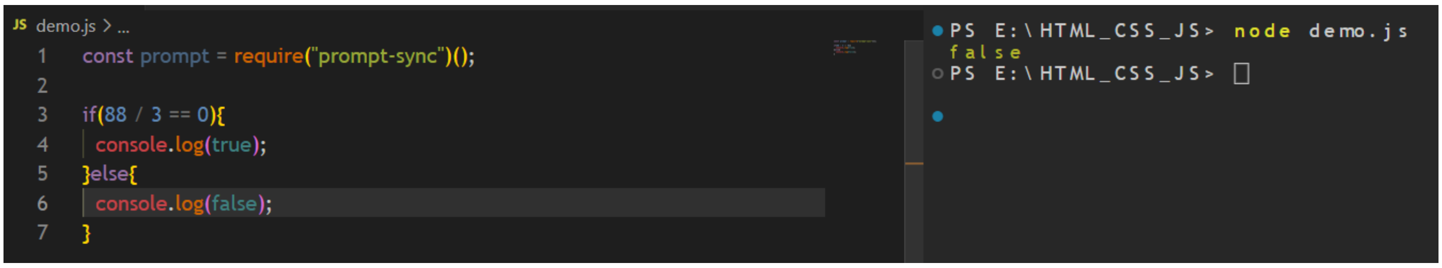
Decision Control In JavaScript:

In this concept system decide the output by checking the condition that is provided by the user. As per the user's given condition the system will decide which output he will displayed to the user. In decision control there are some following techniques are there. If, If Else, Nested If, Nested If Else, Switch statement.

If-else Statement:

In if-else condition the user check whether the given operation is true or not. In the if statement inside the bracket it is mandatory that the operation's output must be a boolean value which is either true or false.

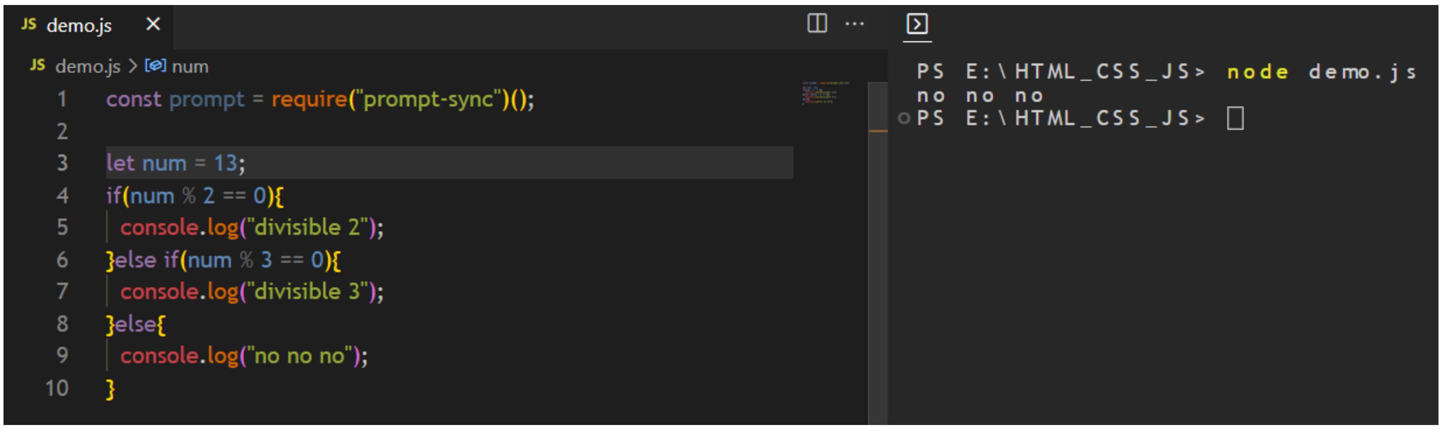


```
JS demo.js > ...
const prompt = require("prompt-sync")();

if(88 / 3 == 0){
  console.log(true);
}else{
  console.log(false);
}
```

```
PS E:\HTML_CSS_JS> node demo.js
false
PS E:\HTML_CSS_JS>
```

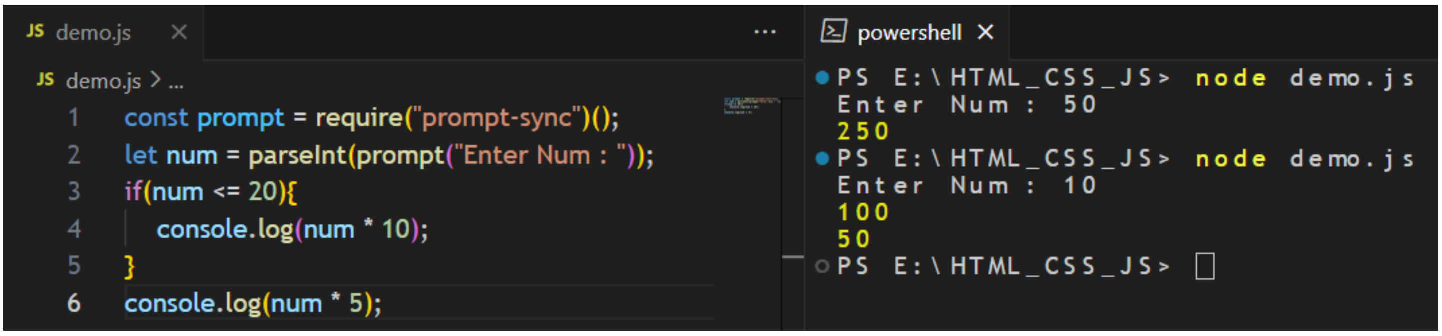
Else-If Statement:

else-if condition is used for nested conditional statement where user use more than one condition with variable. Else-if condition must be used after the if condition. Inside the bracket of else-if the operation's output must be boolean type either true or false.



```
JS demo.js > [@] num
const prompt = require("prompt-sync")();

let num = 13;
if(num % 2 == 0){
  console.log("divisible 2");
}else if(num % 3 == 0){
  console.log("divisible 3");
}else{
  console.log("no no no");
}
```

```
PS E:\HTML_CSS_JS> node demo.js
no no no
PS E:\HTML_CSS_JS>
```

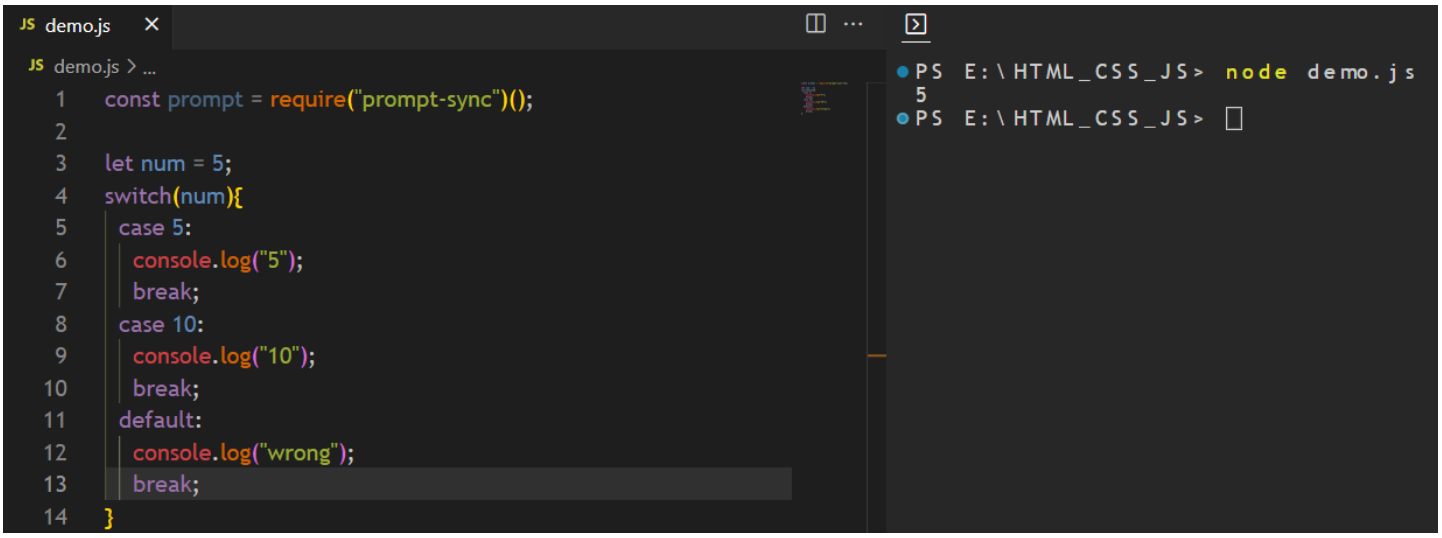


```
JS demo.js > ...
const prompt = require("prompt-sync")();
let num = parseInt(prompt("Enter Num : "));
if(num <= 20){
    console.log(num * 10);
}
console.log(num * 5);
```

```
PS E:\HTML_CSS_JS> node demo.js
Enter Num : 50
250
PS E:\HTML_CSS_JS> node demo.js
Enter Num : 10
100
50
PS E:\HTML_CSS_JS>
```

Switch Statement:

Switch statement is very useful statement nowadays in programming. During the nested if-else condition we need to wrote so many source code of line. But in switch statement using some simple keyword we can define all this things which is more readable and user-friendly.



demo.js

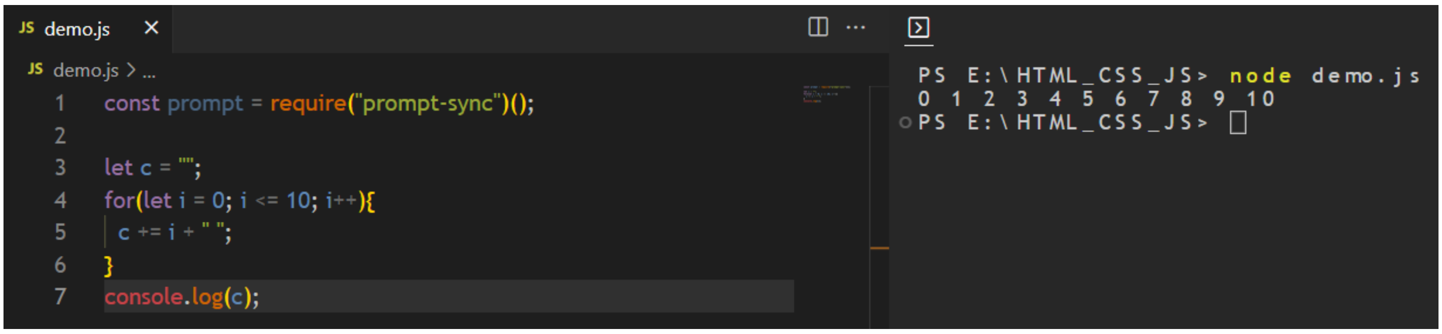
```javascript
const prompt = require("prompt-sync")();

let num = 5;
switch(num){
  case 5:
    console.log("5");
    break;
  case 10:
    console.log("10");
    break;
  default:
    console.log("wrong");
    break;
}
```

```
PS E:\HTML_CSS_JS> node demo.js
5
PS E:\HTML_CSS_JS>
```

Looping Concept:

Looping concept is used for to print more than one output values by initializing the variable, performing a condition and increase or decrease of the variable. In looping concept we use the following concepts For Loop, While Loop, Do-While Loop.

For Loop:

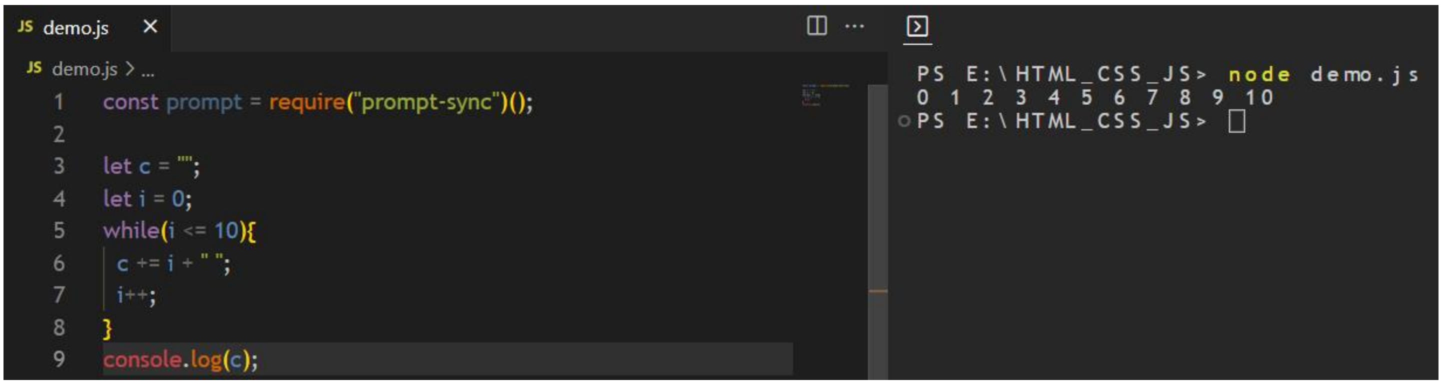


demo.js

```javascript
const prompt = require("prompt-sync")();

let c = "";
for(let i = 0; i <= 10; i++){
  c += i + " ";
}
console.log(c);
```

```
PS E:\HTML_CSS_JS> node demo.js
0 1 2 3 4 5 6 7 8 9 10
PS E:\HTML_CSS_JS>
```

While Loop:

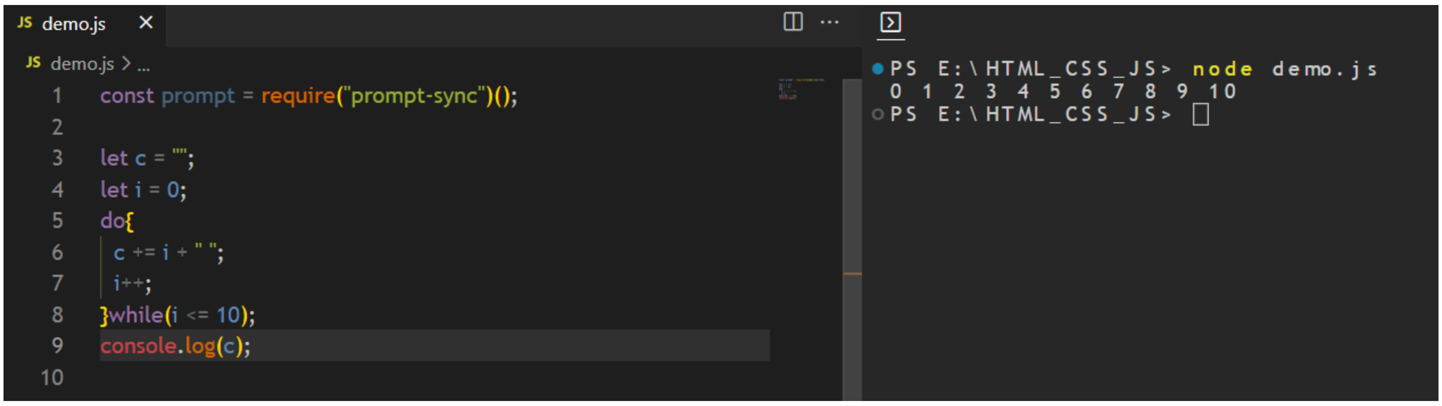


demo.js

```javascript
const prompt = require("prompt-sync")();

let c = "";
let i = 0;
while(i <= 10){
  c += i + " ";
  i++;
}
console.log(c);
```

```
PS E:\HTML_CSS_JS> node demo.js
0 1 2 3 4 5 6 7 8 9 10
PS E:\HTML_CSS_JS>
```

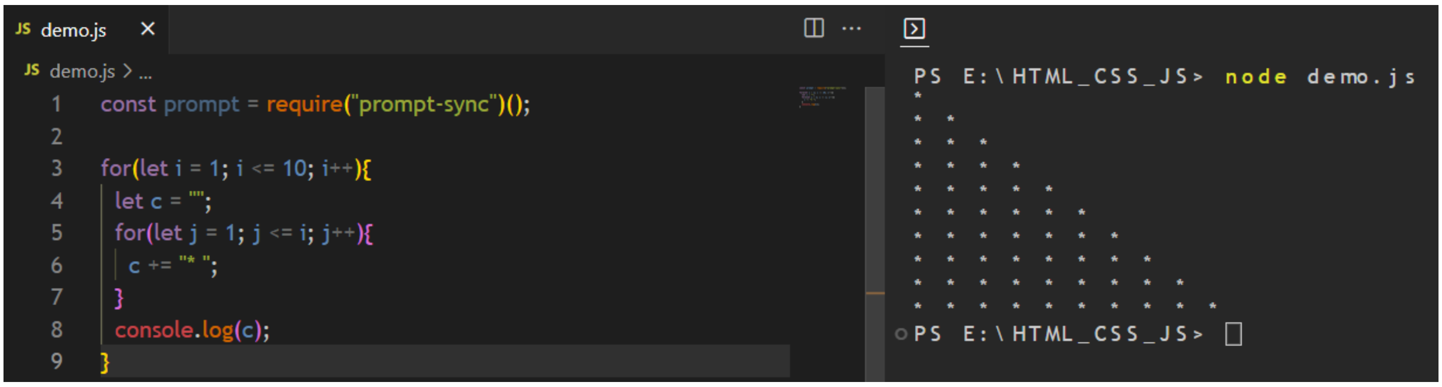
Do-While Loop:



```javascript
const prompt = require("prompt-sync")();

let c = "";
let i = 0;
do{
  c += i + " ";
  i++;
}while(i <= 10);
console.log(c);

```

```
PS E:\HTML_CSS_JS> node demo.js
0 1 2 3 4 5 6 7 8 9 10
PS E:\HTML_CSS_JS> 
```

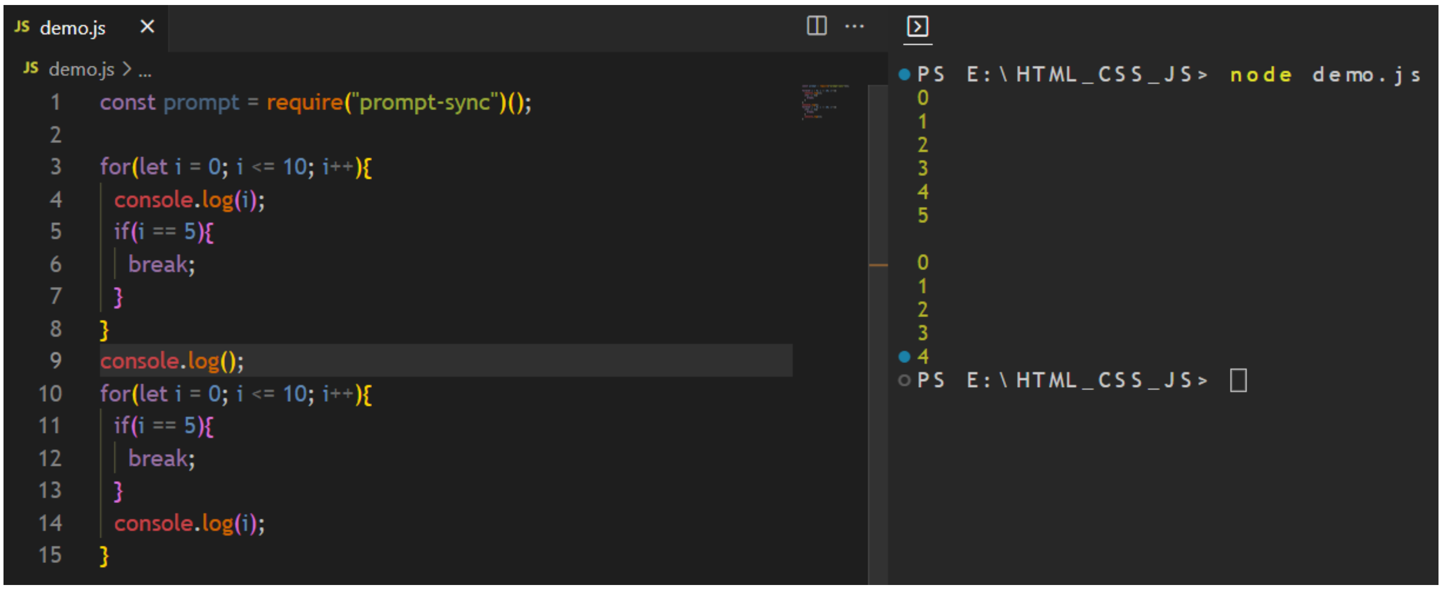
Nested Loop:



```javascript
const prompt = require("prompt-sync")();

for(let i = 1; i <= 10; i++){
  let c = "";
  for(let j = 1; j <= i; j++){
    c += "* ";
  }
  console.log(c);
}
```

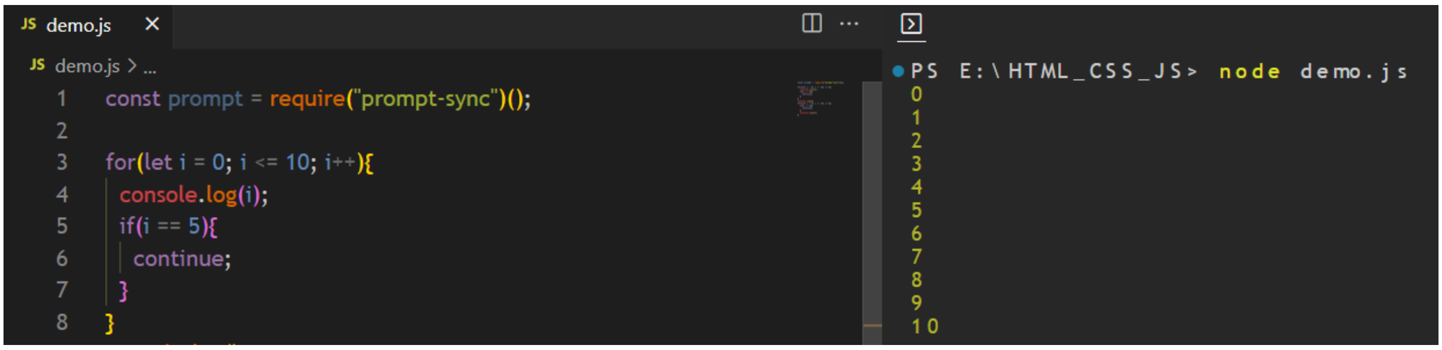
```
PS E:\HTML_CSS_JS> node demo.js
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
PS E:\HTML_CSS_JS> 
```

Break & Continue:



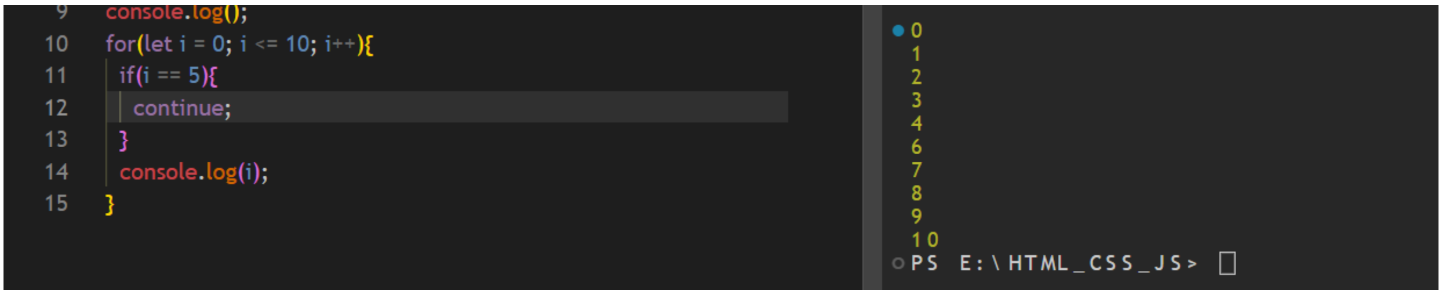
```javascript
const prompt = require("prompt-sync")();

for(let i = 0; i <= 10; i++){
  console.log(i);
  if(i == 5){
    break;
  }
}
console.log();
for(let i = 0; i <= 10; i++){
  if(i == 5){
    break;
  }
  console.log(i);
}
```

```
PS E:\HTML_CSS_JS> node demo.js
0
1
2
3
4
5

0
1
2
3
4
PS E:\HTML_CSS_JS> 
```



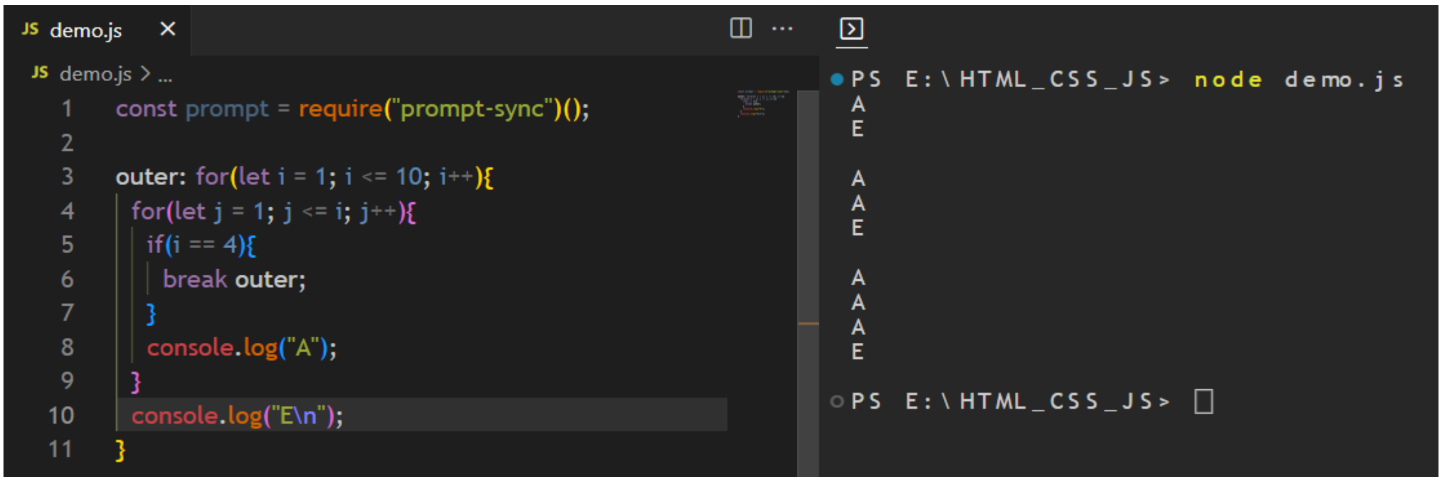
```javascript
const prompt = require("prompt-sync")();

for(let i = 0; i <= 10; i++){
  console.log(i);
  if(i == 5){
    continue;
  }
}
```

```
PS E:\HTML_CSS_JS> node demo.js
0
1
2
3
4
5
6
7
8
9
10
```



```
console.log();
for(let i = 0; i <= 10; i++){
  if(i == 5){
    continue;
  }
  console.log(i);
}
```

```
0
1
2
3
4
6
7
8
9
10
PS E:\HTML_CSS_JS> 
```

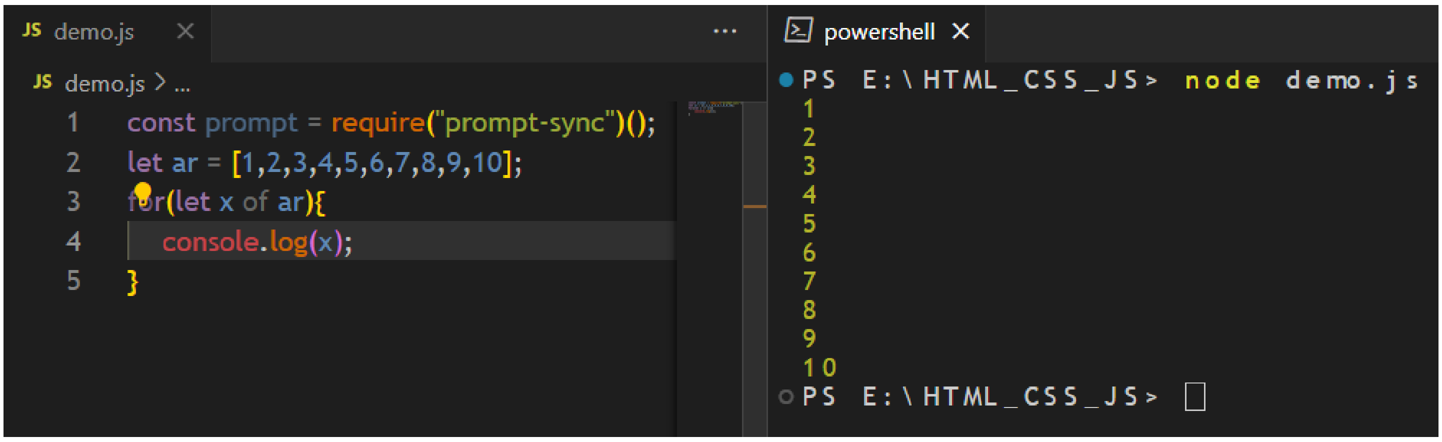


```
const prompt = require("prompt-sync")();

outer: for(let i = 1; i <= 10; i++){
  for(let j = 1; j <= i; j++){
    if(i == 4){
      break outer;
    }
    console.log("A");
  }
  console.log("E\n");
}
```

```
PS E:\HTML_CSS_JS> node demo.js
A
E

A
A
E

A
A
A
E

PS E:\HTML_CSS_JS> 
```

For Of Loop:

This statement loops through the values of an iterable object. It lets you loop over iterable data structures such as Arrays, Strings, Maps, NodeLists, and more.

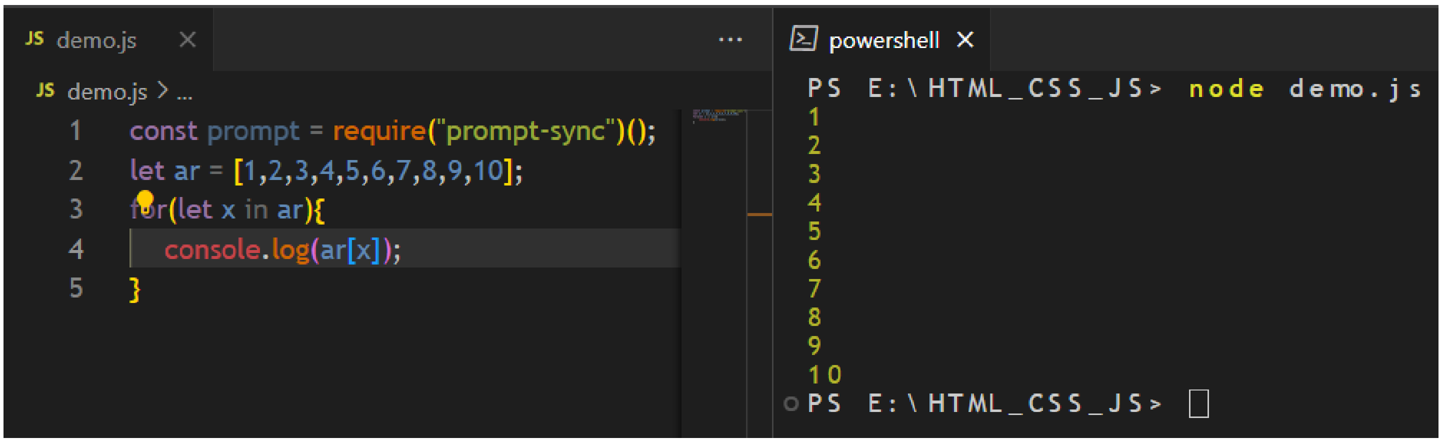


```
const prompt = require("prompt-sync")();
let ar = [1,2,3,4,5,6,7,8,9,10];
for(let x of ar){
  console.log(x);
}
```

```
PS E:\HTML_CSS_JS> node demo.js
1
2
3
4
5
6
7
8
9
10
PS E:\HTML_CSS_JS> 
```

For In Loop:

This statements combo iterates (loops) over the properties of an object. The code block inside the loop is executed once for each property.

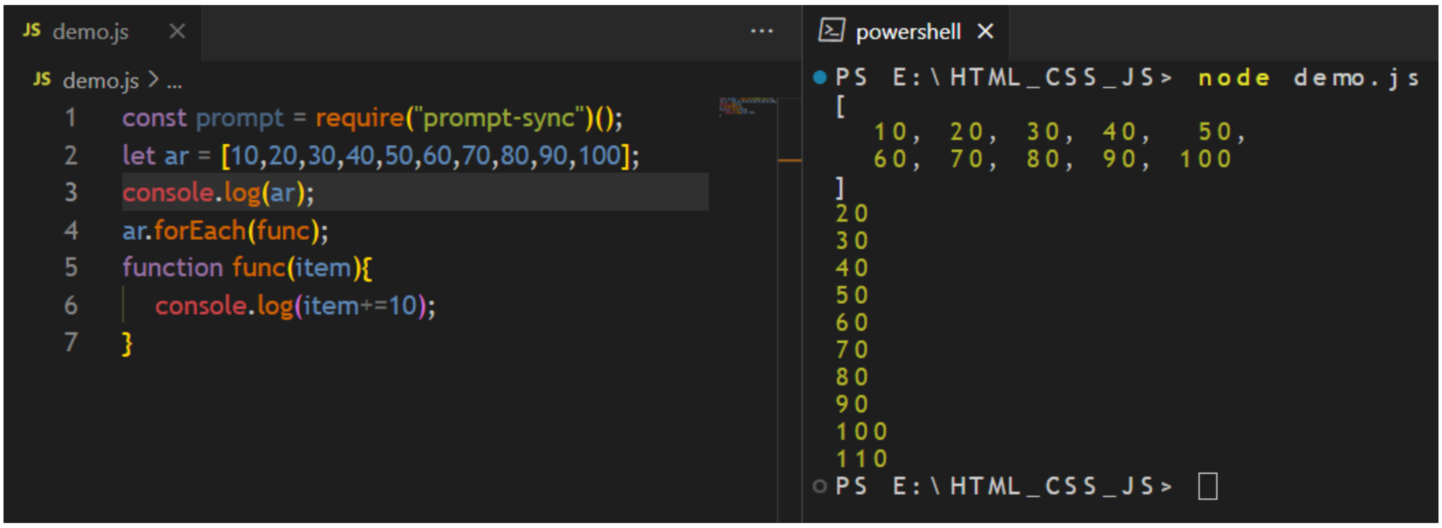


```
const prompt = require("prompt-sync")();
let ar = [1,2,3,4,5,6,7,8,9,10];
for(let x in ar){
  console.log(ar[x]);
}
```

```
PS E:\HTML_CSS_JS> node demo.js
1
2
3
4
5
6
7
8
9
10
PS E:\HTML_CSS_JS> 
```

For Each() Loop:

The forEach() method calls a function for each element in an array. The forEach() method is not executed for empty elements.

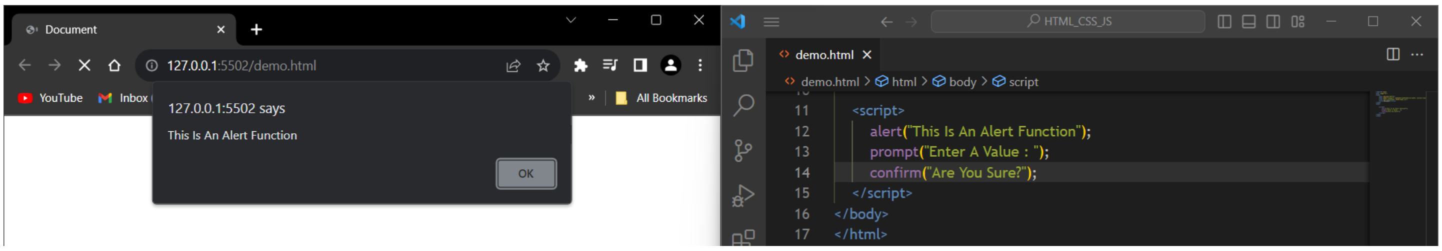


```javascript
const prompt = require("prompt-sync")();
let ar = [10,20,30,40,50,60,70,80,90,100];
console.log(ar);
ar.forEach(func);
function func(item){
    console.log(item+=10);
}
```

```
PS E:\HTML_CSS_JS> node demo.js
[
  10, 20, 30, 40,  50,
  60, 70, 80, 90, 100
]
20
30
40
50
60
70
80
90
100
110
PS E:\HTML_CSS_JS>
```

Alert, Prompt, Confirm:

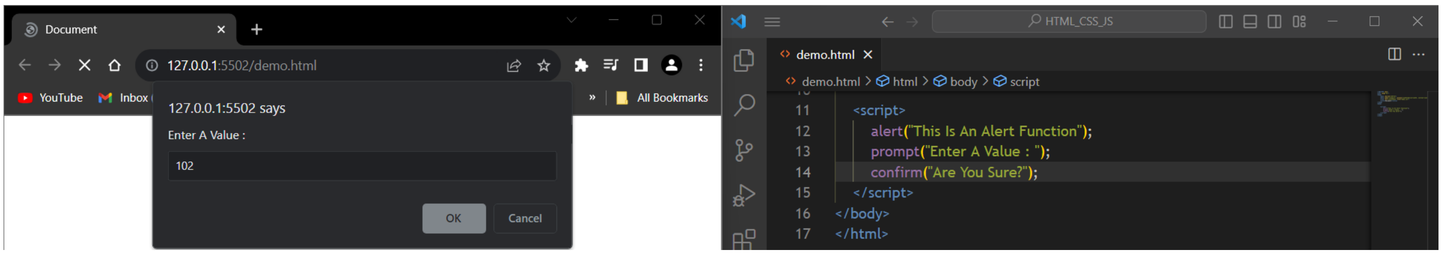
ALERT(): shows a message.



```
127.0.0.1:5502 says
This Is An Alert Function
```

```html
    <script>
      alert("This Is An Alert Function");
      prompt("Enter A Value : ");
      confirm("Are You Sure?");
    </script>
</body>
</html>
```

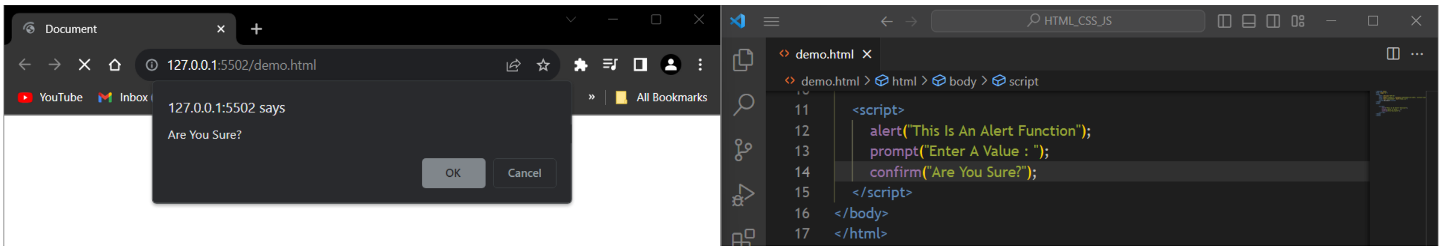
PRMOPT(): shows a message, input test. it returns the test on "ok" or if "cancel" button or esc button is clicked then it gives null.



```
127.0.0.1:5502 says
Enter A Value :
102
```

```html
    <script>
      alert("This Is An Alert Function");
      prompt("Enter A Value : ");
      confirm("Are You Sure?");
    </script>
</body>
</html>
```

CONFIRM(); shows a message confirm with "ok" or "cancel" it returns true for ok and esc is clicked then it return null.

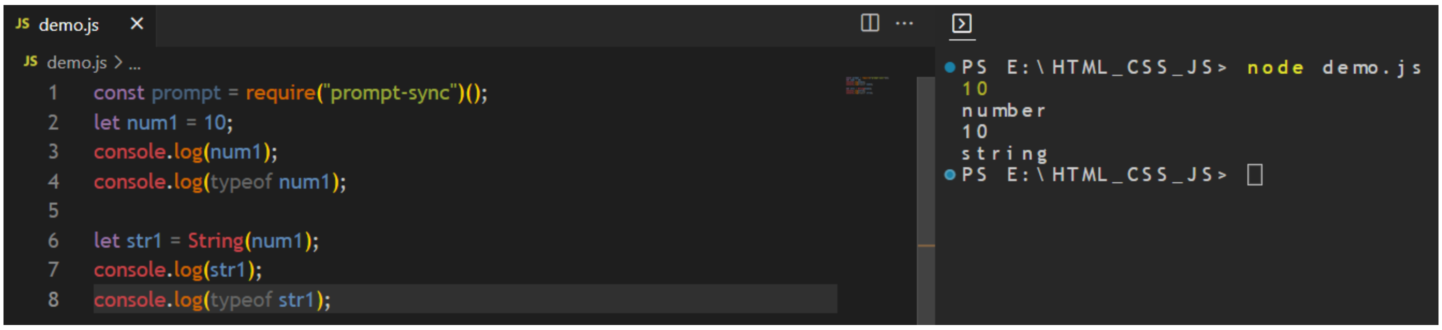


```
127.0.0.1:5502 says
Are You Sure?
```

```html
    <script>
      alert("This Is An Alert Function");
      prompt("Enter A Value : ");
      confirm("Are You Sure?");
    </script>
</body>
</html>
```

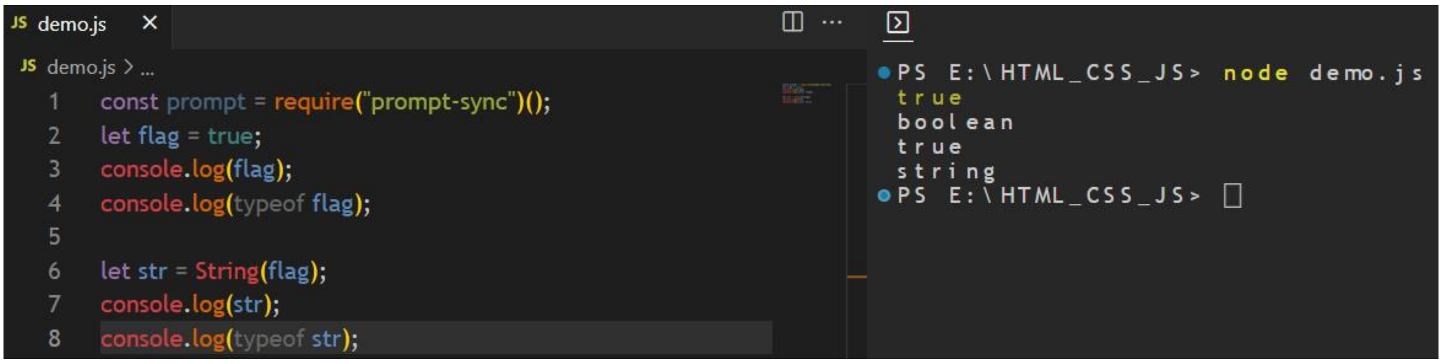
all these pause script execution and don't allow the visitor to interact with the rest of the page until the window has been disabled.

Type Conversion:



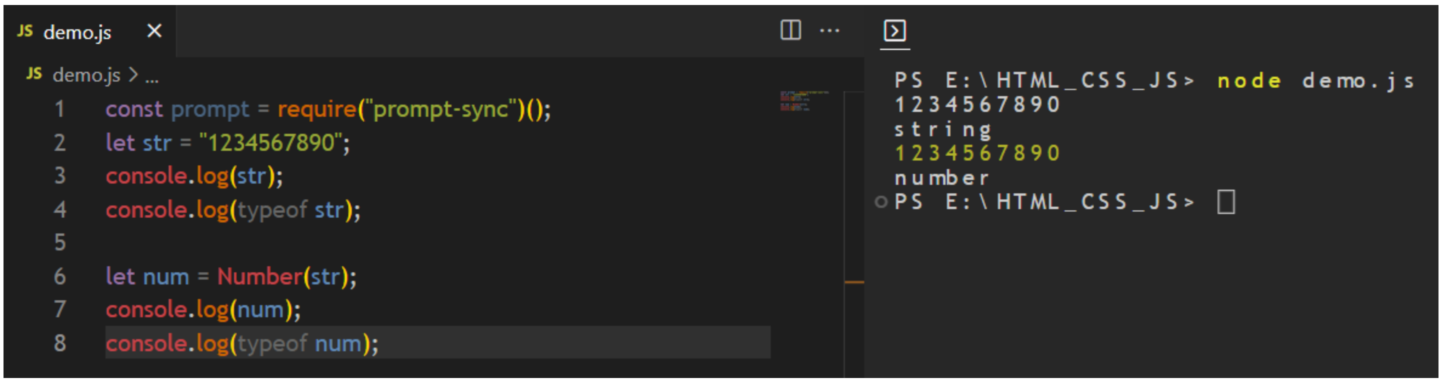
```
const prompt = require("prompt-sync")();
let num1 = 10;
console.log(num1);
console.log(typeof num1);

let str1 = String(num1);
console.log(str1);
console.log(typeof str1);
```

```
PS E:\HTML_CSS_JS> node demo.js
10
number
10
string
PS E:\HTML_CSS_JS>
```



```
const prompt = require("prompt-sync")();
let flag = true;
console.log(flag);
console.log(typeof flag);

let str = String(flag);
console.log(str);
console.log(typeof str);
```

```
PS E:\HTML_CSS_JS> node demo.js
true
boolean
true
string
PS E:\HTML_CSS_JS>
```

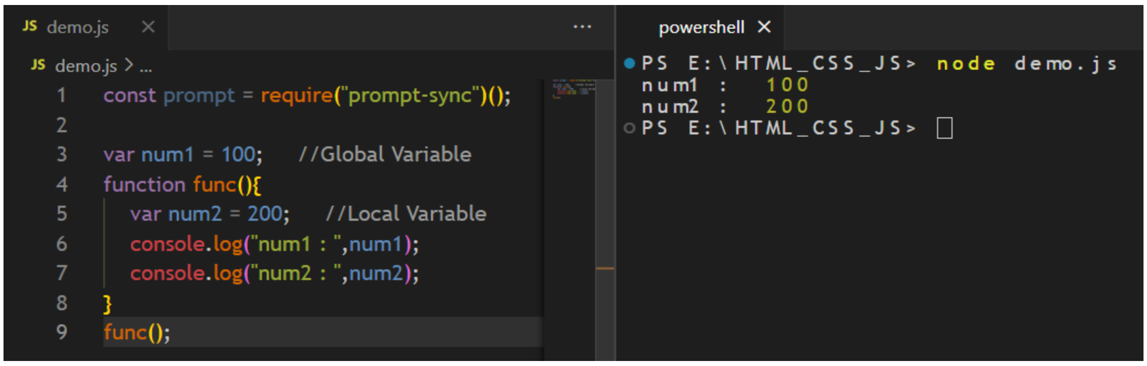


```
const prompt = require("prompt-sync")();
let str = "1234567890";
console.log(str);
console.log(typeof str);

let num = Number(str);
console.log(num);
console.log(typeof num);
```

```
PS E:\HTML_CSS_JS> node demo.js
1234567890
string
1234567890
number
PS E:\HTML_CSS_JS>
```

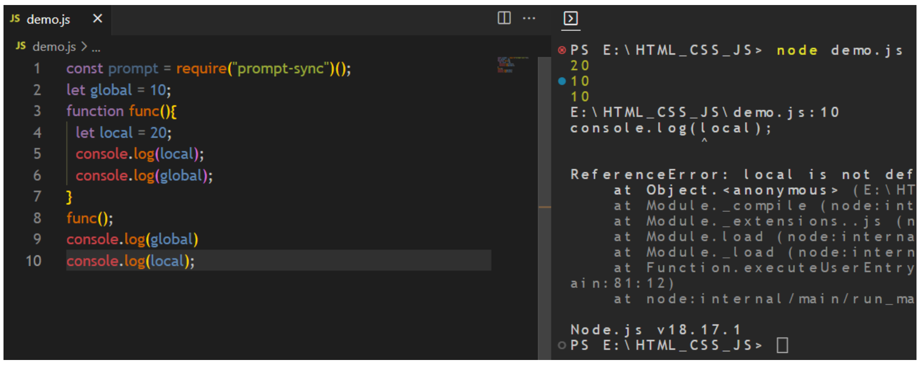
Function:

Local & Global Variable:



```
const prompt = require("prompt-sync")();

var num1 = 100;     //Global Variable
function func(){
    var num2 = 200;     //Local Variable
    console.log("num1 : ",num1);
    console.log("num2 : ",num2);
}
func();
```

```
PS E:\HTML_CSS_JS> node demo.js
num1 :  100
num2 :  200
PS E:\HTML_CSS_JS>
```

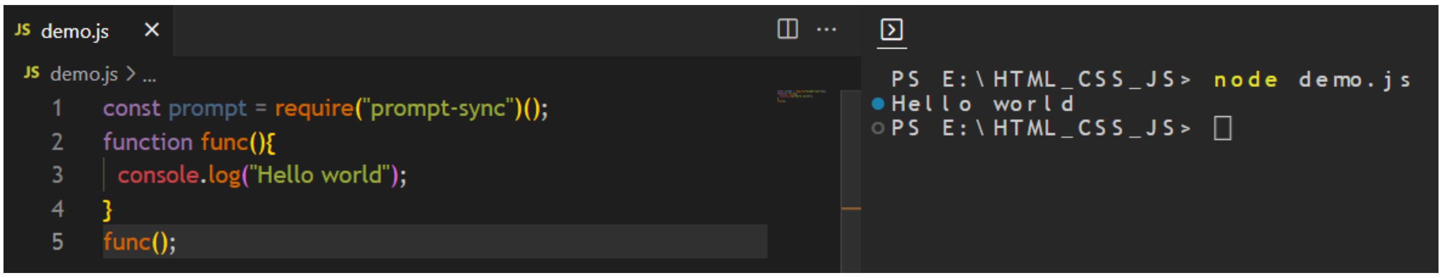


```
const prompt = require("prompt-sync")();
let global = 10;
function func(){
  let local = 20;
  console.log(local);
  console.log(global);
}
func();
console.log(global)
console.log(local);
```

```
PS E:\HTML_CSS_JS> node demo.js
20
10
10
E:\HTML_CSS_JS\demo.js:10
console.log(local);
            ^

ReferenceError: local is not def
    at Object.<anonymous> (E:\HT
    at Module._compile (node:int
    at Module._extensions..js (n
    at Module.load (node:interna
    at Module._load (node:intern
    at Function.executeUserEntry
ain:81:12)
    at node:internal/main/run_ma

Node.js v18.17.1
PS E:\HTML_CSS_JS>
```

Different Types Of Function Variants:

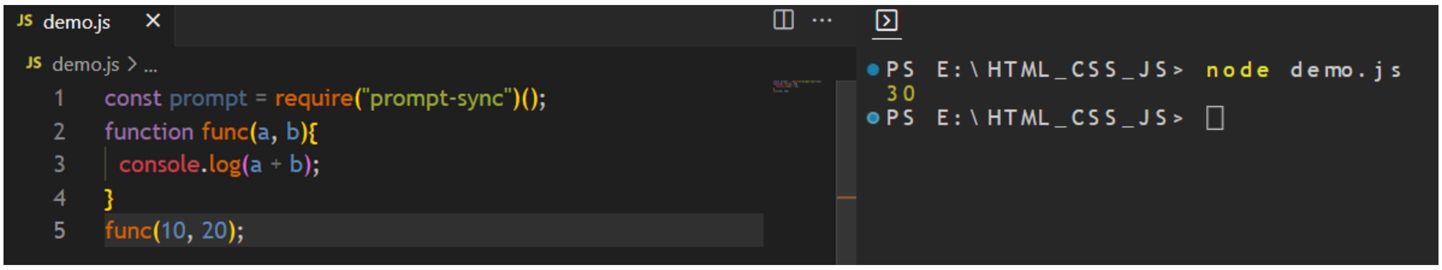
Take Nothing Return Nothing:



```javascript
const prompt = require("prompt-sync")();
function func(){
  console.log("Hello world");
}
func();
```

```
PS E:\HTML_CSS_JS> node demo.js
Hello world
PS E:\HTML_CSS_JS>
```

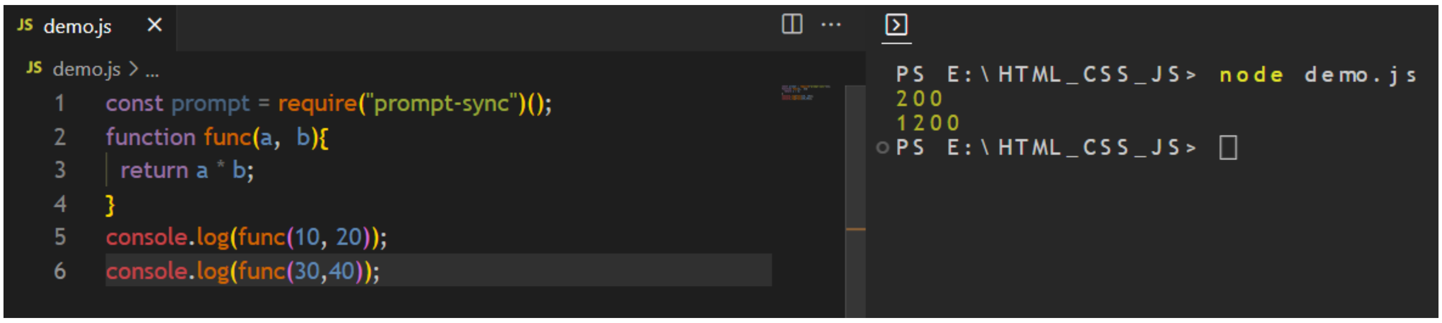
Take Something Return Nothing:



```javascript
const prompt = require("prompt-sync")();
function func(a, b){
  console.log(a + b);
}
func(10, 20);
```

```
PS E:\HTML_CSS_JS> node demo.js
30
PS E:\HTML_CSS_JS>
```

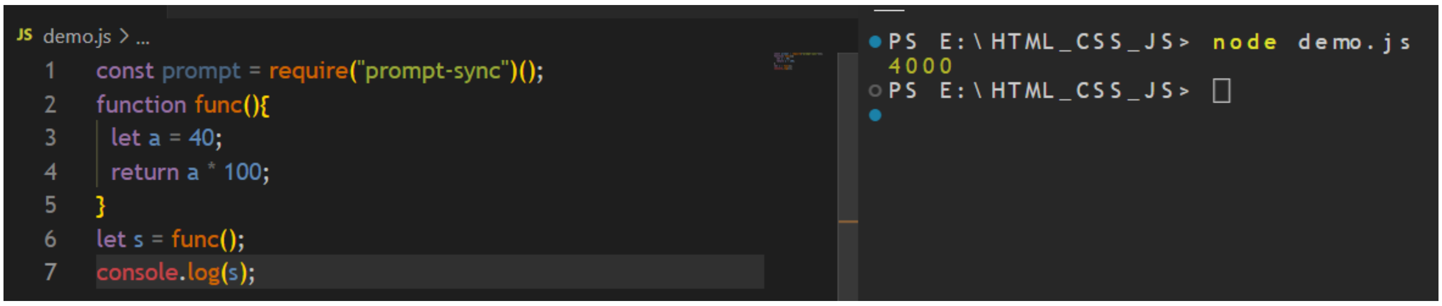
Take Something Return Something:



```javascript
const prompt = require("prompt-sync")();
function func(a,  b){
  return a * b;
}
console.log(func(10, 20));
console.log(func(30,40));
```

```
PS E:\HTML_CSS_JS> node demo.js
200
1200
PS E:\HTML_CSS_JS>
```

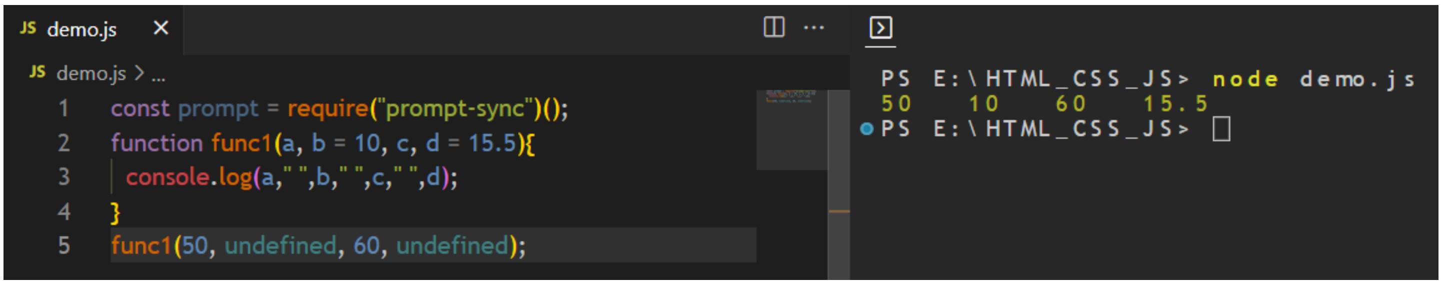
Take Nothing Return Something:



```javascript
const prompt = require("prompt-sync")();
function func(){
  let a = 40;
  return a * 100;
}
let s = func();
console.log(s);
```

```
PS E:\HTML_CSS_JS> node demo.js
4000
PS E:\HTML_CSS_JS>
```

Function Variable:



```javascript
const prompt = require("prompt-sync")();
function func1(a, b = 10, c, d = 15.5){
  console.log(a," ",b," ",c," ",d);
}
func1(50, undefined, 60, undefined);
```

```
PS E:\HTML_CSS_JS> node demo.js
50   10   60   15.5
PS E:\HTML_CSS_JS>
```

Fat Arrow Function:

An arrow function in JavaScript is a concise way to write anonymous functions, also known as lambda functions or fat arrow functions. They were introduced in ECMAScript 6 (ES6) and provide a more concise syntax compared to traditional function expressions. Arrow functions are often used for defining small, single-expression functions.

Parameters: These are the input values that the function takes. You can have zero or more parameters. If there are no parameters, you need to include empty parentheses.

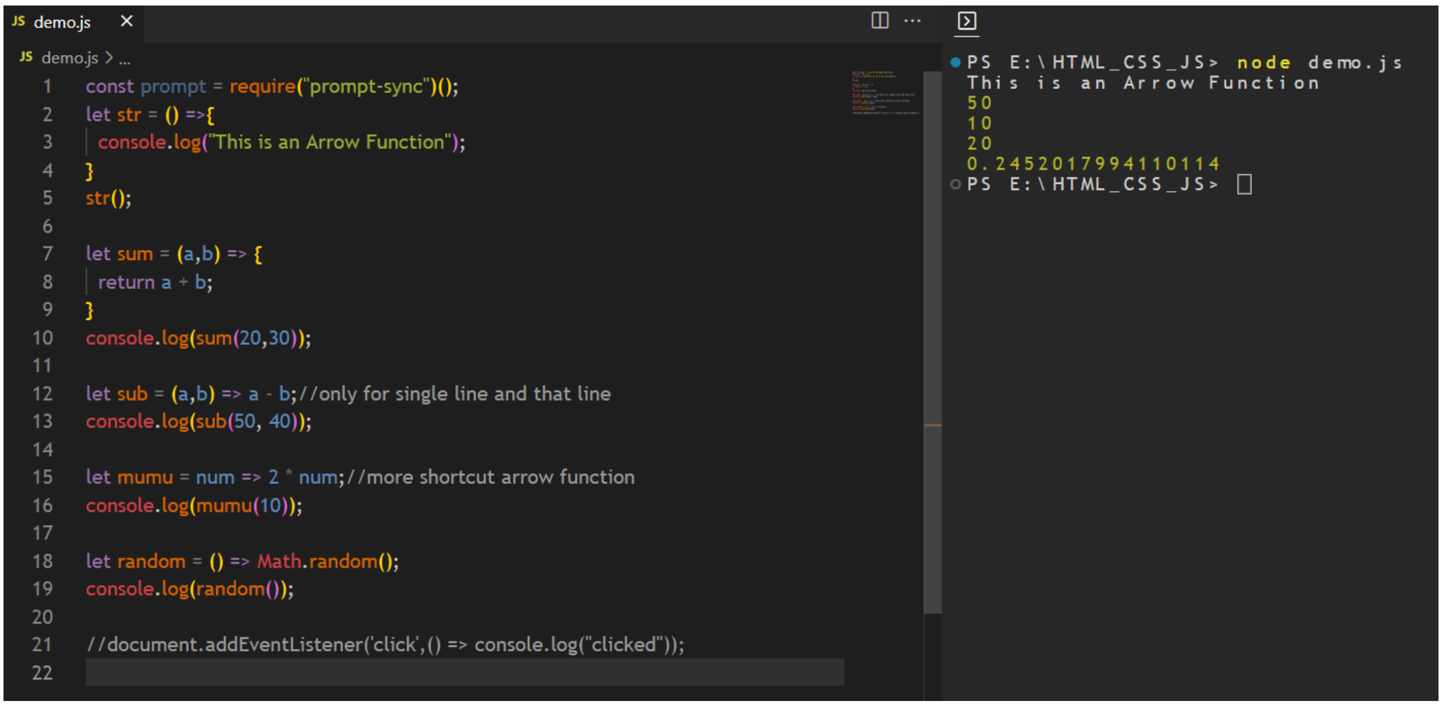
Arrow(=>): The arrow => separates the parameter list from the function body.

Expression: This is the code that gets executed when the arrow function is called. The result of this expression is implicitly returned from the function, which means you don't need to use the return keyword for single expressions.

Characteristics:

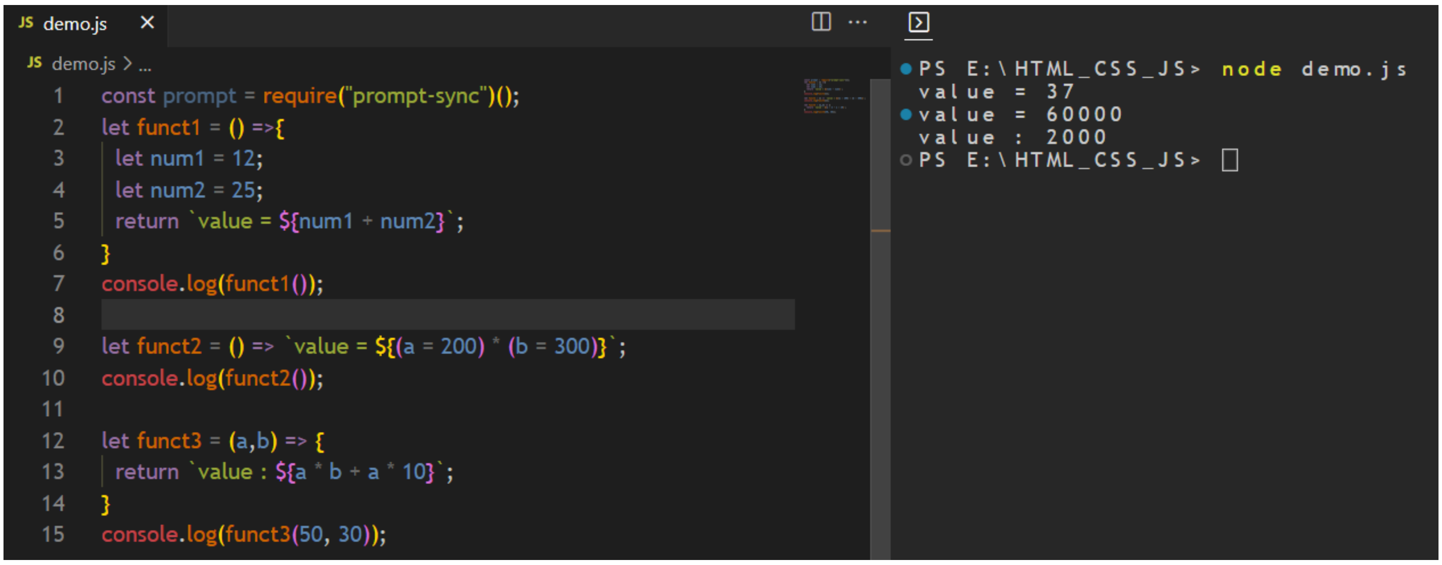
They do not have their own this binding. Instead, they inherit the this value from the enclosing lexical context (usually the surrounding function or the global context). They cannot be used as constructor functions to create objects. They cannot have their own arguments object. They inherit the arguments from the containing function.

Arrow functions are especially useful for short and simple functions and are commonly used in modern JavaScript code for their concise syntax and the way they handle the this keyword, making it easier to manage context in certain situations.



```javascript
const prompt = require("prompt-sync")();
let str = () =>{
    console.log("This is an Arrow Function");
}
str();

let sum = (a,b) => {
    return a + b;
}
console.log(sum(20,30));

let sub = (a,b) => a - b;//only for single line and that line
console.log(sub(50, 40));

let mumu = num => 2 * num;//more shortcut arrow function
console.log(mumu(10));

let random = () => Math.random();
console.log(random());

//document.addEventListener('click',() => console.log("clicked"));

```

```
PS E:\HTML_CSS_JS> node demo.js
This is an Arrow Function
50
10
20
0.2452017994110114
PS E:\HTML_CSS_JS>
```

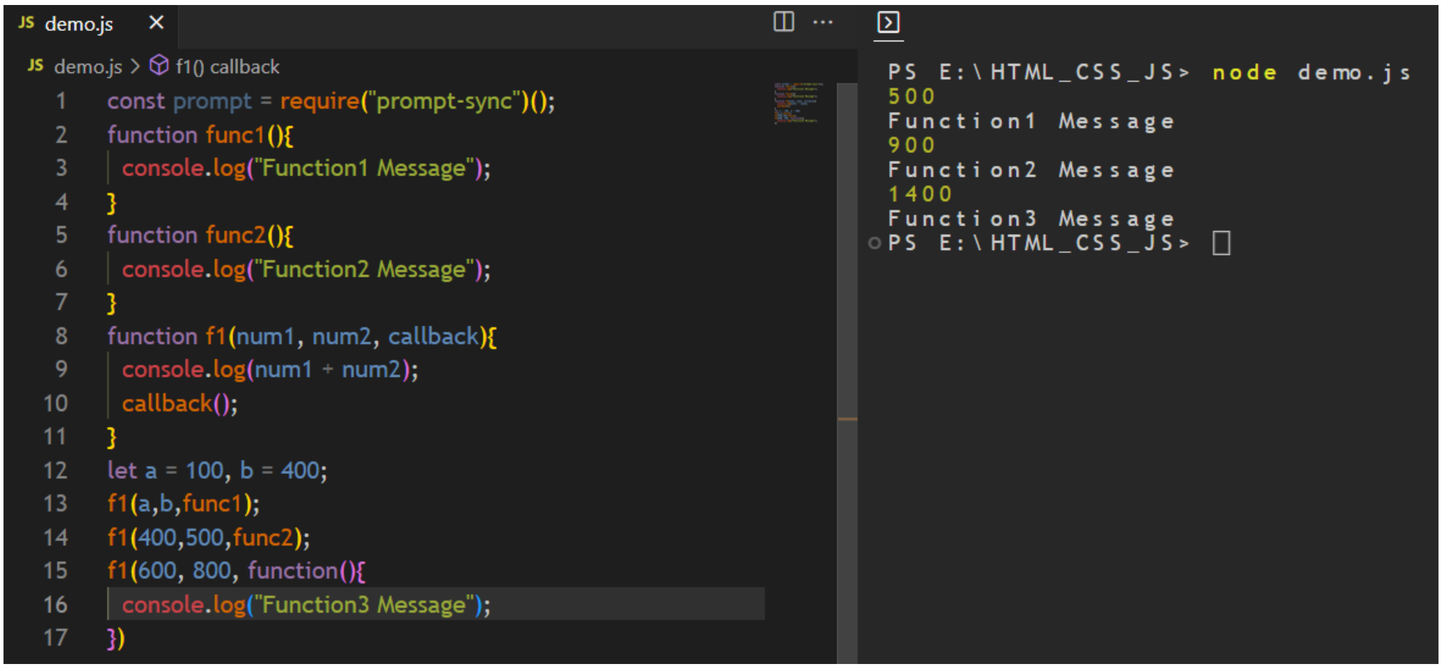


```
const prompt = require("prompt-sync")();
let funct1 = () =>{
  let num1 = 12;
  let num2 = 25;
  return `value = ${num1 + num2}`;
}
console.log(funct1());

let funct2 = () => `value = ${(a = 200) * (b = 300)}`;
console.log(funct2());

let funct3 = (a,b) => {
  return `value : ${a * b + a * 10}`;
}
console.log(funct3(50, 30));
```

```
PS E:\HTML_CSS_JS> node demo.js
value = 37
value = 60000
value : 2000
PS E:\HTML_CSS_JS>
```

Call Back Function:

A callback is a function passed as an argument to another function. This technique allows a function to call another function. A callback function can run after another function has finished.



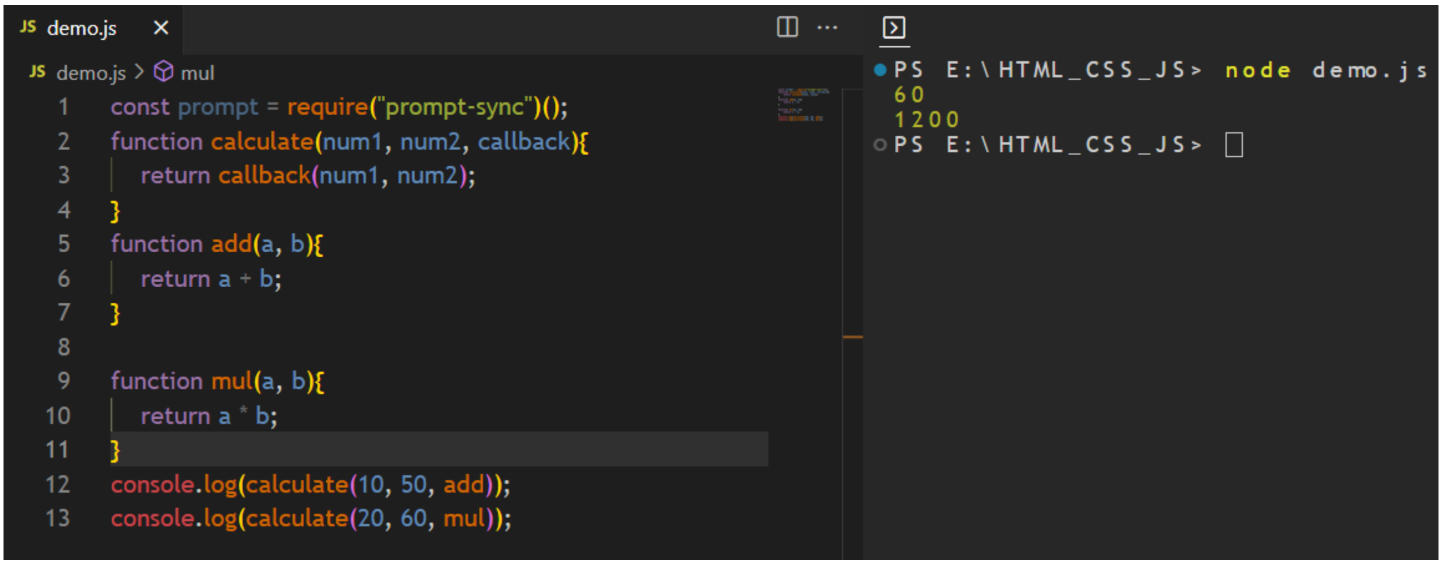
```
const prompt = require("prompt-sync")();
function func1(){
  console.log("Function1 Message");
}
function func2(){
  console.log("Function2 Message");
}
function f1(num1, num2, callback){
  console.log(num1 + num2);
  callback();
}
let a = 100, b = 400;
f1(a,b,func1);
f1(400,500,func2);
f1(600, 800, function(){
  console.log("Function3 Message");
})
```

```
PS E:\HTML_CSS_JS> node demo.js
500
Function1 Message
900
Function2 Message
1400
Function3 Message
PS E:\HTML_CSS_JS>
```

A callback function is a concept in programming where a function is passed as an argument to another function and is executed at a later time or under certain conditions. Callback functions are used in various programming languages, but they are particularly prevalent in JavaScript due to its asynchronous nature.

Passing Function As Argument:

In many programming languages, functions are treated as first-class citizens, which means they can be assigned to variables, passed as arguments to other functions, and returned from functions just like any other data type. Callback functions take advantage of this feature by allowing you to pass a function as an argument to another function.

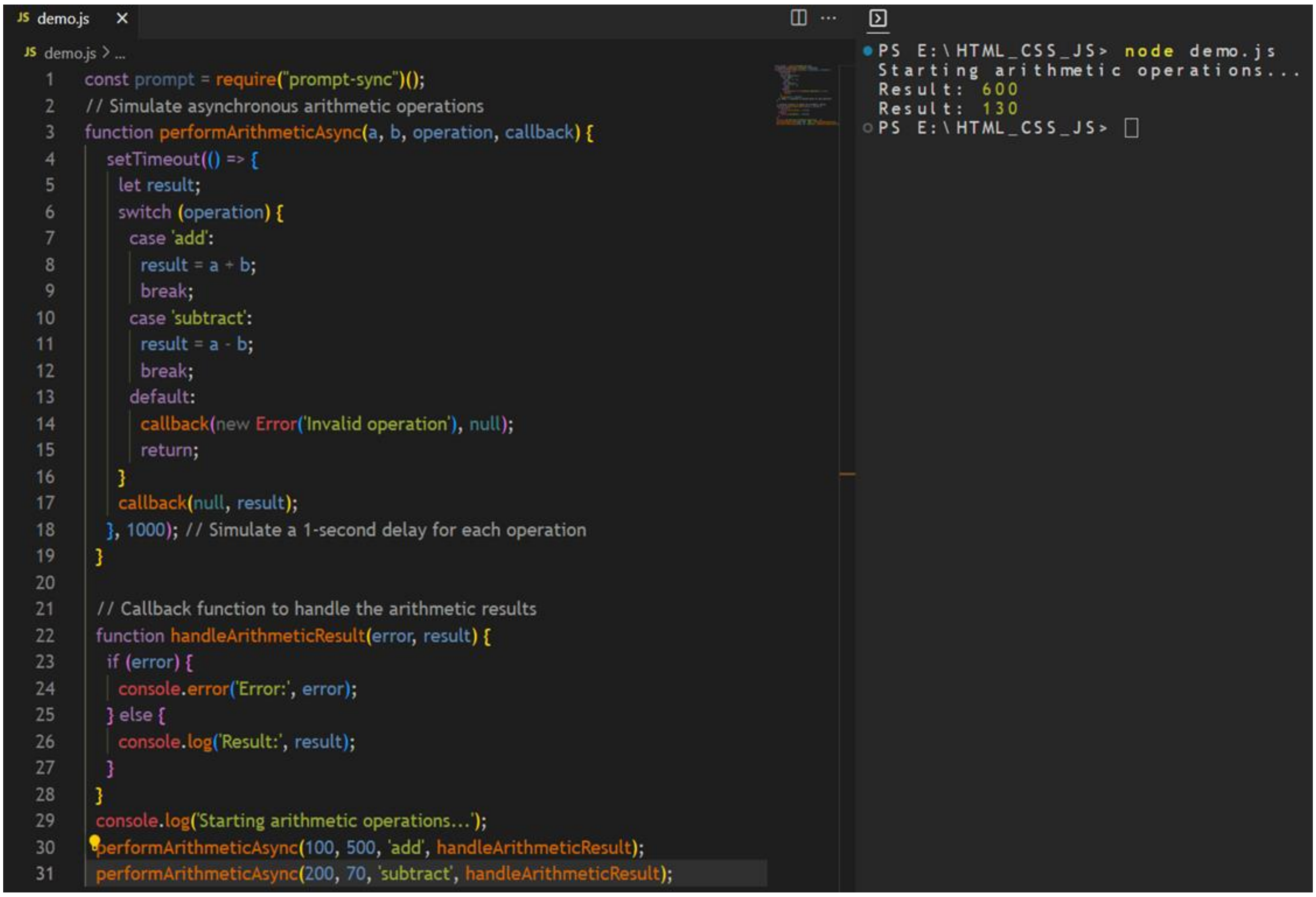


```javascript
const prompt = require("prompt-sync")();
function calculate(num1, num2, callback){
    return callback(num1, num2);
}
function add(a, b){
    return a + b;
}

function mul(a, b){
    return a * b;
}
console.log(calculate(10, 50, add));
console.log(calculate(20, 60, mul));
```

```
PS E:\HTML_CSS_JS> node demo.js
60
1200
PS E:\HTML_CSS_JS>
```

Anonymous & Event-Driven Programming:

Callback functions are commonly used in asynchronous and event-driven programming environments. In these situations, you may need to perform tasks that take some time to complete, such as reading a file, making an HTTP request, or waiting for a user to click a button. Instead of blocking the program's execution while waiting for these tasks to finish, you can specify a callback function to be executed when the task is completed or when a certain event occurs.



```javascript
const prompt = require("prompt-sync")();
// Simulate asynchronous arithmetic operations
function performArithmeticAsync(a, b, operation, callback) {
  setTimeout(() => {
    let result;
    switch (operation) {
      case 'add':
        result = a + b;
        break;
      case 'subtract':
        result = a - b;
        break;
      default:
        callback(new Error('Invalid operation'), null);
        return;
    }
    callback(null, result);
  }, 1000); // Simulate a 1-second delay for each operation
}

// Callback function to handle the arithmetic results
function handleArithmeticResult(error, result) {
  if (error) {
    console.error('Error:', error);
  } else {
    console.log('Result:', result);
  }
}
console.log('Starting arithmetic operations...');
performArithmeticAsync(100, 500, 'add', handleArithmeticResult);
performArithmeticAsync(200, 70, 'subtract', handleArithmeticResult);
```

```
PS E:\HTML_CSS_JS> node demo.js
Starting arithmetic operations...
Result: 600
Result: 130
PS E:\HTML_CSS_JS>
```

Execution Timing:

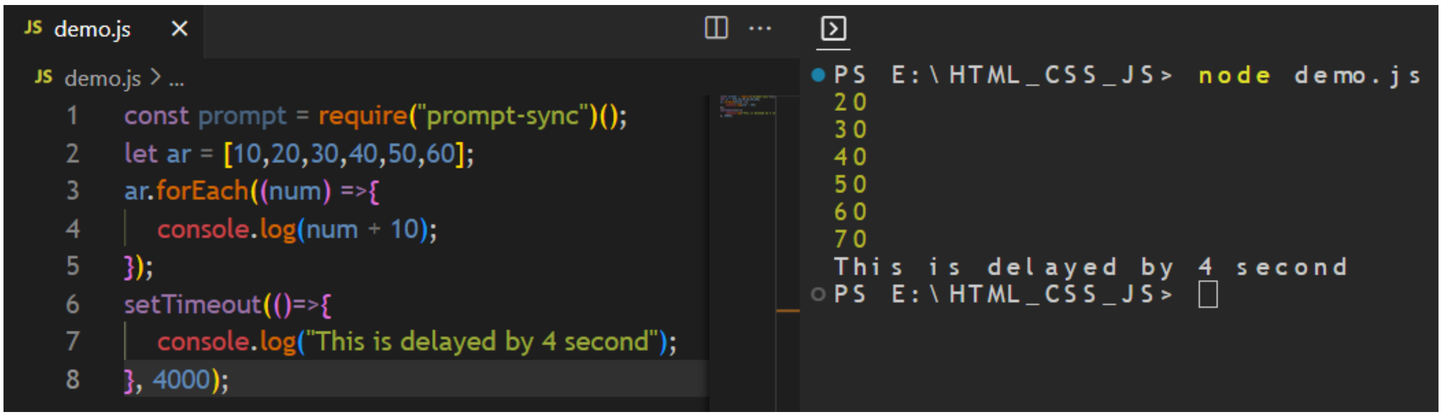
Callback functions can be executed at different times, depending on the context in which they are used. They can be, Synchronous Callback & Asynchronous Callback.

Synchronous Callbacks:

These are executed immediately within the same call stack as the function that uses them. They don't involve asynchronous operations and are typically used for simple operations.

Asynchronous Callbacks:

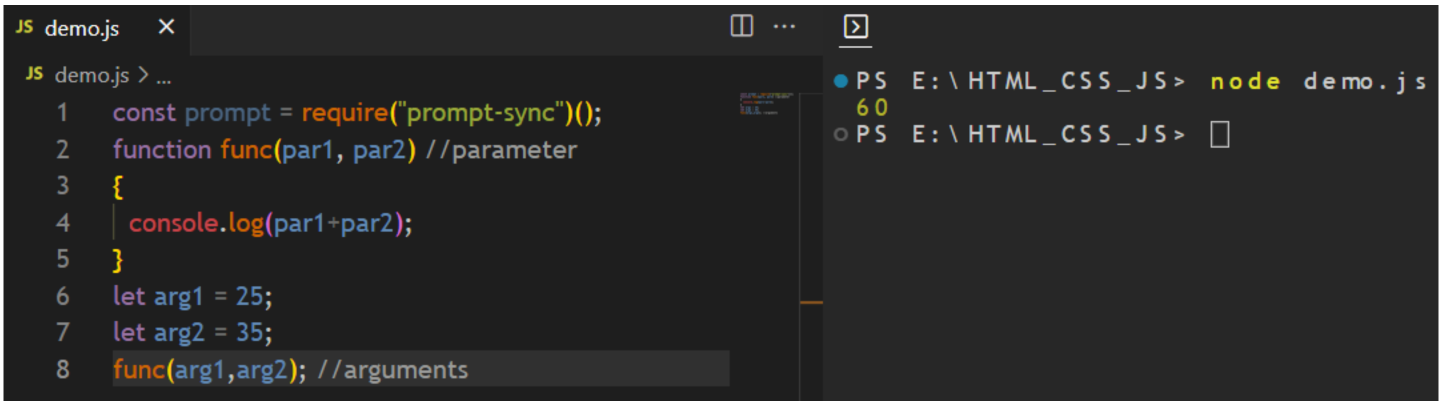
These are executed later, typically after an asynchronous operation like reading a file or making an API request has completed. Asynchronous callbacks are essential for non-blocking code execution. we have an example of both synchronous and asynchronous callback functions. The synchronous forEach method of an array uses a callback function, while the asynchronous setTimeout function uses a callback for delayed execution.



```
const prompt = require("prompt-sync")();
let ar = [10,20,30,40,50,60];
ar.forEach((num) =>{
  console.log(num + 10);
});
setTimeout(()=>{
  console.log("This is delayed by 4 second");
}, 4000);
```

```
PS E:\HTML_CSS_JS> node demo.js
20
30
40
50
60
70
This is delayed by 4 second
PS E:\HTML_CSS_JS>
```

Function Parameters are the names listed in the function definition. Function arguments are the real values passed to and received by the function.

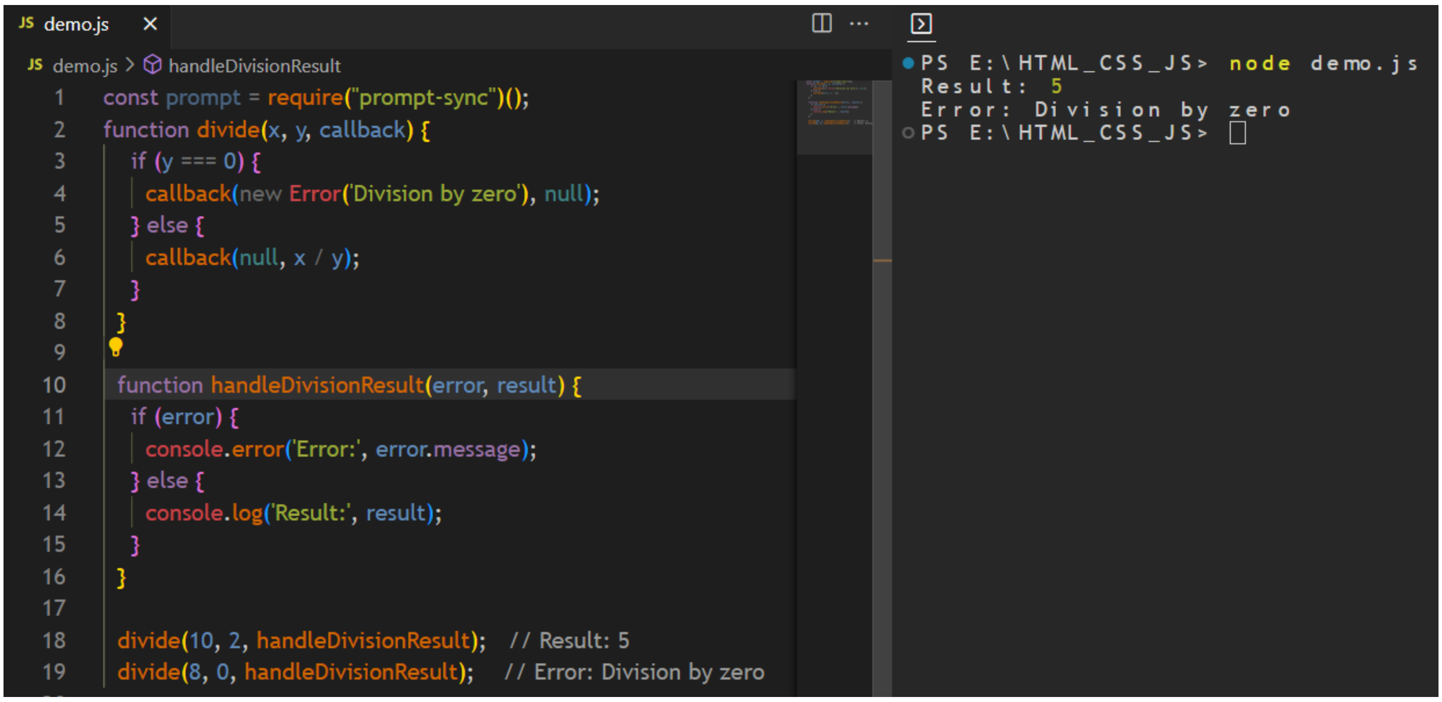


```
const prompt = require("prompt-sync")();
function func(par1, par2) //parameter
{
  console.log(par1+par2);
}
let arg1 = 25;
let arg2 = 35;
func(arg1,arg2); //arguments
```

```
PS E:\HTML_CSS_JS> node demo.js
60
PS E:\HTML_CSS_JS>
```

Error Handling:

Callback functions can also be used to handle errors or exceptions that may occur during the execution of a function. By convention, the first argument passed to a callback is often reserved for an error object, allowing you to check for and handle errors gracefully.



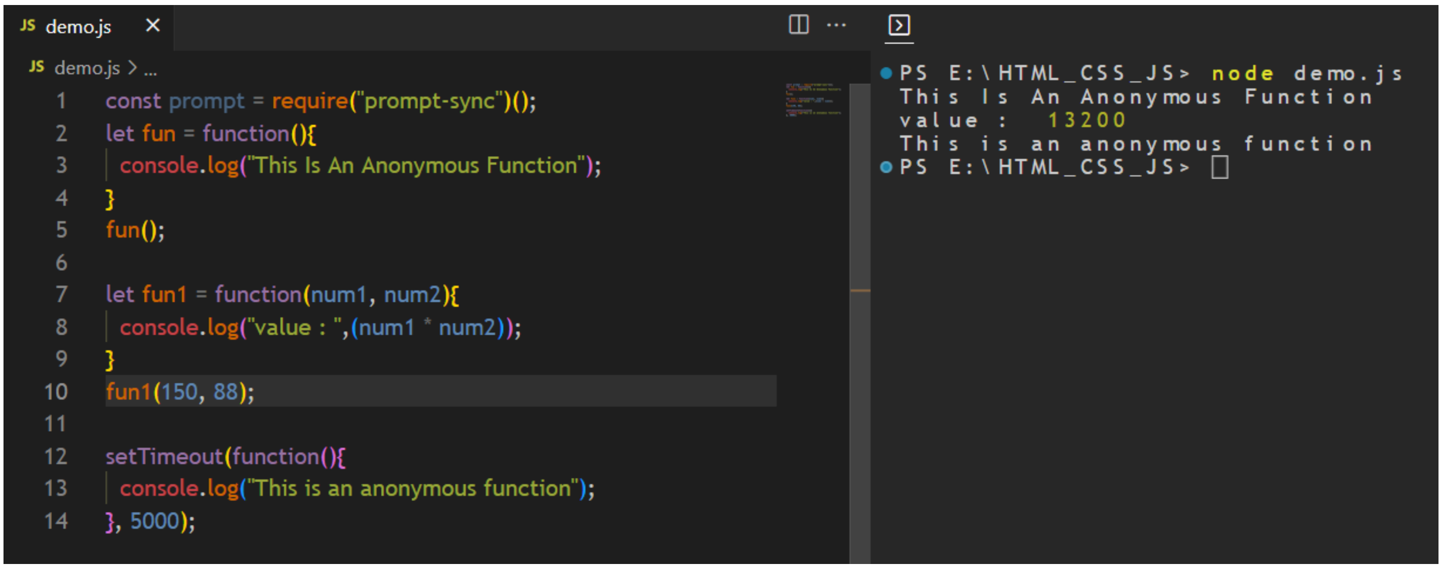
```javascript
const prompt = require("prompt-sync")();
function divide(x, y, callback) {
  if (y === 0) {
    callback(new Error('Division by zero'), null);
  } else {
    callback(null, x / y);
  }
}

function handleDivisionResult(error, result) {
  if (error) {
    console.error('Error:', error.message);
  } else {
    console.log('Result:', result);
  }
}

divide(10, 2, handleDivisionResult);   // Result: 5
divide(8, 0, handleDivisionResult);   // Error: Division by zero
```

```
PS E:\HTML_CSS_JS> node demo.js
Result: 5
Error: Division by zero
PS E:\HTML_CSS_JS>
```

Anonymous Function:

It is a function that does not have any name associated with it. Normally we use the function keyword before the function name to define a function in JavaScript, however, in anonymous functions in JavaScript, we use only the function keyword without the function name.

An anonymous function is not accessible after its initial creation, it can only be accessed by a variable it is stored in as a function as a value. An anonymous function can also have multiple arguments, but only one expression.

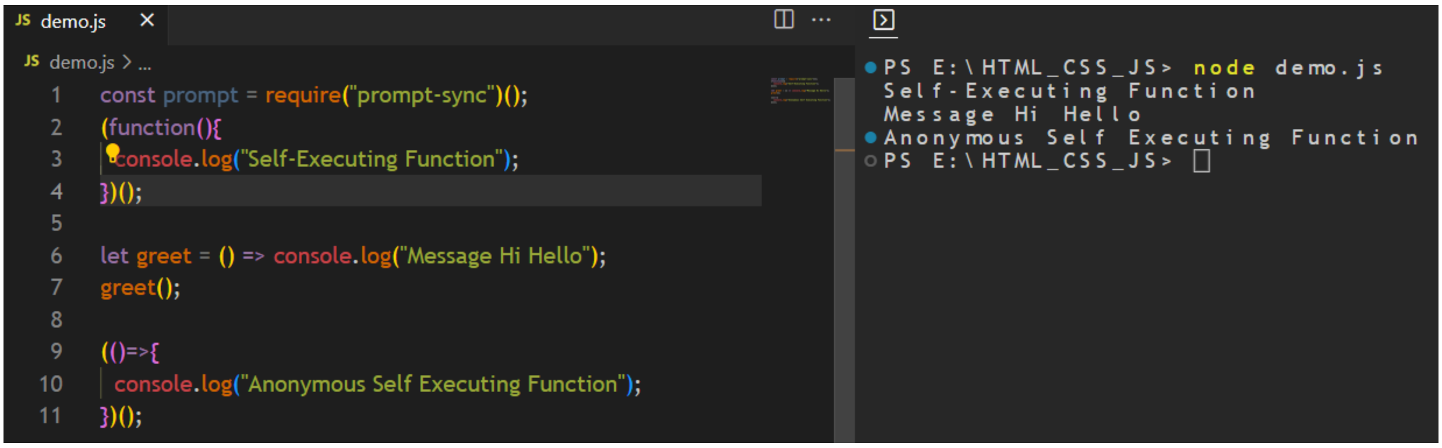


```javascript
const prompt = require("prompt-sync")();
let fun = function(){
  console.log("This Is An Anonymous Function");
}
fun();

let fun1 = function(num1, num2){
  console.log("value : ",(num1 * num2));
}
fun1(150, 88);

setTimeout(function(){
  console.log("This is an anonymous function");
}, 5000);
```

```
PS E:\HTML_CSS_JS> node demo.js
This Is An Anonymous Function
value :  13200
This is an anonymous function
PS E:\HTML_CSS_JS>
```

Self-Executing Function/Immediate Invoked Function:

The self-executing anonymous function is a special function which is invoked right after it is defined. There is no need to call this function anywhere in the script. This type of function has no name and hence it is called an anonymous function. The function has a trailing set of parenthesis. The parameters for this function could be passed in the parenthesis.

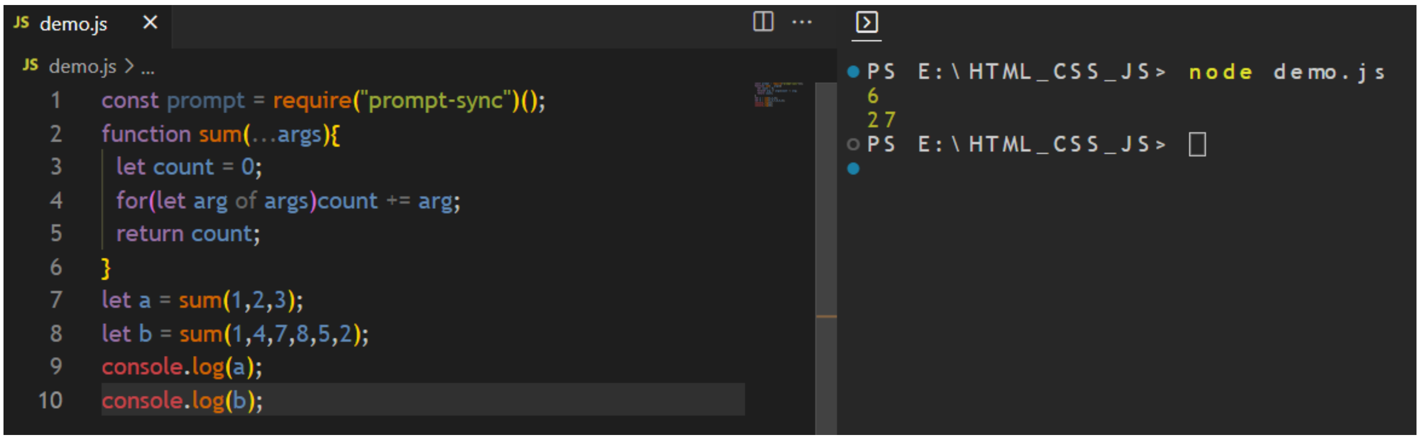


```javascript
const prompt = require("prompt-sync")();
(function(){
  console.log("Self-Executing Function");
})();

let greet = () => console.log("Message Hi Hello");
greet();

(()=>{
  console.log("Anonymous Self Executing Function");
})();
```

```
PS E:\HTML_CSS_JS> node demo.js
Self-Executing Function
Message Hi Hello
Anonymous Self Executing Function
PS E:\HTML_CSS_JS>
```

Rest Parameter:

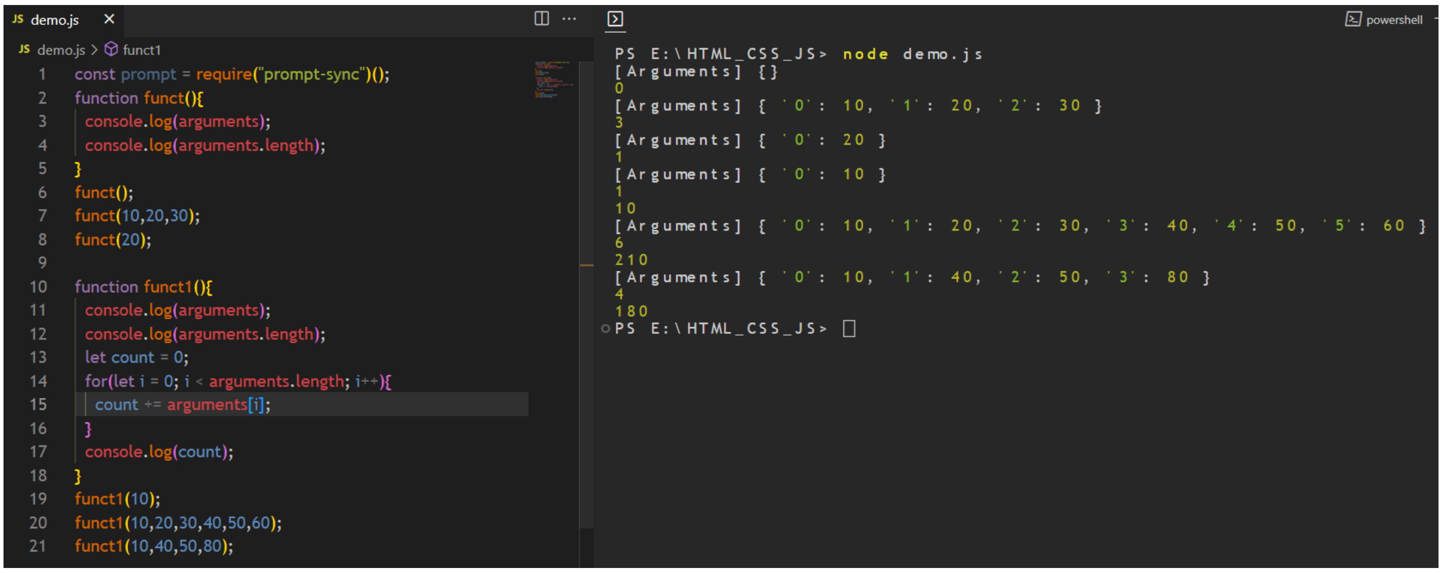
The rest parameter (...) allows a function to treat an indefinite number of arguments as an array.



```javascript
const prompt = require("prompt-sync")();
function sum(...args){
  let count = 0;
  for(let arg of args)count += arg;
  return count;
}
let a = sum(1,2,3);
let b = sum(1,4,7,8,5,2);
console.log(a);
console.log(b);
```

```
PS E:\HTML_CSS_JS> node demo.js
6
27
PS E:\HTML_CSS_JS>
```

Argument Object:



```javascript
const prompt = require("prompt-sync")();
function funct(){
  console.log(arguments);
  console.log(arguments.length);
}
funct();
funct(10,20,30);
funct(20);

function funct1(){
  console.log(arguments);
  console.log(arguments.length);
  let count = 0;
  for(let i = 0; i < arguments.length; i++){
    count += arguments[i];
  }
  console.log(count);
}
funct1(10);
funct1(10,20,30,40,50,60);
funct1(10,40,50,80);
```

```
PS E:\HTML_CSS_JS> node demo.js
[Arguments] {}
0
[Arguments] { '0': 10, '1': 20, '2': 30 }
3
[Arguments] { '0': 20 }
1
[Arguments] { '0': 10 }
1
10
[Arguments] { '0': 10, '1': 20, '2': 30, '3': 40, '4': 50, '5': 60 }
6
210
[Arguments] { '0': 10, '1': 40, '2': 50, '3': 80 }
4
180
PS E:\HTML_CSS_JS>
```