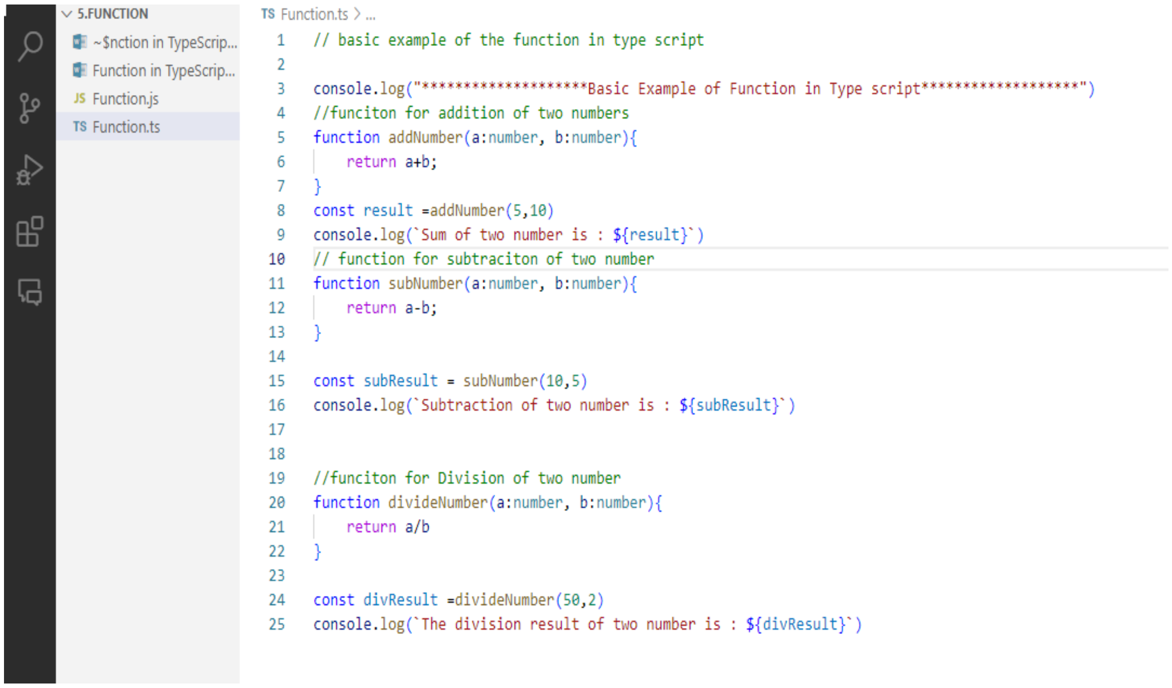
Functions in TypeScript

Function:

Functions are a fundamental concept in programming that allow you to group a sequence of statements together to perform a specific task. They are reusable units of code that can be called or invoked (means to call or execute a function) with a set of inputs, and they can return a result.

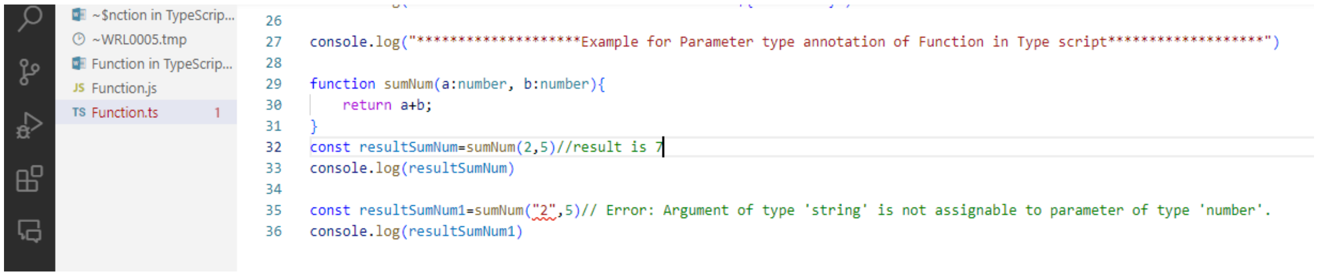
In TypeScript, which is a superset of JavaScript, functions have a similar role. However, TypeScript introduces static typing, which means you can define the types of function parameters and return values. This helps catch errors during development and provides better code completion and documentation.



```
// basic example of the function in type script

console.log("*******************Basic Example of Function in Type script******************")
//funciton for addition of two numbers
function addNumber(a:number, b:number){
    return a+b;
}
const result =addNumber(5,10)
console.log(`Sum of two number is : ${result}`)
// function for subtraciton of two number
function subNumber(a:number, b:number){
    return a-b;
}

const subResult = subNumber(10,5)
console.log(`Subtraction of two number is : ${subResult}`)


//funciton for Division of two number
function divideNumber(a:number, b:number){
    return a/b
}

const divResult =divideNumber(50,2)
console.log(`The division result of two number is : ${divResult}`)
```

Parameter type annotation:

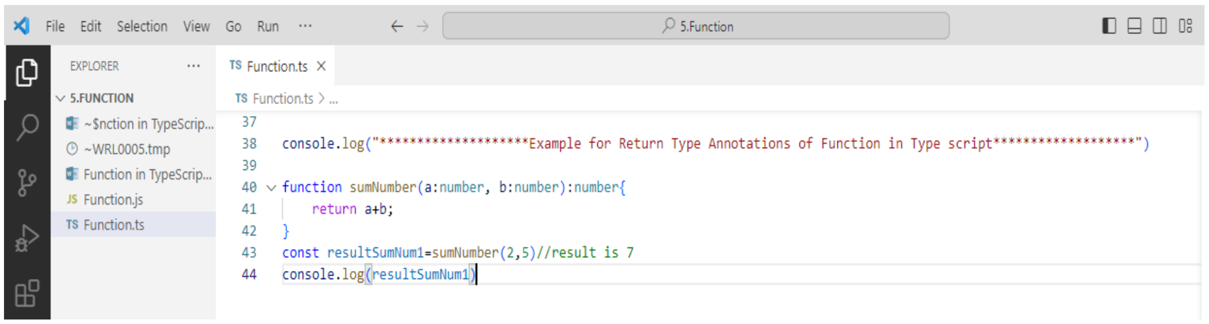
You can provide type annotations for function parameters to specify the expected types of the arguments. This helps catch potential errors during development and provides better documentation for your code.



```

console.log("*******************Example for Parameter type annotation of Function in Type script******************")

function sumNum(a:number, b:number){
    return a+b;
}
const resultSumNum=sumNum(2,5)//result is 7
console.log(resultSumNum)

const resultSumNum1=sumNum("2",5)// Error: Argument of type 'string' is not assignable to parameter of type 'number'.
console.log(resultSumNum1)
```

Return Type Annotations:

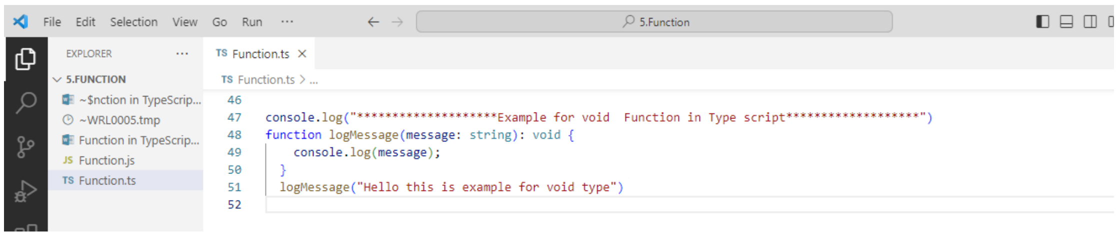
You can provide a return type annotation for a function to specify the type of value that the function is expected to return. This is useful for catching errors during development and providing better documentation.



```typescript

console.log("*******************Example for Return Type Annotations of Function in Type script******************")

function sumNumber(a:number, b:number):number{
    return a+b;
}
const resultSumNum1=sumNumber(2,5)//result is 7
console.log(resultSumNum1)
```

void:

If a function doesn't return any value, you can specify **void** as the return type.



```typescript

console.log("*******************Example for void  Function in Type script******************")
function logMessage(message: string): void {
    console.log(message);
 }
 logMessage("Hello this is example for void type")

```

Purpose of Functions in TypeScript:

1. **Modularity and Reusability:** Functions allow you to break down your code into smaller, manageable pieces. These pieces can be reused in different parts of your program, improving code organization and maintainability.

2. **Abstraction**: Functions abstract away the implementation details, allowing you to focus on what the function does rather than how it does it. This makes your code more readable and easier to understand.

3. **Encapsulation**: Functions help encapsulate logic, meaning that the internal workings of a function are hidden from the outside world. This promotes a clear separation of concerns in your code.

4. **Parameterization**: Functions can take parameters, allowing you to make your code more flexible and customizable. Parameters allow you to pass different values to a function, making it adaptable to various scenarios.

5. **Return Values**: Functions can return values, allowing them to produce results that can be used elsewhere in your code. This is crucial for creating dynamic and responsive programs.