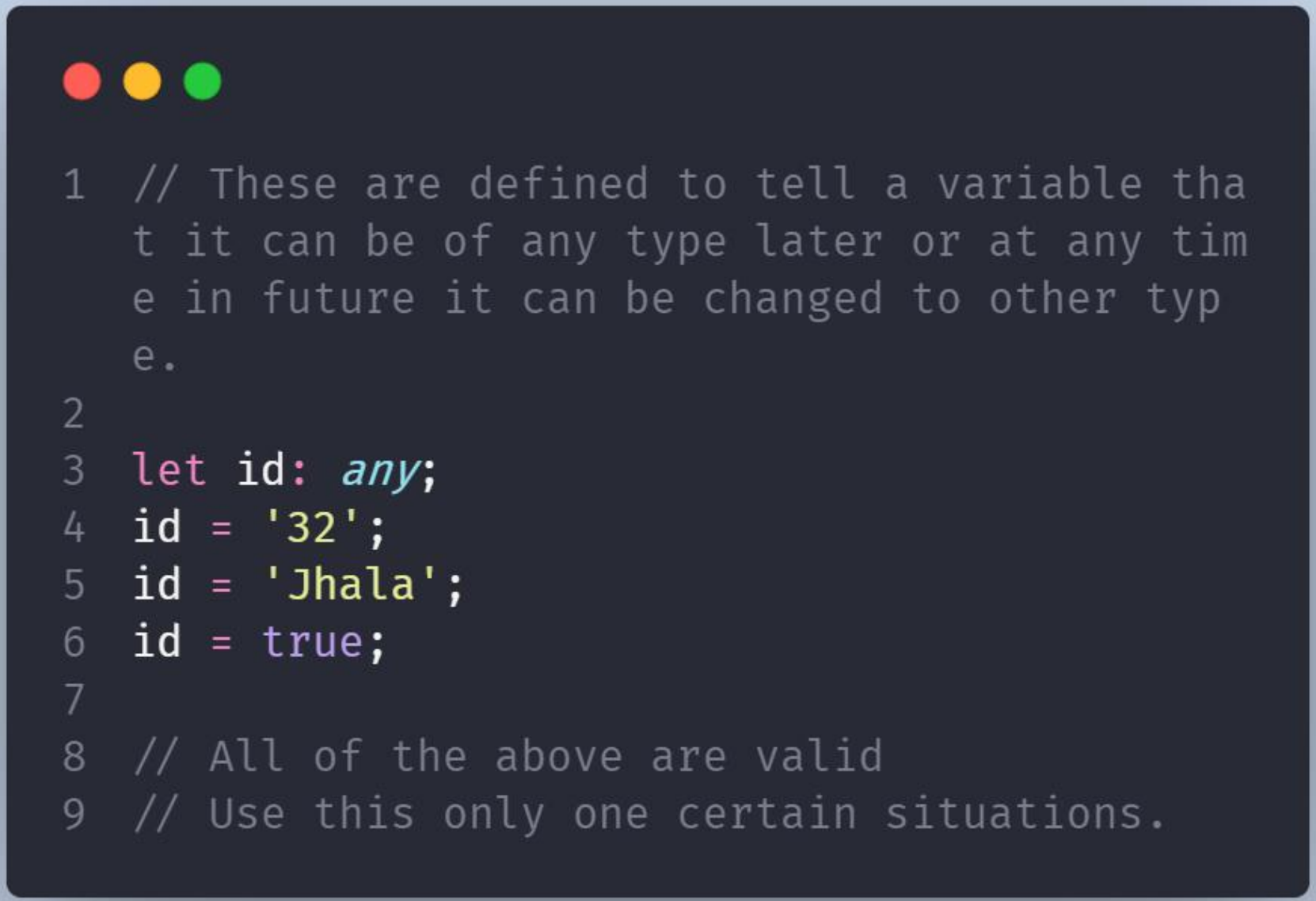
```ts
// What is TypeScript:
// 👉 Superset of JavaScript
// 👉 Allows to use StrictTypes
// 👉 Supports Modern Js functions like Arrow functions, let, const etc.
// 👉 Extra features like Generics, Interfaces, Tuples etc.

// To install TypeScript compiler run the following command:
// 👉 npm (install or i) -g typescript



const coderName = 'DEVRAJ';
console.log(coderName);

// This is a typeScript file hence browser wont run it we will have to compile it to JavaScript.

// If file names are same just type tsc (fileName).
// Use tsc main.ts -w w means watch so everytime we make changes to ts it will be automatically reflected in js.

// In TypeScript once variable is declared once it's type cannot be changed otherwise.

const circumference = (diameter : number) => {
    return diameter * Math.PI;
}

console.log(circumference(4));
// console.log(circumference('hello'));
// diameter : number => you can define the type so that function does not take any other input like string.


```

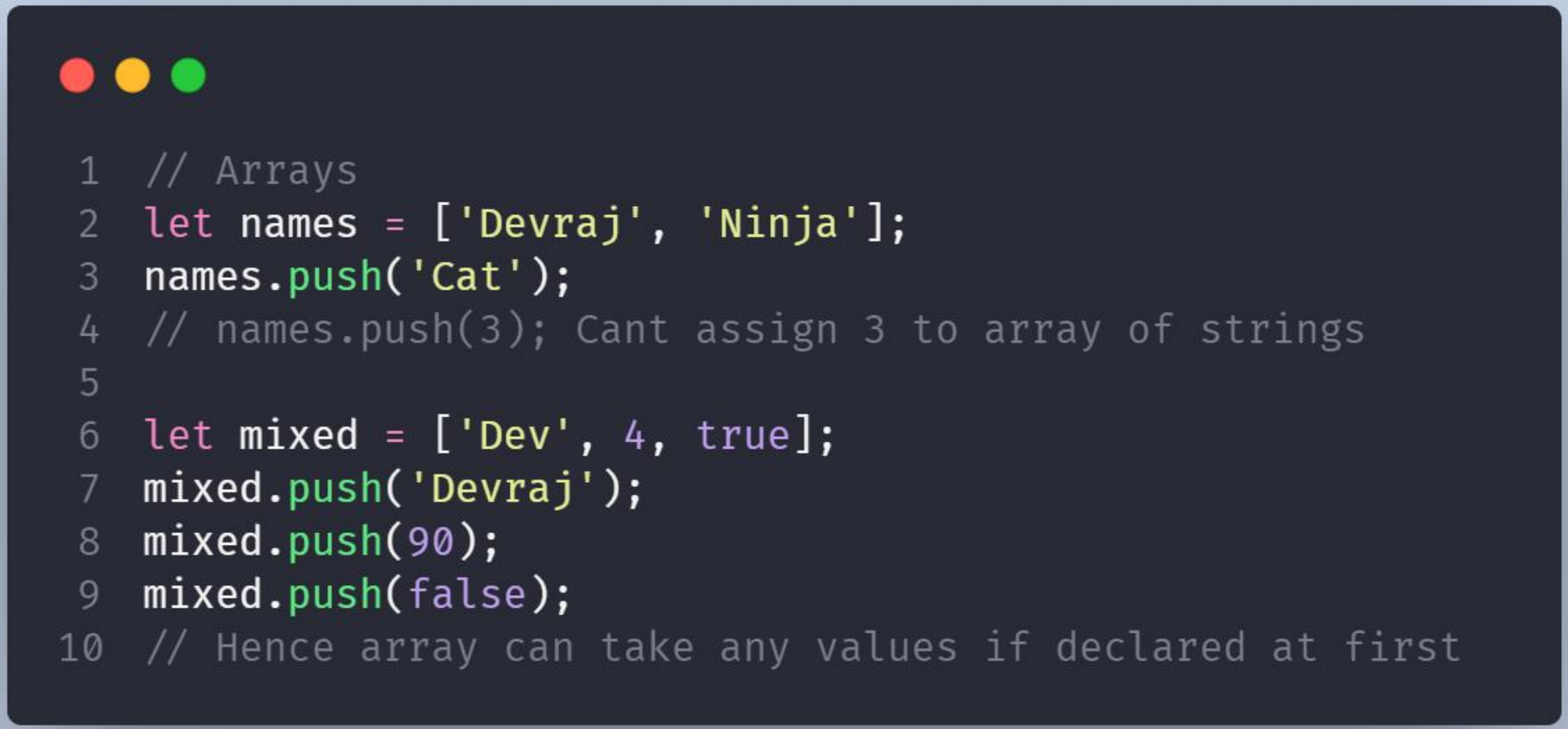
```ts
// Explicit types
let character : string;
let age : number;

age = 30;
// age = 'Dev' Error!


// Arrays
let devJob: string[] = [];
devJob = ['Frontend', 'Backend'];
// It's good practice to declare the array as empty initially otherwise it will give errors on .push() method.


// Union Types
let mixed: (string|number|boolean)[] = [];
mixed.push(20);
mixed.push(false);

let id:  string|number;
id = 'Devraj';
id = 40;


// Objects
let ninjaOne: object;
ninjaOne = {name: 'yoshi', age: 30}

```



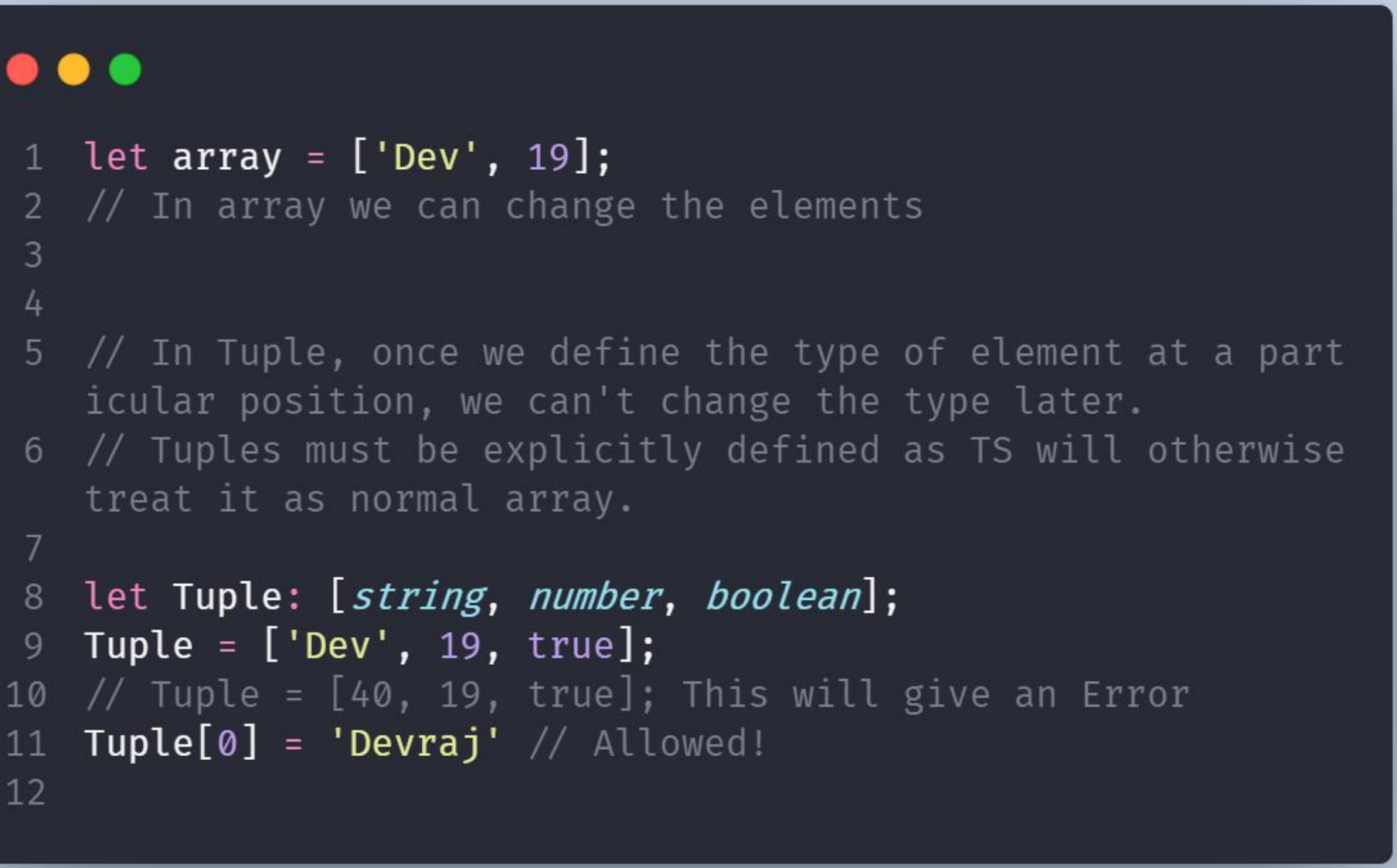
```
// These are defined to tell a variable that it can be of any type later or at any time in future it can be changed to other type.

let id: any;
id = '32';
id = 'Jhala';
id = true;

// All of the above are valid
// Use this only one certain situations.
```

```

let GM = () => {
    console.log("Grand Master");
}

// GM = 'Hello' Will give error because it recognizes that it is of type function

let bookName: Function;
bookName = () => {
    console.log("Do Epic Shit");
}

const subtract = (a: number, b: number, c?: number) => {
    console.log(a - b);
}

subtract(5,10);
// Here I have not passed 3rd argument hence it's giving error so we can set c as optional parameter.
// Adding ? means it's an optional parameter
// Always take optional parameters at the end.



// Type Aliases
let a: number|string;
let b: number|string;

// Here instead of using number|string everytime we can define alias so now:
type StringorNum = string|number;
let c: StringorNum;
// This helps in reducing code duplicity

```

```typescript
// Function Signature represents the flow or structure of the function
// let greet: Function;

// Example of the signature
// Here a,b are parameters and void is the return type
let greet: (a: string, b:string) => void;
greet = (name: string, greeting: string) => {
    console.log(`Hello ${name} says ${greeting}`);
}

// Example 2
let calc: (a: number, b: number, c:string) => number;
calc = (num1: number, num2: number, action:string) => {
    if(action === 'add'){
        return num1 + num2;
    }else{
        return num1 - num2;
    }
}
```



```
// Arrays
let names = ['Devraj', 'Ninja'];
names.push('Cat');
// names.push(3); Cant assign 3 to array of strings

let mixed = ['Dev', 4, true];
mixed.push('Devraj');
mixed.push(90);
mixed.push(false);
// Hence array can take any values if declared at first
```



```ts
let array = ['Dev', 19];
// In array we can change the elements


// In Tuple, once we define the type of element at a particular position, we can't change the type later.
// Tuples must be explicitly defined as TS will otherwise treat it as normal array.

let Tuple: [string, number, boolean];
Tuple = ['Dev', 19, true];
// Tuple = [40, 19, true]; This will give an Error
Tuple[0] = 'Devraj' // Allowed!

```

```
// Objects
let skills = {
    name: 'VS Code',
    level: 'Intermediate',
    theme: 'One Dark Pro',
};
// These attributes also behave as variables so once its
 type is declared it cannot be changed later.

skills.name = 'ReactJS'
// skills.name = 30; You can't do this
// You can override the object but number of attributes m
ust be same
skills={
    name: "NodeJs",
    level: "Begineer",
    theme: "Backend",
}
// skills={
//     name: "NodeJs",
//     level: "Begineer",
// } This will give error
```

```ts
// Classes are the blueprints for the object
// All properties of the class are public by default

class Business {

// readonly means we can just read and cannot change it
    readonly client: string;

// private will not allow property to change outside the class
    private cashIncome: number;
    expenditure: number;

    constructor(client: string, cashIncome: number, expenditure: number){
        this.client = client;
        this.cashIncome = cashIncome;
        this.expenditure = expenditure;
    }

    // Method
    Summary(){
        return `${this.client} has given ${this.cashIncome} rupees and Expenditure was ${this.expenditure}`;
    }
}

const business1 = new Business('Devraj', 25000, 5000);

// The below syntax means that only the objects created by Business class can be added to this array.
let allBusiness: Business[] = [];
// allBusiness.push('hello') Error!

allBusiness.push(business1);
// above one valid
```

```typescript
// Interface allows us to enforce certain structure of class or objects.
// Allows to set structure to our objects and classes and hence makes it secure for us in the future.
interface Person {
    name: string;
    age: number;

    communicate(a: string): void;
    buy(a: number): number;

}

// Below is valid because it is complying with the structure of Interface, if we add or remove anything it will show an error.

const Devraj = {
    name: 'Devraj',
    age: 19,

    communicate(id: string): void {
        console.log(`${id} is sharing some thoughts!`);
    },

    buy(id: number): number {
        console.log(`Bought for ₹{id}`);
        return id;
    }
}



// Interface with Classes

interface Format{
    format(): string;
}

class Business2 implements Format {

// readonly means we can just read and cannot change it
    readonly client: string;

// private will not allow property to change outside the class
    private cashIncome: number;
    expenditure: number;

    constructor(client: string, cashIncome: number, expenditure: number){
        this.client = client;
        this.cashIncome = cashIncome;
        this.expenditure = expenditure;
    }

    // Method
    Summary(){
        return `${this.client} has given ${this.cashIncome} rupees and Expenditure was ${this.expenditure}`;
    }
    format(): string {
        return "I am using Prettier Code formatter"
    }
}
```

```
// Enums are special type in TS which allows to store a set of constants or keywords and associates them with a numeric value.
enum ResourceType {BOOK, AUTHOR, FILM}

interface Resource <T>{
    resourceType: ResourceType;
    data: T;
}

const R1: Resource<object> = {
    resourceType: ResourceType.BOOK,
    data: {title: 'Do Epic Shit'}
}
```