**Class 3**

* **npm I prompt-sync** To install the "prompt-sync" third party package package, you can use the following command in your terminal. The "prompt-sync" package allows you to synchronously read user input from the command line in Node.js applications, making it convenient for creating interactive command-line interfaces.
* The command **"npm i @types/node -D"** is used to install TypeScript type declarations for Node.js as a development dependency in your project. **npm**: npm is the package manager for Node.js and is used to install, manage, and distribute Node.js modules (packages). **i**: Short for "install," this is the npm command used to install packages. **@types/node**: This is the package name of TypeScript type declarations for Node.js. **-D**: This flag stands for "development" and indicates that the package will be installed as a development dependency

Functions:

A function is a **self-contained block** of code that performs a specific task or set of tasks. Functions are designed to be **reusable** and **modular, allowing** developers to break down complex tasks into smaller, manageable pieces of code.

Functions typically have the following components:

1. **Function Signature**: This includes the function name and its parameters (if any). Parameters are placeholders for the values that the function expects to receive when it is called.
2. **Function Body**: This is the block of code inside the function that defines what the function does. It contains the statements and actions that will be executed when the function is called.
3. **Return Statement**: If a function is expected to produce a result, it can use the **return** statement to send that value back to the caller. The **return** statement also ends the function's execution.

* Building functions
* User defined functions

types of functions

1. no input no return
2. pass input but no return

* **return**: Used inside functions to send a value back to the caller and terminate the function's execution.
* **console.log()**: Used to print messages or values to the console for debugging or informational purposes. It does not affect the flow of the program and does not have any impact on the value returned by a function.

Arrow functions are often used as a shorter and more readable alternative to traditional function expressions. They have a few key features:

* **Concise Syntax:**

Arrow functions use a more compact syntax, which can make code more readable, especially for short functions.

* **Implicit Return:**

If the function body has only one expression, the return statement is implicit. You don't need to use the **return** keyword explicitly; the result of the expression is automatically returned.

* **Lexical this Binding:**

Arrow functions do not have their own **this** context. Instead, they lexically capture the **this** value from the surrounding code. This is particularly useful in situations where you need to access the enclosing scope's **this**.

**ARRAY**

An array is a data structure that can hold a collection of values, such as numbers, strings, or even other objects.

Array built-in functions

**PUSH**

**POP**

**SHIFT**

**UNSHIFT**

**SPLICE**

Sure! Let's go through each of the array methods and implement them in TypeScript:

**PUSH:**

The push method is used to add one or more elements to the end of an array.

**typescript**

const numbers: number[] = [1, 2, 3];

numbers.push(4);

console.log(numbers); // Output: [1, 2, 3, 4]

**POP:**

The pop method removes the last element from an array and returns that element.

**Typescript**

const numbers: number[] = [1, 2, 3, 4];

const lastElement = numbers.pop();

console.log(lastElement); // Output: 4

console.log(numbers); // Output: [1, 2, 3]

**SHIFT:**

The shift method removes the first element from an array and shifts the remaining elements to lower indexes.

**typescript**

const numbers: number[] = [1, 2, 3];

const firstElement = numbers.shift();

console.log(firstElement); // Output: 1

console.log(numbers); // Output: [2, 3]

**UNSHIFT:**

The unshift method adds one or more elements to the beginning of an array and shifts the existing elements to higher indexes.

**typescript**

const numbers: number[] = [2, 3];

numbers.unshift(1);

console.log(numbers); // Output: [1, 2, 3]

**SPLICE:**

The splice method is used to add, remove, or replace elements in an array at a specified index.

**SLICE:**

Slices out piece of an array

**typescript**

const numbers: number[] = [1, 2, 3, 4, 5];

// Remove two elements starting from index 1

const removedElements = numbers.splice(1, 2);

console.log(removedElements); // Output: [2, 3]

console.log(numbers); // Output: [1, 4, 5]

// Add elements starting from index 1, removing 0 elements

numbers.splice(1, 0, 6, 7);

console.log(numbers); // Output: [1, 6, 7, 4, 5]

// Replace one element at index 3 with 8

numbers.splice(3, 1, 8);

console.log(numbers); // Output: [1, 6, 7, 8, 5]