**CONTROL STRUCTURES:**

Control structures in JavaScript are fundamental building blocks that allow you to control the flow of your program's execution based on conditions or loops. Here's a breakdown of some common control structures with examples:

**1. Conditional Statements:**

These statements allow your code to make decisions and execute different blocks of code based on certain conditions.

* **if statement:** Executes code only if a specified condition is true.

if (temperature > 30) {

console.log ("It's hot outside!");

}

let age = 20;

if (age >= 18) {

console.log ("You are eligible to vote.");

} else {

console.log ("You are not eligible to vote.");

}

* **If-Else Statement**: Executes a block of code if a condition is true; otherwise, it executes another block of code.

if (temperature > 30) {

console.log ("It's hot outside!");

} else {

console.log ("It's not so hot outside.");

}

* **if-else if statement:** Allows checking multiple conditions and executing the corresponding block if a condition is true.

let grade = 85;

if (grade >= 90) {

console.log("Excellent!");

} else if (grade >= 80) {

console.log ("Very good!");

} else {

console.log ("Good effort, keep practicing!");

}

**Ternary Operator**

* A shorthand for the if-else statement.

let result = (age >= 18) ? "Adult" : "Minor";

console.log(result);

**switch statement:**

* Provides a multi-way branching mechanism based on the value of an expression.
* Selects one of many code blocks to be executed.

let day = "Tuesday";

switch (day) {

case "Monday":

console.log("Start of the week!");

break;

case "Tuesday":

case "Wednesday":

case "Thursday":

console.log("Midweek!");

break;

case "Friday":

console.log("TGIF!");

break;

default:

console.log("Weekend!");

}

**2. Loops:**

These statements allow you to repeat a block of code multiple times, either for a predetermined number of iterations or until a certain condition is met.

* **for loop:** Executes a code block repeatedly for a specific number of times, defined by a start, stop, and increment/decrement step.

for (let i = 1; i <= 5; i++) {

console.log("Iteration", i);

}

* **while loop:** Executes a code block repeatedly as long as a specified condition is true.

let count = 0;

while (count < 3) {

console.log("Count:", count);

count++;

}

* **do-while loop:** Similar to while, but the code block is guaranteed to execute at least once, even if the condition is initially false.
* Executes a block of code once, and then repeats the execution as long as a specified condition is true.

let i = 0;

do {

console.log("Number " + i);

i++;

} while (i < 5);

let message = "";

do {

message = prompt("Enter a message (or 'quit' to exit): ");

console.log("You entered:", message);

} while (message !== "quit");

**3. Break and Continue Statements:**

These statements are used to control the flow within loops.

* **break:** Exits a loop completely, even if the loop condition is still true.
* **continue:** Skips the current iteration of the loop and moves on to the next.

**Break Statement**

The break statement is used to exit a loop prematurely. It immediately terminates the loop and transfers control to the statement following the loop.

for (let i = 0; i < 10; i++) {

if (i === 3) {

break; // Exits the loop when i is 3

}

console.log ("The number is " + i);

}

// Output: The number is 0

// The number is 1

// The number is 2

**Continue Statement**

The continue statement is used to skip the current iteration of the loop and continue with the next iteration.

for (let i = 0; i < 10; i++) {

if (i === 3) {

continue; // Skips the current iteration when i is 3

}

console.log ("The number is " + i);

}

// Output: The number is 0

// The number is 1

// The number is 2

// The number is 4

// The number is 5

// ... up to 9

In the break example, the loop stops running entirely when i is 3. In the continue example, the loop skips the iteration where i is 3 but continues with the subsequent iterations. These statements are particularly useful for managing complex conditions within loops.

control structures

control structures are used to control the flow of execution in a program. Here are the main types of control structures

Control structures are fundamental elements in languages that enable the execution of specific code blocks based on certain conditions or criteria. They allow programmers to dictate the flow of execution within a program. Control structures can broadly be categorized into three types:

1. Sequential: Sequential control structures execute instructions in a sequence, one after the other, without any branching or looping. This is the default behavior of most programming languages unless explicitly modified.

2. Selection: Selection control structures, also known as conditional statements, execute specific blocks of code based on certain conditions. The most common types of selection structures are:

   - If-else statements: These allow the program to make decisions based on whether a given condition is true or false. If the condition is true, one block of code is executed; otherwise, another block is executed.

   - Switch statements (or case statements): These provide a way to execute different blocks of code based on the value of a variable or expression.

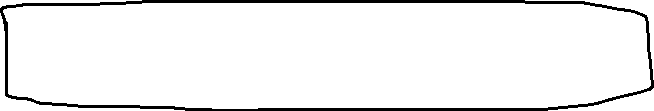
3. Iteration (or Repetition): Iteration control structures, also known as loops, allow a certain block of code to be executed repeatedly until a specific condition is met. The most common types of iteration structures are:

   - For loops: These execute a block of code a fixed number of times, iterating over a sequence of values (e.g., numbers from 1 to 10).

   - While loops: These execute a block of code as long as a specified condition is true. The condition is evaluated before each iteration, and if it's true, the loop continues; otherwise, it terminates.

   - Do-while loops: These are similar to while loops but with one crucial difference: the condition is evaluated after the execution of the loop's block of code. This ensures that the block of code is executed at least once, even if the condition is initially false.

Control structures are essential for creating programs that can perform complex tasks, respond dynamically to user input, and handle various scenarios efficiently. They provide the flexibility and logic necessary for writing functional and adaptable code.

a block is a set of statements enclosed within curly braces **{}**. Blocks are used to group zero or more statements together, forming a single compound statement. Blocks are commonly used in various contexts, including function bodies, loop bodies, conditional statements, and more

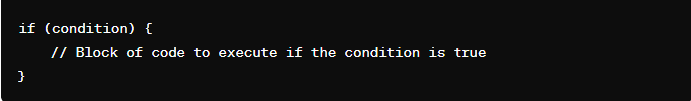
Selection control structures/ conditional statements:

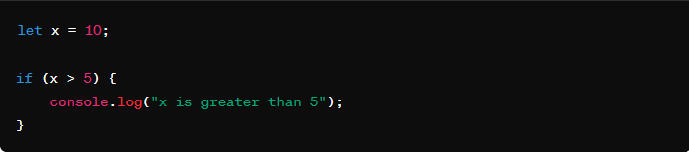
Conditional statements, also known as control structures, are fundamental constructs in programming languages that allow the execution of different blocks of code based on certain conditions or criteria. They enable the program to make decisions and perform different actions depending on the values of variables, user input, or the result of expressions.

Conditional statements provide the ability to create dynamic and flexible programs that can adapt their behavior at runtime. They are essential for implementing logic and decision-making processes within software applications.

There are several types of conditional statements commonly used in programming languages, including:

1. if statement: Executes a block of code if a specified condition is true.

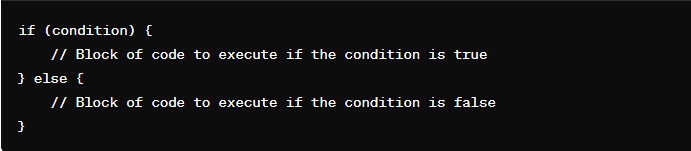




In this example, the condition x > 5 evaluates to true because the value of x is 10, which is indeed greater than 5. Therefore, the block of code inside the if statement (console.log) is executed, and it will output "x is greater than 5" to the console.

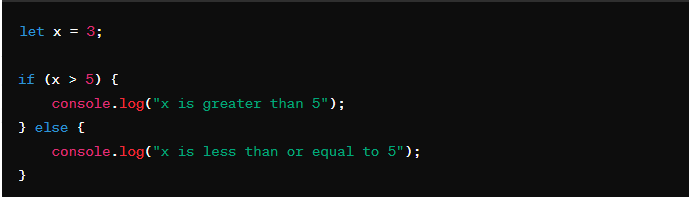
1. else statement: Executes a block of code if the condition of the preceding if statement is false.

else statement: The else statement is used to execute a block of code if the condition of the preceding if statement (or else if statement) is false. It is optional and provides an alternative block of code to execute when the condition of the preceding if statement evaluates to false



If the condition in the if statement evaluates to true, the block of code inside the first set of curly braces {} is executed, and the block of code inside the else statement is skipped.

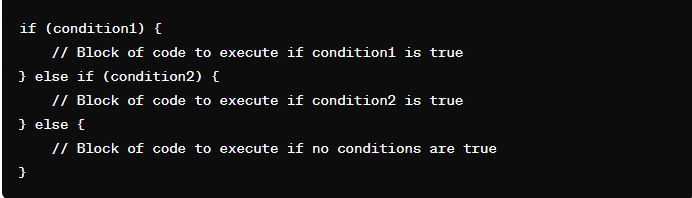
If the condition in the if statement evaluates to false, the block of code inside the else statement is executed, and the block of code inside the first set of curly braces {} is skipped.

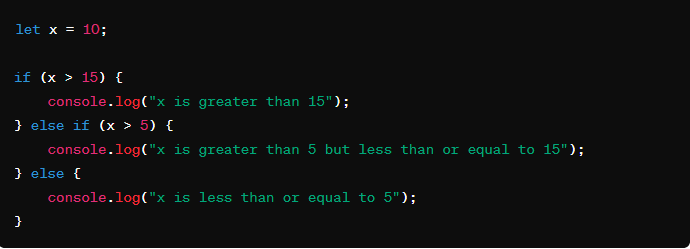


In this example, the condition x > 5 evaluates to false because the value of x is 3, which is less than 5. Therefore, the block of code inside the else statement is executed, and it will output "x is less than or equal to 5" to the console

1. else if statement: Provides an alternative condition to test if the preceding if condition is false. It can be used multiple times after an initial if statement.

else if statement: The else if statement provides an alternative condition to test if the preceding if condition is false. It allows you to check for additional conditions sequentially after the initial if statement. Each else if statement is evaluated only if the preceding if statement (or else if statement) evaluated to false.

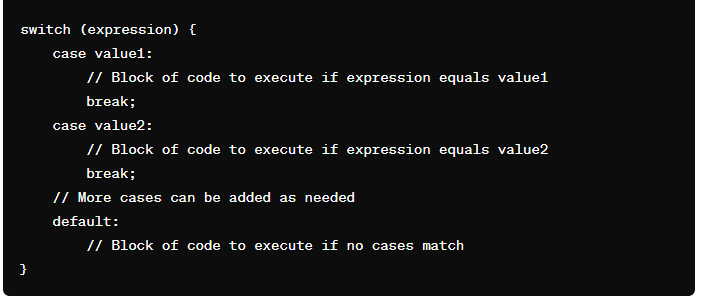




In this example, the value of x is 10. The first condition (x > 15) evaluates to false, so the program moves to the next else if statement. The second condition (x > 5) evaluates to true, so the corresponding block of code is executed, and it will output "x is greater than 5 but less than or equal to 15" to the console.

1. switch statement: Evaluates an expression and executes different blocks of code based on matching cases. It provides an alternative to multiple else if statements.

switch statement: The switch statement evaluates an expression and executes different blocks of code based on matching cases. It provides an efficient way to handle multiple possible outcomes of a single expression.



expression is the expression or variable whose value is to be compared with the cases.

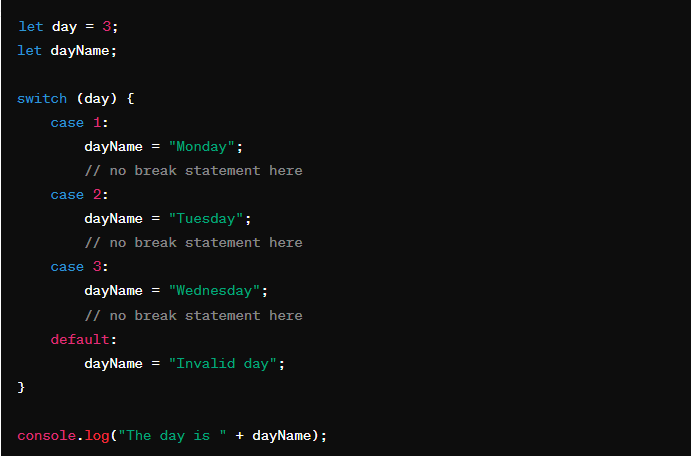
case value1, case value2, etc., represent the different possible values of the expression.

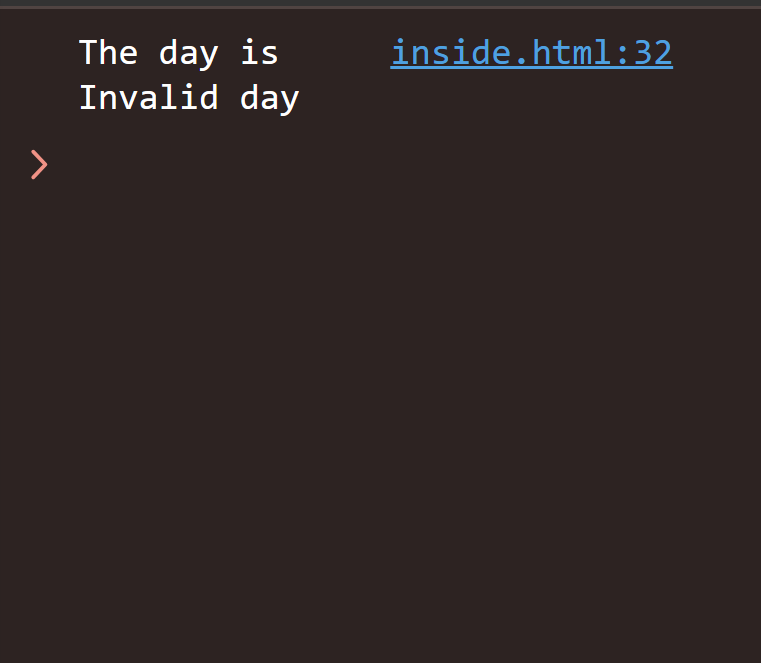
When the switch statement is executed, the expression is evaluated, and control jumps to the case that matches the value of the expression.

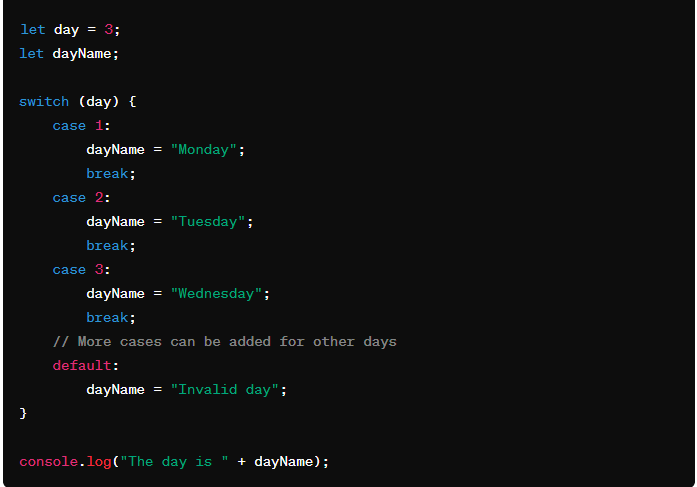
If no matching case is found, the default block (if provided) is executed. The default block is optional.

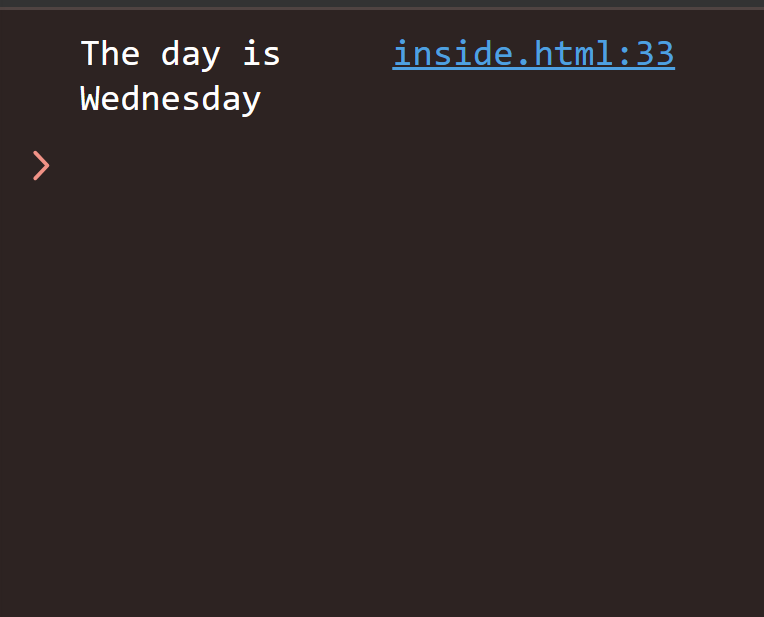
When break is encountered inside a switch statement, it exits the switch block.

Without the break statement, control would fall through to the next case, executing the code for that case regardless of whether its condition is met.









In this example, the value of day is 3. The switch statement evaluates the value of day and executes the corresponding block of code based on the matching case. Since day is 3, it matches the case 3, and the dayName is set to "Wednesday". Finally, it logs "The day is Wednesday" to the console.

5. Ternary operator: A concise way to write conditional statements. It takes three operands and evaluates a Boolean expression, returning one of two expressions depending on whether the expression is true or false.

These conditional statements form the backbone of decision-making in programming, allowing developers to create programs that respond dynamically to various scenarios and conditions. They are essential for creating robust and versatile software applications.

loops

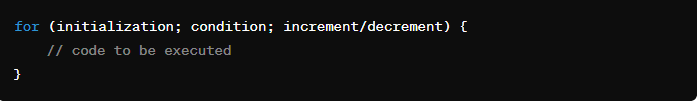
In JavaScript, loops are control structures that allow you to execute a block of code repeatedly as long as a specified condition is true. There are several types of loops available in JavaScript:

In JavaScript, loops are control structures that allow you to execute a block of code repeatedly as long as a specified condition is true. There are several types of loops available in JavaScript:

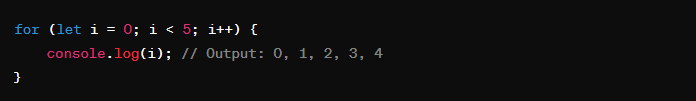
1. for Loop:

   - The for loop repeats a block of code a specified number of times.

   - Syntax:



   - Example:



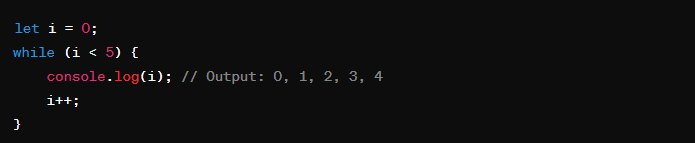
2. while Loop:

   - The while loop repeats a block of code as long as a specified condition is true.

   - Syntax:



   - Example:



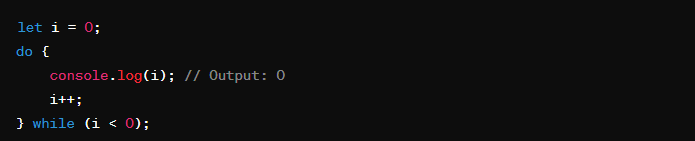
3. do...while Loop:

   - The do...while loop is similar to the while loop, but it always executes the block of code at least once before checking the condition.

   - Syntax:



   - Example:



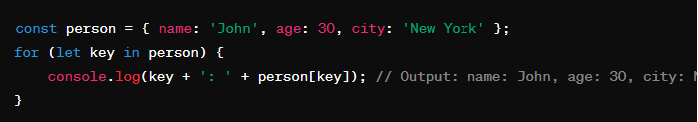
4. for...in Loop:

   - The for...in loop iterates over the enumerable properties of an object.

   - Syntax:



       - Example:



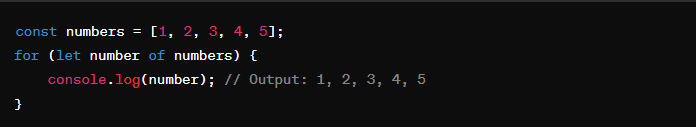
5. for...of Loop:

   - The for...of loop iterates over iterable objects such as arrays, strings, maps, sets, etc.

   - Syntax:



   - Example:



These are the main types of loops available in JavaScript, each serving different purposes depending on the requirements of your code.