

JAVASCRIPT

JavaScript is the world's most popular programming language.

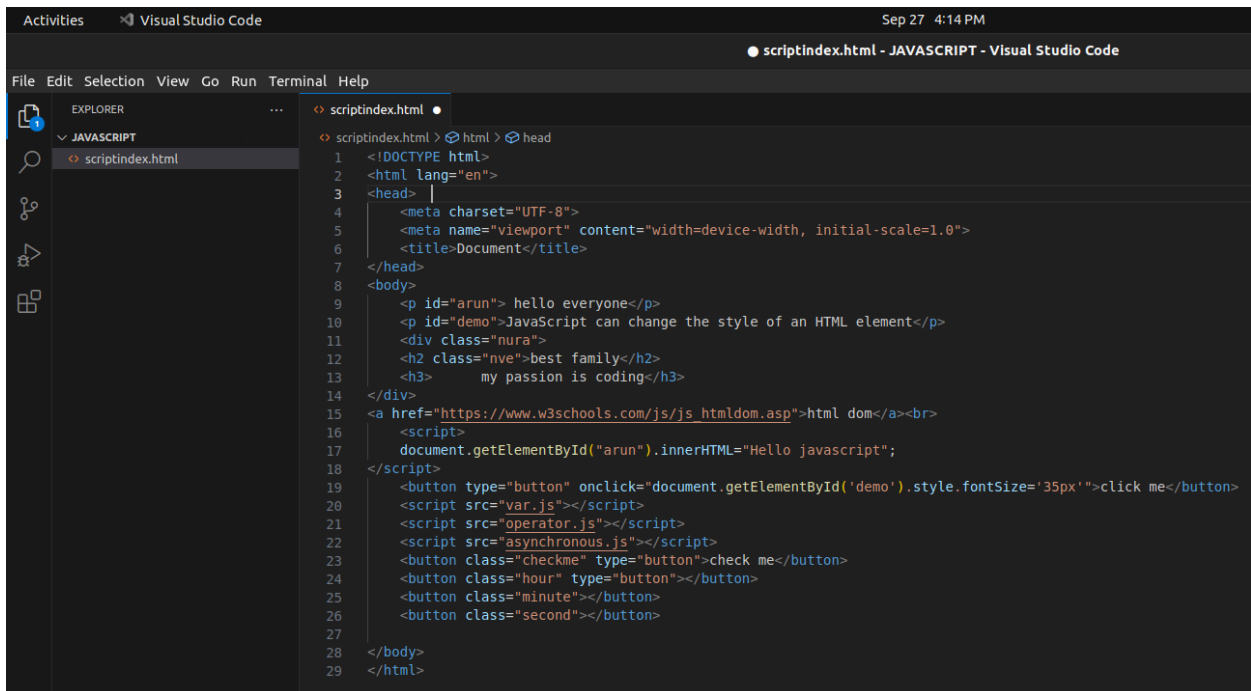
JavaScript is the programming language of the Web.

The <script> Tag

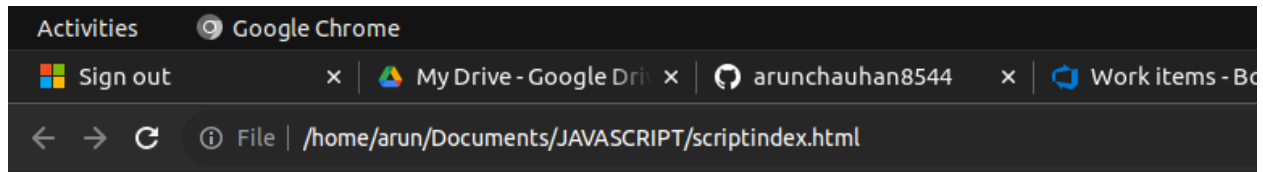
In HTML, JavaScript code is inserted between <script> and </script> Tags.

JavaScript Can Change HTML Content

One of many JavaScript HTML methods is getElementById().



```
1 <!DOCTYPE html>
2 <html lang="en">
3 <head>
4   <meta charset="UTF-8">
5   <meta name="viewport" content="width=device-width, initial-scale=1.0">
6   <title>Document</title>
7 </head>
8 <body>
9   <p id="arun"> hello everyone</p>
10  <p id="demo">JavaScript can change the style of an HTML element</p>
11  <div class="nura">
12    <h2 class="nve">best family</h2>
13    <h3>    my passion is coding</h3>
14  </div>
15  <a href="https://www.w3schools.com/js/js_htmldom.asp">html dom</a><br>
16  <script>
17    document.getElementById("arun").innerHTML="Hello javascript";
18  </script>
19  <button type="button" onclick="document.getElementById('demo').style.fontSize='35px'">click me</button>
20  <script src="var.js"></script>
21  <script src="operator.js"></script>
22  <script src="asynchronous.js"></script>
23  <button class="checkme" type="button">check me</button>
24  <button class="hour" type="button"></button>
25  <button class="minute"></button>
26  <button class="second"></button>
27
28 </body>
29 </html>
```



Hello javascript

JavaScript can change the style of an HTML element

best family

my passion is coding

[html dom](#)

External References

An external script can be referenced in 3 different ways:

- With a full URL (a full web address)
- With a file path (like `/js/`)
- Without any path

JavaScript Display Possibilities

JavaScript can "display" data in different ways:

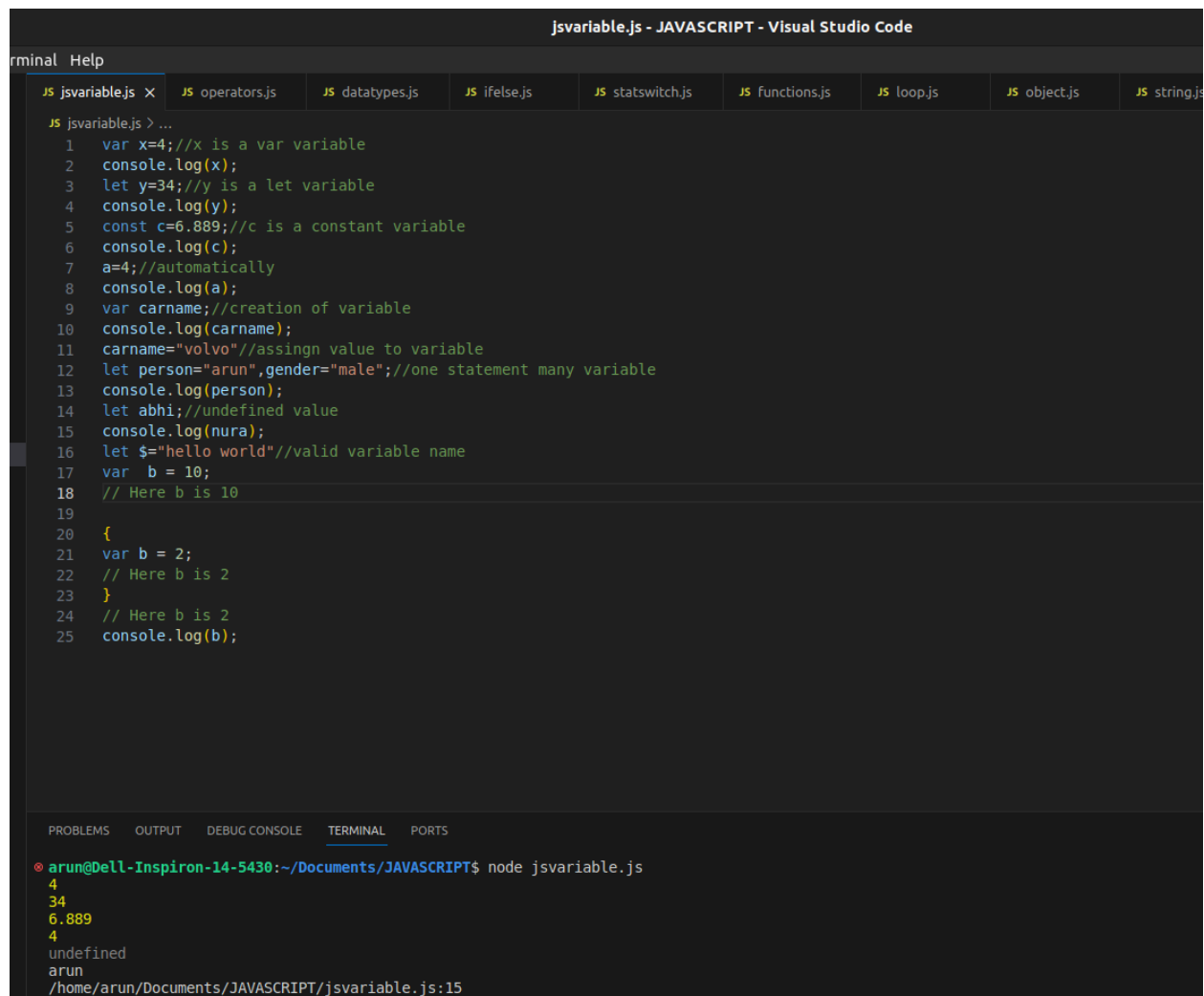
- Writing into an HTML element, using `innerHTML`.
- Writing into the HTML output using `document.write()`.
- Writing into an alert box, using `window.alert()`.
- Writing into the browser console, using `console.log()`.

JavaScript Variables

JavaScript Variables can be declared in 4 ways:

- Automatically
- Using var
- Using let
- Using const

You cannot re-declare a variable declared with let or const.



The screenshot shows the Visual Studio Code editor with a file named `jsvariable.js` open. The code in the file demonstrates various ways to declare and use variables in JavaScript. The terminal at the bottom shows the output of running `node jsvariable.js`.

```
jsvariable.js - JAVASCRIPT - Visual Studio Code
terminal Help
JS jsvariable.js x JS operators.js JS datatypes.js JS ifelse.js JS statswitch.js JS functions.js JS loop.js JS object.js JS string.js
JS jsvariable.js > ...
1 var x=4;//x is a var variable
2 console.log(x);
3 let y=34;//y is a let variable
4 console.log(y);
5 const c=6.889;//c is a constant variable
6 console.log(c);
7 a=4;//automatically
8 console.log(a);
9 var carname;//creation of variable
10 console.log(carname);
11 carname="volvo";//assignn value to variable
12 let person="arun",gender="male";//one statement many variable
13 console.log(person);
14 let abhi;//undefined value
15 console.log(nura);
16 let $="hello world";//valid variable name
17 var b = 10;
18 // Here b is 10
19
20 {
21   var b = 2;
22   // Here b is 2
23 }
24 // Here b is 2
25 console.log(b);

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node jsvariable.js
4
34
6.889
4
undefined
arun
/home/arun/Documents/JAVASCRIPT/jsvariable.js:15
```

Types of JavaScript Operators

There are different types of JavaScript operators:

- Arithmetic Operators
 - Assignment Operators
 - Comparison Operators
 - String Operators
 - Logical Operators
 - Bitwise Operators
-
- Ternary Operators
 - Type Operators

The image shows a Visual Studio Code editor window titled "operators.js - JAVASCRIPT - Visual Studio Code". The editor has several tabs open: "js variable.js", "JS operators.js" (active), "JS datatypes.js", "JS ifelse.js", "JS statswitch.js", "JS functions.js", "JS loop.js", "JS object.js", and "JS string.js". The "operators.js" file contains the following code:

```
JS operators.js > ...
1 //arithmetic operator
2 let x=35;
3 let y=7;
4
5 console.log(x + y); // Addition
6 console.log(x- y); // Subtraction
7 console.log(x * y); // Multiplication
8 console.log(x / y); // Division
9 console.log(x % y); // Modulo
10
11 // assignment Operators
12 let c=10;//=operator
13 c+=30;
14 console.log(c);
15
16 //comparison operator
17 console.log(x == y);
18 console.log(x === y);
19 console.log(x!=y);
20 console.log(x !== y);
21 console.log(x + y);
22 //string operator
23 let text1 = "A";
24 let text2 = "B";
25 console.log(text1 <text2);
```

Below the editor, the "TERMINAL" panel is active, showing the command "node operators.js" and its output:

```
arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node operators.js
42
28
245
5
0
40
false
false
true
true
42
true
arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

JavaScript Data Types

JavaScript has 8 Datatypes

1. String
2. Number

3. BigInt
4. Boolean
5. Undefined
6. Null
7. Symbol
8. Object

The Object Datatype

The object data type can contain:

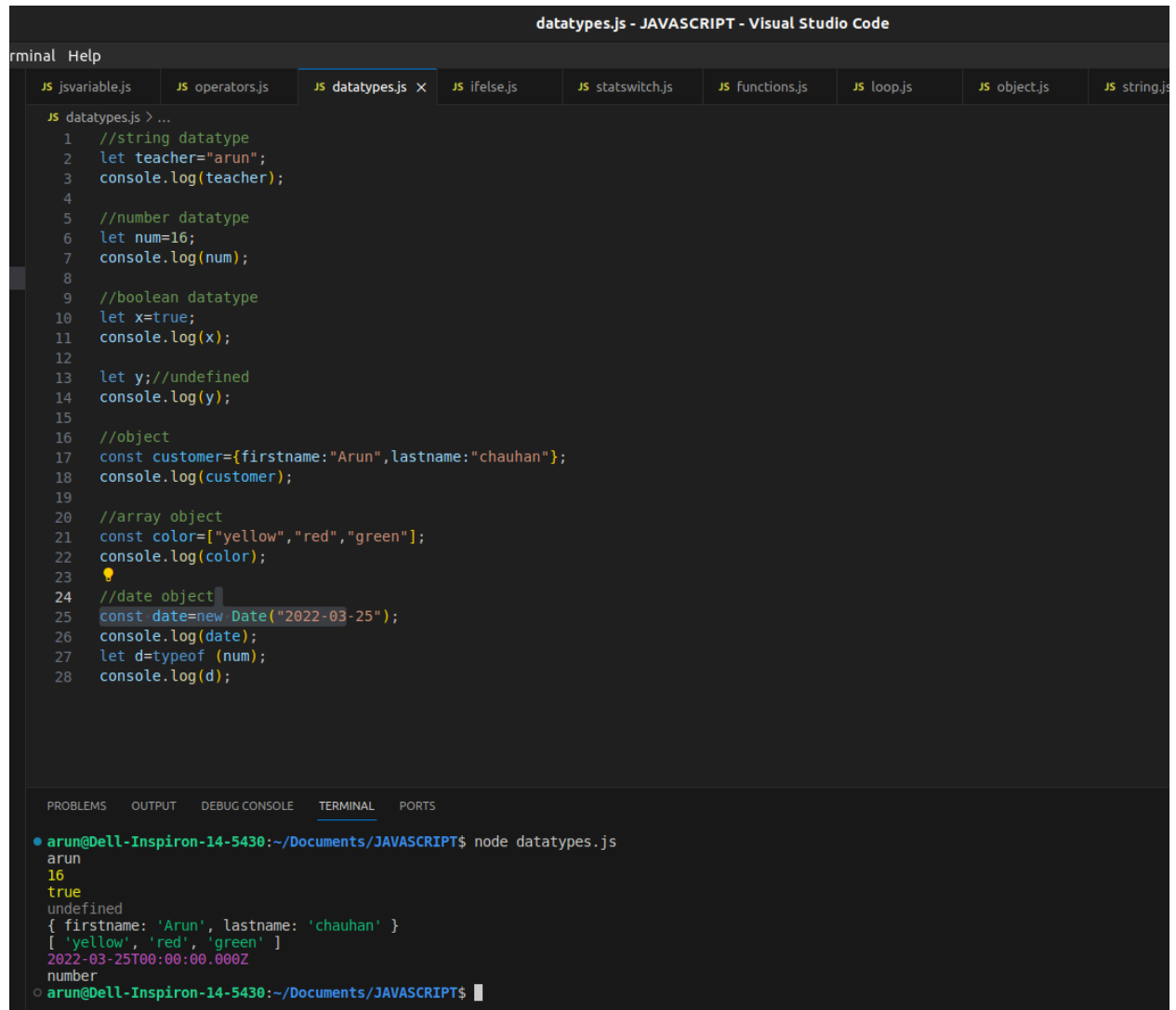
1. An object
2. An array
3. A date

Javascript numbers are always one type:
double (64-bit floating point).

The typeof Operator

You can use the JavaScript typeof operator to find the type of a JavaScript variable.

An empty string has both a legal value and a type.



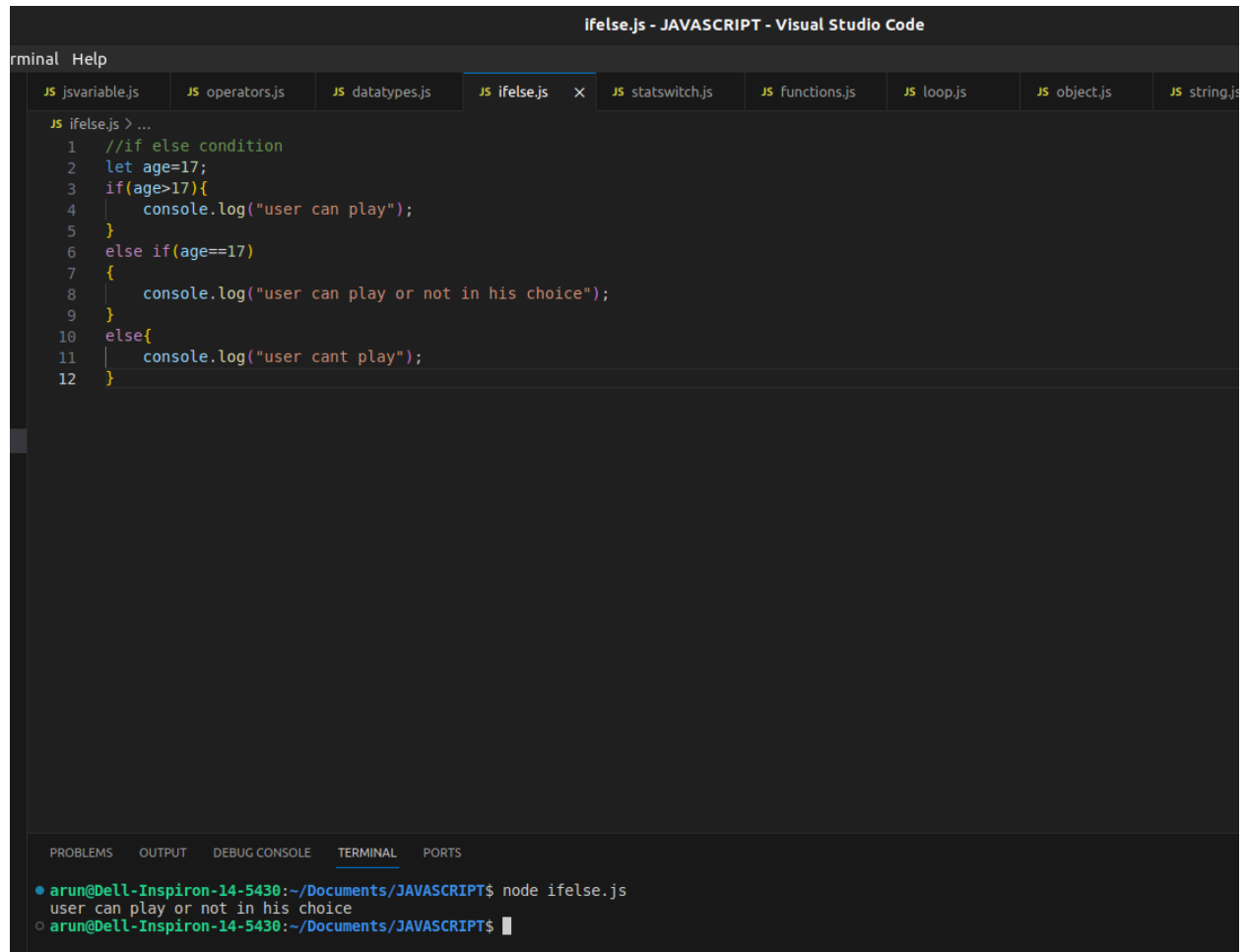
The image shows a Visual Studio Code window titled 'datatypes.js - JAVASCRIPT - Visual Studio Code'. The editor displays a JavaScript file with the following code:

```
1 //string datatype
2 let teacher="arun";
3 console.log(teacher);
4
5 //number datatype
6 let num=16;
7 console.log(num);
8
9 //boolean datatype
10 let x=true;
11 console.log(x);
12
13 let y;//undefined
14 console.log(y);
15
16 //object
17 const customer={firstname:"Arun",lastname:"chauhan"};
18 console.log(customer);
19
20 //array object
21 const color=["yellow","red","green"];
22 console.log(color);
23
24 //date object
25 const date=new Date("2022-03-25");
26 console.log(date);
27 let d=typeof (num);
28 console.log(d);
```

The bottom panel shows the 'TERMINAL' tab with the following output:

```
arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node datatypes.js
arun
16
true
undefined
{ firstname: 'Arun', lastname: 'chauhan' }
[ 'yellow', 'red', 'green' ]
2022-03-25T00:00:00.000Z
number
arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

JavaScript if, else, and else if



The image shows a Visual Studio Code editor window titled "ifelse.js - JAVASCRIPT - Visual Studio Code". The editor has several tabs open: "JS jsvariable.js", "JS operators.js", "JS datatypes.js", "JS ifelse.js" (which is the active tab), "JS statswitch.js", "JS functions.js", "JS loop.js", "JS object.js", and "JS string.js". The code in the active tab is as follows:

```
JS ifelse.js > ...
1 //if else condition
2 let age=17;
3 if(age>17){
4     console.log("user can play");
5 }
6 else if(age==17)
7 {
8     console.log("user can play or not in his choice");
9 }
10 else{
11     console.log("user cant play");
12 }
```

At the bottom of the window, there is a terminal panel with tabs for "PROBLEMS", "OUTPUT", "DEBUG CONSOLE", "TERMINAL", and "PORTS". The "TERMINAL" tab is active, showing the command "node ifelse.js" being executed. The output of the command is "user can play or not in his choice".

The JavaScript Switch Statement

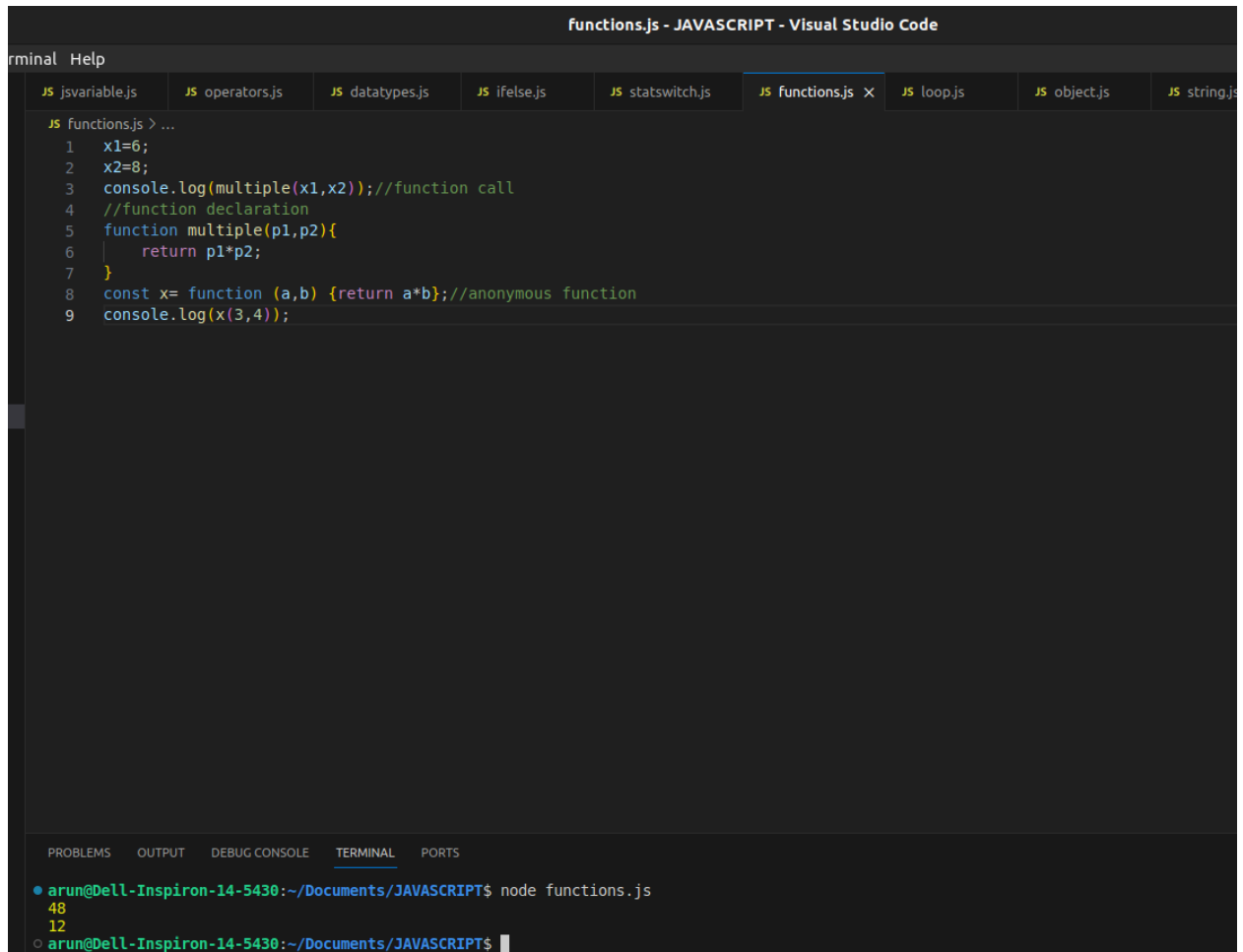

```
statswitch.js - JAVASCRIPT - Visual Studio Code
terminal Help
JS jsvariable.js JS operators.js JS datatypes.js JS ifelse.js JS statswitch.js X JS functions.js JS loop.js JS object.js JS string.js

JS statswitch.js > ...
1 //conditional operator
2 let age=18;
3 let drink= age>=5? "coffee" : "milk";
4 console.log(drink);
5 let day=3;
6 switch(day){
7   case 0:
8     console.log("sunday");
9     break;
10  case 1:
11    console.log("monday");
12    break;
13  case 2:
14    console.log("sunday");
15    break;
16  case 3:
17    console.log("sunday");
18    break;
19  case 4:
20    console.log("sunday");
21    break;
22  case 5:
23    console.log("sunday");
24    break;
25  case 6:
26    console.log("sunday");
27    break;
28  default:
29    console.log("not found");
30    break;
31 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node statswitch.js
coffee
sunday
○ arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

JavaScript Functions

A JavaScript function is a block of code designed to perform a particular task. A JavaScript function is executed when "something" invokes it (calls it).



The screenshot shows the Visual Studio Code interface with a file named 'functions.js' open. The code in the editor is as follows:

```
1 x1=6;
2 x2=8;
3 console.log(multiple(x1,x2)); //function call
4 //function declaration
5 function multiple(p1,p2){
6     return p1*p2;
7 }
8 const x= function (a,b) {return a*b}; //anonymous function
9 console.log(x(3,4));
```

The terminal at the bottom shows the command 'node functions.js' being executed, with the output '48' and '12' displayed on separate lines.

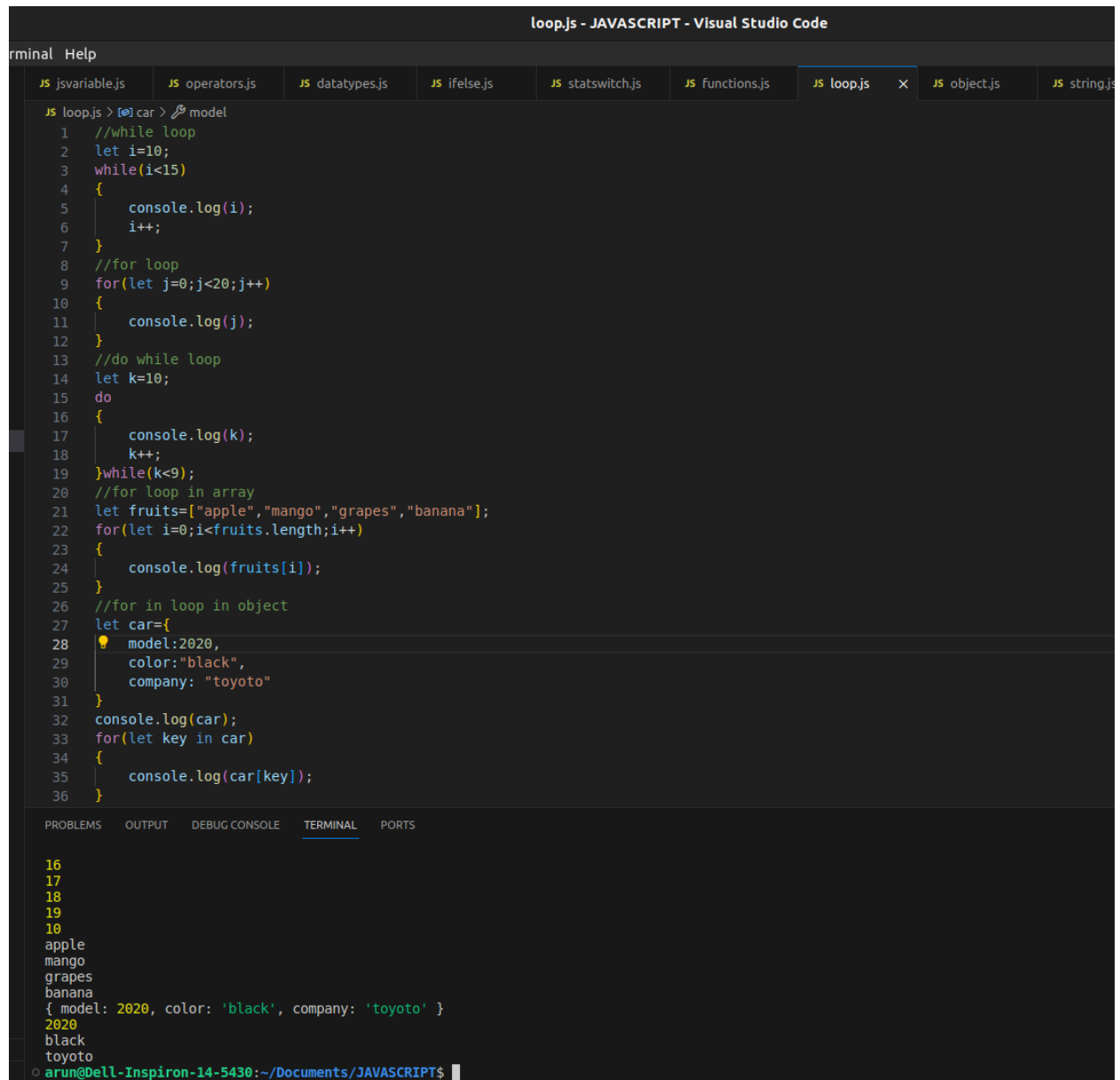
JavaScript Loop

Different Kinds of Loops

JavaScript supports different kinds of loops:

- for - loops through a block of code a number of times
- for/in - loops through the properties of an object
- for/of - loops through the values of an iterable object
- while - loops through a block of code while a specified condition is true

- do/while - also loops through a block of code while a specified condition is true



The screenshot shows the Visual Studio Code editor with a file named `loop.js`. The code contains several JavaScript loops and a console log for an object. The terminal at the bottom shows the output of the code execution.

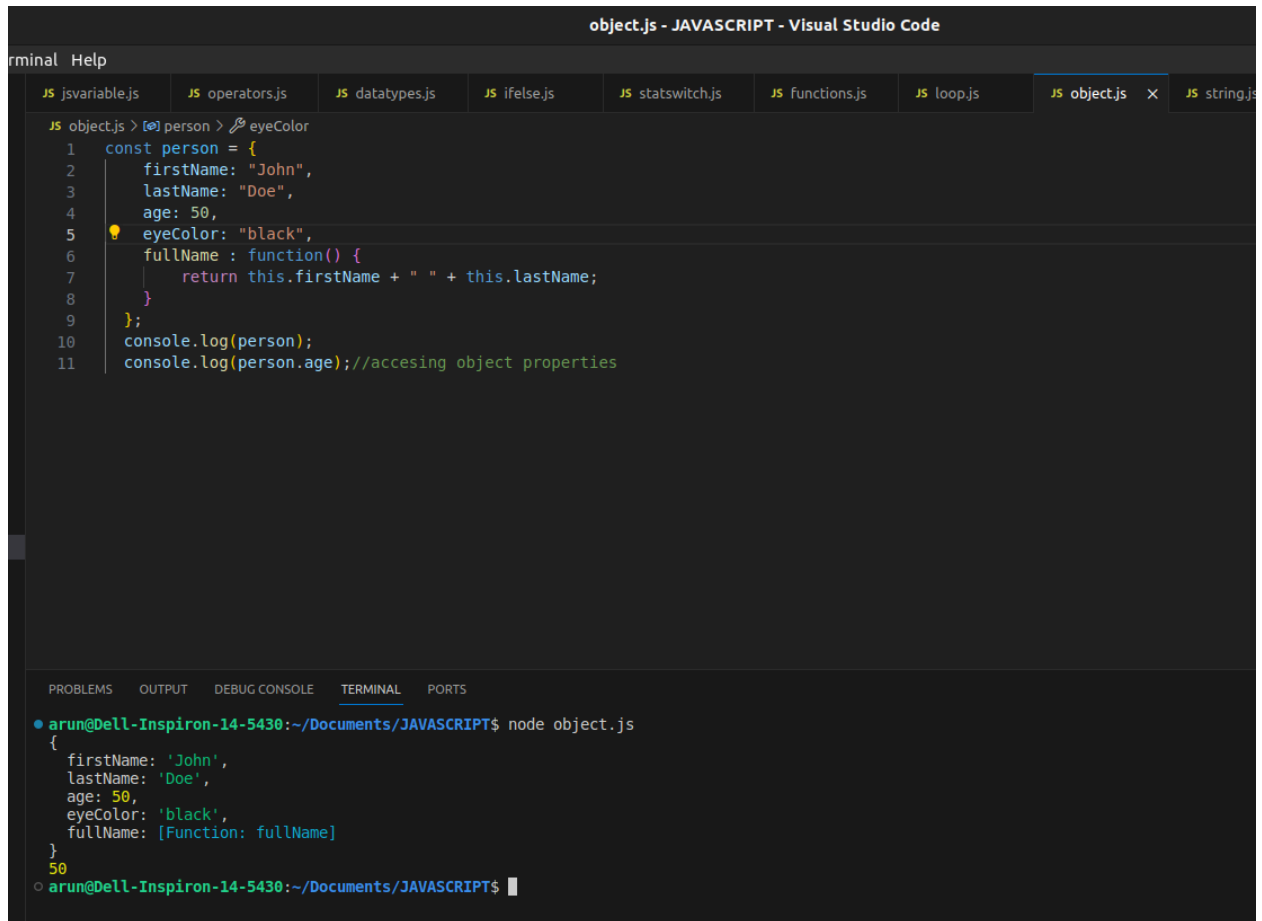
```
JS loop.js > [0] car > model
1 //while loop
2 let i=10;
3 while(i<15)
4 {
5     console.log(i);
6     i++;
7 }
8 //for loop
9 for(let j=0;j<20;j++)
10 {
11     console.log(j);
12 }
13 //do while loop
14 let k=10;
15 do
16 {
17     console.log(k);
18     k++;
19 }while(k<9);
20 //for loop in array
21 let fruits=["apple","mango","grapes","banana"];
22 for(let i=0;i<fruits.length;i++)
23 {
24     console.log(fruits[i]);
25 }
26 //for in loop in object
27 let car={
28     model:2020,
29     color:"black",
30     company: "toyoto"
31 }
32 console.log(car);
33 for(let key in car)
34 {
35     console.log(car[key]);
36 }
```

Terminal Output:

```
16
17
18
19
20
apple
mango
grapes
banana
{ model: 2020, color: 'black', company: 'toyoto' }
2020
black
toyoto
arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

JavaScript Objects

The values are written as name:value pairs (name and value separated by a colon).



The screenshot shows the Visual Studio Code interface with a file named 'object.js' open. The code defines a JavaScript object 'person' with properties 'firstName', 'lastName', 'age', 'eyeColor', and a 'fullName' function. The terminal shows the output of running 'node object.js', displaying the object's structure and the value of 'age' (50).

```
object.js - JAVASCRIPT - Visual Studio Code
terminal Help
JS jsvariable.js JS operators.js JS datatypes.js JS ifelse.js JS statswitch.js JS functions.js JS loop.js JS object.js JS string.js

JS object.js > person > eyeColor
1 const person = {
2   firstName: "John",
3   lastName: "Doe",
4   age: 50,
5   eyeColor: "black",
6   fullName: function() {
7     return this.firstName + " " + this.lastName;
8   }
9 };
10 console.log(person);
11 console.log(person.age); //accessing object properties

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
• arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node object.js
{
  firstName: 'John',
  lastName: 'Doe',
  age: 50,
  eyeColor: 'black',
  fullName: [Function: fullName]
}
50
○ arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

JavaScript Strings

JavaScript strings are for storing and manipulating text.

Javascript string methods-

1. **Length**: The length property returns the number of characters in the string. It doesn't require parentheses as it is not a method but a property of the string.
2. **Character Access**: Strings are zero-indexed, meaning you can access individual characters using their index positions. You can use the `charAt(index)` method or directly access the characters using square brackets notation (e.g., `str[0]`).

3. **Concatenation:** The `concat()` method is used to join two or more strings together, creating a new string that includes all the concatenated parts.

4. **Substring Extraction:** You can extract a substring from a string using methods like `substring(startIndex, endIndex)` and `slice(startIndex, endIndex)`. Both methods return the extracted substring based on the provided indices. `substring()` will swap the indices if `startIndex` is greater than `endIndex`, while `slice()` allows negative indices to count from the end of the string.

5. **Case Conversion:** To convert the case of a string, you can use `toLowerCase()` and `toUpperCase()` methods, which return new strings with all characters in either lowercase or uppercase, Respectively.

6. **Trimming:** The `trim()` method removes leading and trailing whitespaces from a string, creating a new string with the trimmed Content.

7. **Searching:** For finding substrings within a string, you can use `indexOf(substring, startIndex)` method. It returns the index of the first occurrence of the substring or -1 if the substring is not found. To search from the end of the string, you can use `lastIndexOf(substring, startIndex)`.

8. **Splitting:** The `split(separator)` method is used to split a string into an array of substrings based on the specified separator. For example, splitting a comma-separated string into an array of Values.

9. **Replacing:** The `replace(searchValue, replaceValue)` method

string.js - JAVASCRIPT - Visual Studio Code

terminal Help

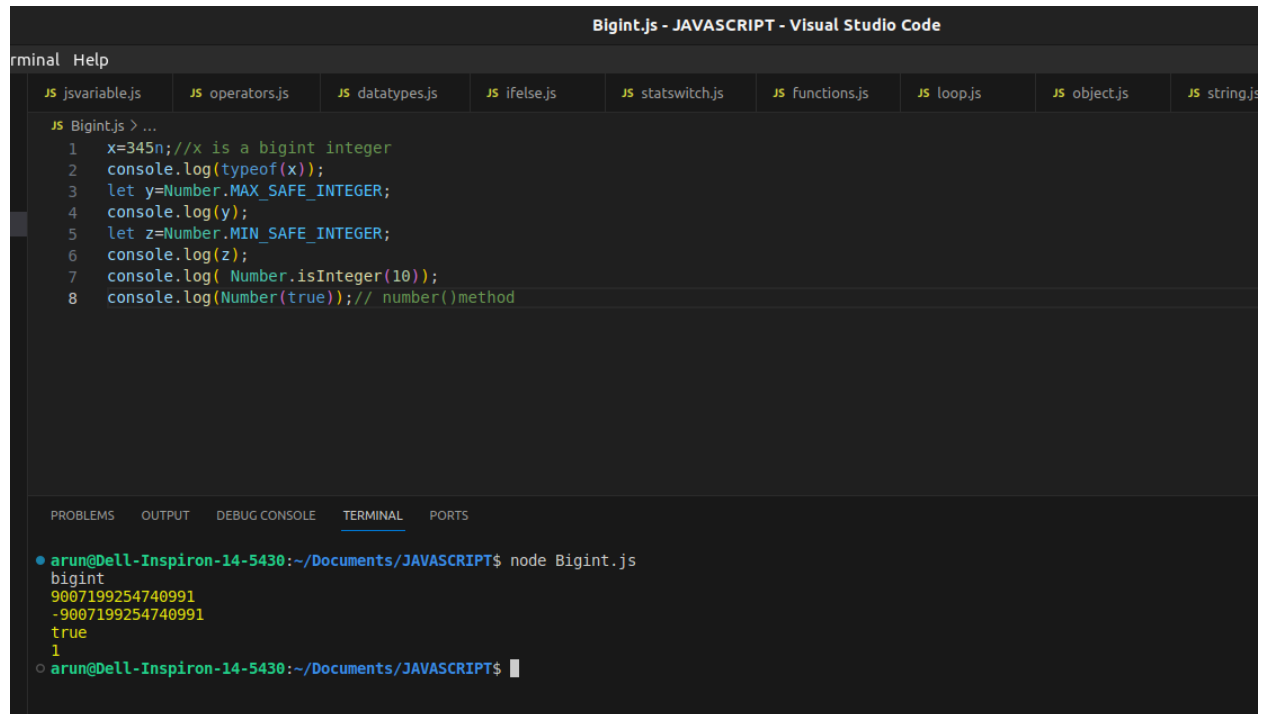
JS jsvariable.js JS operators.js JS datatypes.js JS ifelse.js JS statswitch.js JS functions.js JS loop.js JS object.js JS string.js

```
JS string.js > [0] arun
1 let firstName="Arun";
2 console.log(firstName[1]); //string indexing
3 let lastName=" Chauhan "
4 console.log(lastName);
5 let newString=lastName.trim(); //trim()
6 console.log(lastName.length); //length()
7 console.log(newString.length);
8 let text="abcdEfgHijkLMn";
9 console.log(text.toUpperCase()); //toUpperCase()
10 console.log(text.slice(4,8)); //string slice()
11 //convert string to a number
12 let num=+"abcdEfgHijkLMn";
13 console.log(typeof(num));
14 console.log(typeof(text));
15 //convert number to string
16 num=num+" "
17 console.log(typeof(num));
18 num=Number(num);
19 console.log(typeof(num));
20 num=String(num);
21 console.log(typeof(num));
22 let text2="welcome to js";
23 let newText=text2.replace("js","css");
24 text3= "Hello" + " " + "World!"; //string concatenation
25 console.log(text3);
26 newarr1=text3.split(" ");
27 console.log(newarr1[1]);
28 let arun=`arun chauhan`; //template literals
29 console.log(arun);
30 let text4= `Welcome ${text2}, ${arun}!`;
31 console.log(text4);
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node string.js
r
Chauhan
13
7
ABCDEFGHGIJKLMN
EfgH
number
string
string
number
string
Hello World!
World!
arun chauhan
Welcome welcome to js, arun chauhan!
○ arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

JavaScript BigInt



The image shows a Visual Studio Code editor window titled "Bigint.js - JAVASCRIPT - Visual Studio Code". The editor has a sidebar with several JavaScript files: "jsvariable.js", "js operators.js", "js datatypes.js", "js ifelse.js", "js statswitch.js", "js functions.js", "js loop.js", "js object.js", and "js string.js". The main editor area displays the content of "Bigint.js", which contains the following code:

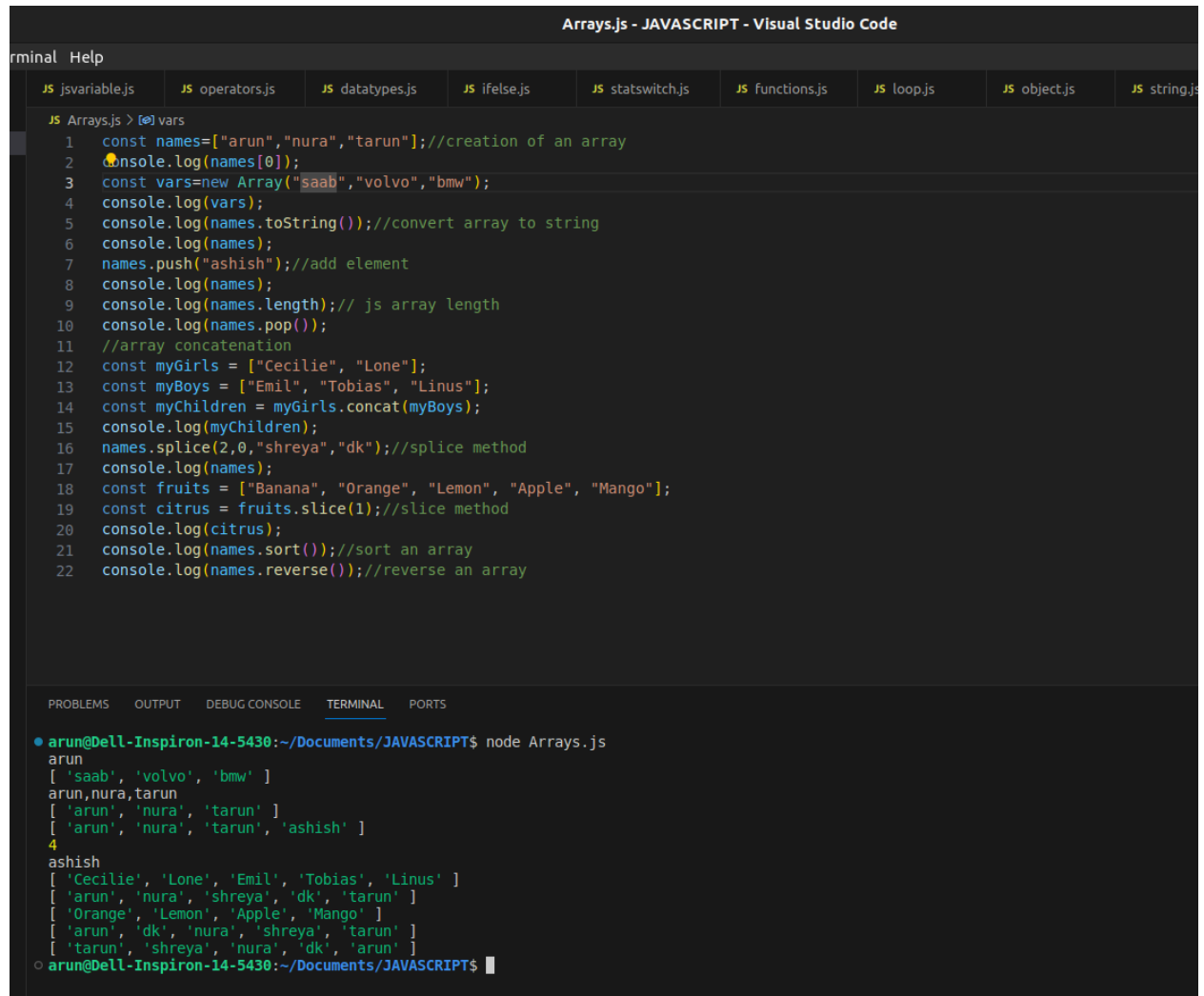
```
JS Bigint.js > ...
1  x=345n;//x is a bigint integer
2  console.log(typeof(x));
3  let y=Number.MAX_SAFE_INTEGER;
4  console.log(y);
5  let z=Number.MIN_SAFE_INTEGER;
6  console.log(z);
7  console.log( Number.isInteger(10));
8  console.log(Number(true));// number()method
```

Below the editor, the "TERMINAL" tab is active, showing the command "node Bigint.js" and its output:

```
arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node Bigint.js
bigint
9007199254740991
-9007199254740991
true
1
arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

JavaScript Arrays

An array is a special variable, which can hold more than one value:



```
Arrays.js - JAVASCRIPT - Visual Studio Code
JS jsvariable.js JS operators.js JS datatypes.js JS ifelse.js JS statswitch.js JS functions.js JS loop.js JS object.js JS string.js

JS Arrays.js > vars
1 const names=["arun","nura","tarun");//creation of an array
2 console.log(names[0]);
3 const vars=new Array("saab","volvo","bmw");
4 console.log(vars);
5 console.log(names.toString());//convert array to string
6 console.log(names);
7 names.push("ashish");//add element
8 console.log(names);
9 console.log(names.length);// js array length
10 console.log(names.pop());
11 //array concatenation
12 const myGirls = ["Cecilie", "Lone"];
13 const myBoys = ["Emil", "Tobias", "Linus"];
14 const myChildren = myGirls.concat(myBoys);
15 console.log(myChildren);
16 names.splice(2,0,"shreya","dk");//splice method
17 console.log(names);
18 const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];
19 const citrus = fruits.slice(1);//slice method
20 console.log(citrus);
21 console.log(names.sort());//sort an array
22 console.log(names.reverse());//reverse an array

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node Arrays.js
arun
[ 'saab', 'volvo', 'bmw' ]
arun,nura,tarun
[ 'arun', 'nura', 'tarun' ]
[ 'arun', 'nura', 'tarun', 'ashish' ]
4
ashish
[ 'Cecilie', 'Lone', 'Emil', 'Tobias', 'Linus' ]
[ 'arun', 'nura', 'shreya', 'dk', 'tarun' ]
[ 'Orange', 'Lemon', 'Apple', 'Mango' ]
[ 'arun', 'dk', 'nura', 'shreya', 'tarun' ]
[ 'tarun', 'shreya', 'nura', 'dk', 'arun' ]
○ arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

JAVASCRIPT DATE METHOD-

JavaScript's Date object provides methods to work with dates and times.

Here's a summary of the theory behind some commonly used date methods without specific syntax:

1. **new Date():** Creates a new Date object representing the current date and time.
2. **Date.parse(dateString):** Parses a date string and returns the

number of milliseconds since January 1, 1970 (Unix Epoch).

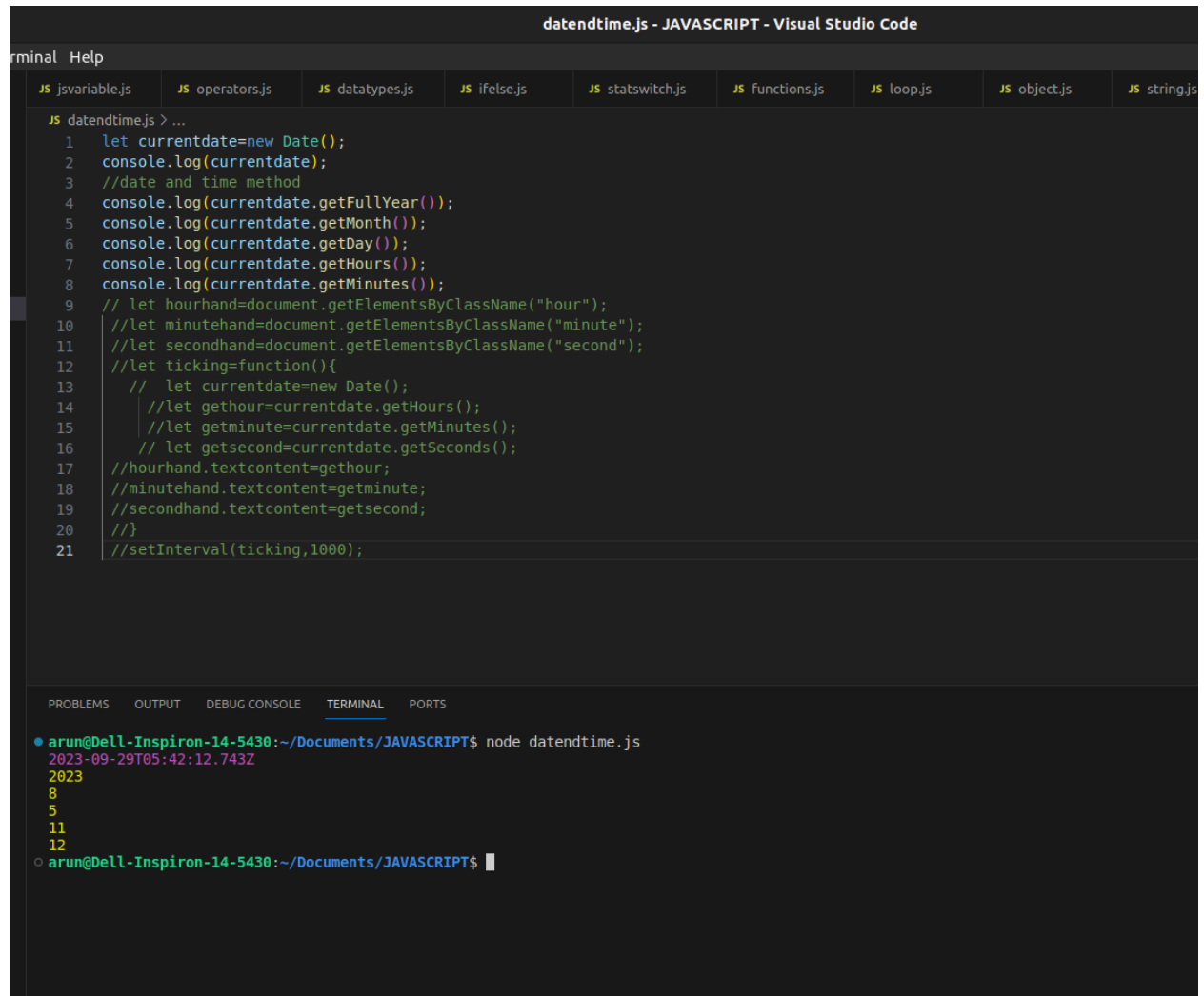
3. **getFullYear()**: Gets the year (4-digit) of the Date object.

4. **getMonth()**: Gets the month (0-indexed) of the Date object (0 for January, 1 for February, and so on).

5. **getDate()**: Gets the day of the month (1-31) of the Date object.

6. **getDay()**: Gets the day of the week (0-6) of the Date object (0 for Sunday, 1 for Monday, and so on).

7. **getHours()**, **getMinutes()**, **getSeconds()**, **getMilliseconds()**: Gets the hours (0-23), minutes (0-59), seconds (0-59), and milliseconds (0-999) of the Date object.



The image shows a screenshot of the Visual Studio Code editor. The top bar indicates the file is 'datendtime.js - JAVASCRIPT - Visual Studio Code'. Below the top bar, there is a tab bar with several files: 'jsvariable.js', 'js operators.js', 'js datatypes.js', 'js ifelse.js', 'js statswitch.js', 'js functions.js', 'js loop.js', 'js object.js', and 'js string.js'. The 'js datatypes.js' tab is active, showing the following code:

```
JS datendtime.js > ...
1 let currentdate=new Date();
2 console.log(currentdate);
3 //date and time method
4 console.log(currentdate.getFullYear());
5 console.log(currentdate.getMonth());
6 console.log(currentdate.getDay());
7 console.log(currentdate.getHours());
8 console.log(currentdate.getMinutes());
9 // let hourhand=document.getElementsByClassName("hour");
10 //let minutehand=document.getElementsByClassName("minute");
11 //let secondhand=document.getElementsByClassName("second");
12 //let ticking=function(){
13     // let currentdate=new Date();
14     //let gethour=currentdate.getHours();
15     //let getminute=currentdate.getMinutes();
16     // let getsecond=currentdate.getSeconds();
17 //hourhand.textContent=gethour;
18 //minutehand.textContent=getminute;
19 //secondhand.textContent=getsecond;
20 //}
21 //setInterval(ticking,1000);
```

At the bottom of the editor, there is a panel with tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS'. The 'TERMINAL' tab is active, showing the following output:

```
● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node datendtime.js
2023-09-29T05:42:12.743Z
2023
8
5
11
12
○ arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

JavaScript Hoisting

Hoisting is JavaScript's default behavior of moving declarations to the top.

JavaScript JSON

JSON is a format for storing and transporting data.

JSON is often used when data is sent from a server to a web page.

Sep 29 11:14 AM

data.json - JAVASCRIPT - Visual Studio Code

Terminal Help

... JS operators.js JS datatypes.js JS ifelse.js JS statswitch.js JS Functions.js JS loop.js JS object.js JS string.js JS Bi

```
{ } data.json > { } 2 > name
1  [
2    { "name": "mario", "age": 26 },
3    { "name": "arun", "age": 24 },
4    { "name": "nura", "age": 29 }
5  ]
6  ]
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

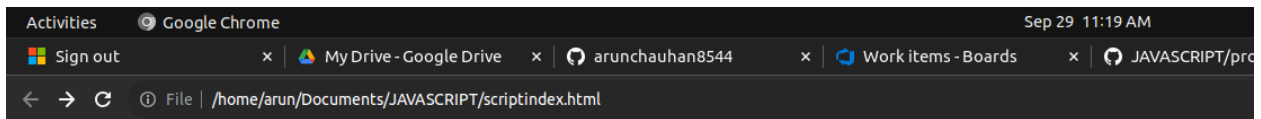
- arun@Dell-Inspiron-14-5430: ~/Documents/JAVASCRIPT\$ node data.json
- arun@Dell-Inspiron-14-5430: ~/Documents/JAVASCRIPT\$

JAVASCRIPT DOM

```
... JS datatypes.js JS ifelse.js JS statswitch.js JS functions.js JS loop.js JS object.js JS string.js JS Bigint.js JS Ar

JS dom.js > [?] parentElement
1 //Javascript DOM(document object model)
2 //created by browser as the html load into the browser
3 //browser create object of html called "document object"
4 //in document object there is model of complete html in tree like structure
5 //console.log(document.URL);
6
7 //getting html elements using querySelector/All
8 //querySelector returns "first element" that matches css selector
9 //to get all matched selector we use querySelectorAll
10
11 let ans=document.querySelector('p');
12 console.log(ans);
13 let ans2=document.querySelectorAll('p');//select all gives nodelist type
14 console.log(ans2);
15 let ans3=document.querySelector('#arun');
16 console.log(ans3);
17 //getElementsByTagName
18 let tagname=document.getElementsByTagName('p');
19 console.log(tagname);//this gives html collection type
20 //foreach method weill not work on collection.
21 //getElemntByClassName you can also use this.
22 //getElemntById
23 let id=document.getElementById('arun');
24 console.log(id);
25 //updating and changing the content
26 //inner text,ignore spaces
27 //retrieve and set content in plain text
28 let heading=document.querySelector('.nve');
29 console.log(heading.innerText);
30
31 //innertext,it does not ignore spaces
32 //retrieve and set content in html formate
33 console.log(heading.innerHTML);
34 heading.innerText += "arunchauhan";//update
35 //if u dont want to delete previous text then add +
36 console.log(heading.innerText);
37 //getting and setting attribute of element
38 let htmldomlink=document.querySelector('a');
39 console.log(htmldomlink.getAttribute('href'));
```

```
... JS datatypes.js JS ifelse.js JS statswitch.js JS functions.js JS loop.js JS object.js JS string.js JS Bigint.js JS Ar
JS dom.js > [0] parentElement
39 console.log(htmlDomLink.getAttribute('href'));
40 //set the attribute
41 htmlDomLink.setAttribute('href','https://www.w3schools.com/css/default.asp');
42 console.log(htmlDomLink.getAttribute('href'));
43 htmlDomLink.innerText="css tutorial";
44 console.log(htmlDomLink.innerText);
45 //adding the style
46 let color1=document.querySelector('h2');
47 color1.style.color="red";
48 color1.style.backgroundColor="yellow";
49
50 //add,remove and replace class of element
51 //this also helps to change css dynamically
52 let heading2=document.querySelector('h2');
53 heading2.classList.add('newClass');
54 //remove the class
55 heading2.classList.remove("newClass");
56 //replace the class
57 heading2.classList.replace("nve","newClass");
58 //parent,children and sibling elements
59 let parentElement=document.querySelector('.nura');
60 //all children of parent
61 console.log(parentElement.children);
62 //we can not run foreach method on html collection so
63 //first convert it into array
64 console.log(Array.from(parentElement.children));
65 Array.from(parentElement.children).forEach(function(element)
66 {
67 element.classList.add("coders");
68 })
69 let childElement=document.querySelector('.nve');
70
71 //console.log(childElement.parentElement);
72 //find next sibling of child
73 //console.log(childElement.nextElementSibling);
74 //console.log(childElement.previousElementSibling);
75 //event basics(click event)
76 let eventElement=document.getElementsByClassName('checkme');
77 console.log(eventElement);
78 //addeventlistner
79 // eventElement.addEventListener("click", function(){
80 // console.log("clicked me")
81 // })
82 //form events(submit form)
```



Hello javascript

JavaScript can change the style of an HTML element

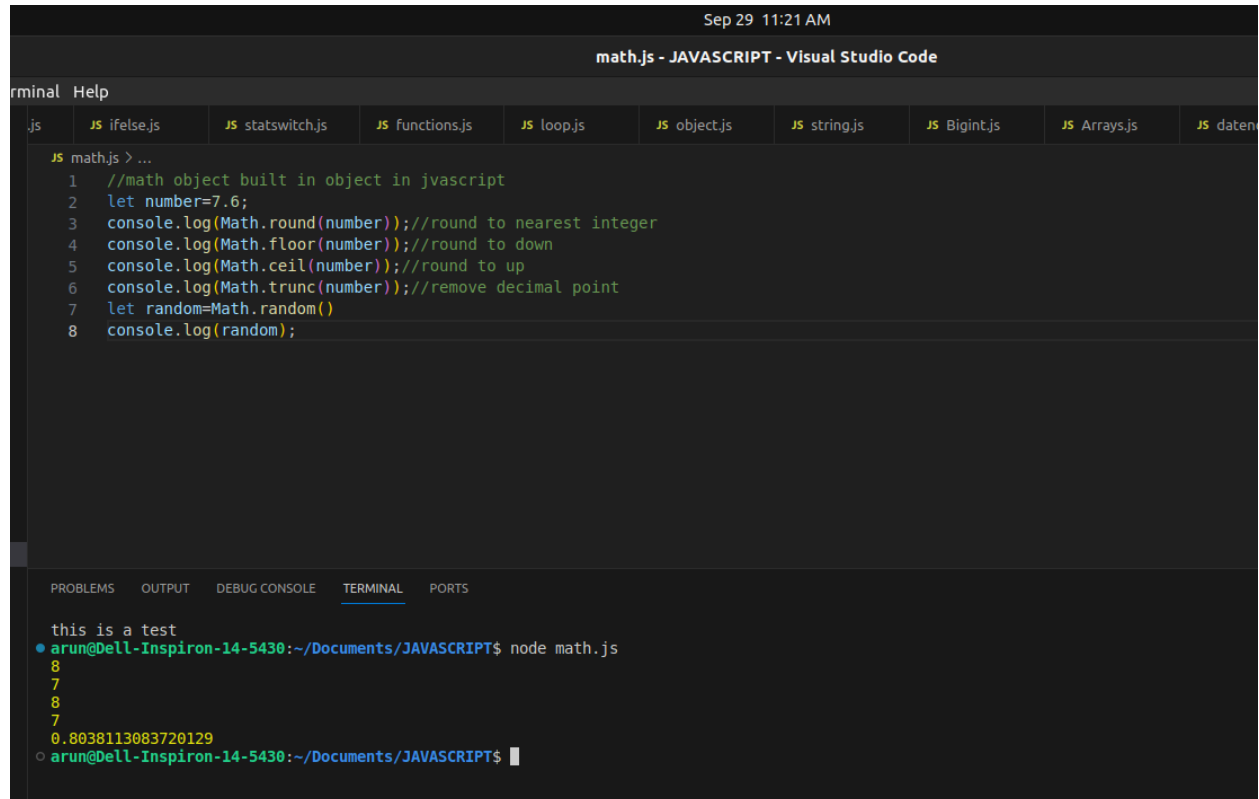
best family

my passion is coding

[html dom](#)

JavaScript Math Object

The JavaScript Math object allows you to perform mathematical tasks on Numbers.



The screenshot shows the Visual Studio Code editor with a file named `math.js` open. The code in the file uses the `Math` object to perform various mathematical operations on the number 7.6. The terminal at the bottom shows the output of running `node math.js`, displaying the results of these operations.

```
JS math.js > ...
1 //math object built in object in javascript
2 let number=7.6;
3 console.log(Math.round(number)); //round to nearest integer
4 console.log(Math.floor(number)); //round to down
5 console.log(Math.ceil(number)); //round to up
6 console.log(Math.trunc(number)); //remove decimal point
7 let random=Math.random()
8 console.log(random);
```

terminal Help

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
this is a test
● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node math.js
8
7
8
7
0.8038113083720129
○ arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```

Call and apply method-

```
JS function_call_apply.js > user > lastname
1 //with the help of call and apply we can manually change the value
2 //of this keyword
3 let details={
4     firstname: "arun",
5     lastname:"chauhan",
6     fullname:function(){
7
8         console.log(this.firstname + " "+this.lastname);
9     }
10 }
11 let person1={
12     firstname: "shrey",
13     lastname:"gupta",
14     age:24
15 }
16 }
17 let newdetails=details.fullname;
18 newdetails.call(person1);
19 //apply method
20 //there is a difference in call and apply in arguments
21 //in apply we have to pass arguments in an array
22 const person = {
23     fullName: function(city, country) {
24         console.log(this.firstName + " " + this.lastName + "," + city + "," + country);
25     }
26 }
27
28 const person2= {
29     firstName:"John",
30     lastName: "Doe"
31 }
32
33 person.fullName.apply(person2, ["Oslo", "Norway"]);
34 //bind method
35 function greet()
36 {
37     console.log(this.firstname + " "+this.lastname);
38 }
39 let user={
40     firstname: "arun",
41     lastname: "chauhan"
42 }
43 let greets= greet.bind(user);
44 greets()
```

```
15 }
16 }
17 let newdetails=details.fullname;
18 newdetails.call(person1);
19 //apply method
20 //there is a difference in call and apply in arguments
21 //in apply we have to pass arguments in an array
22 const person = {
23   fullName: function(city, country) {
24     console.log(this.firstName + " " + this.lastName + "," + city + "," + country);
25   }
26 }
27
28 const person2= {
29   firstName:"John",
30   lastName: "Doe"
31 }
32
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT\$ node function_call_apply.js
shrey gupta
John Doe,Oslo,Norway
arun chauhan
○ arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT\$

Foreach method

Sep 29 11:25 AM

foreach.js - JAVASCRIPT - Visual Studio Code

terminal Help

JS functions.js JS loop.js JS object.js JS string.js JS Bigint.js JS Arrays.js JS datendtime.js {} data.json JS dom.js

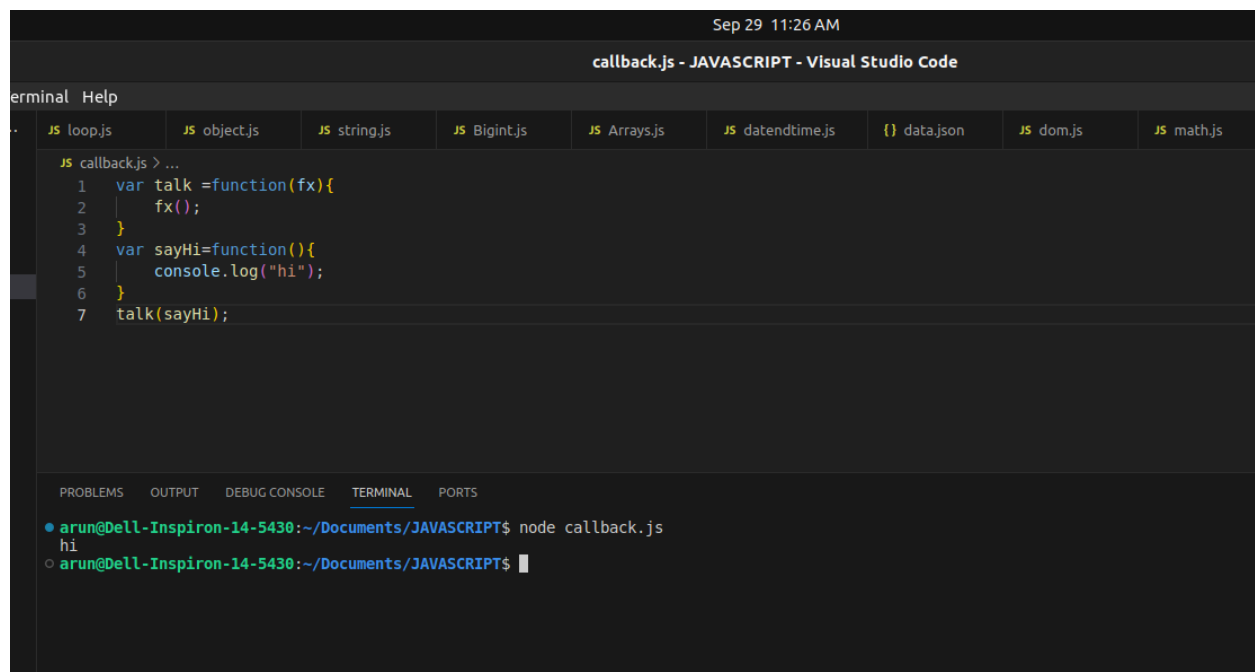
```
JS foreach.js > myfunction
1 let numbers=[45,5,9,67,987];
2 let txt="";
3 numbers.forEach(myfunction);
4 function myfunction(x)
5 {
6   txt=x;
7   console.log(txt);
8 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT\$ node foreach.js
45
5
9
67
987
○ arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT\$

JAVASCRIPT CALLBACK FUNCTION-

In JavaScript, a callback function is a function that is passed as an argument to another function and is intended to be executed at a later time or after the completion of an asynchronous operation. Callback functions are a fundamental concept in JavaScript, especially when dealing with asynchronous code, event handling, and functional programming.



The screenshot shows the Visual Studio Code editor with a file named 'callback.js' open. The code in the file is as follows:

```
1 var talk =function(fx){
2   fx();
3 }
4 var sayHi=function(){
5   console.log("hi");
6 }
7 talk(sayHi);
```

Below the editor, the terminal window shows the command 'node callback.js' being executed, which results in the output 'hi'.

Asynchronous and and AJAX

Async/Await:

"async" and "await" are keywords used in JavaScript to simplify working with asynchronous code. "async" is used to define asynchronous functions, while "await" is used inside an asynchronous function to pause its execution until a Promise is resolved or rejected. This makes the code look more synchronous and easier to read.

AJAX (Asynchronous JavaScript and XML):

AJAX is a technique in JavaScript that allows web pages to make asynchronous requests to the server without reloading the entire page. It enables data exchange between the web browser and the server in the background, providing a more responsive and interactive user experience. AJAX requests can retrieve data from the server, send data to the server, or interact with web APIs.

```
JS asyncndazax.js > ...
1 //asynchronous javascript
2 //async code example
3
4 console.log(1);
5 console.log(2);
6 console.log(3);
7
8 //async code
9 setTimeout(()=>{
10     console.log("print after 5 sec");
11 },5000)//after 5 sec
12 console.log(3);
13 console.log(4);
14 //azax code//asynchronous example
15 //making http request(example)
16 let todos=(callback)=>{
17     let request= new XMLHttpRequest();
18     console.log(request);
19     request.addEventListener('readystatechange',()=>{
20         if(request.readystate==4)
21         {
22             console.log(undefiend,request.responseText);
23         }
24         else{
25             console.log("error comes",undefiend);
26         }
27     })
28     //set up to request
29     request.open("Get", "https://jsonplaceholder.typicode.com/todos")
30
31
32     //sending the request
33     request.send()}
34
35 console.log(6);
36 todos((error,data)=>
37 {
38     if(error)
39     {
40         console.log(error);
41     }
42     else{
43         console.log(data);
44     }
45 })
46 console.log(7);
47 console.log(8);
```

```

19 request.addEventListener("readystatechange",()=>{
20     if(request.readyState==4)
21     {
22         console.log(undefiend,request.responseText);
23     }
24     else{
25         console.log("error comes",undefiend);
26     }
27 })
28 //set up to request
29 request.open("Get","https://jsonplaceholder.typicode.com/todos")
30
31
32 //sending the request
33 request.send()}
34
35 console.log(6).

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

this is a test
● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node asyncndazax.js
1
2
3
3
4
6
/home/arun/Documents/JAVASCRIPT/asyncndazax.js:17
let request= new XMLHttpRequest();
               ^
ReferenceError: XMLHttpRequest is not defined
    at todos (/home/arun/Documents/JAVASCRIPT/asyncndazax.js:17:14)
    at Object.<anonymous> (/home/arun/Documents/JAVASCRIPT/asyncndazax.js:36:1)
    at Module._compile (internal/modules/cjs/loader.js:999:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1027:10)
    at Module.load (internal/modules/cjs/loader.js:863:32)
    at Function.Module._load (internal/modules/cjs/loader.js:708:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:60:12)
    at internal/main/run_main_module.js:17:47
● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ 

```

CALLBACK HELL-Callback hell, also known as "Pyramid of Doom," is a term used to describe a situation in JavaScript code where multiple nested callback functions make the code structure difficult to read and maintain.

```
Sep 29 11:32 AM
azaxndjson.js - JAVASCRIPT - Visual Studio Code

final Help
ng.js JS Bigint.js JS Arrays.js JS datendtime.js {} data.json JS dom.js JS math.js JS function_call_apply.js JS foreach.js

JS azaxndjson.js > ...
1 //callbackhell
2 let todos=(resource,callback)=>{
3   let request= new XMLHttpRequest();
4   console.log(request);
5   request.addEventListener('readystatechange',()=>{
6     if(request.readyState==4)
7     {
8       callback(undefined,request.responseText);
9     }
10    else{
11      callback("error comes",undefined);
12    }
13  })
14  //set up to request
15  request.open("Get",resource)
16
17
18  //sending the request
19  request.send()}
20
21 console.log(6);
22 todos("data.json", (error,data)=>
23 {
24   if(error)
25   {
26     console.log(error);
27   }
28   else{
29     console.log(data);
30   }
31 }
32 )
33
34 console.log(8);
```

Promises

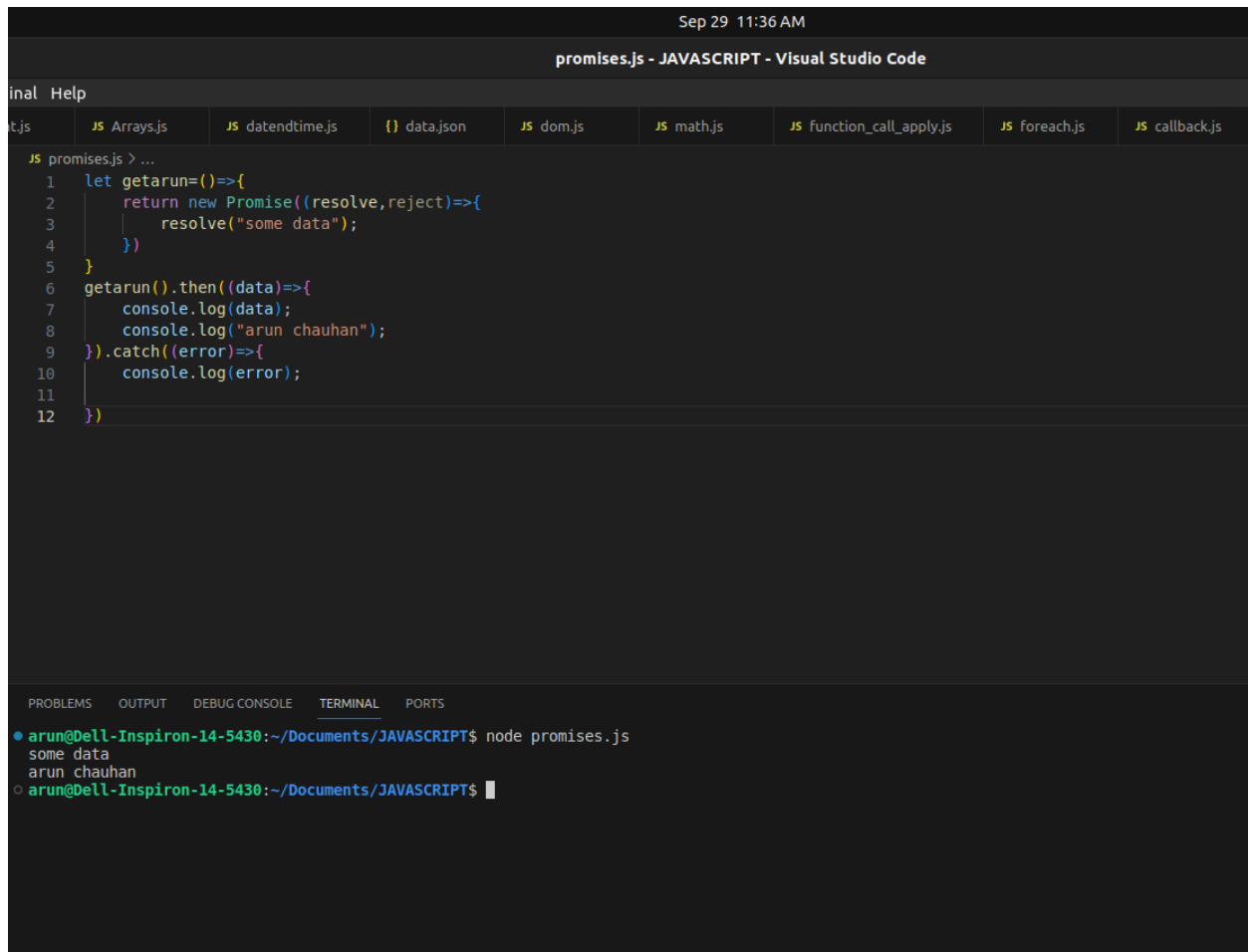
Promises help avoid the callback hell, a situation where nested callbacks make the code difficult to read and maintain.

A Promise can be in one of three states:

1. **Pending:** The initial state of the Promise, representing an ongoing asynchronous operation.
2. **Fulfilled/Resolved:** The Promise has successfully completed, and

the associated value (result) is available. This is when the `resolve()` function is called.

3. Rejected: The Promise encountered an error during execution. This is when the `reject()` function is called.



The screenshot shows the Visual Studio Code editor with a file named `promises.js` open. The code defines a function `getarun` that returns a Promise. The Promise is resolved with the string "some data". The `then` method is used to log the data, and the `catch` method is used to log any errors. The terminal output shows the command `node promises.js` being executed, resulting in the output "some data" and "arun chauhan".

```
Sep 29 11:36 AM
promises.js - JAVASCRIPT - Visual Studio Code
File Explorer Search
JS promises.js > ...
1 let getarun={()=>{
2   return new Promise((resolve,reject)=>{
3     resolve("some data");
4   })
5 }
6 getarun().then((data)=>{
7   console.log(data);
8   console.log("arun chauhan");
9 }).catch((error)=>{
10  console.log(error);
11 }
12 })

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
● arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$ node promises.js
some data
arun chauhan
○ arun@Dell-Inspiron-14-5430:~/Documents/JAVASCRIPT$
```