Introduction to JavaScript

Topics Covered:

- What is JavaScript?
- How is JavaScript executed?
- How to include scripts in HTML?
- How to change the HTML content via JavaScript?
- How can JS modify/add styles to HTML elements?

Topics in Detail:

What is JavaScript?

- JavaScript is a powerful, flexible, and fast programming language.
- JavaScript powers the dynamic behavior on websites.
- JavaScript remains at the core of web development.
- JavaScript is used mainly for enhancing the interaction of a user with the webpage.
- With the help of JavaScript, web pages can be made live and interactive.

How is JavaScript executed?

- The browser is responsible for running JavaScript code, being a scripting language, JavaScript cannot run on its own.
- When a user requests an HTML page with JavaScript in it, the script is sent to the browser and it is up to the browser to execute it.
- The main advantage of JavaScript is that all modern web browsers support JavaScript.

How to include scripts in HTML?

- HTML <script> tag is used to embed data or executable client side scripting language in HTML pages.
- Example:

```
<!DOCTYPE html>
<html>
<head>
</head>
<body>
    <h1> JavaScript Tutorials</h1>

    <script>
        //write JavaScript code here..
        alert('Hello, how are you?')
    </script>
</body>
</html>
```

- A <script> tag can also be used to include an external script file to an HTML web page by using the src attribute.
- If you don't want to write inline JavaScript code in the <script></script> tag, then you can also write JavaScript code in a separate file with .js extension and include it in a web page using <script> tag and reference the file via src attribute.
- The <script> tag can be included in the <head> or <body> part of the HTML.
- Example:

```
<!DOCTYPE html>
<html>
<head>
    <script src="/MyJavaScriptFile.js" ></script>
</head>
<body>
    <h1> JavaScript Tutorials</h1>

</body>
</html>
```

How to change the HTML content via JavaScript?

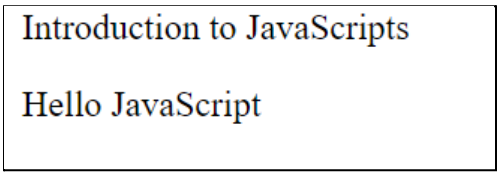
- HTML elements can be accessed from scripts by using getElementById() method.
- The getElementById() method accesses the HTML element based on its unique id.
- Example - HTML Code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Intro to JS</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <p id="intro">Introduction to JavaScripts</p>
    <p id="p1"></p>
    <script src="script.js"></script>
  </body>
</html>
```

- JS Code:

```
document.getElementById("p1").innerHTML = "Hello JavaScript";
```

- Output:



How can JS modify/add styles to HTML elements?

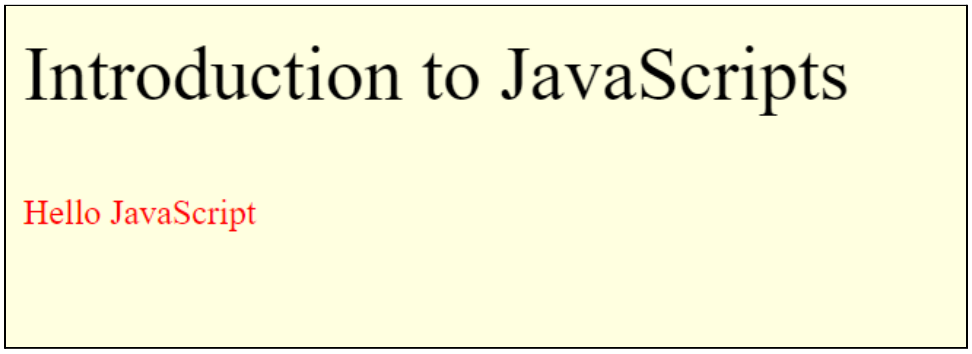
- The style of the HTML elements can be modified by accessing them from JavaScript.
- Syntax:
  document.object.style.Property-name = "property-value"
- Example - HTML Code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Intro to JS</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <p id="intro">Introduction to JavaScripts</p>
    <p id="p1"></p>
    <script src="script.js"></script>
  </body>
</html>
```

- JS Code:

```
document.getElementById("p1").innerHTML = "Hello JavaScript";
document.getElementById('p1').style.color='red';
document.getElementById('intro').style.fontSize='35px';
document.body.style.backgroundColor  = "lightyellow";
```

- Output:



Coding is practice... Practice your JavaScript Code, with help of following steps, which will give you a clear understanding about the basics of JavaScript.

Challenge 1: Accessing the HTML elements
Steps:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension. Example: index.html
- Copy and Add the below given html code in the file.

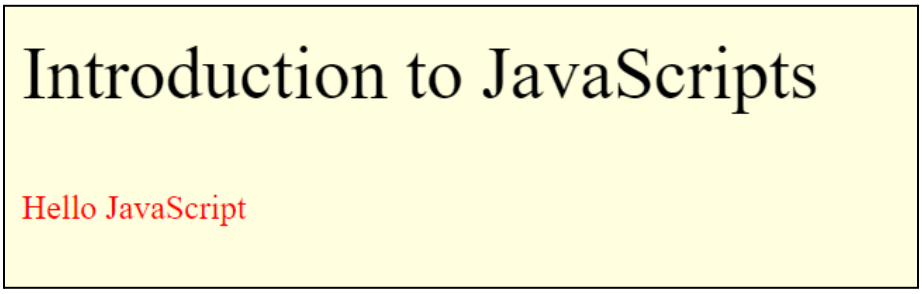
```
<!DOCTYPE html>
<html>
  <head>
    <!--Title of the document-->
    <title>Intro to JS</title>
  </head>
  <body>
    <!--Pragraphs with ID's-->
    <p id="intro">Introduction to JavaScripts</p>
    <p id="p1"></p>
    <!--Included with the link of external script file-->
    <script src="script.js"></script>
  </body>
</html>
```

- Select New File in the same folder.
- Save the file in the with js extension. Example: script.js → the file name mentioned in the html code. If you name the file with another name remember to include the same file name in the HTML code.
- Copy the following code and paste it in the script file.

```
// accessing the second paragraph and adding text to the paragraph
document.getElementById("p1").innerHTML = "Hello JavaScript";
// accessing the second paragraph and adding text color to it
document.getElementById('p1').style.color='red';
// accessing the first paragraph and modifying the font size of it
document.getElementById('intro').style.fontSize='35px';
// accessing the whole body elements of the HTML and adding the
background color to it
document.body.style.backgroundColor  = "lightyellow";
```

- Preview the output and validate.

- Sample output:



Challenge 2:
With you new gained knowledge in JavaScript, create a web page with help of following hints:

- HTML: Elements with unique id and no content.
- CSS: no styles.
- JS:
    - Access the HTML elements by using the ***getElementById()*** method and add content to the elements.
    - Add basic styles to the element either by using ***getElementById()*** method or by accessing the HTML ***body*** element.

Sample code:

```
<!DOCTYPE html>
<html>
  <head>
    <!--Title of the document-->
    <title>JavaScript</title>
  </head>
  <body>
    <!--Pragraphs with ID's-->
    <p id="heading"></p>
    <p id="p1"></p>
    <p id="p2"></p>
    <p id="p3"></p>
  </body>
</html>
```

- Select New File in the same folder.
- Save the file in the with js extension.

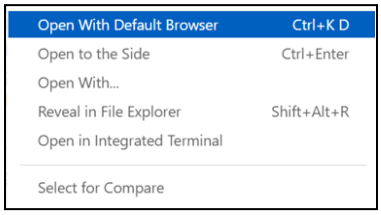
- Link the external javascript file in the HTML code either inside the ***<head> or <body>*** tag. Remember, do not include the script file inside the paragraphs or heading or div elements.

```
<script src="script.js"></script>
```

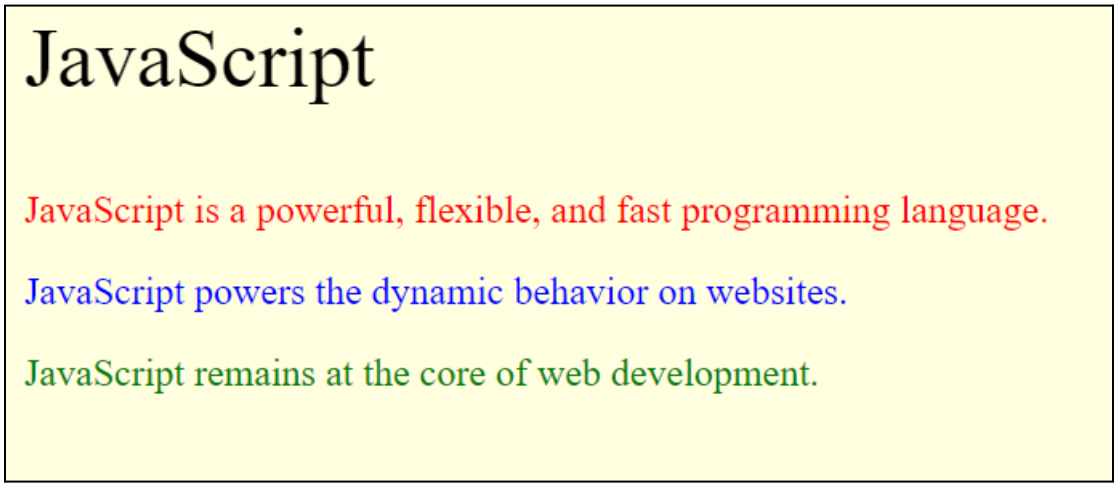
- Add scripts in script.js.

```
// accessing the whole body elements of the HTML and adding the
background color to it
document.body.style.backgroundColor  = "lightyellow";
// accessing the first paragraph and adding text to the paragraph
document.getElementById("heading").innerHTML = "JavaScript";
// accessing the first paragraph and modifying the font size of it
document.getElementById('heading').style.fontSize='35px';
// accessing the second paragraph and adding text to the paragraph
document.getElementById("p1").innerHTML = "JavaScript is a
powerful, flexible, and fast programming language.";
// accessing the third paragraph and adding text to the paragraph
document.getElementById("p2").innerHTML = "JavaScript powers the
dynamic behavior on websites.";
// accessing the fourth paragraph and adding text to the paragraph
document.getElementById("p3").innerHTML = "JavaScript remains at
the core of web development.";
// accessing the second paragraph and adding text color to it
document.getElementById('p1').style.color='red';
// accessing the third paragraph and adding text color to it
document.getElementById('p2').style.color='blue';
// accessing the fourth paragraph and adding text color to it
document.getElementById('p3').style.color='green';
```

- View the file in the browser, by right click on the HTML file name in the left pane.



- **Sample Output:**



JavaScript Syntax

Topics Covered:

- JavaScript variables
- Types of variables
- Literals
- Features of Literals
- Types of Operators
- Comment statement
- Rules of writing the identifiers
- JavaScript data types

Topics in Detail:

JavaScript Variables

- **Variables** are the name of the data storing containers.
- The variable values may vary.
- JavaScript variables must have **unique names**. Hence, it is also called **identifiers**.
- Variables can be initialized at any point in the code.
- **var** keyword should only be used at the time of declaration or initialization.
- The keywords **var**, **let** and **const** are used to declare the variables.
- The same variable should not be redeclared twice.
- JavaScript is an untyped language, i.e a variable can hold any data type values.

```
var money;
var name;
```

Types of Variables

JavaScript variables are of two types based on the scope of the variable where it is defined.

- Global variables - It can be defined anywhere in the JS code.

```
var data=200;//gloabal variable declaration
function a(){
document.writeln(data);
}
```

- Local variables - It will be available only for a particular function where it is defined.

```
function b(){
var data=50;//local variable declaration
document.writeln(data);
}
```

Literals

- Literals are fixed values.
- Before ES6, the strings were created using single quotes (‘) or double quotes (“) and had very limited functionality.
- In ES6, the strings are created using backtick(`) and gives more control over dynamic strings.
- Syntax:
  var a = `some string`;

```
var p = 1000;
var n = 1;
var r = 7;
var SI = `Simple Interest is ${(p *
    n * r)/100}`;
```

Features of Literals

- **Multiline string** is the ability to span **multiple lines**.
  Before Template Literals, an escape sequence \n was used to give new line character
  In Template Literals, no need to add \n.

```
// With template literal
console.log(`List of Fruits
Apple
Orange
Mango`);
```

- **String formatting** is the ability to substitute a part of a string for **expression** or **variable values**. **${}** syntax expression produces the value. This value can be any computation operations or even a string.

```
    //Expression
var p = 1000;
var n = 1;
var r = 7;
var SI = `Simple Interest is ${(p *
      n * r)/100}`;
console.log("Simple Interest is" + SI);
```

- **Tagged Template** is like **function** definition, but the only difference is at the time of call there will be **no parenthesis ()** and simply **an array of string** will be passed as **parameter**.

```
    //TaggedLiteral
function TaggedLiteralEg(strings) {
  document.write(strings);
}
TaggedLiteralEg `Hello Javascript`;
```

- **String.raw()** creates a raw string just like the default function and does string concatenation.

```
    //String.raw
    var s=String.raw`Value of expression is ${2+3}`;
alert(s);
```

- **Nested Template** allows checking multiple conditions

```
    //nested template
    function maximum(x, y, z) {
var c = `value ${ (y>x && y>z) ? 'y is greater' :
`${x>z ? 'x is greater' : 'z is greater'}` }`;
return (c);
}
```

JavaScript Operator

- An Operator is a symbol reserved for performing a special task.
- Operators perform operations on one or more operands. Operands can be variables, string or numeric literals.

Types of Operators

- Arithmetic Operators
- Relational Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Ternary Operators
- TypeOf Operator

Comment Statement

- JavaScript comment is used to **add information** about the code, warnings or suggestions etc. and helps in the easy **interpretation** of the code.
- These comments will be **ignored** by the JavaScript engine.

Advantages of adding comment statements

- It helps in easy **understanding** of the code.
- It is also used to **disable** a part of code from being executed.

Types of JavaScript Comments

- **Single Line Comment** - **Double forward slashes (//)** is used before or after the statement

```
let x = 5;       // Declare x, give it the value of 5
```

- **Multi Line Comment** - It can be used as a single or multi line comment. It is represented as **/* comment */.**

```
/*
The code below will change
the heading with id = "myH"
and the paragraph with id = "myP"
in my web page:
*/
document.getElementById("myH").innerHTML = "My First Page";
document.getElementById("myP").innerHTML = "My first paragraph.";
```

JavaScript Identifiers

- The names given to variables, functions, parameters, classes, etc. are called Identifiers.
- Rules for declaring a JavaScript variable
    - Names can begin with a **letter** (a to z or A to Z).
    - Names can also begin with (_) or ($).
    - Variable names should not start with digits(0-9).
    - Variable names are case-sensitive (a and A are different variables).
    - Variable names can have letters, digits(0-9), underscore(_) or dollar($).
    - JavaScript keywords cannot be used as variable names.

JavaScript Data Types

- JavaScript is a **dynamic type language** because the JS engine chooses the data type itself dynamically.
- The variables can hold **different data types**.

Types of Data Types

- Primitive Data Type

Data Type	Description
String	represents sequence of characters e.g. "hello"
Number	represents numeric values e.g. 100
Boolean	represents boolean value either false or true
Undefined	represents undefined value
Null	represents null i.e. no value at all

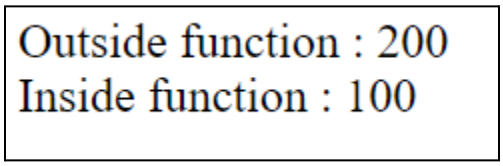
- Non - Primitive Data Type

Data Type	Description
Object	represents instance through which we can access members
Array	represents group of similar values
RegExp	represents regular expression

JavaScript Syntax - Practice Code

Problem Statement 1: Global Vs Local Variable

Differentiate the difference between the scope of global and local variables with help of the JavaScript function.

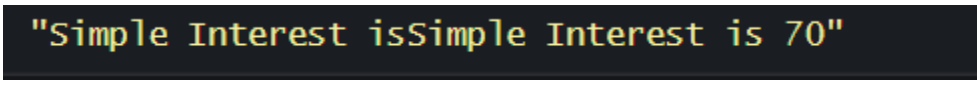


Problem Statement 2: Display methods

Write a simple JavaScript program that displays the multi line text with and without template literals.

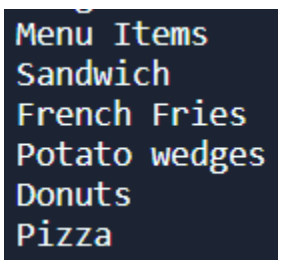
Problem Statement 3: Simple Interest

Using Expressions and simple arithmetic operators calculate and display the simple interest value in the console.



Problem Statement 4: Menu list

Create a menu list using JavaScript template literals and print the Menu list in the console log.



Solution:

Problem Statement 1:

```
<!DOCTYPE html>
<html>
<head>
  <title>Global VS Local Variable</title>
</head>
<body>
 <script>
var data=200;//global variable declaration
function a() {
var data = 100; //Local Variable
document.write("Inside function : ", data); // inside function
}
document.write("Outside function : ", data); // outside function
document.write("<br>");
a();
</script>
</body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html>
<head>
  <title>Literals</title>
</head>
<body>
  <script>
  //Multiline
// Without template literal
console.log('List of Fruits \nApple \nOrange \nMango');

// With template literal
console.log(`List of Fruits
Apple
Orange
Mango`);
</script>
</body>
</html>
```

Problem Statement 3:

```
<!DOCTYPE html>
<html>
<head>
  <title>Global VS Local Variable</title>
</head>
<body>
```

```
 <script>
//Expressions
var p = 1000;
var n = 1;
var r = 7;
//use of operators and assigning values
var SI = `Simple Interest is ${(p *
            n * r)/100}`;
console.log("Simple Interest is" + SI);

</script>
</body>
</html>
```

Problem Statement 4:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Literals</title>
</head>
<body>
  <script>

// With template literal
console.log(`Menu Items
Sandwich
French Fries
Potato wedges
Donuts
Pizza`);
  </script>
</body>
</html>
```

JavaScript Variables

Topics Covered

- Variables
- Types of Variables
    - Global variables
    - Local variables
- Different Ways of Declaring Variables
    - Using let
    - Using const
    - Using var
    - Using nothing

Topics in Detail

Variables

- **Variables** are the data storing containers.
- Values in the variable may vary..
- JavaScript variables must have **unique names**.
- The keywords **var**, **let** and **const** are used to declare the variables.
- The same variable should not be redeclared twice.
- JavaScript is an untyped language, i.e a variable can hold any data type values.
- Syntax:

```
var money;
var name;
```

- Variables can be declared in multiple lines or even in single line with only one **var** keyword as below

```
var money, name;
```

- The variables can be initialized at the time of variable creation or at any point when you need that variable.

Types of Variables

JavaScript variables are of two types based on the scope of the variable where it is defined

- Global variables
- Local variables

Global Variable

- The Global variable can be defined anywhere in the JS code.

```
var data=200;//gloabal variable declaration
function a(){
document.writeln(data);
}
```

Local Variable

- The Local variable will be visible only within a particular function where it is defined.
- Parameters of a function are always local to that particular function.

```
function b(){
var data=50;//local variable declaration
document.writeln(data);
}
```

Different ways to declare variables

Declare JavaScript variables using any of the below keywords.

Until 2015, **the var** keyword was used to declare JavaScript variables.

let and **const** keywords were added to JavaScript later.

- Using **let**
- Using **var**
- Using **const**
- Using **nothing**

Using 'let' keyword

- If we try to **Redeclare global** variables with the **let** keyword, we will get a **syntax error**.

```
let x = "John Doe";

let x = 0;

// SyntaxError: 'x' has already been declared
```

- After ES6, JavaScript keywords **'let'** and **'const'** provide **block scope**.

```
{
  let x = 2;
}
// x can NOT be used here
```

- This variable cannot be accessed outside the block

- Variables defined with **let** must be declared before use.

- If we declare with **'let'** keyword, the value of the variables can **change**.
- Redeclaring Variables globally and local simultaneously with let keyword won't impose a problem

```
let x = 10;
// Here x is 10

{
let x = 2;
// Here x is 2
}

// Here x is 10
```

Redeclaring a variable inside a block won't actually redeclare the variable outside the block

Using 'var' keyword

- We can **Redeclare** variables with the **var** keyword.

```
var x = "John Doe";

var x = 0;
```

- Variables declared with **var** keyword don't have **block scope**.

```
{
  var x = 2;
}
// x CAN be used here
```

This variable can be accessed outside the block.

- Redeclaring Variables globally and local simultaneously with **var** keyword will impose a problem

```
var x = 10;
// Here x is 10

{
var x = 2;
// Here x is 2
}

// Here x is 2
```

Redeclaring a variable inside a block will actually redeclare the variable outside the block

Using 'const' keyword

- We **cannot Redeclare** the variables defined with **const** keyword in the same scope. But, we can redeclare the **const** keyword variables in another block or scope.

```
const x = 2;       // Allowed

{
  const x = 3;   // Allowed
}

{
  const x = 4;   // Allowed
}
```

- We cannot **reassign** the variables defined with **const** keyword. It must be assigned at the time of declaration itself.

```
const PI = 3.141592653589793;
PI = 3.14;      // This will give an error
PI = PI + 10;   // This will also give an error
```

- Always, variables that remain unchanged will be declared with the **const** keyword.
- Variables defined with **const** have Block Scope.

```
const x = 10;
// Here x is 10

{
const x = 2;
// Here x is 2
}

// Here x is 10
```

- **Const** keyword is mostly used to declare
  - New Array
  - New Object
  - New Function
  - New RegExp

- **const** keyword defines a constant reference to a value, but not a constant value.
- We cannot **reassign** a constant value, constant array or constant object, but we can **change the elements** of a constant **array** or change the **properties** of a constant **object**.

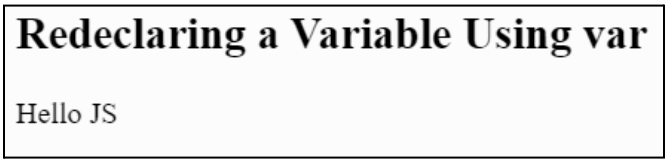
Example

```
// You can create a const object:
const car = {type:"Fiat", model:"500", color:"white"};

// You can change a property:
car.color = "red";
```

JavaScript Variables - Practice Code

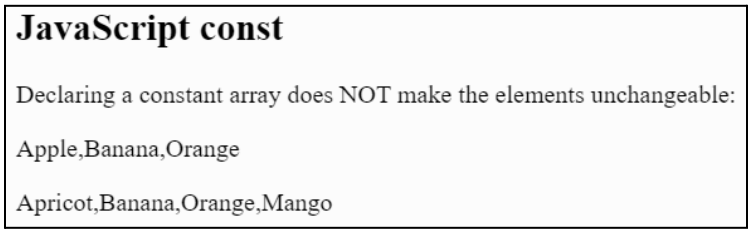
Problem Statement 1: Redeclaration of Variable

Write a simple JavaScript program that displays the redeclared variable.



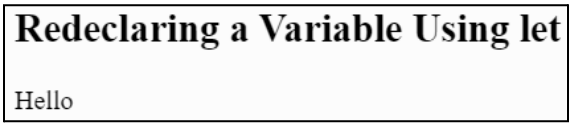
Problem Statement 2: Arrays

Write a JavaScript program with an array, and display the array, display only the element of an array, add a new element to an array, and display the modified array.



Problem Statement 3: Redeclaring Let

Write a simple JavaScript program that tries to redeclare the let variable.



Problem Statement 4: Const

Write a simple JavaScript program that modifies the value of an array element and adds a new element that is declared as const and displays the array before and after modification.

Solution:

Problem Statement 1:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Var_Declaration</title>
</head>
<body>
  <h2>Redeclaring a Variable Using var</h2>
<script>
var  x = 'Hello';
// Here x is Hello
{
  var x = 'Hello JS';
  // Here x is Hello JS
}
// Here x is Hello JS
document.write(x);
</script>
</body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>replit</title>
  </head>
<body>
  <h2>JavaScript const</h2>
<p>Declaring a constant array does NOT make the elements unchangeable:</p>
<p id="demo"></p>
  <p id="demo1"></p>
<script>
```

```
// Create an Array:
const fruits = ["Apple", "Banana", "Orange"];
// Display the Array:
document.getElementById("demo").innerHTML = fruits;
// Change an element:
fruits[0] = "Apricot";

// Add an element:
fruits.push("Mango");

// Display the Array:
document.getElementById("demo1").innerHTML = fruits;
</script>
</body>
</html>
```

Problem Statement 3:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Let Declaration</title>
</head>
<body>
  <h2>Redeclaring a Variable Using let</h2>
<p id="demo"></p>
<script>
let  x = 'Hello';
// Here x is Hello
{
  let x = 'Hello JS';
  // Here x is Hello JS
}
// Here x is Hello
document.getElementById("demo").innerHTML = x;
</script>
</body>
</html>
```

Problem Statement 4:

```
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Const Declaration</title>
  </head>
<body>
  <h2>JavaScript const</h2>

<p id="demo"></p>
  <p id="demo1"></p>
<script>
// Create an Array:
const mobile = ["Iphone", "Redmi", "Oppo"];
// Display the Array:
document.getElementById("demo").innerHTML = mobile;
// Change an element:
mobile[1] = "Oneplus";

// Add an element:
mobile.push("Motto");

// Display the Array:
document.getElementById("demo1").innerHTML = mobile;
</script>
</body>
</html>
```

JavaScript Operators

Topics Covered

- JavaScript Operators
- Types of Operators
    - Arithmetic Operator
    - Comparison (or) Relational Operator
    - Bitwise Operator
    - Logical Operator
    - Assignment Operator
    - Miscellaneous Operator
        - Conditional (or) Ternary Operator
        - TypeOf Operator
    - String Operators

Topics in Detail

JavaScript Operators

- An **Operator** is a **symbol** reserved for performing a special task.
- Operators perform operations on one or more **operands**. Operands can be variables, string or numeric literals.

Types of Operators

- Arithmetic Operators
- Comparison (Relational) Operators
- Bitwise Operators
- Logical Operators
- Assignment Operators
- Miscellaneous Operator
    - Conditional (or) Ternary Operator
    - TypeOf Operator
- String Operators

Arithmetic Operator

Arithmetic Operators perform **arithmetic operations** on numbers.

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
**	Exponentiation
/	Division
%	Modulus(Division Remainder)
++	Increment
–	Decrement

Comparison / Relational Operator

Relational Operators perform **comparison** between two operands. A **boolean** value will be returned as a result of this operation.

Operator	Description
==	Is equal to
===	Identical (equal & of same type)
!=	Not equal to
!==	Not Identical
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Bitwise Operator

Bitwise Operators perform **bitwise** operations on operands.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise XOR
~	Bitwise NOT
<<	Bitwise Left Shift
>>	Bitwise Right Shift
>>>	Bitwise Right Shift with Zero

Logical Operator

Logical Operators perform **logical** operations on the operands.

Operator	Description
&&	Logical AND
||	Logical OR
!	Logical NOT

Assignment Operator

Assignment Operators assign values to the **JavaScript variables.**

Operator	Description
=	Assign
+=	Add and assign
-=	Subtract and assign
*=	Multiply and assign
/=	Divide and assign
%=	Modulus and assign

Miscellaneous Operator

Conditional or Ternary Operator

- Conditional Operator returns value based on the **condition**. It is similar to an **if-else** statement.
- Syntax:

  (Condition) ? value x : value y;
- Example:

```
var a = 10;
var b = 20;
result = (a > b) ? 100 : 200;
```

TypeOf Operator

TypeOf Operator is a **Unary** Operator i.e it uses a **single operand**. This Operator evaluates the data type of the operand.

Type	String Returned by Typeof
Number	“number”
String	“string”
Boolean	“boolean”
Object	“object”
Function	“function”
Undefined	“undefined”
Null	“object”

String Operators

- **(+)** Operator is used to **concatenate** strings.

```
let text1 = "John";
let text2 = "Doe";
let text3 = text1 + " " + text2;
```

- **Assignment Operator (+=)** is also used to **concatenate** strings.

```
let text1 = "What a very ";
text1 += "nice day";
```

- **(+)** operator when used on **strings** it is also known as **Concatenation Operator**.
- The **Concatenation Operator** also allows us to **add a number and string**.

```
let y = "5" + 5;
let z = "Hello" + 5;
```

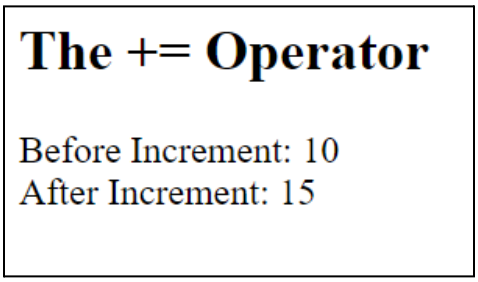
JavaScript Operators - Practice Code

Problem Statement 1: Assignment Operators

Create a JavaScript program that calculates the sum of two variables, where you create two new variables and assign values to them as the first step and then create a new variable that calculates the sum of the other two variables. Print the sum using the **document.write** method.

Problem Statement 2: Addition Assignment

Using the addition assignment operator, increment the value of the existing variable and print the values before and after increment.

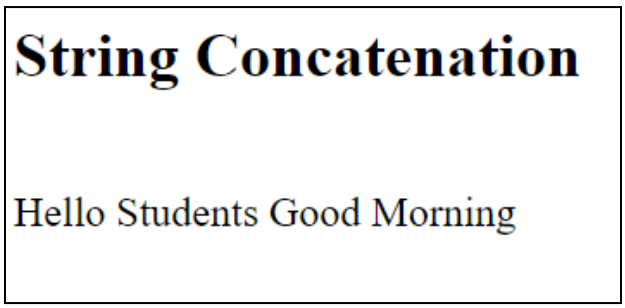


Problem Statement 3: Number of Weeks

Using JavaScript operators calculate the number of weeks and remaining days in a year.

Problem Statement 4: String Concatenation

Using the “+” operator concat two strings and print them. Where you create two variables and assign strings as value to them as a first step. Then create a new variable that concates other variables.



Solution:

Problem Statement 1:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Operators</h2>
<script>
let x = 5;
let y = 2;
let z = x + y;
document.write("The sum is ", z);
</script>
</body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html>
<body>
<h2>The += Operator</h2>
<script>
var x = 10;
document.write("After increment: ", x);
x += 5;
document.write("</br>");
document.write("After increment: ", x);
</script>

</body>
</html>
```

Problem Statement 3:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Operators</title>
</head>
<body>
  <h1>Arithmetic Operator</h1>
  <h1>Number of weeks in a year</h1>
<script>
var days = 365;
var week_days = 7;
var week = Math.trunc(days/week_days); //divide Operation with trunc method
to remove digits after decimal
var remaining_days = days % week_days; //Modulo Operation
document.write(week + " Weeks and "+remaining_days+" Day is there in a
year");
</script>
</body>
</html>
```

Problem Statement 4:

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>Operators</title>
  <h2>String Concatenation</h2>
</head>
<body>
<script>
    var string1 = "Hello Students";
    var string2 = "Good Morning";
    var string = string1 +" "+ string2;
    document.write("</br>");
    document.write(string);
</script>
</body>
</html>
```

JavaScript Conditions

Topics Covered:

- Conditional Statements
- Types of Conditional Statement
    - if
    - else
    - else if
    - Switch Statement

Topics in Detail:

Conditional Statements

- Conditional Statements **control** the **behavior** of the javascript.
- Based on **different conditions** the conditional statements are used to perform **different actions**.

Types of Conditional Statements

- "**If**" Statement
- "**Else**" Statement
- "**Else if**" Statement

"If" Statement

- "**If**" Statement allows JavaScript to make **decisions** and conditionally execute statements.
- "**If**" Statement is used when a specific block of code is to be executed, If a condition is **true**.

Syntax

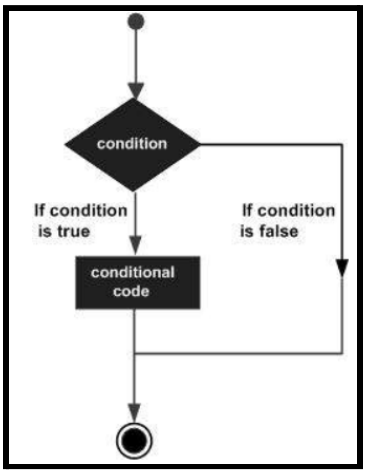
```
if (condition) {
  //  block of code to be executed if the condition is true
}
```

“Else” Statement

- “Else” Statement is used when a specific block of code is to be executed If a condition is **false**.

How “Else” Statement works:

- If the condition is true, then the code inside the if block will be executed.
- If the condition is false, then the code inside the else block will be executed.



Syntax

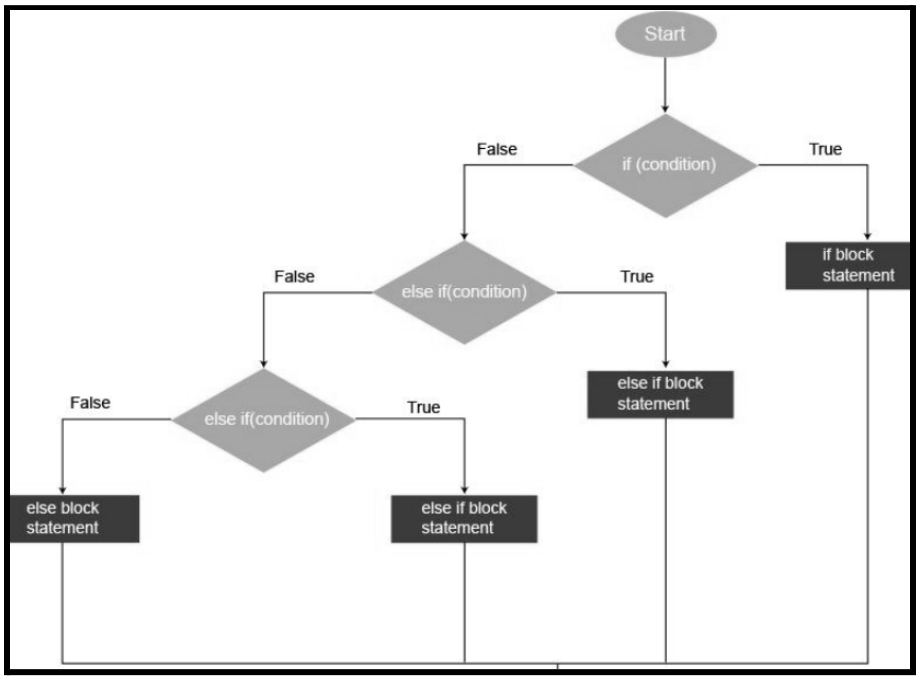
```
if (condition) {
  //  block of code to be executed if the condition is true
} else {
  //  block of code to be executed if the condition is false
}
```

Else If Statement

- Else if Statement is used to make the correct decision out of several decisions.

How “Else If” statement works:

- If the **IF** condition is **true**, then the code inside the **if block** will be executed.
- If the **IF** condition is **false**, then the JS engine checks with the **else if** condition.
- If the **else if** condition is **true**, then the code inside the **else if block** will be executed.
- If the **else if** condition is **false**, then the code in the **else block** will be executed.



- **Syntax**

```
if (condition1) {
  //  block of code to be executed if condition1 is true
} else if (condition2) {
  //  block of code to be executed if the condition1 is false and condition2 is true
} else {
  //  block of code to be executed if the condition1 is false and condition2 is false
}
```

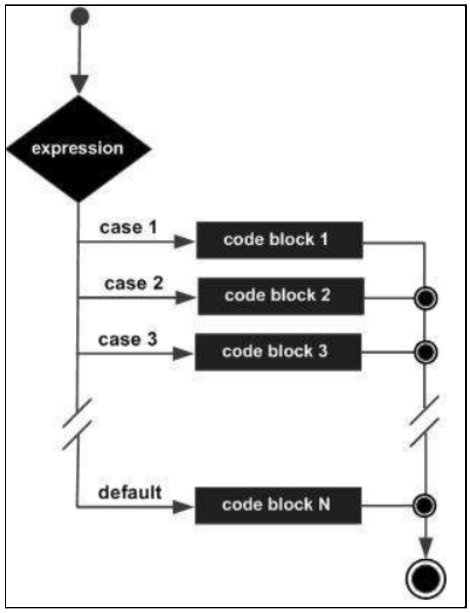
Switch Statement

- To perform **different actions** based on **different conditions** we use **switch** statements.
- The switch statement is more efficient than the else if statement.

How “switch” statement works:

- The switch statement will be evaluated only **once**.
- The value will be **compared** with the value in each case.
- If the correct **match** is found, then the code block inside that case will be executed and at last breaks from the switch statements.
- If the **break statement** is omitted then the interpreter will continue executing each statement.

- If there is **no match** found, then the **default block** will be executed.
- The break statement is **not necessary** in the **last case** of the switch statement.



- **Syntax**

```
switch(expression) {
  case x:
    // code block
    break;
  case y:
    // code block
    break;
  default:
    // code block
}
```

JavaScript Conditions - Practice code

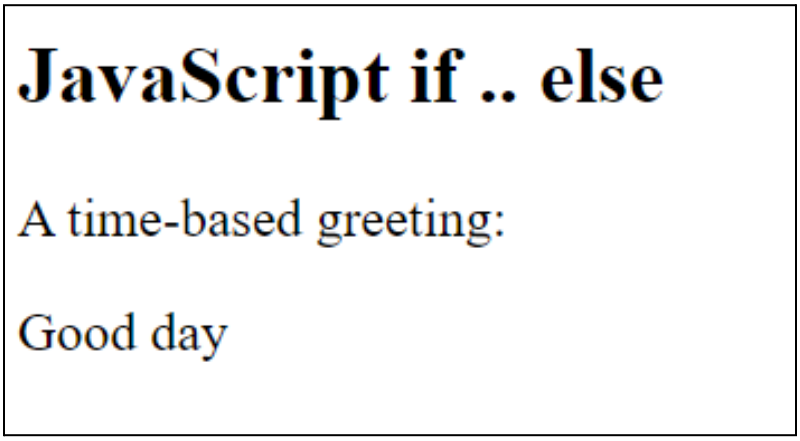
Problem Statement 1: If Else

Using If/Else condition, check whether a person is eligible to vote in India or not.

or

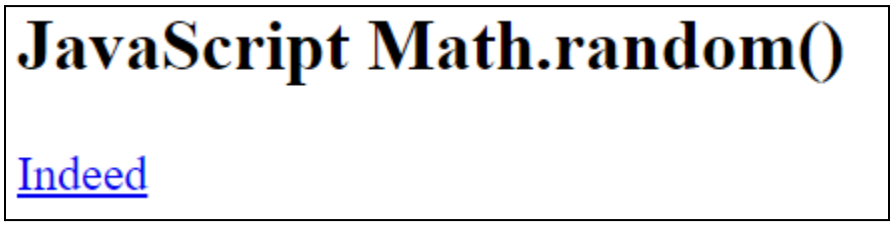
Problem Statement 2: Greetings of the Day

By using Date and getHours method, get the current time and display the greetings (good day/evening) depending on the time.

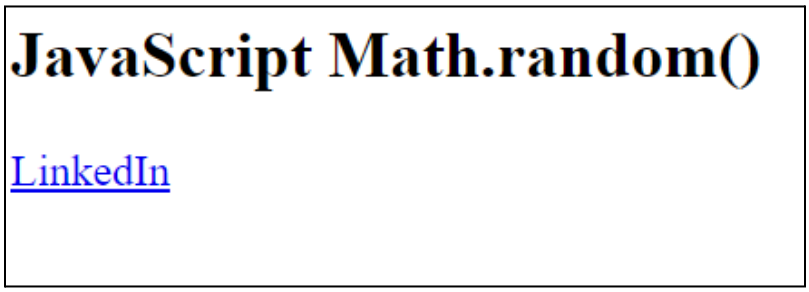


Problem Statement 3: Random links

Display random links in your code, using Math.random() method and conditions. The Math.random() generates random numbers if that number is less than 1 that display a **LinkedIn** link in the output, else display **indeed** link in the output. The output will vary each time.

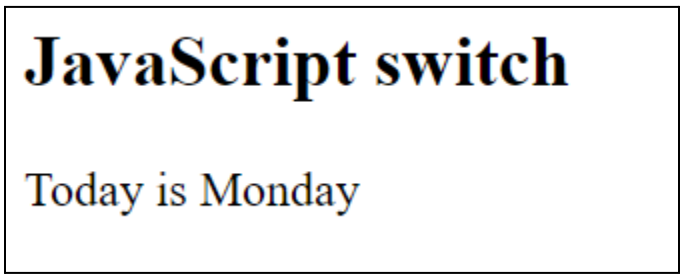


Or



Problem Statement 4: Switch Case

Display the day in the output using switch statement and Date methods.



Solution:

Problem Statement 1:

```
<!DOCTYPE html>
<html>
<head>
  <title>If_Else Statement</title>
</head>
```

```
<body>
  <h1>Voting Eligibility Criteria</h1>
   <script type = "text/javascript">
            var age = 15;
            if( age > 18 )//Conditional Statement
            {
               document.write("<b>Eligible for voting</b>");
            }
     else
            {
             document.write("<b>Not Eligible for voting</b>");
            }
      </script>
</body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript if .. else</h2>
<p>A time-based greeting:</p>
<p id="demo"></p>
<script>
const hour = new Date().getHours();
let greeting;
if (hour < 18) {
  greeting = "Good day";
} else {
  greeting = "Good evening";
}
document.getElementById("demo").innerHTML = greeting;
</script>
</body>
</html>
```

Problem Statement 3:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript Math.random()</h2>
<p id="demo"></p>
<script>
let text;
if (Math.random() < 0.5) {
  text = "<a href='linkedin.com'>LinkedIn</a>";
```

```
} else {
  text = "<a href='indeed.com'>Indeed</a>";
}
document.getElementById("demo").innerHTML = text;
</script>
</body>
</html>
```

Problem Statement 4:

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript switch</h2>
<p id="demo"></p>
<script>
let day;
switch (new Date().getDay()) {
  case 0:
    day = "Sunday";
    break;
  case 1:
    day = "Monday";
    break;
  case 2:
    day = "Tuesday";
    break;
  case 3:
    day = "Wednesday";
    break;
  case 4:
    day = "Thursday";
    break;
  case 5:
    day = "Friday";
    break;
  case  6:
    day = "Saturday";
}
document.getElementById("demo").innerHTML = "Today is " + day;
</script>
</body>
</html>
```

skill academy
by testbook

JavaScript Loops

Topics Covered

- JavaScript Loops
- Types of Loops
    - for
    - for/in
    - for/of
    - while
    - do/while

Topics in Detail

JavaScript Loops

- **Loops** can execute a block of code any **number of times**.
- **Loops** helps to run the **same code** any number of times with **different values** each time.

Entry Controlled Loop: The **Condition** is tested **before** the entry of the Loop body.

Eg: For Loop and While Loop.

Exit Controlled Loop: The **Condition** is tested at the **end** of the Loop body. In this case the loop will be executed at least **once** irrespective of the condition.

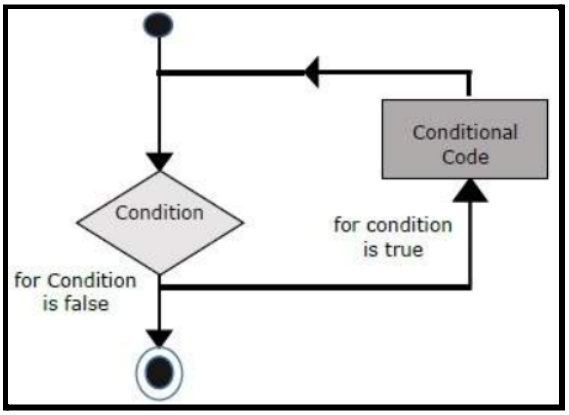
Eg: Do While Loop.

Types of Loops

- for
- for/in
- for/of
- while
- do/while

The For Loop

- The For Loop is used when the number of iterations is known.



- **Syntax**

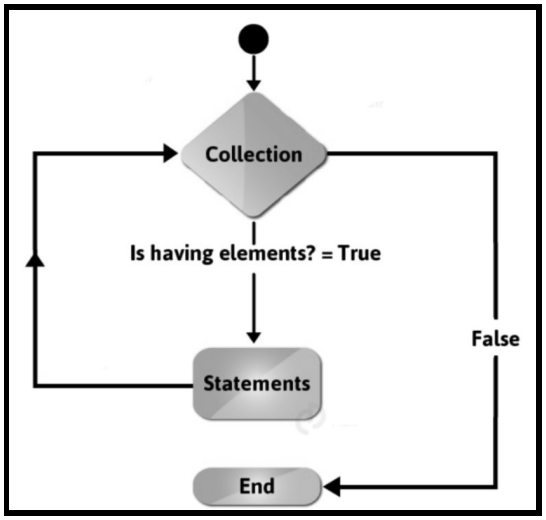
```
for (initialization; condition; increment)
{
   code to be executed
}
```

Initialization	**Initialize** the variable in use.
Condition	**Exit** condition is tested here. It returns a **boolean value**. Condition is checked **prior** to the execution of the loop statement.
Code	If the condition is **true**, then the Loop statement is executed.
Increment/Decrement	**Update** values for next **iteration**.

The For In Loop

- The For In Loop is used to **iterate** through an **object's keys**.
- **Syntax**

```
for (variablename in object) {
   statement or block to execute
}
```



- The **order** of the properties iterated **may not match** with that defined in the object.
- So generally **for in loop** is **not** recommended for use over **arrays**.
- If the **index order** of the array is important, then in that case **don't** use For In Loop.

The For Of Loop

- The For of loop is used to **iterate** over the **values** of the **iterable object** such as
  - Array
  - String
  - Maps
  - NodeList etc.
- **Syntax**

```
for (variable of iterable) {
  // code block to be executed
}
```

- The For of Loop is **not** supported in **Internet Explorer**.
- To execute on the value of each property of the object the For Of Loop calls a **custom iteration hook**.

JavaScript Loops

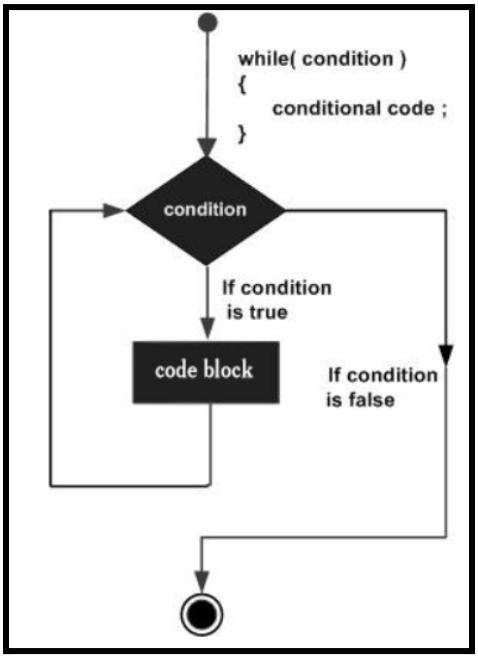
Topics Covered

- while loop.
- do/while loop.

Topics in Detail

The While Loop

- The While Loop is a control flow statement used to execute the code repeatedly as long as the condition is true.
- Similar to repeated if statements.
- The while loop is an entry controlled loop, i.e. The block control condition is at the starting of the loop.
- The JavaScript while loop is used when the number of iterations are not known.



- **Syntax**

```
while (condition) {
  // code block to be executed
}
```

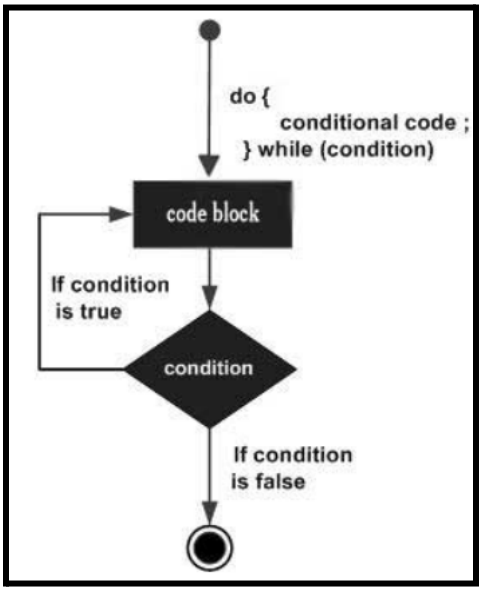
```
let text = "";
let i = 0;
while (i < 10) {
  text += "<br>The number is " + i;
  i++;
}
document.getElementById("demo").innerHTML = text;
```

- In the code above, the statement inside the loop will run until the value of **i** is **less than 5.**

- The loop **variable** should be **incremented or decremented**. If **not, the** Loop will **never end** and will **crash** the system.

The Do While Loop

- The Do While Loop is an Exit controlled loop, i.e. the block condition is at the end of the loop.
- The Do While Loop executes the loop statement before checking the condition.
- It is similar to a while loop. The code will be executed at least once because the condition check happens at the end of the loop.
- The Do While Loop is executed an infinite number of times like While loop.



- **Syntax**

```
do {
  // code block to be executed
}
while (condition);
```

- Example:

```
let str = ""
let i = 0;
do {
  str += "<br>The number is " + i;
  i++;
}
while (i < 5);
```

- In the above code, the statement inside the loop will run until the value of **i** is **less than 5.**
- Since the condition **check** is at the **end point,** the loop will be executed at least **once** even if the **condition** is **false**.
- The loop **variable** should be **incremented or decremented**. If **not,** the Loop will **never end** and will **crash** the system.

Difference Between While and Do While Loop

While Loop	Do While Loop
It is an **entry controlled loop**.	It is an **exit controlled loop**.
The **number of iterations** is based on the loop **condition**.	The loop will be executed at least **once**, even before the execution of the **condition**.
The **condition** check happens at the **start** of the loop.	The **condition** check happens at the **end** of the loop.

Nested Loops

Topics Covered:

- What are nested loops?
- Algorithm of nested loops.
- Purpose and advantages of nested loops.
- How to create a star pattern using nested loops?

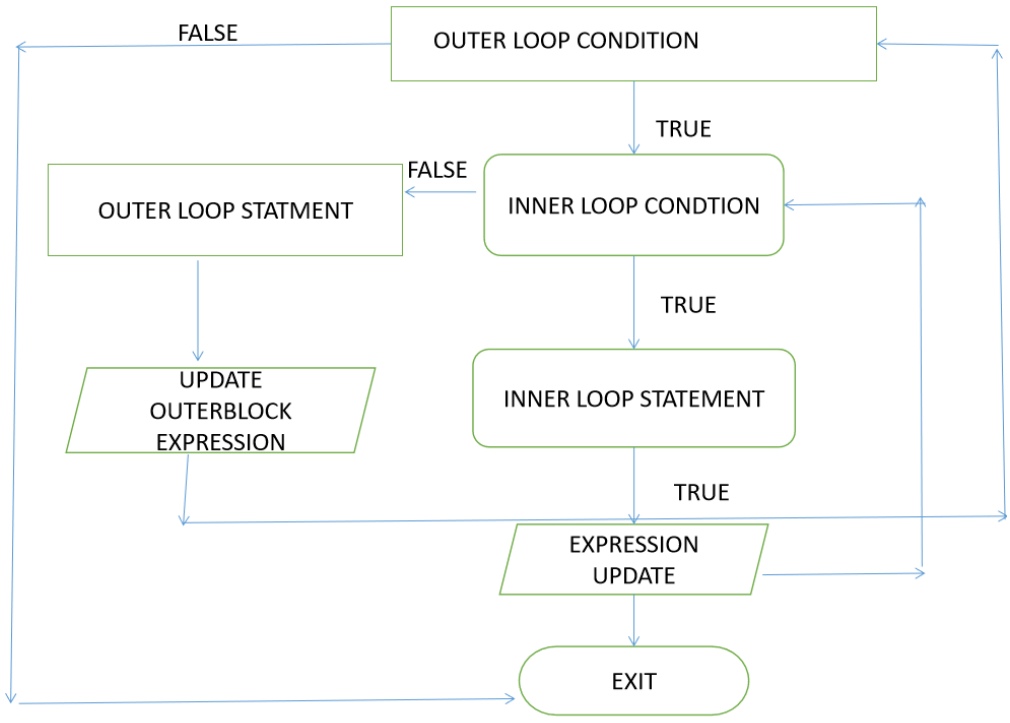
Topics in Detail:

Nested Loops:

- A nested loop is when a **loop** runs inside another **loop**.
- The inner loop will run all its iterations for *each* iteration of the outer loop.

```
Outerloop{
    Innerloop{
    // statements to execute inside inner loop
    }
    // statements to execute inside outer loop
}
```

Nested Loop Algorithm:



- Firstly, check for the outer-loop condition if the condition satisfies or is true, then control goes to the inner loop.
- if the inner-loop condition is true, the inner loop statement gets executed and its expression gets updated.
- if the inner loop condition is false, control goes to the outer-loop and its expression gets updated and then the normal flow goes on from outer to inner then it exits.
- Time arises when the outer-loop condition becomes false and it gets exited from the iteration.

Purpose and advantages of Nested Loops:

- It is mainly used to print or visualize table, matrix, pattern, or multidimensional array.
- Usage of memory is less.
- Suitable when the number of iterations is more in a program.

Creating Star pattern using Nested Loops:

```
let star=""
let n=5
for(i=0;i<=5;i++)
{
    for(j=0;j<i;j++){

        star+="*"
    }
    star+="\n"
}
console.log(star)
```

Code

```
*
**
***
****
*****
```

Output

Nested loops - Practice code

Problem Statement 1: Nested For Loops

Create a JavaScript program that displays the value of inner for loop for each outer iteration, as well as outer for loop.

```
Nested For loop

1
1 2 3 4
2
1 2 3 4
3
1 2 3 4
```

Problem Statement 2: Tables using nested for loop

Print tables using nested for loop.

```
Tables:
1 * 1 = 1          2 * 1 = 2
1 * 2 = 2          2 * 2 = 4
1 * 3 = 3          2 * 3 = 6
1 * 4 = 4          2 * 4 = 8
1 * 5 = 5          2 * 5 = 10
1 * 6 = 6          2 * 6 = 12
1 * 7 = 7          2 * 7 = 14
1 * 8 = 8          2 * 8 = 16
1 * 9 = 9          2 * 9 = 18
1 * 10 = 10        2 * 10 = 20
```

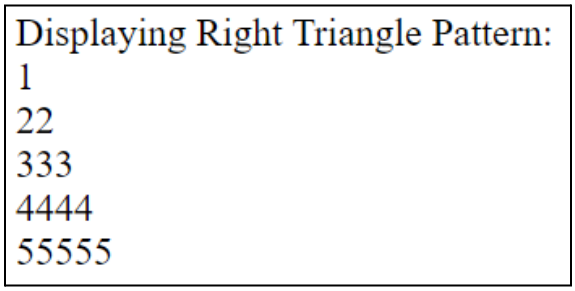
Problem Statement 3: Triangle numbers

Write a JavaScript program to display a triangle of numbers using a nested for loop.

```
Displaying a triangle of numbers:
1
12
123
1234
12345
```

Problem Statement 4: Right Triangle pattern

Create a JavaScript program to display the right triangle pattern of numbers.



Solution Scripts:

Problem Statement 1:

```
// Outer for loop.
  for(var i = 1; i <= 3; i++)
  {
   document.write(i, "<br>"); // will execute 3 times.
// Inner for loop.
  for(var j = 1; j <= 4; j++)
  {
    document.write(j, " "); // will execute 12 (3 * 4) times.
  }
 document.write("<br>");
}
```

Problem Statement 2:

```
 let i, j;
  document.write("Tables:", "<br>");
// Outer for loop.
  for(i = 1; i <= 2; i++)
  {
// Inner for loop.
   for(j = 1; j <= 10; j++)
   {
    document.write(i+ " * " +j+" = "+ (i*j), "<br>");
   }
  document.write("");
 }
```

Problem Statement 3:

```
 let i, j;
  document.write("Displaying a triangle of numbers:", "<br>");
// Outer for loop.
```

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

Problem Statement 4:

```
 let i, j;
   document.write("Displaying Right Triangle Pattern:", "<br>");
// Outer for loop.
   let k = 1;
   for(i = 1; i <= 5; i++)
   {
 // Inner for loop.
    for(j = 1; j <= i; j++)
    {
      document.write(k);
    }
    document.write("<br>");
    k++;
}
```

JavaScript Functions

Topics Covered:

- JavaScript Functions
    - Importance of Functions
- Function Definition
- Function Invocation
- Return Statement
- Local Variable

Topics in Detail:

JavaScript Functions

- JavaScript Functions are a **block of code** used to perform a particular **task**.
- JavaScript Code performing **similar tasks** can be grouped together as **functions** and can be **reused** any number of times.
- Functions can be called from **anywhere** in a program.
- A **big program** can be divided into a **number of small, manageable functions**.

Importance of Functions

- JavaScript Functions help in **code reusability.**
- JavaScript Functions help in creating compact programs with **less number of lines**.

Function Definition

- A **function** keyword is used to **define** a function.
- Function keywords should be followed by a **unique function name,** followed by **parentheses ().**
- Parentheses may include a **list of parameters** separated by comma **(*parameter1, parameter2, ...*)**
- The **code** to be executed should be surrounded by **curly braces**.
- When **invoked,** the Function receives values called **Function Arguments**.
- The arguments or parameters behave as **local variables** to a function.
- Functions can have **zero or any number of parameters**.

- The **Variable naming rule** can be used for **naming functions** as well.

  Syntax

```
function name(parameter1, parameter2, parameter3) {
  // code to be executed
}
```

- This is an example of zero parameters function

```
function sayHello() {
   document.write ("Hello there!");
}
```

Function Invocation

- The code inside a function should be executed by calling/invoking it.

  Syntax

```
functionName(arguments);
```

- We specify **parameters** within the parentheses while **declaring a function**.
- While **calling a function**, we pass **arguments** within the parentheses to the function.
- If **no argument** should be passed, then call the function with **empty** parentheses.
- If calling a function **without parentheses (),** it will return the **function definition** instead of the result.

```
<script>
function sayHello(f) {
  alert("Hello");
}
document.getElementById("demo").innerHTML = sayHello;
</script>
```

The above code will return the function definition itself

```
function sayHello(f) { alert("Hello"); }
```

Return Statement

- By **default,** every function in JavaScript returns **undefined implicitly**.
- The **Return** statement should be the **last statement** in a function, i.e. the function will stop executing.

- If you want to **return a value** from a function, then you should **explicitly** mention a return statement.
- The **return statement without a value** is used to **exit** the function prematurely.
- The **return value** is "**returned**" back to the function **invoking statement**.

Syntax

```
function functionName(parameter1,parameter2)
{
  return value;
}
```

```
function add(a, b) {
    return a + b;
}
```

The above example will return the result of the expression to the actual code calling the function.

Local Variable

- **Variables** declared **inside a function** will behave **local** to that function.
- These **Local variables** can be **accessed only** by that particular **function**.
- The **variables** with the **same name** can be used in **different functions**.
- When the function **starts** the local variables are **created** and when the function **completes** these variables will be **deleted.**

```
// code here can NOT use carName

function myFunction() {
  let carName = "Volvo";
  // code here CAN use carName
}

// code here can NOT use carName
```

Getting Started with Functions - Practice code

Problem Statement 1: Display Text

Display a text in the console window using JavaScript Functions.

```
Hello there!
```

Problem Statement 2: Function with Parameters

Print the username in the console window, using function parameters. Get the user name using the prompt window.

```
Enter a name: Simon
Hello Simon :)
```

Problem Statement 3: Add two numbers

Add two numbers and display the sum in the console window by passing the numbers as arguments of the function.

```
7
11
```

Problem Statement 4: Sum of two numbers

Display the sum of two numbers in the console window by using a function with parameter, function return, and prompt for getting user input.

```
Enter first number: 3.4
Enter second number: 4
The sum is 7.4
```

Solution

Problem Statement 1:

```
// program to print a text
// declaring a function

function greet() {
    console.log("Hello there!");
}

// calling the function
greet();
```

Problem Statement 2:

```
// program to print the text
// declaring a function

function greet(name) {
    console.log("Hello " + name + ":)");
}

// variable name can be different
let name = prompt("Enter a name: ");

// calling function
greet(name);
```

Problem Statement 3:

```
// program to add two numbers using a function
// declaring a function

function add(a, b) {
    console.log(a + b);
}

// calling functions
add(3,4);
add(2,9);
```

Problem Statement 4:

```
// program to add two numbers
// declaring a function

function add(a, b) {
    return a + b;
}

// take input from the user
let number1 = parseFloat(prompt("Enter first number: "));
let number2 = parseFloat(prompt("Enter second number: "));

// calling function
let result = add(number1,number2);

// display the result
console.log("The sum is " + result);
```

JavaScript Function Binding & Closure

Topics Covered:

- Function Binding
- Need for Binding
- this Keyword
- Function Closure
- Closure in Loops

Topics in Detail:

Function Binding

- **Bind()** method is used for **binding a function**.
- **Function Binding** is nothing but **binding a method** from **one object** to **another** object.
- **Bind()** method is used to call a **function** with the **this** value.
- **Bind()** easily sets the **object** to be **bound** with the **this** keyword when the function is **invoked**.

Syntax

```
fn.bind(thisArg[, arg1[, arg2[, ...]]])
```

- **Bind()** will return a **new function** that is the copy of the function **fn.**
- **Binding** that **new function** with that **thisArg** object and arguments **(arg1, arg2, …).**

Need for Binding

- Whenever **this** keyword is **not bound** to an **object**, we need the **Bind()** method for **function binding.**
- **this** will be **lost** when the function is a **callback function**.

```
const employee = {
  firstName:"Bruce",
  lastName: "Lee",
  display: function() {
    let x = document.getElementById("demo");
    x.innerHTML = this.firstName + " " + this.lastName;
  }
}
setTimeout(employee.display, 3000);
```

In the above example, display() method is called back by setTimeout() method.

In case of the callback function, this will be lost, and the result will be as below.

Result

undefined undefined

This can be resolved by bind() function.

```
<script>
const employee = {
  firstName:"Bruce",
  lastName: "Lee",
  display: function() {
    let x = document.getElementById("demo");
    x.innerHTML = this.firstName + " " + this.lastName;
  }
}
let display = employee.display.bind(employee);
setTimeout(display, 3000);
</script>
```

Result

Bruce Lee

- **Bind()** allows an **object** to **borrow** a **method** from **another object** without making a copy of that method.

```
    let animal = {
    name: 'Rabbit',
    run: function(speed) {
        document.write(this.name + ' runs at ' + speed + '
mph.');
    }
};
    let bird = {
    name: 'Eagle',
    fly: function(speed) {
        document.write(this.name + ' flies at ' + speed + '
mph.');
    }
};
    let fn = animal.run.bind(bird, 20);
fn();
```

Result

Eagle runs at 20 mph.

Here, the bird object borrows the run method from the animal object.

this Keyword

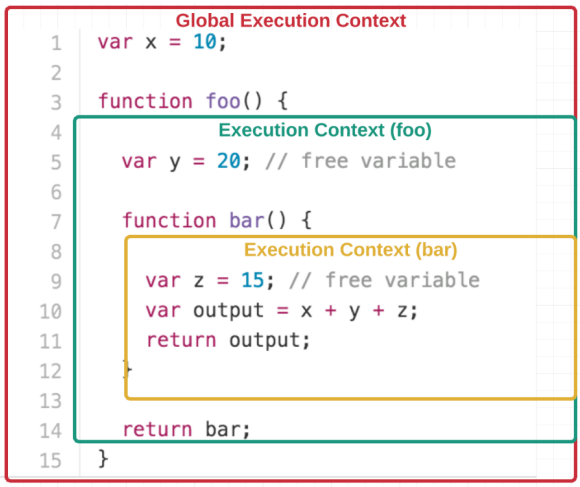
- The **this** keyword refers to an **object**.

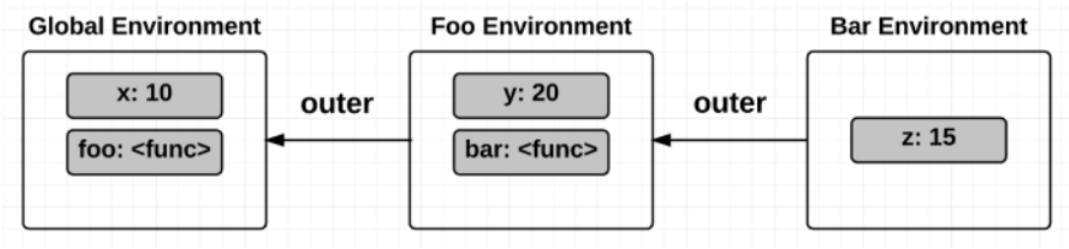
The this keyword refers to different objects depending on how it is used:

Places used	Reference
object method	this refers to the object
this Keyword	this refers to the global object
function	this refers to the global object
function, in strict mode	this is undefined
Event	this refers to the element that received the event
Methods like call(), apply(), and bind()	this refers to any object

Function Closure

- **Closure** is a feature in which an **inner function** can access the **outer function variable**.
- **Closure** is **created** every time with the **creation of the function**.
- Closure **preserves** the **outer scope** within the **inner scope**.
- Scope chains of Closure:
    - Access to its **own scope**.
    - Access to the **variables** of the **outer function**.
    - Access to the **global variables**.
- **Lexical Scoping** defines the **scope of the variable** depending on the **position** of that variable in source code.





Closure in Loops

In the below example, we will see the difficulties while using closure function in loops

```
for (var index = 1; index <= 3; index++) {
    setTimeout(function () {
        console.log('after ' + index + ' second(s):' + index);
    }, index * 1000);
}
```

Actual Output	Expected Output
after 4 second(s):4 after 4 second(s):4 after 4 second(s):4	after 1 second(s):1 after 2 second(s):2 after 3 second(s):3

- Our intention is to display a message in loop after 1, 2 and 3 seconds at the time of each iteration.
- But We see the **same message after 4 seconds** is that the callback passed to the setTimeout() a closure because the JS engine remembers that last iteration value, i.e 4.
- All **three closures** created by the **for-loop** share the **same global scope** and access the **same value of i**.
- To resolve this issue, we have the below solutions.

Solution 1: immediately invoked function expression

```
for (var index = 1; index <= 3; index++) {
    (function (index) {
        setTimeout(function () {
            console.log('after ' + index + ' second(s):' + index);
        }, index * 1000);
    })(index);
}
```

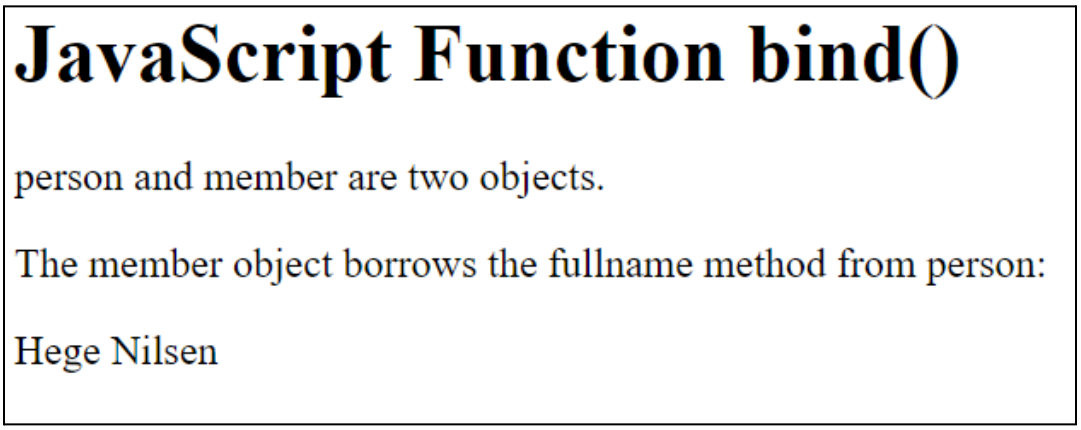
Solution 2: Using let keyword

```
for (let index = 1; index <= 3; index++) {
    setTimeout(function () {
        console.log('after ' + index + ' second(s):' + index);
    }, index * 1000);
}
```

Function Binding and Closure - Practice Code

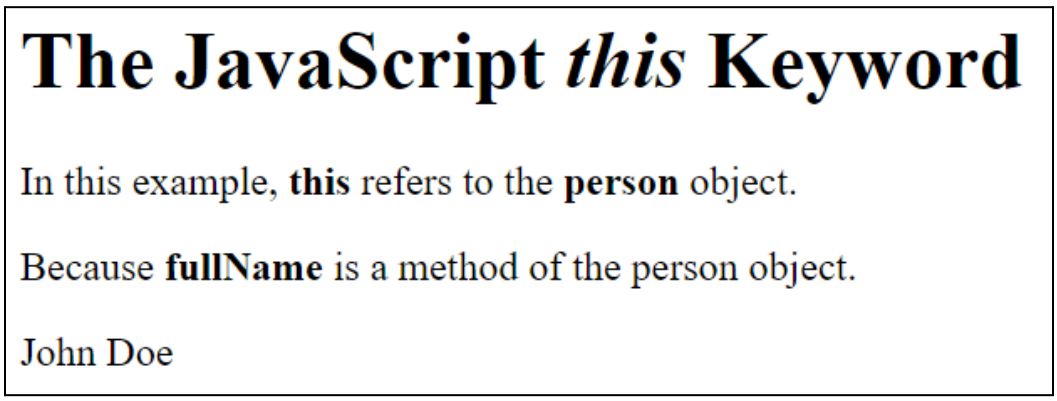
Problem Statement 1: Function borrowing

With the **bind()** method, create two objects and make an object borrow a method from another object.



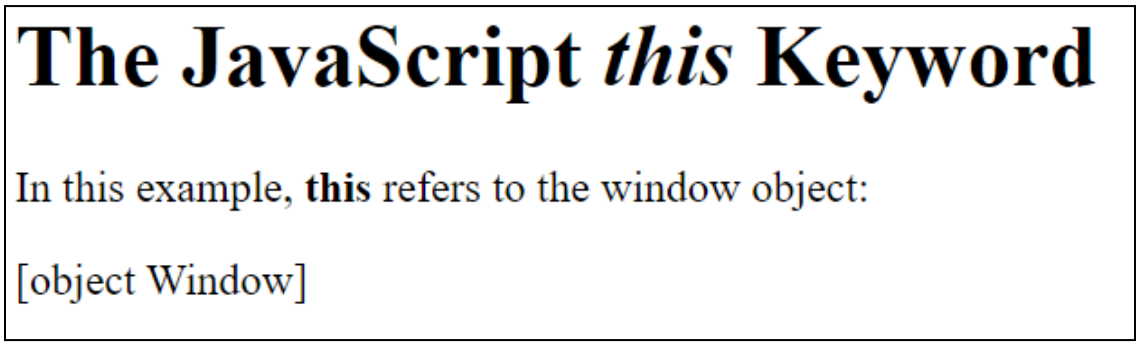
Problem Statement 2: this in method

When used in an object method, **this** refers to the object. Create an object, where you have multiple properties and values, and define a function as one value. With help of **this** keyword access the values of properties in that function. Invoke the function to display the data from the object.



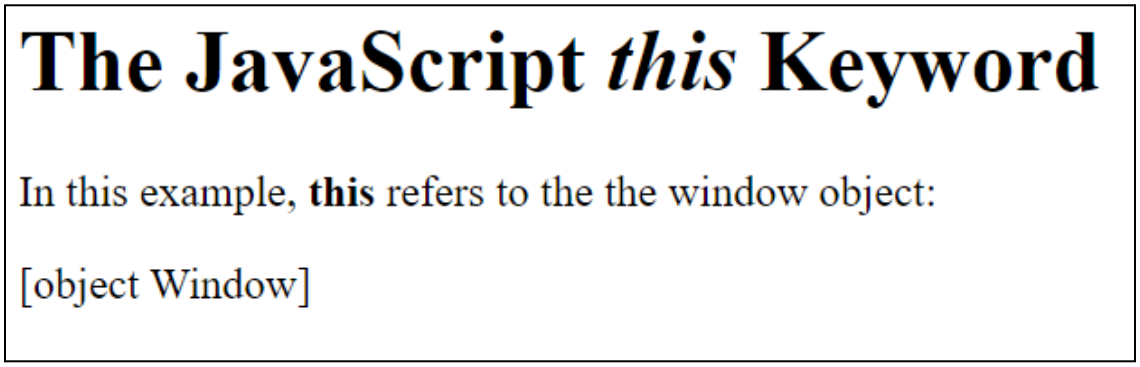
Problem Statement 3: this as global object

When used alone, **this** refers to the global object. In a browser window the global object is **[object Window]**, Display the global object in a browser window by assigning **this** as a value of a variable and print the variable.



Problem Statement 4: this in function (Default)

In a function, the global object is the default binding for **this**. Create a function that returns this. And invoke the function to see what gets printed in the output window.



Solution

Problem Statement 1:

```
<!DOCTYPE html>
<html>
<body>

<h1>JavaScript Function bind()</h1>

<p>person and member are two objects.</p>
<p>The member object borrows the fullname method from person:</p>

<p id="demo"></p>
```

```
<script>
const person = {
  firstName:"John",
  lastName: "Doe",
  fullName: function() {
    return this.firstName + " " + this.lastName;
  }
}

const member = {
  firstName:"Hege",
  lastName: "Nilsen",
}

let fullName = person.fullName.bind(member);

document.getElementById("demo").innerHTML = fullName();
</script>

</body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html>
<body>

<h1>The JavaScript <i>this</i> Keyword</h1>
<p>In this example, <b>this</b> refers to the <b>person</b> object.</p>
<p>Because <b>fullName</b> is a method of the person object.</p>

<p id="demo"></p>

<script>
// Create an object:
const person = {
  firstName: "John",
  lastName: "Doe",
  id: 5566,
  fullName : function() {
    return this.firstName + " " + this.lastName;
  }
};

// Display data from the object:
document.getElementById("demo").innerHTML = person.fullName();
</script>

</body>
</html>
```

Problem Statement 3:

```
<!DOCTYPE html>
<html>
<body>

<h1>The JavaScript <i>this</i> Keyword</h1>

<p>In this example, <b>this</b> refers to the window object:</p>

<p id="demo"></p>

<script>
let x = this;
document.getElementById("demo").innerHTML = x;
</script>

</body>
</html>
```

Problem Statement 4:

```
<!DOCTYPE html>
<html>
<body>

<h1>The JavaScript <i>this</i> Keyword</h1>

<p>In this example, <b>this</b> refers to the the window object:</p>
<p id="demo"></p>

<script>
document.getElementById("demo").innerHTML = myFunction();

function myFunction() {
  return this;
}
</script>

</body>
</html>
```

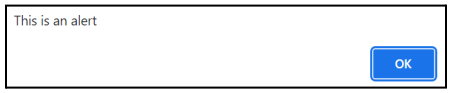
Event Driven Functions - Practice Code

Problem Statement 1: Onclick event

Create a function that displays something and invoke the function on the click event of the button.

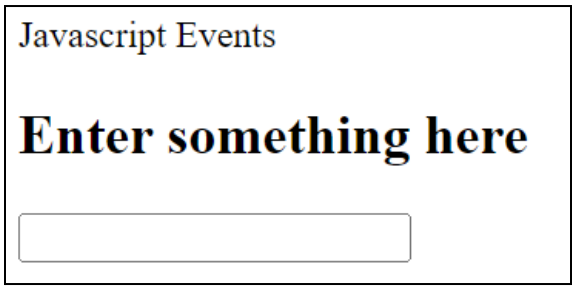
Problem Statement 2: Mouse over event

Create a function that throws an alert, and invoke the function on the mouseover event of the paragraph.



Problem Statement 3: Focus event

Apply styles to the input text box element on the focus event with help of a function.



Problem Statement 4: Keydown event

On the keydown event of an input text box display an alert window with help of function.

Solution:

Problem Statement 1:

```
<html>
<h2> Javascript Events </h2>
<body>
<form>
<input type="button" onclick="clickevent()" value="Who's this?"/>
</form>
<script>

      function clickevent()
      {
            document.write("This is JS Click event");
      }

</script>
</body>
</html>
```

Problem Statement 2:

```
<html>
<head>
<h1> Javascript Events </h1>
</head>
<body>

<p onmouseover="mouseoverevent()"> Keep cursor over me</p>
<script>

    function mouseoverevent()
    {
        alert("This is an alert");
    }

</script>
```

```
</body>
</html>
```

Problem Statement 3:

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onfocus="focusevent()"/>
<script>

    function focusevent()
    {
        document.getElementById("input1").style.background=" aqua";
    }

</script>
</body>
</html>
```

Problem Statement 4:

```
<html>
<head> Javascript Events</head>
<body>
<h2> Enter something here</h2>
<input type="text" id="input1" onkeydown="keydownevent()"/>
<script>
    function keydownevent()
    {
        alert("Pressed a key");
    }
</script>
</body>
</html>
```

JavaScript Event Driven Function

Topics Covered:

- JavaScript Events
- Mouse Events
- Keyboard Events
- Document Events
- Form Events
- Example

Topics in Detail:

JavaScript Events

- When the **user manipulates a page**, there will be a **change in the state of the object** known as **Event**.
- JavaScript **interacts** with the HTML. This interaction is **handled by Events**.
- The process of **handling the interaction** of **JavaScript** with the **HTML** is called **Event Handling**.
- The Events are **handled** via **Event Handlers**.
- **Event handler** is a **block of code** which will be **executed** when an **event** occurs.
- Event handlers are also known as **Event Listeners**.
- These events help JavaScript in providing a **dynamic interface** to a **webpage**.

Syntax

```
<element event='some JavaScript'>
```

Example

```
<button onclick="document.getElementById('demo').innerHTML=Date()">The time is?</button>
```

Mouse Events

Event	Event Handler	Description

click	onclick	When mouse click on an element
		HTML
		`<element onclick="myScript">`
		JavaScript
		`object.onclick = function(){myScript};`

mouseover	onmouseover	When the cursor of the mouse comes over the element
		HTML
		`<element onmouseover="myScript">`
		JavaScript
		`object.onmouseover = function(){myScript};`

mouseout	onmouseout	When the cursor of the mouse leaves an element
		HTML
		`<element onmouseout="myScript">`
		JavaScript
		`object.onmouseout = function(){myScript};`

mousedown	onmousedown	When the mouse button is pressed over the element
		HTML
		`<element onmousedown="myScript">`
		JavaScript
		`object.onmousedown = function(){myScript};`

mouseup	onmouseup	When the mouse button is released over the element
		HTML
		`<element onmouseup="myScript">`
		JavaScript
		`object.onmouseup = function(){myScript};`

mousemove	onmousemove	When the mouse movement takes place
		HTML
		`<element onmousemove="myScript">`
		JavaScript
		`object.onmousemove = function(){myScript};`

Keyboard Events

Event	Event Handler	Description

Keyup	onkeyup	When the user press up key and then releases it
		HTML
		`<element onkeyup="myScript">`
		JavaScript
		`object.onkeyup = function(){myScript};`

Keydown	onkeydown	When the user press down key and then releases it
		HTML
		`<element onkeydown="myScript">`
		JavaScript
		`object.onkeydown = function(){myScript};`

Window/ Document Events

Event	Event Handler	Description

load	onload	When the browser finishes the loading of the page
		HTML
		`<element onload="myScript">`
		JavaScript
		`object.onload = function(){myScript};`

unload	onunload	When the visitor leaves the current page, the browser unloads it
		HTML
		`<element onunload="myScript">`
		JavaScript
		`object.onunload = function(){myScript};`

resize	onresize	When the visitor resizes the window of the browser
		HTML
		`<element onresize="myScript">`
		JavaScript
		`object.onresize = function(){myScript};`

Form Events

Event	Event Handler	Description

focus	onfocus	When the user focuses on an element
		HTML
		`<element onfocus="myScript">`
		JavaScript
		`object.onfocus = function(){myScript};`

submit	onsubmit	When the user submits the form
		HTML
		`<element onsubmit="myScript">`
		JavaScript
		`object.onsubmit = function(){myScript};`

blur	onblur	When the focus is away from a form element
		HTML
		`<element onblur="myScript">`
		JavaScript
		`object.onblur = function(){myScript};`

change	onchange	When the user modifies or changes the value of a form element
		HTML
		`<element onchange="myScript">`
		JavaScript
		`object.onchange = function(){myScript};`

Example

- **Onload Event**
  - The onload event occurs when an object has been loaded.
  - The onload event can be used in HTML and in JavaScript.
  - **HTML:**

```
<element onload="myScript">
```

  - **JavaScript:**

```
object.onload = function(){myScript};
```

- **Example:**

```
window.onload = function () {
  //script to be executed
}
```

- **Onclick event:**
  - The on-click event occurs when the user clicks on an element.
  - The onclick event can be used in HTML and JavaScript.
  - **HTML:**

```
<element onclick="myScript">
```

  - **JavaScript:**

```
object.onclick = function(){myScript};
```

  - **Example:**

```
buttonStart.onclick = function(){
// script to be executed
}
```

JavaScript Arrays

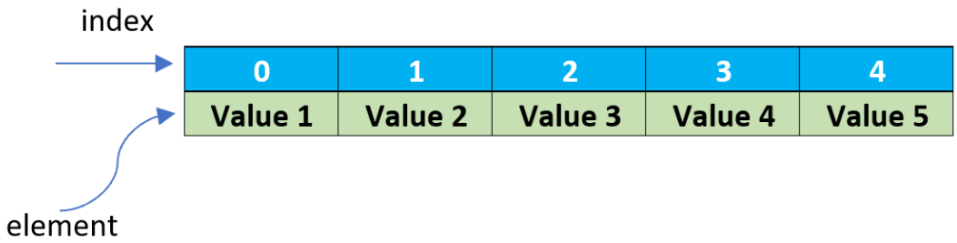
Topics Covered:

- JavaScript Arrays
- Need for Array
- Creating an Array
- Accessing Array Elements
- Length property of an Array
- Array Methods
- Looping Array Elements

Topics in Detail:

JavaScript Arrays

- **An array** is a single variable holding a **list of elements**.
- Each **element** of the array is referenced by the **index**.
- Each **element** in the list can be **individually accessed**.
- The array can hold **mixed types of values**.
- The **size** of the array is **dynamic** and **auto-growing**. So it is not necessary to mention the array size explicitly.



Need for an Array

- If we are supposed to work with many items, say 100 or more. It will be difficult for us to declare each item, but arrays will help us in this situation.
- We can store **many items** under a **single variable name**.
- **Values** can be **accessed** by referring to the **index number**.

Creating an Array

- The Array can be created in three ways
    - By array literal
    - By creating an instance of Array directly (using a new keyword)
    - By using an Array constructor (using a new keyword)

- **JavaScript array literal**

Syntax

```
var arrayname=[value1,value2.....valueN];
```

Example

```
var emp=["Sonoo","Vimal","Ratan"];
```

- **JavaScript Array directly (new keyword)**

Syntax

```
var arrayname=new Array();
```

The **new keyword** is used to create an **instance of an array**.

Example

```
var emp = new Array();
emp[0] = "Arun";
emp[1] = "Varun";
emp[2] = "John";
```

- **JavaScript array constructor (new keyword)**

The instance of the array is created by **passing arguments** to the **constructor** instead of providing the elements explicitly.

Syntax

```
var arrayname = new Array(value1, value2,...valueN);
```

Example

```
var emp=new Array("Jai","Vijay","Smith");
```

- Among these ways, creating an array by using an **array literal** is the **easiest way** to create a JavaScript Array.
- **'const'** Keyword is commonly used to **declare an array**.

Accessing Array Elements

- We can access the **array elements** using **array indexes**.
- Array indexes always start with zero.

Syntax

```
arrayName[index]
```

Example

```
let car = cars[0];
```

- We can access the **full array** simply by referring to the **array name**.

Example

```
const cars = ["Saab", "Volvo", "BMW"];
document.getElementById("demo").innerHTML = cars;
```

Length Property of an array

- The **length of an array** or the **number of elements** in an array can be returned from the **length property** of an array.
- The length property will always return **one plus the highest array index**. Since the array **index starts from zero**.

Example

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
let length = fruits.length;
```

The above example code will give **4** as output.

Array Methods

JavaScript has lots of **built-in array methods**. Some important methods are listed below

Methods	Description
push()	It **adds elements** to the **end** of an array.
pop()	It **removes** and returns the **last element** of an array.
shift()	It **removes** and returns the **first element** of an array.
unshift()	It **adds elements** in the **beginning** of an array.
concat()	It returns a **new array** object that contains **merged arrays**.
sort()	It returns the **element** of the given array in a **sorted order**.
isArray()	It tests if the **passed value** is an **array**.
indexOf()	It **searches** the specified **element** in the given **array** and returns the **index** of the **first match.**

Looping Array Elements

- Only **for loop** and **array.forEach()** is used to loop through the array.
- In **forEach()** the function is called **once** for **each** element in an array.

For Loop example

```
let arr = ["Apple", "Orange", "Pear"];

for (let i = 0; i < arr.length; i++) {
  alert( arr[i] );
}
```

forEach Loop example

```
let fruits = ["Apple", "Orange", "Plum"];

// iterates over array elements
for (let fruit of fruits) {
  alert( fruit );
}
```

Advantages of For each loop

- **For Each loop** makes the **code shorter** and **easier to understand**.
- **No** need to create an **extra counter variable** in **for each loop,** which will help in **easy debugging**.
- For each loop **automatically stops after iterating** all elements in an array.

Arrays - Practice Code

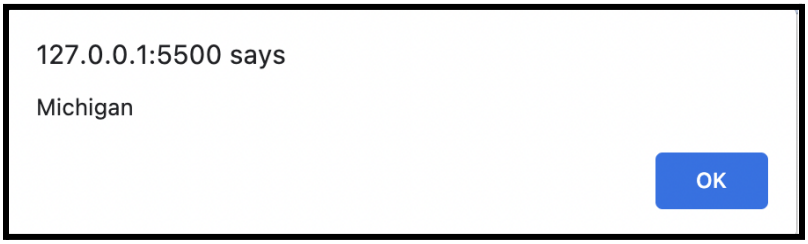
Steps to see the output:

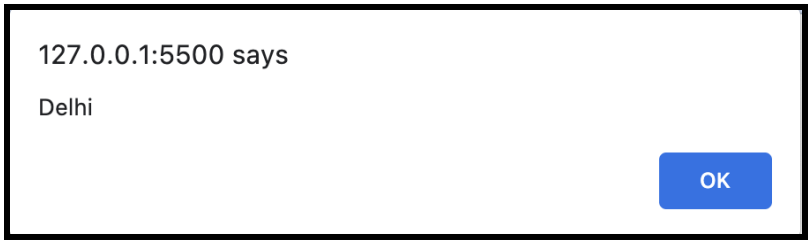
- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.
- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.

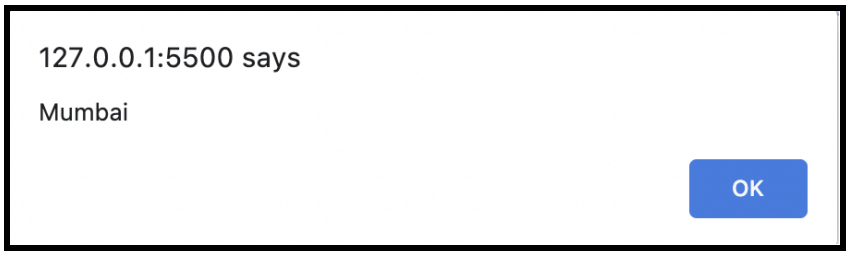
Problem Statement 1: Arrays and 2D Arrays

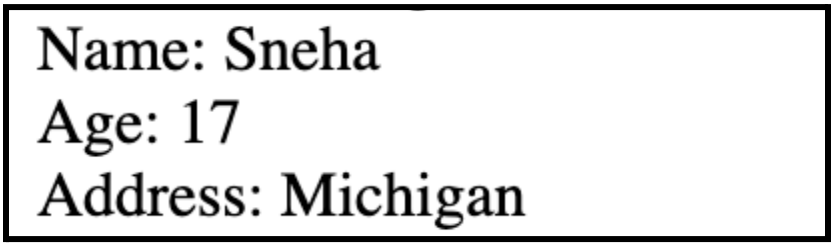
Write a javascript file to store addresses of the places in an array and display it on using alert command, also make use of 2d array and store 3 people names, age, addresses and then display any of the following as an output!

Output:-





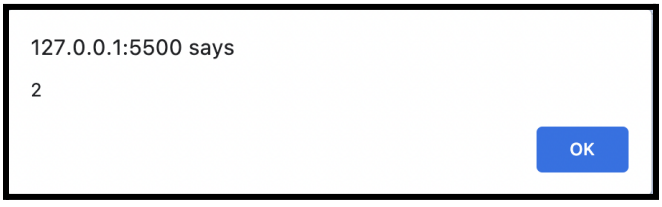




Problem Statement 2: Array Filter

Write a javascript file to store addresses of the people in an array and using filter function print only those addresses that have their length greater than 5, pop an address and then find if pune and delhi exist in the array or not?

Output:-



Solutions

Problem Statement 1:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>JS Arrays Practice Code </title>
</head>
<body>
<script type="text/javascript">
   const addresses = ["London", "Michigan", "Delhi", "Mumbai"];
   for (let addr of addresses){
       alert(addr);
   }
   var personnel = new Array ()
   personnel [0] = new Array ()
   personnel [0] [0] = "Rahul"
   personnel [0] [1] = '18'
   personnel [0][2] = 'London'
   personnel[1] = new Array ()
   personnel [1] [0] = "Sneha"
   personnel [1] [1] = '17'
   personnel [1][2] = 'Michigan'
   personnel[2] = new Array ()
   personnel [2][0] = "Raj"
   personnel [2] [1] = '19'
   personnel [2] [2] = 'Delhi'
   document.write ("Name: " + personnel[1][0] + "<br>")
   document.write ("Age: " + personnel [1] [1] + "<br>")
   document.write ("Address: " + personnel[1][2] + "<br>")
</script>
</body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>JS Arrays Practice Code </title>
</head>
<body>
<script type="text/javascript">
   const addresses = ["London", "Michigan", "Delhi", "Mumbai"];
   function isLong(addr) {
       return addr.length > 5;
   }
   const longer = addresses.filter(isLong);
```

```
    document.write(longer + "<br>");
    addresses.pop();
    document.write(addresses);
    alert(addresses.indexOf('Pune'));
    alert(addresses.indexOf('Delhi'));
</script>
</body>
</html>
```

JavaScript Objects - Practice Code

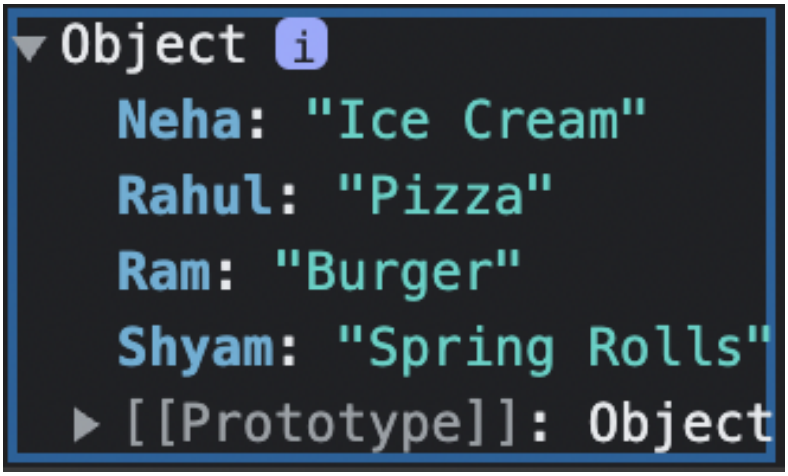
Steps to see the output:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.
- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.

Problem Statement 1: 2D Arrays to Objects

Write a javascript file to convert a two-dimensional array into an object in JavaScript, print it on the console window of your browser?

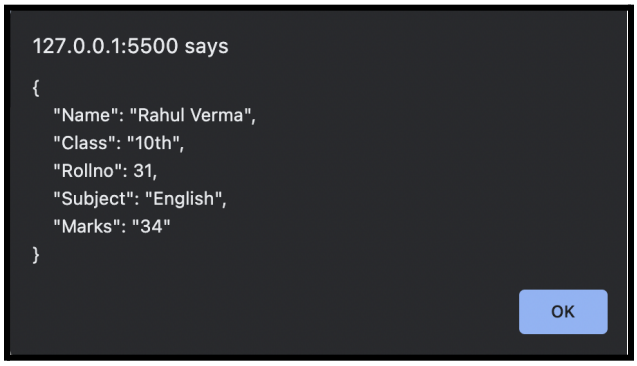
Output:-

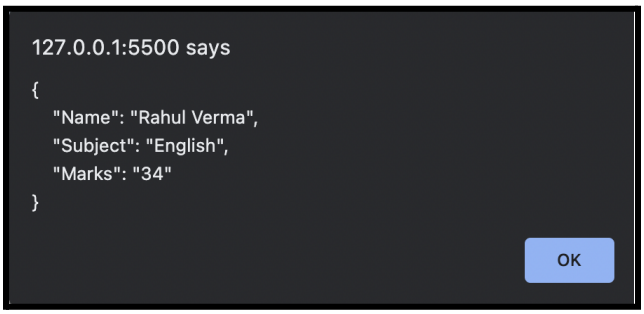


Problem Statement 2: Delete Property

Write a javascript file to create a student object of any 5 properties and then delete the 2 properties from the student object. Also console/alert the object output before or after deleting the property.

Output:-

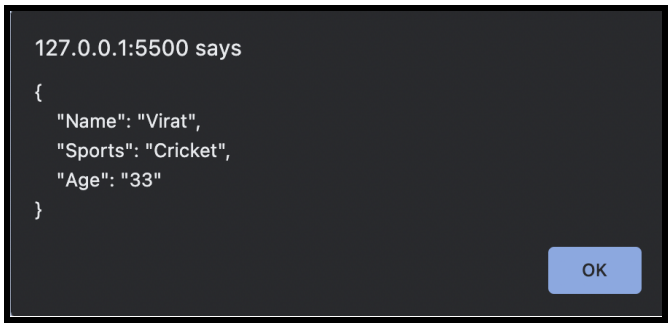




Problem Statement 3: Swap Values and Keys

Write a javascript file to get a copy of the object where the keys have become the values and the values the keys.

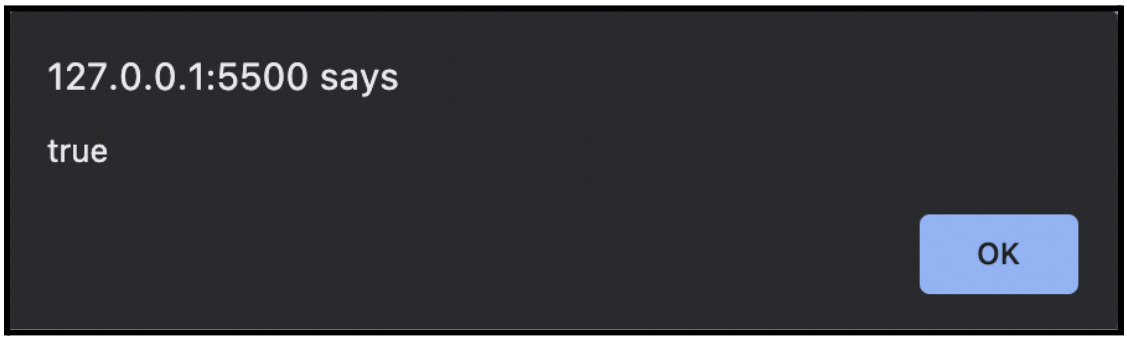
Output:-



Problem Statement 4: Search for a given property (key)

Write a javascript file to create a new object and then check whether an object contains any given property.

Output:-



Solutions

Problem Statement 1:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>JS Objects Practice Code </title>
</head>
<body>
    <h1>Output is in your console window</h1>
<script type="text/javascript">
   function arr2obj(arr) {

      // Create an empty object
      let obj = {};

      arr.forEach((v) => {
            // Extract the key and the value
            let key = v[0];
            let value = v[1];
            // Add the key and value to the new object
            obj[key] = value;
      });
      // Return the object
      return obj;
}

console.log(
      arr2obj([
            ["Ram","Burger" ],
            ["Shyam", "Spring Rolls"],
            ["Rahul", "Pizza"],
            ["Neha", "Ice Cream"],
      ])
);
</script>
</body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>JS Objects Practice Code </title>
</head>
<body>
    <h1>Output is in your console window</h1>
<script type="text/javascript">
```

```
    var student =  {
        Name : "Rahul Verma",
        Class : "10th",
        Rollno : 31,
        Subject : 'English',
        Marks : '34'
      };
      console.log(student);
      alert(JSON.stringify(student, null, 4));
      delete student.Class;
      delete student.Rollno;
      console.log(student);
      alert(JSON.stringify(student, null, 4));
</script>
</body>
</html>
```

Problem Statement 3:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>JS Objects Practice Code </title>
</head>
<body>
    <h1>Output is in your console window</h1>
<script type="text/javascript">
    function invert_key_value(obj) {
        var result = {};
        var keys = _keys(obj);
        for (var i = 0, length = keys.length; i < length; i++) {
          result[obj[keys[i]]] = keys[i];
        }
        return result;
      }
    function _keys(obj)
      {
        if (!isObject(obj)) return [];
        if (Object.keys) return Object.keys(obj);
        var keys = [];
        for (var key in obj) if (_.has(obj, key)) keys.push(key);
        return keys;
      }
    function isObject(obj)
     {
        var type = typeof obj;
        return type === 'function' || type === 'object' && !!obj;
      }
    let obj = {Name: "Virat", Sports: "Cricket", Age: "33"};
    console.log(obj);
    alert(JSON.stringify(obj,null,4));
    let newobj=invert_key_value(obj);
```

```
    alert(JSON.stringify(newobj,null,4));
    console.log(newobj);
</script>
</body>
</html>
```

Problem Statement 4:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>JS Objects Practice Code </title>
</head>
<body>
    <h1>Output is in your console window</h1>
<script type="text/javascript">
    function hasKey(obj, key) {
        return obj != null && hasOwnProperty.call(obj, key);
      }
    let obj={Virat: "33", Rohit: "35", Dhoni: "41"};

    alert(hasKey(obj,"Virat"));
    console.log(hasKey(obj,"Virat"));
</script>
</body>
</html>
```

JavaScript Objects

Topics Covered:

- JavaScript Objects
- Object Properties
- Object Methods
- Accessing object properties
- Accessing object methods
- this Keyword
- Constructors
- Object Maps

Topics in Detail:

JavaScript Objects

- **JavaScript** is an **Object-based** programming language.
- **JavaScript Objects** are a collection of key-value pairs.
- The **Key** of the property is a **string** and the **value** of the property can have **any value,** even a function.
- An **object** is a **reference data type**.
- **Objects** are the **building blocks** of JavaScript.

Example: Key name and **value are** separated by a **colon (:)**

```
const person = {firstName:"John", lastName:"Doe", age:50, eyeColor:"blue"};
```

Object Properties

- In JavaScript, the **named variables** are called **properties**.
- Object **properties** are **variables** that are used **internally** in the **methods** of objects.
- These properties can also be **globally visible**.
- Considering the above example
    - **firstName, lastName, age, and eyeColor** are **keys.**
    - **John, Doe, 50, and blue** are **values**.
- Each of these **key-value** pairs is a **property** of the object.

Object Methods

- The Object with the **function** as a member is known as **Object Methods**.
- Object **methods** are functions that allow **objects to do something**.
- **Methods** are always **attached** to an **object** and are **referenced** by **this** keyword.

```
let school = {
    name: 'Vivekananda School',
    location : 'Delhi',
    established : '1971',
    displayInfo : function(){
        console.log(`${school.name} was established
              in ${school.established} at ${school.location}`);
    }
}
school.displayInfo();
```

- If property names are **numbers**, they can be accessed using the **bracket notation** as follows

```
let school = {
    name: 'Vivekananda School',
    location : 'Delhi',
    established : '1971',
    20 : 1000,
    displayInfo : function(){
        console.log(`The value of the key 20 is ${school['20']}`);
    }
}
school.displayInfo();
```

- Object **Properties** that are **inherited** from an **object's prototype** are known as **inherited properties** of that object. **hasOwnproperty** method can be used to check whether that property is the **object's own property**.

```
const object1 = new Object();
object1.property1 = 42;

console.log(object1.hasOwnProperty('property1'));
```

Accessing Object Properties

- Object's properties can be accessed in two ways
    - **The Dot Notation (.)**

Syntax

```
(objectName.memberName)
```

- **The Array-Like Notation ([ ])**

Syntax

```
objectName["memberName"]
```

Accessing Object Methods

- Object's methods can be accessed as follows

Syntax

```
objectName.methodName()
```

- Object's method when invoked **with ( )** the **method** will be **executed**.
- Object's method, when accessed **without ( )** the **function definition,** will be **returned**.

'this' Keyword

- The **'this' keyword** refers to an object.
- The value of **this** cannot be changed.
- In the **function** definition, **this** refers to the **owner of the function**.

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

this.firstName means the **firstName** property of a **person** object.

The ‘this’ keyword refers to different objects depending on how it is used:

Places used	Reference
object method	this refers to the object
this Keyword	this refers to the global object
function	this refers to the global object
function, in strict mode	this is undefined
Event	this refers to the element that received the event
Methods like call(), apply(), and bind()	this refers to any object

Object Constructors

- The **object constructor function** is used to create an **object type**.
- In the below example,

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
}
```

- **Function Person()** is an object constructor function.
- By calling the **object constructor function** with the **new keyword,** we can create the objects of the same type.

```
const myFather = new Person("John", "Doe", 50, "blue");
const myMother = new Person("Sally", "Rally", 48, "green");
```

- To add a **new property or method** to a constructor, first **add it** to the **constructor function** as follows

```
function Person(first, last, age, eye) {
  this.firstName = first;
  this.lastName = last;
  this.age = age;
  this.eyeColor = eye;
  this.nationality = "English";
  this.name = function() {
    return this.firstName + " " + this.lastName
  };
}
```

Built-in JavaScript Constructors

```
new String()
new Number()
new Boolean()
new Object()
new Array()
new RegExp()
new Function()
new Date()
```

Object Maps

- A Map has **key-value** pairs. The key can be of any datatype.
- Map has a property to represent the **size of the map**.

Methods of Object Map

Method	Description
new Map()	Creates a new Map object
set()	Sets the value for a key in a Map
get()	Gets the value for a key in a Map
clear()	Removes all the elements from a Map
delete()	Removes a Map element specified by a key
has()	Returns true if a key exists in a Map

Method	Description
forEach()	Invokes a callback for each key/value pair in a Map
entries()	Returns an iterator object with the [key, value] pairs in a Map
keys()	Returns an iterator object of the keys in a Map
values()	Returns an iterator object of the values in a Map

Properties of Object Map

Property	Description
size	Returns the number of Map elements

JavaScript Arrays II

Topics Covered:

- Multidimensional Array
    - Creating multidimensional Array
    - Accessing multidimensional Array
    - Adding elements to multidimensional Array
    - Removing elements from multidimensional Array
- Referential Array
    - Array as Objects
    - Array Methods
    - Array Properties

Topics in Detail:

Multidimensional Array

- Multidimensional Array is also known as **array of array**.
- JavaScript **does not** have inbuilt **support** for **multi-dimensional arrays**.
- So we need to **create an array inside an array** to make it work as a Multidimensional Array.

Create a multidimensional Array

```
1st, need to define some 1D array
var arr1 = ["ABC", 24, 18000];
var arr2 = ["EFG", 30, 30000];
var arr3 = ["IJK", 28, 41000];
var arr4 = ["EFG", 31, 28000];
var arr5 = ["EFG", 29, 35000];
// "salary" defines like a 1D array but it already contains some 1D array
var salary = [arr1, arr2, arr3, arr4, arr5];
```

OR

```
var salary = [
   ["ABC", 24, 18000],
   ["EFG", 30, 30000],
   ["IJK", 28, 41000],
   ["EFG", 31, 28000],
];
```

Accessing Multidimensional Array elements

- **Simple index-based notation**

```
// This notation access the salary of "ABC" person which is 18000,
// [0] selects 1st row, and [2] selects the 3rd element
// of that 1st row which is 18000
salary[0][2];
```

- **For many iterations**

```
// This loop is for outer array
for (var i = 0, l1 = salary.length; i < l1; i++) {

    // This loop is for inner-arrays
    for (var j = 0, l2 = salary[i].length; j < l2; j++) {

        // Accessing each elements of inner-array
        documents.write( salary[i][j] );
    }
}
```

Adding elements in multidimensional array

There are two ways to add an element in multidimensional array

- Inner Array
- Outer Array

Adding elements to inner Array

- **Square Bracket Notation**

```
salary[3][3] = "India";

// It adds "India" at the 4th index of 4th sub-array,
// If we print the entire 4th sub-array, document.write(salary[3]);
// the output will be :  ["EFG", 31, 28000, "India"]
```

- **push() Method**

```
salary[3].push("India", "Mumbai");

// It add "India" at the 4th index and "Mumbai" at
// 5th index of 4th sub-array
// If we print the entire 4th sub-array,
// document.write(salary[3]);
// The output will be :  ["EFG", 31, 28000, "India", "Mumbai"]
```

Adding elements to outer Array

```
salary.push(["MNO", 29, 33300]);
// This row added after the last row in the "salary" array
```

Removing Elements in Multidimensional Array

- The **pop()** method is used to **remove elements** from the **inner array**.

```
// Remove last element from 4th sub-array
// That is 28000 indexing starts from 0
salary[3].pop();
```

- The **entire inner array** can be **removed** from the outer array with the **pop()** method.

```
// Removes last sub-array
// That is "["EFG", 31, 28000]"
salary.pop();
```

Referential Array

Array as Objects

- Arrays are a **special** type of **object**.
- In Javascript, **typeOf** operator is used to return an **array object**.
- **Numbers** are used to access the **elements of an array**.

```
const person = ["John", "Doe", 46];
document.getElementById("demo").innerHTML = person[0];
```

- **Names** are used to access the **members of an object**.

```
const person = {firstName:"John", lastName:"Doe", age:46};
document.getElementById("demo").innerHTML = person.firstName;
```

- An array can have different types of variables.

  I.e Array can have arrays

  Array can have functions

  Array can have objects

```
myArray[0] = Date.now;
myArray[1] = myFunction;
myArray[2] = myCars;
```

- In Javascript, arrays with **named indexes** are **not supported**.
- If we use **named indexes** in javascript, arrays will be **redefined as objects**.

JavaScript Array Methods

Method	Description
concat()	Joins two or more arrays and returns a copy of the joined arrays
copyWithin()	Copies array elements within the array, to and from specified positions
entries()	Returns a key/value pair Array Iteration Object
every()	Checks if every element in an array pass a test
fill()	Fill the elements in an array with a static value
filter()	Creates a new array with every element in an array that pass a test
find()	Returns the value of the first element in an array that pass a test
findIndex()	Returns the index of the first element in an array that pass a test
forEach()	Calls a function for each array element
from()	Creates an array from an object
includes()	Check if an array contains the specified element
indexOf()	Search the array for an element and returns its position
isArray()	Checks whether an object is an array
join()	Joins all elements of an array into a string
keys()	Returns a Array Iteration Object, containing the keys of the original array
lastIndexOf()	Search the array for an element, starting at the end, and returns its position

Method	Description
map()	Creates a new array with the result of calling a function for each array element
pop()	Removes the last element of an array, and returns that element
push()	Adds new elements to the end of an array, and returns the new length
reduce()	Reduce the values of an array to a single value (going left-to-right)
reduceRight()	Reduce the values of an array to a single value (going right-to-left)
reverse()	Reverses the order of the elements in an array
shift()	Removes the first element of an array, and returns that element
slice()	Selects a part of an array, and returns the new array
some()	Checks if any of the elements in an array pass a test
sort()	Sorts the elements of an array
splice()	Adds/Removes elements from an array
toString()	Converts an array to a string, and returns the result
unshift()	Adds new elements to the beginning of an array, and returns the new length
valueOf()	Returns the primitive value of an array

JavaScript Array Properties

Property	Description
constructor	Returns the function that created the Array object's prototype
length	Sets or returns the number of elements in an array
prototype	Allows you to add properties and methods to an Array object

Advanced Arrays- Practice Code

Steps to see the output:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.
- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.

Problem Statement 1: What is the output of the following code?

Code:-

```
const arrayOfOddNumbers = [1, 3, 5];
arrayOfOddNumbers[100] = 199;
console.log(arrayOfOddNumbers.length);
```

Problem Statement 2: Write a simple JavaScript program to join all elements of the following array into a string

Sample array : myColor = ["1", "2", "3", "Black"];
Expected Output :
"1,2,3,Black"

Problem Statement 3: Write a JavaScript program to find a pair of elements (indices of the two numbers) from an given array whose sum equals a specific target number.

Input: numbers= [10,20,10,40,50,60,70], target=50

Output: 2, 3

Solutions

Sol 1: 101

The reason for this solution is as follows: JavaScript places empty as a value for indices 3 - 99. Thus, when you set the value of the 100th index, the array looks like:

Sol 2:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>JS Objects Practice Code </title>
</head>
<body>
    <h1>Output is in your console window</h1>
<script type="text/javascript">
    myColor = ["1", "2", "3", "Black"];
    console.log(myColor.toString());
    console.log(myColor.join());
</script>
</body>
</html>
```

Sol 3:

```
function twoSum(nums, target_num) {
  var map = [];
  var indexnum = [];

  for (var x = 0; x < nums.length; x++)
  {
  if (map[nums[x]] != null)
  {
  var index = map[nums[x]];
  indexnum[0] = index;
  indexnum[1] = x;
  break;
  }
  else
  {
  map[target_num - nums[x]] = x;
  }
  }
  return indexnum;
  }
console.log(twoSum([10,20,10,40,50,60,70],50));
```

JSON

Topics Covered:

- JSON
- JSON Data Types
- JSON Parse
- JSON Stringify
- JSON Objects
- JSON Arrays

Topics in Detail:

JSON

- **JSON** is the short form of **JavaScript Object Notation**.
- JSON **stores** and **transports data** in **text format**.
- **Using JSON** we can **send data** between computers.
- JSON is **language independent**.

JSON Data Types

- **Numbers**

  In JSON, Numbers must be either an **integer** or a **floating-point**.

  ```
  {"age":30}
  ```

- **String**

  In JSON, Strings are written **within double quotes**.

  ```
  {"name":"John"}
  ```

- **Boolean**

  In JSON, Boolean values can be **either true or false**.

  ```
  {"sale":true}
  ```

- **Array**

  In JSON, values can be **arrays**.

```
{
"employees":["John", "Anna", "Peter"]
}
```

- **Object**

  In JSON, values can be **objects**.

```
{
"employee":{"name":"John", "age":30, "city":"New York"}
}
```

- **null**

  In JSON, values can be **null**.

```
{"middlename":null}
```

JSON Parse

- The **Data** received from the **web server** is always a **string.**
- **JSON.parse()** method is used to parse the data to a **javascript object**.

```
const txt = '{"name":"John", "age":30, "city":"New York"}'
const obj = JSON.parse(txt);
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;
```

- When **JSON.parse()** is used on an **array**, it will return a **javascript array** instead of a **javascript object**.

```
const text = '[ "Ford", "BMW", "Audi", "Fiat" ]';
const myArr = JSON.parse(text);
document.getElementById("demo").innerHTML = myArr;
```

Exceptions

- **Parsing Date**
  - **JSON** does not allow **date format**.
  - So write it as a **string** and **convert** it as a **date object**.

```
const text = '{"name":"John", "birth":"1986-12-14", "city":"New York"}';
const obj = JSON.parse(text);
obj.birth = new Date(obj.birth);

document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;
```

- **Parsing Functions**
  - **JSON** does not allow **Functions**.
  - So write it as a **string** and **convert** it as a **function**.

```
const text = '{"name":"John", "age":"function () {return 30;}", "city":"New York"}';
const obj = JSON.parse(text);
obj.age = eval("(" + obj.age + ")");

document.getElementById("demo").innerHTML = obj.name + ", " + obj.age();
```

JSON Stringify

- The Data **sent** to the **web server** should be a **string**.
- **JSON.stringify()** method is used to **convert JavaScript objects** into a **string**.

```
const obj = {name: "John", age: 30, city: "New York"};
const myJSON = JSON.stringify(obj);
```

- **JSON.stringify()** methods can also convert **JavaScript Array** into a **string**.

```
const arr = ["John", "Peter", "Sally", "Jane"];
const myJSON = JSON.stringify(arr);
```

- **Storing Data**
    - In **JSON,** JavaScript objects can be stored as text.

```
// Storing data:
const myObj = {name: "John", age: 31, city: "New York"};
const myJSON = JSON.stringify(myObj);
localStorage.setItem("testJSON", myJSON);

// Retrieving data:
let text = localStorage.getItem("testJSON");
let obj = JSON.parse(text);
document.getElementById("demo").innerHTML = obj.name;
```

Exceptions

- **Stringify Date**
    - **JSON** does not allow **date objects**.
    - **Date objects** can be **converted** into **strings** using **JSON.stringify()** method.

```
const obj = {name: "John", today: new Date(), city : "New York"};
const myJSON = JSON.stringify(obj);
```

- **Stringify Functions**
    - **JSON** does not allow **Functions** as object values.
    - **Function** from a **javaScript object** will be **removed** while using **JSON.stringify()** method.
    - **Convert functions** into **string** before **JSON.stringify()** method to include functions.

```
const obj = {name: "John", age: function () {return 30;}, city: "New York"};
obj.age = obj.age.toString();
const myJSON = JSON.stringify(obj);
```

JSON Objects

- There is a **JSON object literal** inside every **JSON string**.
- In JSON, **Object literals** are surrounded by **curly braces {}**.
- These object literals contain **key/value pairs**.
- **A colon** separates the keys and values.
- **Keys** must be a **string**.

- **Values** should always be **valid JSON Datatype**.
- **A comma** separates each key-value pair.

```
{"name":"John", "age":30, "car":null}
```

Creation of JavaScript Objects

- Javascript objects can be created in two ways
  - From a JSON object literal

```
myObj = {"name":"John", "age":30, "car":null};
```

OR

  - By parsing a JSON String

```
myJSON = '{"name":"John", "age":30, "car":null}';
myObj = JSON.parse(myJSON);
```

Accessing Object Values

- Object values can be accessed in two ways
  - **The Dot Notation (.)**

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);
x = myObj.name;
```

  - **The Array-Like Notation ([ ])**

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);
x = myObj["name"];
```

Looping an Object

- **for-in loop** is used to loop through object properties.

```
const myJSON = '{"name":"John", "age":30, "car":null}';
const myObj = JSON.parse(myJSON);

let text = "";
for (const x in myObj) {
  text += x + ", ";
}
```

JSON Arrays

- Arrays in JSON are similar to JavaScript Array and can have values of the following types
    - string
    - number
    - object
    - array
    - boolean
    - null

Creating a JSON Array

- Arrays can be created using a **literal**.

```
myArray = ["Ford", "BMW", "Fiat"];
```

- Arrays can be created by parsing a **JSON string**.

```
myJSON = '["Ford", "BMW", "Fiat"]';
myArray = JSON.Parse(myJSON);
```

Accessing Array Values

- Array values can be accessed by index.

```
myArray[0];
```

Arrays in Objects

- **Objects** can **contain arrays**, these **array values** can be accessed using an **index**.

```
const myJSON = '{"name":"John", "age":30, "cars":["Ford", "BMW", "Fiat"]}';
const myObj = JSON.parse(myJSON);
document.getElementById("demo").innerHTML = myObj.cars[0];
```

Looping through an array

- The values of the entire array can be accessed by using **for** or **for in** loop.

```
const myJSON = '{"name":"John", "age":30, "cars":["Ford", "BMW", "Fiat"]}';
const myObj = JSON.parse(myJSON);
let text = "";
for (let i = 0; i < myObj.cars.length; i++) {
  text += myObj.cars[i] + ", ";
}
```

JSON

Challenge 1:

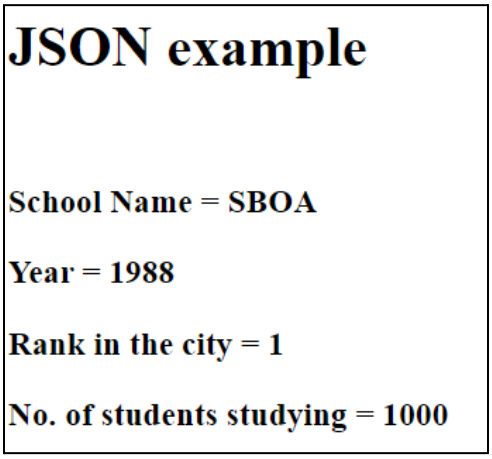
Steps:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width">
  <title>JSON</title>
</head>
<body>
  <h2>JSON</h2>
<p id="demo"></p>
<script>
const myDetails = '{"name":"Prem", "age":50, "car":"Creta"}';
const object = JSON.parse(myDetails);
document.getElementById("demo").innerHTML = object.car;
</script>
</body>
</html>
```

- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.

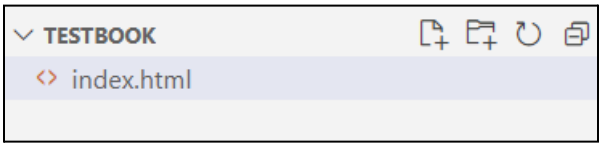
Problem statement:
With your knowledge gained on JSON consider creating an object in javascript listing details of a school giving the following output.



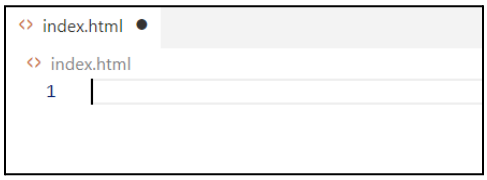
Steps:

- Open VS code.
- Select New File in the opened folder…

- Save the file in the appropriate folder/ create the folder and save the file with extension.



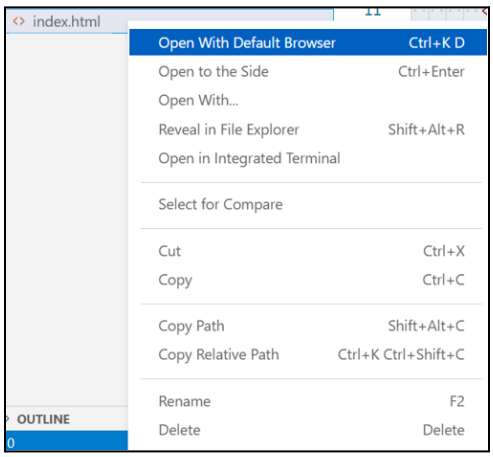
- The file in the VS Code appears with file extension after saving



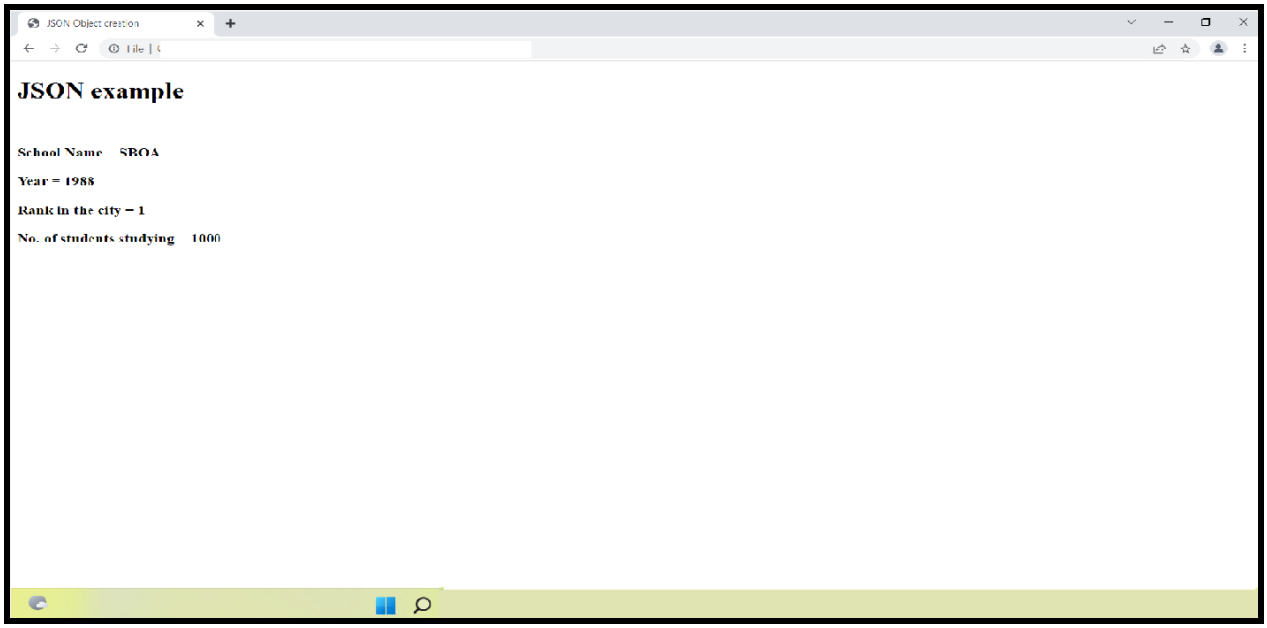
Sample code:

```
<!DOCTYPE html>
<html>
   <head>
      <title>JSON Object creation</title>
      <script>
         var JSONObj = { "name" : "SBOA",
                         "year"  : 1988,
                         "rank" : 1,
                         "students" : 1000 };

         document.write("<h1>JSON example</h1>");
         document.write("<br>");
         document.write("<h3>School Name = "+JSONObj.name+"</h3>");
         document.write("<h3>Year = "+JSONObj.year+"</h3>");
         document.write("<h3>Rank in the city = "+JSONObj.rank+"</h3>");
         document.write("<h3>No. of students studying =
"+JSONObj.students+"</h3>");
      </script>
   </head>
   <body>
   </body>
</html>
```

- View the file in the browser, by right click on the file name in the left pane.



Sample Output:


JSON Object creation
JSON example
School Name SBOA
Year = 1988
Rank in the city = 1
No. of students studying 1000

JavaScript DOM

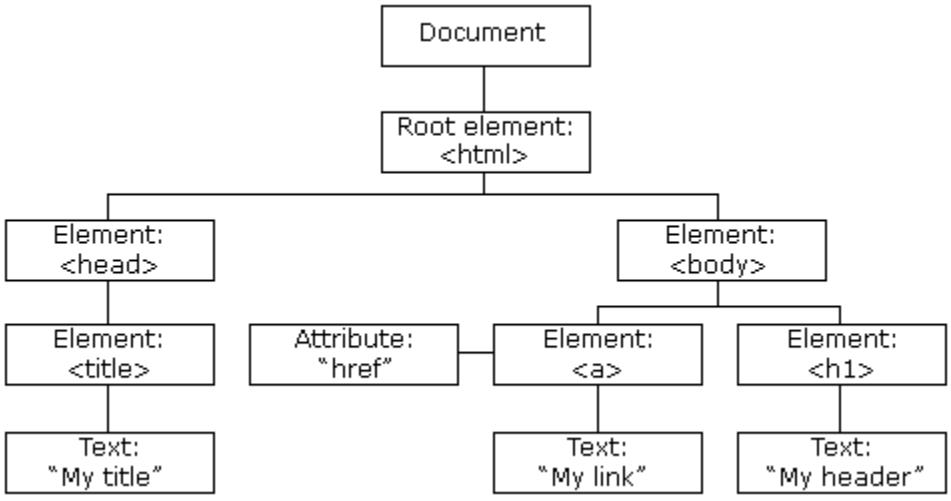
Topics Covered:

- What is DOM?
- DOM Properties.
- DOM Methods.
- Event Listeners.

Topics in Detail:

Document Object Method:

- The **document object** represents the whole html document.
- When an html document is loaded in the browser, it becomes a document object.
- <html> is the **root element** that represents the html document. It has properties and methods.
- By the help of document object, we can add dynamic content to our web page.
- The HTML DOM is a standard object model and programming interface for HTML. It defines:
    - The HTML elements as objects
    - The properties of all HTML elements
    - The methods to access all HTML elements
    - The events for all HTML elements



Properties of Document object:

Property	Description
element.innerHTML = new html content	Change the inner HTML of an element. ***let content = element.innerHTML;***
element.attribute = new value	Change the attribute value of an HTML element. ***var attrs = paragraph.attributes;***
element.style.property = new style	Change the style of an HTML element. ***document.body.style.background = "red";***

DOM Methods:

Methods	Description
write(“string”)	Write the given string on the document. ***document.write("Hello World!");***
getElementById()	Returns the element having the given ID value. ***const element = document.getElementById("intro");***
getElementsByName()	Returns all the elements having the given name value. ***const x = document.getElementsByClassName("main");***
getElementsByTagName()	Returns all the elements having the given tag name. ***const element = document.getElementsByTagName("p");***
getElementsByClassName()	Returns all the elements having the given class name. ***const x = document.getElementsByClassName("intro");***
querrySelectorAll()	Returns all the elements that matches the specified CSS selector(id/class/element/name/type/value of attribute) ***const x = document.querySelectorAll("p.intro");***

Event Listeners:

- The **addEventListener()** method is used to attach an event handler to a particular element.
- It does not override the existing event handlers. Events are said to be an essential part of JavaScript.
- The addEventListener() method is an inbuilt function of JavaScript.
- We can add multiple event handlers to a particular element without overwriting the existing event handlers.
- Syntax:
  element.addEventListener(event, function, useCapture);
- The parameters ***event*** and ***function*** are widely used. The third parameter is optional to define.
  - **event:** It is a required parameter. It can be defined as a string that specifies the event's name.
  - **function:** It is also a required parameter. It is a JavaScript function which is invoked on the event.
  - **useCapture:** It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**.
- Example:

```
myForm.addEventListener("submit",
function(event){
  event.preventDefault()
  createItem(myInput.value)
})

function createItem(inputItems){
  var items = `<li>${inputItems}</li>`
  myItem.insertAdjacentHTML("beforeend", items)
  myInput.value = ""
  myInput.focus()
}
```

JavaScript DOM - Practice Code

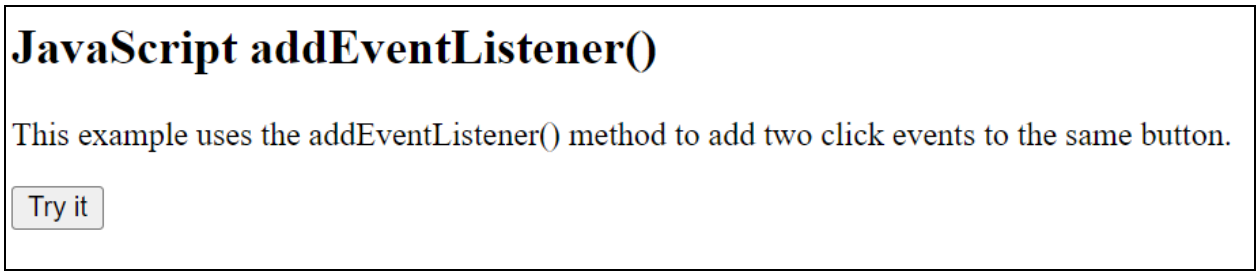
Coding is practice... Practice your JS Code, with help of following steps, which will give you a clear understanding about DOM.

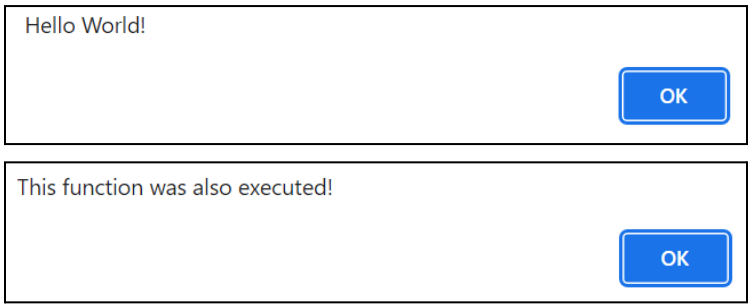
Challenge 1: Events on Click
Steps:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.

```
<!DOCTYPE html>
<html>
<body>
<h2>JavaScript addEventListener()</h2>
<p>This example uses the addEventListener() method to add two click events to
the same button.</p>
<button id="myBtn">Try it</button>
<script>
var x = document.getElementById("myBtn");
x.addEventListener("click", myFunction);
x.addEventListener("click", someOtherFunction);
function myFunction() {
  alert ("Hello World!");
}
function someOtherFunction() {
  alert ("This function was also executed!");
}
</script>
</body>
</html>
```

- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.





Onclick of the button two alert windows pop up

Challenge 2:
With your knowledge gained in DOM, create a web page with a button that invokes different functions in different events.

HTML elements:

- Button with ID → To add events by accessing this from JavaScript.
- <p> with ID → To display the name of the event by accessing this from JavaScript.

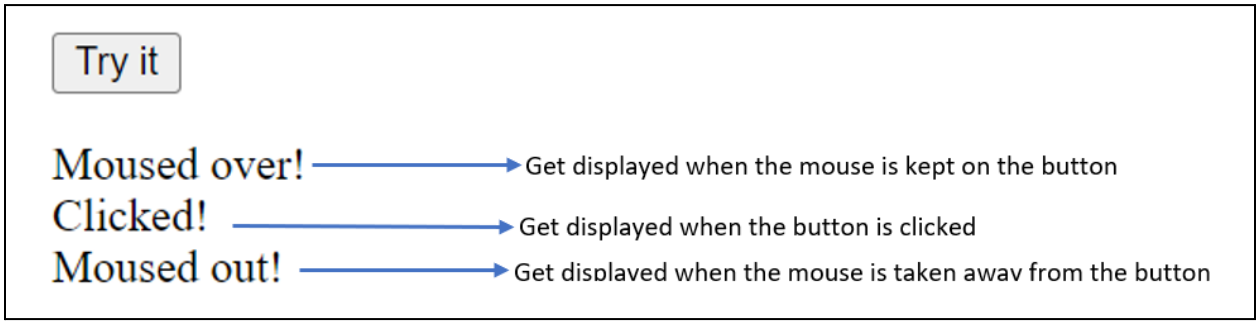
JavaScript:

- Get the instance of the button and store it in a variable using ***getElementById()*** method.
- Create three functions that access the <p> tag in HTML and add the content to the element using the ***.innerHTML()*** method.
- The content should be different for all three functions:

Function	Display content
function1	Display Mouse over
function2	Display Click
function3	Display Mouse out

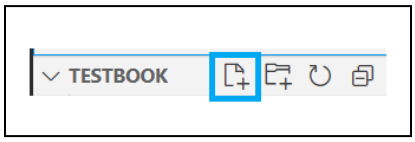
- Add event Listener to the button element with different events and respective functions:

Event	function
mouseover	function1
click	function2
mouseout	function3

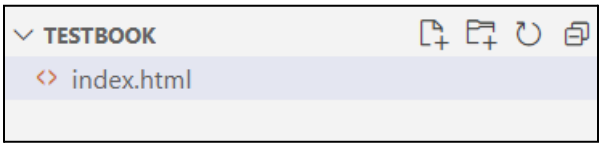


Steps:

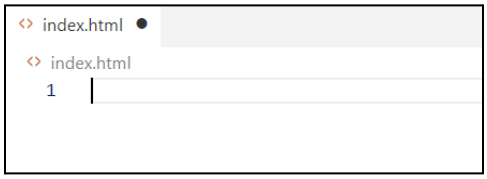
- Open VS code.
- Select New File in the opened folder…



- Save the file in the appropriate folder/ create the folder and save the file with extension.



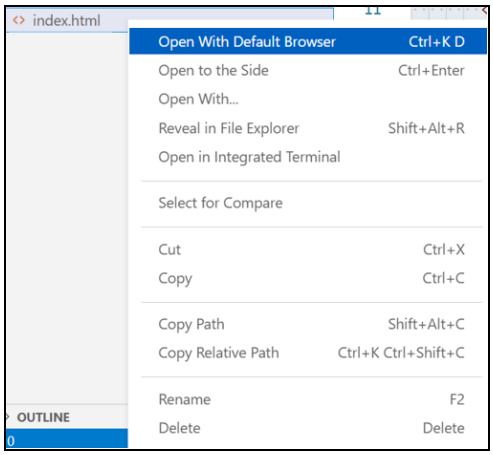
- The file in the VS Code appears with file extension after saving



Sample code:

```
<!DOCTYPE html>
<html>
<body>

<h2>JavaScript addEventListener()</h2>

<p>This example uses the addEventListener() method to add many events on the
same button.</p>

<button id="myBtn">Try it</button>

<p id="demo"></p>

<script>
var x = document.getElementById("myBtn");
x.addEventListener("mouseover", myFunction);
x.addEventListener("click", mySecondFunction);
x.addEventListener("mouseout", myThirdFunction);

function myFunction() {
  document.getElementById("demo").innerHTML += "Moused over!<br>";
}

function mySecondFunction() {
  document.getElementById("demo").innerHTML += "Clicked!<br>";
}

function myThirdFunction() {
  document.getElementById("demo").innerHTML += "Moused out!<br>";
}
</script>

</body>
</html>
```

- View the file in the browser, by right click on the file name in the left pane.



Sample Output:

JavaScript addEventListener()

This example uses the addEventListener() method to add many events on the same button.

Try it

JavaScript addEventListener()

This example uses the addEventListener() method to add many events on the same button.

Try it

Moused over!
Moused out!
Moused over!
Clicked!
Moused out!
Moused over!
Moused out!

JavaScript Timing Events and Callback

Topics Covered:

- Timing Events
- setTimeout() Method
- setInterval() Method
- Function Sequence
- JavaScript CallBack

Topics in Detail:

Timing Events

- At specified time intervals, the **window object** allows **code execution**.
- **Timing Events** are nothing but these **time intervals**.
- The two key methods are
    - ***setTimeout(function, milliseconds)***
    - ***setInterval(function, milliseconds)***

setTimeout() method

The **function** is executed after **waiting** for certain **milliseconds**.

Syntax:

```
window.setTimeout(function, milliseconds);
```

- The **window** prefix can be **omitted.**
- The **first parameter** has the **function** to be executed**.**
- The **second parameter** has the **wait time before execution in milliseconds.**

```
<button onclick="setTimeout(myFunction, 3000)">Try it</button>

<script>
function myFunction() {
  alert('Hello');
}
</script>
```

How to stop the execution?

- To stop the function execution, use the **clearTimeout().**

```
window.clearTimeout(timeoutVariable)
```

- **The window** prefix can be **omitted.**
- The **variable returned from setTimeout()** method is used in the **clearTimeout()** method.

```
myVar = setTimeout(function, milliseconds);
clearTimeout(myVar);
```

setInterval() Method

- The **function** is executed **repeatedly** after a given **time interval**.

Syntax

```
window.setInterval(function, milliseconds);
```

- The **window** prefix can be **omitted.**
- The **first parameter** has the **function** to be executed**.**
- The **second parameter** has the **time interval** between each execution**.**

```
<button onclick="setInterval(myFunction, 1000);">Try it</button>

<script>
function myFunction() {
  alert('Hello');
}
</script>
```

How to stop the execution?

- To stop the function execution, use the **clearInterval().**

```
window.clearInterval(timerVariable)
```

- **The window** prefix can be **omitted.**
- The **variable returned from setInterval()** method is used in the **clearInterval()** method.

```
let myVar = setInterval(function, milliseconds);
clearInterval(myVar);
```

Function Sequence

- The functions in JavaScript are **executed in the sequence** they are **called**.

```
function myFirst() {
  myDisplayer("Hello");
}

function mySecond() {
  myDisplayer("Goodbye");
}

mySecond();
myFirst();
```

- It is better to have **control** over the **function execution**.
- To **control the sequence of function execution**, we go for JavaScript **callbacks**.

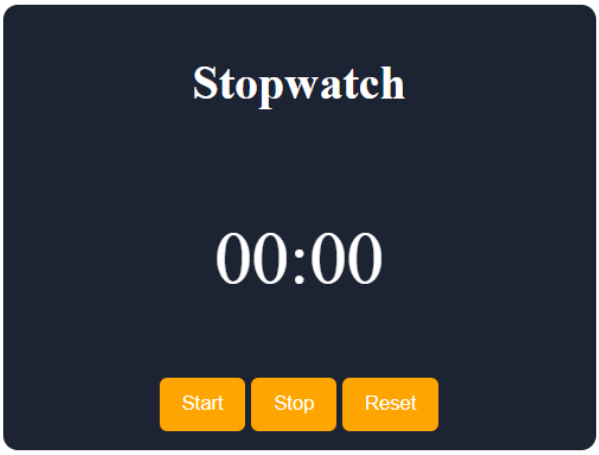
JavaScript Callbacks

- When a **function** is passed as an **argument** to **another function**, it is called a **callback**.
- **Callback functions** are used in the case of **asynchronous functions**, where one function **waits** for another function.

```
function myDisplayer(some) {
  document.getElementById("demo").innerHTML = some;
}

function myCalculator(num1, num2, myCallback) {
  let sum = num1 + num2;
  myCallback(sum);
}

myCalculator(5, 5, myDisplayer);
```

Stopwatch

Problem statement:
With your knowledge gained on JS Timing Events and callback design a Minute stopwatch with the time display that displays minute and seconds and having a button to start, stop and reset the time.



Hint

- The stopwatch display should be in the minutes and seconds format having start, stop and reset button.
- When the start button is clicked, the timer will start counting the time.
- When the stop button is clicked, the timer will be stopped at that moment.
- When the reset button is clicked, the timer will be reset to 0.

HTML

- Create a division to add description.
- Create a division to display the stopwatch where it should be accessed and displayed from the JavaScript.
- The Display division should have 3 buttons to perform start, stop and reset function.

JavaScript

- Create required variables.
- Invoke the function when the window loads with the stopwatch showing the initialized values i.e 0.
- Create a separate function to start the timer.
- Create a function to stop and reset the timer.
- Invoke the functions start, stop and reset on the button click.

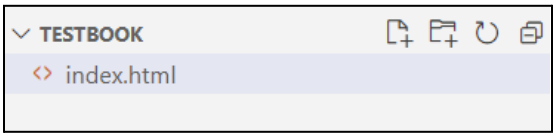
CSS

- Add your own styles to make the webpage attractive.

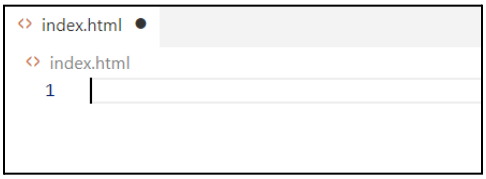
Steps:

- Open VS code.
- Select New File in the opened folder…

- Save the file in the appropriate folder/ create the folder and save the file with extension.



- The file in the VS Code appears with file extension after saving



Sample code:

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <div class="wrapper">
      <div class="card">
        <h1>Stopwatch</h1>
        <p class="timer"><span id="seconds">00</span>:<span
id="milliseconds">00</span></p>
        <div class="btnWrapper" >
          <button id="button-start">Start</button>
          <button id="button-stop">Stop</button>
          <button id="button-reset">Reset</button>
        </div>
      </div>
```

```
      </div>
      <script src="script.js"></script>
    </body>
  </html>
```

script.js

```
window.onload = function () {
    var seconds = 00;
    var milliseconds = 00;
    var appendmilliseconds = document.getElementById("milliseconds");
    var appendSeconds = document.getElementById("seconds");
    var buttonStart = document.getElementById('button-start');
    var buttonStop = document.getElementById('button-stop');
    var buttonReset = document.getElementById('button-reset');
    var Interval ;

    buttonStart.onclick = function(){

        clearInterval(Interval);
        Interval = setInterval(startTimer, 10);
    }

    buttonStop.onclick = function(){
        clearInterval(Interval);
    }

    buttonReset.onclick = function(){
        clearInterval(Interval);
        milliseconds = "00";
        seconds = "00";
        appendmilliseconds.innerHTML = milliseconds;
        appendSeconds.innerHTML = seconds;
    }

    function startTimer(){
        milliseconds++;

        if(milliseconds <= 9){
            appendmilliseconds.innerHTML = "0" + milliseconds;
        }
```

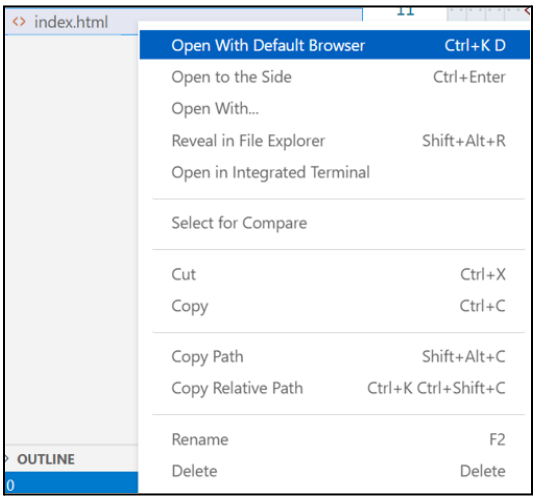
```
        if (milliseconds > 9){
            appendmilliseconds.innerHTML = milliseconds;
        }
        if (milliseconds > 99) {
            seconds++;
            appendSeconds.innerHTML = "0" + seconds;
            milliseconds = 0;
            appendmilliseconds.innerHTML = "0" + 0;
        }
        if (seconds > 9){
            appendSeconds.innerHTML = seconds;
        }
    }
}
```

style.css

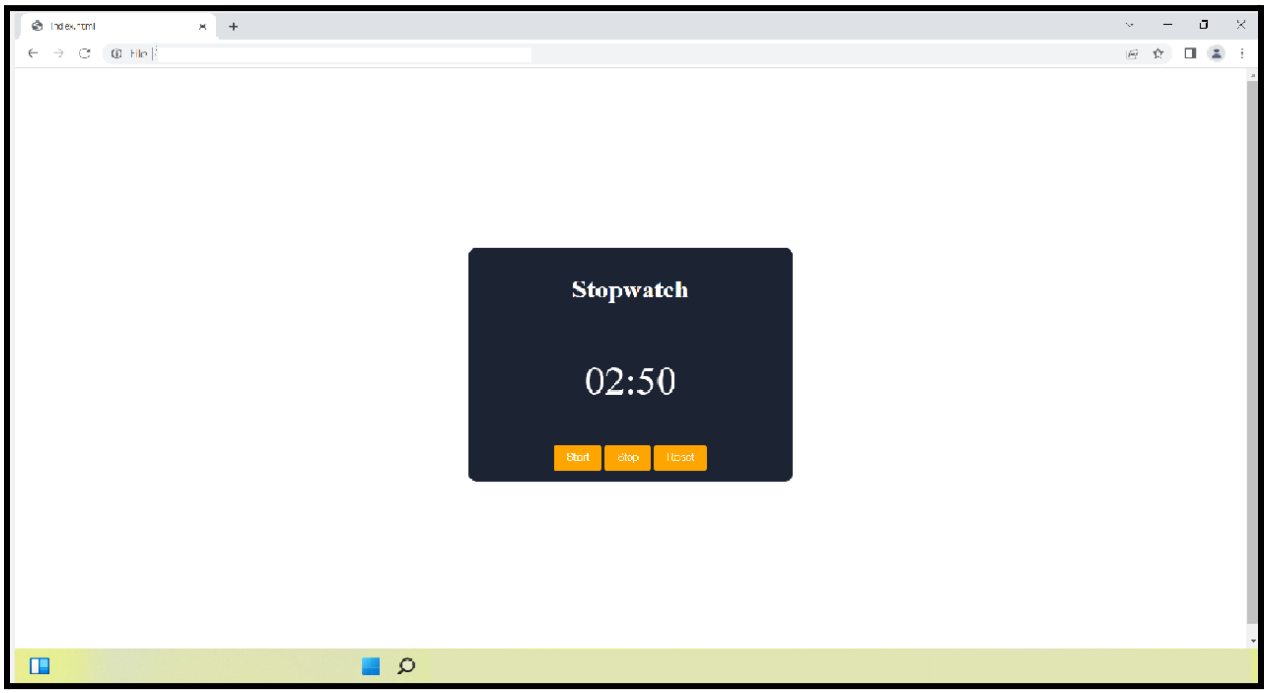
```
.wrapper, .card {
    height: 100vh;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
  }
  .card {
    width: 400px;
    height: 300px;
    background-color: #1c2333;
      color: #ffffff;
      border-radius: 10px;
  }
  .timer {
    font-size: 50px
  }
  button{
    background-color: orange;
    color: #fff;
    border: none;
    border-radius: 5px;
    padding: 10px 15px;
```

```
    cursor: pointer;
}
```

- View the file in the browser, by right click on the file name in the left pane.



Sample Output:



AJAX

Topics Covered:

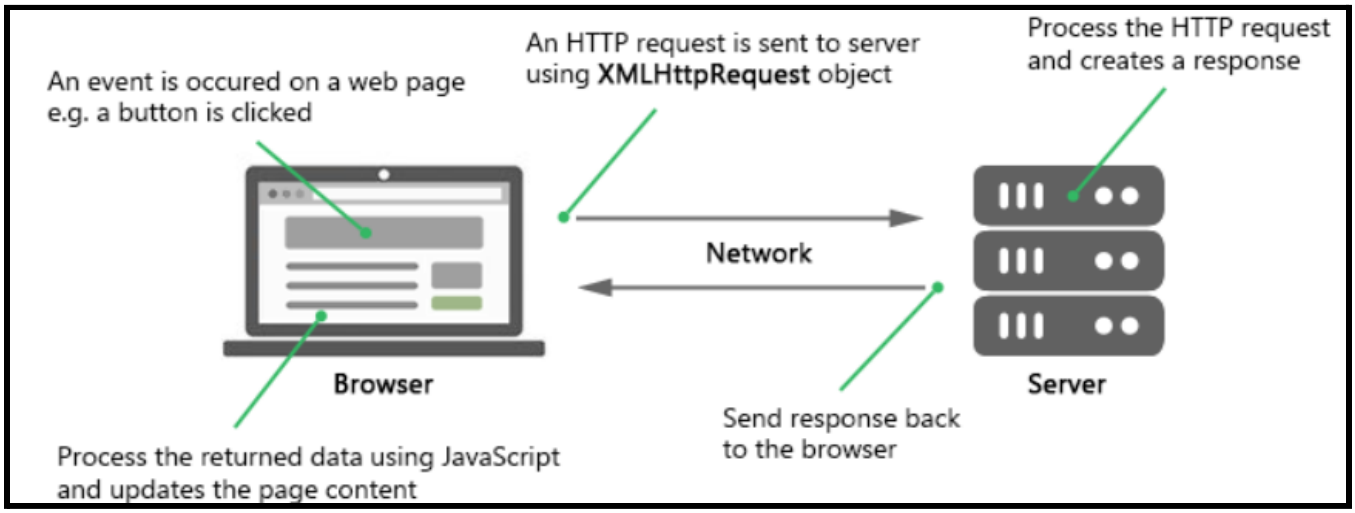
- AJAX Introduction
- XMLHttpRequest
- XMLHttp Response
- AJAX Application
- fetch()

Topics in Detail:

AJAX Introduction

- AJAX stands for **A**synchronous JavaScript **A**nd **X**ML.
- It is a combination of **XMLHttpRequest**, **JavaScript** and **HTML DOM**.
- The web pages are updated **asynchronously** by **exchanging data** with the **web server**.
- Instead of **reloading** the whole web page, **AJAX** helps in updating **parts of a web page**.

How AJAX works



XMLHttpRequest

- To request data from a server, **XMLHttpRequest object** is used.
- **Open()** and **send()** methods of XMLHttpRequest object are used to **send a request** to the server.

```
xhttp.open("GET", "ajax_info.txt", true);
xhttp.send();
```

Method	Description
open (method, url, async)	Specifies the type of request method - GET or POST url - file location async - true (asynchronous) or false (synchronous)
send()	Sends request to the server(GET)
send (string)	Sends request to the server (POST)

- **url - A file on a server**

```
xhttp.open("GET", "ajax_test.asp", true);
```

- The file can be of any kind like **.txt, .xml** or **server scripting files like .asp and .php**.

Asynchronous - True or False?

- **Asynchronous** parameter of open is set to **true**.
- JavaScript **does not wait** for server **response**, instead
    - Start **executing other scripts** while waiting.
    - After the **response is ready,** deal with the responses.
- The **default** value of async parameter is **true**.
- **async = false** is **not recommended**.

GET or POST

- **GET** is used in **most cases, and** it is **simpler** and **faster** than POST.
- **POST** is **more secure** and **robust** than **GET**.
- **POST** is used in the following situations
    - When **sending large amount of data** to the server.
    - When a **cached file** is **not an option**.
    - When sending **user input**.

GET Requests

- A simple GET request.

```
xhttp.open("GET", "demo_get.asp");
xhttp.send();
```

- To **avoid** getting **cached result**, add **a unique ID** to the URL.

```
xhttp.open("GET", "demo_get.asp?t=" + Math.random());
xhttp.send();
```

- To **send information** with the GET method, **add the information to the URL**.

```
xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford");
xhttp.send();
```

POST Requests

- A simple POST request.

```
xhttp.open("POST", "demo_post.asp");
xhttp.send();
```

- The Data-like HTML form are requested by adding an HTTP header with **setRequestHeader().**
- The data want to be sent is specified in the send() method.

```
xhttp.open("POST", "ajax_test.asp");
xhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
xhttp.send("fname=Henry&lname=Ford");
```

Method	Description
setRequestHeader(header, value)	Adds HTTP headers to the request header: specifies the header name value: specifies the header value

Synchronous Request

- **Synchronous request** is achieved by setting the **async** parameter to **false**.
- For quick testing, async = false is set.
- The onreadystatechange function is not needed as the server waits for server completion.

```
xhttp.open("GET", "ajax_info.txt", false);
xhttp.send();
document.getElementById("demo").innerHTML = xhttp.responseText;
```

XMLHttp Response

Server Response Properties

Property	Description
responseText	Get the response data as a string
responseXML	Get the response data as an XML data

The responseText Property

- The **responseText** property returns the **server response** as a **JavaScript string.**

```
document.getElementById("demo").innerHTML = xhttp.responseText;
```

The responseXML Property

- An in-built **XML parser** is there in the XMLHttpRequest **object**.
- The **responseXML** property returns the **server response** as an **XML DOM object**.

```
const xmlDoc = xhttp.responseXML;
const x = xmlDoc.getElementsByTagName("ARTIST");

let txt = "";
for (let i = 0; i < x.length; i++) {
  txt += x[i].childNodes[0].nodeValue + "<br>";
}
document.getElementById("demo").innerHTML = txt;

xhttp.open("GET", "cd_catalog.xml");
xhttp.send();
```

Server Response Methods

Method	Description
getResponseHeader()	Returns specific header information from the server resource
getAllResponseHeaders()	Returns all the header information from the server resource

The getAllResponseHeaders() Method

All the **header information** from the server response is returned by the **getAllResponseHeaders().**

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
    document.getElementById("demo").innerHTML =
    this.getAllResponseHeaders();
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

The getResponseHeader() Method

The **specific header information** from the server response is returned by **getResponseHeader() method**.

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
    document.getElementById("demo").innerHTML =
    this.getResponseHeader("Last-Modified");
}
xhttp.open("GET", "ajax_info.txt");
xhttp.send();
```

XML Applications

Display XML data in an HTML Table

- The values of <ARTIST> and <TITLE> elements are displayed in an HTML table.

```
<table id="demo"></table>

<script>
function loadXMLDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    const xmlDoc = xhttp.responseXML;
    const cd = xmlDoc.getElementsByTagName("CD");
    myFunction(cd);
  }
  xhttp.open("GET", "cd_catalog.xml");
  xhttp.send();
}

function myFunction(cd) {
  let table="<tr><th>Artist</th><th>Title</th></tr>";
  for (let i = 0; i < cd.length; i++) {
    table += "<tr><td>" +
    cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
    "</td><td>" +
    cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
    "</td></tr>";
  }
  document.getElementById("demo").innerHTML = table;
}
</script>

</body>
</html>
```

Display the first CD element in an HTML div

```
const xhttp = new XMLHttpRequest();
xhttp.onload = function() {
  const xmlDoc = xhttp.responseXML;
  const cd = xmlDoc.getElementsByTagName("CD");
  myFunction(cd, 0);
}
xhttp.open("GET", "cd_catalog.xml");
xhttp.send();

function myFunction(cd, i) {
  document.getElementById("showCD").innerHTML =
  "Artist: " +
  cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "<br>Title: " +
  cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "<br>Year: " +
  cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```

Navigate between CDs

To navigate between CDs, create next() and previous() functions.

```
function next() {
  // display the next CD, unless you are on the last CD
  if (i < len-1) {
    i++;
    displayCD(i);
  }
}

function previous() {
  // display the previous CD, unless you are on the first CD
  if (i > 0) {
    i--;
    displayCD(i);
  }
}
```

Show Album information when clicking on a CD

To show album information when clicking on a CD, call displayCD() function on the onclick event.

```
function displayCD(i) {
  document.getElementById("showCD").innerHTML =
  "Artist: " +
  cd[i].getElementsByTagName("ARTIST")[0].childNodes[0].nodeValue +
  "<br>Title: " +
  cd[i].getElementsByTagName("TITLE")[0].childNodes[0].nodeValue +
  "<br>Year: " +
  cd[i].getElementsByTagName("YEAR")[0].childNodes[0].nodeValue;
}
```

fetch() Method

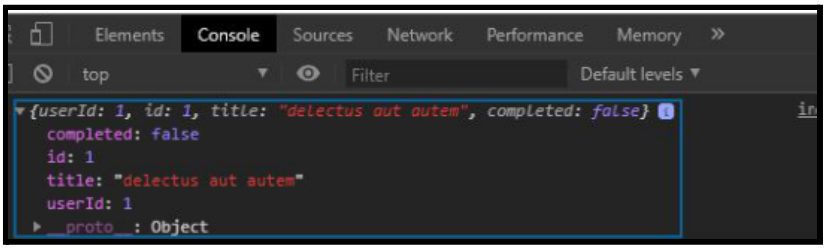
- The **fetch()** method in JavaScript **sends request to the server** and the fetched **information is being loaded** in the webpages.
- **Syntax**

```
fetch( url, options )
```

- **url -** url to which the request is to be made.
- **options -** It is an **optional parameter**. It is an array of properties.
- **Returns value** - The return data is of **JSON** or **XML format**. It can be a single object or array of objects.
- fetch() method without options will act as a get request.

```
    <script>
        // API for get requests
        let fetchRes = fetch(
"https://jsonplaceholder.typicode.com/todos/1");
        // fetchRes is the promise to resolve
        // it by using.then() method
        fetchRes.then(res =>
            res.json()).then(d => {
                console.log(d)
            })
    </script>
```

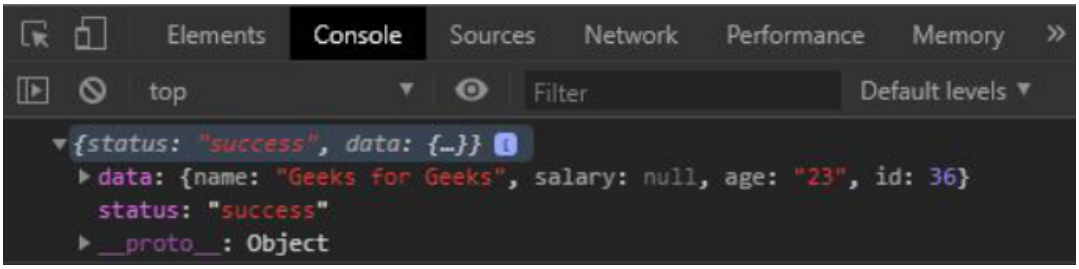
Output



- fetch() method with options given below will act as a post request.

```
    <script>
        user = {"name": "Geeks for Geeks",
            "age": "23"}
// Options to be given as parameter in fetch for making requests
other then GET
        let options = {
            method: 'POST',
            headers: {
                'Content-Type':'application/json;charset=utf-8'},
            body: JSON.stringify(user)}
        // Fake api for making post requests
        let fetchRes = fetch(
"http://dummy.restapiexample.com/api/v1/create",options);
        fetchRes.then(res =>
            res.json()).then(d => {console.log(d)})
    </script>
```

Output



AJAX introduction and Fetch Method - Practice Code

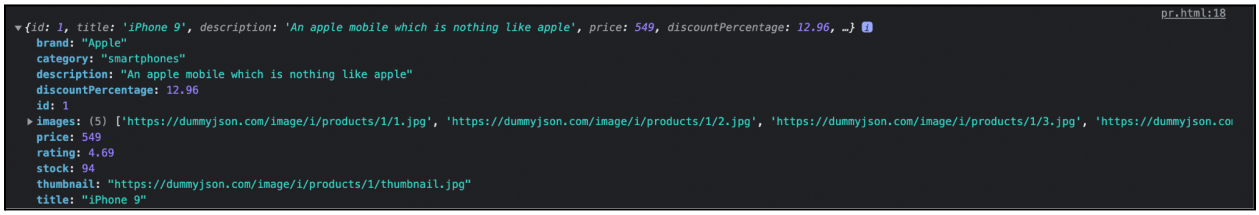
Steps to see the output:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.
- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.

Problem Statement 1: SmartPhone API

Write a javascript file to print the output of the data by fetching it from URL: 'https://dummyjson.com/products/1' ?

Output:-



Problem Statement 2: US Universities

Write a javascript file to make a table of US universities by making a call on below API URL:- http://universities.hipolabs.com/search?country=United+States

Output:-

Solutions

Problem Statement 1:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>AJAX introduction and Fetch Method Practice Code</title>
   <style>
    body{
        height:100%;
            background-image: linear-gradient(to right, rgb(77, 168, 183),
rgb(131, 120, 202));
        font-family: "Montserrat","Helvetica Neue","sans-serif";
    }
    </style>
</head>
<body>
    <h1>Fetching Data...</h1>
    <script>
        fetch('https://dummyjson.com/products/1')
        .then(res => res.json())
        .then(json => console.log(json))
    </script>
    </body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>AJAX introduction and Fetch Method Practice Code</title>
                                                                    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.cs
s"                                                          rel="stylesheet"
integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9B
v1WTRi" crossorigin="anonymous">
   <style>
    body{
        height:100%;
           background-image: linear-gradient(to right, rgb(211, 235, 238),
rgb(37, 169, 173));
        font-family: "Montserrat","Helvetica Neue","sans-serif";
    }
    </style>
</head>
<body>
    <div class="container">
        <h1>Fetching and Displaying Data in Table...</h1>
```

```
        <table id="table" class="table"></table>
    </div>
    <script>

fetch('http://universities.hipolabs.com/search?country=United+States')
        .then(res => res.json())
        .then(json => show(json))

        function show(universities) {
            let table = document.getElementById('table');
            for (let i = 0; i < universities.length; i++){
                let obj = universities[i];
                console.log(obj);
                let row = document.createElement('tr');
                let name = document.createElement ('td');
                let domains = document.createElement('td');
                let country = document.createElement('td');
                name.innerHTML=obj.name;
                domains.innerHTML=obj.domains;
                country.innerHTML=obj.country;
                row.appendChild(name)
                row.appendChild(domains)
                row.appendChild(country)
                table.appendChild(row)
            }
        }
    </script>
    </body>
</html>
```

Assignment - ES6

(This is a minor assignment and submission is not required. Will release solution next week so that you can self- evaluate)

Challenge 1:Block and Function Scope
Steps:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given HTML code in the file.

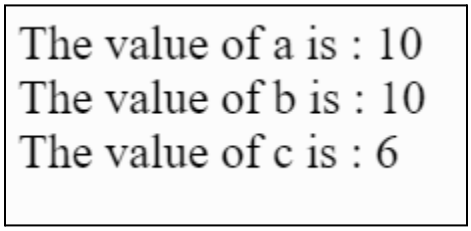
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>repl.it</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <script src="script.js"></script>
  </body>
</html>
```

- Copy and Add the below given JS code in the file.

```
var a=10; // Here value of a is 10
var b=8; // Here value of b is 8
var c=6; // Here value of c is 6
{
  let a = 2;
  // Here value of a is 2 but only within the block
  var b = 10;
  // Here value of b is 10 inside and outside the block it will remain 10
  const c = 8;
  // Here value of c is 8 but only within the block
  // const cannot be reassigned
  // c=5;
}
document.write("The value of a is : " +a);
// let has block scope
document.write("<br>");
```

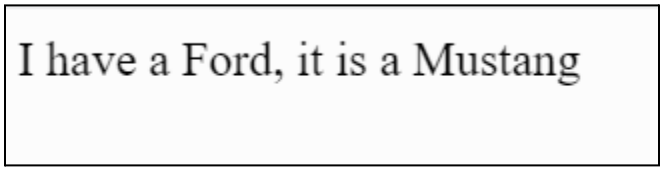
```
document.write("The value of b is : " +b);
// var does not have block scope
document.write("<br>");
document.write("The value of c is  : " +c);
// const has block scope
```

- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.



Challenge 2: ES6 Class

With your new gained knowledge in ES6 class, create a class with constructor, child class, method, object. Using the object access the method and class.

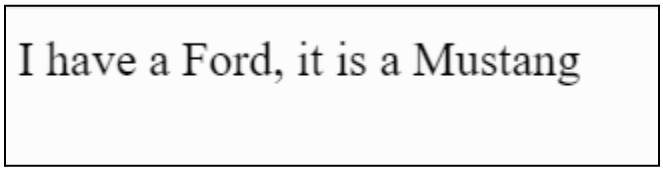


This is the sample output image, the text given is just for sample

ES6 Assignment - Solution

Problem statement:

With your new gained knowledge in ES6 class, create a class with constructor, child class, method, object. Using the object access the method and class.



This is the sample output image, the text given is just for sample

Steps:

- Open VS code.
- Select New File in the opened folder…

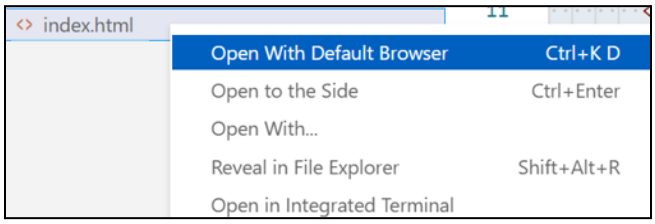
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Add the following code in the index.html file.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>repl.it</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <script src="script.js"></script>
  </body>
</html>
```

- Add the following code in the scripts.js file.

```
class Car {
  constructor(name) {
    this.brand = name;
  }
  present() {
    return 'I have a ' + this.brand;
  }
}

class Model extends Car {
  constructor(name, mod) {
    super(name);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model
  }
}

mycar = new Model("Ford", "Mustang");

document.write(mycar.show());
```

- View the file in the browser, by right click on the file name in the left pane.



Sample Output:

Async and Sync

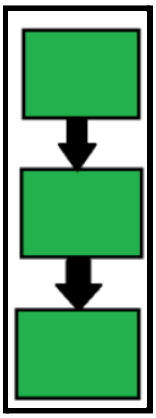
Topics Covered:

- Synchronous JavaScript
- Asynchronous JavaScript
- Memory Allocation
- Function call stack
- Event Loop
- Callback Hell

Topics in Detail:

Synchronous JavaScript

- Every statement in a **code** is **executed** in a **sequence,** one after the other.
- Every statement will **wait** for one statement to **complete execution**.
- JavaScript is a **single-threaded synchronous programming language**.
- The JavaScript code **does not** run in **parallel, but** it can only **run one at a time**.



```
console.log("Before delay");

function delayBySeconds(sec) {
   let start = now = Date.now()
   while(now-start < (sec*1000)) {
     now = Date.now();
   }
}

delayBySeconds(5);

// Executes after delay of 5 seconds
console.log("After delay");
```

Output

```
Before delay
(... waits for 5 seconds)
After delay
```

Asynchronous JavaScript

- The program will be executed **immediately** in **asynchronous code**.
- Many operations can be performed simultaneously in **AJAX**.

```
<script>
    document.write("Hi");
    document.write("<br>");

    setTimeout(() => {
        document.write("Let us see what happens");
    }, 2000);

    document.write("<br>");
    document.write("End");
    document.write("<br>");
</script>
```

Output

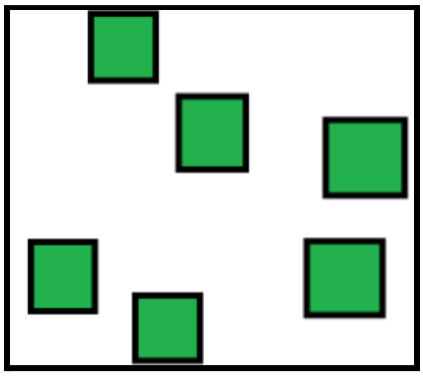
```
Hi
End
Let us see what happens
```

- At first, **Hi statement** will get logged.
- Then, JavaScript passes **setTimeout function to web API** and **rest of the code** will be **executed**.
- **After executing** all the code, the **setTimeout function** is pushed to the **call stack** and finally gets **executed**.

Memory Allocation

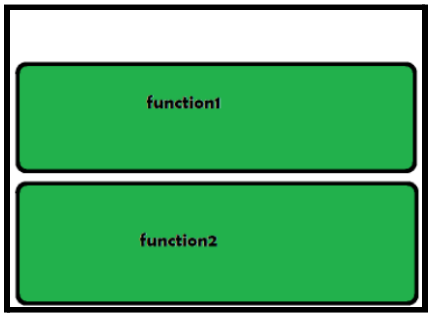
Heap Memory

- The **data** will be **stored randomly** and **memory** is also allocated in the same manner.



Stack Memory

- The **memory** will be allocated in the **form of a stack**. In case of **functions, stack memory** is used.

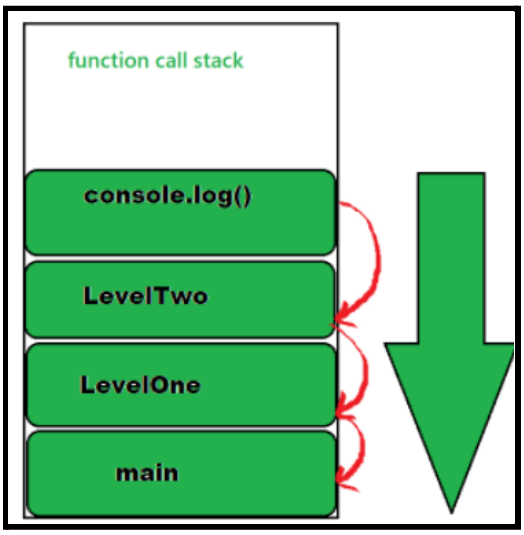


Function call stack

- The **function stack** is a **function** which **keeps track of all the functions** that are executed during the **run time**.
- When an **error** is occurred, we can see a **function stack** being printed at that time.

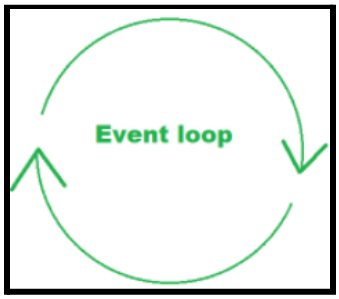
```
function LevelTwo() {
   console.log("Inside Level Two!")
}

function LevelOne() {
   LevelTwo()
}

function main() {
   LevelOne()
}

main()
```

- The function gets **popped out of stack** after the function's purpose gets over.

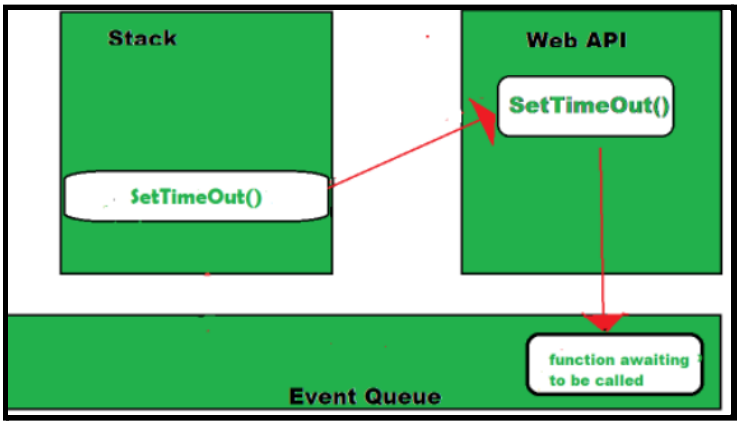


Event Loop

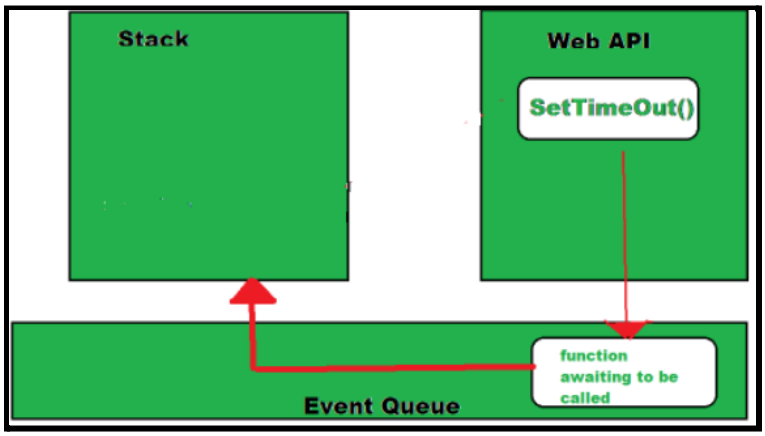
- Whenever a **function stack** is **empty**, the **event loop** pulls the stuff **out of queue** and places it over the **function stack**.



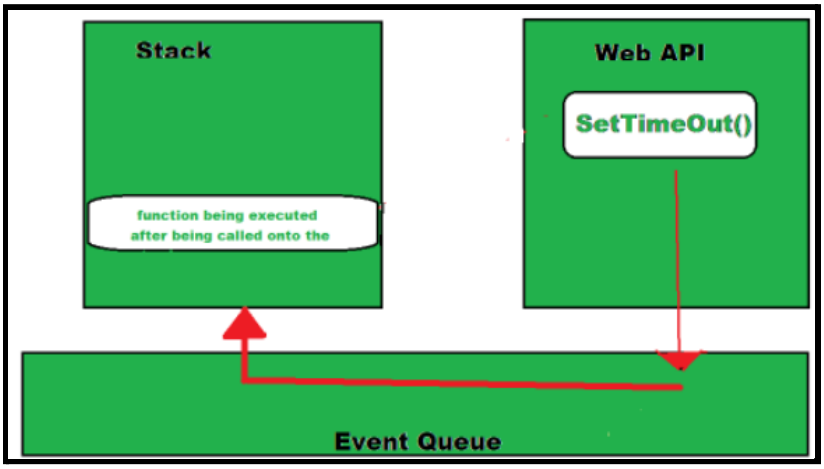
- The event loop gives the illusion of **multithreaded**.



- The **callback** is in the **event queue** is **waiting** for its turn in the stack to run when **setTimeout()** is being **executed**. When the **function stack** becomes **empty**, it is **loaded** to the **stack**.



- The **first event** from the event queue is now being placed on the **stack**. This cycle is called **event loop** and this is how JavaScript handles **events**.



Callback Hell

- The code with **complex nested callbacks** will cause a **big issue** called **Callback Hell**.
- The **result** of the **previous callback** is taken up by the **upcoming callbacks**.
- The **code structure** will look like a **pyramid**.
- It is **difficult** to **read** and **maintain**.
- If anyone **function** has an **error**, it will **affect** all the **other function**.

How to avoid callback hell?

- In JavaScript, **event queue** and **promises** help to **escape** from a **callback hell**.
- Any **asynchronous function** will **return an object** called **promise**. A **callback method** can be added to a **promise**.

- **.then()** method is used by **promises** to **call async callbacks**. As many callbacks can be chained together. The **order** of the callbacks is also **strictly maintained**.
- Promise uses
    - **.fetch()** method to **fetch an object** from the network.
    - **.catch()** method to **catch any exception** when any block fails.
- The subsequent JS code **doesn't block** if these **promises** are put in **event queue**. The event queue finishes its operations once the results are returned.
- The keywords and methods like **async, wait, settimeout()** are used to **simplify** and make **callbacks used better**.

JavaScript Async - Practice Code

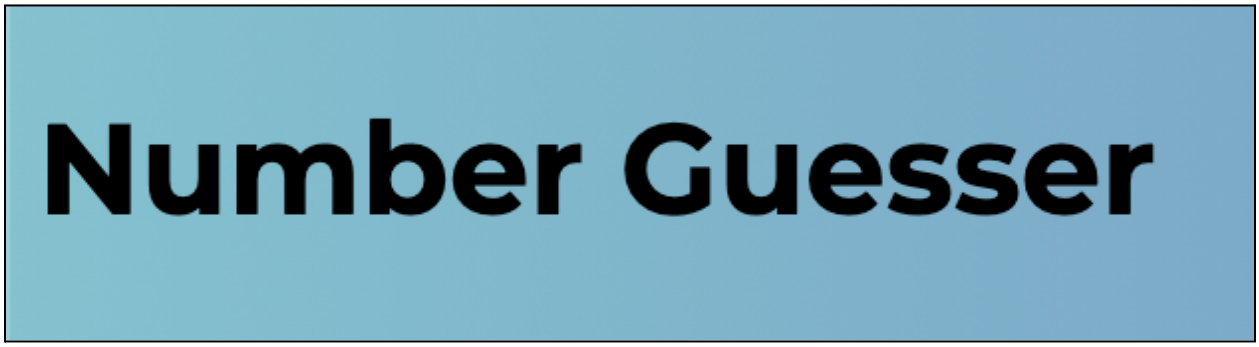
Steps to see the output:

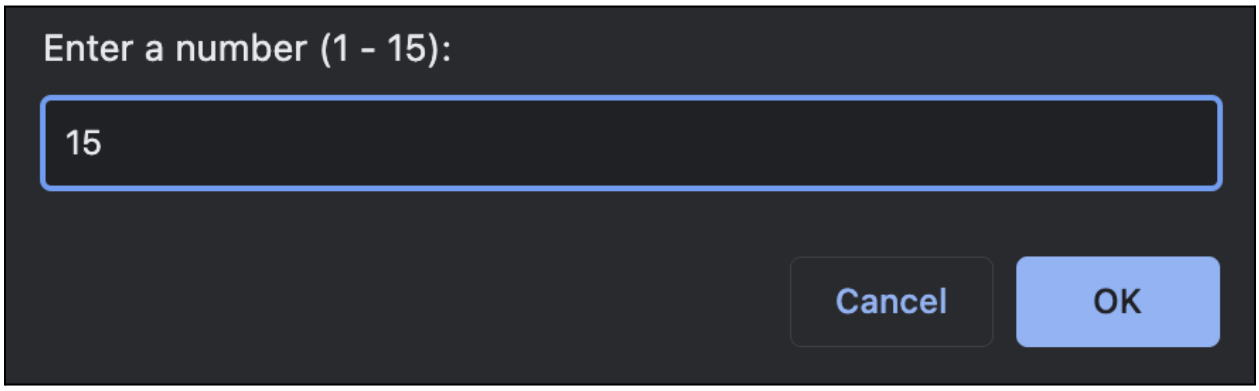
- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.
- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.

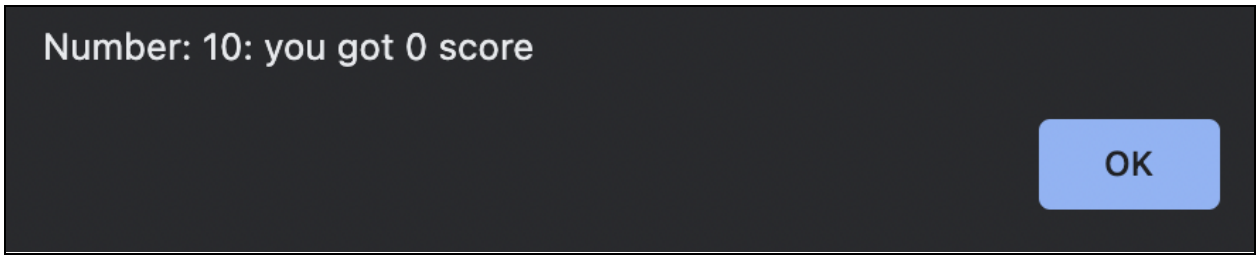
Problem Statement 1: Number Guesser

Write a javascript file to make a number guesser game where you ask the user for the input number from 1 to 15 and check whether the number guessed is right or wrong?

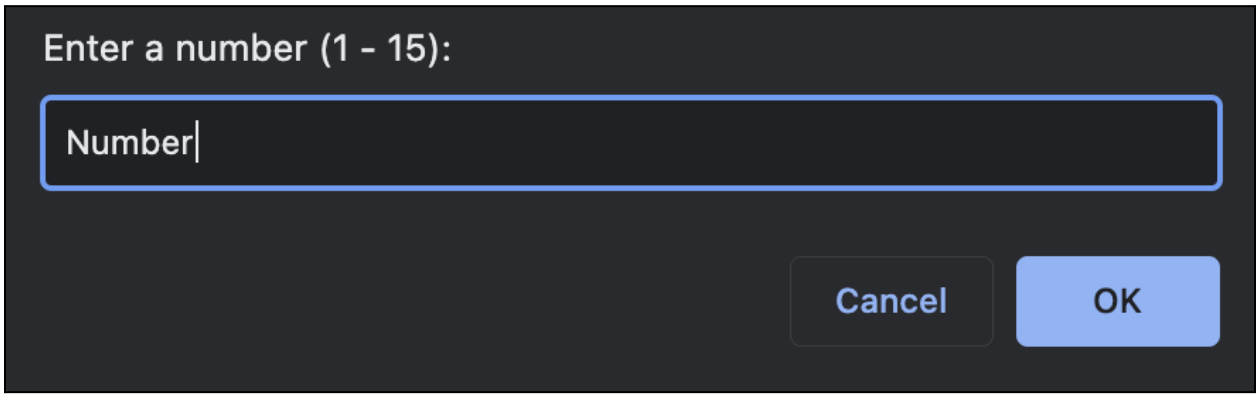
Output:-






Number: 10: you got 0 score
OK


Do you want to continue?
Cancel
OK


Enter a number (1 - 15):
Number
Cancel
OK


Error: Wrong Input Type
OK


Game ends
OK

Problem Statement 2: Random Image Generator

Write a javascript file to make a random image generator using an API call?

Output:-

Solutions

Problem Statement 1:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Number Guesser</title>
   <style>
    body{
        height:100%;
        background-image: linear-gradient(to right, rgb(113, 196, 209),
rgb(102, 89, 187));
        font-family: "Montserrat","Helvetica Neue","sans-serif";
    }
    </style>
</head>
<body>
    <h1>Number Guesser</h1>
    <script>
        const enterNumber = () => {
            return new Promise((resolve, reject) => {
              const userNumber = Number(window.prompt("Enter a number (1 -
15):")); // Ask the user to enter a number
              const randomNum = Math.floor(Math.random() * 15 + 1); // Pick
a random number between 1 and 6

              if (isNaN(userNumber)) {
                reject(new Error("Wrong Input Type")); // If the user enters
a value that is not a number, run reject with an error
              }

              if (userNumber === randomNum) { // If the user's number
matches the random number, return 2 score
                resolve({
                  score: 2,
                  randomNum,
                });
              } else if (
                userNumber === randomNum - 1 ||
                userNumber === randomNum + 1
              ) { // If the user's number is different than the random
number by 1, return 1 point
                resolve({
                  score: 1,
                  randomNum,
                });
              } else { // Else return 0 score
                resolve({
                  score: 0,
                  randomNum,
                });
              }
```

```
            });
          };

          const continueGame = () => {
            return new Promise((resolve) => {
              if (window.confirm("Do you want to continue?")) { // Ask if
the user want to continue the game with a confirm modal
                resolve(true);
              } else {
                resolve(false);
              }
            });
          };

          const handleGuess = async () => {
            try {
              const result = await enterNumber(); // Instead of the then
method, we can get the result directly by just putting await before the
promise

              alert(`Number: ${result.randomNum}: you got ${result.score}
score`);

              const isContinuing = await continueGame();

              if (isContinuing) {
                handleGuess();
              } else {
                alert("Game ends");
              }
            } catch (error) { // Instead of catch method, we can use the
try, catch syntax
              alert(error);
            }
          };
          handleGuess(); // Run handleGuess function
    </script>
    </body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Random Image Generator</title>
   <style>
    body{
        height:100%;
        background-image: linear-gradient(to right, rgb(161, 113, 209),
rgb(18, 8, 85));
        font-family: "Montserrat","Helvetica Neue","sans-serif";
```

```
    }
    .container{
        display:flex;
        flex-direction: column;
        align-items:center;
        justify-content: center;
    }
    #image{
        width:350px;
        height:350px;
        border: 5px solid #fff;
        box-sizing: border-box;
        overflow: hidden;
    }
    </style>
</head>
<body>
    <div class="container">
        <h1 style="text-align:center; color:azure;">Random Image
Generator</h1>
        <img id="image"/>
    </div>
    <script>
        const url = 'https://source.unsplash.com/random'
        const img = document.getElementById("image");
        const getImage = async (url) => {
        return await fetch(url).then(res => res.url)
        }
        const img_url=getImage;
        getImage(url).then(result => {
            img.setAttribute('src', result)
        });
    </script>
    </body>
</html>
```

Async/Await

Topics Covered:

- Async
- Await

Topics in Detail:

Async

- To make a function **return a promise,** add an **async** keyword before a function.

Syntax

```
async function myFunction() {
  return "Hello";
}
```

Same as

```
function myFunction() {
  return Promise.resolve("Hello");
}
```

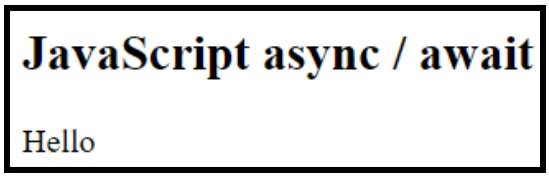
How to use a promise

```
myFunction().then(
  function(value) { /* code if successful */ },
  function(error) { /* code if some error */ }
);
```

Example

```
async function myFunction() {
  return "Hello";
}
myFunction().then(
  function(value) {myDisplayer(value);},
  function(error) {myDisplayer(error);}
);
```

Output



Await

- To make a function **wait for a promise,** add **await** keyword before a function.

Syntax

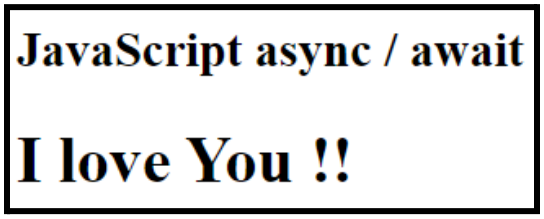
```
let value = await promise;
```

- **Await** keyword can be used **only within an async** function.

Example

```
async function myDisplay() {
  let myPromise = new Promise(function(resolve, reject) {
    resolve("I love You !!");
  });
  document.getElementById("demo").innerHTML = await myPromise;
}

myDisplay();
```

Output



- JavaScript has **two pre-defined arguments** (**resolve** and **reject**).
- When the **executor** function is **ready**, **call one** of them.
- The **reject** function is **not needed very often**.

Async and Await - Practice Code

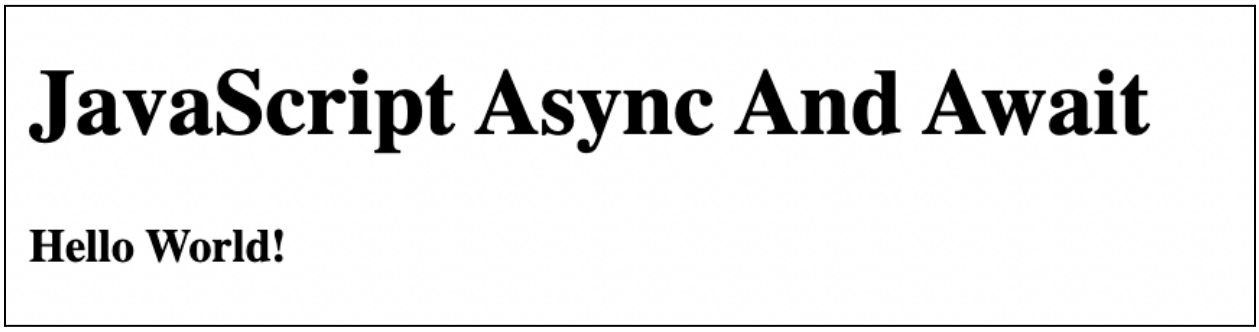
Steps to see the output:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.
- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.

Problem Statement 1: Timeout Function

Write a javascript file to have a timeout function with the help of async and await?

Output:-



Problem Statement 2: Anime/Movies Series Title

Write a javascript file to display movies/anime series title by storing it in an array or an object using async and await.

Output:-

Solutions

Problem Statement 1:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Async And Await Practice Code</title>
</head>
<body>
    <h1>JavaScript Async And Await</h1>
    <h4 id="text"></h4>
    <script>
        async function display() {
            let myPromise = new Promise(function(myResolve, myReject) {
              setTimeout(function() { myResolve("Hello World!"); }, 5000);
            });
            document.getElementById("text").innerHTML = await myPromise;
          }
          display();
    </script>
    </body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Async And Await Practice Code</title>
</head>
<body>
    <h1>JavaScript Async And Await</h1>
    <h4 id="text"></h4>
    <script>
        const anime = [
        { title: `Naruto`, body:`Naruto is a Japanese manga series written
and illustrated by Masashi Kishimoto. It tells the story of Naruto Uzumaki,
a young ninja who seeks recognition from his peers and dreams of becoming
the Hokage, the leader of his village.`},
        { title: `Doraemon`, body: `Nobita Nobi is a ten-year-old Japanese
school boy, who is kind-hearted and honest, but also lazy, unlucky, weak,
gets bad grades and is bad at sports. One day, a robot cat from the 22nd
century named Doraemon is sent back to the past by Nobita's descendants to
take care of Nobita so that his descendants can have a better life.`}
    ]

        function getanime(){
            setTimeout(() => {
                anime.forEach((anime_series, index) => {
```

```
                    console.log(anime_series.title)
                })
            }, 1000);
        }

        function createanime(anime_series){
            return new Promise((resolve, reject) => {
                setTimeout(() => {
                    anime.push(anime_series);
                    const error = false;
                    if(!error){
                        resolve();
                    }
                    else{
                        reject('Error: Something went wrong!')
                    }
                }, 2000);
            })
        }
        async function display(){
            await createanime({ title: `Dragon Ball`, body:`Earth, known as
the Dragon World and designated as "Planet 4032-877" by the celestial
hierarchy, is the main setting for the entire Dragon Ball series, as well as
related media such as Dr. Slump, Nekomajin, and Jaco the Galactic Patrolman.
It is mainly inhabited by Earthlings , a term used inclusively to refer to
all of the intelligent races native to the planet, including humans,
anthropomorphic beings, and monsters.`});
            getanime();
        }
        display();
    </script>
    </body>
</html>
```

Promises & Error Handling

Topics Covered:

- Promises
- Error Handling

Topics in Detail:

Promises

- **Asynchronous operations** in JavaScript are **handled** using **promises**.
- **Multiple asynchronous operations** can make the code unmanageable by creating **callback hell**. Promises can **easily manage** this situation.
- **Events** and **callback** functions are used to **handle asynchronous operations** before promises, but they had **limited functionalities** making the **code unmanageable**.
- **Callback hell** created by **multiple callback functions** make the **code unmanageable**.
- **Multiple callbacks** at the **same time** are **not easy** to **handle**.
- **Promises** can **handle multiple asynchronous operations** easily.
- They can handle multiple callbacks at the same time, **avoiding a callback hell** situation.
- Promises **improve** the code **readability** in the most effective and efficient manner.

Benefits of promises

- Improves **Code Readability**
- Better handling of **asynchronous operations**
- Better **flow of control** definition in **asynchronous logic**
- Better **Error Handling**

States of Promises

1. **fulfilled**: Promise is **succeeded**
2. **rejected**: Promise is **failed**
3. **pending**: Promise is **still pending,** i.e. not fulfilled or rejected yet
4. **settled**: Promise has **fulfilled** or **rejected**

Create a promise using the promise constructor

Syntax

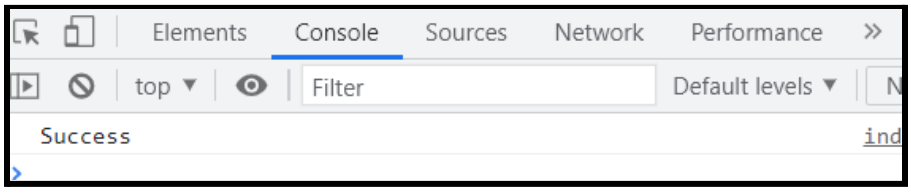
```
var promise = new Promise(function(resolve, reject){
     //do something
});
```

Parameters

- **The promise constructor** can have only **one argument,** which is a **callback function**.
- The callback function can take **two arguments**
    - resolve
    - reject
- If the operations inside a **callback function performed** well, then call **resolve**.
- If **not performed** well then call **reject**.

```
<!DOCTYPE html>
<html>
<body>
<script>
var promise = new Promise(function(resolve, reject) {
const x = "JS Promises";
const y = "JS Promises";
if(x === y) {
  resolve();
} else {
  reject();
}
});

promise.
  then(function () {
    console.log('Success');
  }).
  catch(function () {
    console.log('Some error has occurred');
  });
</script>
</body>
</html>
```

Output



Promise Consumers

- Promises can be consumed by using **.then** and **.catch** methods.
- **then()**
- When a promise is either **resolved or rejected,** then() is invoked. It acts as a career taking **data from the promise** and further **executes** it successfully.

Parameters

- **then()** has two functions as parameters
- If the promise is **resolved** and the **result is received**, then the **first function** will be executed.
- If the promise is **rejected** and an **error is received**, then the **second function** will be executed.
- **Syntax**

```
.then(function(result){
        //handle success
    }, function(error){
        //handle error
    })
```

- **catch():** When a promise is **either rejected** or if some **error** has occurred, **catch()** is invoked. If there is any chance of getting an error, it is used as an **Error handler**.

Parameters

- **then()** has one function as parameters
- If the promise is **rejected or the error** has occurred, then the function can handle it.

  Syntax

```
.catch(function(error){
        //handle error
    })
```

Applications

- To handle **asynchronous events**, promises are used.
- To handle **asynchronous http requests**, promises are used.

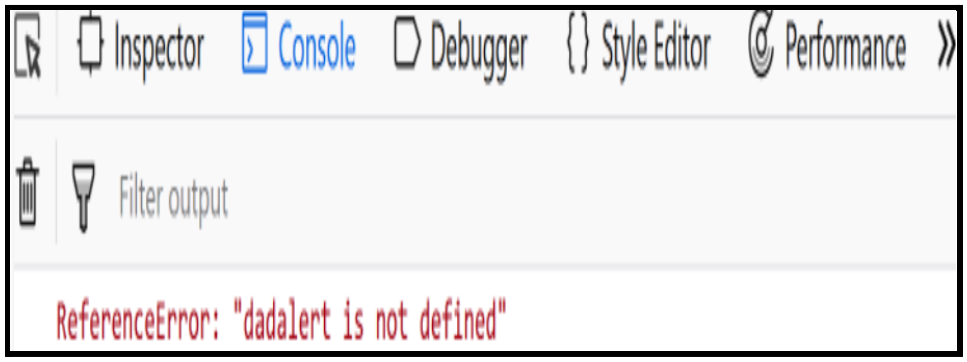
Error Handling

- **Errors** will definitely occur while **executing JavaScript code**.
- Error can occur in the following situations
  - When there is a **fault from the programmer side**
  - When the **input is wrong**
  - When there is a **problem with the logic** of the program
- Using the below five statements, we can solve the errors
  - **try** - **Check for errors** in a block of code.
  - **catch** - **Handles the error** if there are any
  - **throw** - lets you make your **own error**
  - **finally** - Execute the code **after try and catch**
- This block of code will run **regardless of the result** of the **try-catch block**.
- **Example**

```
try {
  dadalert("Welcome Fellow Geek!");
}
catch(err) {
  console.log(err);
}
```

- **Dadalert** is not a reserved keyword and neither it is defined, hence we get an error.

Output



Try and catch block

- The try statement will let you **check** whether there is an **error** in a specific block of code.
- The catch statement will **display the error** if there is any in the **try block**.
- **Syntax**

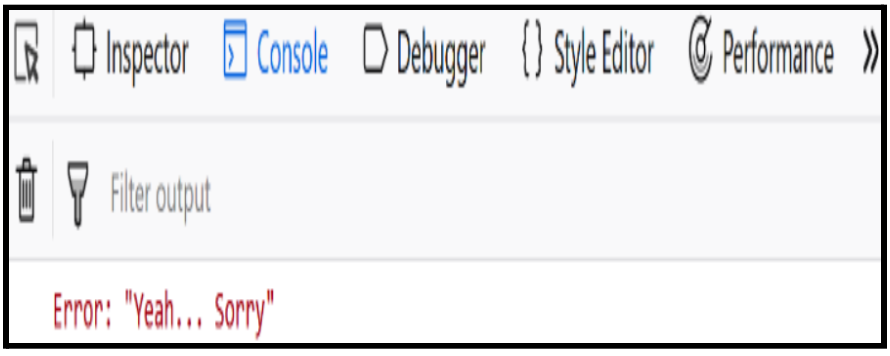
```
try {
 Try Block to check for errors.
}
catch(err) {
  Catch Block to display errors.
}
```

Throw

- JavaScript will stop and **generate an error message** when any **error** occurs. The **throw** statement will allow us to create any **custom-made errors**.

```
try {
    throw new Error('Yeah... Sorry');
}
catch(e) {
    console.log(e);
}
```

- **Output**



Finally Block

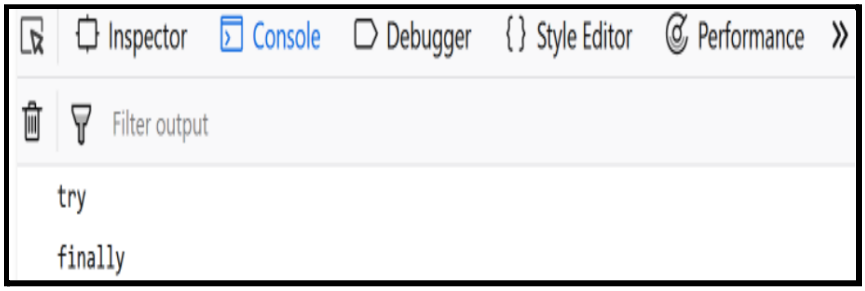
- After the execution of **try/catch block**, the **finally block** runs **unconditionally.**
- **Syntax**

```
 try {
 Try Block to check for errors.
}
catch(err) {
  Catch Block to display errors.
}
finally {
  Finally Block executes regardless of the try / catch result.
}
```

- **Example**

```
try {
  alert( 'try' );
} catch (e) {
  alert( 'catch' );
} finally {
  alert( 'finally' );
}
```

- **Output**



Promise and Error Handling - Practice Code

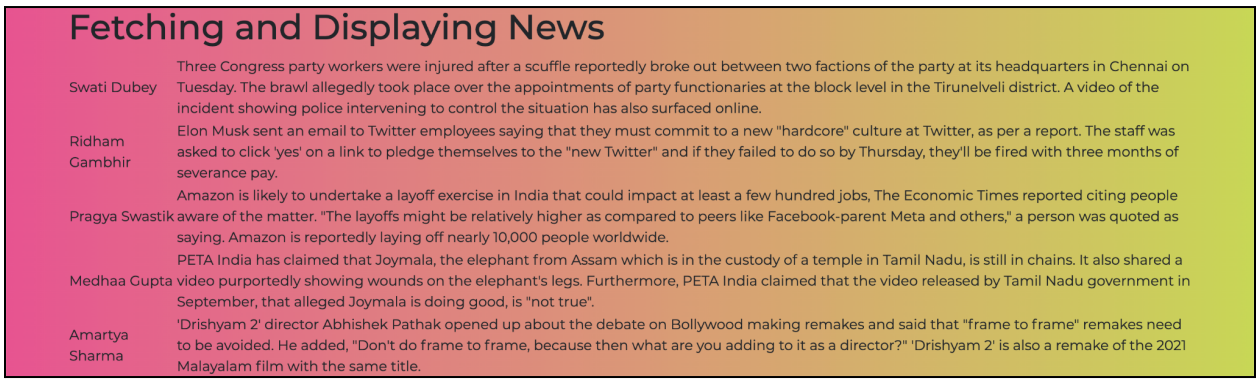
Steps to see the output:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.
- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.

Problem Statement 1: News App

Write a javascript file to implement a news app by using the following api = '**https://inshorts.deta.dev/news?category=all**'. Also add a catch statement if any error is encountered while fetching the api.

Output:-

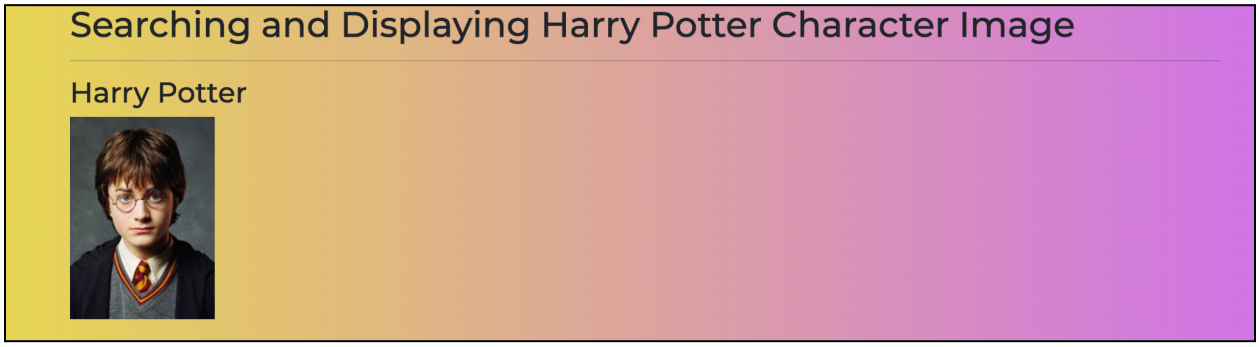


Problem Statement 2: Harry Potter Character Name and Image

Write a javascript file to display the harry potter character image and name after taking the character's name as input from the user using the following

api = '**https://hp-api.herokuapp.com/api/characters/**' and also display any error message if the character is not found!

Output:-



Solutions

Problem Statement 1:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Promise and Error Handling - Practice Code</title>
   <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.cs
s" rel="stylesheet"
integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9B
v1WTRi" crossorigin="anonymous">
   <style>
    body{
        height:100%;
        background-image: linear-gradient(to left, rgb(200, 215, 86),
rgb(231, 84, 145));
        font-family: "Montserrat","Helvetica Neue","sans-serif";
    }
    </style>
</head>
<body>
    <div class="container">
        <h1>Fetching and Displaying News</h1>
        <table id="table" class="table"></table>
    </div>
    <script>
        fetch('https://inshorts.deta.dev/news?category=all')
        .then(res => res.json())
        .then(json => {
            json=json.data;
            let table = document.getElementById('table');
            for (let i = 0; i < json.length; i++){
                let obj = json[i];
                console.log(obj);
                let row = document.createElement('tr');
                let author = document.createElement ('td');
                let content = document.createElement('td');
                author.innerHTML=obj.author;
                content.innerHTML=obj.content;
                row.appendChild(author)
                row.appendChild(content)
                table.appendChild(row)
            }
            console.log(json);
            }).catch(error => console.log(error))
    </script>
    </body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Promise and Error Handling - Practice Code</title>
   <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.2/dist/css/bootstrap.min.cs
s" rel="stylesheet"
integrity="sha384-Zenh87qX5JnK2Jl0vWa8Ck2rdkQ2Bzep5IDxbcnCeuOxjzrPF/et3URy9B
v1WTRi" crossorigin="anonymous">
   <style>
    body{
        height:100%;
        background-image: linear-gradient(to left, rgb(208, 113, 234),
rgb(231, 214, 84));
        font-family: "Montserrat","Helvetica Neue","sans-serif";
    }
    </style>
</head>
<body>
    <div class="container">
        <br>
        <h1>Searching and Displaying Harry Potter Character Image</h1>
        <hr>
        <h2 id="h2"></h2>
        <img id="image1"/>
        <table id="table" class="table"></table>
    </div>
    <script>
        let str=prompt('Input Your Harry Potter Character Name','Harry
Potter');
        let url='https://hp-api.herokuapp.com/api/characters/';
        fetch(url)
        .then(res => res.json())
        .then(json => {
            let val = json.find(o => o.name === str);
            console.log(val);
            let img = document.getElementById('image1');
            img.setAttribute('src',val.image);
            let h2 = document.getElementById('h2');
            h2.innerText = val.name;
            }).catch(error => console.log(error))
    </script>
    </body>
</html>
```

RegEx

Topics Covered:

- What is RegEx?
- Brackets in RegEx.
- Quantifiers in RegEx.
- Characters in RegEx.
    - Characters.
    - Literal characters.
    - Metacharacters.
- Modifiers in RegEx.
- RegEx properties.
- RegEx methods.

Topics in Detail:

What is RegEX?

- A regular expression is an object that describes a pattern of characters.
- The JavaScript RegExp class represents regular expressions and defines methods that use regular expressions to perform powerful pattern-matching and search-and-replace functions on text.
- A regular expression could be defined with the RegExp () constructor.
- Syntax:
  var pattern = new RegExp(pattern, attributes);
  or
  var pattern = /pattern/attributes;
- Where
    - pattern → A string that specifies the pattern of the regular expression or another regular expression.
    - attributes → An optional string containing any of the "g", "i", and "m" attributes that specify global, case-insensitive, and multiline matches, respectively.

Brackets:

Brackets ([]) have a special meaning when used in the context of regular expressions. They are used to find a range of characters.

Bracket	Description
[...]	Any one character between the bracket.
[^...]	Any one character not between the bracket.
[0-9]	It matches any decimal digit from 0 through 9.
[a-z]	It matches any character from lowercase a through lowercase z.
[A-Z]	It matches any character from uppercase A through uppercase Z.
[a-Z]	It matches any character from lowercase a through uppercase Z.

- The ranges shown above are general.
- We can use the range [0-3] to match any decimal digit ranging from 0 through 3.
- The range [b-v] to match any lowercase character ranging from b through v.

Quantifiers:

- The frequency of position of bracketed character sequences and single characters can be denoted by a special character.
- Each special character has a specific connotation. The +, *, ?, and $ flags all follow a character sequence.

Expression	Description
p+	It matches any string containing one or more p's.
p*	It matches any string containing zero or more p's.
p?	It matches any string containing at most one p.
p{N}	It matches any string containing a sequence of N p's
p{2,3}	It matches any string containing a sequence of two or three p's.
p{2,}	It matches any string containing a sequence of at least two p's.
p$	It matches any string with p at the end of it.
^p	It matches any string with p at the beginning of it.

Matching Characters:

Following examples explain more about matching characters

Expression	Description
[^a-zA-Z]	It matches any string not containing any of the characters ranging from a through z and A through Z.
p.p	It matches any string containing p, followed by any character, in turn followed by another p.
^.{2}$	It matches any string containing exactly two characters.
(.*)	It matches any string enclosed within and .
p(hp)*	It matches any string containing a p followed by zero or more instances of the sequence hp.

Literal characters:

Character	Description
Alphanumeric	Itself
\0	The NUL character (\u0000)
\t	Tab (\u0009
\n	Newline (\u000A)
\v	Vertical tab (\u000B)
\f	Form feed (\u000C)
\r	Carriage return (\u000D)
\xnn	The Latin character specified by the hexadecimal number nn; for example, \x0A is the same as \n
\uxxxx	The Unicode character specified by the hexadecimal number xxxx; for example, \u0009 is the same as \t
\cX	The control character ^X; for example, \cJ is equivalent to the newline character \n

Metacharacters:

- A metacharacter is simply an alphabetical character preceded by a backslash that acts to give the combination a special meaning.
- For instance, you can search for a large sum of money using the '\d' metacharacter: /([\d]+)000/, Here \d will search for any string of numeric characters.
- The following table lists a set of metacharacters which can be used in PERL Style Regular Expressions.

Character	Description
.	A single character
\s	a whitespace character (space, tab, newline)
\S	non-whitespace character
\d	a digit (0-9)
\D	a non-digit
\w	a word character (a-z, A-Z, 0-9, _)
\W	a non-word character
[\b]	a literal backspace (special case).
[aeiou]	matches a single character in the given set
[^aeiou]	matches a single character outside the given set
(foo|bar|baz)	matches any of the alternatives specified

Modifiers:

Several modifiers are available that can simplify the way you work with regexps, like case sensitivity, searching in multiple lines, etc.

Modifiers	Description
i	Perform case-insensitive matching.
m	Specifies that if the string has newline or carriage return characters, the ^ and $ operators will now match against a newline boundary, instead of a string boundary
g	Performs a global matchthat is, find all matches rather than stopping after the first match.

RegExp properties:

Properties	Description

constructor	Specifies the function that creates an object's prototype.
	`var re = new RegExp( "string" );`
	`document.write("re.constructor is:" + re.constructor);`

global	Specifies if the "g" modifier is set.
	`var re = new RegExp( "string" );`
	
	`if ( re.global ) {`
	`document.write("Test1 - Global property is set");`
	`} else {`
	`document.write("Test1 - Global property is not set");`
	`}`

ignoreCase	Specifies if the "i" modifier is set.
	`var re = new RegExp( "string" );`
	
	`if ( re.ignoreCase ) {`
	`document.write("Test1-ignoreCase property is set");`
	`} else {`
	`document.write("Test1-ignoreCase property is not set")`
	`}`

lastIndex	The index at which to start the next match.
	`var str = "Javascript is an interesting scripting language";`
	`var re = new RegExp( "script", "g" );`
	
	`re.test(str);`

multiline	Specifies if the "m" modifier is set.
	`var re = new RegExp( "string" );`
	
	`if ( re.multiline ) {`
	`document.write("Test1-multiline property is set");`
	`} else {`
	`document.write("Test1-multiline property is not set");`
	`}`

source	The text of the pattern.
	`var str = "Javascript is an interesting scripting language";`
	`var re = new RegExp( "script", "g" );`
	
	`re.test(str);`
	`document.write("The regular expression is : " + re.source);`

RegExp Methods:

Method	Description

exec()	Executes a search for a match in its string parameter.
	`var str = "Javascript is an interesting scripting language";`
	`var re = new RegExp( "script", "g" );`
	
	`var result = re.exec(str);`

test()	Tests for a match in its string parameter.
	`var str = "Javascript is an interesting scripting language";`
	`var re = new RegExp( "script", "g" );`
	
	`var result = re.test(str);`

toSource()	Returns an object literal representing the specified object; you can use this value to create a new object.
	
	`var str = "Javascript is an interesting scripting language";`
	`var re = new RegExp( "script", "g" );`
	
	`var result = re.toSource(str);`

toString()	Returns a string representing the specified object.
	`var str = "Javascript is an interesting scripting language";`
	`var re = new RegExp( "script", "g" );`
	
	`var result = re.toString(str);`

Regular Expressions - Practice Code

Steps to see the output:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given JS code in the file.
- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.

Problem Statement 1: Hex Color Checker

Write a javascript function to check whether a given value is a valid hex color value or not.

Example:- #333, #7777

Output:-

```
true
false
```

Problem Statement 2: Domain Checker

Write a javascript file to check whether a given value represents a domain or not.

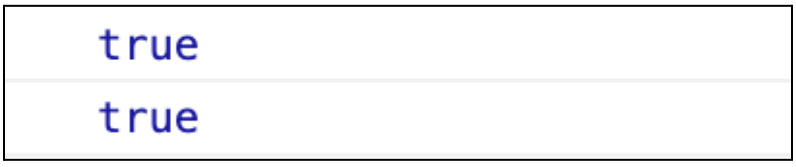
Output:-


```
true
```

Problem Statement 3: Password Checker

Write a javascript function to check whether the input string contains alphabet, numbers, dash, underscore.

Output:-



Solutions

Problem Statement 1:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Regular Expression Practice Code</title>
</head>
<body>
    <script>
        function is_hexcolor(str)
        {
        regexp = /^#?([0-9a-fA-F]{3}|[0-9a-fA-F]{6})$/;

                if (regexp.test(str))
                {
                    return true;
                }
                else
                {
                    return false;
                }
        }
        console.log(is_hexcolor("#333")); // true
        console.log(is_hexcolor("#7777")); // false
    </script>
    </body>
</html>
```

Problem Statement 2:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Regular Expression Practice Code</title>
</head>
<body>
    <script>
        let domain=window.prompt("Enter domain name","www.google.com");
        function is_domain(str)
        {
            regexp = /^[a-z0-9]+([\-\.]{1}[a-z0-9]+)*\.[a-z]{2,6}$/i;

                    if (regexp.test(str))
                    {
                        return true;
                    }
                    else
                    {
                        return false;
```

```
                    }
            }
        console.log(is_domain(domain));
    </script>
    </body>
</html>
```

Problem Statement 3:

```
<!DOCTYPE html>
<html lang="en">
<head>
   <title>Regular Expression Practice Code</title>
</head>
<body>
    <script>
        let password = prompt("Enter your password","12_345");
        function passwordChecker(password)
        {
        regexp = /^[a-z0-9_\-]+$/i;

                if (regexp.test(password))
                {
                    return true;
                }
                else
                {
                    return false;
                }
        }
        let value = passwordChecker(password);
        console.log(value);
        console.log(passwordChecker('100_23')); // true
    </script>
    </body>
</html>
```

ES6

Topics Covered:

- JavaScript ES6.
- JavaScript Let.
- JavaScript Const.
- JavaScript Arrow Function.
- JavaScript Array methods.
- JavaScript Objects.
- JavaScript Classes.
- JavaScript Strings.

JavaScript ES6:

- JavaScript ES6 is also known as ECMAScript 2015 or ECMAScript 6.
- ECMAScript is the standard that JavaScript programming language uses.
- ECMAScript provides the specification on how the JavaScript programming language should work.

JavaScript Let:

- JavaScript ***let*** is used to declare variables. Previously, variables were declared using only the ***var*** keyword.
- The variables declared using let are block-scoped.
- The variables declared using let are only accessible within a particular block.
- Example:

```
// variable declared using let
let name = 'Sara';
{
    // can be accessed only inside
    let name = 'Peter';

    console.log(name); // Peter
}
console.log(name); // Sara
```

Difference between let and var:

let	var
Block scoped variable	Function scoped variable
Does not allows to redeclare variables	Allows to redeclare variables
Hoisting does not occur in let	Hoisting occurs in var

JavaScript Const:

- The const keyword is used to declare constants in JavaScript.
- Constants are similar to let variables, except that the value cannot be changed.
- Once declared the value of the const variable cannot be changed.
- Example:

```
// name declared with const cannot be changed
const name = 'Sara';
```

JavaScript Arrow Function:

- Arrow functions are the short syntax for writing function expressions.
- To write an arrow function you don't need function keyword, return keyword, and curly brackets.
- The return statement and the curly braces are not needed only when the function is a single statement.
- Arrow functions allows to create a function in a cleaner way compared to regular functions.
- Syntax:

```
let myFunction = (arg1, arg2, ...argN) => {
    statement(s)
}
```

- Where,
    - myFunction → name of the function.
    - arg1, arg2, ….argN → function arguments.
    - statement(s) → function body.
- Arrow function VS Regular function:

```
// function expression using arrow function
let x = (x, y) => x * y;
```

```
// function expression
let x = function(x, y) {
   return x * y;
}
```

- Example: Arrow function with no argument:

```
let greet = () => console.log('Hello');
greet(); // Hello
```

- Example: Arrow function with one argument:

```
let greet = x => console.log(x);
greet('Hello'); // Hello
```

- Example: Arrow function as expression:

```
let age = 5;

let welcome = (age < 18) ?
  () => console.log('Baby') :
  () => console.log('Adult');

welcome(); // Baby
```

- Example: Multiline arrow function:

```
let sum = (a, b) => {
    let result = a + b;
    return result;
}

let result1 = sum(5,7);
console.log(result1); // 12
```

- Arrow functions do not have their own this.
- Arrow functions are not hoisted; they must be defined before they are used.

- Using const keywords is safer than using var/let keywords, because a function expression is always a constant value.
- Arrow function should not be used to create methods inside objects.

```
let person = {
    name: 'Jack',
    age: 25,
    sayName: () => {

        // this refers to the global .....
        //
        console.log(this.age);
    }
}

person.sayName(); // undefined
```

- Arrow function cannot be used as a constructor.

```
let Foo = () => {};
let foo = new Foo(); // TypeError: Foo is not a constructor
```

JavaScript Array methods:

Array.form()

- The Array.form() method returns an array object with a length property or any iterable object.
- HTML:

```
<p id="demo"></p>
```

- JS:

```
const myArr = Array.from("ABCDEFG");
document.getElementById("demo").innerHTML = myArr;
```

- Output:

A,B,C,D,E,F,G

Array keys()

- The keys() method returns an array iterator object with keys of an array.
- HTML:

```
<p id="demo"></p>
```

- JS:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const keys = fruits.keys();

let text = "";
for (let x of keys) {
  text += x + "<br>";
}

document.getElementById("demo").innerHTML = text;
```

- Output:

0
1
2
3

Array find()

- The find() method returns the value of the first array element that passes a test function.
- HTML:

```
<h2>JavaScript Array.find()</h2>
<p id="demo"></p>
```

- JS:

```
const numbers = [4, 9, 16, 25, 29];
let first = numbers.find(myFunction);

document.getElementById("demo").innerHTML = "First number over 18 is " +
first;

function myFunction(value, index, array) {
  return value > 18;
}
```

- Output:

JavaScript Array.find()

First number over 18 is 25

Array findIndex()

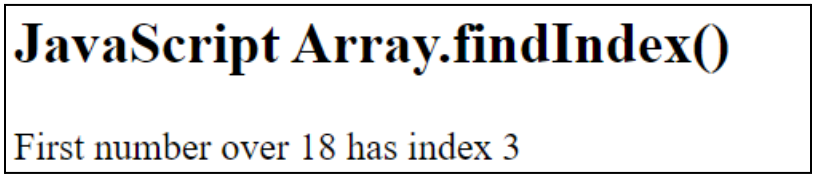
- The findIndex() method returns the index of the first array element that passes a test function.
- HTML:

```
<h2>JavaScript Array.findIndex()</h2>
<p id="demo"></p>
```

- JS:

```
const numbers = [4, 9, 16, 25, 29];

document.getElementById("demo").innerHTML = "First number over 18 has
index " + numbers.findIndex(myFunction);

function myFunction(value, index, array) {
  return value > 18;
}
```

- Output:



JavaScript Objects:

object.entries() and object.values()

- There are two ways to access objects, object.entries() and object.values().
- object.values() returns an array of all values of the object.
- object.entries() returns an array that contains both keys and values.
- Example - object.values():
  - HTML:

```
<p id="demo"></p>
```

  - JS:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const f = fruits.values();

for (let x of f) {
  document.getElementById("demo").innerHTML += x + "<br>";
}
```

  - Output:

Banana
Orange
Apple
Mango

- Example - object.entries():
  - HTML:

```
<p id="demo"></p>
```

  - JS:

```
const fruits = ["Banana", "Orange", "Apple", "Mango"];
const f = fruits.entries();

for (let x of f) {
  document.getElementById("demo").innerHTML += x + "<br>";
}
```

  - Output:

0,Banana
1,Orange
2,Apple
3,Mango

JavaScript Classes:

- A class is a blueprint for the object. Objects can be created from class.
- Class is a sketch of a house, which contains all details about the floors, doors, windows, etc…
- JavaScript class is similar to constructor function.
- Constructor function:

```
// constructor function
function Person () {
    this.name = 'John',
    this.age = 23
}

// create an object
const person1 = new Person();
```

- Instead of using a function keyword, a class keyword should be used to create a class.

- Example:

```
// creating a class
class Person {
  constructor(name) {
    this.name = name;
  }
}
```

- Properties are assigned in a constructor function, and objects can be created for the class.
- Objects for above mentioned class “Person”:

```
// creating an object
const person1 = new Person('John');
const person2 = new Person('Jack');

console.log(person1.name); // John
console.log(person2.name); // Jack
```

- person1 and person2 are objects of Person class.
- The constructor method inside the class gets called automatically each time an object is created.
- It is easy to define and access methods in JavaScript class by
    - Defining: Giving a method name followed by ().
    - Accessing: Call the method using its name followed by ().

```
class Person {
    constructor(name) {
    this.name = name;
  }

    // defining method
    greet() {
        console.log(`Hello ${this.name}`);
    }
}

let person1 = new Person('John');

// accessing property
console.log(person1.name); // John

// accessing method
person1.greet(); // Hello John
```

- In JavaScript, getter methods get the value of an object and setter methods set the value of an object.
- Example:

```
class Person {
    constructor(name) {
        this.name = name;
    }

    // getter
    get personName() {
        return this.name;
    }

    // setter
    set personName(x) {
        this.name = x;
    }
}

let person1 = new Person('Jack');
console.log(person1.name); // Jack

// changing the value of name property
person1.personName = 'Sarah';
console.log(person1.name); // Sarah
```

JavaScript Strings:

string.includes()

- The includes() method returns true if a string contains a specific value, else false.
- HTML:

```
<p>Check if a string includes "world":</p>

<p id="demo"></p>
```

- JS:

```
let text = "Hello world, welcome to the universe.";
document.getElementById("demo").innerHTML = text.includes("world");
```

- Output:

Check if a string includes "world":

true

string.startsWith()

- The startsWith() method returns true if a string begins with a specified value, otherwise false.
- HTML:

```
<p>Check if a string starts with "Hello":</p>

<p id="demo"></p>
```

- JS:

```
let text = "Hello world, welcome to the universe.";
document.getElementById("demo").innerHTML = text.startsWith("Hello");
```

- Output:

Check if a string starts with "Hello":

true

string.endsWith()

- The endsWith() method returns true if a string ends with a specified value, otherwise false.
- HTML:

```
<p>Check if a string ends with "end":</p>

<p id="demo"></p>
```

- JS:

```
let text = "Hello World";
document.getElementById("demo").innerHTML = text.endsWith("end");
```

- Output:

Check if a string ends with "end":

false

Assignment - ES6

(This is a minor assignment and submission is not required. Will release solution next week so that you can self- evaluate)

Challenge 1:Block and Function Scope
Steps:

- Open VS code.
- Select New File in the opened folder…
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Copy and Add the below given HTML code in the file.

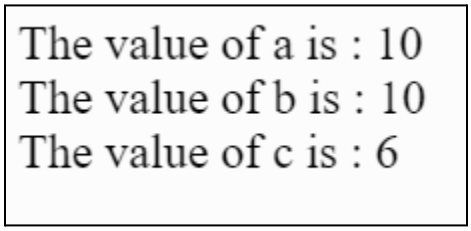
```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>repl.it</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <script src="script.js"></script>
  </body>
</html>
```

- Copy and Add the below given JS code in the file.

```
var a=10; // Here value of a is 10
var b=8; // Here value of b is 8
var c=6; // Here value of c is 6
{
  let a = 2;
  // Here value of a is 2 but only within the block
  var b = 10;
  // Here value of b is 10 inside and outside the block it will remain 10
  const c = 8;
  // Here value of c is 8 but only within the block
  // const cannot be reassigned
  // c=5;
}
document.write("The value of a is : " +a);
// let has block scope
document.write("<br>");
```

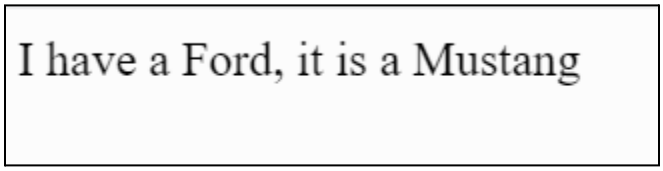
```
document.write("The value of b is : " +b);
// var does not have block scope
document.write("<br>");
document.write("The value of c is  : " +c);
// const has block scope
```

- View the file in the browser, by right click on the file name in the left pane.
- View of file in browser.



Challenge 2: ES6 Class

With your new gained knowledge in ES6 class, create a class with constructor, child class, method, object. Using the object access the method and class.

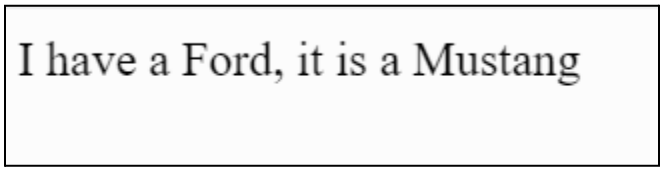


This is the sample output image, the text given is just for sample

ES6 Assignment - Solution

Problem statement:

With your new gained knowledge in ES6 class, create a class with constructor, child class, method, object. Using the object access the method and class.



This is the sample output image, the text given is just for sample

Steps:

- Open VS code.
- Select New File in the opened folder…

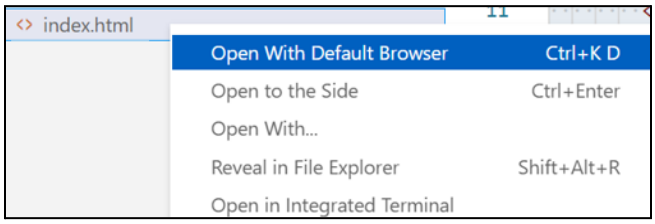
- Save the file in the appropriate folder/ create the folder and save the file with extension.
- Add the following code in the index.html file.

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width">
    <title>repl.it</title>
    <link href="style.css" rel="stylesheet" type="text/css" />
  </head>
  <body>
    <script src="script.js"></script>
  </body>
</html>
```

- Add the following code in the scripts.js file.

```
class Car {
  constructor(name) {
    this.brand = name;
  }
  present() {
    return 'I have a ' + this.brand;
  }
}

class Model extends Car {
  constructor(name, mod) {
    super(name);
    this.model = mod;
  }
  show() {
    return this.present() + ', it is a ' + this.model
  }
}

mycar = new Model("Ford", "Mustang");

document.write(mycar.show());
```

- View the file in the browser, by right click on the file name in the left pane.



Sample Output: