Build a Strong MERN Foundation

JavaScript Fundamentals

A Day in the Life of a MEAN Stack Developer

You are working as a MERN stack developer, but your current focus is on mastering the fundamentals of JavaScript. While your typical role encompasses the broader MERN stack (MongoDB, Express.js, React, and Node.js), your current lesson centers solely on JavaScript's core concepts.

Your project entails building a strong foundation in JavaScript to tackle a wide range of web development tasks. To succeed in this endeavor, you need to grasp key concepts related to JavaScript programming.

To achieve these objectives, you will be learning fundamental JavaScript concepts in this lesson, equipping you with the skills needed to excel in web development tasks that require JavaScript.

Learning Objectives

By the end of this lesson, you will be able to:

- Describe JavaScript and DOM for creating interactive and dynamic web pages
- Comprehend the basics of variables, data types, and operators for effective programming and problem-solving
- Apply control flow statements to navigate program execution through effective use of conditionals and loops
- Use arrays to enhance data organization and manipulation skills in programming

Learning Objectives

By the end of this lesson, you will be able to:

- Develop proficiency in handling and manipulating strings for effective data processing in JavaScript
- Develop prototypes of functions for efficient and scalable data manipulation in JavaScript applications
- Analyze the functionalities and applications of AI coding assistants to effectively integrate them into software development workflows

Introduction to JavaScript

What is JavaScript?

JavaScript is a lightweight scripting programming language used to make webpages interactive. It can also calculate, validate, and manipulate the data.

- It can insert dynamic text into HTML and CSS and make the webpage interactive.
- It can be used in front-end and back-end web development.

Why JavaScript?

The most important features of JavaScript are:

01

Tasks related to data manipulation are much easier on JavaScript.

02

JavaScript is completely compatible with HTML and CSS.

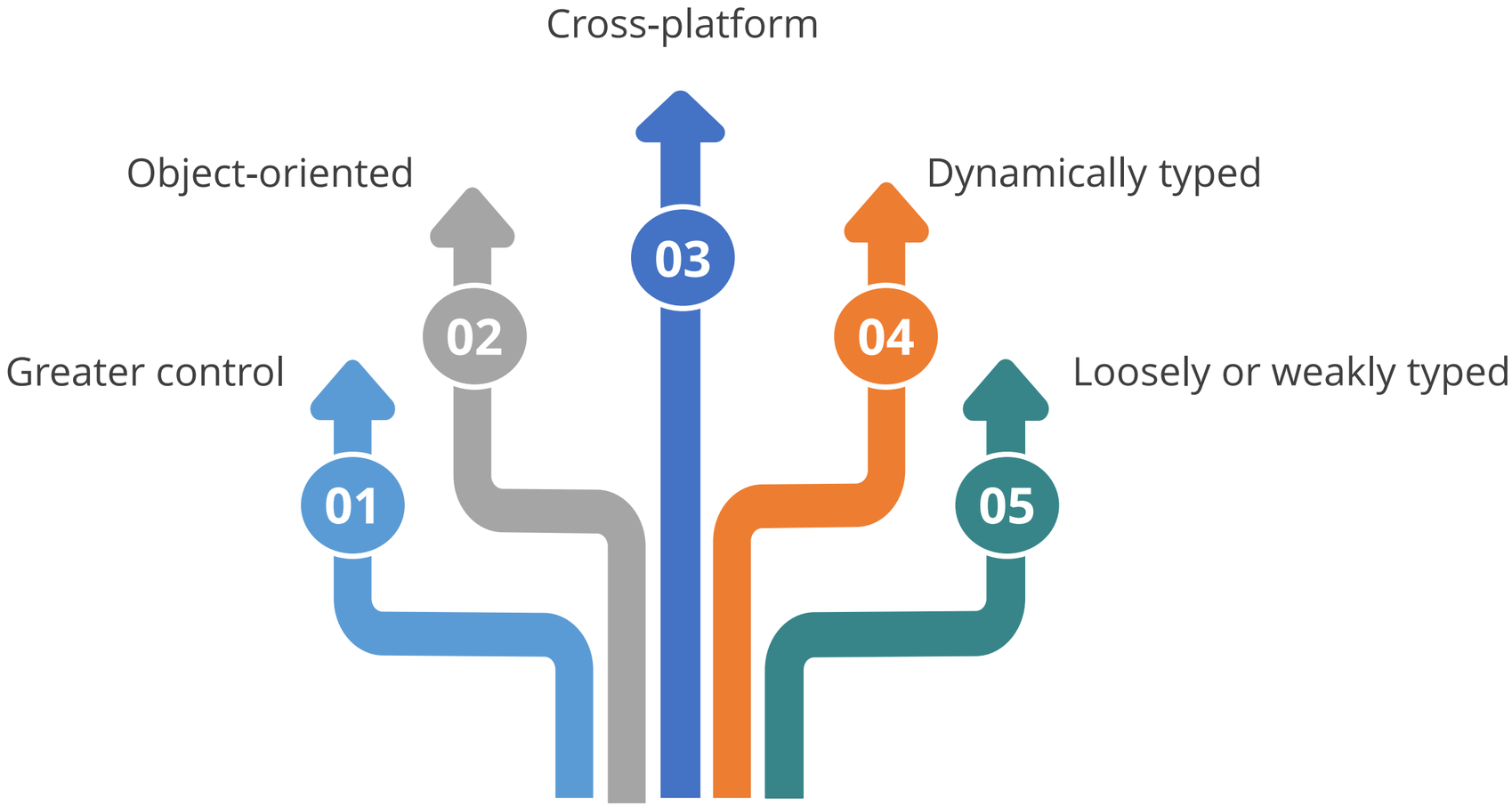
03

JavaScript is the foundation for many front-end frameworks like Angular and React.

04

JavaScript is supported by most of the browsers by default.

Benefits of JavaScript



JavaScript Application

01 Mobile app creation

02 Game development

03 Web development

04 Smart watch application

05 Server-side development

JavaScript Basics

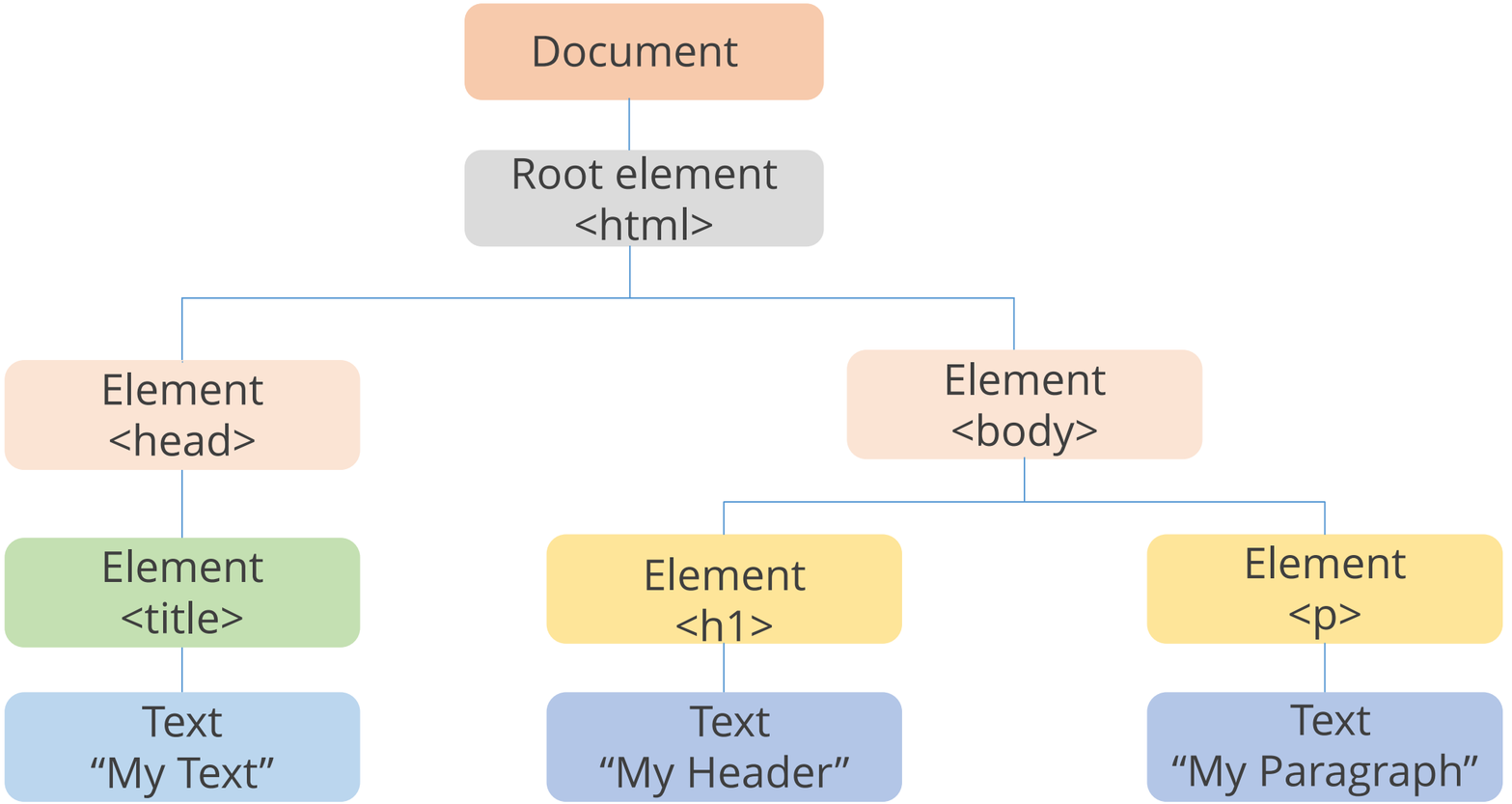
What Is DOM?

The document object model (DOM) is a programming interface for web documents.

Nodes and objects are used to represent the document in the DOM.

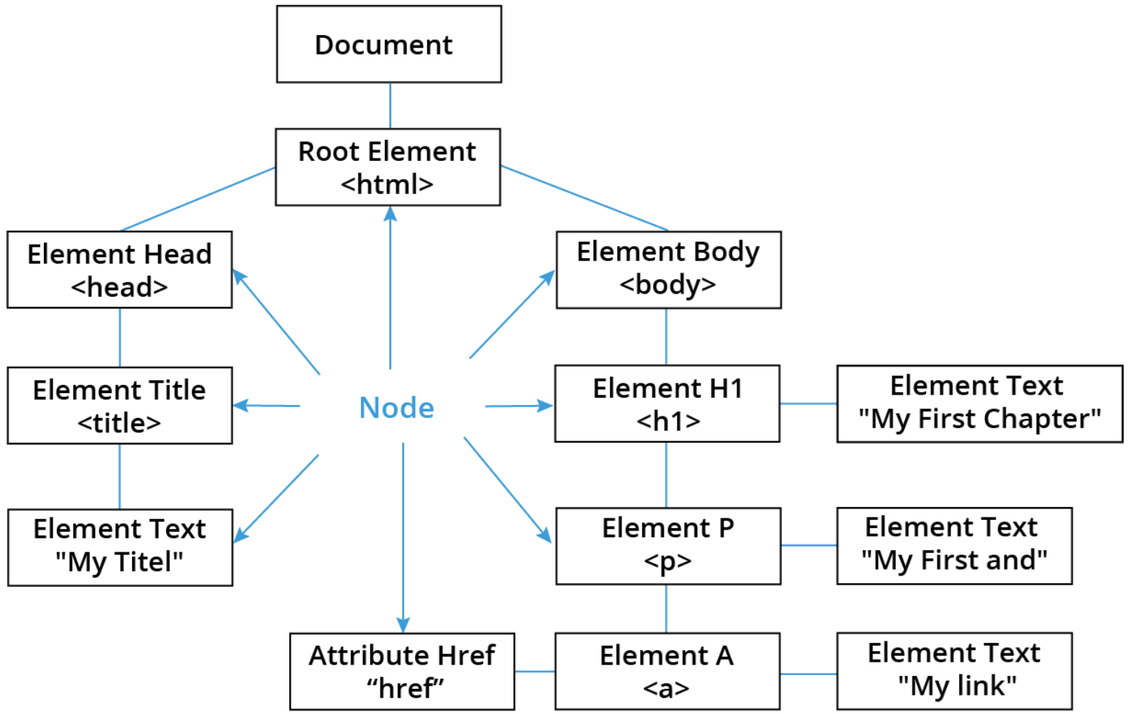
DOM Model

DOM defines the structural representation of the document, as shown below:



DOM Nodes

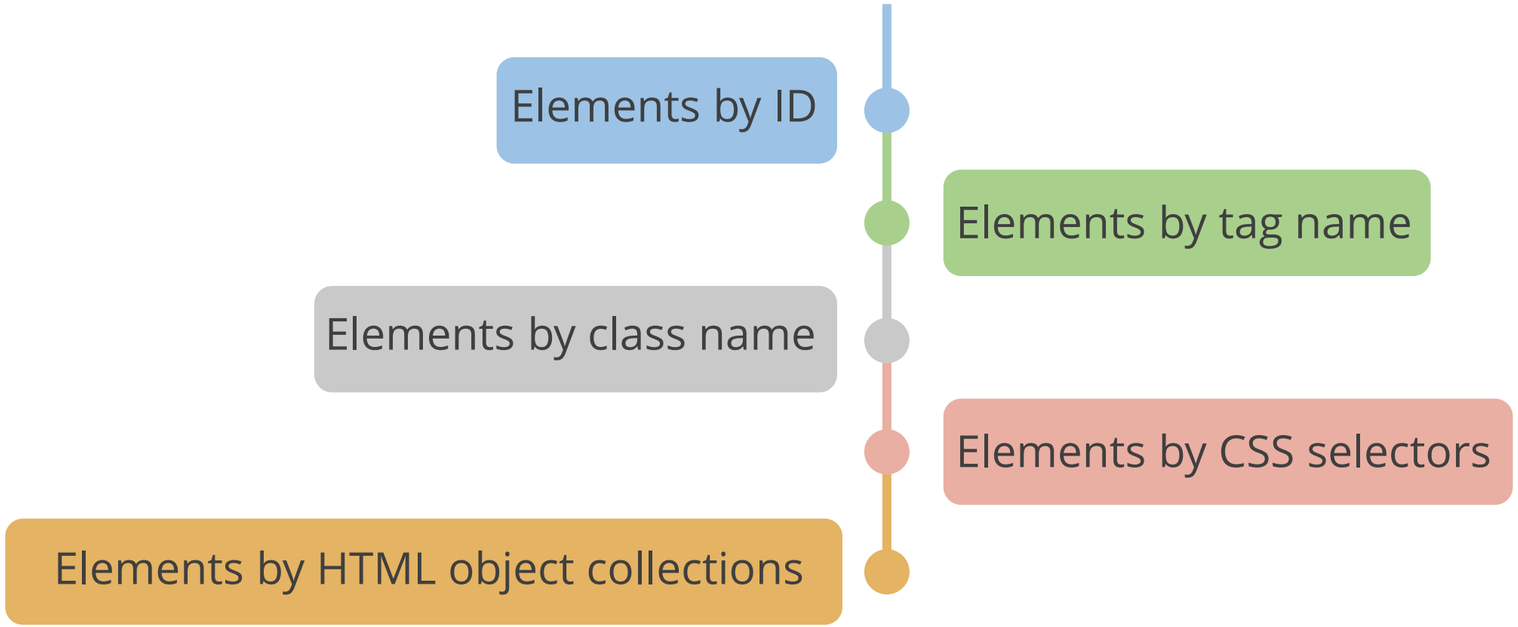
All elements, attributes, text, and other aspects of the page are grouped in a hierarchical tree-like structure in the DOM.



Nodes are the individual components of a document.

DOM Elements

DOM elements include DIV, HTML, and BODY elements. Users can access and manipulate DOM elements using any of the following methods:



DOM and JavaScript

DOM is not a programming language, but it is required for JavaScript to have a model or notion of web pages (HTML documents) and SVG documents.

Methods of DOM

Some important methods of document object:

Method	Description
write("string")	The string is written to the document.
writeln("string")	The given string is written to the document with a newline character at the end.
getElementById("string")	The value of the element with the given ID is returned.

Methods of DOM

Some important methods of document object:

Method	Description
getElementsByName()	All elements with the specified name value are returned.
getElementsByTagName()	All elements with the specified tag name are returned.
RegExp	Regular expression is represented.

DOM Programming Interface

All elements within a web page are controlled by the HTML Document Object Model (DOM).

01 JavaScript can access the HTML document object model.

02 All HTML elements are defined as objects in the DOM.

DOM Programming Interface

03 Each object's properties and methods make up its programming interface.

04 A property is a value that can be obtained or changed by the user (like changing the content of an HTML element).

05 A method is an action that a user can perform (like adding or deleting an HTML element).

JavaScript and HTML

The **innerHTML** property is the simplest way to change the content of an HTML element.

Example:

```
<html>
<body>
<p id="p1">Hello World!</p>
<script>
document.getElementById("p1").innerHTML = "New text!";
</script>
</body>
</html>
```

JavaScript and HTML

JavaScript can create dynamic HTML content.

Example:

```
<!DOCTYPE html>
<html>
<body>
<script>
document.getElementById("demo").innerHTML = "Date : " + Date();
</script>
</body>
</html>
```

Variables and Data Types

Variables

Variables are the names users give to the memory locations in a computer program where values are stored.

There are two types of variables in JavaScript:

1. **Local variable**
2. **Global variable**

Variables

Local variable

A local variable is declared inside a block or function. It is accessible within the function or block only.

Global variable

A global variable is accessible from any function. A variable declared outside the function or declared with a window object.

Variables are declared using the **var** or **let** keyword, and they can be reassigned new values.

Variables: Example

Example:

```
// Variable declared with 'var'
var varVariable = 42;
console.log("Original varVariable:", varVariable); // Output: 42

// 'var' allows reassignment
varVariable = 20;
console.log("Reassigned varVariable:", varVariable); // Output: 20

// Variable declared with 'let'
let letVariable = "Hello";
console.log("Original letVariable:", letVariable); // Output: Hello

// 'let' allows reassignment
letVariable = "World";
console.log("Reassigned letVariable:", letVariable); // Output: World
```

Constants

It represents an unchanging value throughout a program's execution, enhancing code clarity and preventing reassignment.

Example:

```
const constantExample = 10;

// Uncommenting the line below will result in an error

// constantExample = 20;
// TypeError: Assignment to constant variable.
```

Constants are declared using the *const* keyword.

Constants cannot be reassigned after their initial assignment.

let and const

The difference between *let* and *const* keywords:

let

- *let* is used when you need to reassign a variable.
- It declares a local variable in a block scope and can be used for loops or mathematical operations.

const

- *const* means that the identifier cannot be reassigned.
- The scope of the *const* statement is like the scope of the *let* statement.

Blocks

- A *block statement* is a group of zero or more statements
- Identifiers declared with *let* and *const* do have block scope

let

Example:

```
let x=1;
{
let x=5;
}

console.log(x); //answer will be 1
```

const

Example:

```
const y=20;
{
const y=40;
}

console.log(y); //answer will be 20
```

Blocks

The block scope limits outside access to variables that are declared inside a certain block.

Example:

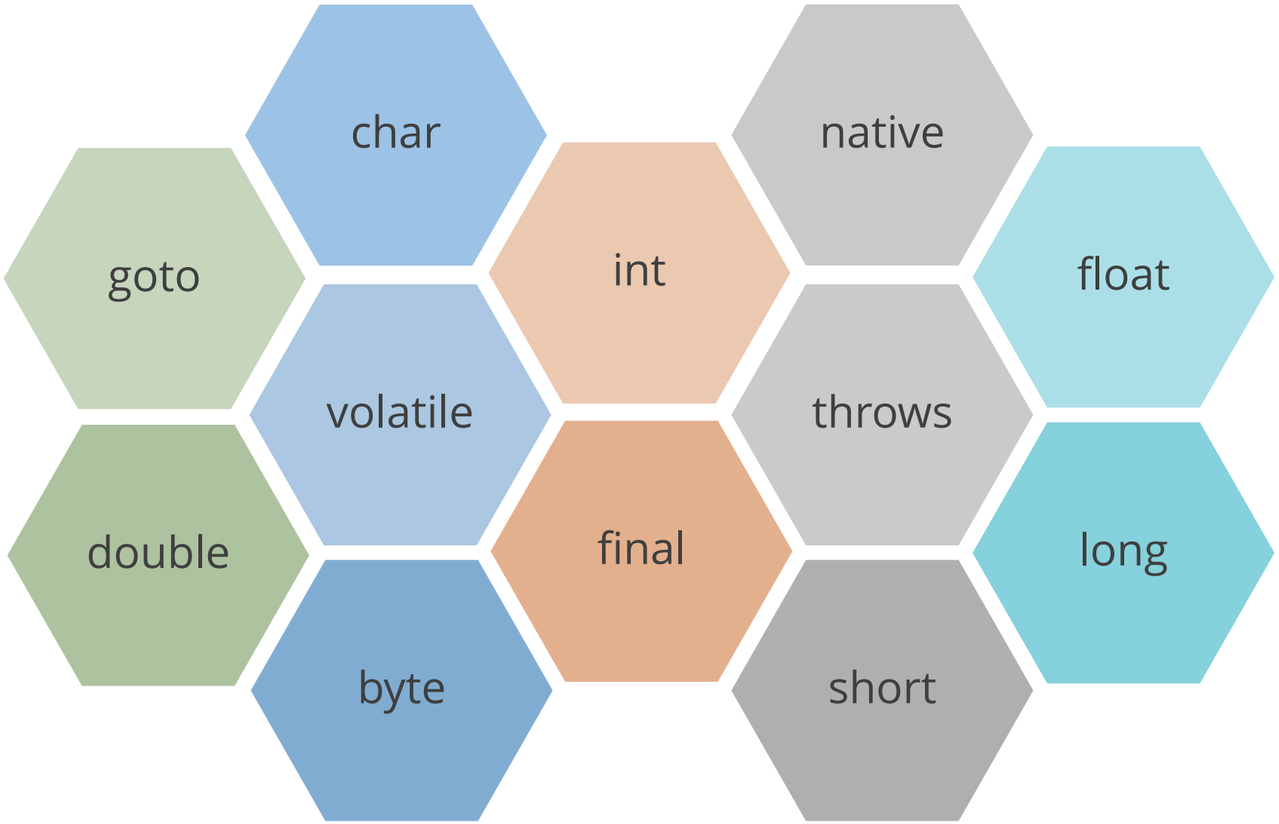
```
function display(){
    if(true){
        var item1 = 'apple';         //exist in function scope
        const item2 = `ball';         //exist in block scope
        let item3 = `cloud';      //exist in block scope
    }
    console.log(item1);
    console.log(item2);
    console.log(item3);
}

display();
//result:
//apple
//error: item2 is not defined
//error: item3 is not defined
```

Keyword

Keywords are reserved words that have a specific meaning for the compiler.

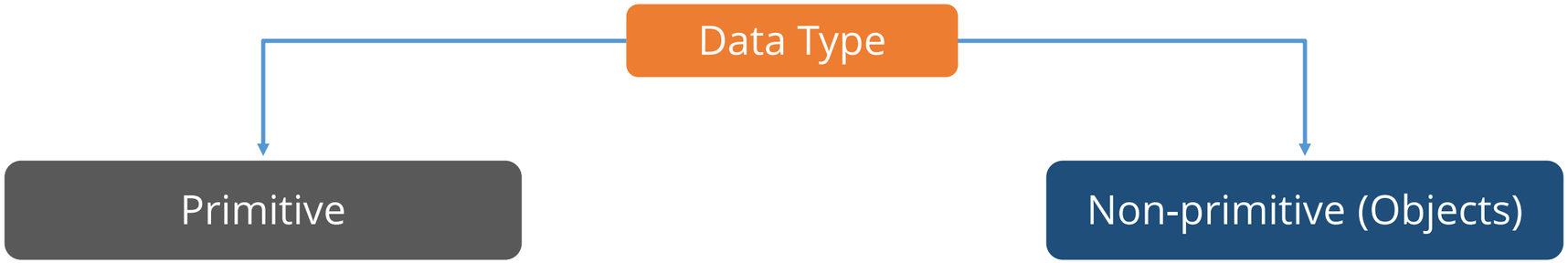
Keyword

The lists of some standard reserved words are given below:



Data Types in JavaScript

A data type is a classification that specifies a variable's value and what mathematical, relational, or logical operations can be applied to it without any error.

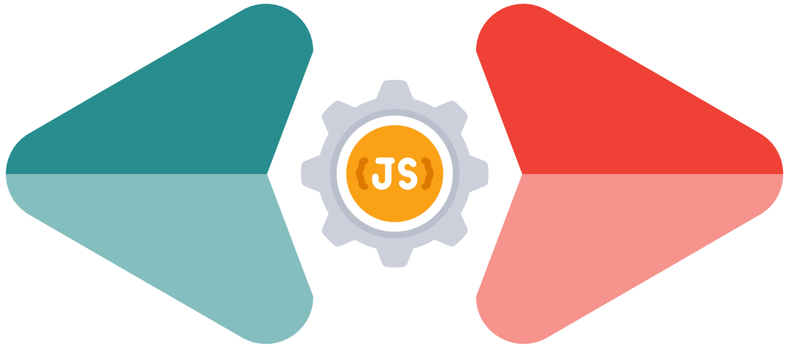


Data Types in JavaScript

There are two types of data types in JavaScript:

Primitive data type

Primitive data types are predefined types in the programming language.



Non-Primitive data type

Non-primitive data types are reference types that refer to objects.

Primitive Data Type

A primitive data type is not an object and has no methods and properties.

The types of primitives data types are:

1. Numbers

0 1 2 3 4 5 6

2. Strings

“Hello World”

3. Boolean

1010101

4. Null

{ }

5. Undefined

?

Non-Primitive Data Type

Non-primitive data types (Objects) are not defined by the programming language but can be created by the programmer.

The types of non-primitives data types are:

1 Objects

2 Arrays

3 Functions

4 Date

5 Regex

{}*?@[^]\.

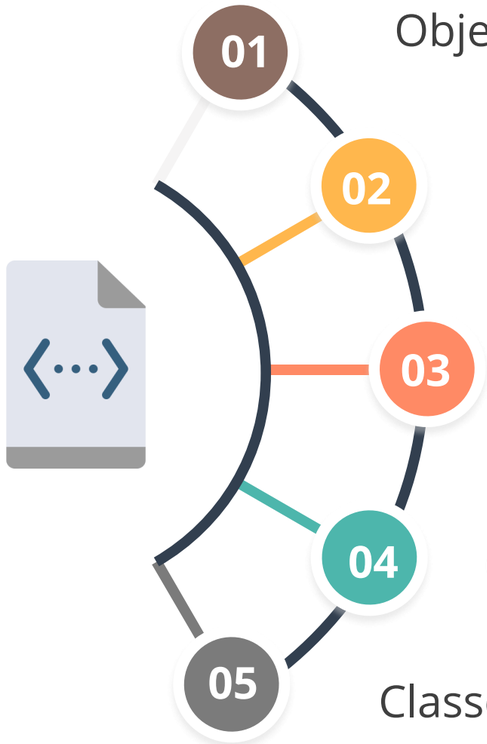
Primitive Vs Non-Primitive

The differences between primitive and non-primitives are:

Primitive	Non-Primitives
The primitive data types are not mutable, the values cannot be changed after creation.	The non-primitive data types are mutable, the values can be changed after creation.
Primitive Data types are predefined.	Non-primitive data types are user-defined.
The primitive data types are compared by values.	The non-primitive data types are compared by reference.

Non-Primitive (Objects)

The different methods to create objects are:



Object Literal

new keyword with object constructor

using factory functions

Object.create()

Classes

Objects Creation

The object literal can be created by using key-value pairs.

Example:

```
var greeting = {
Fullname: "Ben Shapiro",
greet: (message, name) => {
console.log(message + " " + name + "!!");
}
};
```

The object literal is an array of key-value pairs with commas (,) following each key-value pair and colons (:) separating the keys and values.

Objects Creation

Using the new operator, the programmer can create an instance of a user-defined or built-in object type.

new operator with object constructor

Example:

```
let student = new Object();

student.Name = 'testName';
student.Rollno = 'testRollno';
```

new operator with a constructor function

Example:

```
function student(Name, Rollno)
{
  this.FullName = Name;
  this.Rollno = rollno;
}

let student1 = new
Student('Fullname', 'Rollno');
```

Objects Creation

The Object.create() method generates a new object by using an existing object as a prototype.

Example:

```
let org = { company: 'Simplilearn' };

let student = Object.create(org, { name: { value: 'student1' } });

console.log(student);
console.log(student.name);
```

Objects Creation

The class method is like using new operator with a user-defined constructor function.

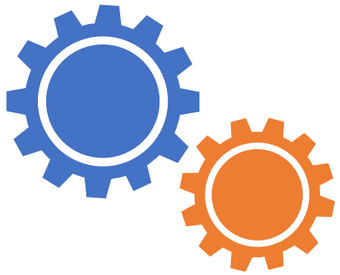
Example:

```
class student {

  constructor(Name, Rollno) {
    this.Name = fullname;
    this.Rollno = rollno;
  }

}
```

Data Type Conversion

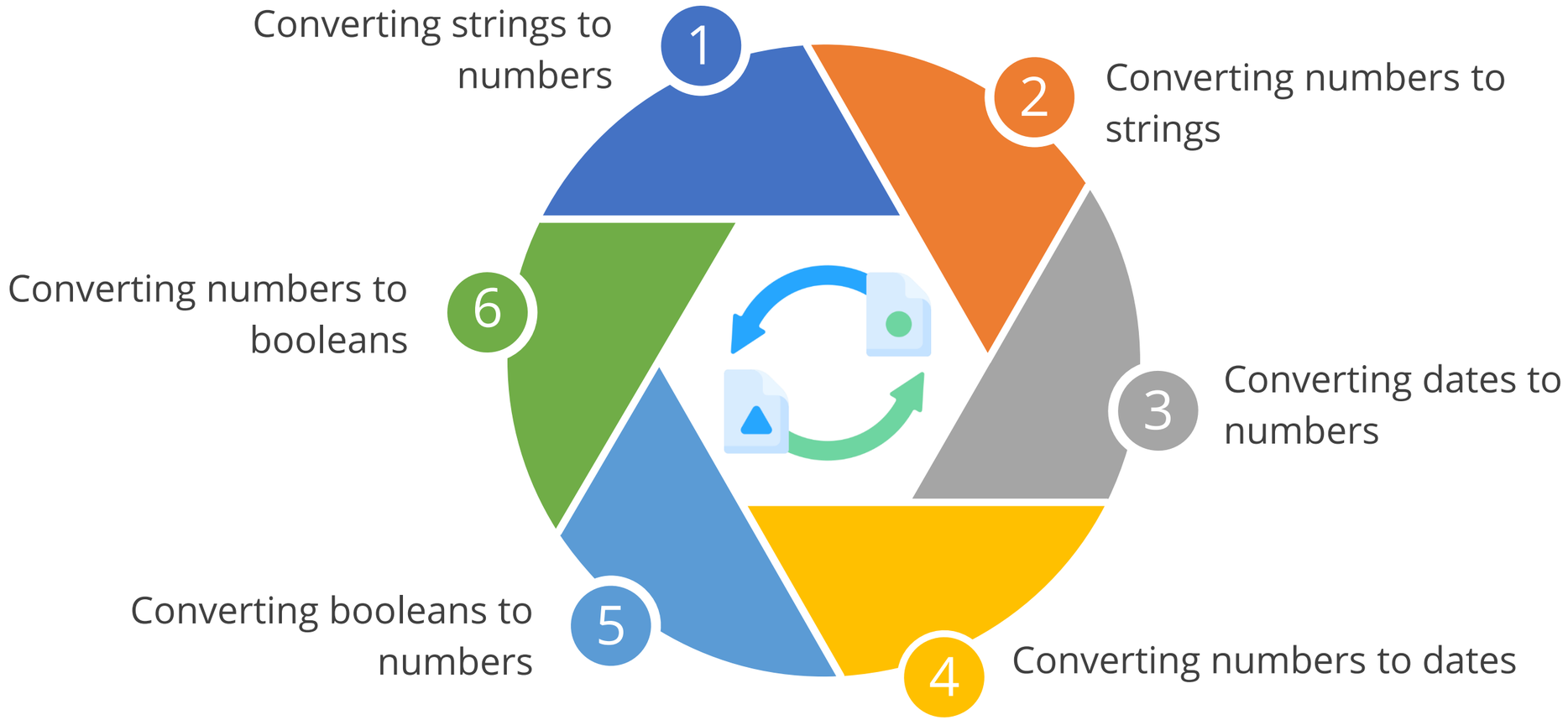
Also known as type coercion, it is a fundamental aspect of JavaScript allowing developers to transform data from one type to another.

Implicit conversion:
JavaScript automatically performs implicit conversion when needed for an operation, trying to convert values to the required types.



Explicit conversion:
It involves manually converting a value from one type to another using built-in functions or methods.

Data Type Conversion: Common Conversion Functions



Assisted Practice

Demonstrating String Methods **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use of variables and primitive datatypes in JavaScript.

Assisted Practice: Guidelines

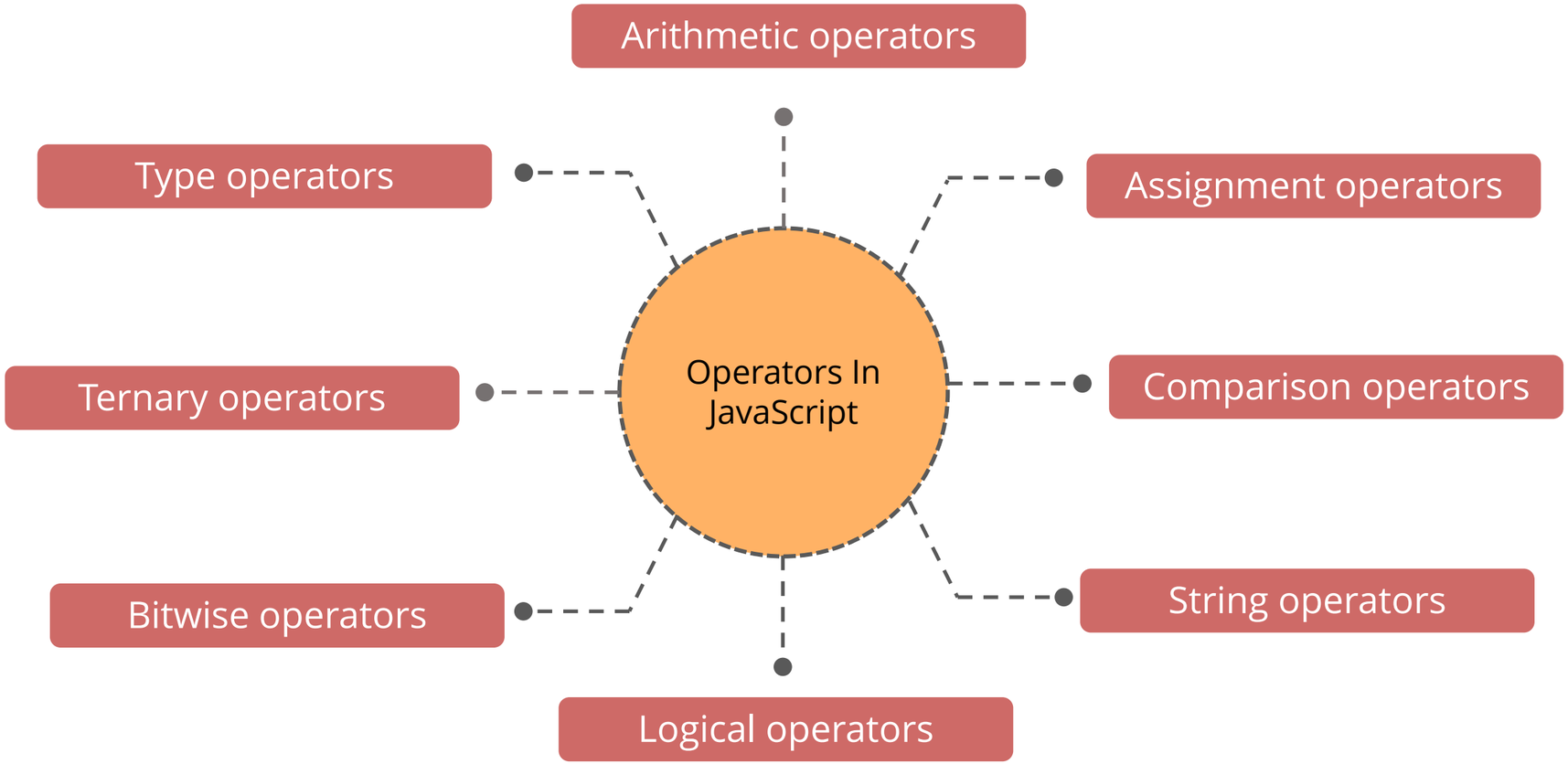
Steps to be followed:

1. Create and execute JS file

Operators and Expressions

Operators

The following are the different types of JavaScript operators:



Arithmetic Operators

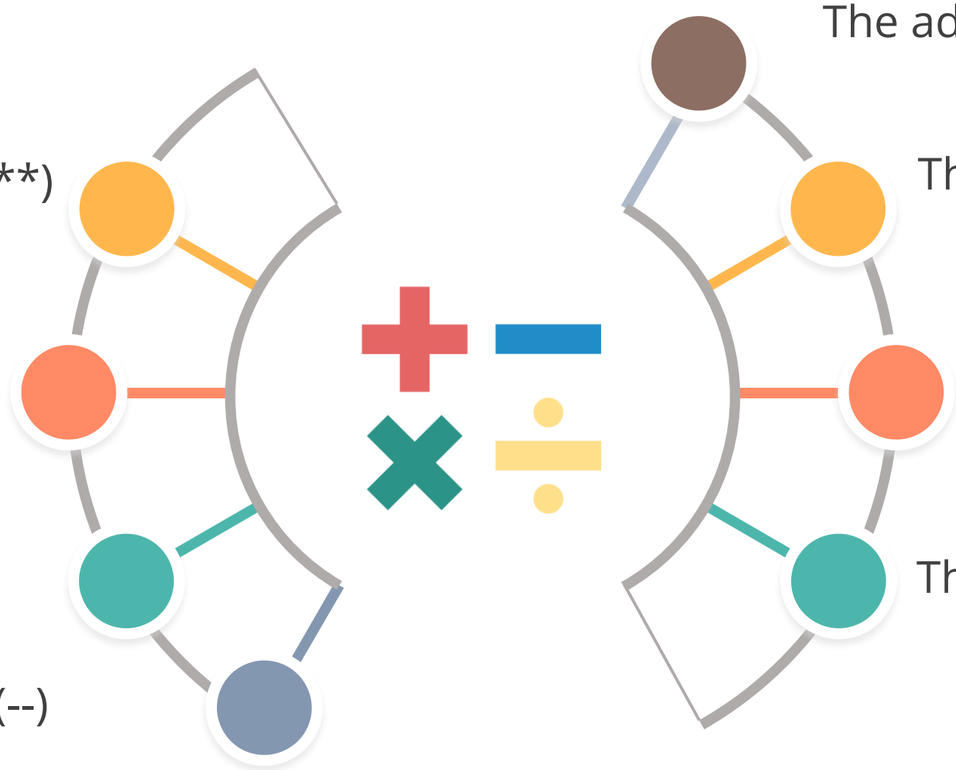
They are used to perform mathematical operations like addition, subtraction, multiplication, division, and modulus on the given operands.

The Exponentiation operator (**)

The Modulus operator (%)

The Increment operator (++)

The Decrement operator (--)



The addition operator (+)

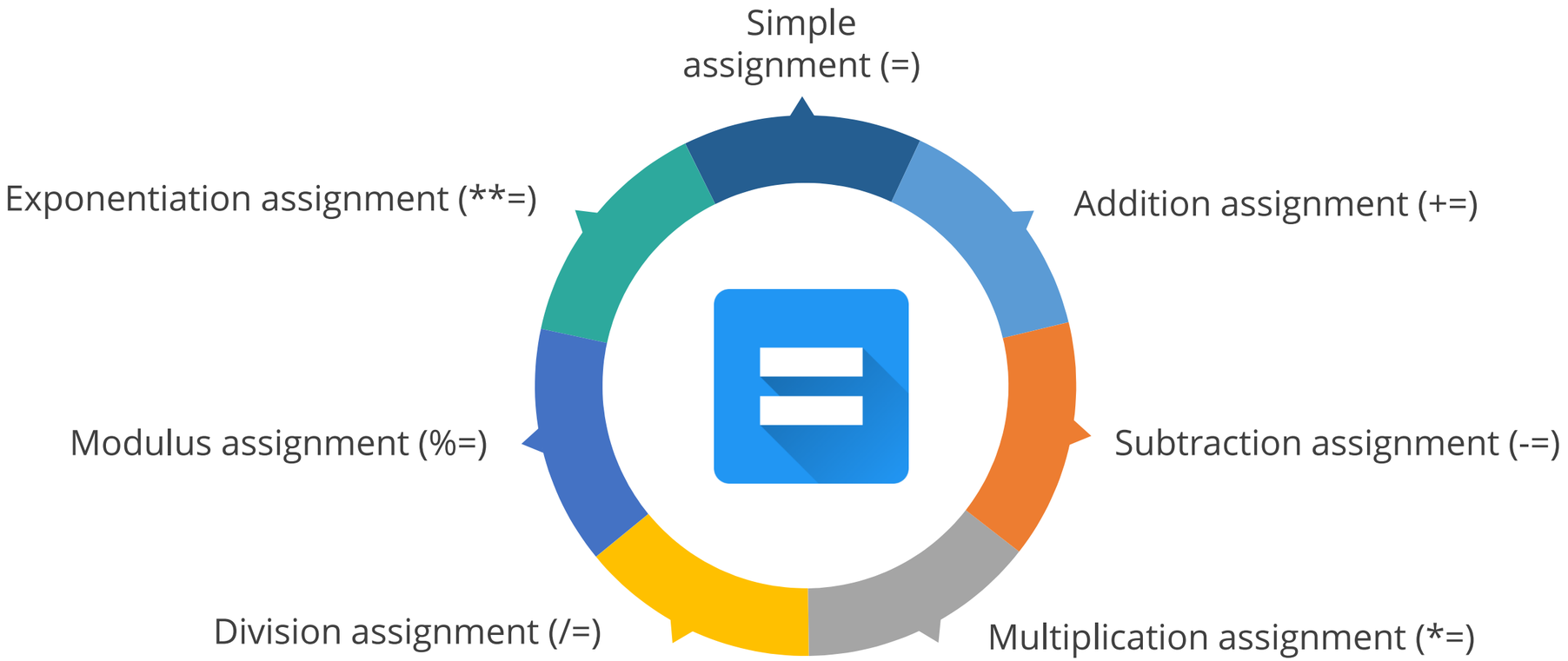
The subtraction operator (-)

The asterisk operator (*)

The slash operator (/)

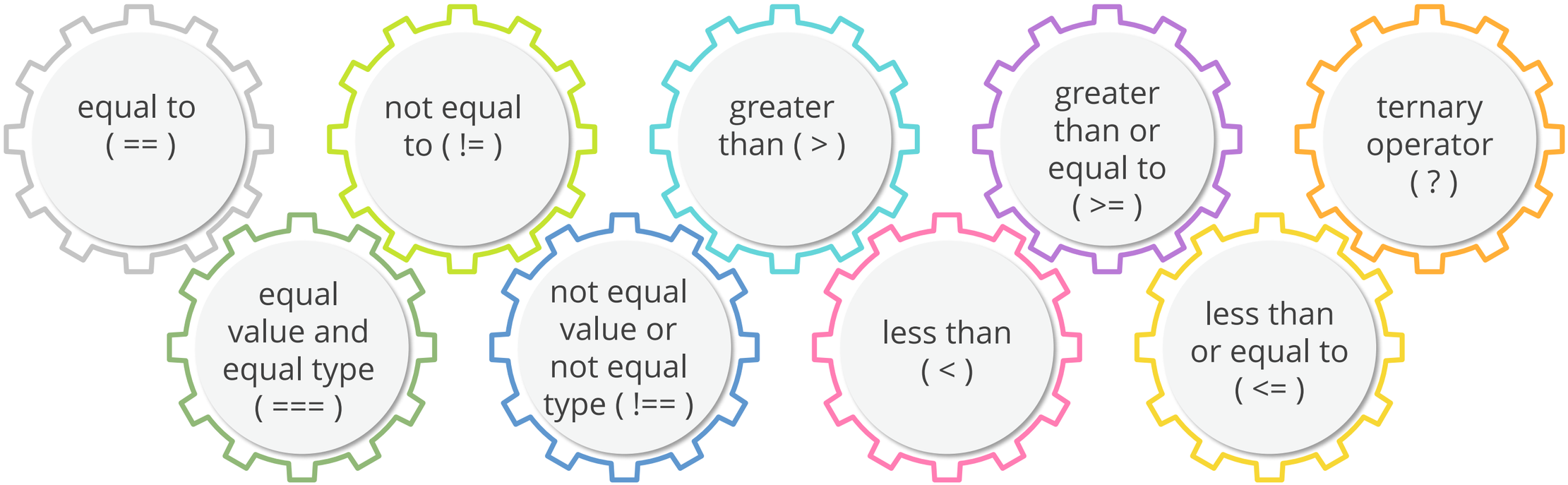
Assignment Operators

They assign values to variables, following are the operators used for assigning values in JavaScript:



Comparison Operators

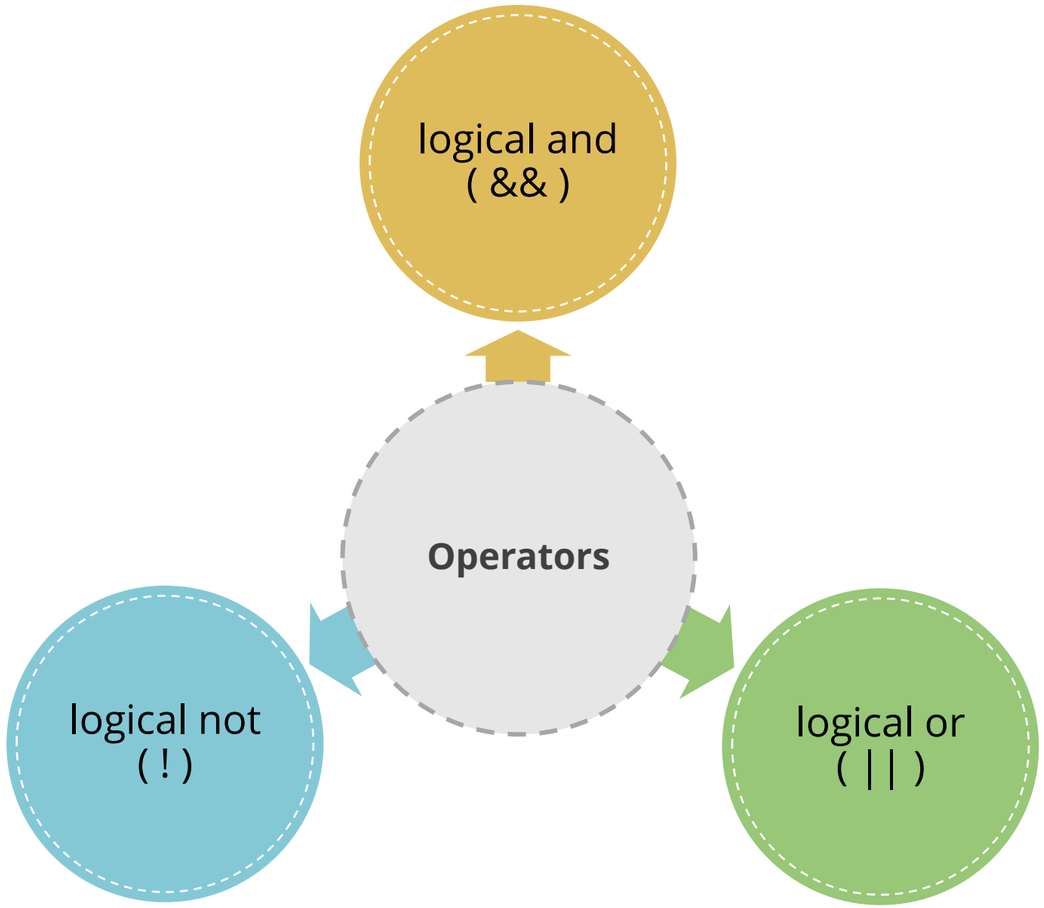
They compare values and return a boolean result, following are the operators used to compare the values:



All the comparison operators above can also be used on strings.

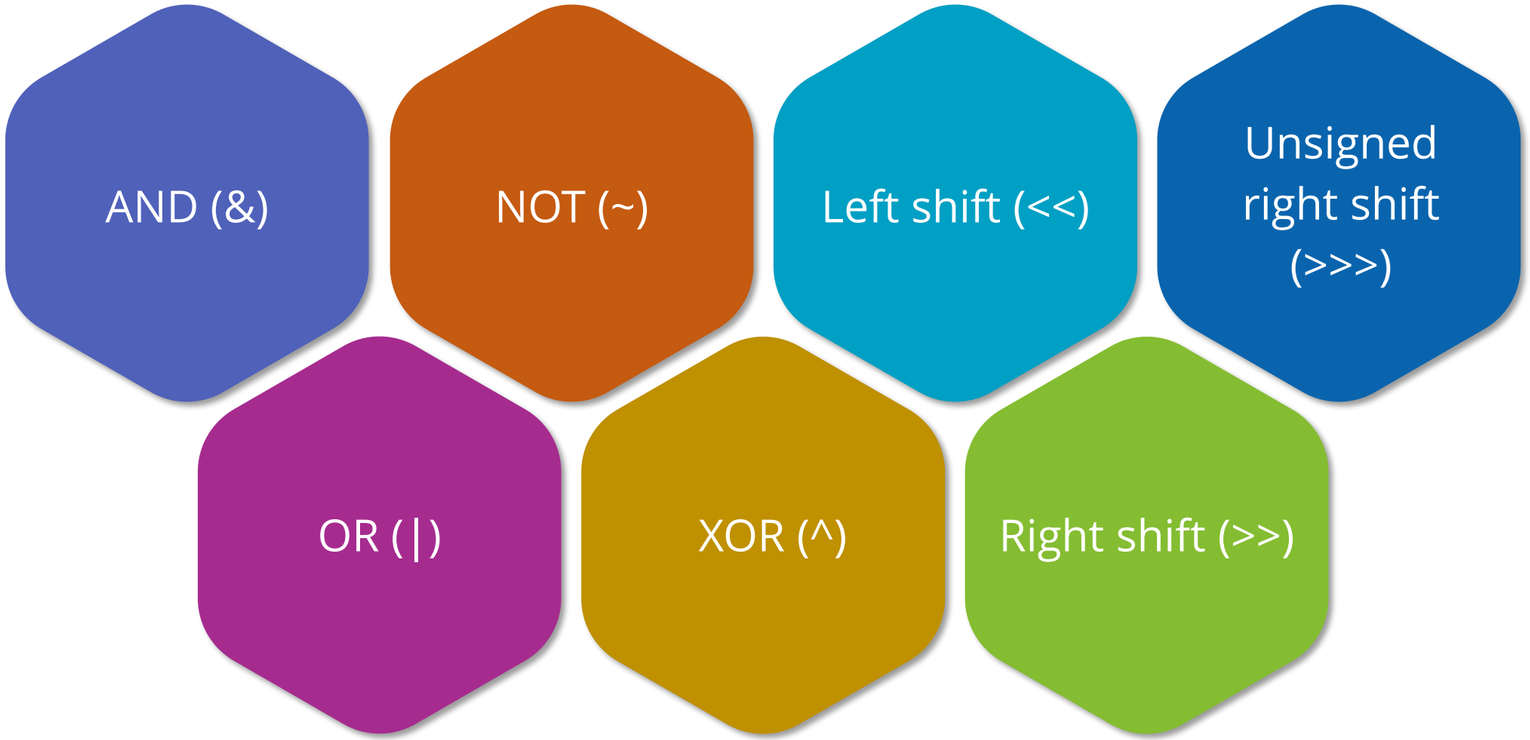
Logical Operators

They perform logical operations and return a boolean result, following are the operators used for logical operations:



Bitwise Operators

They perform operations on binary representations of numbers, following are the operators used for bitwise operations:

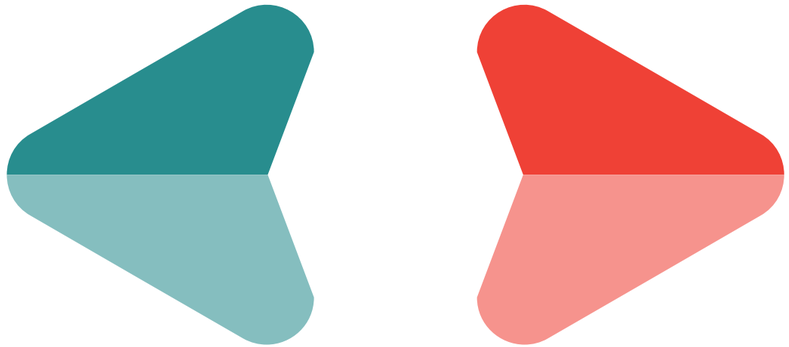


Type Operators

They perform operations related to data types. The two main type operators in JavaScript are as follows:

typeof

Returns the type of a variable



instanceof

Returns true if an object is an instance of an object type

Operator Precedence and Associativity

Operator precedence

It determines the order in which operators are evaluated in an expression.

Higher precedence operators are evaluated first.

Parentheses can be used to override the default precedence.

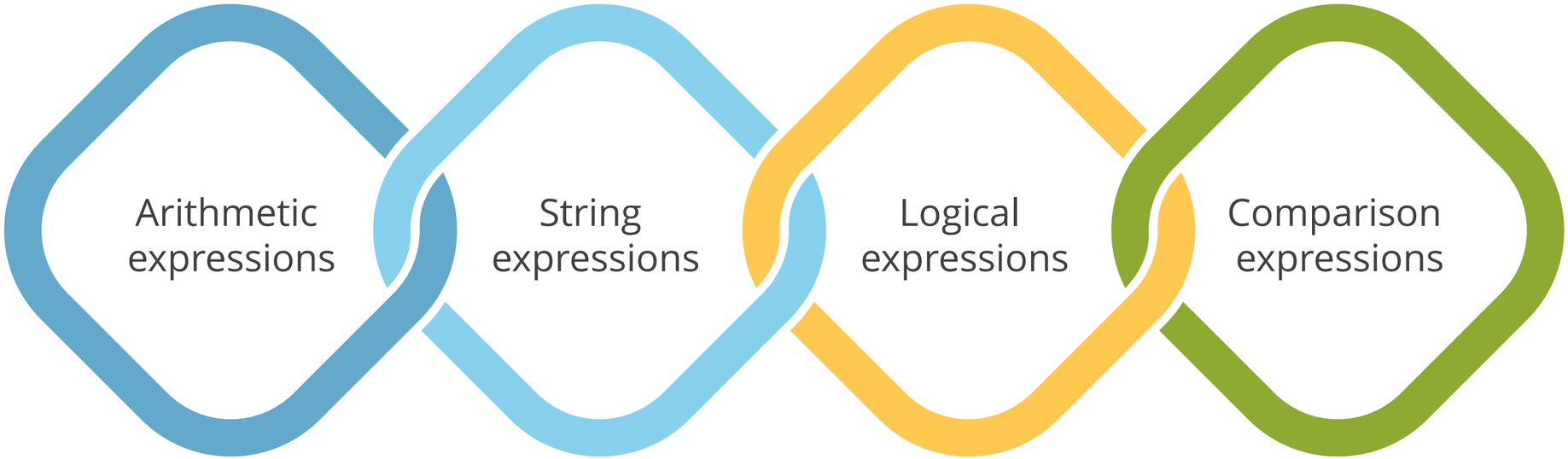
Operator associativity

It defines the order in which operators of the same precedence are evaluated.

Left-to-right associativity means the leftmost operation is evaluated first, and vice versa.

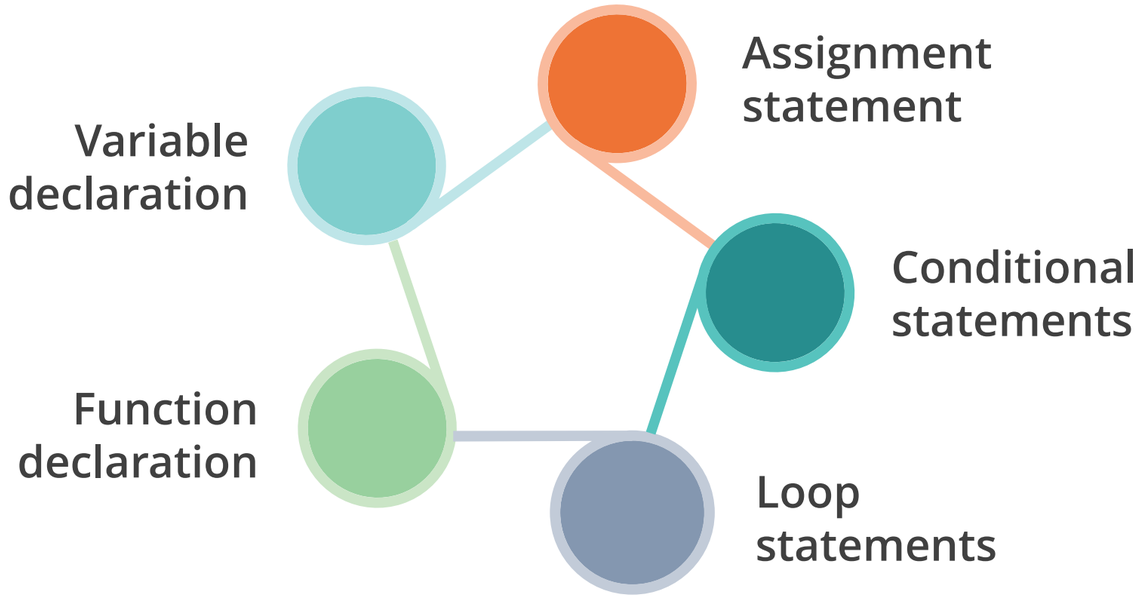
Expression

It is the combination of values, variables, and operators that can be evaluated to a single value. The following are the types of expressions:



Statement

It is a complete line of code that performs a specific action. The following are the types of statements:



Unlike expressions, statements do not necessarily result in a value.

Assisted Practice

Demonstrating Operators and Expressions **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use of operators and expressions in JavaScript.

Assisted Practice: Guidelines

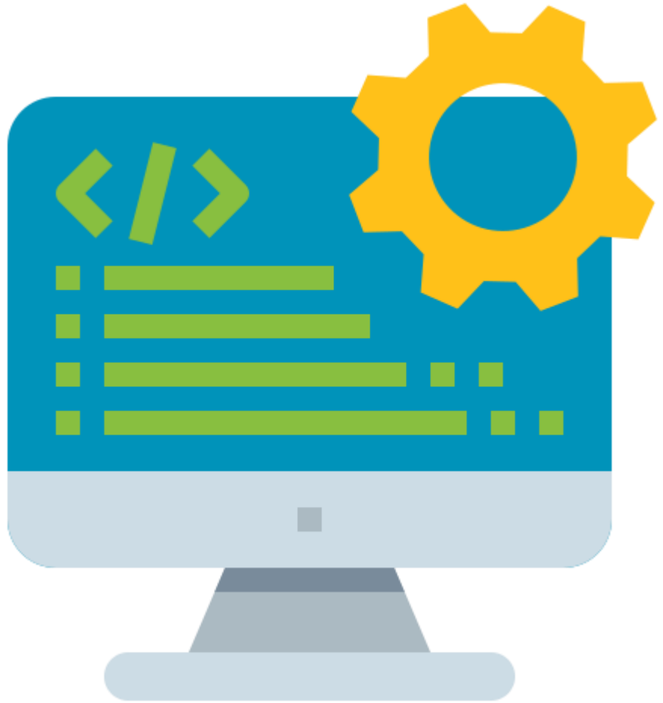
Steps to be followed:

1. Create and execute JS file

Control Flow Statements

Control Statements

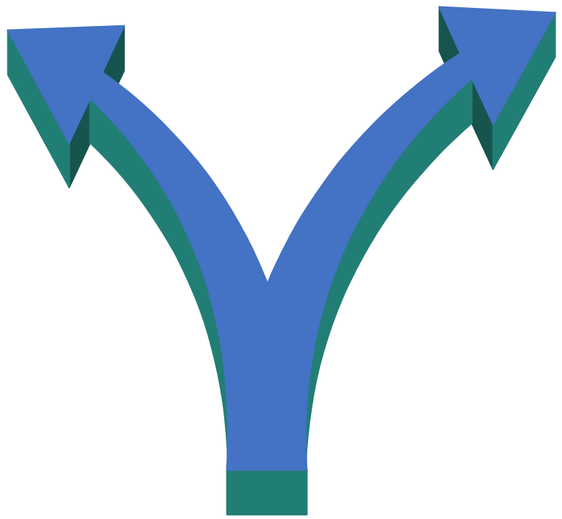
A control statement allows structured control of the sequence of application statements, such as loops and conditional tests.



Control Statements

Control statements are divided into two categories:

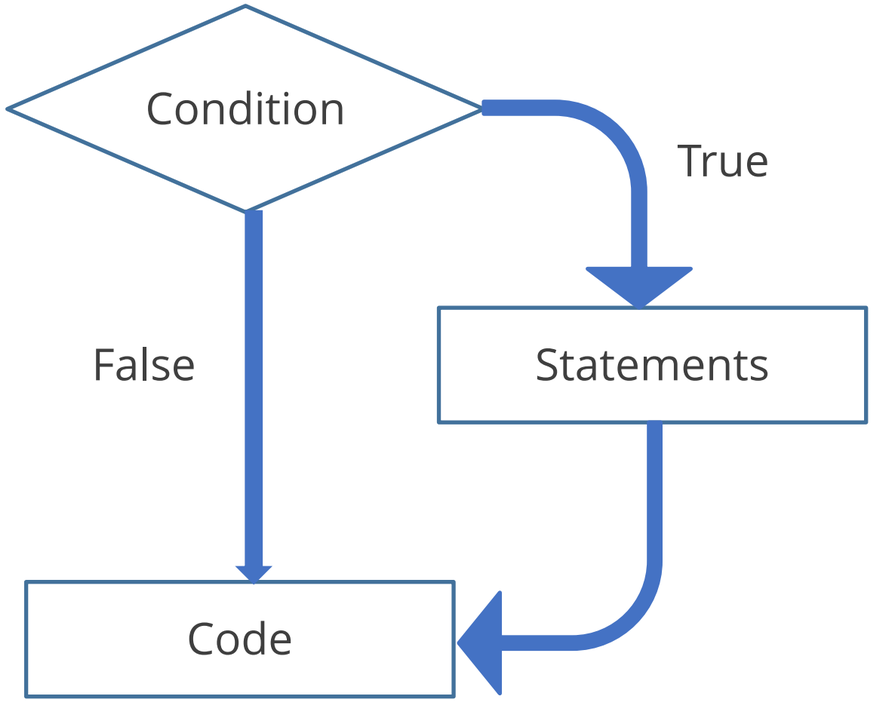
Conditional statements



Iterative statements

The control statement allows switching the input stream to another file, opening and closing files, and various other operations.

Conditional Statements



- When the condition is passed, if that is true, then the program moves to the next step.
- If the condition is false, then the program moves to another step.

Conditional Statements

If statement

It evaluates the content only if the expression is true.

Example:

```
<script>
var a=20;
if(a>10){
document.write("value of a is greater than 10"
);
}
</script>
```

If else statement is an extended version of the if statement.

Conditional Statements

If else statement

It evaluates the content if the condition is true or false.

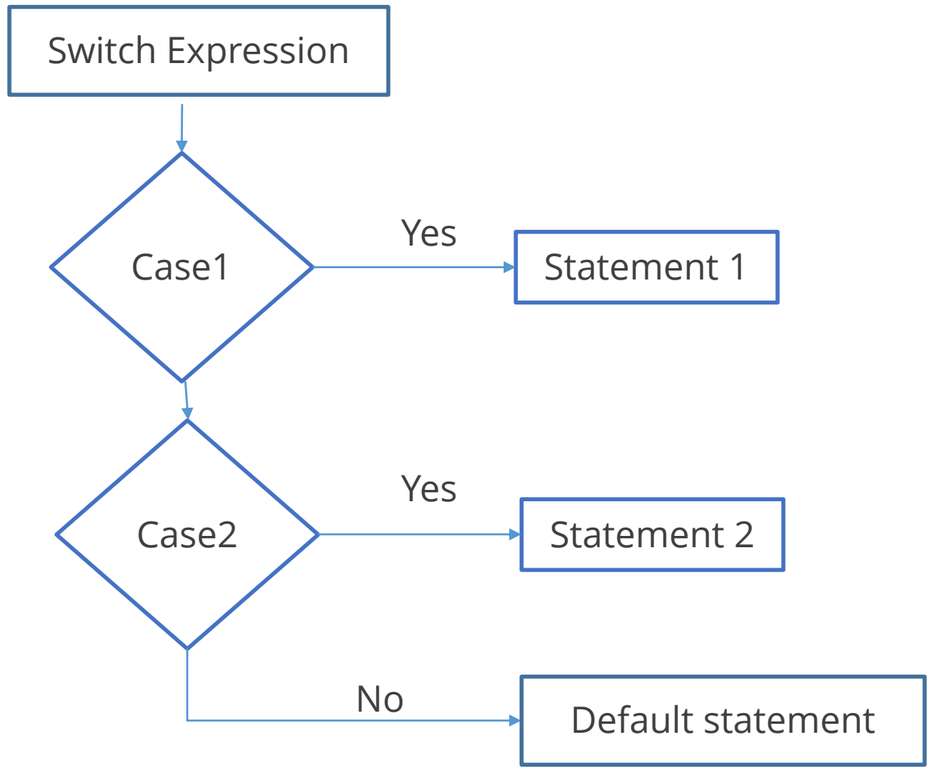
Example:

```
<script>
var a=20;
if(a%2==0){
document.write("a is even number");
}
else{
document.write("a is odd number");
}
</script>
```

Conditional Statements

Switch statement

The JavaScript switch statement is used to execute one code from multiple cases.



- The switch statement compares the value of an expression to several case clauses before executing it.
- Unless a break statement is found, statements after the first case clause with a matching value are executed.

Conditional Statements

The below code is an example of a switch statement:

Example:

```
<script>
var grade='B';
var result;
switch(grade){
case 'A':
result="A Grade";
break;
case 'B’:
result="B Grade";
break;
default:
result="No Grade";
}
document.write(result);
</script>
```

Iterative Statements

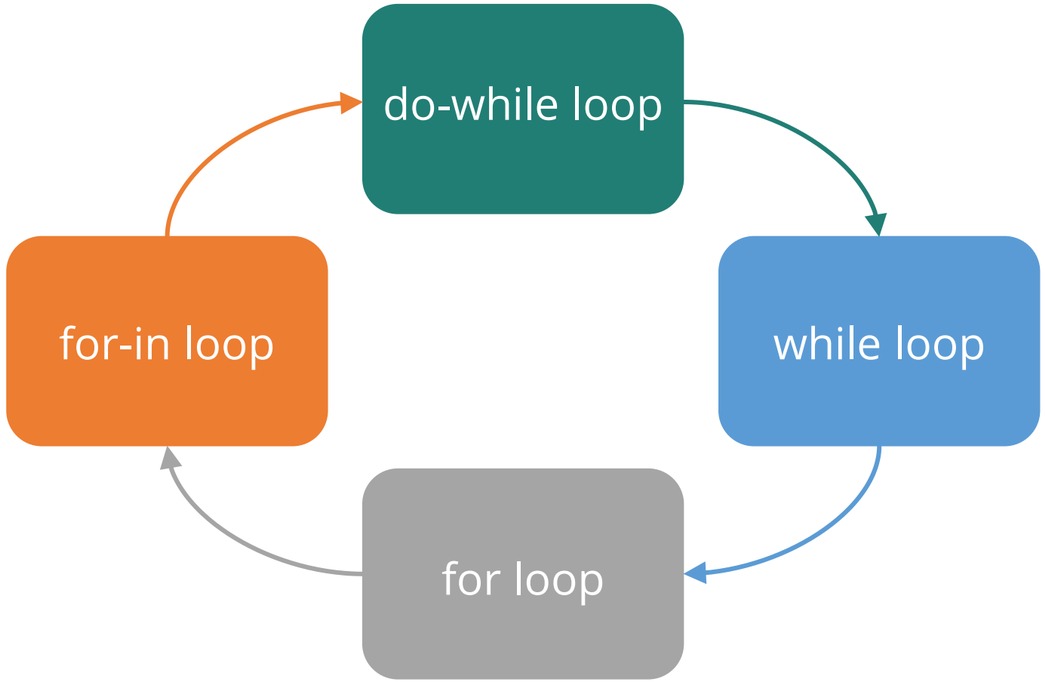
Iterative statements (compound statements) are run zero or more times and are subject to loop-termination criteria.

The types of iterative statements are:

1. While statement
2. Do-While statement
3. For statement

Loop

A loop is a set of instructions that is repeatedly executed until a specific condition is met.



It makes the code compact, and it is mostly used to iterate a collection like an array.

While Loop

The elements are iterated an infinite number of times in the while loop. If the number of iterations is unknown, it should be used.

Example:

```
<script>
var i=11;
while (i<=15)
{
document.write(i + "<br/>");
i++;
}
</script>
```

The difference between **if** and **while statements**:

- **if** executes code if the condition is satisfied.
- **while** keeps repeating itself until the condition is satisfied.

Do-While Loop

The elements are iterated an infinite number of times in the do-while loop. The code is run at least once regardless of whether the condition is true or false.

Example:

```
<script>
var i=21;
do{
document.write(i + "<br/>");
i++;
}
while (i<=25);
</script>
```

For Loop

The elements are iterated a set number of times in a loop.

Example:

```
<script>
for (i=1; i<=5; i++)
{
document.write(i + "<br/>")
}
</script>
```

For-In Loop

The for-in loop is a simple control statement that lets users loop through an object's properties.

Example:

```
<script>
for (let x in location)
{
  text += x + "";
}
</script>
```

For each property of the object, the for-in loop will be executed once.

Break and Continue Statement

Example:

```
for (let i = 0; i < 5; i++) {
  if (i === 3) {
    break; // Exit the loop when i equals 3
  }
  console.log(i);
}
// Output: 0, 1, 2
```

Break statement

- It is used to exit a loop prematurely, terminating the loop's execution even if the loop condition is not fully satisfied.
- It is commonly employed in for, while, and do-while loops.

Example:

```
for (let i = 0; i < 5; i++) {
  if (i === 2) {
    continue; // Skip the current iteration
when i equals 2
  }
  console.log(i);
}
// Output: 0, 1, 3, 4
```

Continue statement

- The continue statement is used to skip the rest of the loop's code block for the current iteration and proceed to the next iteration.
- It is beneficial when specific conditions should be bypassed without exiting the entire loop.

Assisted Practice

Demonstrating Control Flow Statements **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use of control flow concepts and conditional statements in JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Assisted Practice

Demonstrating Form Validation **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the form validation

ASSISTED PRACTICE

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Arrays

Introduction to Arrays

An array is a fundamental data structure in JavaScript used to store a collection of items in a specific order.

- They can contain any type of data, including strings, numbers, objects, and other arrays.
- They provide efficient methods for manipulating and accessing elements.

```
//syntax for creating an Array
const arr = new Array ();
//alternative way
const arr = [];
```

Arrays: Example

The following is an example of creating an array in JavaScript:

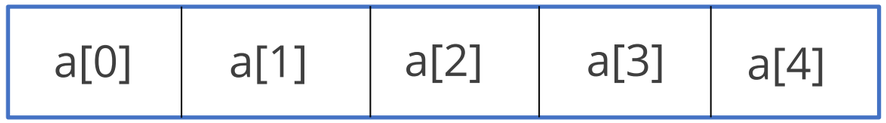
To create a grocery shopping list, use an array to represent each item.

Example:

```
<script>
const groceryList = ["apples", "bananas",
"oranges"];
</script>
```

Arrays: Indexing

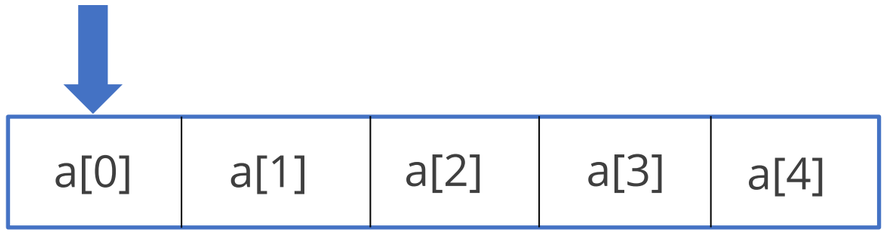
The following diagram shows indexing in an array:



- Arrays are indexed starting from 0, the first element has an index of 0, the second has an index of 1, and so on.
- Elements within an array are accessed using their respective indices.

Array Positioning: Calculating Memory Location

The below diagram depicts the position and memory location of an element in an array:



- Users can calculate the position of each element by adding an offset to a base value.
- The memory location of the first element of the array is generally denoted by the name of the array.
- The base value is index 0, and the difference between the two indexes is the offset.

Properties of Arrays

The following are the built-in properties for effective data manipulation:

1. **length:** Sets or returns the number of elements in an array
2. **constructor**: Returns the function that created the array's prototype
3. **prototype:** Facilitates adding properties and methods to array objects

Arrays: Methods

They provide powerful tools for adding, removing, and manipulating elements in arrays.

The following are some of the methods in an array:

push()

Adds one or more elements to the end of an array

Example:

```
const numbers = [1, 2, 3];
numbers.push(4, 5);
console.log(numbers);
//Output: [1, 2, 3, 4, 5]
```

pop()

Removes the last element from an array

Example:

```
const numbers = [1, 2, 3];
const removedElement = numbers.pop();
console.log(removedElement);
 // Output: 3
console.log(numbers);
// Output: [1, 2]
```

Arrays: Methods

shift()

Removes the first element from an array

Example:

```
const numbers = [1, 2, 3];
const removedElement = numbers.shift();
console.log(removedElement);  // Output: 1
console.log(numbers);  // Output: [2, 3]
```

unshift()

Adds one or more elements to the beginning of an array

Example:

```
const numbers = [2, 3];
numbers.unshift(0, 1);
console.log(numbers);
// Output: [0, 1, 2, 3]
```

Arrays: Methods

concat()

Merges two or more arrays, creating a new array

Example:

```
const array1 = [1, 2];
const array2 = [3, 4];
const newArray = array1.concat(array2);
console.log(newArray);
// Output: [1, 2, 3, 4]
```

join()

Joins all elements of an array into a string

Example:

```
const fruits = ['apple', 'banana', 'orange'];
const joinedString = fruits.join(', ');
console.log(joinedString);
// Output: 'apple, banana, orange'
```

Arrays: Methods

slice()

Selects a part of an array and returns a new array

Example:

```
const numbers = [1, 2, 3, 4, 5];
const slicedArray = numbers.slice(1, 4);
console.log(slicedArray);
// Output: [2, 3, 4]
```

splice()

Adds or removes elements from an array at a specified index

Example:

```
const numbers = [1, 2, 3, 4, 5];
numbers.splice(2, 1, 6);
console.log(numbers);
// Output: [1, 2, 6, 4, 5]
```

Arrays: Iterator Methods

They enable the traversal and processing of array elements efficiently.

forEach()

Executes a provided function once for each array element

Example:

```
const numbers = [1, 2, 3];
numbers.forEach(num => {
  // Process each element using num
  console.log(num);
});
// Output:
// 1
// 2
// 3
```

filter()

Creates a new array with elements that satisfy a condition

Example:

```
const numbersFilter = [1, 2, 3, 4, 5];
const evenNumbers =
numbersFilter.filter(num => num % 2 === 0);
console.log(evenNumbers);
// Output:
// [2, 4]
```

Arrays: Iterator Methods

map()

Creates a new array by applying a function to each element

Example:

```
const numbersFilter = [1, 2, 3, 4, 5];
const evenNumbers =
numbersFilter.filter(num => num % 2 ===
0);
console.log(evenNumbers);
// Output:
// [2, 4]
```

reduce()

Reduces an array to a single value through a provided function

Example:

```
const numbersReduce = [1, 2, 3, 4, 5];
const sum = numbersReduce.reduce((acc,
num) => acc + num, 0);
console.log(sum);
// Output:
// 15
```

Assisted Practice

Demonstrating Array Methods **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use of array methods in JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Multidimensional Array

It is an array that holds other arrays, providing a way to organize data in a hierarchical structure.

Example:

```
const gameBoard = [[0, 1, 2], [3, 4, 5], [6, 7, 8]];
```

To create a game board, utilize a multidimensional array to represent the board, where each element in the array signifies a square on the board.

Arrays: Sorting

It refers to the process of arranging the elements of an array in a specified order, either ascending or descending.

It helps in organizing the data for easier access and analysis.

Arrays: Sorting

The following is the syntax for the sort() method:

```
arr.sort(compareFunction)
```

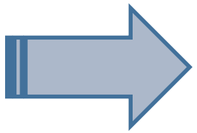
CompareFunction (optional) defines a custom sort order.

Arrays: Sorting

The sort() method sorts the items of an array in a specific order.

Example: sort() method

```
let city = ['Ludhiana', 'Chandigarh', 'Jalandhar',
'Patiala'];

// sort the city array in ascending order

let sortArray = city.sort();
console.log(sortArray);
```



Output

```
['Chandigarh', 'Jalandhar',
'Ludhiana', 'Patiala']
```

Arrays: Searching

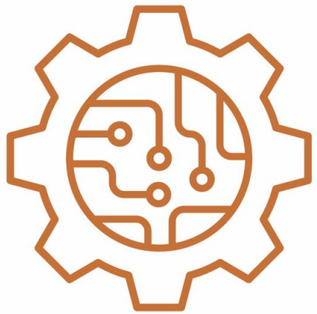
It is a process of finding the position or existence of specific elements within an array.

It helps to locate and retrieve elements based on specific criteria in an array.

Arrays: Searching

The following are the common methods for array searching:

- filter()
- find()



- includes()
- indexOf()

Filter Method

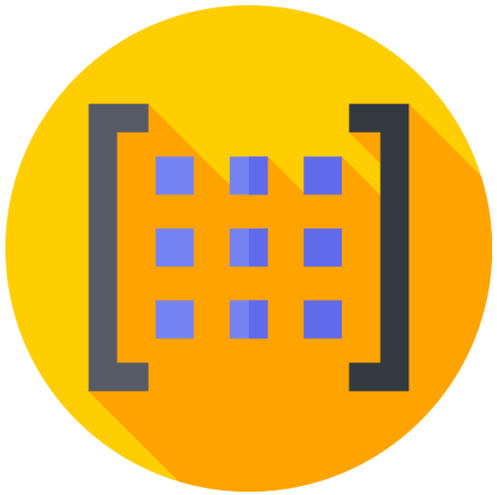
The filter() method returns a new array containing all elements that pass a test provided by a function.

```
//syntax:
array.filter(callback (currentValue, index, arr), thisValue)
```

It does not execute the function for empty elements or change the original array.

Filter Method

The parameters are:



callback()
A function to run for each array element (required)

currentValue
The value of the current element (required)

Index
The index of the current element index (optional)

arr
The array of the current element (optional)

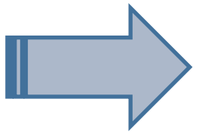
thisValue
The value passed to the function as this value (optional)

Filter Method: Example

It is useful for obtaining a subset of an array that satisfies a particular criterion and returns an array rather than a single element.

filter() method

```
const array = [7, 2, 13, 10, 5];

const greaterThanFive = array.filter(element => element > 5);

console.log(greaterThanFive)
```



Output

```
[ 7, 13, 10 ]
```

Find Method

The find () method is used to find the element in an array.

This function returns the value of the first element that passes a test.

The function for empty elements is not executed by this technique.

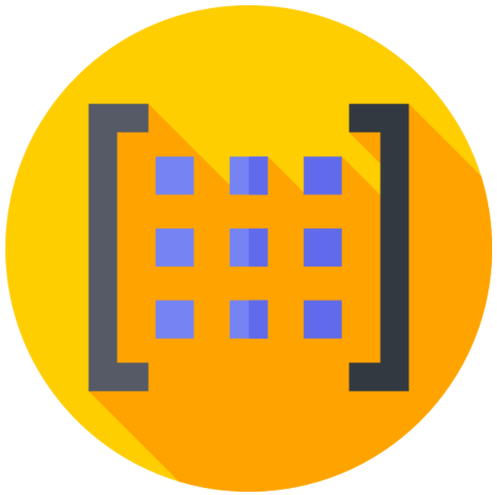
Find Method

It provides flexibility by allowing the use of a custom condition through the callback function. It returns the actual element, not just a boolean value.

```
//syntax:
array.find(callback (currentValue,index,arr), thisValue)
```

Find Method

The parameters are:



callback()
A function to run for each array element (required)

currentValue
The value of the current element (required)

Index
The index of the current element index (optional)

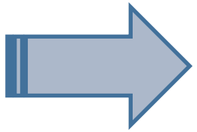
arr
The array of the current element (optional)

thisValue
The value passed to the function as this value (optional)

Find Method: Example

find() method

```
const array = [9, 12, 13, 20, 7];

const greaterThanNum = array.find(element => element > 9);

console.log(greaterThanNum)
```

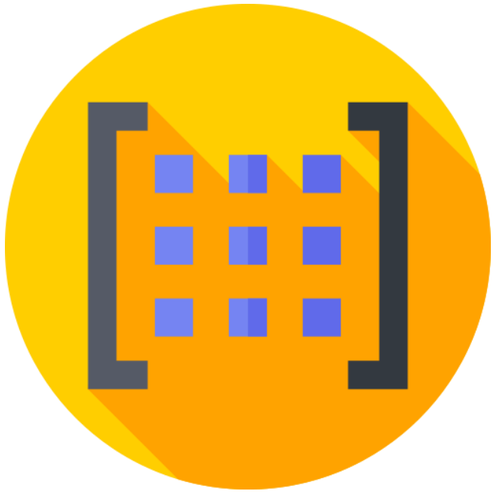


Output

```
12
```

Includes Method

This method provides a concise way to check if an array contains a particular element.



```
//syntax:
array.includes(searchElement, startIndex)
```

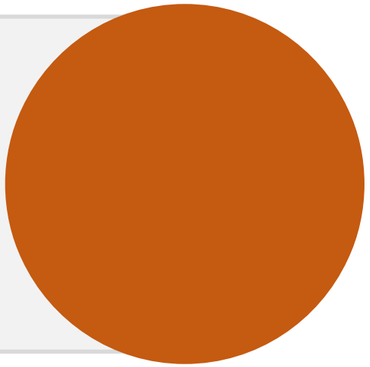
It returns a boolean value (true or false) indicating whether the element is present in the array.

Includes Method

The parameters are:

searchElement:
The element to search for within the array

startIndex:
The index to start the search from (optional)



The primary difference between this and other methods is that it accepts a value rather than a callback as the argument.

Includes Method: Example

includes() method

```
const array = [9, 12, 13, 20, 7];

const includesNum = array.includes(13);

console.log(includesNum)
```

Output

```
true
```

IndexOf Method

It is a simple and straightforward method for finding the index of a specific element in an array.

```
//syntax:
array.indexOf(searchElement, startIndex)
```

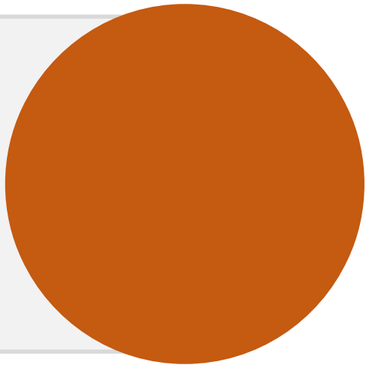
It returns the first occurrence of the element and supports an optional parameter to specify the starting index for the search.

IndexOf Method

The parameters are:

searchElement:
The element to search for within the array

startIndex:
The index to start the search from (optional)

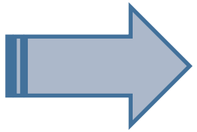


Both the includes() and indexOf() methods employ strict equality (===) when searching the array. If the values differ in type, such as '4' and 4, they will respectively return false and -1.

IndexOf Method: Example

indexOf() method

```
const array = [9, 12, 13, 20, 7];

const includesNum = array.indexOf(20);

console.log(includesNum)
```



Output

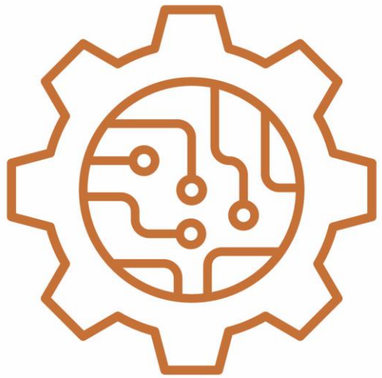
```
3
```

Searching Methods: Summary

Here is a summary of when to use each method:

Use filter() to find all items in an array that meet a specific condition

Use find() to check if at least one item meets a specific condition



Use includes() to check if an array contains a particular value

Use indexOf() to find the index of a particular item in an array

Assisted Practice

Demonstrating Advance Array Operations **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use of advance array operations in JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Spread Syntax (...)

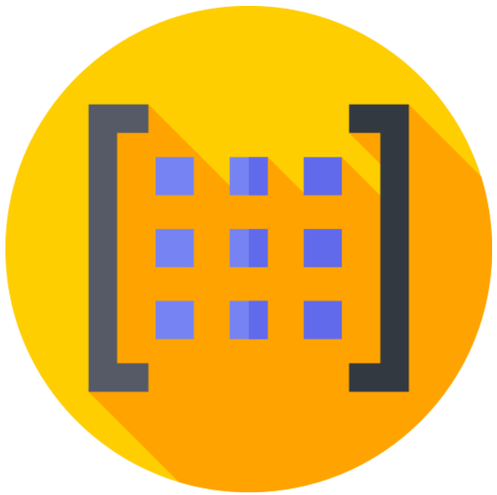
It is a syntactic feature introduced in ECMAScript 6 (ES6) that allows an iterable (like an array or a string) to be expanded into individual elements or characters.


```
// Syntax:
const newArray = [...iterable];
```

The spread operator is versatile and can be used in various contexts, such as array manipulation, function arguments, and object merging.

Spread Syntax: Use cases

The parameters are:



Copying or cloning
It is commonly used to create shallow copies of arrays or objects.

Array concatenation
It facilitates the merging or concatenation of arrays, enabling the combination of multiple arrays into a single array.

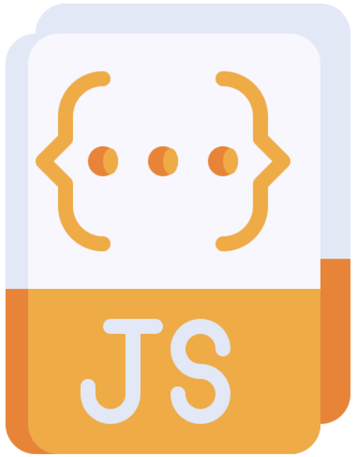
Function arguments
It can be used to pass elements of an array as individual arguments to a function.

Object merging
It allows the merging of objects or arrays into a new object or array.

String to array conversion
It can convert a string into an array of characters, treating each character as a separate element.

Arrays Mutability

In JavaScript, arrays are mutable, so their elements can be changed or modified after the array is created.

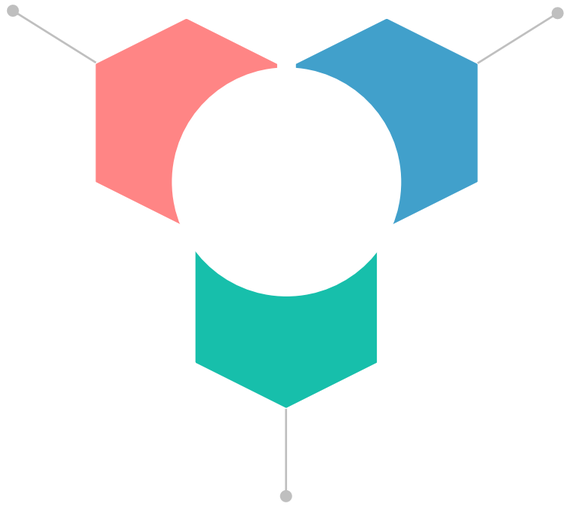


This mutability allows for dynamic modifications to the array's content, such as adding or removing elements, changing values at specific indices, and altering the array's length.

Arrays Mutability

The following are the key points involved in the mutability of the arrays:

Modifying the elements



Adding and removing the elements

Changing array length

Modifying the Elements

Individual elements within a mutable array can change using their index.

Example:

```
let numbers = [1, 2, 3];
numbers[1] = 10;

// Modifying the second element

console.log(numbers);

// Output: [1, 10, 3]
```

Adding and Removing the Elements

Methods such as *push()*, *pop()*, *shift()*, and *unshift()* can add or remove elements, dynamically altering the array's length.

Example:

```
let fruits = ['apple', 'banana'];
fruits.push('orange’);

// Adding an element

fruits.pop();

// Removing the last element

console.log(fruits);

// Output: ['apple', 'banana']
```

Changing Array Length

The length property of an array can be directly modified to change the array's size.

Example:

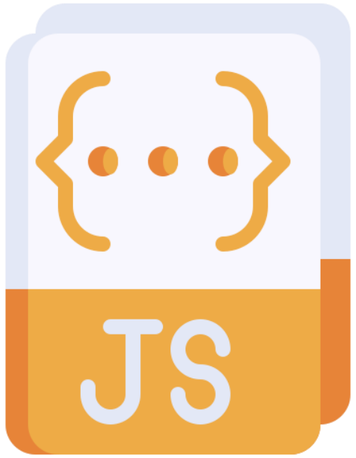
```
let colors = ['red', 'green', 'blue'];
colors.length = 2;

// Changing the array length

console.log(colors);

// Output: ['red', 'green']
```

Array Mutability: Benefits

The following are a few benefits of mutability in JavaScript arrays:

Arrays Immutability

It refers to the inability of an object to be modified after it is created.

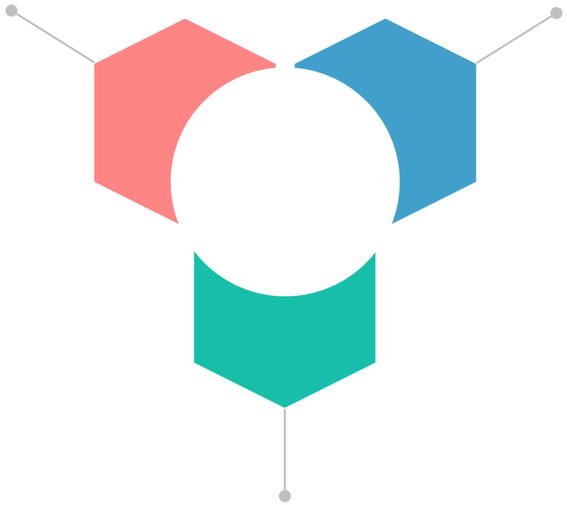


JavaScript arrays are mutable; immutability can be achieved by creating new arrays or using methods that return new arrays rather than modifying the existing ones.

Arrays Immutability

The following are the key points involved in the immutability of the arrays:

Creating new arrays



Array methods returning new arrays

Avoiding direct mutations

Creating New Arrays

Instead of modifying an existing array, immutability involves creating a new array with the desired changes.

Example:

```
let originalArray = [1, 2, 3];
let newArray = [...originalArray, 4];

// Creating a new array with an additional element

console.log(originalArray);

// Output: [1, 2, 3]

console.log(newArray);

// Output: [1, 2, 3, 4]
```

Array Methods Returning New Arrays

Methods like map(), filter(), and concat() return new arrays without modifying the original array.

Example:

```
let numbers = [1, 2, 3];

let doubledNumbers = numbers.map(num => num * 2);

// Creating a new array with doubled values

console.log(numbers);

// Output: [1, 2, 3]

console.log(doubledNumbers);

// Output: [2, 4, 6]
```

Avoiding Direct Mutations

In the following example, the push() method is used directly on mutatedArray, which also modifies originalArray because they reference the same array:

Example:

```
// Original array
const originalArray = [1, 2, 3, 4, 5];

// Direct mutation (avoided for immutability)
const mutatedArray = originalArray;
mutatedArray.push(6);

console.log(originalArray);
// Output: [1, 2, 3, 4, 5, 6]

console.log(mutatedArray);
// Output: [1, 2, 3, 4, 5, 6] (originalArray is also modified)
```

Avoiding Direct Mutations

Users can achieve immutability by creating a new array.

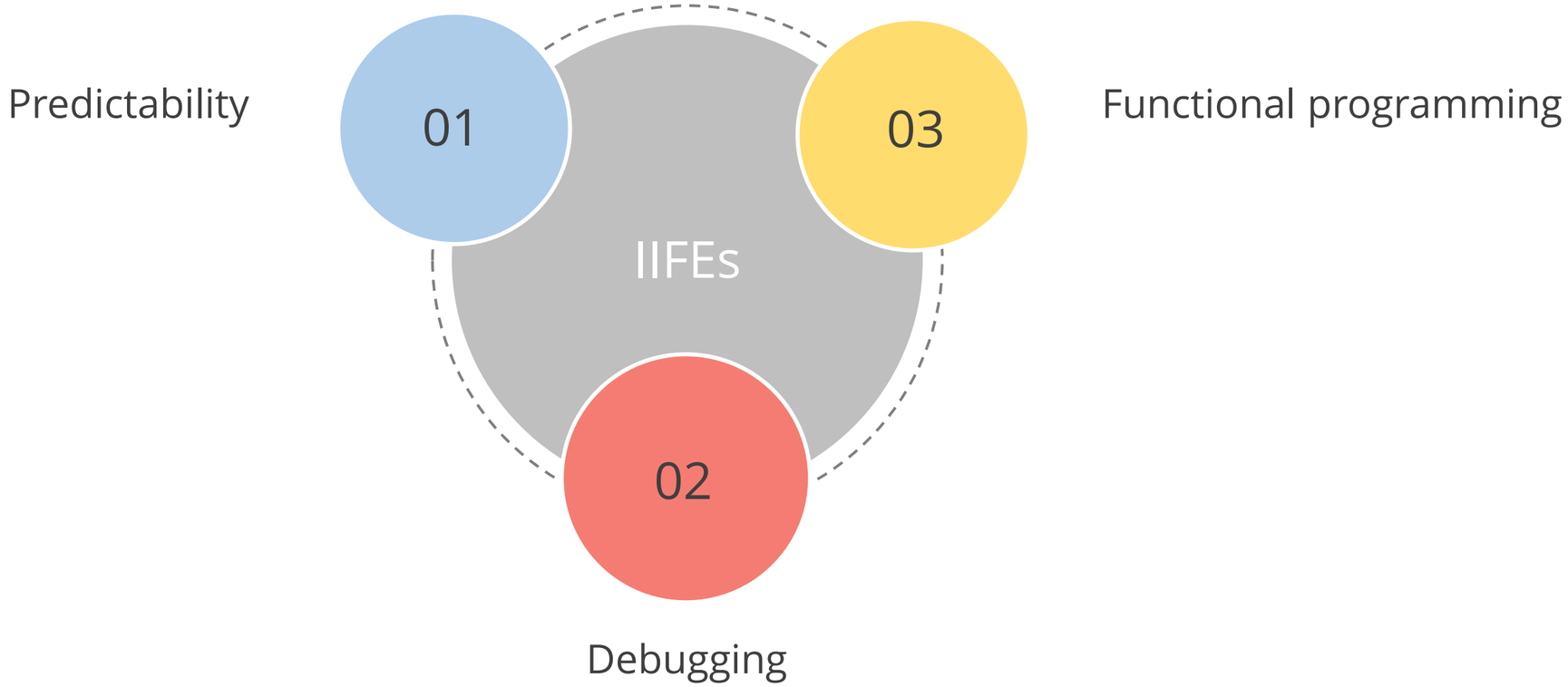
Example:

```
// Original array
const originalArray = [1, 2, 3, 4, 5];

// Creating a new array without direct mutation
const newArray = [...originalArray, 6];

console.log(originalArray);
// Output: [1, 2, 3, 4, 5]

console.log(newArray);
// Output: [1, 2, 3, 4, 5, 6] (originalArray remains unchanged)
```

The spread operator (...) creates a new array (newArray) that includes all elements from the originalArray along with the new element (6). This ensures immutability, as the originalArray is not modified directly.

Array Immutability: Benefits

The following are some benefits of immutability in JavaScript arrays:



Array Destructuring

It is a powerful feature introduced in ECMAScript 6 (ES6) that allows users to extract values from arrays and assign them to variables in a concise and expressive manner.

This feature simplifies the process of working with arrays and enhances code readability.

Array Destructuring

The following are some ways of destructuring an array:

- Basic syntax
- Skipping elements
- Default values
- Swapping variables

Array Destructuring

The following are some ways of destructuring an array:

- Rest operator
- Nested destructuring
- Ignoring the rest
- Function parameter destructuring

Assisted Practice

Demonstrating Array Mutability, Immutability, and Advanced Destructuring **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use array mutability, immutability, and destructuring in JavaScript.

Assisted Practice: Guidelines

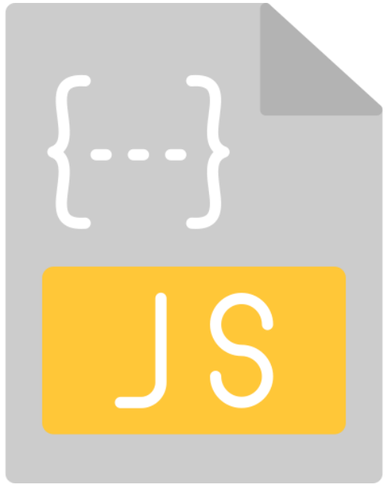
Steps to be followed:

1. Create and execute JS file

Introduction to Strings

What Are Strings?

A string is a data type representing a sequence of characters, encompassing letters, numbers, symbols, and whitespace.

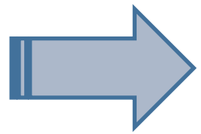


Strings can be created by enclosing a sequence of characters within either single quotes (' '), double quotes (" "), or template literals (` `).

Strings: Example

To create strings in JavaScript, users can use straightforward syntax with either single or double quotes or template literals, as shown below:

```
// String with single quotes
let singleQuotesString = 'Hello, World!';

// String with double quotes
let doubleQuotesString = "Greetings, Universe!";

// String with template literal
let templateLiteralString = `The result of 2 + 2 is ${2 + 2}.`;
```



Output

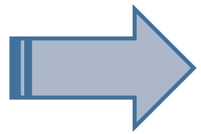
```
Hello, World!
Greetings, Universe!
The result of 2 + 2 is 4.
```

Template literals allow dynamic string creation by embedding expressions within the string, making them useful for constructing strings with variable content.

String Object

In JavaScript, strings are instances of the String object, and the creation of a new instance is achieved using the **new** keyword, as demonstrated below:

```
// String with single quotes

let stringObject = new String('String Object');

console.log(stringObject);
```



Output

```
[String: 'String Object']
```

These strings have properties and methods that can be accessed and used.

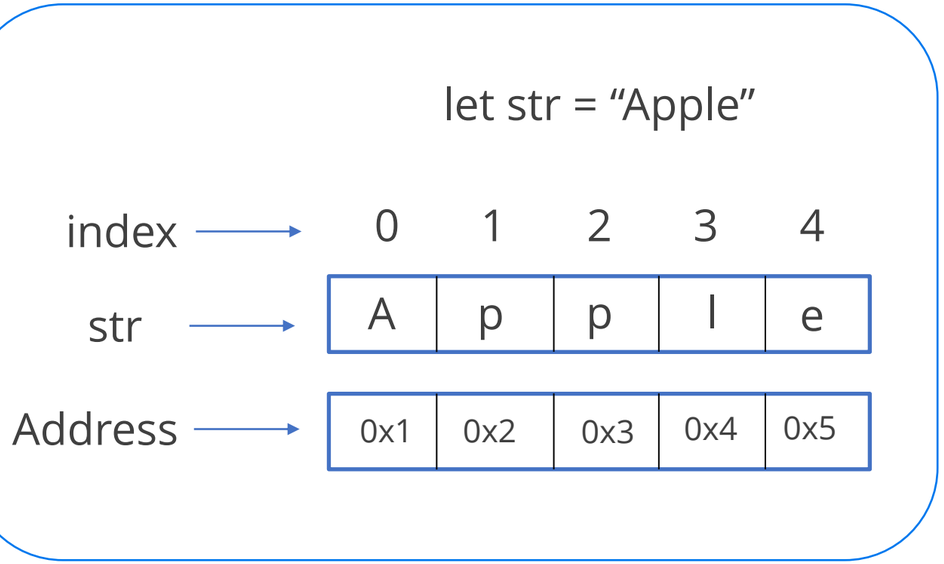
String Object: Limitations

The following are some limitations and considerations associated with using string objects:

1	Immutability	String objects cannot be changed after creation.
2	Automatic conversion	Automatic conversions between string objects and primitives can occur.
3	Performance	Efficient string handling and processing are crucial for optimal performance.
4	Complexity	String objects may introduce complexity in certain scenarios.

Strings: Indexing

The following diagram shows the indexing of a string:



- Each character is associated with a specific index.
- The first character **‘A'** is at index 0, the second character **‘p'** is at index 1, and so on.
- The address refers to the memory location where individual characters within a string, like **'A'**, **'p'**, **'p'**, **'l’**, and **'e'**, are stored.

Properties of Strings

The properties for effective data manipulation and type identification are:

1. **length:** Returns the number of characters in a string
2. **constructor**: Returns a reference to the constructor function that created the string object
3. **prototype:** Allows adding properties and methods to an object

Strings: Methods

A string object provides a rich set of methods for manipulating and extracting information from strings.

Method	Description
charAt()	It provides the char value present at the specified index.
charCode()	It provides the Unicode value of a character present at the specified index.
concat()	It provides the combination of two or more strings.
indexOf()	It provides the position of a char value present in the given string.
lastIndexOf()	It provides the position of a char value present in the given string by searching for a character from the last position.
search()	It searches for a specified regular expression in each string and returns its position if the match occurs in it.

Strings: Methods

Method	Description
match()	It searches for a specified regular expression in each string and returns the regular expression if a match occurs.
replace()	It replaces a given string with the specified replacement.
substr()	It is used to fetch the part of the given string based on the specified starting position and the length.
substring()	It is used to fetch a part of the given string based on the specified index.
slice()	It is used to fetch the part of the given string. It allows you to assign a positive as well as a negative index.
toLowerCase()	It changes the uppercase letter in the given string to a lowercase letter.

Strings: Methods

Method	Description
toLocaleLowerCase()	It changes the uppercase letter in the given string to a lowercase letter based on the host's current locale.
toUpperCase()	It converts the given string into an uppercase letter.
toLocaleUpperCase()	It converts the given string into an uppercase letter based on the host's current locale.
toString()	It provides a string representing a particular object.
valueOf()	It provides the primitive value of the string object.
split()	It splits a string into a substring array and returns the newly created array.
trim()	It trims the white space from the left and right-side string.

Assisted Practice

Demonstrating String Methods **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use string methods in JavaScript.

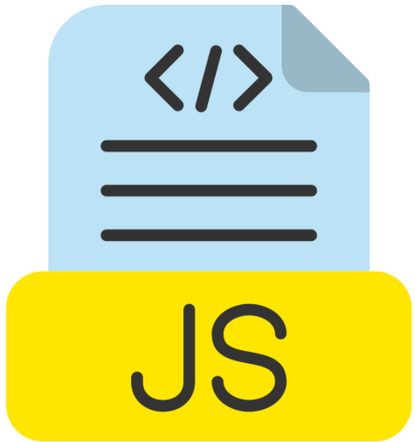
Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

String Interpolation

It serves as a powerful technique in JavaScript, enabling developers to seamlessly embed expressions or variables within strings.

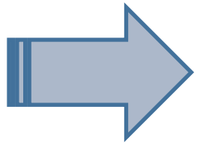


This facilitates the dynamic construction of strings, enhancing code readability and flexibility.

String Interpolation

The introduction of template literals in ECMAScript 2015 (ES6) has revolutionized string interpolation, offering a more elegant and convenient alternative to traditional string concatenation methods.

```
// String Interpolation with Template Literals
let name = "John";
let age = 30;

// Template Literal
let introduction = `Hello, my name is ${name} and I am ${age}
years old.`;

// Output
console.log(introduction);
```



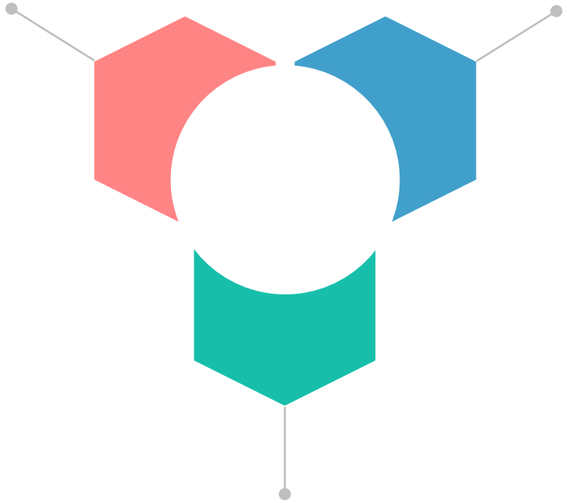
Output

```
Hello, my name is John and
I am 30 years old.
```

Template literals provide a syntax that allows expressions to be embedded directly within the string, and this feature simplifies the process of creating complex strings.

String Interpolation: Benefits

Readability



Elegance

Flexibility

Escape Charecters

In JavaScript, strings often contain special characters such as single quotes, double quotes, and line breaks.

Users can utilize escape sequences to represent these characters within strings, which are a combination of a backslash (\) followed by a specific character.

Escape Charecters

The following are some common escape characters:

Single quote (\')

Represents a single quote within a string

Double quote (\")

Represents a double quote within a string

Backslash (\\)

Represents a literal backslash within a string

New line (\n)

Represents a line break within a string

Carriage return (\r)

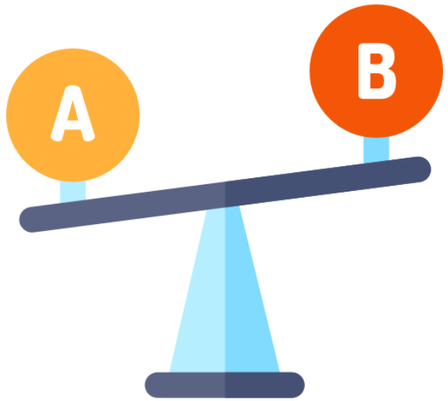
Represents a carriage return within a string

Tab (\t)

Represents a tab character within a string

String Comparison

In JavaScript, comparing strings is a common and essential operation involving tasks such as checking for equality or sorting.

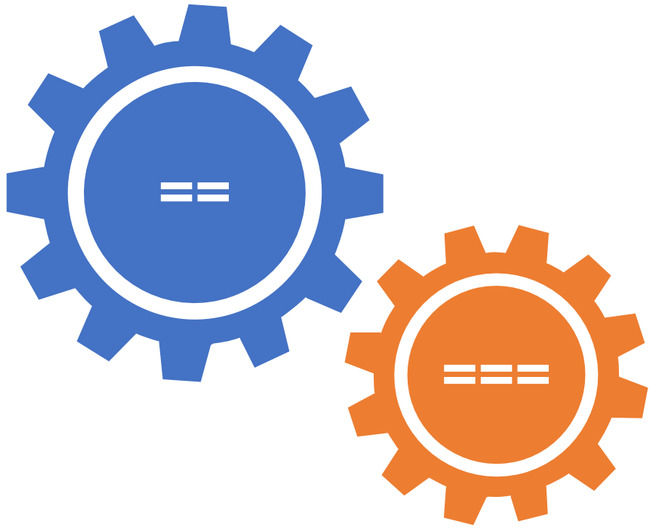


Two primary methods for string comparison include using **equality operators** (== and ===) and the **localeCompare** method.

Equality Operators

The following are the equality operators:

Loose equality:
The loose equality operator compares strings without considering the data types, attempting conversion if necessary.

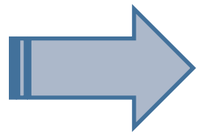


Strict equality:
The strict equality operator compares both value and data type, ensuring that operands are of the same type.

LocaleCompare Method

It offers a detailed way to compare strings, especially in internationalization and sorting.

```
let str1 = "apple";
let str2 = "banana";

let comparisonResult = str1.localeCompare(str2);

console.log(comparisonResult);
```



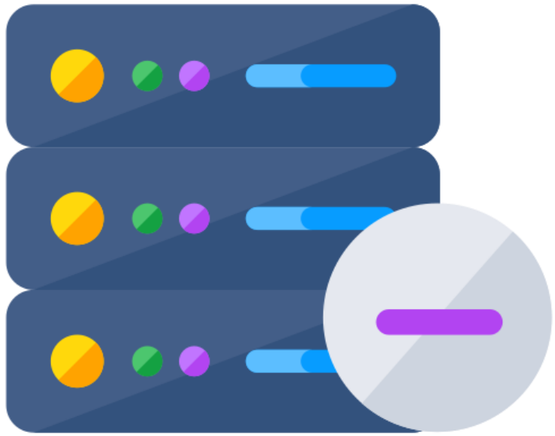
Output

```
-1
```

In the above example, the localeCompare method returns -1, indicating that "apple" comes before "banana" in the sorting order.

String Manipulation

It is a fundamental skill that empowers developers to transform, modify, and optimize textual data.



Various techniques, ranging from changing cases to splitting strings, play a crucial role in enhancing the flexibility and functionality of the code.

String Manipulation

The following are some of the most commonly used techniques for string manipulation:

- Changing case
- Trimming whitespace
- Replacing text
- Splitting strings into arrays

String Iteration

To explore and manipulate characters within a string in JavaScript, users can effortlessly use traditional for loops and dedicated string methods.

String Iteration

The following are a few different ways to iterate through characters in a string in JavaScript:

1. Using traditional for loops
2. Leveraging the charAt method
3. Reversed string construction

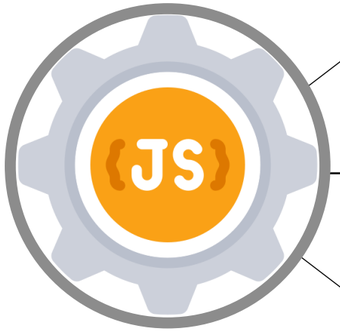
Unicode and Characters

In JavaScript, Unicode is like a universal language for characters, covering everything from letters to symbols.

JavaScript naturally supports Unicode, using unique codes for each character. This means users can easily include characters from various scripts.

Unicode and Characters

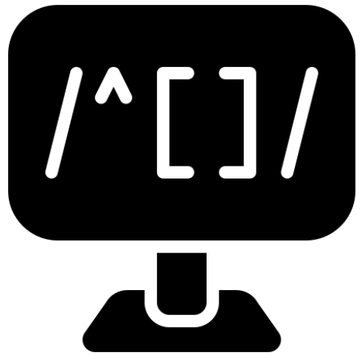
The following primarily focuses on explaining key concepts and use cases related to Unicode in JavaScript strings:



- Check Unicode code points
- Deal with different characters
- Perform simple string manipulation with Unicode

Regular Expressions

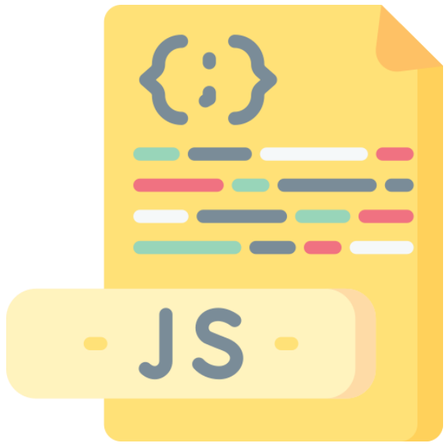
They are also known as Regex, a powerful tool in programming that helps efficiently navigate and manipulate strings by finding specific patterns.



Regex is a character sequence defining a search pattern, enabling the specification of rules for matching strings, from single characters to intricate combinations.

RegExp Object

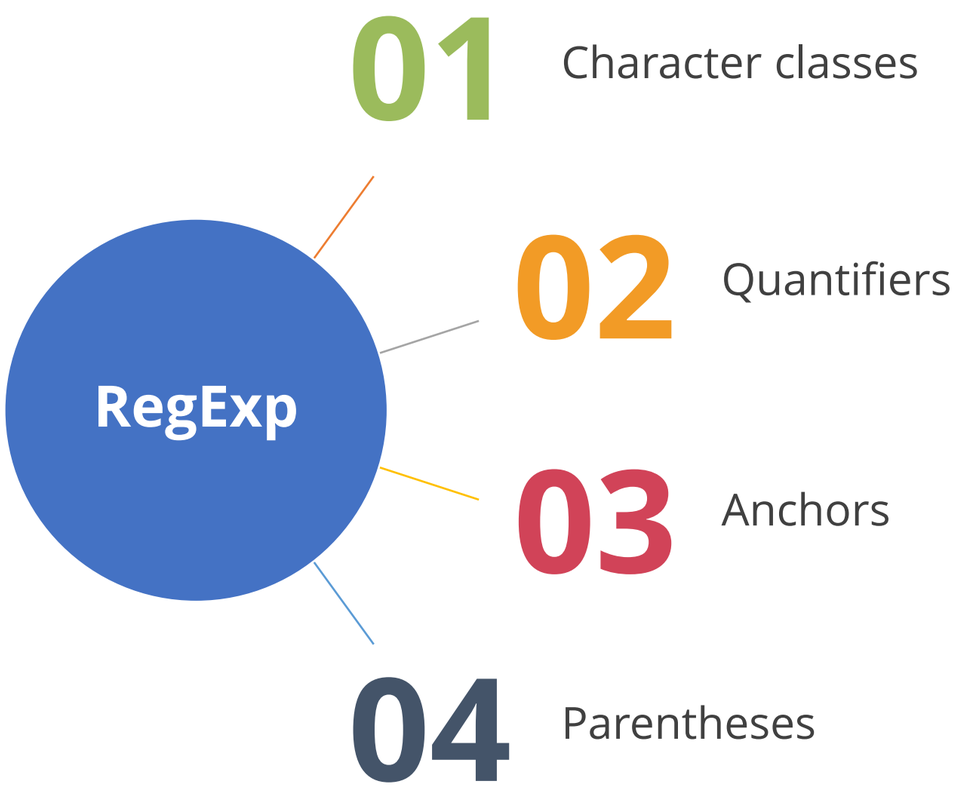
In JavaScript, the primary tool for working with regular expressions is the RegExp object.



This object provides methods for matching text with a pattern, searching for patterns within strings, and replacing matched patterns with new text.

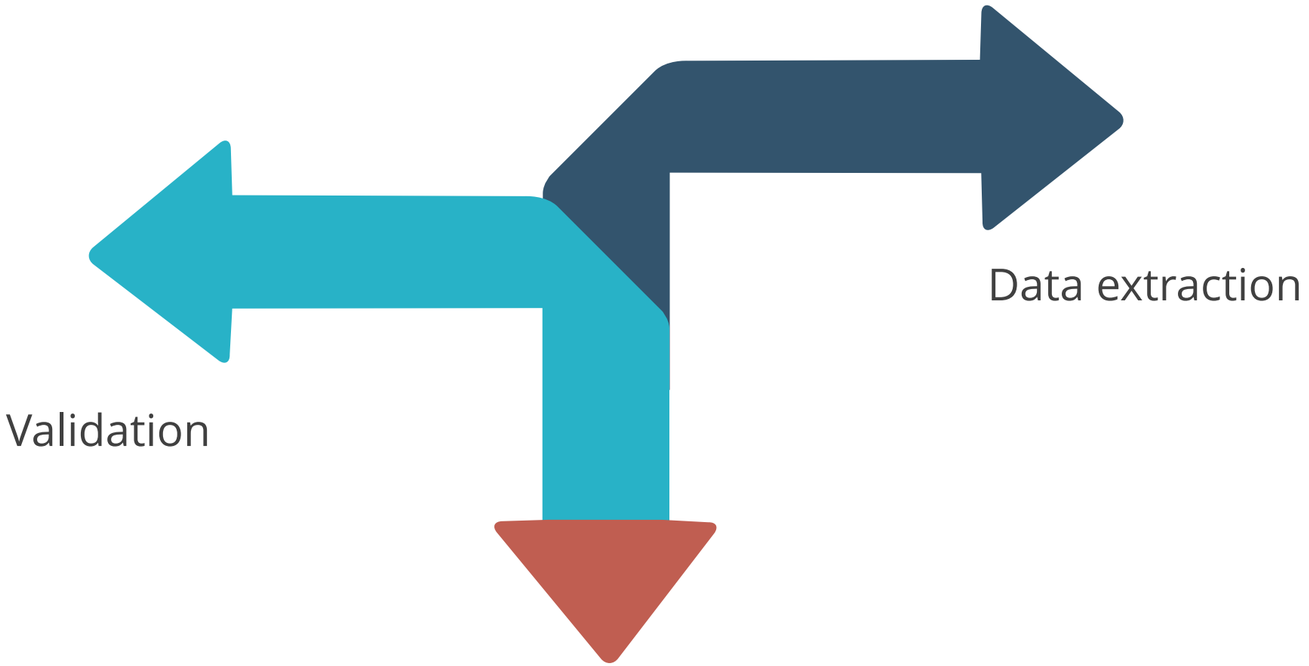
Regular Expressions: Key Concepts

The following are the key concepts of Regex:



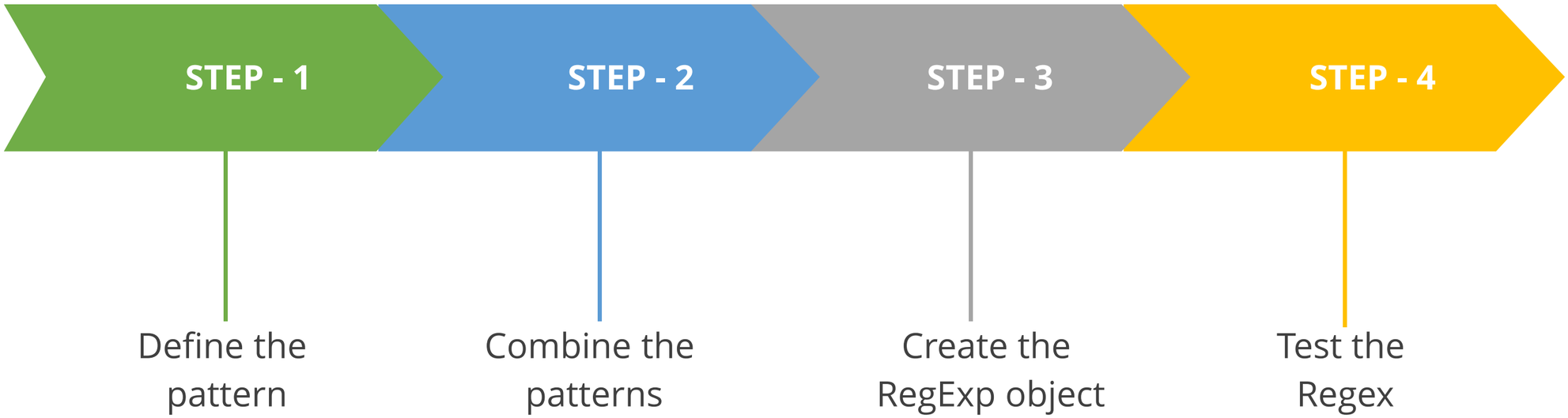
Regular Expressions: Practical Applications

The following are the practical applications of Regex:



Regular Expressions: Example

To create a regular expression for validating email addresses in JavaScript, define a pattern that matches strings. The following are the steps involved:



Assisted Practice

Demonstrating Advance String Operations **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use advance string operations in JavaScript.

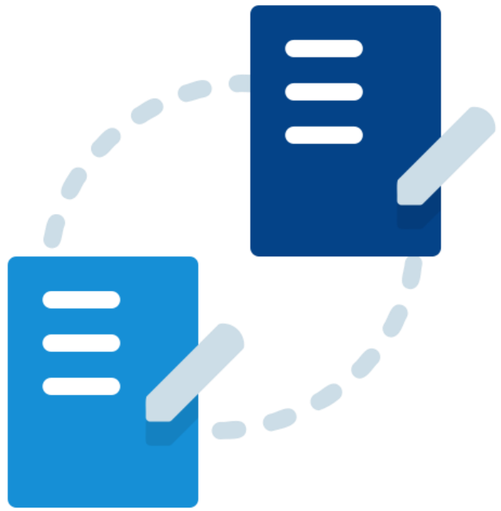
Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

String Conversions

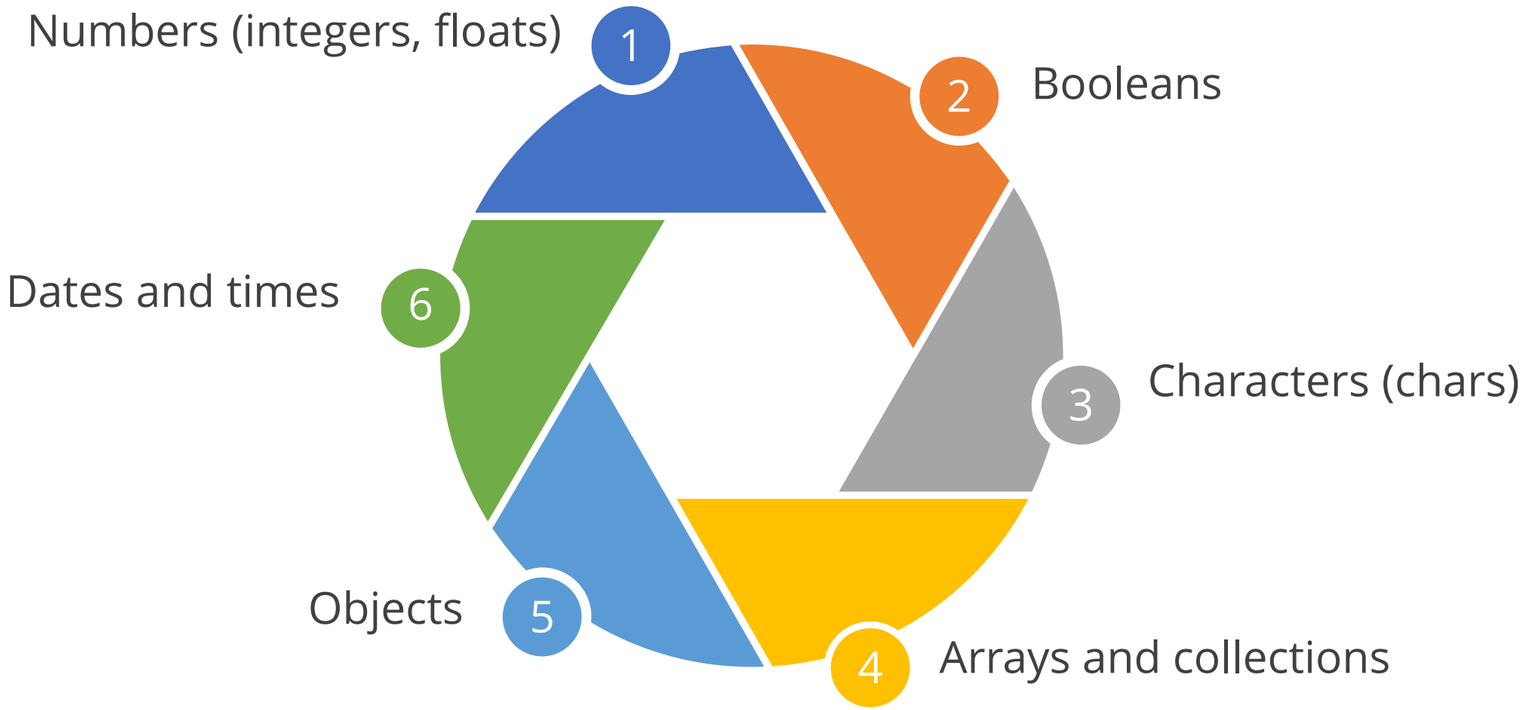
In programming, managing diverse data types is common. Essential tools include string conversions for transforming data to and from string format.



These conversions involve turning numbers and Booleans into strings and parsing strings back into their original forms, playing a pivotal role in managing data from various sources.

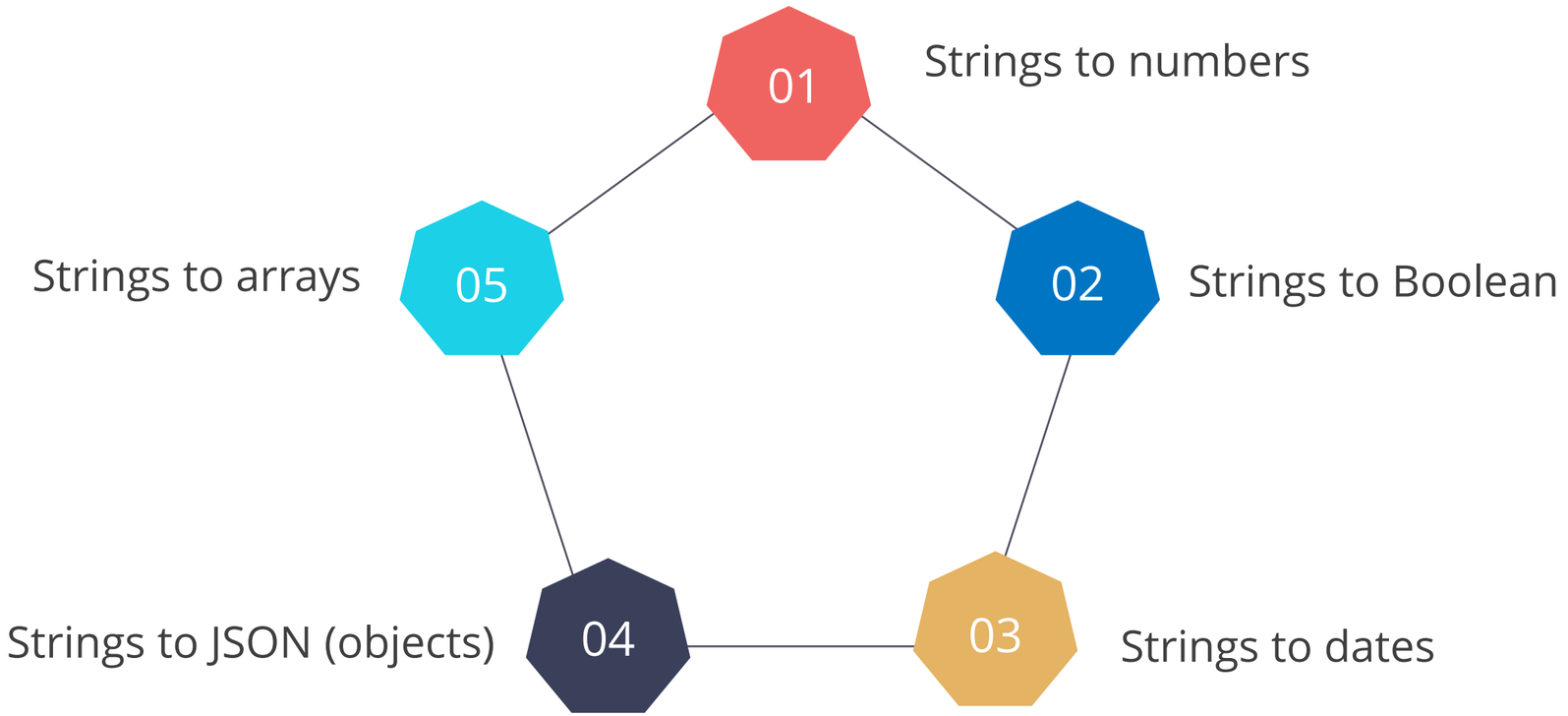
String Conversions: Data Types to String

The following are the data types that can be converted into strings:



Parsing Strings to Other Data Types

It involves converting string representations of data back into their original data types. Here are some common methods to perform this in JavaScript:

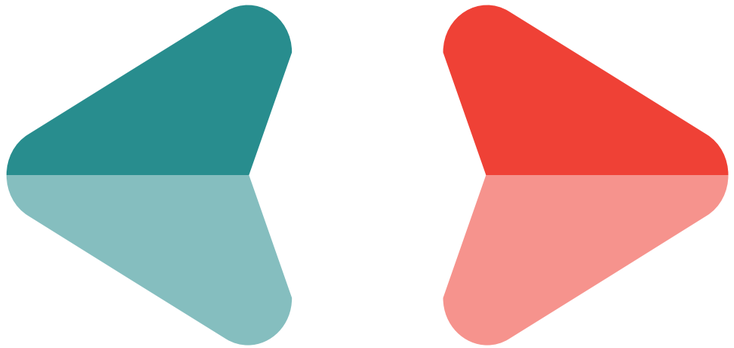


String Conversions: Common Challenges

The following are some of the common challenges involved in string conversion:

Error handling

Anticipate and handle potential conversion errors in robust programs



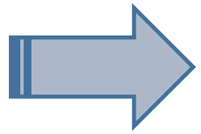
Locale considerations

Consider locale-specific conventions when parsing strings to numbers, crucial for internationalized applications

String Immutability

It refers to the unalterable nature of strings; once a string is declared in JavaScript, it is not possible to change its characters directly.

```
// String immutability example

let greeting = "Hello";

greeting[0] = "J";

// This will not modify the string

console.log(greeting);
```



Output

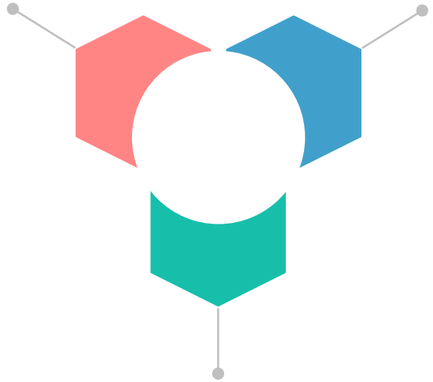
```
Hello
```

In the above example, attempting to change the first character of the greeting string has no effect, and the original string remains unchanged.

String Immutability

Due to the immutability of strings, modifying their content involves creating new strings rather than altering existing ones. The following are some methods to perform this task:

Concatenation



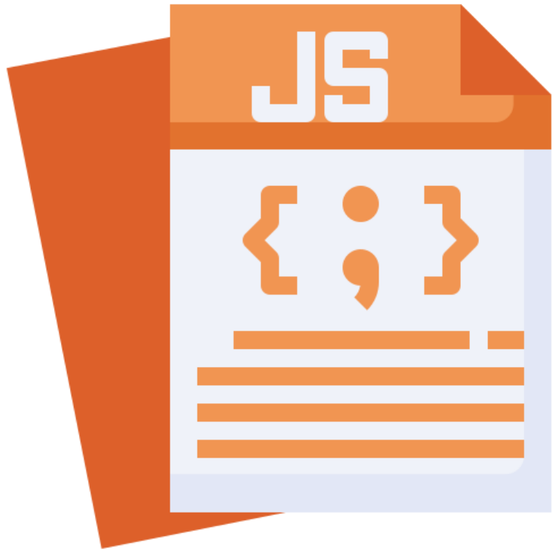
String methods

Template literals

Some of the benefits of string immutability include predictable behavior, reference stability, and support for functional programming.

String Formatting

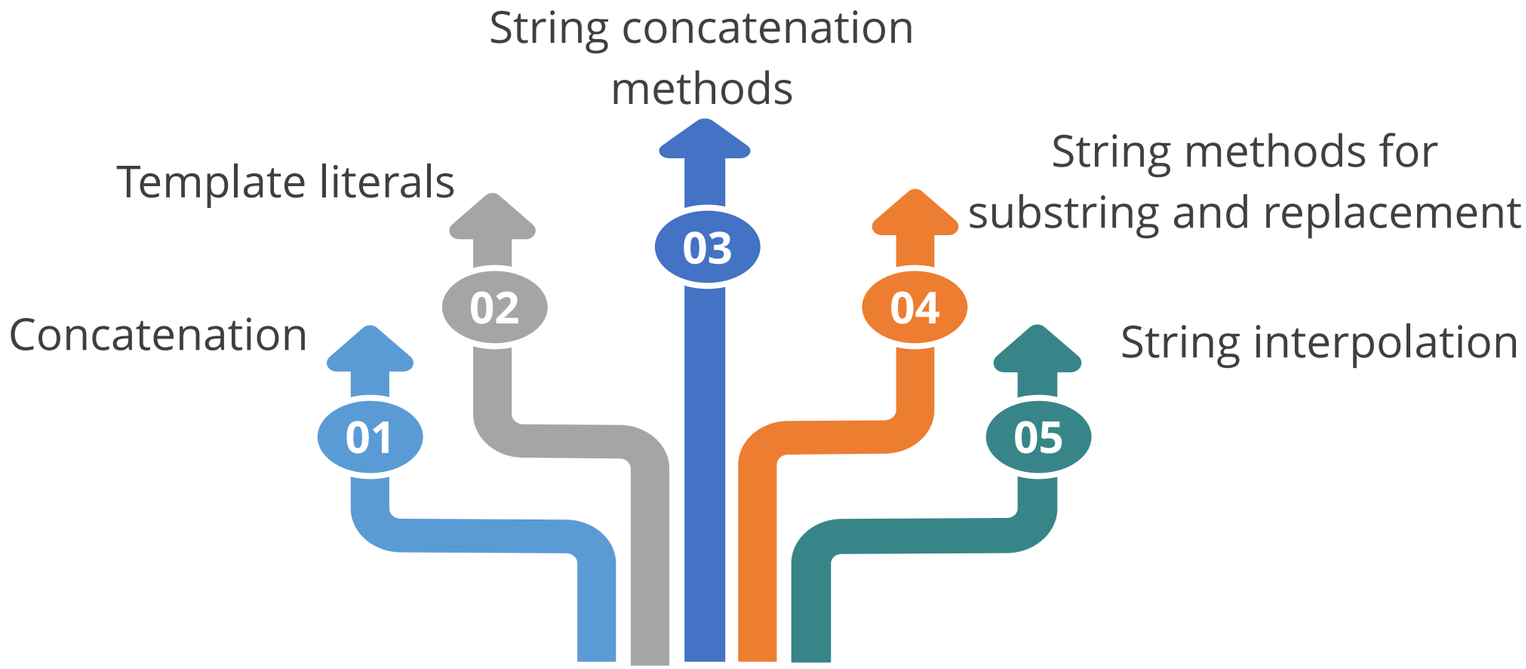
It involves manipulating and arranging text in a way that enhances readability and presentation in programming languages like JavaScript.



To enhance the clarity and usability of your applications, it is essential to understand and implement proper string formatting techniques for dates, numbers, or custom text.

String Formatting

The following are some methods and techniques for formatting strings:



These methods provide options for string formatting, allowing developers to choose the best approach for UI presentation, data processing, or other use cases.

String Encoding and Decoding

It ensures the secure and reliable transfer of information.

The following are the two ways of string encoding and decoding:

URL encoding and decoding

It transforms special characters in a string into a URL-safe format, reverting it back to its original form

Base64 encoding and decoding

It converts binary data to a plain text ASCII format, reverting it to its original binary form.

URL Encoding and Decoding

URL encoding prevents special characters in data from disrupting the URL structure by converting them into a safe format, often done using the encodeURIComponent function in JavaScript.

URL Encoding

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

let encodedString =
encodeURIComponent(originalString);

console.log(encodedString);

// Output: "Hello%2C%20World%21"
```

URL Decoding

```
let decodedString =
decodeURIComponent(encodedString);

console.log(decodedString);

// Output: "Hello, World!"
```

Upon receiving the encoded string, URL decoding is necessary to revert it to its original form using the decodeURIComponent function.

Base64 Encoding and Decoding

Base64 encoding is commonly employed for encoding binary data, such as images or files, into ASCII text format. In JavaScript, the btoa function is used for Base64 encoding.

Base64 Encoding

```
let binaryData = "This is binary data.";

let base64Encoded = btoa(binaryData);

console.log(base64Encoded);

// Output: "VGhpcyBpcyBiaW5hcnkgZGF0YSE="
```

Base64 Decoding

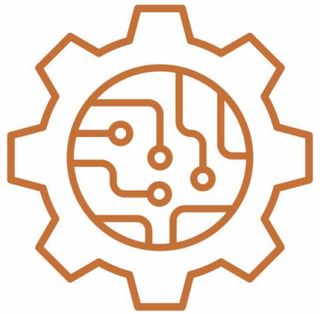
```
let decodedBinaryData = atob(base64Encoded);

console.log(decodedBinaryData);

// Output: "This is binary data."
```

To retrieve the original binary data, Base64 decoding is performed using the atob function.

String Encoding and Decoding: Significance

The following are the significance of string encoding and decoding:

- Security in data transmission
- Compatibility in URLs



- Binary data handling
- Data integrity

String Best Practices

Efficient and secure string manipulation is crucial in JavaScript development. The following are some of the best practices to be followed for the optimization of the code:

Use template literals for concatenation

Prefer string methods over regular expressions

Handle empty strings gracefully

Be mindful of Unicode characters

String Best Practices

Sanitize user input for security

Utilize regular expressions wisely

Optimize string concatenation in loops

Leverage string interpolation for readability

Assisted Practice

Demonstrating String Manipulation and Optimization **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use advance string operations for manipulation and optimization in JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

1. Create and execute JS file

Functions and Prototyping

Function

A function is a block of code to perform a particular task. It is executed when something invokes it.

Syntax:

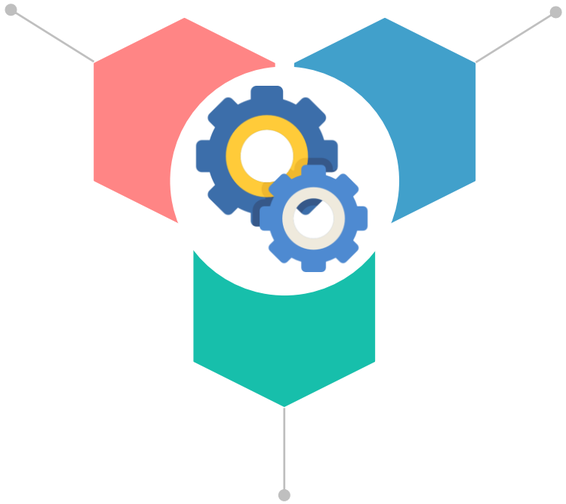
```
function functionName(parameter1,
parameter2, parameter3)
{
    statements 1;
    statements 2;
    statements 3;
return value;
}
```

A JavaScript function is defined with the **function** keyword, followed by a name and parentheses ().

Function

The characteristics of the function:

Function parameters are listed inside the parentheses () in the function definition.

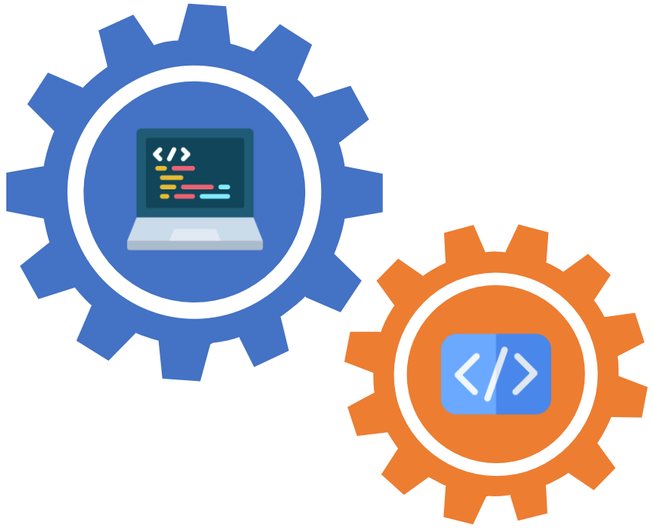


Function arguments are the values received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

Why Use Functions?

Code reusability:
You can define the code once and can use it multiple times



Different results:
You can use the same code multiple times with different arguments and can get different results

Function Definition

Example:

```
function Functionname(n1, n2) {
  return n1 * n2;
}
```

- The function definition is sometimes mentioned as a *function declaration* or *function statement.*
- Every function definition should begin with the *function* keyword. The user-defined function name should be unique.

Function Calling

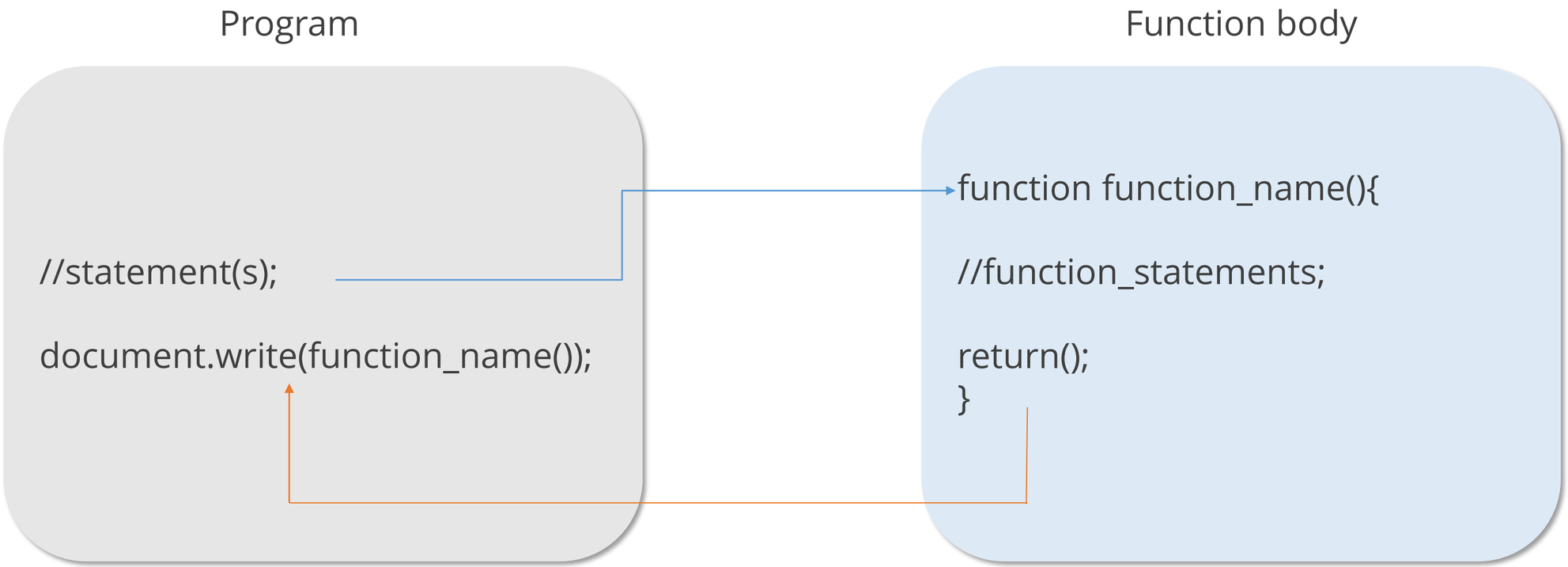
An expression that transfers control and arguments to a function is called a function call.

Example:

```
function add(a, b) {
  return a + b;
}
var result = add.call(this, 14, 16);

console.log(result);
```

Function Execution

The following diagram depicts the function execution in the program:



Function Objects

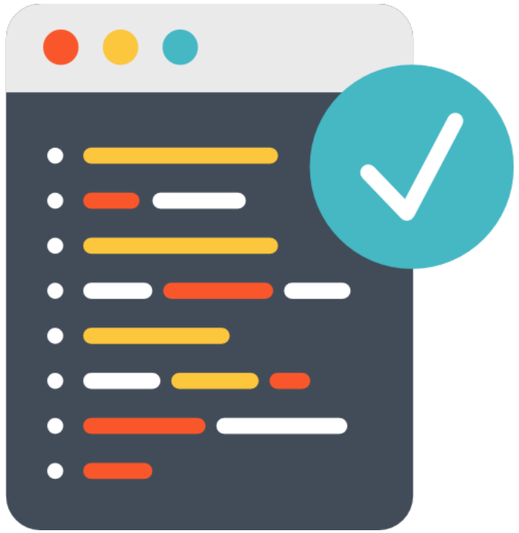
JavaScript functions have a special type of object called function objects.

Example:

```
function message() {
    console.log("Greetings
Simplilearners!");
}

console.log(typeof message);
// => function
```

- The function object executes the code globally.
- The function constructors are used to create function objects.

Passing Functions as Arguments



- Functions can be variables in JavaScript. So, you can pass a function as an argument to the other function
- The function passed in can also be called a *Callback* function

Passing Functions as Arguments

Example:

```
function functionOne(x) {
alert(x);
 }

function functionTwo(var1, callback) {
callback(var1);
 }

functionTwo(2, functionOne);
```

- In the example, *functionOne* takes in an argument and issues an alert with **x** as its argument.
- *functionTwo* takes in an argument and a function and then passes the argument to the function.
- *functionOne* is the callback function in this case.

Function Returning Function

Example:

```
            function sqr()  {
          return function cal(x)  {
              return x * x;
   }
}
function functionTwo(var1, callback) {
            callback(var1);
}

            var ans=sqr();
               ans(5);
```

- Return statement passes information from inside a function back to the point in the main program where the function was called.
- Returning a function is useful when you are using a prototype-based object model.
- We can return a sub-function to the main function as shown in the example.

Function Constructors

A new Function object is created using the **Function()**.

Example:

```
const add = new Function('a', 'b',
 'return a + b');

console.log(add(10, 6));
```

When any function is called with a *new* keyword:

- It creates a new empty anonymous object.
- It uses the anonymous object within the call.
- It implicitly returns the new object at the end of the call.

Function Constructors

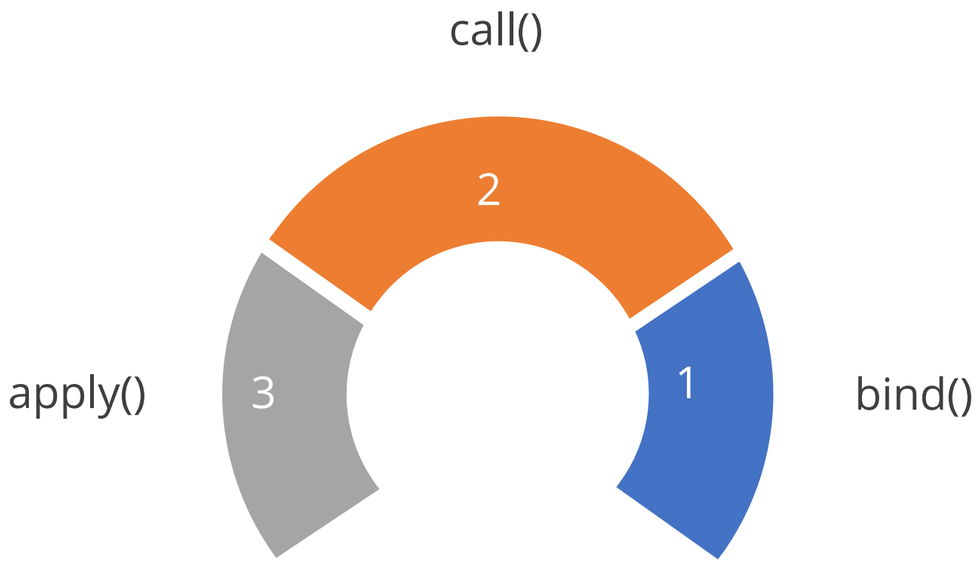
Example:

```
const person = {
  firstName: "Peter",
  lastName : "Parker",
  id       : 5566,
  fullName : function() {
    return  this.firstName  +  "  "  +
this.lastName;
  }};
```

- The *this* keyword refers to the object that is currently executing the code.
- The global object will be referred to by "this" if the function is not a property of any object.
- The value of 'this' can be changed using the call(), apply() and bind() methods, or by using arrow functions (es6).

Function Context

Ways to change the context in which functions are executed:



Bind()

The bind() method creates a new function with the specified context (the value of this), and any additional arguments passed to bind() are passed as arguments to the new function when it is called.

Example:

```
let bike = {
    data:[
       {name:"Ducati", year:2021},
       {name:"Harley-Davidson", year:2021}
   ]
}
bike.showData = user.showData.bind (bike);
bike.showData ();
```

Call()

The call() method allows you to call a function and specify the context (the value of this) for that function call.

Example:

```
function Elements () {
    var args = Array.prototype.slice.call (arguments);
    console.log (args);
 }

 Elements('Water', 'fire', 'wind', 'Earth');
```

Apply()

The apply() method works similarly to call(), but it takes an array of arguments instead of a list of arguments.

Example:

```
function Food(type){
    this.type = type;
}
function setFood(Variety){
    Food.apply(this, ["Vegetarian", "vegan", "seafood"]);
    this.Variety = Variety;
}
```

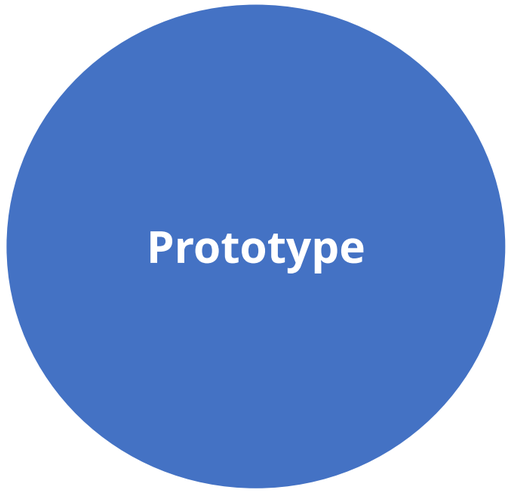
Prototype

Prototype allows the objects to inherit features from one another.

Example:

```
const myObject = {
  city: "Los Angeles",
  greet() {
    console.log(`Greetings from ${this.city}`);
  },
};

myObject.greet();
```

Prototype



01 All JavaScript objects inherit properties and methods from a prototype.

02 Function’s prototype property is modifiable, but object’s prototype property is not visible.

03 Every function includes prototype object by default.

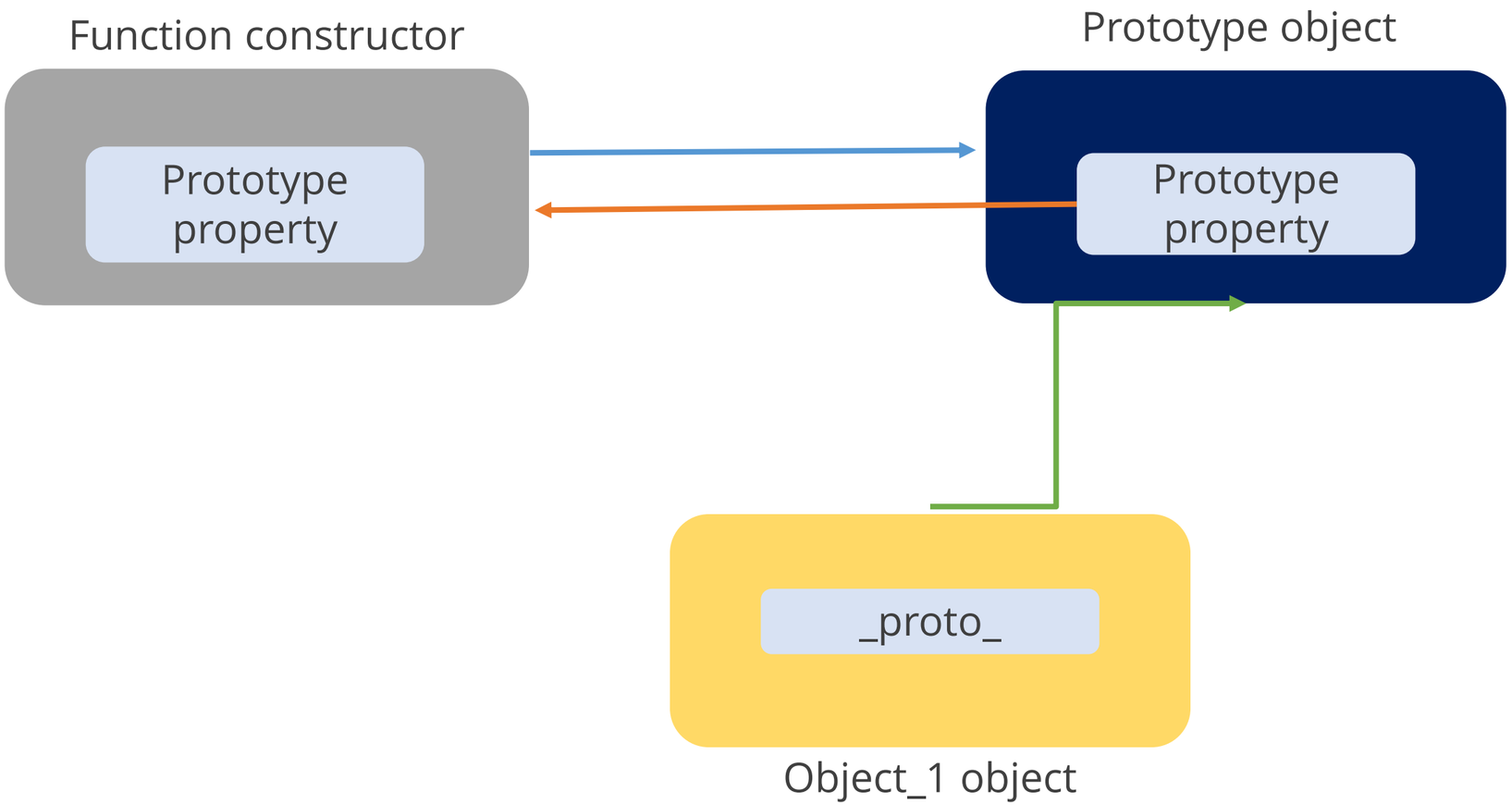
Prototype Chaining

Every object has a prototype and the prototype of an object is also an object, creating a nesting of prototypes.

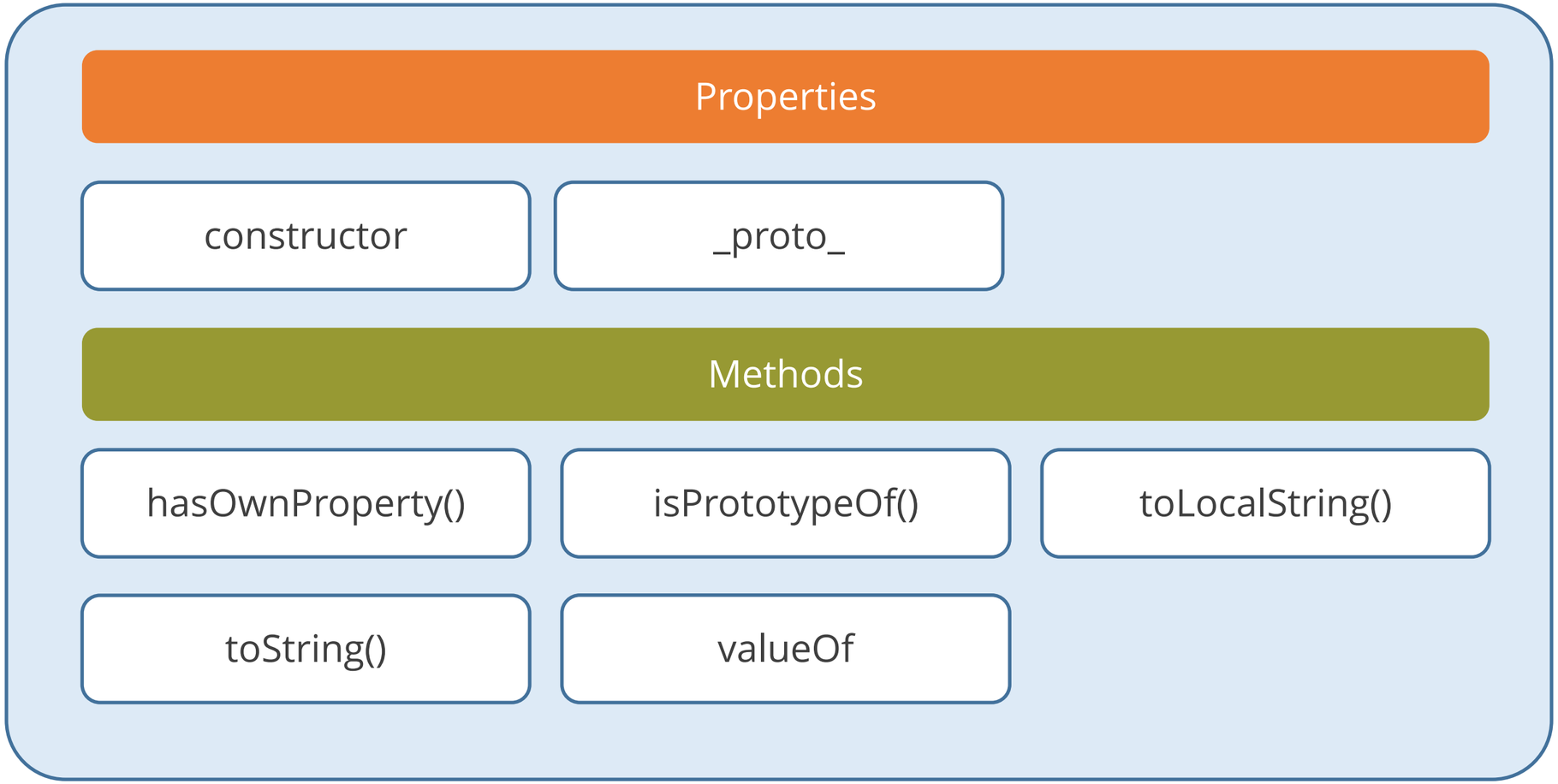
Example:

```
User {name: "Mike"}
  name: "Mike"
  _proto_:
const: class User
print: f print
_proto_: Object
```

Prototype Chaining



Prototype: Properties and Methods



Assisted Practice

Working with Functions **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate how to work with functions.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a JavaScript project in your IDE.

2. Write a program in JavaScript to demonstrate how the function works, how to pass a function as an argument to the other function, and how to return a function to a function.

Assisted Practice

Functions and Prototyping **Duration: 15 Min.**

Problem Statement:

You have been assigned a task to demonstrate the use of functions and prototypes in JavaScript.

Assisted Practice: Guidelines

Steps to be followed:

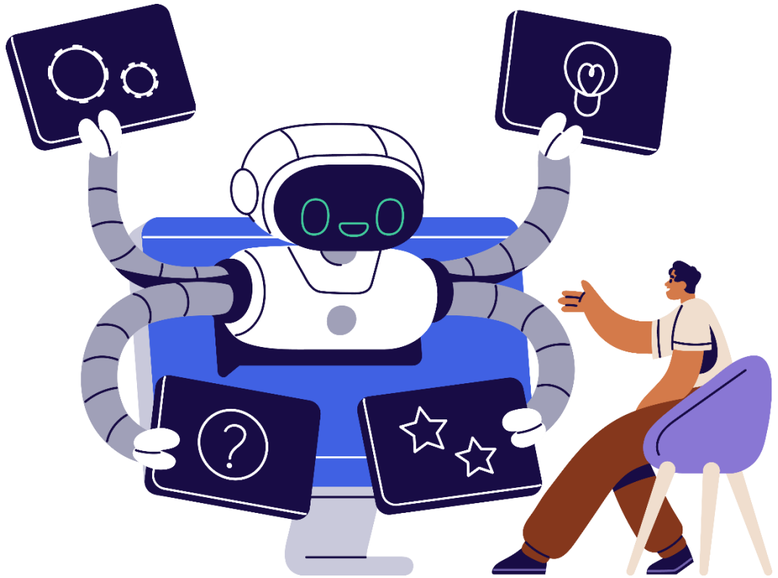
1. Create a JavaScript project in your IDE.

2. Write a program in JavaScript using prototypes of functions to display the employee information of an organization.

Understanding AI Coding Assistants

Introduction to AI Coding Assistant Tools

AI coding assistants use artificial intelligence and machine learning to aid in software development by providing real-time suggestions, code snippets, error detection, and documentation.



They can be integrated into Integrated Development Environments (IDEs), standalone applications, and browser extensions.

Evolution of Coding Assistant Tools

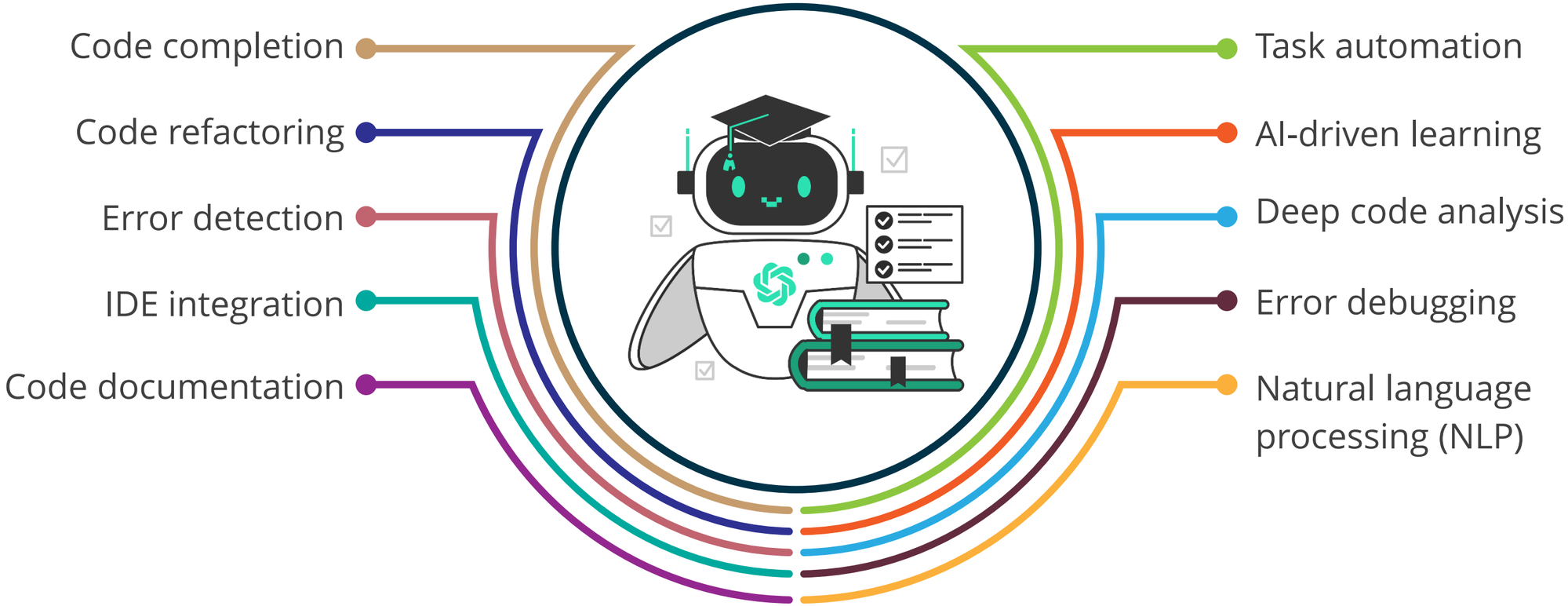
The following shows the evolution of integrating AI coding assistants into the software development lifecycle:

Early stages
Basic code completion tools based on simple pattern matching and token analysis

Rise of intelligent assistants
Context-aware code completion tools, machine learning integration, and semantic code analysis

Deep learning revolution
Tools integrated with deep learning models, language models, and NLP

Key Features and Benefits



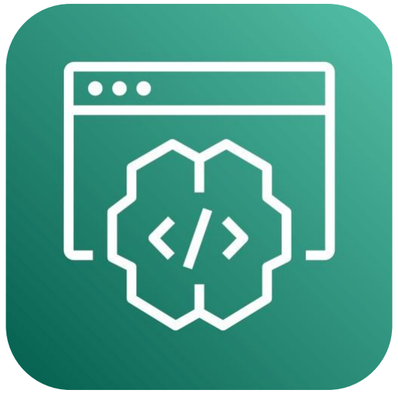
Popular AI Coding Assistant Tools

Following are a few popular AI coding assistant tools:

GitHub Copilot

tabnine

ChatGPT



Amazon CodeWhisperer

watsonx™

codeium

MUTABLEAI

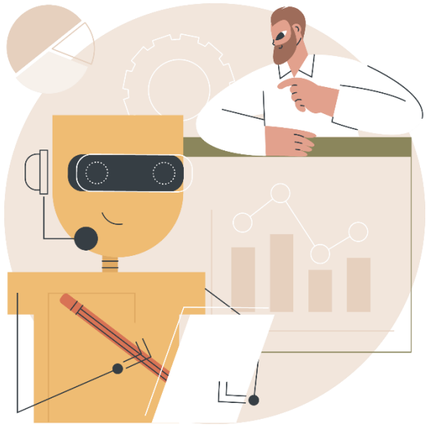
Types of AI Coding Assistant Tools

The types of AI coding assistants with specific functionalities that cater to different aspects of coding and software development are as follows:

Predictive coding assistants:
These AI coding assistants predict and suggest the next few words or lines of code.

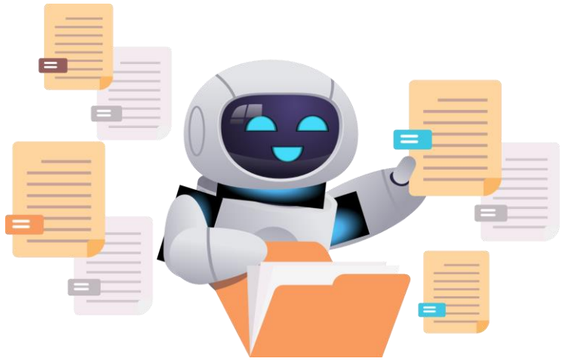
For example: GitHub Copilot, Tabnine, AWS Code Whisperer, OpenAI Codex, and PolyCoder

Types of AI Coding Assistant Tools



Code generation assistants:
These AI coding assistants generate code samples based on a description or comments provided by the developer.

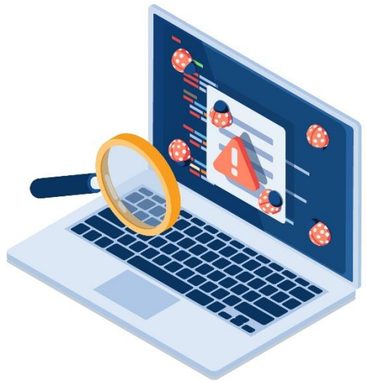
For example: OpenAI Codex, GitHub Copilot, CodePal, Amazon, and CodeWhisperer



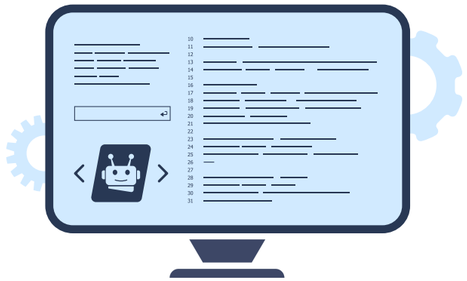
Documentation assistants:
These assistants quickly provide relevant documentation for the code directly in the IDE.

For example: IntelliCode can display documentation for Visual Studio Code

Types of AI Coding Assistant Tools



Code review and debugging assistants:
These tools review code for potential bugs and security vulnerabilities and suggest fixes to improve code quality
.
For example: GitHub's AI Code Reviewer, Tabnine, and DeepCodeAI



Snippet generation:
These tools can generate reusable code snippets based on user input and manage those snippets efficiently.

For example: OpenAI Codex, Tabnine, CodeT5, and Polycoder

Companies Using AI Assistant Tools

Several companies actively utilize generative AI tools for software development to enhance their productivity and creativity. The key examples include:

Codeium: An Open-Source AI Assistant

Introduction to Codeium

Codeium is an advanced AI-powered code acceleration tool designed to assist developers in coding and software development tasks.

It is primarily known for its ability to automatically generate code snippets and provide coding suggestions based on the context of the user's project.

Core Features and Functionalities

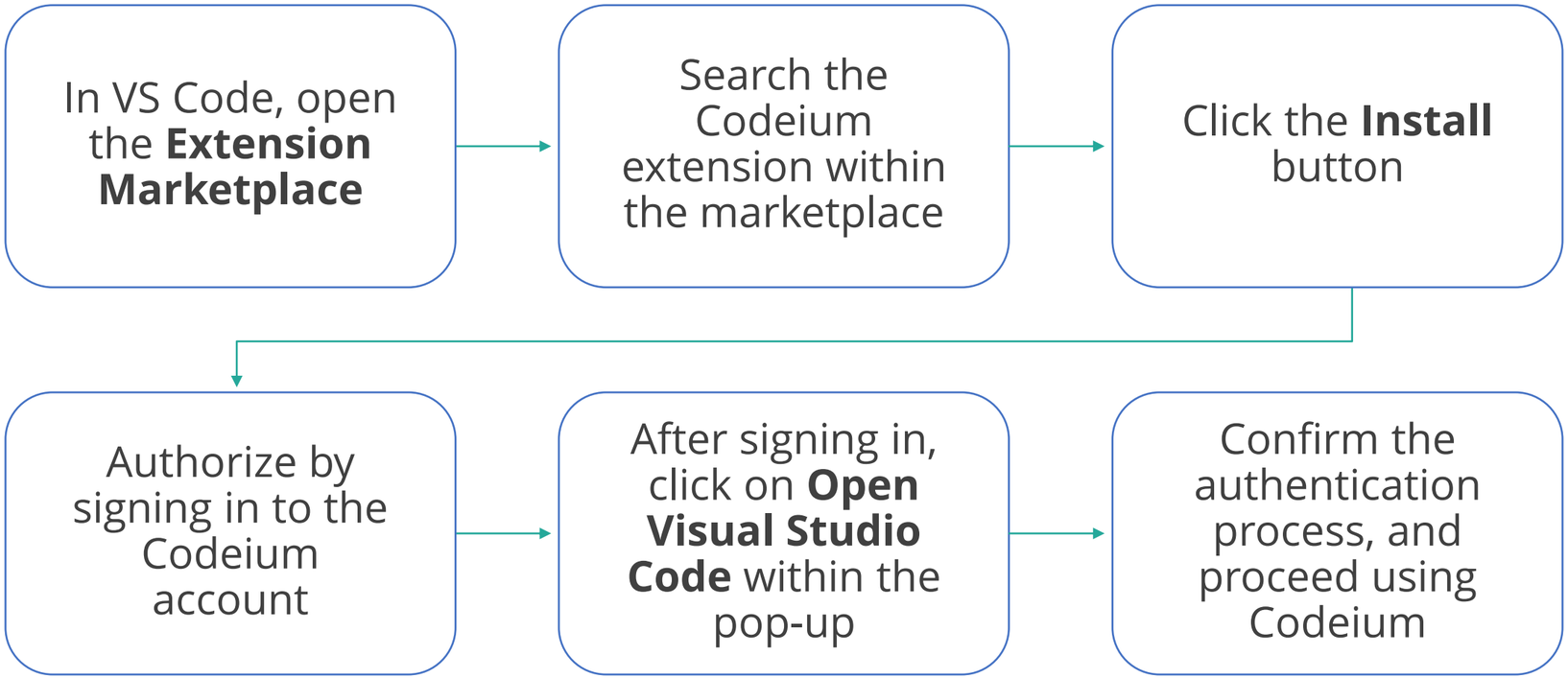
- AI-driven code completion
- Code optimizing suggestions
- Integration with multiple IDEs
- Error detection and debugging
- Support multiple programming languages
- IDE-integrated chat
- Large community support
- Code generation using prompts

Supported IDEs

Codeium supports a wide range of Integrated Development Environments (IDEs) for its AI code completion and chat functionalities. The supported IDEs include:

Installation and Setup

Following are the steps to install and set up Codeium in VS Code and browser-based VS Code IDEs:



Note: The installation and setup process for other IDEs is similar to that of VS Code.

Assisted Practice

Installing Codeium in VS Code **Duration: 5 Min**

Problem Statement:

You have been asked to install Codeium within the VS Code IDE to explore its AI-powered features and functionalities.

Assisted Practice: Guidelines

Steps to be followed:

1. Install the Codeium extension in VS Code
2. Create a Codeium account
3. Provide a manual authentication token to complete the installation

Assisted Practice

Generating Code Using Codeium **Duration: 10 Min**

Problem Statement:

You have been asked to demonstrate the use of Codeium by showcasing its capabilities of generating, modifying, and explaining code, as well as generating documentation for the code.

Assisted Practice: Guidelines

Steps to be followed:

1. Create an HTML page using the code generation feature
2. Write a JavaScript code using the autocomplete feature

Codeium vs. GitHub Copilot

Aspects	Codeium	GitHub Copilot
Functionality	Single and multi-line code generation with IDE-integrated chat	Single and multi-line code generation with chat interface
IDE integration	Broad range of IDE integration	Primarily designed for VS Code
Supported languages	Supports more than 70 programming languages	Supports a wide range of programming languages
Pricing	Free for individual use	Paid for individual use
User feedback	Positive user feedback for being a free alternative	Positive user feedback for efficient code generation

Codeium vs. Tabnine

Aspects	Codeium	Tabnine
Functionality	Single and multi-line code generation with IDE-integrated chat	Single and multi-line code generation
IDE integration	Wide range of IDE integration	Works with most of the popular IDEs
Supported languages	Supports more than 70 programming languages	Supports most of the programming languages
Pricing	Free for individual use	Paid for individual use
Enterprise support	Targeted at individual as well as enterprise users	Targeted mainly for enterprise solutions

Codeium vs. Amazon CodeWhisperer

Aspects	Codeium	Amazon CodeWhisperer
Functionality	Single and multi-line code generation with IDE-integrated chat	Code generation with security reviews but no chat window
IDE integration	Broad range of IDE integration	Primarily designed for the AWS platform
Supported languages	Supports more than 70 programming languages	Supports major programming languages
Pricing	Free for individual use	Cost may vary based on specific AWS service usage

Codeium vs. ChatGPT

Aspects	Codeium	ChatGPT
Functionality	Single and multi-line code generation with IDE-integrated chat	Large language model used for various text generation
IDE integration	Broad range of IDE integration	Mostly used as a standalone tool
Capabilities	Capable of performing coding related tasks	Capable of performing a wide range of tasks
Pricing	Free for individual use	Free for individual with some usage limitations
User experience	Designed only for coding assistance	General purpose with conversational interaction

Assisted Practice

Exploring the AI Chat Feature of Codeium **Duration: 10 Min**

Problem Statement:

You have been asked to create a login page using the AI chat feature of Codeium.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a simple login page
2. Modify the login page

ASSISTED PRACTICE

Assisted Practice

Debugging the Code Using Codeium **Duration: 10 Min**

Problem Statement:

You have been asked to identify and debug the errors in code using the debugging feature of Codeium.

Assisted Practice: Guidelines

Steps to be followed:

1. Create a sample JavaScript file
2. Debug the code using Codeium

Key Takeaways

- JavaScript is light-weight, cross-platform, and object-oriented programming language used to create client-side dynamic pages.
- There are two types of data types in JavaScript: primitive data type and non-primitive (object) data type.
- A control statement allows structured control of the sequence of application statements, such as loops and conditional tests.
- An array is a fundamental data structure in JavaScript used to store a collection of items in a specific order.

Key Takeaways

- String formatting involves manipulating and arranging text in a way that enhances readability and presentation in programming languages like JavaScript.
- An expression that transfers control and arguments to a function is called a function call.
- AI coding assistant tools assist developers by generating code snippets, suggesting code completions, detecting and fixing bugs, and providing relevant documentation.
- AI coding assistant tools are innovative software solutions designed to enhance programming efficiency and accuracy.
- Codeium is an advanced AI-powered open-source code acceleration tool designed to assist developers in coding and software development tasks.

Thank You