1: Grading System
Let's create a program that assigns a grade based on a score.

```javascript
var score = 85; // Change this value to test different scores

if (score >= 90) {
    console.log("Grade: A");
} else if (score >= 80) {
    console.log("Grade: B");
} else if (score >= 70) {
    console.log("Grade: C");
} else if (score >= 60) {
    console.log("Grade: D");
} else {
    console.log("Grade: F");
}

//Output: Grade: B
```

Example 3: Season Checker

Let's write a program to find out the current season based on the month of the year!

What this program should do:

1. Check the **month** (number from 1 to 12).
2. Determine the season based on the month.

Seasons Based on Months:

- **Spring:** March (3), April (4), May (5)
- **Summer:** June (6), July (7), August (8)
- **Fall (Autumn):** September (9), October (10), November (11)
- **Winter:** December (12), January (1), February (2)

JavaScript
```javascript
let month = 6; // Change this value to test different months

if (month === 3 || month === 4 || month === 5) {
    console.log("It's Spring!");
} else if (month === 6 || month === 7 || month === 8) {
    console.log("It's Summer!");
} else if (month === 9 || month === 10 || month === 11) {
    console.log("It's Fall!");
} else {
    console.log("It's Winter!");
}

// Output: It's Summer!
```

JavaScript Loops

JavaScript loops are very helpful when you want to repeat the same task multiple times. They allow you to run a block of code over and over, as long as a certain condition is true. This is a great way to save time and make your code shorter and easier to read.

For example, let's say we want to print "Hello World" 5 times. Instead of writing the print statement 5 times, we can use a loop, which only needs the statement written once but will automatically repeat it 5 times.

Advantages of Loops in JavaScript:

1. **Code Reusability:** Loops allow us to reuse code, so we don't have to write the same instructions over and over.

2. **Less Repetition:** With loops, we avoid repeating the same code multiple times, making programs shorter and easier to read.
3. **Easy Data Traversal:** Loops make it simple to go through elements in data structures like arrays or linked lists, allowing us to work with each element one by one.

There are different types of loops in JavaScript.

- **For loop:** A `for` loop repeats code a specific number of times. It's useful when you know exactly how many times you want the code to run.

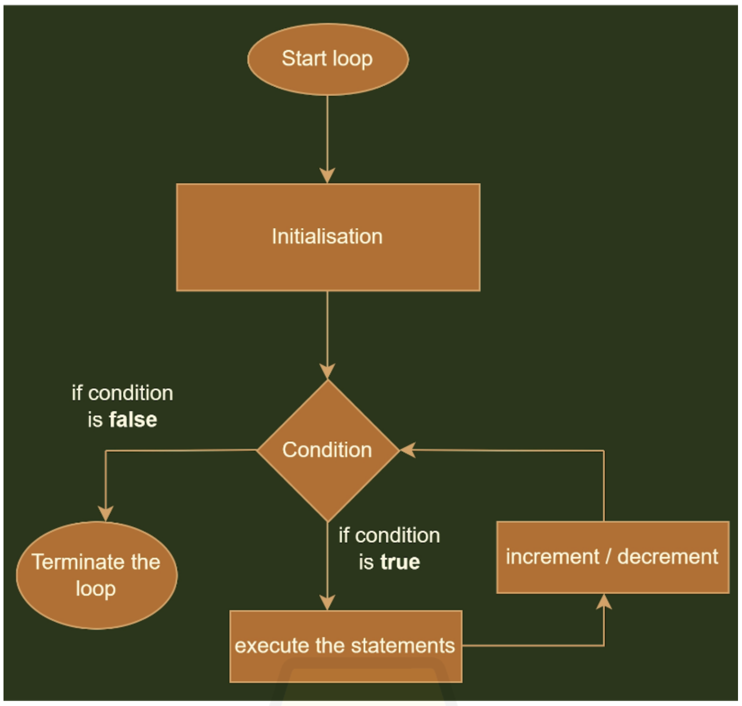
- **While loop:** A while loop keeps running as long as a condition is true. It's useful when you don't know exactly how many times the code should repeat.

- **Do while loop:** A do...while loop runs the code once, then keeps repeating it as long as the condition is true. This loop always runs at least one time.

For Loop

A **for loop** is a way to repeat a block of code a specific number of times. It's helpful when you know exactly how many times you want something to happen. For example, if you want to print "Hello" 5 times, a for loop can help do it without writing `console.log("Hello")` five times.

A **for loop** is a control flow statement in JavaScript that allows code to be executed repeatedly based on a starting point, a condition, and an increment or decrement. The loop runs until the specified condition becomes false.



Syntax

```
JavaScript
for (initialization; condition; increment/decrement) {
    // code to run in each loop
}
```

- **Initialization**: Set the starting value (often a counter).
- **Condition**: The loop continues as long as this is true.
- **Increment/Decrement**: Changes the counter by adding or subtracting after each loop.

Example:

```
JavaScript
for (let i = 0; i < 5; i++) {
    console.log("Hello");
}
```

Explanation:

- `let i = 0;` sets the starting value of `i` to 0.
- `i < 5;` tells the loop to keep running as long as `i` is less than 5.

- `i++` increases `i` by 1 after each run of the loop.

This example prints "Hello" 5 times.

Example 1:

Printing Numbers

This example shows how to print numbers from 1 to 5 using a for loop.

```javascript
JavaScript
for (let i = 1; i <= 5; i++) {
    console.log(i);
}
```

-

Explanation:

- This loop starts at 1 and goes up to 5.
- Each time the loop runs, it prints the current value of i.

  The output will be:

  1

  2

  3

  4

  5

Example 2:

Summing Numbers

In this example, we'll calculate the sum of the numbers from 1 to 5.

```
JavaScript
let sum = 0; // Start with a sum of 0

for (let i = 1; i <= 5; i++) {
    sum += i; // Add the current value of i to sum
}

console.log("The sum is: " + sum);
```

Explanation:

- We initialize sum to 0.
- The loop adds each number from 1 to 5 to sum.
- After the loop finishes, it prints the total sum, which will be 15.

Example 3:

Multiplication Table

This example generates a multiplication table for the number 3.

```
JavaScript
let number = 3; // The number for which we want the multiplication table

console.log("Multiplication Table for " + number);
for (let i = 1; i <= 10; i++) {
    console.log(number + " x " + i + " = " + (number * i));
}
```

Explanation:

- We set number to 3.
- The loop runs from 1 to 10.
- In each iteration, it multiplies number by i and prints the result.

  The output will look like this:

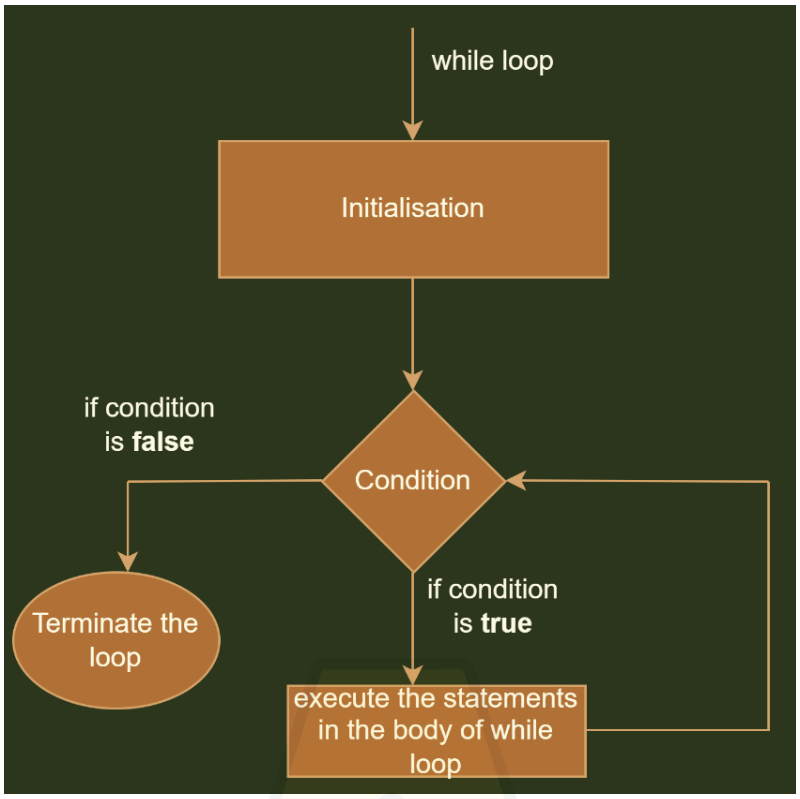
  Multiplication Table for 3

```
3 x 1 = 3

3 x 2 = 6

3 x 3 = 9

3 x 4 = 12

3 x 5 = 15

3 x 6 = 18

3 x 7 = 21

3 x 8 = 24

3 x 9 = 27

3 x 10 = 30
```

While Loop

A **while loop** is a way to repeat a block of code as long as a certain condition is true. It's useful when you don't know in advance how many times you will need to run the code. For example, if you want to keep asking a user for input until they enter a specific word, a while loop can help.

A **while loop** is a control flow statement that repeatedly executes a block of code as long as a specified condition evaluates to true. If the condition is false, the loop stops.



Syntax

```
JavaScript
while (condition) {
    // code to run as long as the condition is true
}
```

Condition: This is a statement that is checked before each loop. The loop continues to run as long as this condition is true.

Example

```
JavaScript
let i = 1; // Start with 1
while (i <= 5) { // Continue while i is less than or equal to 5
    console.log(i); // Print the value of i
    i++; // Increase i by 1
}
```

Explanation:

- `let i = 1;` initializes the counter `i` at 1.
- `i <= 5;` checks if `i` is less than or equal to 5. If it is, the loop continues.
- `console.log(i);` prints the current value of `i`.
- `i++;` increases `i` by 1 after each loop.

The output will be:
1
2
3
4
5

Example: Counting Down

This example counts down from 5 to 1.

```
JavaScript
let count = 5; // Start at 5

while (count > 0) { // Continue as long as count is greater than 0
    console.log(count); // Print the current count
    count--; // Decrease count by 1
}
```

Explanation:

- The loop starts with count set to 5.
- It checks if count is greater than 0. If it is, the loop runs.
- Each time it runs, it prints the current value of count, then subtracts 1 from count.

  The output will be:

  5

  4

  3

2

1

Infinite Loops in While Loops

What is an Infinite Loop?

An **infinite loop** is a loop that never stops running because its condition is always true. This happens when the loop is missing a way to finish, usually because we forgot to change a value inside the loop.

Example of an Infinite Loop

Here's a simple example of an infinite loop:

```javascript
let i = 1; // Start at 1

while (i <= 5) { // This condition will always be true
    console.log(i); // Print the current value of i
    // We forgot to change i here, so it will always stay 1
}
```

Explanation:

- The loop starts with `i` at 1.
- The condition `i <= 5` is always true because we never change `i` inside the loop.
- So, it keeps printing `1` forever, and that's why it's called an infinite loop.

How to Stop an Infinite Loop

If you find yourself in an infinite loop:

- You can close the tab or window if you are running the code in a web browser.
- In a coding program, you can usually stop it using a stop button or by pressing `Ctrl + C`.

How to Avoid Infinite Loops

To avoid making an infinite loop, make sure:

- The loop condition will eventually become false.
- You change the variables inside the loop that control how long it runs.

Example of a Safe While Loop

Here's a safe version that does not create an infinite loop:

```javascript
let i = 1; // Start at 1

while (i <= 5) { // This condition is true until i is greater than 5
    console.log(i); // Print the current value of i
    i++; // Increase i by 1, so the loop will eventually stop
}
```

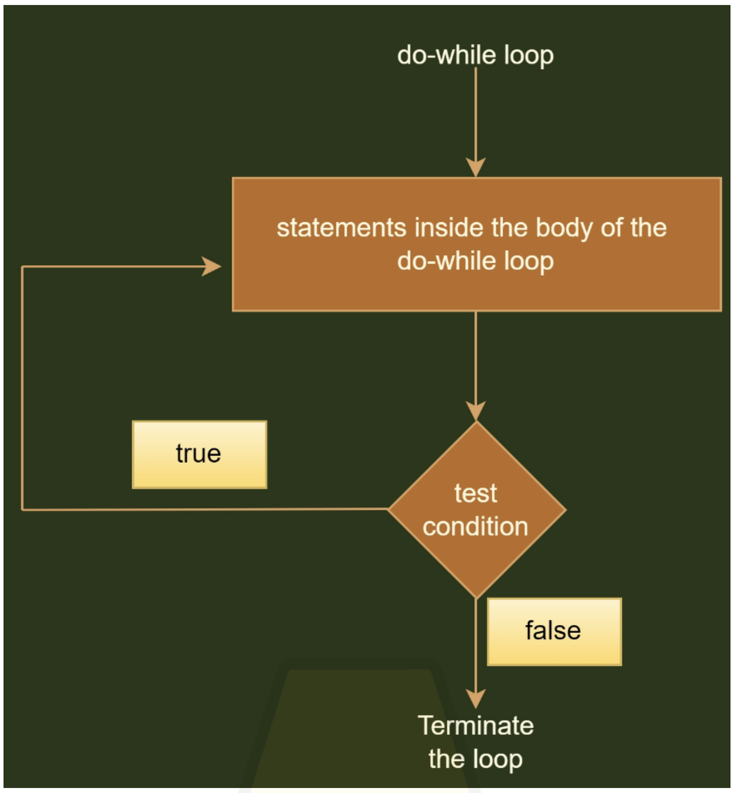
Explanation:

- In this example, i is increased inside the loop.
- When i becomes 6, the condition i <= 5 becomes false, and the loop stops.

Do-While Loop

A **do-while loop** is a type of loop that runs a block of code **at least once** and then continues to run as long as a specified condition is true. This means that the code inside the loop will always execute at least one time, even if the condition is false.

A **do-while loop** is a control flow statement that executes a block of code once and then repeatedly executes it as long as the given condition remains true. The condition is checked after each execution of the loop.



Syntax of a Do-While Loop

JavaScript
```javascript
do {
    // code to run at least once
} while (condition);
```

Code Block: The code inside the {} runs once before checking the condition.
Condition: After running the code, the loop checks this condition. If it's true, the code runs again.

Example

JavaScript
```javascript
let i = 1; // Start at 1

do {
    console.log(i); // Print the current value of i
    i++; // Increase i by 1
} while (i <= 5); // Continue as long as i is less than or equal to 5
```

Explanation:

- The loop starts with i set to 1.
- The code inside the do block runs first, printing the value of i.
- Then, it increases i by 1.
- After that, it checks if i is less than or equal to 5.

  This loop will print:

  1

  2

  3

  4

  5

Difference Between For Loop, While Loop, and Do-While Loop

Feature	For Loop	While Loop	Do-While Loop
Definition	Runs a block of code a specific number of times.	Runs a block of code as long as a condition is true.	Runs a block of code at least once, then checks the condition.
Structure	Combines initialization, condition, and increment in one line.	Has separate initialization, condition, and increment.	Has a code block that runs first, then checks the condition.
When to Use	Use when you know how many times to repeat.	Use when the number of repetitions is not known.	Use when you want the code to run at least once, regardless of the condition.

JavaScript Break and Continue

In JavaScript, **break** and **continue** are statements used to control how loops work. They let you stop a loop early or skip parts of it, which can be very useful when you don't need to process every item in a loop.

1. break Statement

Purpose: The **`break`** statement is used to **stop a loop immediately**. When JavaScript sees `break`, it exits the loop, no matter how many times it was supposed to run.

Where It's Used: It can be used in both loops and `switch` statements.

Example of `break`:

```javascript
let content = "";
for (let i = 1; i < 1000; i++) {
    if (i === 6) {
        break; // Stop the loop when i is 6
    }
    content += "Aimerz" + i + "\n";
}
console.log(content);
```

Output:

```
Aimerz1
Aimerz2
Aimerz3
Aimerz4
Aimerz5
```

Explanation: The loop stops when `i` is 6, so "Aimerz6" and further numbers do not get added to `content`.

2. continue Statement

Purpose: The **continue** statement is used to **skip the rest of the code in the loop for that one round** and go directly to the next round.

Where It's Used: It can only be used in loops.

Example of continue

```javascript
JavaScript
let content = "";
for (let i = 1; i < 7; i++) {
    if (i === 4) {
        continue; // Skip this round when i is 4
    }
    content += "Aimerz" + i + "\n";
}
console.log(content);
```

Output:

```
Aimerz1
Aimerz2
Aimerz3
Aimerz5
Aimerz6
```

Explanation: The loop skips adding "Aimerz4" to `content` but continues adding the other numbers.

Switch Statement

A **switch statement** is a way to choose one option from many based on the value of a variable. Instead of using many if statements, you can use a switch to check one variable against multiple cases. This makes the code cleaner and easier to read.

The **switch statement** is a control structure that evaluates an expression and executes the code block associated with the first matching case. If no cases match, it can execute a default block if provided.

Syntax

```javascript
switch (expression) {
    case value1:
        // Code to execute if expression === value1
        break; // Optional
    case value2:
        // Code to execute if expression === value2
        break; // Optional
    // You can add more cases here
    default:
        // Code to execute if no case matches
}
```

Using Switch Statement with Break

With Break: When you use break, it stops the execution of the switch block after the matching case. If you don't use break, the code will continue executing into the next case (known as "fall-through").

Example:

```javascript
let fruit = "apple";

switch (fruit) {
    case "banana":
        console.log("Banana is yellow.");
        break; // Stops here
    case "apple":
        console.log("Apple is red.");
        break; // Stops here
    case "orange":
        console.log("Orange is orange.");
        break; // Stops here
    default:
        console.log("Unknown fruit.");
}
```

Explanation: In this example, since fruit is "apple", it prints "Apple is red." and stops due to the break.

Using Switch Statement without Break

Without Break: If you don't include a break, the code will "fall through" and execute the following cases until it hits a break or the end of the switch.

Example:

```javascript
let fruit = "banana";

switch (fruit) {
    case "banana":
        console.log("Banana is yellow.");
    case "apple":
        console.log("Apple is red.");
        break; // Stops here
    default:
        console.log("Unknown fruit.");
}
```

Explanation: In this example, since fruit is "banana", it prints "Banana is yellow." and then continues to print "Apple is red." because there is no break after the banana case.

Using Switch Statement with Return

- **With Return**: You can use a switch statement inside a function and use `return` to send a value back to the caller.

Example:

```javascript
JavaScript
function getFruitColor(fruit) {
    switch (fruit) {
        case "banana":
            return "Yellow";
        case "apple":
            return "Red";
        case "orange":
            return "Orange";
        default:
            return "Unknown fruit color";
    }
}

let color = getFruitColor("apple");
console.log(color); // Outputs: Red
```

Explanation: In this function, when you call getFruitColor("apple"), it returns "Red" because of the matching case. The return statement exits the function immediately.

Data Types in Case Statements

The expression in the switch statement is compared with the values in the case statements using strict equality (===). This means that both the value and the type must match. For example, the number 5 is not equal to the string "5".

Example:

```javascript
JavaScript
let num = 5;

switch (num) {
    case "5": // This case will not match
        console.log("Matched string 5");
        break;
    case 5: // This case will match
        console.log("Matched number 5");
        break;
    default:
        console.log("No match");
}
```

Fall-Through Behavior

If you omit the break statement, the code will continue executing the subsequent cases until it hits a break or the end of the switch. This behavior can be useful when you want multiple cases to execute the same code.

Default Case

The default case is optional but serves as a fallback if none of the specified cases match. It's a good practice to include a default case to handle unexpected values.

Example:

```javascript
let day = 8;

switch (day) {
    case 1:
        console.log("Monday");
        break;
    case 2:
        console.log("Tuesday");
        break;
    default:
        console.log("Not a valid day"); // This will execute
}
```

Nested Switch Statements

We can nest switch statements within each other, though this can make the code harder to read.

Example:

JavaScript
```javascript
let fruit = "apple";
let color = "red";

switch (fruit) {
    case "banana":
        console.log("Banana is yellow.");
        break;
    case "apple":
        switch (color) {
            case "red":
                console.log("Apple is red.");
                break;
            case "green":
                console.log("Apple is green.");
                break;
            default:
                console.log("Apple has another color.");
        }
        break;
    default:
        console.log("Unknown fruit.");
}
```

THANK YOU

AIMERZ.ai
Aim.Act.Achieve

Stay updated with us!

Vishwa Mohan

Vishwa Mohan

vishwa.mohan.singh

Vishwa Mohan