

JAVA TASK

Anukrishnan MK

1. Write a program to find the area of a triangle using function

```
DAY8 > task.js > area
1 //Write a program to find the area of a triangle using function
2 //5*b*h
3
4 function area(b,h){
5     area=1/2*b*h;
6     return area;
7 }
8
9 console.log(area(4,6));
10
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> node task.js
12
○ PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> 
```

2. Write a JavaScript function that accepts a number as a parameter and check the number is prime or not.

```
DAY8 > task.js > ...
10
11 //Write a JavaScript function that accepts a number as a parameter and check the number is prime or not.
12
13 function prime(p) {
14     let t = 0;
15     let f = 0;
16     for (let i = 1; i <= p; i++) {
17         if (p % i == 0) t++;
18     }
19     return t == 2 ? "prime" : "Not prime";
20 }
21
22 console.log(prime(4));
23
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

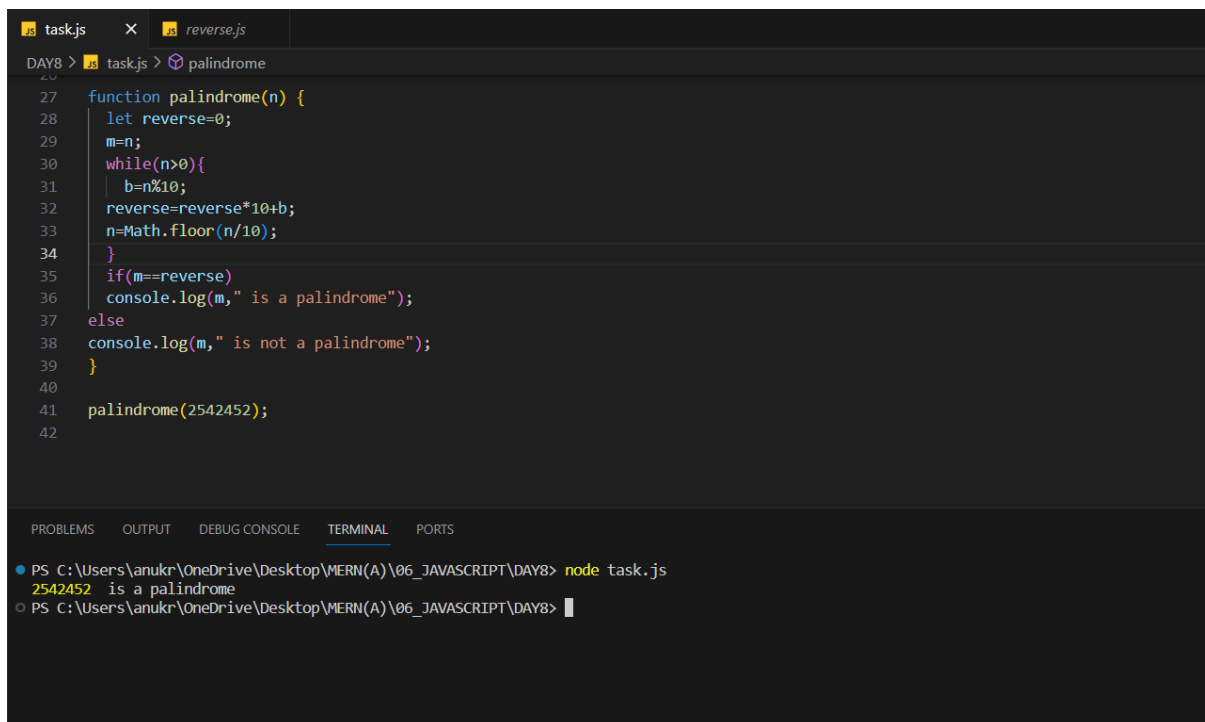
```
● PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> node task.js
Not prime
○ PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> 
```

```
DAY8 > task.js > ...
10
11 //Write a JavaScript function that accepts a number as a parameter and check the number is prime or not.
12
13 function prime(p) {
14     let t = 0;
15     let f = 0;
16     for (let i = 1; i <= p; i++) {
17         if (p % i == 0) t++;
18     }
19     return t == 2 ? "prime" : "Not prime";
20 }
21
22 console.log(prime(11));
23
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
● PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> node task.js
Not prime
● PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> node task.js
prime
○ PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> 
```

3. Write a JavaScript function that checks whether a passed string is palindrome or not?

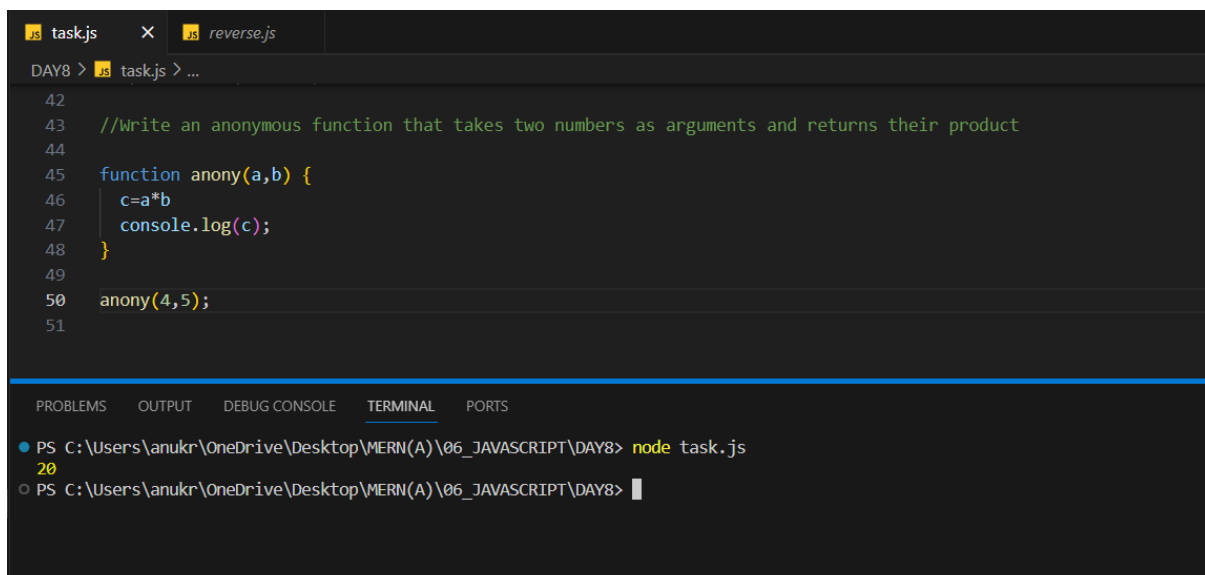


```
task.js X reverse.js
DAY8 > task.js > palindrome
27 function palindrome(n) {
28   let reverse=0;
29   m=n;
30   while(n>0){
31     b=n%10;
32     reverse=reverse*10+b;
33     n=Math.floor(n/10);
34   }
35   if(m==reverse)
36     console.log(m," is a palindrome");
37   else
38     console.log(m," is not a palindrome");
39   }
40
41   palindrome(2542452);
42
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> node task.js
2542452 is a palindrome
PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8>
```

4. Write an anonymous function that takes two numbers as arguments and returns their product.

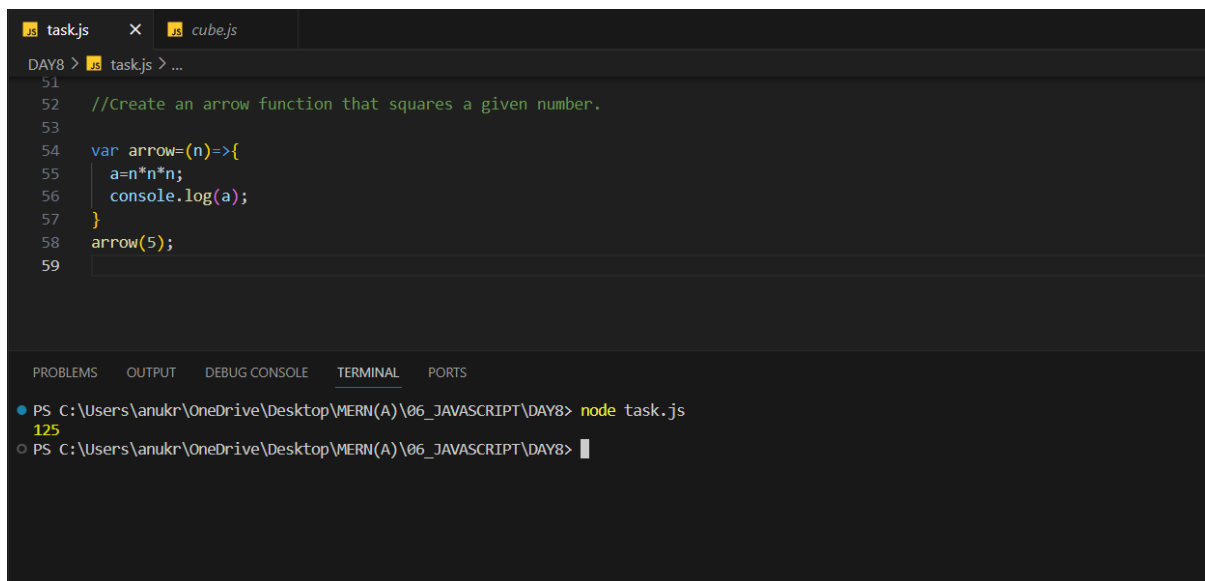


```
task.js X reverse.js
DAY8 > task.js > ...
42
43 //Write an anonymous function that takes two numbers as arguments and returns their product
44
45 function anony(a,b) {
46   c=a*b
47   console.log(c);
48 }
49
50 anony(4,5);
51
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> node task.js
20
PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8>
```

5. Create an arrow function that squares a given number.

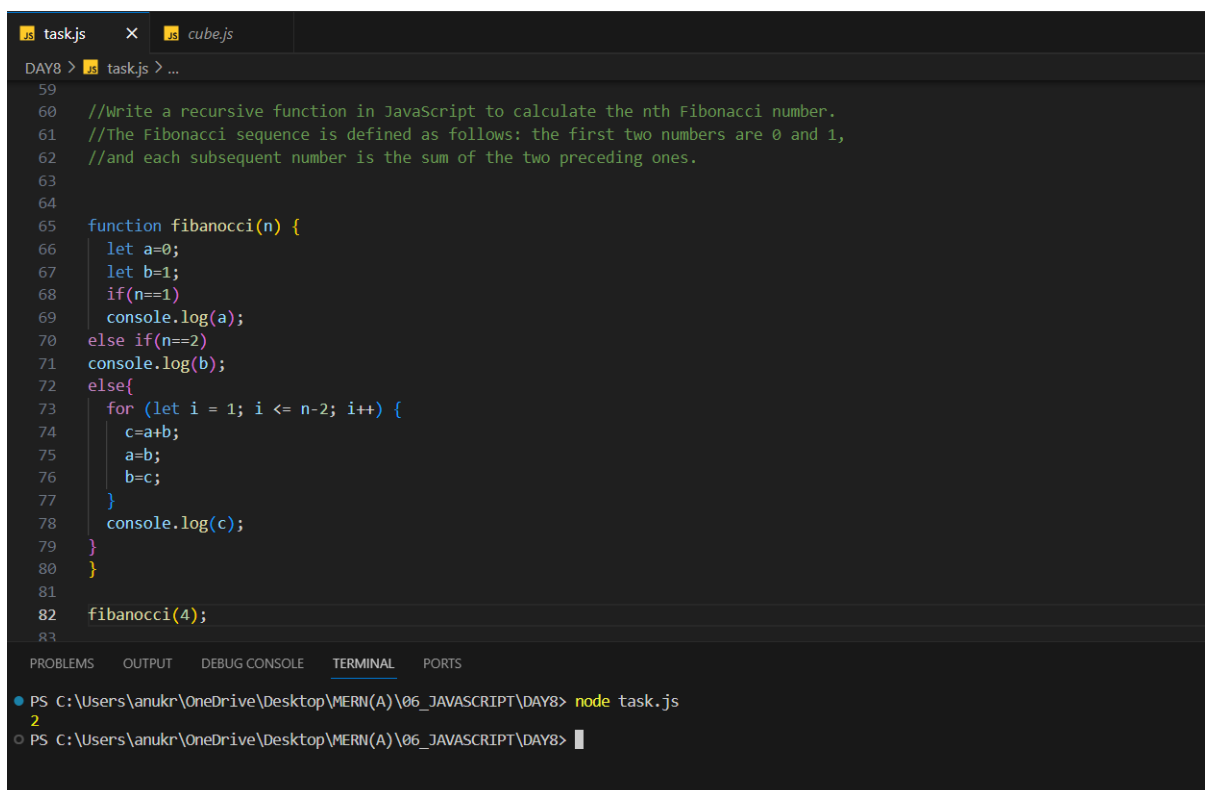


```
task.js
DAY8 > task.js > ...
51
52 //Create an arrow function that squares a given number.
53
54 var arrow=(n)=>{
55   a=n*n*n;
56   console.log(a);
57 }
58 arrow(5);
59
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> node task.js
125
○ PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> █

6. Write a recursive function in JavaScript to calculate the nth Fibonacci number. The Fibonacci sequence is defined as follows: the first two numbers are 0 and 1, and each subsequent number is the sum of the two preceding ones.



```
task.js
DAY8 > task.js > ...
59
60 //Write a recursive function in JavaScript to calculate the nth Fibonacci number.
61 //The Fibonacci sequence is defined as follows: the first two numbers are 0 and 1,
62 //and each subsequent number is the sum of the two preceding ones.
63
64
65 function fibanocci(n) {
66   let a=0;
67   let b=1;
68   if(n==1)
69     console.log(a);
70   else if(n==2)
71     console.log(b);
72   else{
73     for (let i = 1; i <= n-2; i++) {
74       c=a+b;
75       a=b;
76       b=c;
77     }
78     console.log(c);
79   }
80 }
81
82 fibanocci(4);
83
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

● PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> node task.js
2
○ PS C:\Users\anukr\OneDrive\Desktop\MERN(A)\06_JAVASCRIPT\DAY8> █

7.What is the difference between function parameters and arguments?

Parameters:

- Parameters are the variables listed in the function declaration.
- They act as placeholders for values that will be provided when the function is called.
- Parameters are part of the function's signature and are used within the function body to perform operations.

```
function add(x, y) {  
  // x and y are parameters  
  return x + y;  
}
```

In this example, `add` is a function with two parameters, `x` and `y`.

Arguments:

- Arguments are the actual values or expressions passed to a function when it is invoked.
- They correspond to the parameters defined in the function, and their values are used within the function.
- Arguments are specific to a particular function call and are provided in the parentheses during the function invocation.

```
const result = add(3, 5);  
// 3 and 5 are arguments passed to the add function
```

In the `add(3, 5)` call, `3` and `5` are the arguments passed to the `add` function, and they are assigned to the parameters `x` and `y` respectively.

8. What is the purpose of the return statement in a function, and what does it do?

In JavaScript, the `return` statement is used within a function to specify the value that the function should return when it is called or invoked. The `return` statement serves several purposes:

Returning a Value

- The primary purpose of the `return` statement is to provide a value as the result of a function.
- When the function is called, the expression following the `return` keyword is evaluated, and the result becomes the value returned by the function.

Ending Function Execution:

- The `return` statement also serves to exit the function immediately, stopping further execution of the function's code.
- Any code following the `return` statement within the function will not be executed.

Undefined Return:

- If a function does not have a `return` statement, or if the `return` statement is used without an expression, the function implicitly returns `undefined`.
- Functions without a `return` statement also return `undefined`.

Returning Objects, Arrays, or Any Value:

- The `return` statement can be used to return any valid JavaScript value, including objects, arrays, strings, numbers, etc.

9. What are the differences between arrow functions and regular functions in JavaScript?

Arrow functions and regular (or traditional) functions in JavaScript have some key differences, including syntax, behavior, and handling of the `this` keyword. Here are the main distinctions:

Syntax:

- Arrow functions have a concise syntax, especially when the function has only one statement. They don't require the `function` keyword or curly braces if the function body consists of a single expression.
- Regular functions have a more verbose syntax and always require the `function` keyword.

`this` Binding:

- Arrow functions do not have their own `this` binding. Instead, they inherit the `this` value from the enclosing scope (lexical scoping).
- Regular functions have their own `this` binding, and its value is determined by how the function is called.

Arguments Object:

- Arrow functions do not have their own `arguments` object. If you need access to the arguments, you can use the rest parameters syntax.
- Regular functions have an `arguments` object that holds all passed arguments.

`new` Keyword:

- Arrow functions cannot be used as constructors with the `new` keyword. They do not have their own `this` and cannot be instantiated.
- Regular functions can be used as constructors with `new`, allowing you to create instances of objects.

Use Cases:

- Arrow functions are often preferred for short, concise functions, especially when dealing with functional programming constructs like higher-order functions.
- Regular functions are more suitable for scenarios where you need the flexibility of `this` binding, or when defining methods in an object.

10. What is the difference between local and global scope in JavaScript functions?

Global Scope:

- Variables declared outside of any function or block have global scope.
- Global variables are accessible from any part of the code, including inside functions.
- They are typically declared using the `var` keyword (though in modern JavaScript, `let` and `const` can also be used for global variables).

Local Scope:

- Variables declared inside a function or block have local scope.
- Local variables are only accessible within the function or block where they are declared.
- They are typically declared using `var`, `let`, or `const` keywords.
- Variables declared with `let` or `const` have block scope, meaning they are only accessible within the block (enclosed by curly braces) where they are declared.