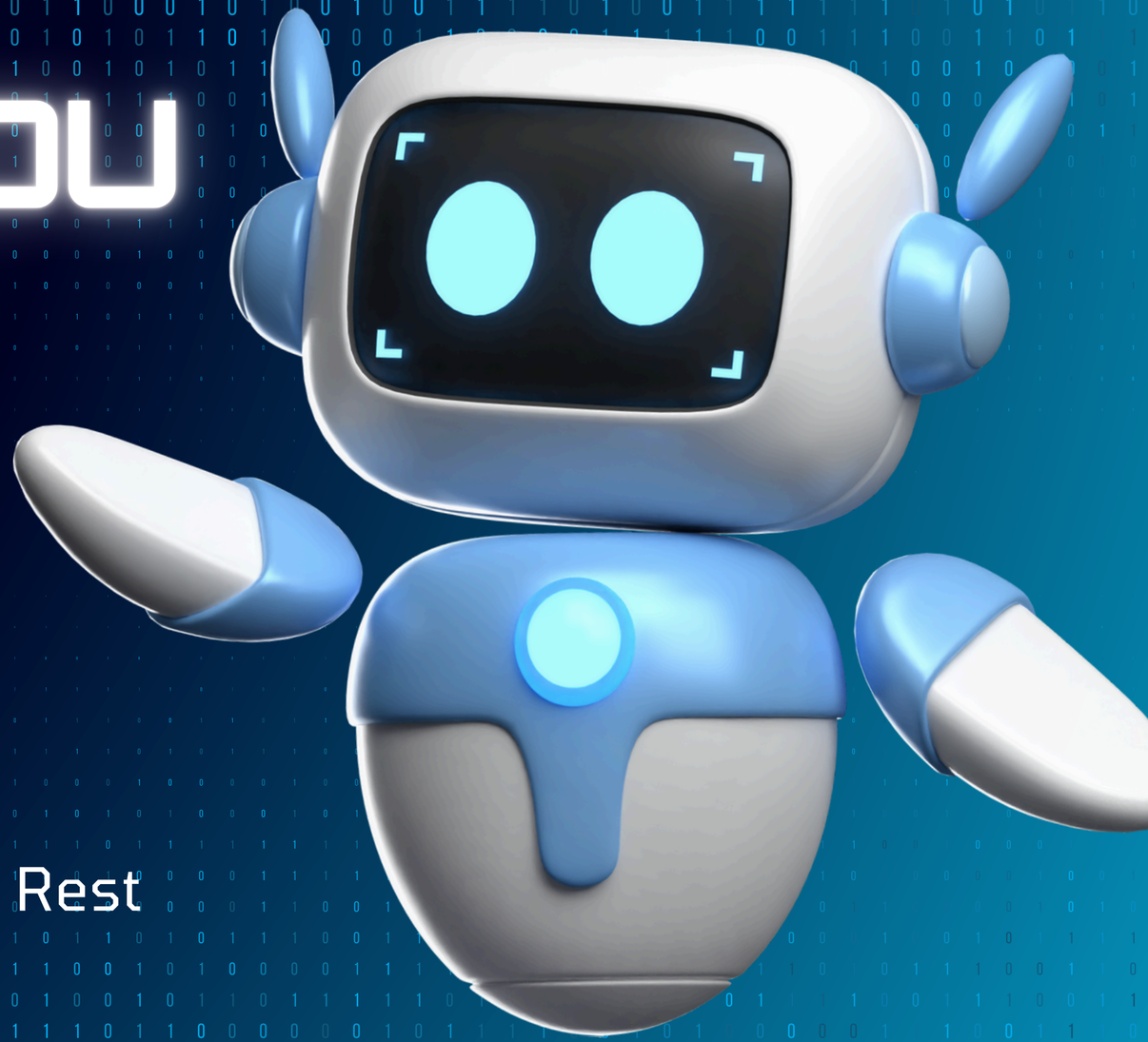
DEFINITION IN ROMAN URDU

1. const , let
2. Modules
3. Enum , const enum
4. ECMAScript Modules
5. Function optional Parameter , default & Rest
6. And Much More



Design By : *Ayan Sheikh*

1 CONST , LET

const aur let TypeScript mein variables declare karne ke tareeqe hain. const immutable variable create karta hai jo ek bar initialize hone ke baad change nahi ho sakta. let block-scope variable banata hai jo sirf us block ke andar accessible hota hai jahan declare kiya gaya ho.

EXAMPLE:

```
const pi = 3.14;
let count = 0
```

2 MODULES

Modules TypeScript mein code ko alag alag files mein organize karne ke liye use hote hain. Yeh code reusability aur maintainability ko improve karte hain.

EXAMPLE:

```
export class MyClass { }
```

3 ENUM & CONST ENUM :

ENUM :

Enum TypeScript mein aik special data type hai jo aik group of related constants ko define karta hai. Yeh readable aur manageable code likhne mein madad deta hai.

EXAMPLE:

```
enum Color { Red, Green, Blue } let c: Color = Color.Green;
```

CONST ENUM :

Const enum TypeScript mein compile time constants ko define karta hai. Iska fayda yeh hai ke runtime pe yeh kisi bhi additional code ko generate nahi karta.

EXAMPLE:

```
const enum Direction { Up, Down, Left, Right } let dir: Direction = Direction.Up;
```

4 ECMASCRIPT MODULES :

ECMA Script Modules standardized modules hain jo JavaScript mein use hote hain. Yeh modules import/export keywords ko use karte hain.

EXAMPLE:

```
import { MyClass } from './myModule';
let count = 0
```

5 FUNCTION PARAMETER

REST PARAMETER:

Rest parameters TypeScript mein variable number of arguments ko handle karne ke liye use hote hain. Yeh parameters ko array mein convert kar dete hain.

EXAMPLE:

```
function sum(...numbers: number[]): number { return numbers.reduce((a, b) => a + b, 0); }
```

DEFAULT PARAMETER:

Default parameters TypeScript mein woh parameters hain jo function definition mein default value ke sath define hote hain. Agar value na mile to default use hoti hai.

EXAMPLE:

```
function buildName(firstName: string, lastName = 'Smith') { return firstName + ' ' + lastName; }
```

OPTIONAL PARAMETER:

Optional parameters TypeScript mein woh parameters hain jo function call karte waqt zaroori nahi hote. Yeh parameters ko '?' symbol se denote karte hain.

EXAMPLE:

```
function buildName(firstName: string, lastName?: string) { return firstName + ' ' + (lastName)}
```

6 FUNCTIONS

Functions TypeScript mein code ke reusable blocks hote hain jo ek specific task ko perform karte hain. Jab bhi humein ek task ko baar baar perform karna ho, us task ko function mein define kar ke baar baar use kar sakte hain. Function ko define karte waqt, humein parameters (jo optional bhi ho sakte hain) aur return type specify karna hota hai.

EXAMPLE:

```typescript
// Example of a function that adds two numbers
function add(x: number, y: number): number {
    return x + y;
}

// Calling the add function
let result = add(5, 10);
console.log(result); // Output: 15
```

7 ASYNC

Async functions TypeScript mein asynchronous operations ko handle karne ke liye use hoti hain. Yeh functions 'async' keyword se define hoti hain aur 'await' keyword se asynchronous operations ko pause kar sakti hain.

EXAMPLE:

```
async function fetchData() { let response = await fetch('url');
let data = await response.json(); console.log(data); }
```

8 FUNCTION OVERLOADS

Function overloads se murad ek hi naam ke functions hain jo mukhtalif parameter types ya numbers ko accept karte hain. Matlab aap ek function ke mukhtalif versions bana sakte hain jo alag alag tarike se call ho sakte hain depending on the parameters.

EXAMPLE:

```
function add(a: number, b: number): number;
function add(a: string, b: string): string;
function add(a: any, b: any): any {
  return a + b;
}

console.log(add(5, 10));         // Output: 15
console.log(add('Hello, ', 'World!')); // Output: 'Hello,
```

9 TUPLES

Tuples TypeScript mein ek data structure hai jo kisi bhi number of elements ko ek fixed order mein hold karta hai. Har element ko specific type ka hona zaroori hai, lekin har element ka type alag alag ho sakta hai. Tuples ko mainly use karne ke liye jab humein ek fixed number of elements ki specific order mein data store karna ho, jaise ke kisi function se multiple values return karte waqt.

Tuples ek tarah ki array hai jisme har element ka specific type fixed hota hai aur unka order bhi important hota hai. Har tuple ka length fixed hota hai, isliye humein tuple ko declare karte waqt uski length aur har element ka type specify karna zaroori hota hai. Tuples useful hote hain jab hum fixed order mein multiple types ke data ko store aur access karna chahte hain.

EXAMPLE:

```typescript
// Declaring a tuple to store a person's name and age
let person: [string, number] = ['John', 25];

// Accessing elements of the tuple
console.log(person[0]); // Output: 'John'
console.log(person[1]); // Output: 25

// Incorrect usage (TypeScript error)
// person = [25, 'John']; // Error: Type 'number' is not assignable to type 'string'

// Correct usage (same types and order)
person = ['Alice', 30]; // Valid assignment
```

10 INTERFACE

Interface TypeScript mein ek tareeqa hai jis se dusre classes ya objects ke liye ek contract ya shape define ki jati hai. Is se object ki structure describe hoti hai jis mein uske properties aur optional methods ke naam aur types specify kiye jate hain. Interfaces objects banane ke liye blueprint provide karte hain jo specific guidelines ko follow karna zaroori hota hai.

EXAMPLE:

- **Interface Person**: Ek structure define kiya gaya hai jisme properties firstName, lastName, aur age hain, sabhi ke liye specific types (string aur number). Isme ek optional method getFullName() bhi hai jo ek string return karta hai.
- **Class Employee**: Person interface ko implement karta hai, matlab is class mein Person interface mein define kiye gaye sabhi properties aur methods include hone zaroori hain. Employee class mein yeh properties aur methods ka concrete implementation provide kiya gaya hai.
- **Object Creation** (emp): Employee class ka ek object emp new keyword se create kiya gaya hai, jo Person interface mein define ki gayi structure ko follow karta hai.
- **Accessing Properties and Method**: Interface mein define ki gayi properties (firstName) aur method (getFullName()) class mein implement ki gayi hain aur unhe as expected access kiya ja sakta hai.

```typescript
// Example of an interface defining a Person
interface Person {
    firstName: string;
    lastName: string;
    age: number;

    // Optional method
    getFullName(): string;
}

// Implementing the Person interface in a class
class Employee implements Person {
    firstName: string;
    lastName: string;
    age: number;

    constructor(firstName: string, lastName: string, age: number) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.age = age;
    }

    getFullName(): string {
        return `${this.firstName} ${this.lastName}`;
    }
}

// Creating an object using the Employee class
let emp: Employee = new Employee('John', 'Doe', 30);

// Accessing properties and calling method
console.log(emp.firstName);    // Output: John
console.log(emp.getFullName());  // Output: John Doe
```

11 INTERSECTION TYPES

Intersection types TypeScript mein aise types hote hain jo aapko mazeed flexibility dete hain multiple types ko combine karne mein. Jab aap do ya zyada types ko & (and) operator se combine karte hain, to resulting type mein har ek type ke properties aur methods shamil ho jate hain.

EXAMPLE:

- **Interfaces Printable aur Loggable:** Ye do separate interfaces hain jo print() aur log() methods ko define karte hain.
- **Intersection Type LoggableAndPrintable**: Ye aik intersection type hai jo Printable aur Loggable interfaces ko & operator se combine karta hai. Resulting type LoggableAndPrintable mein dono interfaces ke print() aur log() methods shamil hote hain.
- **Class ConsoleLogger**: LoggableAndPrintable intersection type ko implement karta hai, iska matlab ye hai ke is class mein print() aur log() dono methods ka implementation hona zaroori hai.
- **Object Creation (logger)**: ConsoleLogger class se LoggableAndPrintable type ka aik object logger banaya jata hai. Is object mein Printable aur Loggable interfaces mein define kiye gaye print() aur log() methods ko access aur invoke kiya ja sakta hai.

```typescript
// Intersection types ka misal
interface Printable {
    print(): void;
}

interface Loggable {
    log(): void;
}

// Intersection type jo Printable aur Loggable ko combine karta hai
type LoggableAndPrintable = Printable & Loggable;

// Intersection type ko aik class mein implement karna
class ConsoleLogger implements LoggableAndPrintable {
    print(): void {
        console.log('Printing...');
    }

    log(): void {
        console.log('Logging...');
    }
}

// ConsoleLogger class se object bananay ka tareeka
let logger: LoggableAndPrintable = new ConsoleLogger();

// Dono interfaces ke methods ko call karna
logger.print(); // Output: Printing...
logger.log();   // Output: Logging...
```

Thank You

Follow Our’s Accounts

 Ayan Sheikh

 03198187639

 a_y_a_n__sheikh