Control Structures In javascript

Control structures are essential components of programming languages that direct the execution flow of a program. In JavaScript, control structures enable developers to make decisions, iterate through data, handle errors, and create complex algorithms.

Control Statement

JavaScript statements can be used to control the execution of programs based on set conditions. If the condition meets a specific block of action will be executed otherwise another block of action that satisfies that condition.

Types of control statements:

Condition statements are statements used for decision making, a decision is made by a conditional statement based on an expression that is passed. Either Yes or No.
Iterative Statement: This is a statement that iterates repeatedly until a certain condition is met. In other words, if we have an expression, the statement wil keep repeating itself until it's satisfied.

There are several methods that can be used to perform control statement in javascripts;
If Statements:
In this approach, we are using an if statement to check a specific condition, the code block gets executed when the given condition is satisfied.
Syntax:

```
 If (condition is given here ){
                If condition is met,
             the code will be executed.
}
```

Example:

```
Const  age = 16;
If (age < 18){
        console.log(“Sorry!! You must be 18+ to qualify.”);
};
```

Output

```
Sorry!! You must be 18+ to qualify.
```

If-Else statement:

The if-else statement will perform some action for a specific condition. If the condition is met, then a particular code of action will be executed, otherwise it will execute another code of action that satisfies that particular condition.

Syntax:

```
If (condition1) {
        Execute when condition 1 is true
}
else (condition 2) {
        Execute if condition 2 is true
 }
```

Example:

```
Const  age = 16;
if(age > 18){
        console.log(“Proceed”);
Else{
        console.log(“minor”) ;
};
```

Output:

```
Minor
```

Nested if-else Statement:

As we know by this stage, the body of an if statement and an else statement can be any valid statement.

Let's suppose that we stick to our idea of using a block statement to encompass multiple sub-statements. In that case, we can put any number of statements inside the block, including more *ifs* and *elses*.
Nesting conditional statements, i.e. putting conditional statements inside conditional statements, is very common not just in JavaScript but in almost all high-level programming languages.

Switch Statement:
The switch case statement in JavaScript is also used for decision-making purposes. In some cases, using the switch case statement is seen to be more convenient than if-else statements.
We start with the switch keyword, followed by a pair of parentheses (()), followed by the case block of the switch statement.

The **case** block defines various cases to match expression against and even the code to execute, i.e. statements, when either of them gives a match.

Each **case** begins with the case keyword, followed by a value to match against expression, followed by a colon (:), followed by the case's corresponding set of statements. If this value matches expression, the corresponding block of code following the colon (:) is executed.

The break keyword, at the end of each cases's corresponding block of code, serves to break execution out of the switch statement once that block of code gets executed. It's redundant to give it in the last case since after executing the last case, execution moves out of the switch statement anyways.

Syntax:
```
switch (expression) {
  case value1:
     statement1;
     break;
  case value2:
     statement2;
```

```
        break;
    .
    .
    case valueN:
        statementN;
        break;
    default:
        statementDefault;
}
```

Example:

```
let num = 5;

switch (num) {
    case 0:
        console.log("Number is zero.");
        break;
    case 1:
        console.log("Nuber is one.");
        break;
    case 2:
        console.log("Number is two.");
        break;
    default:
        console.log("Number is greater than 2.");
};
```

Output:

```
Number is greater than 2.
```

Using Ternary operator

The conditional operator, also referred to as the ternary operator (?:), is a shortcut for expressing conditional statements in JavaScript.

Syntax:

Condition ? Value if true : if false

In this example, we are using the ternary operator to find whether the given number is positive or negative.

```
let num = 10;

let result = num >= 0 ? "Positive" : "Negative";

console.log(`The number is ${result}.`);
```

Output:
Positive

For Loops:
In this approach, we are using for loop in which the execution of a set of instructions repeatedly until some condition evaluates and becomes false.
Syntax:

```
for (statement 1; statement 2; statement 3) {
  // Code here . . .
}
```

Example:

```
for (let i = 0; i <= 10; i++) {
  if (i % 2 === 0) {
    console.log(i);
  }
};
```

Output:

```
0
2
4
6
8
10
```