

# Understanding TypeScript Enums

**A Comprehensive Guide to Enumerations in  
TypeScript**

**Presented by Parizah Shaikh With  
co-ordination of M.Anees,Aleeze,Tayaba,Suman**


# History of enums

- So enums were not available in javascript.

Enums were introduced in TypeScript with version **2.0**, which was released in **April 2016**.



```
const ProcessStatuses = Object.freeze({  
  STARTED: 1,  
  IN_PROGRESS: 2,  
  FINISHED: 3  
});
```

 enum.ts > ...

1    //

2

3

4

5    const obj1 = {  
6      |     name: 'object',  
7    }

8

9    let result:string = obj1.name + true

10   console.log(result)    //    ans: objecttrue

11

12

ts enum.ts > ...

```
1  //
2
3
4
5  enum obj1 {           //its string enum
6      |   name = 'object'
7  }
8
9  let result:obj1 = obj1.name
10 result = result + 23
11 console.log(result)    //   ans: cause error due to assigning number
12
```

# what is an Enum?

- **Explanation:**
- **Definition:** Enums (short for "enumerations") are a feature in TypeScript that allow you to define a set of named constants. Enums make it easier to work with sets of related values by giving them meaningful names. They are useful when you need a predefined list of values that represent some type of category or state.

# Automatic Assignment of Numeric Values:

- In TypeScript, when enum constants are not explicitly assigned numeric values, they are automatically assigned incremental numeric values starting from 0. The default numeric value for the first enum constant is 0, and subsequent enum constants receive values incremented by 1.

# EXAMPLE

- `// Enum will initialize the first value 0 and add 1 to each additional value`
- `enum Colors { Red= 100, Blue, Black, Pink, Purple, Orange, Green, White}`
- `let favoriteColor :Colors = Colors.Black`
- `console.log(favoriteColor); // return index 102`
- `let favoritecolorName = Colors[105]`
- `console.log(favoritecolorName); // orange`



# Special Class for Constants:

- An Enum is a special 'class' that represents a group of constants (unchangeable variables). Enums come in two flavors: string and numeric.
- **Enums as Sets of Values:**
- Enum is the set of values. Enums return indexes or defined indexes or return values.

# Simple Examples:

- **Numeric Enums:** Numeric enums are the default type of enums in TypeScript. They are backed by numeric values, which can either be specified explicitly or auto-incremented by default.
- `enum Direction { Up, Down, Left, Right}`
- In this example, `Direction.Up` will have the value 0, `Direction.Down` will be 1, `Direction.Left` will be 2, and `Direction.Right` will be 3.
- You can also explicitly set the values:
- `enum Direction { Up = 1, Down = 2, Left = 3, Right = 4}`
- **String Enums:** String enums allow you to assign string values to the enum members.
- `enum Direction { Up = "UP", Down = "DOWN", Left = "LEFT", Right = "RIGHT"}`
- Here, `Direction.Up` will be "UP", `Direction.Down` will be "DOWN", `Direction.Left` will be "LEFT", and `Direction.Right` will be "RIGHT".

# Usage Example:

- Enums are often used in situations where you have a set of related constants. For example, you might use an enum to represent the directions in which a character can move in a game:
- ```
function move(direction: Direction) {
```
- ```
  switch (direction) {
```
- ```
    case Direction.Up:
```
- ```
      console.log("Moving up");
```
- ```
      break;
```
- ```
    case Direction.Down:
```
- ```
      console.log("Moving down");
```
- ```
      break;    case Direction.Left:
```
- ```
        console.log("Moving left");
```
- ```
        break;    case Direction.Right:
```
- ```
      console.log("Moving right");
```
- ```
      break;  }
```
- ```
move(Direction.Up); // Output: Moving up
```
- In this example, the move function takes a Direction enum as an argument and performs different actions based on the value of the enum. This makes the code more readable and less error-prone compared to using plain strings or numbers.

# Conclusion

- **Summary of Key Points:**
- **Enums Overview:** Enums are a TypeScript feature that allows you to define a set of named constants, making your code more readable and maintainable.
- **Types of Enums:**
- **Numeric Enums:** Use numeric values, which can be auto-incremented or explicitly set.
- **String Enums:** Use string values, providing a clearer representation for debugging and logging.
- **Using Enums:**
- Enums can be used in functions, switch cases, and as types to ensure type safety. They improve code readability by replacing magic numbers and strings with meaningful names.
- **Best Practices:** Use enums to define sets of related constants. Avoid overusing enums; consider alternative structures if the set of values is likely to change.

# Invitation for Questions and Answers:

Thank you for your attention. I hope this presentation has given you a clear understanding of TypeScript enums, their benefits, and how to use them effectively.

Now, I'd like to open the floor for any questions you may have. Feel free to ask about any specific aspects of enums, their usage, or any other related topics. Your questions are welcome!