

TRAINING REPORT (TR-102)

DAY: -14 (26 JUNE 2024)

JavaScript Loops: -

Loops are handy if you want to run the same code over and over again, each time with a different value.

Different Kinds of Loops: -

JavaScript supports different kinds of loops:

- for - loops through a block of code several 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

For Loop

The for statement creates a loop with 3 optional expressions:

Syntax: - for (expression1; expression2; expression3)

```
{  
  
// code block to be executed  
  
}
```

Expression 1 is executed (one time) before the execution of the code block.

Expression 2 defines the condition for executing the code block.

Expression 3 is executed (every time) after the code block has been executed.

Example: -

```
for (let i = 0; i < 5; i++)  
{  
  text += "The number is " + i + "<br>";  
}
```

From the example above, you can read: -

Expression 1 sets a variable before the loop starts (let i = 0).

Expression 2 defines the condition for the loop to run (i must be less than 5).

Expression 3 increases a value (i++) each time the code block in the loop has been executed.

While loop: -

The **while loop** executes a block of code as long as a specified condition is true. In JavaScript, this loop evaluates the condition before each iteration and continues running as long as the condition remains true. The loop terminates when the condition becomes false, enabling dynamic and repeated operations based on changing conditions.

Syntax: - while (condition)

```
{  
    //Code block to be executed  
}
```

Do-While loop: -

A **Do-While loop** is another type of loop in JavaScript that is similar to the while loop but with one key difference: the do-while loop guarantees that the block of code inside the loop will be executed at least once, regardless of whether the condition is initially true or false.

Syntax: -

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

For-in loop: -

The **for-in loop in JavaScript** iterates over the enumerable properties of an object, executing a specified block of code for each property. It's commonly used for iterating through object properties or elements in an array.

Syntax: -

```
for (let i in obj1)  
{  
    // Prints all the keys in  
    // obj1 on the console  
    console.log(i);  
}
```

```
}
```

for in Loop Important Points: -

- Use the for-in loop to iterate over non-array objects. Even though we can use a for-in loop for an array, it is generally not recommended. Instead, use a for loop for looping over an array.
- The properties iterated with the for-in loop also include the properties of the objects higher in the prototype chain.
- The order in which properties are iterated may not match the properties that are defined in the object.

JavaScript for...of the loop is used to iterate over iterable objects such as arrays, strings, maps, sets, etc. It provides a simpler syntax compared to traditional loops.

Syntax: -

for (variable of iterableObjectName)

```
{  
    // code block to be executed  
}
```

Parameters:

- **Variable:** Represents the current value of each iteration from the iterable.
- **Iterable:** Any object that can be iterated over (e.g., arrays, strings, maps).

Do-while loop: -

A do...while loop in JavaScript is a control structure where the code executes repeatedly based on a given boolean condition. It's similar to a repeating if statement. One key difference is that a do...while loop guarantees that the code block will execute at least once, regardless of whether the condition is met initially or not.

There are mainly two types of loops: -

- **Entry Controlled loops:** In this type of loop, the test condition is tested before entering the loop body. **For Loop** and **While Loops** are entry-controlled loops.
- **Exit Controlled Loops:** In this type of loop the test condition is tested or evaluated at the end of the loop body. Therefore, the loop body will execute at least once, irrespective of whether the test condition is true or false. the **do-while loop** is exit controlled loop.

Syntax: do

```
{  
    //Statements  
}  
while(conditions)
```

EXAMPLE: -

```
JS script.js
JS script.js > ...
1  for (let num=2; num<=25; num++)
2  {
3      if (num%2==0)
4      {
5          console.log(num)
6      }
7  }
8
```

OUTPUT: -

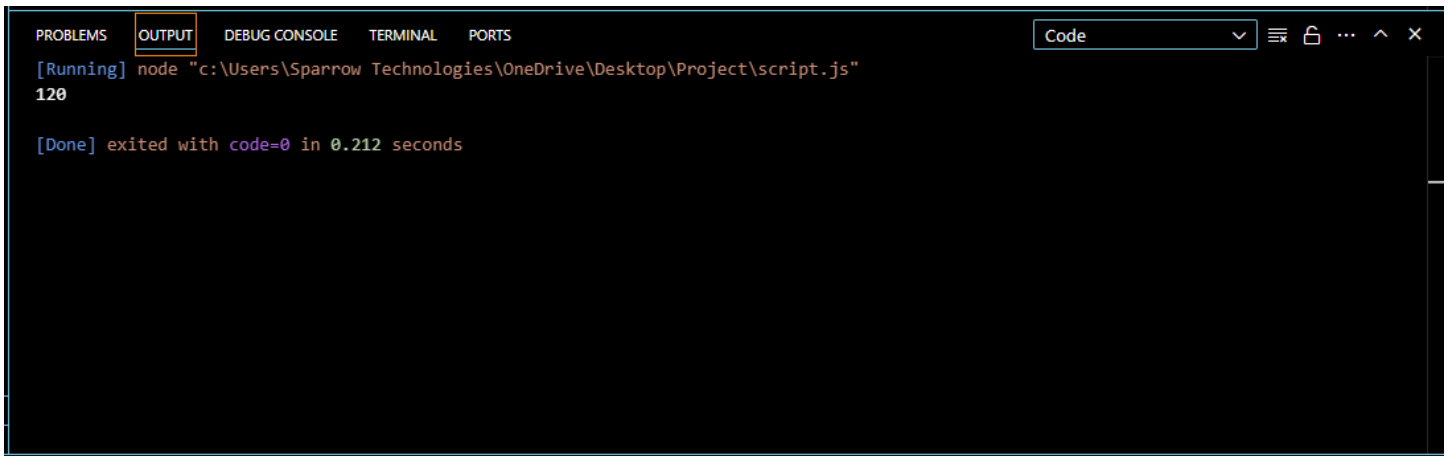
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Code
2
4
6
8
10
12
14
16
18
20
22
24

[Done] exited with code=0 in 2.178 seconds
```

EXAMPLE: -

```
JS script.js
JS script.js > ...
1  let n = 5;
2  function factorial(n) {
3      let ans = 1;
4
5      if(n === 0)
6      {
7          return 1;
8      }
9      for (let i = 2; i <= n; i++)
10     {
11         ans = ans * i;
12     }
13     return ans;
14 }
15 console.log(factorial(n));
```

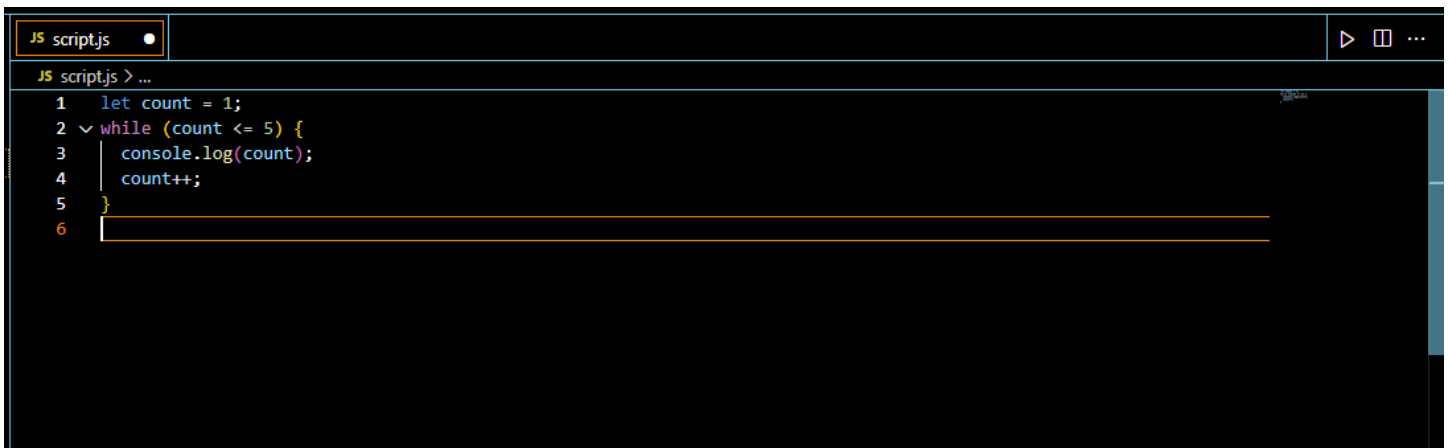
OUTPUT: -



A screenshot of the Visual Studio Code interface. The 'OUTPUT' tab is selected in the top-left pane. The main editor area shows the output of a Node.js script execution. The text reads: [Running] node "c:\Users\Sparrow Technologies\OneDrive\Desktop\Project\script.js" 120 [Done] exited with code=0 in 0.212 seconds.

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[Running] node "c:\Users\Sparrow Technologies\OneDrive\Desktop\Project\script.js"
120
[Done] exited with code=0 in 0.212 seconds
```

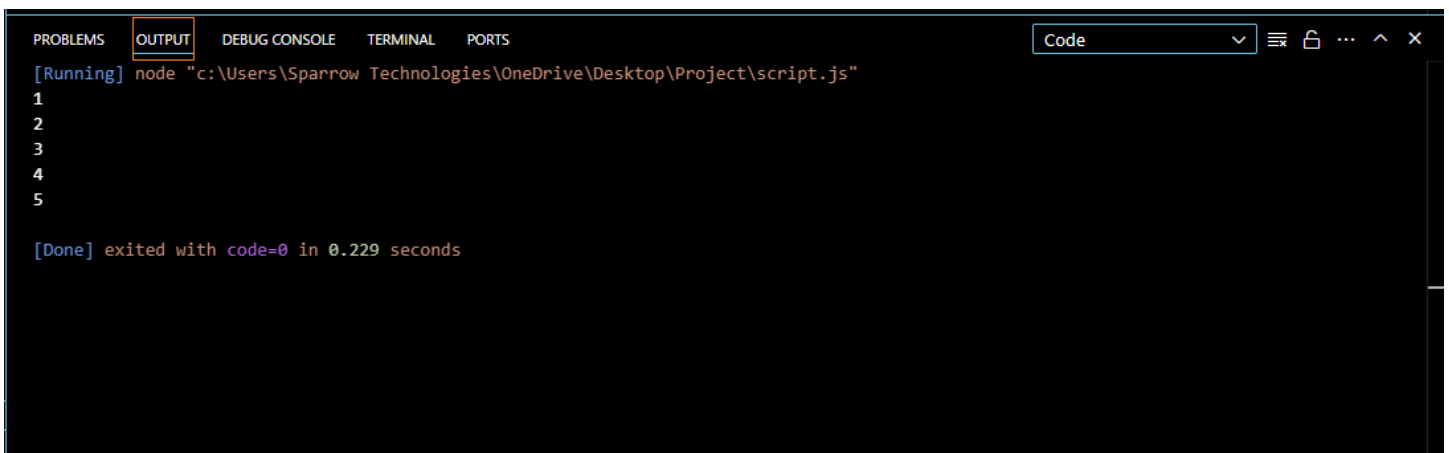
EXAMPLE: -



A screenshot of the Visual Studio Code editor. The file 'script.js' is open. The code is a JavaScript script that initializes a variable 'count' to 1 and enters a while loop that logs the count to the console and increments it until it reaches 5.

```
JS script.js
JS script.js > ...
1 let count = 1;
2 while (count <= 5) {
3   console.log(count);
4   count++;
5 }
6
```

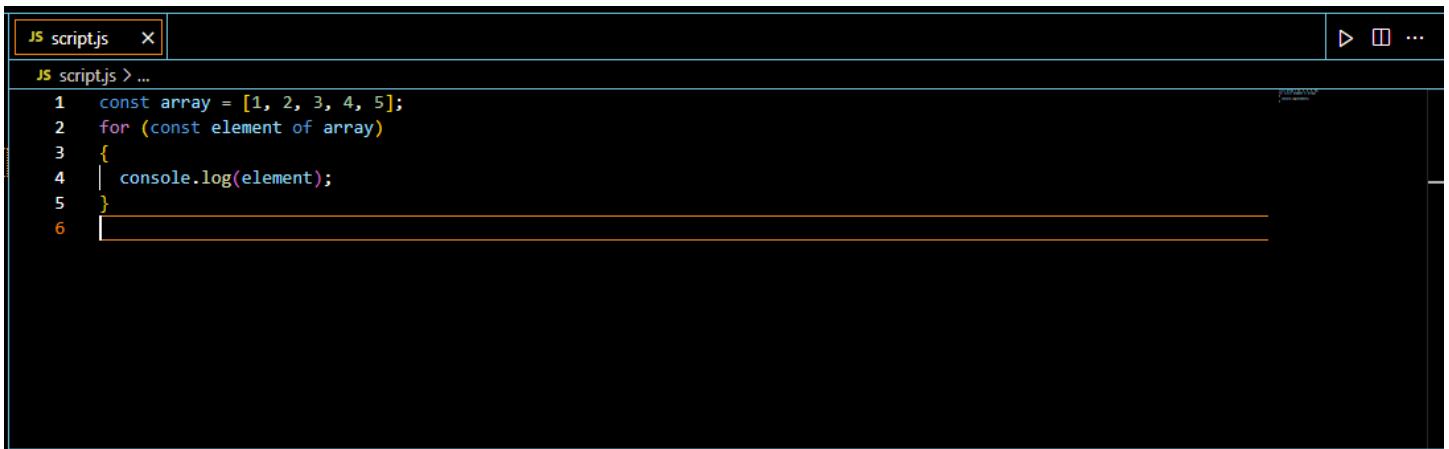
OUTPUT: -



A screenshot of the Visual Studio Code interface. The 'OUTPUT' tab is selected. The main editor area shows the output of the example script. The text reads: [Running] node "c:\Users\Sparrow Technologies\OneDrive\Desktop\Project\script.js" 1 2 3 4 5 [Done] exited with code=0 in 0.229 seconds.

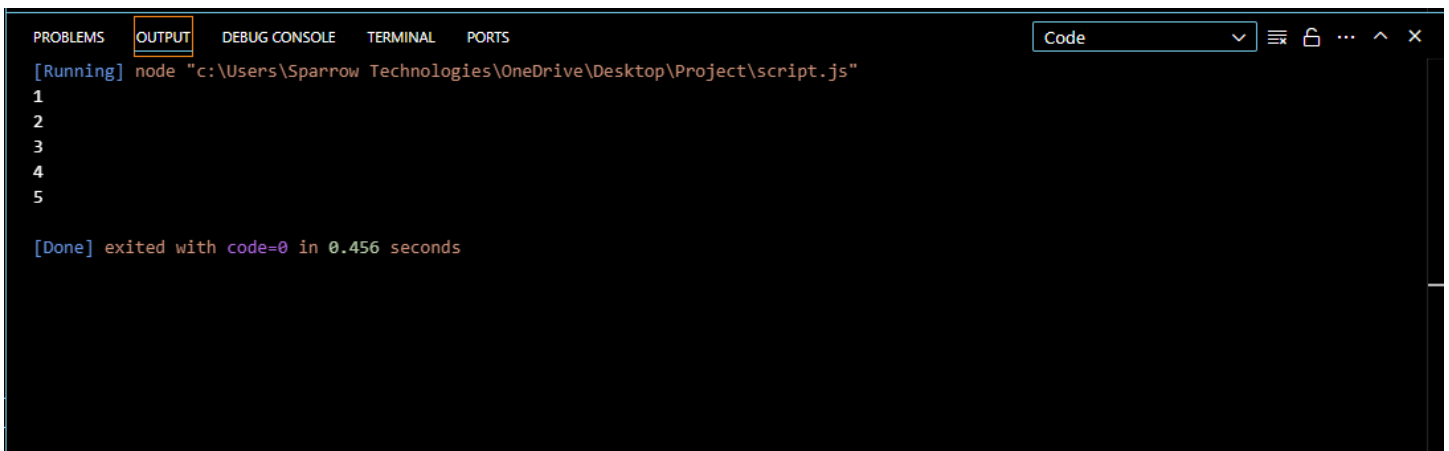
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
[Running] node "c:\Users\Sparrow Technologies\OneDrive\Desktop\Project\script.js"
1
2
3
4
5
[Done] exited with code=0 in 0.229 seconds
```

EXAMPLE: -



```
JS script.js X
JS script.js > ...
1  const array = [1, 2, 3, 4, 5];
2  for (const element of array)
3  {
4  |  console.log(element);
5  |
6  |
```

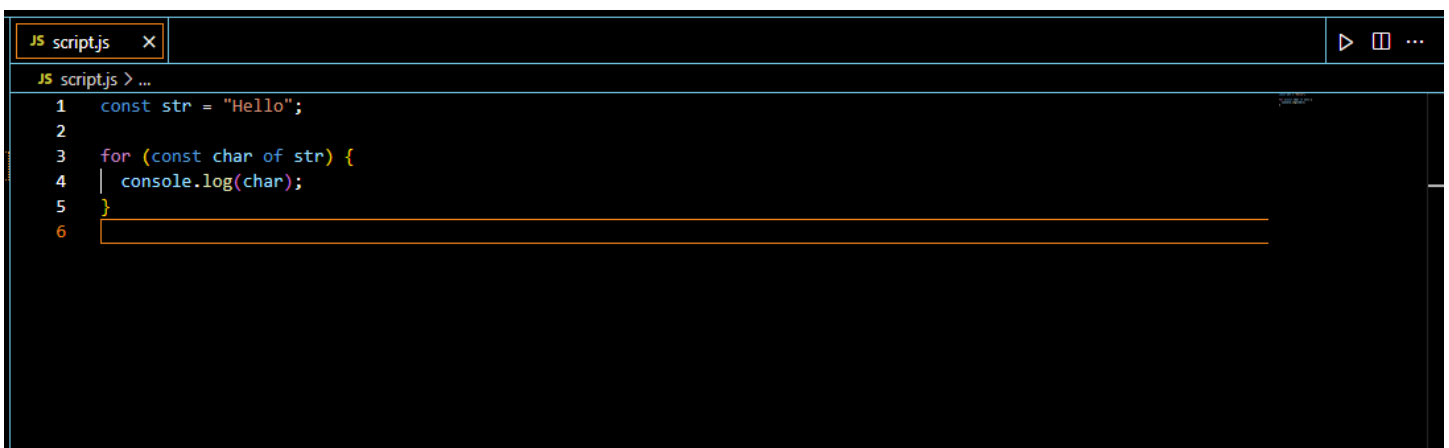
OUTPUT: -



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Code
[Running] node "c:\Users\Sparrow Technologies\OneDrive\Desktop\Project\script.js"
1
2
3
4
5

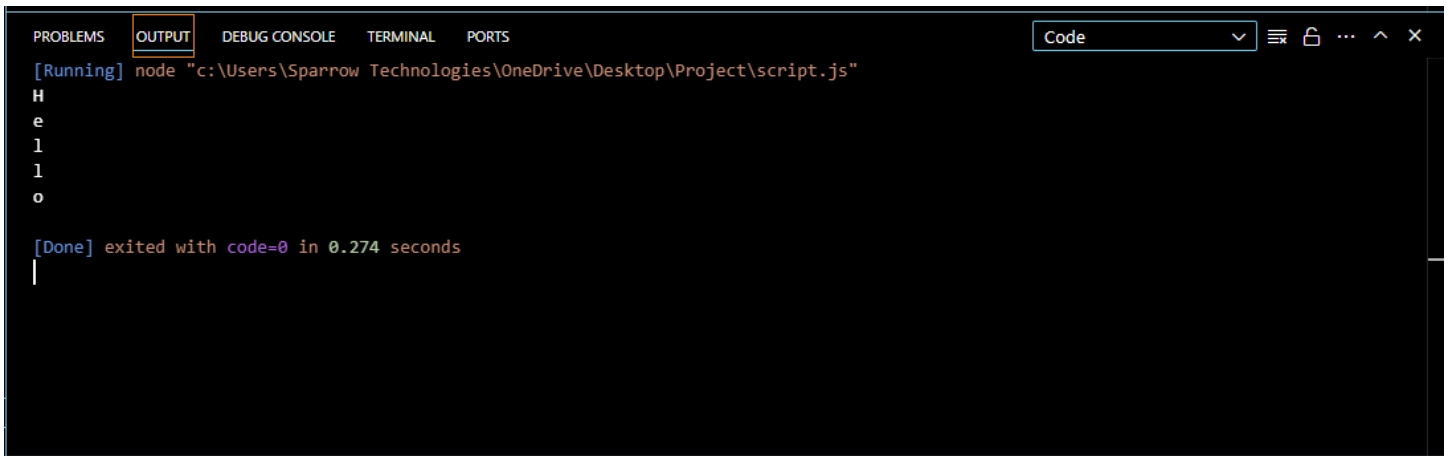
[Done] exited with code=0 in 0.456 seconds
```

EXAMPLE: -



```
JS script.js X
JS script.js > ...
1  const str = "Hello";
2
3  for (const char of str) {
4  |  console.log(char);
5  |
6  |
```

OUTPUT: -

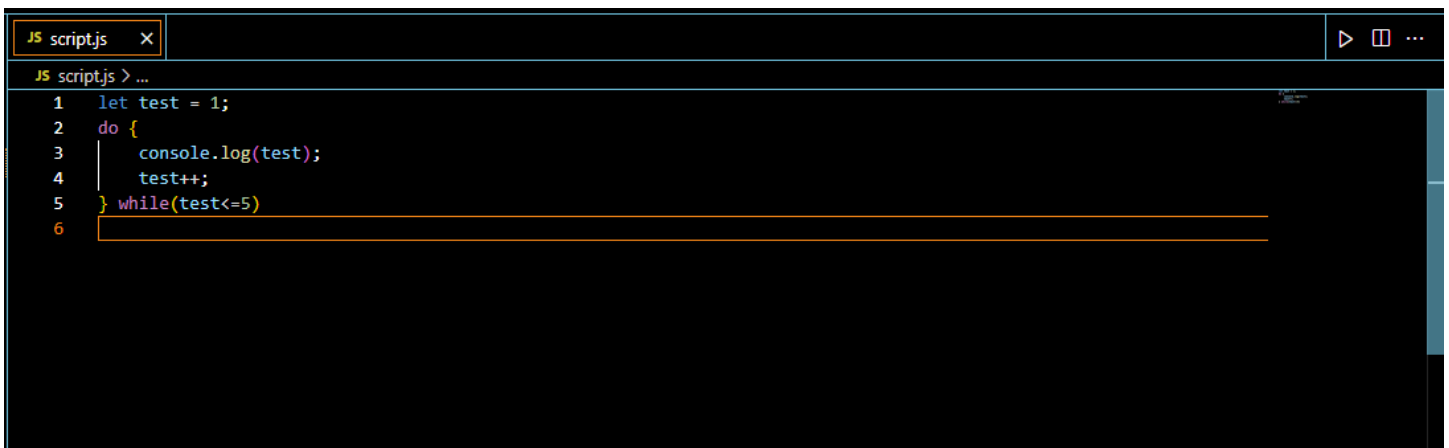


A screenshot of the Visual Studio Code interface. The 'OUTPUT' tab is selected, showing the output of a Node.js script. The command executed is `node "c:\Users\Sparrow Technologies\OneDrive\Desktop\Project\script.js"`. The output displays the word 'Hello' on separate lines. Below the output, it states `[Done] exited with code=0 in 0.274 seconds`. The 'PROBLEMS', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS' tabs are visible at the top, along with a 'Code' dropdown menu and window control icons.

```
[Running] node "c:\Users\Sparrow Technologies\OneDrive\Desktop\Project\script.js"
H
e
l
l
o

[Done] exited with code=0 in 0.274 seconds
```

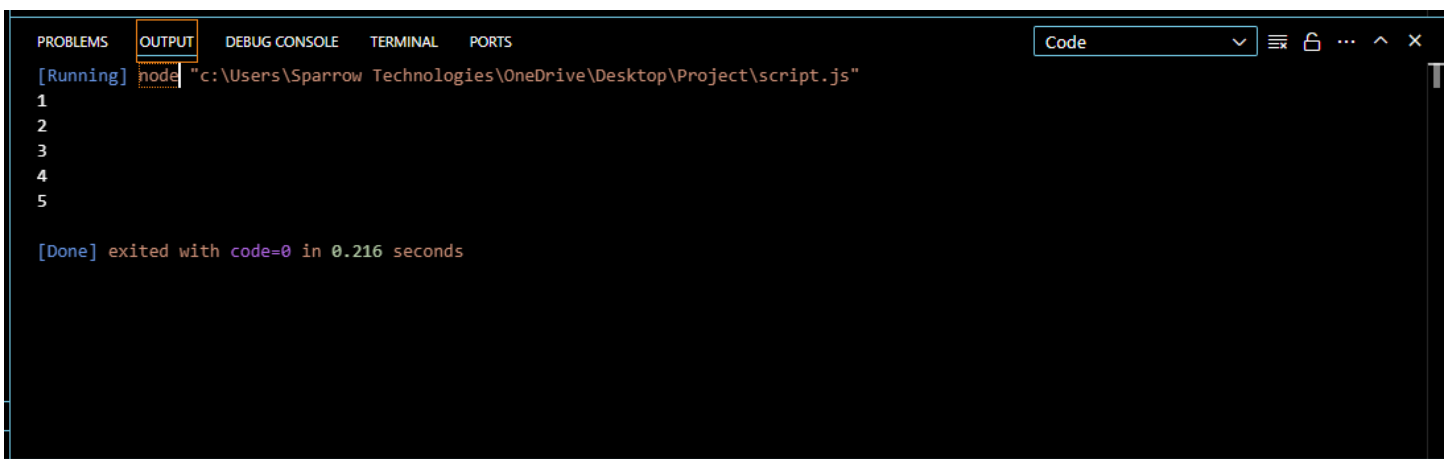
EXAMPLE: -



A screenshot of the Visual Studio Code editor. A file named `script.js` is open, showing a JavaScript code example. The code is as follows:

```
1 let test = 1;
2 do {
3   console.log(test);
4   test++;
5 } while(test<=5)
6
```

OUTPUT: -



A screenshot of the Visual Studio Code interface. The 'OUTPUT' tab is selected, showing the output of the example code. The command executed is `node "c:\Users\Sparrow Technologies\OneDrive\Desktop\Project\script.js"`. The output displays the numbers 1 through 5 on separate lines. Below the output, it states `[Done] exited with code=0 in 0.216 seconds`. The 'PROBLEMS', 'DEBUG CONSOLE', 'TERMINAL', and 'PORTS' tabs are visible at the top, along with a 'Code' dropdown menu and window control icons.

```
[Running] node "c:\Users\Sparrow Technologies\OneDrive\Desktop\Project\script.js"
1
2
3
4
5

[Done] exited with code=0 in 0.216 seconds
```