Conditional Statements in TypeScript

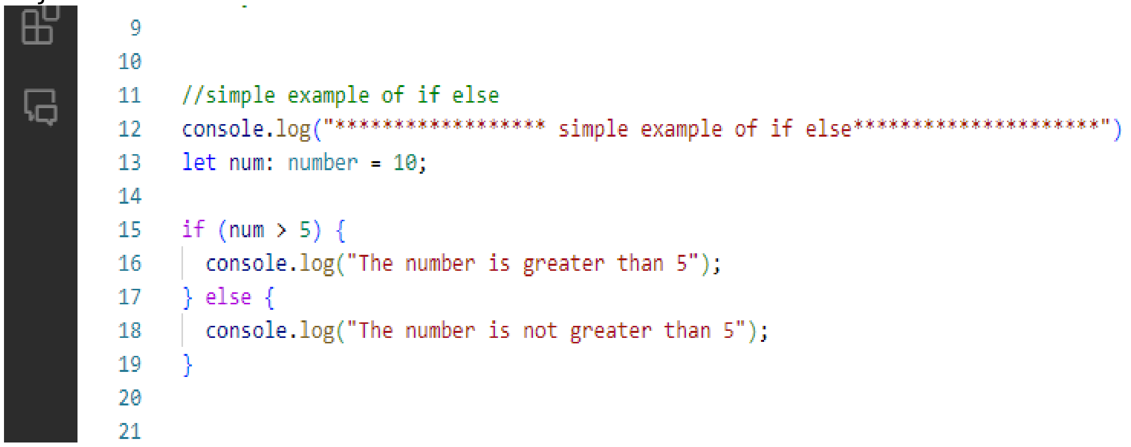
The if-else statement is used for conditional execution of code. It allows you to specify a block of code **(if block)** that should be executed if a given condition is true, and an alternative block **(else block)** of code to be executed if the condition is false.

The purpose of using if-else statements, or conditional statements in general, is to control the flow of a program based on certain conditions. This enables your program to make decisions and execute different sets of instructions depending on the values or states of variables.

Common use cases for if-else statements include:

- **Validation**: Checking if user inputs are valid before processing them.
- **Control Flow**: Executing different code paths based on certain conditions.
- **Error Handling**: Handling exceptional cases or error conditions.
- **Switching Between Options**: Providing different behavior based on the state of your application.

```
//The basic syntax of the if-else statement in TypeScript is as follows:
      if (condition) {
          // Code to be executed if the condition is true
        } else {
          // Code to be executed if the condition is false
        }
```

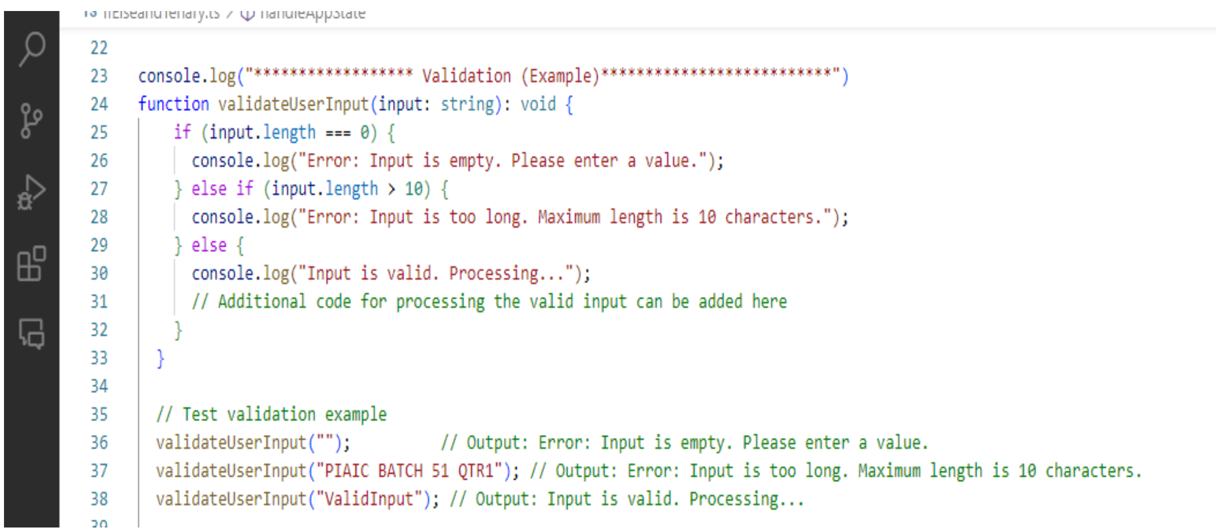


```


//simple example of if else
console.log("***************** simple example of if else********************")
let num: number = 10;

if (num > 5) {
  console.log("The number is greater than 5");
} else {
  console.log("The number is not greater than 5");
}


```

Validation (Example)

Example of using an if-else statement for input validation in TypeScript:

This kind of input validation is commonly used in user interfaces and data processing to ensure that the input meets certain criteria before further processing or execution.

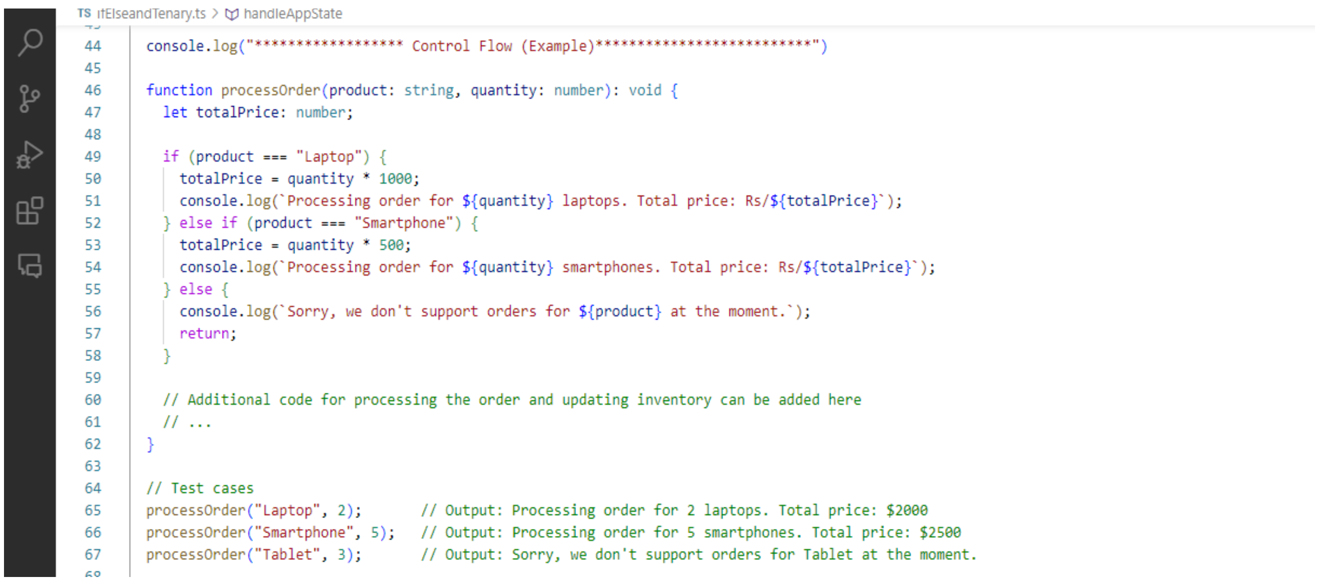


```
console.log("****************** Validation (Example)**************************")
function validateUserInput(input: string): void {
    if (input.length === 0) {
      console.log("Error: Input is empty. Please enter a value.");
    } else if (input.length > 10) {
      console.log("Error: Input is too long. Maximum length is 10 characters.");
    } else {
      console.log("Input is valid. Processing...");
      // Additional code for processing the valid input can be added here
    }
  }

  // Test validation example
  validateUserInput("");          // Output: Error: Input is empty. Please enter a value.
  validateUserInput("PIAIC BATCH 51 QTR1"); // Output: Error: Input is too long. Maximum length is 10 characters.
  validateUserInput("ValidInput"); // Output: Input is valid. Processing...
```

Control Flow (Example)

Example of executing different code paths based on certain conditions.

This kind of if-else structure allows for the execution of different code paths based on the value of the product variable, demonstrating control flow in the program.



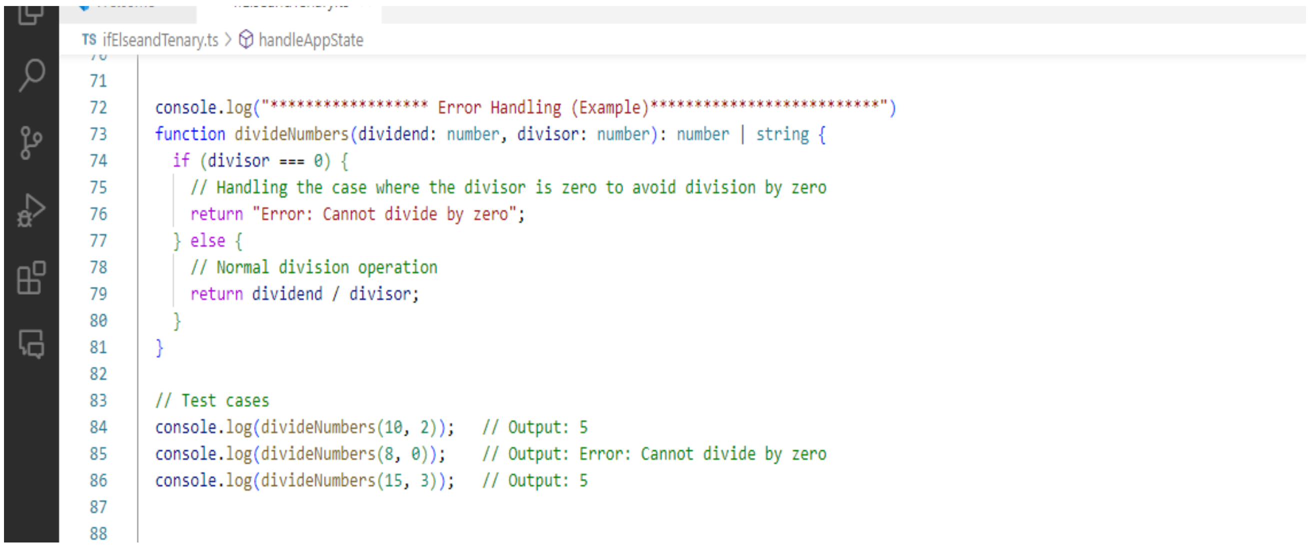
```
console.log("****************** Control Flow (Example)**************************")

function processOrder(product: string, quantity: number): void {
  let totalPrice: number;

  if (product === "Laptop") {
    totalPrice = quantity * 1000;
    console.log(`Processing order for ${quantity} laptops. Total price: Rs/${totalPrice}`);
  } else if (product === "Smartphone") {
    totalPrice = quantity * 500;
    console.log(`Processing order for ${quantity} smartphones. Total price: Rs/${totalPrice}`);
  } else {
    console.log(`Sorry, we don't support orders for ${product} at the moment.`);
    return;
  }

  // Additional code for processing the order and updating inventory can be added here
  // ...
}

// Test cases
processOrder("Laptop", 2);       // Output: Processing order for 2 laptops. Total price: $2000
processOrder("Smartphone", 5);   // Output: Processing order for 5 smartphones. Total price: $2500
processOrder("Tablet", 3);       // Output: Sorry, we don't support orders for Tablet at the moment.
```

Error Handling (Example)

Example of handling exceptional cases or error conditions.

This is a basic example of error handling, where the program checks for a specific condition that might lead to an error and responds appropriately, providing an error message instead of allowing the division by zero operation to proceed.

Error handling is essential for ensuring that programs can gracefully handle unexpected or exceptional conditions, preventing crashes and providing meaningful feedback to users or developers.

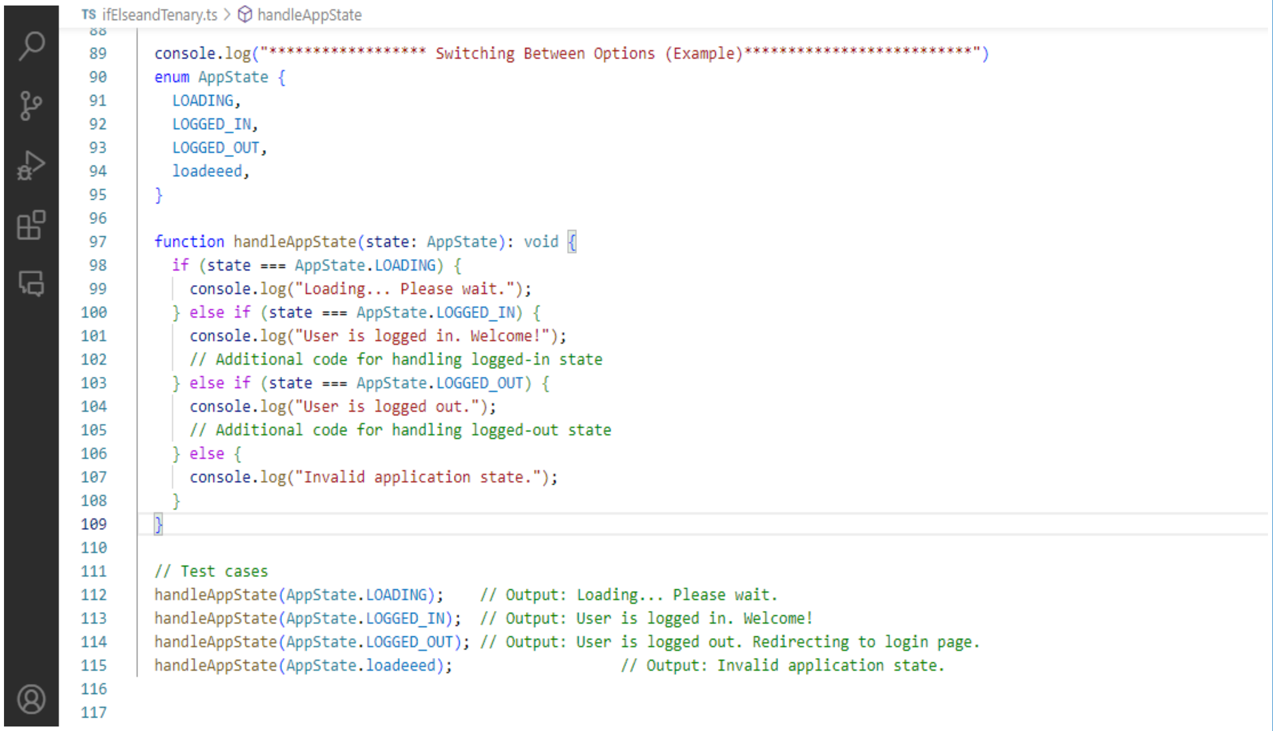


```
console.log("****************** Error Handling (Example)**************************")
function divideNumbers(dividend: number, divisor: number): number | string {
  if (divisor === 0) {
    // Handling the case where the divisor is zero to avoid division by zero
    return "Error: Cannot divide by zero";
  } else {
    // Normal division operation
    return dividend / divisor;
  }
}

// Test cases
console.log(divideNumbers(10, 2));   // Output: 5
console.log(divideNumbers(8, 0));    // Output: Error: Cannot divide by zero
console.log(divideNumbers(15, 3));   // Output: 5
```

Switching Between Options (Example):

Example of providing different behavior based on the state of your application.

This kind of structure allows you to define different behaviors or actions based on the state of your application, making it easy to switch between options and handle various scenarios. The use of an enum ensures that you are working with well-defined and self-documenting state values.

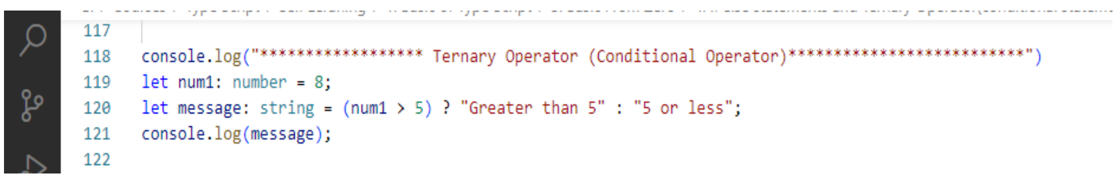


TS ifElseandTenary.ts > handleAppState

```
console.log("***************** Switching Between Options (Example)**************************")
enum AppState {
  LOADING,
  LOGGED_IN,
  LOGGED_OUT,
  loadeeed,
}

function handleAppState(state: AppState): void {
  if (state === AppState.LOADING) {
    console.log("Loading... Please wait.");
  } else if (state === AppState.LOGGED_IN) {
    console.log("User is logged in. Welcome!");
    // Additional code for handling logged-in state
  } else if (state === AppState.LOGGED_OUT) {
    console.log("User is logged out.");
    // Additional code for handling logged-out state
  } else {
    console.log("Invalid application state.");
  }
}

// Test cases
handleAppState(AppState.LOADING);    // Output: Loading... Please wait.
handleAppState(AppState.LOGGED_IN);  // Output: User is logged in. Welcome!
handleAppState(AppState.LOGGED_OUT); // Output: User is logged out. Redirecting to login page.
handleAppState(AppState.loadeeed);                  // Output: Invalid application state.
```

Ternary Operator (Conditional Operator):

The ternary operator is a shorthand way to write an if-else statement in a single line.



```
console.log("***************** Ternary Operator (Conditional Operator)**************************")
let num1: number = 8;
let message: string = (num1 > 5) ? "Greater than 5" : "5 or less";
console.log(message);
```