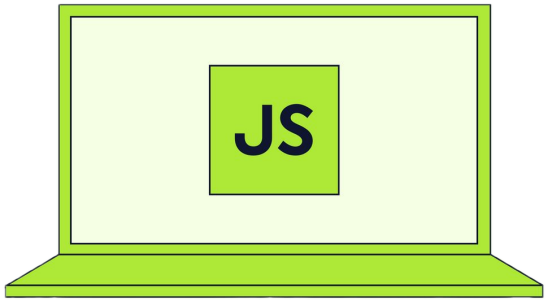
Newton School of Technology

The Complete Javascript Course



@newtonschool

Lecture 3: conditionals and loops

-Vishal Sharma

JS

Newton School of Technology

Table of Contents

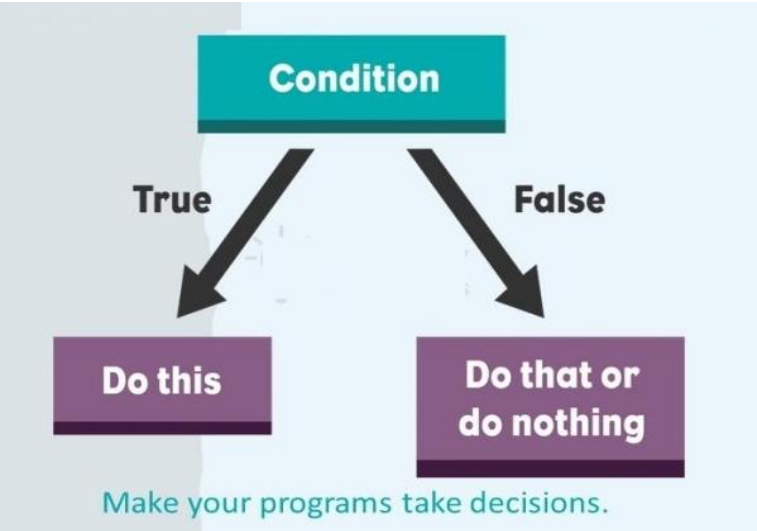
Newton School of Technology

Conditional Statements in Javascript

Newton School of Technology

What are conditional statements?

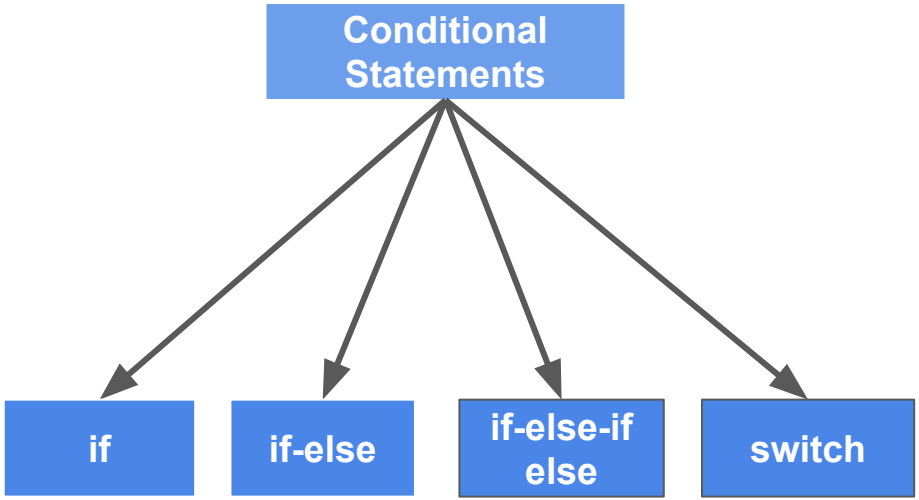
Conditional statements help determine the flow of execution in a program based on specific conditions.



Newton School of Technology

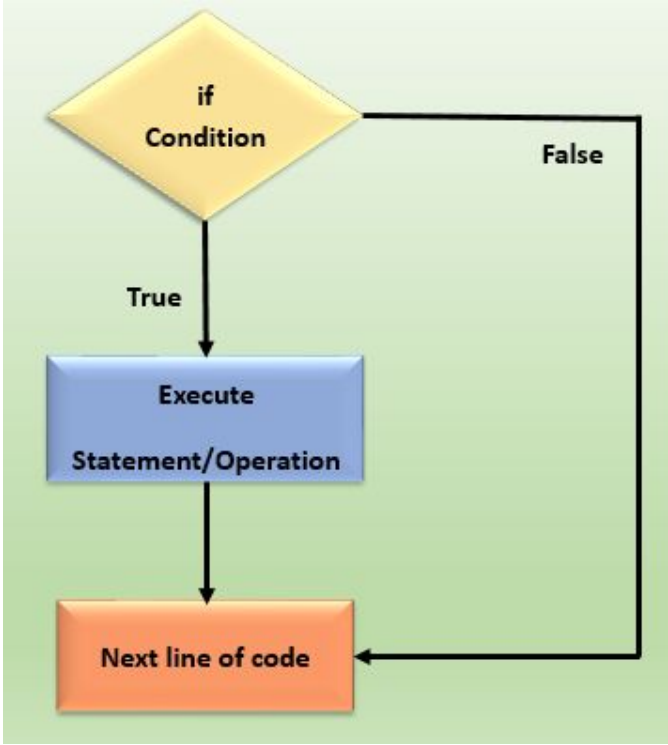
Types of conditional statements

Here are the main types of conditional statements:-



if statement

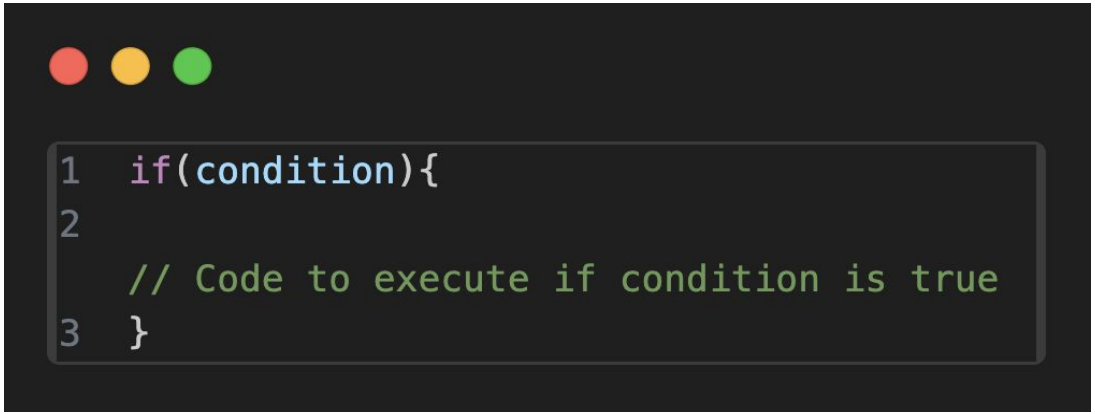
if statement is the most basic form of conditional statement. Body of *if* statement runs if test condition is true.



Newton School of Technology

if statement: Syntax

Let’s have a look at if statement syntax.



```
if(condition){

// Code to execute if condition is true
}
```

if statement: Example

Let’s understand with an example:-

```
1  // Declaring and assigning the age
2  let age = 19;
3
4  // Checking the condition
5  if (age > 18) {
6      // Executing the code
7      console.log("I am an adult.");
8  }
```

Newton School of Technology

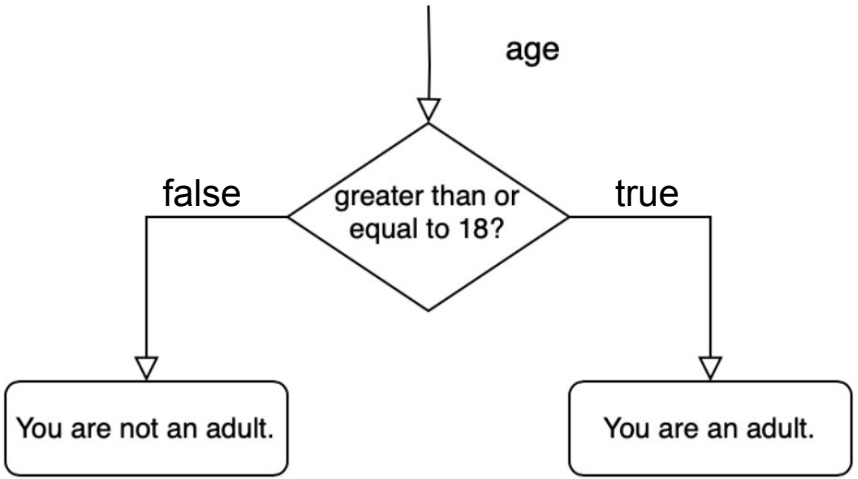
if-else: Building on if statement

If you understand the if statement, adding the else part is straightforward. It simply provides an alternative action when the condition is false.

Newton School of Technology

if-else: Building on if statement

In the previous example, we checked if a person is under 18 using only an if statement." Modify the code to include an else condition that prints "You are not eligible." when the person age is less than 18 years.



Newton School of Technology

if-else: Example

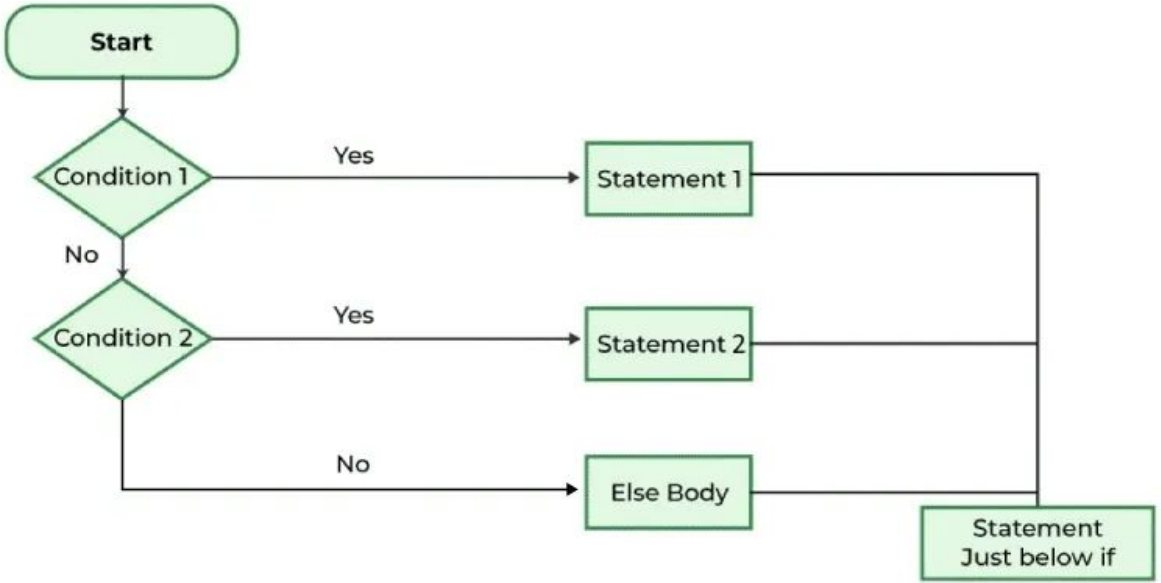
Here age is less than 18 so else statement is executed:-

```
let age = 17;

if (age > 18) {
    console.log("I am an adult.");
} else {
    console.log("I am not an adult.");
}

```

Newton School of Technology

Going beyond: if-else-if-else ladder

The if-else-if-else ladder helps the program check multiple conditions one by one and take an action based on the first condition that’s true.



Going beyond: if-else-if-else ladder

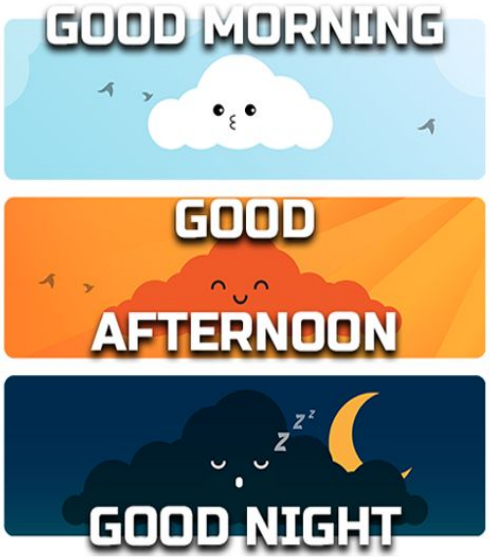
Think of it like making plans: Plan A, then Plan B if A fails, then Plan C, and only facing the worst if all fail. The if-else-if-else ladder works the same way.

Newton School of Technology

If-else-if-else: example

Let’s write code for delivering salutations at different parts of day:-

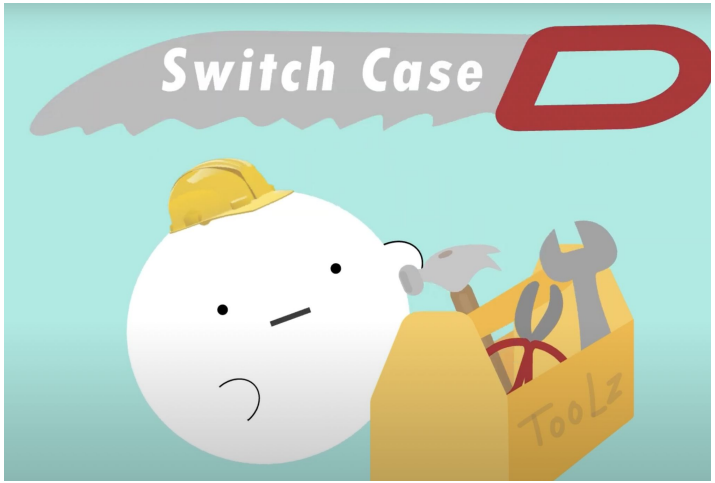
```
let hour = 15;
// Time in 24-hour format

if (hour >= 5 && hour < 12) {
    console.log("Good Morning!");
} else if (hour >= 12 && hour < 18) {
    console.log("Good Afternoon!");
} else {
    console.log("Good Evening!");
}
```



Newton School of Technology

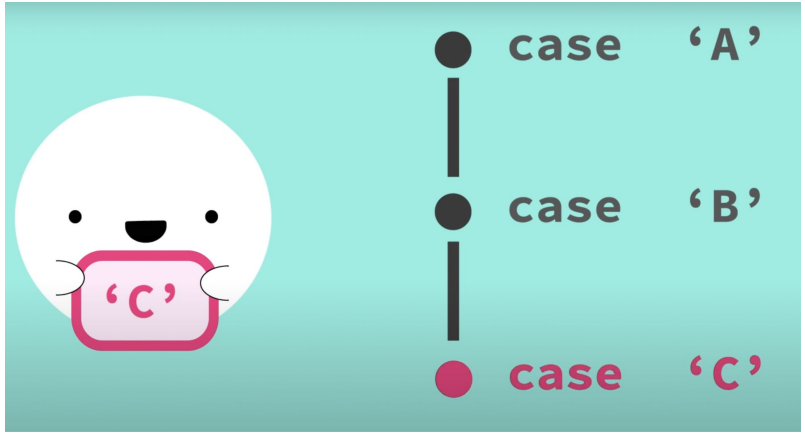
switch statement

switch statement is one among many tools for condition based execution of statements.



switch statement

It selects one among many cases whichever matches with value passed inside switch.



Newton School of Technology

switch statement: syntax

Let’s have a look at its syntax:-

```
switch(variable) {
  case 1:
    // execute you code
  break;
  case n:
    // execute you code
  break;
  default:
    // execute you code
  break;
}
```

We pass a variable inside switch and whichever case matches that variable value gets executed.

Newton School of Technology

switch statement: example

Here switch checks fruit against each case. If no match is found, the default block runs as a fallback.

```
let fruit = "apple";

switch (fruit) {
    case "apple":
        console.log("It's an apple!");
        break;
    case "banana":
        console.log("It's a banana!");
        break;
    default:
        console.log("Unknown fruit!");
}
```

Since case ‘apple’ matches the fruit variable, “This is an apple.” gets printed.

But what is the break statement just after every test case end??

Newton School of Technology

switch statement: break

In this example we have added break after every test case. Do you know why?

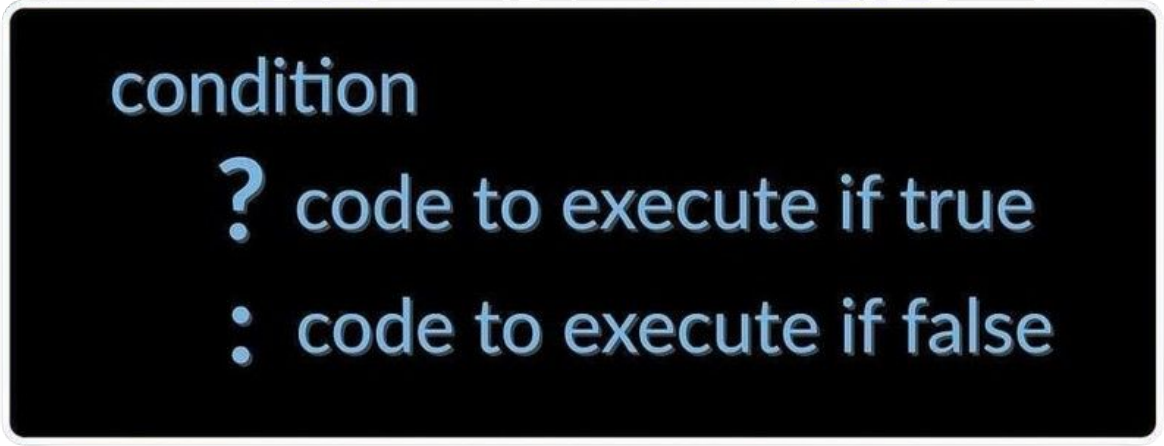
```
let fruit = "apple";

switch (fruit) {
    case "apple":
        console.log("It's an apple!");
        break;
    case "banana":
        console.log("It's a banana!");
        break;
    default:
        console.log("Unknown fruit!");
}
```

Whenever a case matches statements under it run and to avoid other cases to run we break its execution by just adding break at the end of it.

Newton School of Technology

Ternary Operator: A Shortcut for If-Else

The ternary operator is a concise way to write an if-else statement in a single line. It evaluates a condition and returns one value if true, and another if false.



Newton School of Technology

Ternary Operator: example

The ternary operator simplifies if-else code when only one condition is involved.

```
let age = 18;
let category = (age >= 18) ? "Adult" : "Minor";
console.log(category); // Output: Adult
```

Newton School of Technology

Ternary Operator: Comparison with if..else

Compared to if-else, the ternary operator offers smoother flow, improving readability and understanding.

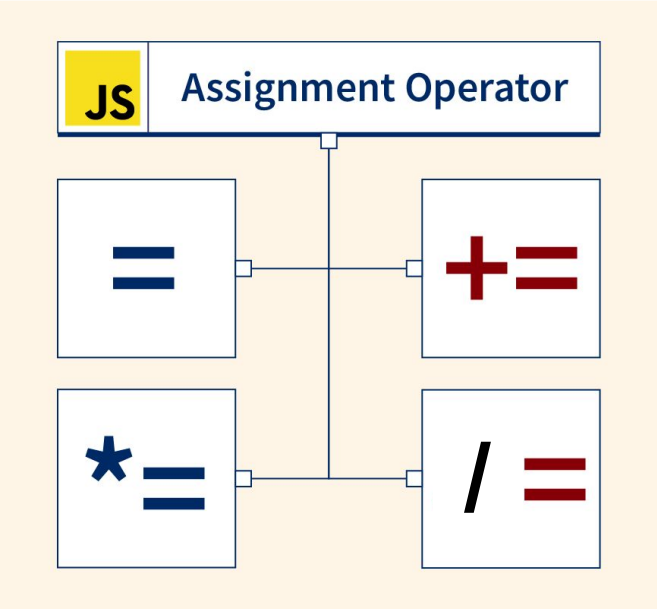
```
let age = 18;
let category;

if (age >= 18) {
    category = "Adult";
} else {
    category = "Minor";
}

console.log(category);
// Output: Adult
```

Now, after comparison you will appreciate ternary operator.

Assignment Operators

Newton School of Technology

Assignment operators are used to assign values to variables. The basic assignment operator is =, but there are also shorthand operators like +=, -=, *= etc.



Usually we use shorthand notations in writing javascript, it saves time and energy

Assignment Operators: Example

Newton School of Technology

Let’s have a look at an example:-

```
// Basic Assignment
let x = 10;
console.log("Basic Assignment: x =", x); // Output: 10

// Addition Assignment (x += y)
x += 5;  // Equivalent to x = x + 5
console.log("Addition Assignment: x += 5 =", x);
// Output: 15

// Subtraction Assignment (x -= y)
x -= 3;  // Equivalent to x = x - 3
console.log("Subtraction Assignment: x -= 3 =", x);
// Output: 12

// Multiplication Assignment (x *= y)
x *= 2;  // Equivalent to x = x * 2
console.log("Multiplication Assignment: x *= 2 =", x);
// Output: 24
```

First one is the basic assignment and rest of it are shorthands.

We need not to write shorthands necessarily, but it saves time

Newton School of Technology

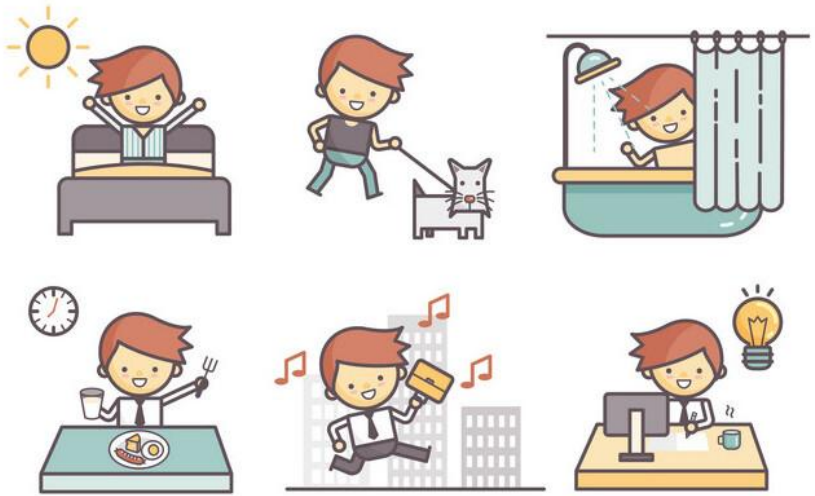
Loops in real life

Every day we follow certain routine which we repeat all over the week, like brushing your teeth, taking shower etc.

Newton School of Technology

Loops in real life

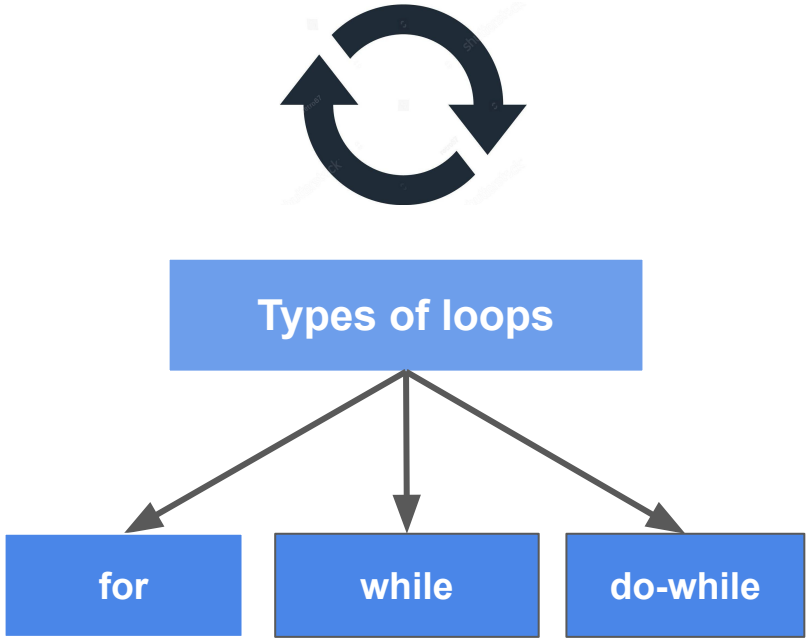
We wake up, go for a walk, take a shower, then take lunch and so on. And next day we repeat the same process.



Newton School of Technology

Loops in Programming

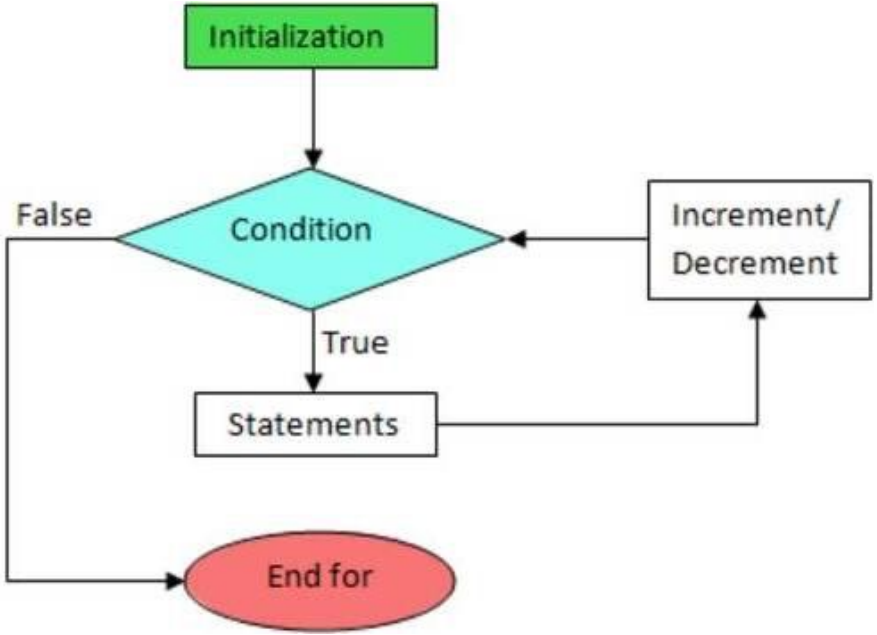
Just like we repeat several tasks in daily like, we need some tasks to be repeated in programming and we repeat those tasks with the help of loops.

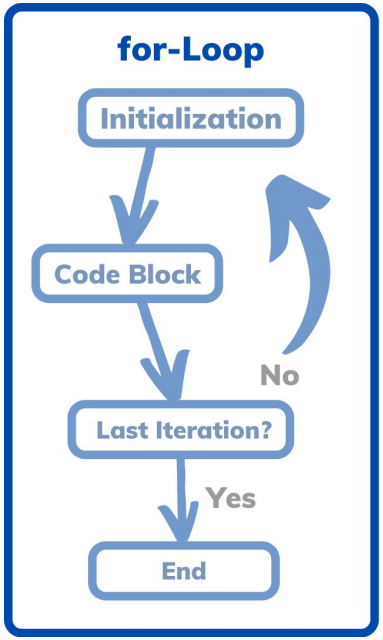


for loop

Newton School of Technology

A for loop in JavaScript repeats a block of code a set number of times, typically when the number of iterations is known in advance.

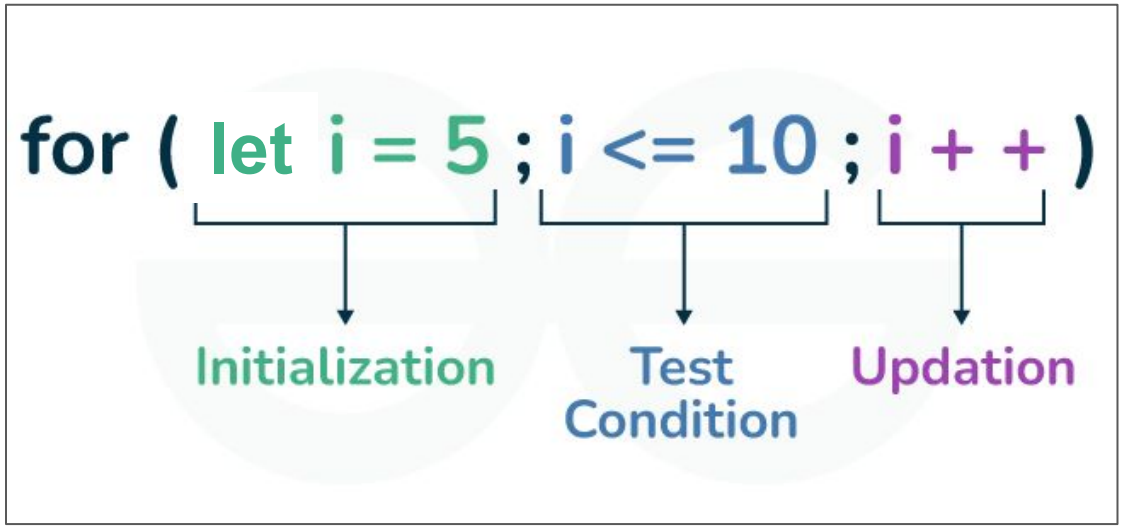




for loop: syntax

Newton School of Technology

for loop has three parts, initial state, increment/decrement and end condition.



for loop: example

Newton School of Technology

Let’s see an example:-

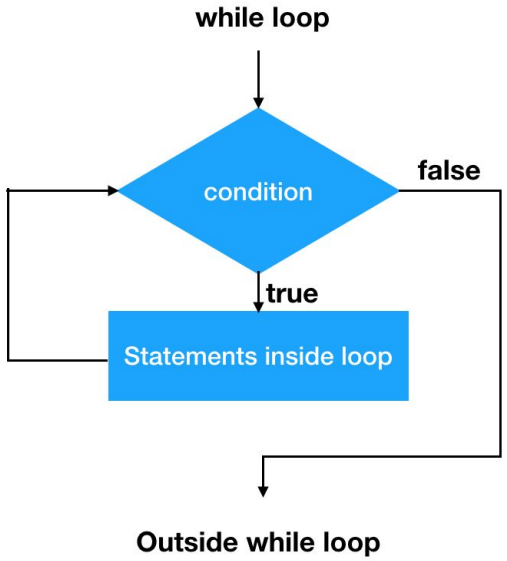
```
1  for (let i = 1; i <= 5; i++) {
2      console.log("Iteration Number: ", i);
3  }
4  // Output
5  // Iteration Number: 1
6  // Iteration Number: 2
7  // ...
```

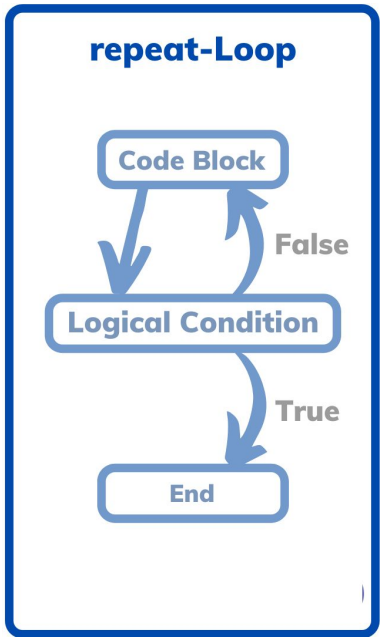
Here console.log statement is printed 5 times as loop runs for exact 5 times.

while loop

Newton School of Technology

A while loop in JavaScript repeatedly executes a block of code as long as the condition remains true, making it ideal when the number of iterations depends on dynamic conditions.

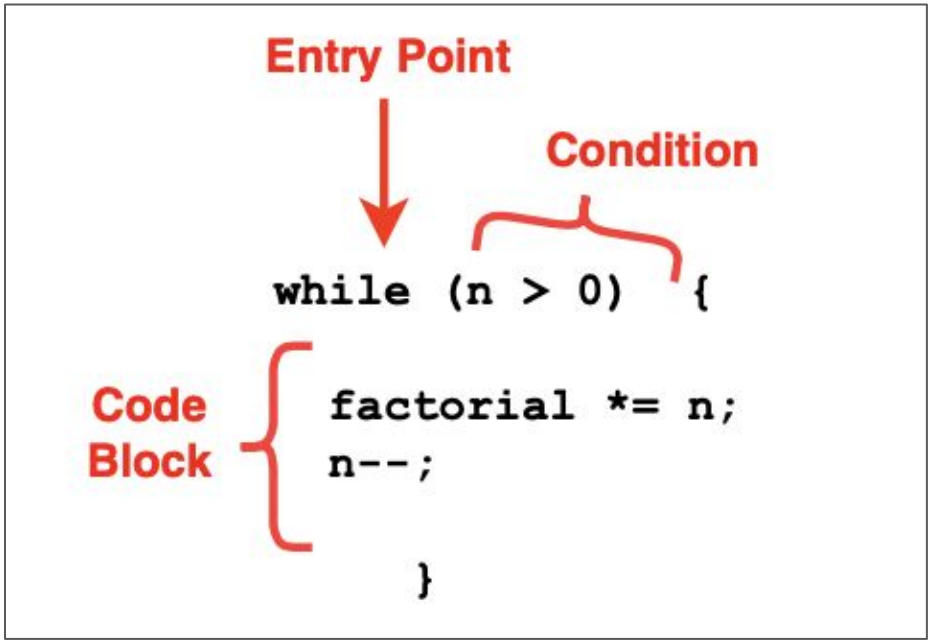




while loop: syntax

Newton School of Technology

for loop has just end condition, increment/decrement is done in body and initialization is done before the loop starts.



while loop: example

Newton School of Technology

Let’s see an example:-

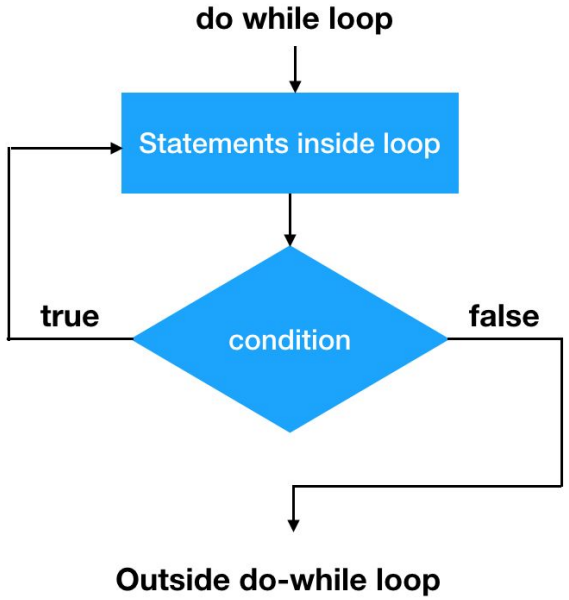
```
let i = 1;
while (i <= 5) {
    console.log("Iteration Number: ", i);
    i++;
}
// Output
// Iteration Number: 1
// Iteration Number: 2
// ...
```

Here console.log statement is printed 5 times as loop runs for exact 5 times. But we did increment/decrement inside body and initialization before the loop

do-while loop

Newton School of Technology

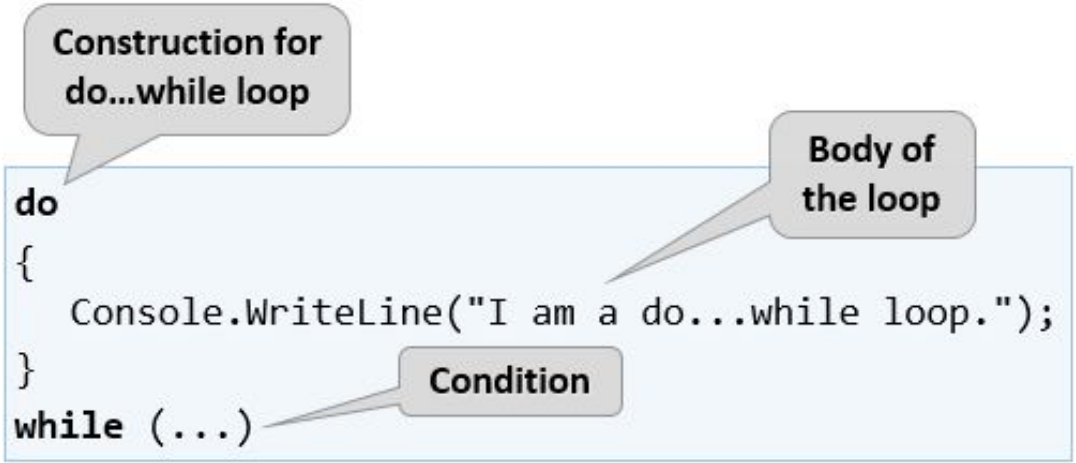
A do-while loop is a control flow statement that ensures a block of code runs at least once, regardless of the condition.



It runs at least once because at start body is run before actually checking the test condition but only for the first time.

do-while loop: syntax

Let’s have a look at it’s syntax:-



It is same as while loop with only difference that condition is checked at the end.

do-while loop: example

Newton School of Technology

Let’s see an example:-

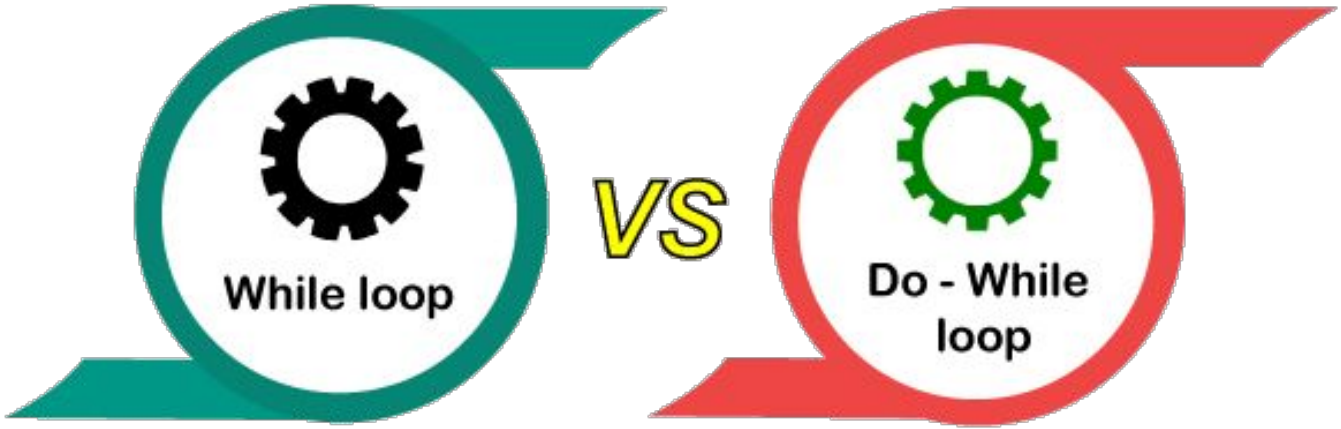
```
let i = 1;

do {
    console.log("this is iteration number: ", i);
    i++; // Increment to move to next iteration
} while (i <=5 );

// Output:
// This is iteration number: 1
// Tjhis iteration number: 2
// ........................
```

Here condition got checked at the end of the loop.

Difference: while and do-while loop

Newton School of Technology

The while loop checks the condition before execution, potentially skipping the loop entirely if false. The do-while loop runs at least once before checking the condition.



Difference: while loop

Newton School of Technology

Here while loop will not run at all since test condition fails before first iteration.

```
let i = 6;

while (i <= 5) {
    console.log("This will not run.");
    i++;
}

// Output
// Nothing would appear as while loop didn't run
```

Difference: do-while loop

Newton School of Technology

Here do-while loop will run since body gets executed before it can run test condition for the first time.

```
1  do {
2      console.log("This will run once.");
3      i++;
4  } while (i <= 5);
5
6  // Output
7  // This will run once
```

Newton School of Technology

Control Statements: break and continue

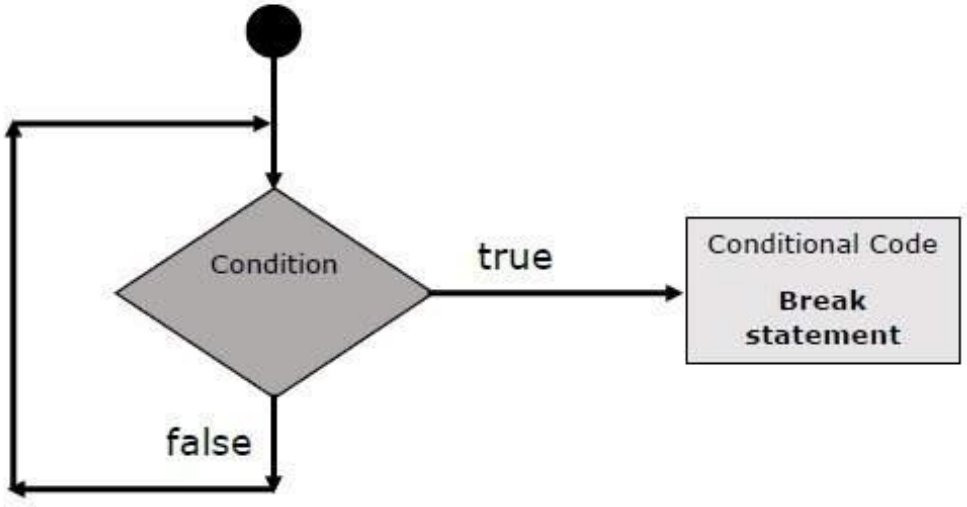
Control statement: break

The break statement stops a loop or switch case from running further, allowing you to exit early when a condition is met.

Newton School of Technology

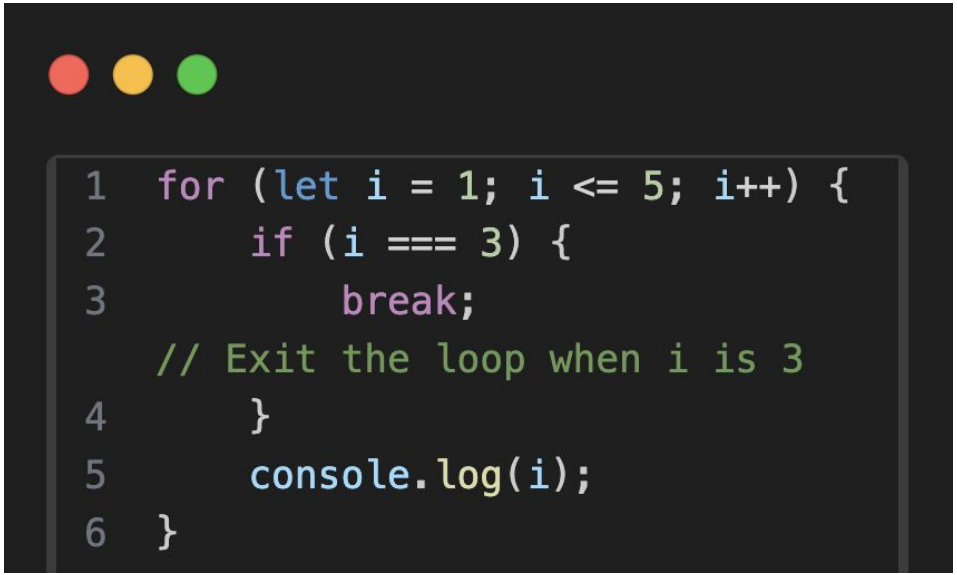
Control statement: break

When the specified condition is met, the break statement immediately terminates the loop and transfers control to the next statement after the loop.



Control statement: break (with)

Let’s understand with example:-



```
for (let i = 1; i <= 5; i++) {
    if (i === 3) {
        break;
// Exit the loop when i is 3
    }
    console.log(i);
}
```

Here we broke out of loop when condition i === 3 was true

Output:
1
2

Newton School of Technology

Control statement: break (without)

Outcome without break would be different

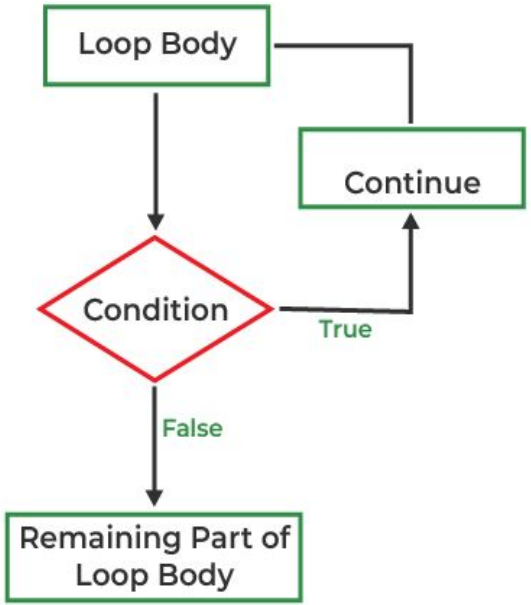
```
1  for (let i = 1; i <= 5; i++) {
2      if (i === 3) {
3          // No break here
4      }
5      console.log(i);
6  }
```

Since there is no break statement all the values got printed

Output:
1
2
3
4
5

Control statement: continue

The continue statement skips the current iteration of a loop and moves to the next one.

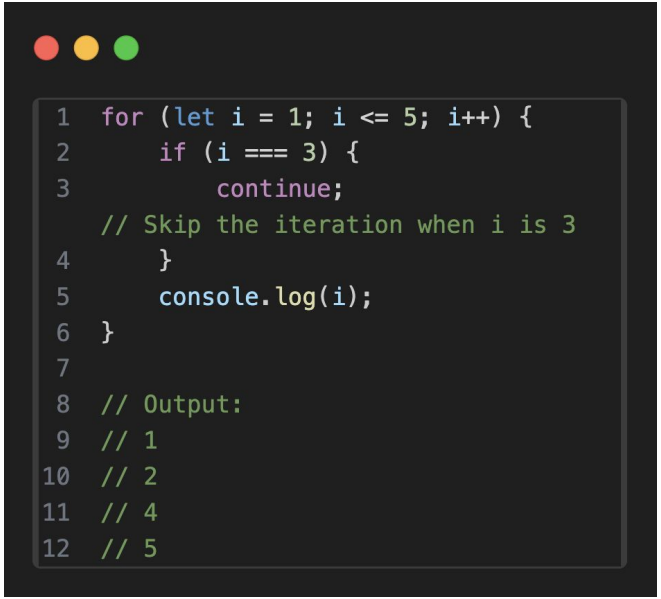


Newton School of Technology

Control statement: continue example

In this example, the continue statement skips the iteration when i is 3, so 3 is not printed, but the loop continues with the next values.

Go back in the previous slides and compare it with break statement.

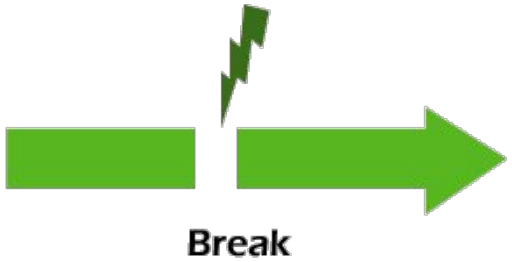


```
for (let i = 1; i <= 5; i++) {
    if (i === 3) {
        continue;
// Skip the iteration when i is 3
    }
    console.log(i);
}

// Output:
// 1
// 2
// 4
// 5
```

Newton School of Technology

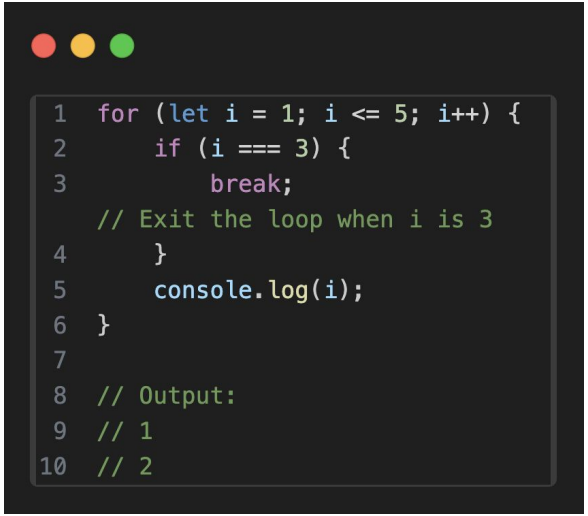
Difference: break vs continue

The break statement exits the loop entirely, while the continue statement skips the current iteration and moves to the next one.



Difference: break vs continue

Let’s compare the break and continue examples: In the break example, the loop stops entirely before reaching 3, while in the continue example, only 3 is skipped, and the loop continues until 5.



```
for (let i = 1; i <= 5; i++) {
    if (i === 3) {
        break;
// Exit the loop when i is 3
    }
    console.log(i);
}

// Output:
// 1
// 2
```

break

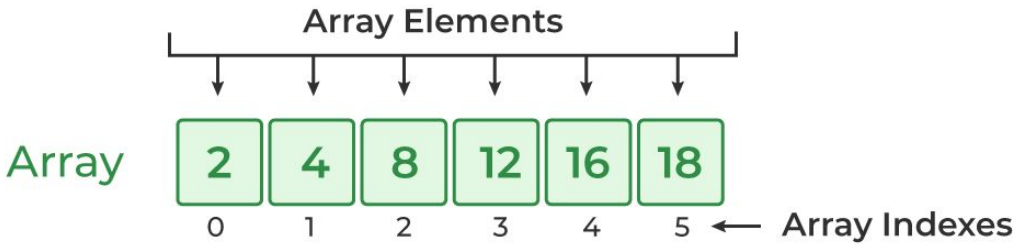
Newton School of Technology

Arrays and Objects

Newton School of Technology

What are arrays?

Arrays are one of the best data structures for storing multiple values in a single variable. Loops, such as for, while, or do-while are commonly used with arrays to process or manipulate their elements efficiently.



Newton School of Technology

Using array with for loop

Let’s use arrays along with loop. Here we will iterate array elements using for loop:-

```
1  // Create an array with various elements
2  const myArray = [1, 2, 3, 4, "Hello", true, null];
3
4  // Array declaration
5  const arr = [2, 4, 8, 12, 16];
6
7  // Using a for loop to iterate over the array
8  for (let i = 0; i < arr.length; i++) {
9      console.log(`Element at index ${i}: ${arr[i]}`);
10 }
```

Newton School of Technology

Objects: Real world dictionary

Think of an object as a real-world dictionary, where the key is the word you're looking up (e.g., "name" or "age"), and the value is the definition or meaning of that word (e.g., "Alice" or 20).

Key	Value
name	Alice
age	20

```
const person = {
    "name": "Alice",
    "age": 20
}
```

In Javascript

Newton School of Technology

Objects: Practical Example

When you have data that involves key-value pairs, objects are the most appropriate choice. Here is an example:-

```
1  const book = {
2      title: "The Great Gatsby",
3      author: "F. Scott Fitzgerald",
4      year: 1925,
5      genre: "Fiction"
6  };
```

Here, title, author, year, and genre are the keys (or properties) of the book object.

Newton School of Technology

Storing Objects in array

You can store several objects inside an array too:-

```
const books = [
    { title: "The Great Gatsby", year: 1925 },
    { title: "To Kill a Mockingbird", year: 1960 },
    { title: "1984", year: 1949 },
    { title: "Moby-Dick", year: 1851 },
    { title: "Pride and Prejudice", year: 1813 }
];
```

We often store multiple objects in an array is to keep things organized.

Newton School of Technology

Object Array: Iterating using for loop

And then iterate over them using for loop:-

```
const books = [
    { title: "The Great Gatsby", year: 1925 },
    { title: "To Kill a Mockingbird", year: 1960 },
    { title: "1984", year: 1949 },
    { title: "Moby-Dick", year: 1851 },
    { title: "Pride and Prejudice", year: 1813 }
];

for (let i = 0; i < books.length; i++) {
    console.log(`Title: ${books[i].title}, Year: ${books[i].year}`);
}
```

Newton School of Technology

Object Array: Iterating using for...in loop

It would be lot simpler if we use for...in loop instead:-

```
1  const books = [
2      { title: "The Great Gatsby", year: 1925 },
3      { title: "To Kill a Mockingbird", year: 1960 },
4      { title: "1984", year: 1949 },
5      { title: "Moby-Dick", year: 1851 },
6      { title: "Pride and Prejudice", year: 1813 }
7  ];
8
9  for (let i in books) {
10     console.log(`Title: ${books[i].title}`);
11     console.log(`Year: ${books[i].year}`);
12 }
```

In for...in loop we get keys and in this case '*i*' is the key. And then we use use that key to access the object in the array.

Newton School of Technology

Object Array: Iterating using for...of loop

If we want to access objects directly then we can use for...of loop

```
1  const books = [
2      { title: "The Great Gatsby", year: 1925 },
3      { title: "To Kill a Mockingbird", year: 1960 },
4      { title: "1984", year: 1949 },
5      { title: "Moby-Dick", year: 1851 },
6      { title: "Pride and Prejudice", year: 1813 }
7  ];
8
9  for (let book of books) {
10     console.log(`Title: ${book.title}`);
11     console.log(`Year: ${book.year}`);
12 }
```

for...of loop directly provides values stored in the array.

Which iteration method would you prefer?

Newton School of Technology

Object: for...in vs for...of

The main difference between them with respect to objects is that for...in is iterable over the keys of an object, whereas for...of can only iterate over an array.

```
1  // Using for...in
2  console.log('Using for...in:');
3  for (const key in obj) {
4      console.log(`${key} : ${obj[key]}`);
5  }
6
7  // Output
8  // a : 1
9  // b : 2
10 // c : 3
```

```
1  // will throw an error if used on an object
2  console.log('\nUsing for...of:');
3  try {
4      for (const value of obj) {
5          console.log(`Value: ${value}`);
6      }
7  } catch (error) {
8      console.log('Error: for...of cannot be used');
9  }
```

Newton School of Technology

Some Real World Examples

Calculating Discounts

Newton School of Technology

When shopping, it's crucial to calculate the best price after discounts. For example, if an item costs $100 and has a 20% discount, arithmetic operators can help us determine the discount amount and the final price.

```
let originalPrice = 100;
let discountPercentage = 20;
let discountAmount = (originalPrice * discountPercentage) / 100;
let finalPrice = originalPrice - discountAmount;

console.log("Original Price: $", originalPrice);
console.log("Discount: $", discountAmount);
console.log("Final Price: $", finalPrice);
```

Job Offers: Making Right Choice

Newton School of Technology

To choose the better job offer, you can use comparison operators to evaluate salaries. For example, using a greater-than operator helps you quickly see which offer provides a higher salary.

```
let offer1Salary = 50000;
let noffer2Salary = 55000;

if (offer2Salary > offer1Salary) {
    console.log("Offer 2 has a higher salary.");
}j else {
    console.log("Offer 1 has a higher salary.");
}
```

Shopping Cart: Calculating Bill

Newton School of Technology

You are creating a shopping cart where each item has a price. You want to calculate the total cost of all items in the cart.

```
const cart = [
    { item: "Laptop", price: 1200 },
    { item: "Headphones", price: 150 },
    { item: "Mouse", price: 50 },
    { item: "Keyboard", price: 80 }
];

let totalPrice = 0;

for (let i = 0; i < cart.length; i++) {
    totalPrice += cart[i].price;
    // Add each item's price to totalPrice
}

console.log(`Total Price: ${totalPrice}`);
```

Arrays, objects, and for loops (and their variations) are commonly used together to solve many programming problems.

In Class Questions

Newton School of Technology

Thanks for watching!