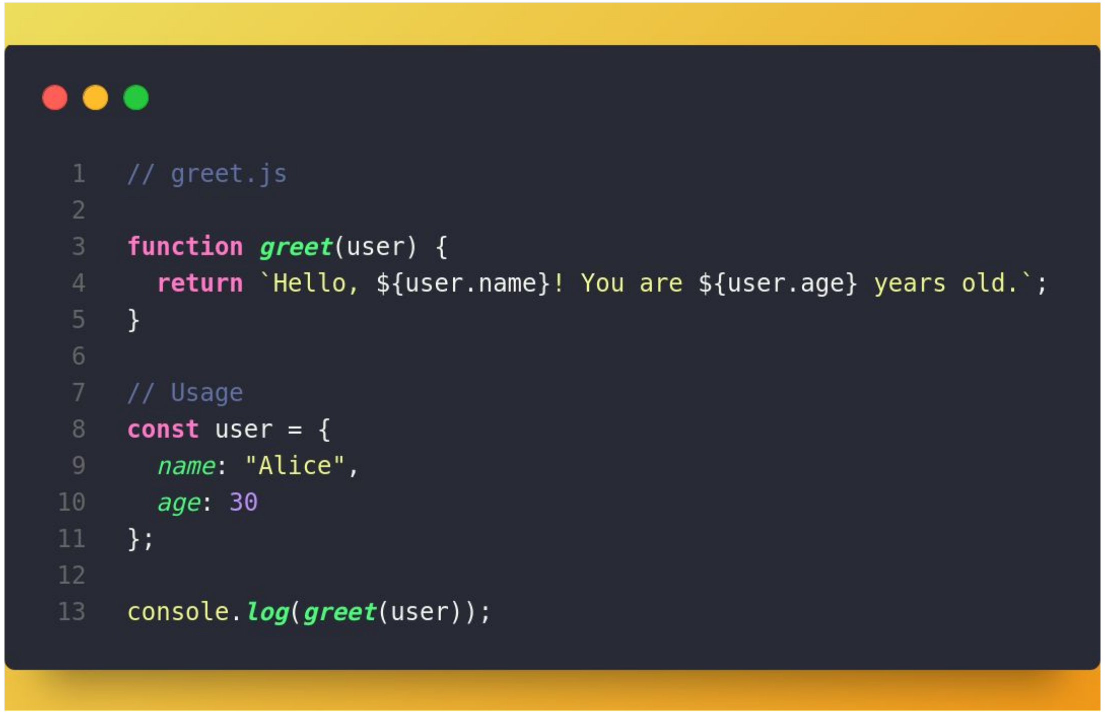
Typescript Essentials

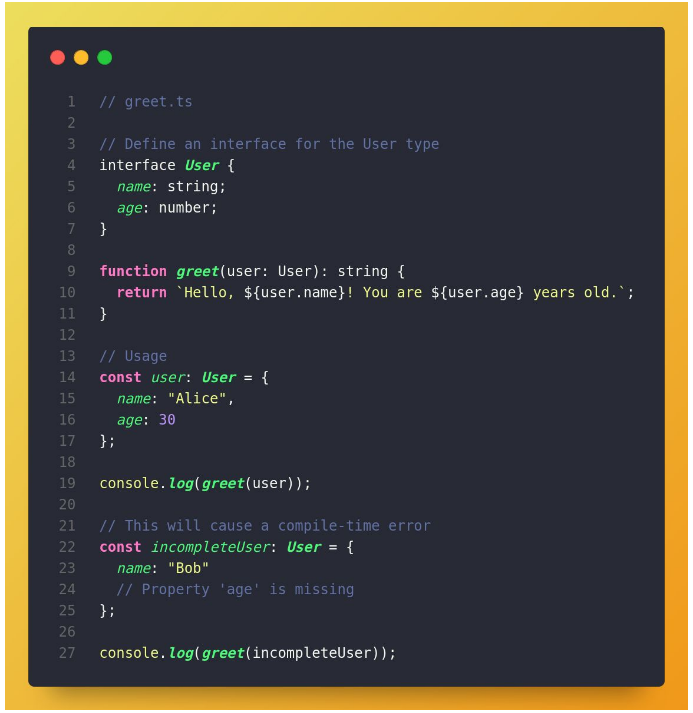
Quaid-i-Azam University
Department of Computer Science
WAF-CS457
By Mohammad Rashid Stanikzai

Outline

Introduction - Example



```javascript
// greet.js

function greet(user) {
  return `Hello, ${user.name}! You are ${user.age} years old.`;
}

// Usage
const user = {
  name: "Alice",
  age: 30
};

console.log(greet(user));
```



```typescript
// greet.ts

// Define an interface for the User type
interface User {
  name: string;
  age: number;
}

function greet(user: User): string {
  return `Hello, ${user.name}! You are ${user.age} years old.`;
}

// Usage
const user: User = {
  name: "Alice",
  age: 30
};

console.log(greet(user));

// This will cause a compile-time error
const incompleteUser: User = {
  name: "Bob"
  // Property 'age' is missing
};

console.log(greet(incompleteUser));
```

Introduction - Key Ideas

- It is JavaScript but with types.
- Improved code maintainability and scalability.
- Early error detection through type checking.
- It has to be compiled first to JavaScript using compiler and then it will run

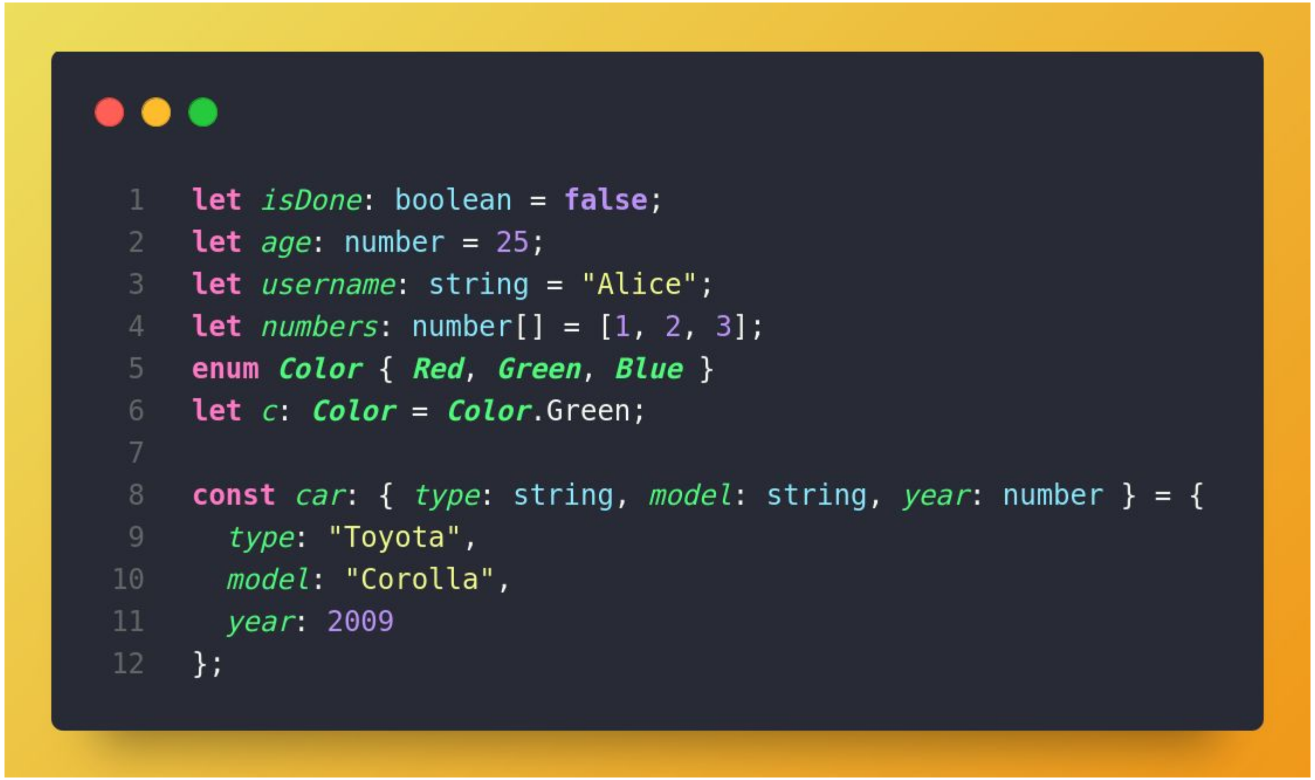
Setting Up the Development Environment

1. Initialize your project.
2. Install `typescript` package in your project.
3. Compile your code using `npx tsc <ts_filename>`
4. Now you can run your compiled TypeScript file which is now a simple JavaScript file and can be run like normal JavaScript file

Basic Types - Key Ideas

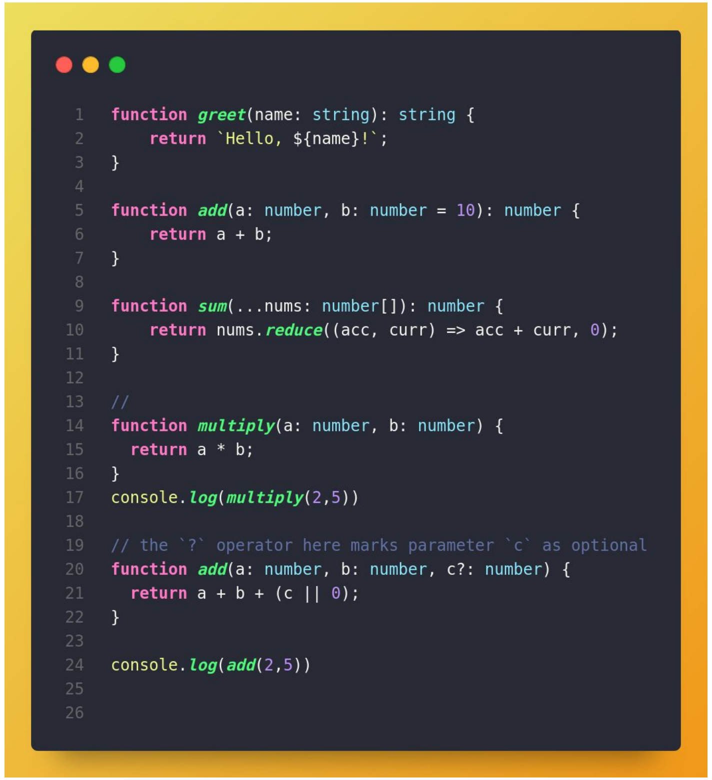
- Primitive Types
  - number, string, boolean
- Complex Types
  - Array, Objects Types
- Special Types
  - enum
- One type cannot be assigned to another type

Basic Types - Examples



```
let isDone: boolean = false;
let age: number = 25;
let username: string = "Alice";
let numbers: number[] = [1, 2, 3];
enum Color { Red, Green, Blue }
let c: Color = Color.Green;

const car: { type: string, model: string, year: number } = {
  type: "Toyota",
  model: "Corolla",
  year: 2009
};
```

Functions in TypeScript - Example

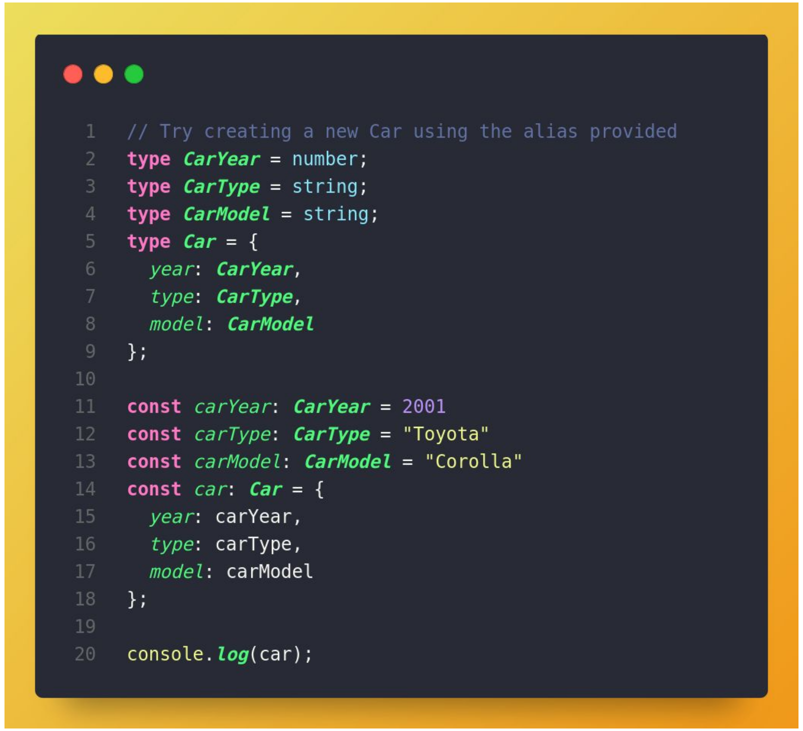


```typescript
function greet(name: string): string {
    return `Hello, ${name}!`;
}

function add(a: number, b: number = 10): number {
    return a + b;
}

function sum(...nums: number[]): number {
    return nums.reduce((acc, curr) => acc + curr, 0);
}

//
function multiply(a: number, b: number) {
  return a * b;
}
console.log(multiply(2,5))

// the `?` operator here marks parameter `c` as optional
function add(a: number, b: number, c?: number) {
  return a + b + (c || 0);
}

console.log(add(2,5))


```

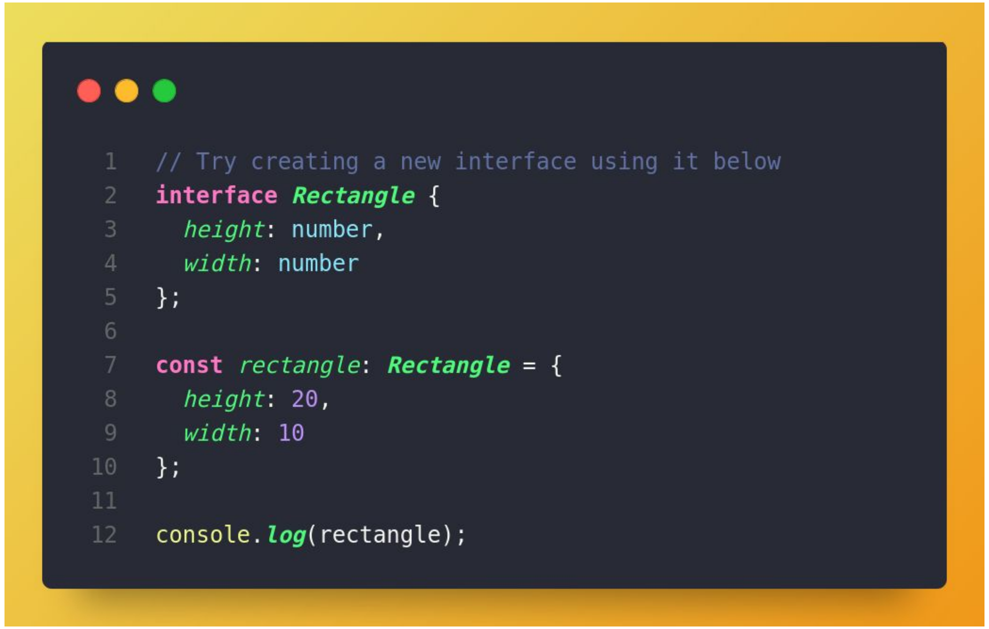
Basic Types - Key Ideas

- Every function must define types of its parameters
- Every function has to define its return type
- Every function must be passed an expecting type and every function that returns a value, has to be assigned to its compatible type

Interfaces and Type Aliases - Examples



```
// Try creating a new Car using the alias provided
type CarYear = number;
type CarType = string;
type CarModel = string;
type Car = {
  year: CarYear,
  type: CarType,
  model: CarModel
};

const carYear: CarYear = 2001
const carType: CarType = "Toyota"
const carModel: CarModel = "Corolla"
const car: Car = {
  year: carYear,
  type: carType,
  model: carModel
};

console.log(car);
```

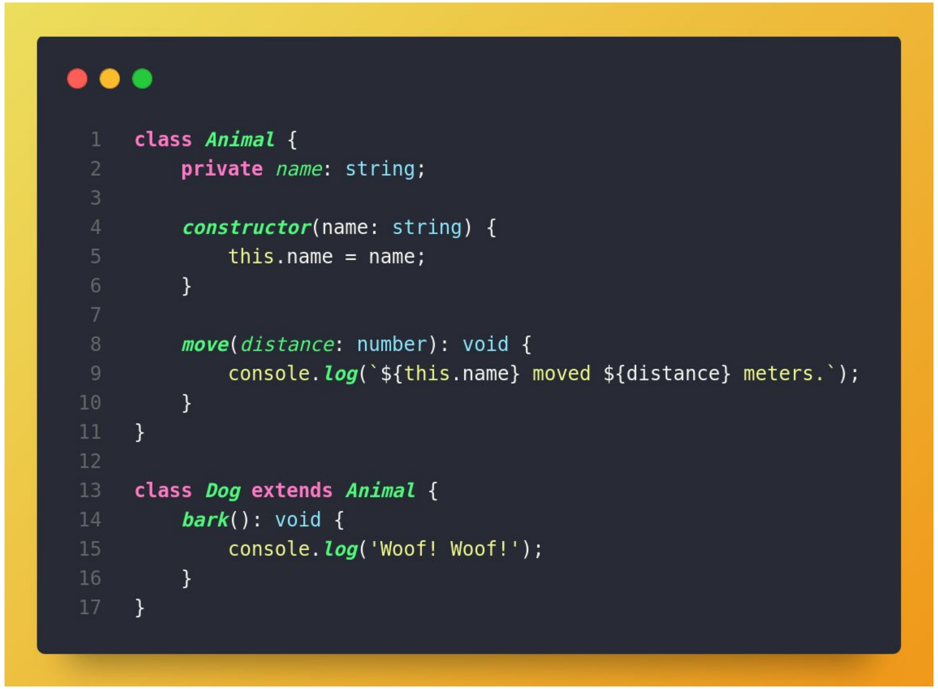


```
// Try creating a new interface using it below
interface Rectangle {
  height: number,
  width: number
};

const rectangle: Rectangle = {
  height: 20,
  width: 10
};

console.log(rectangle);
```

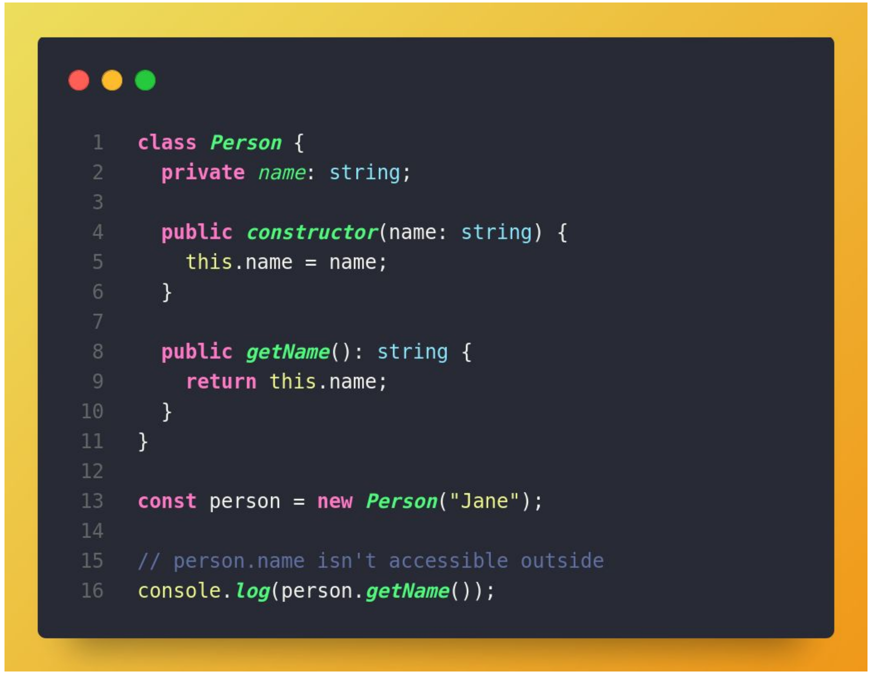
Interfaces and Type Aliases - Key Ideas

- Both Interfaces and Type Aliases allow us define new types
- We then create a variable that has our new defined custom type
- Type Aliases and Interfaces are the same except that Interfaces can be only used to object types

TypeScript Classes - Examples



```typescript
class Animal {
    private name: string;

    constructor(name: string) {
        this.name = name;
    }

    move(distance: number): void {
        console.log(`${this.name} moved ${distance} meters.`);
    }
}

class Dog extends Animal {
    bark(): void {
        console.log('Woof! Woof!');
    }
}
```

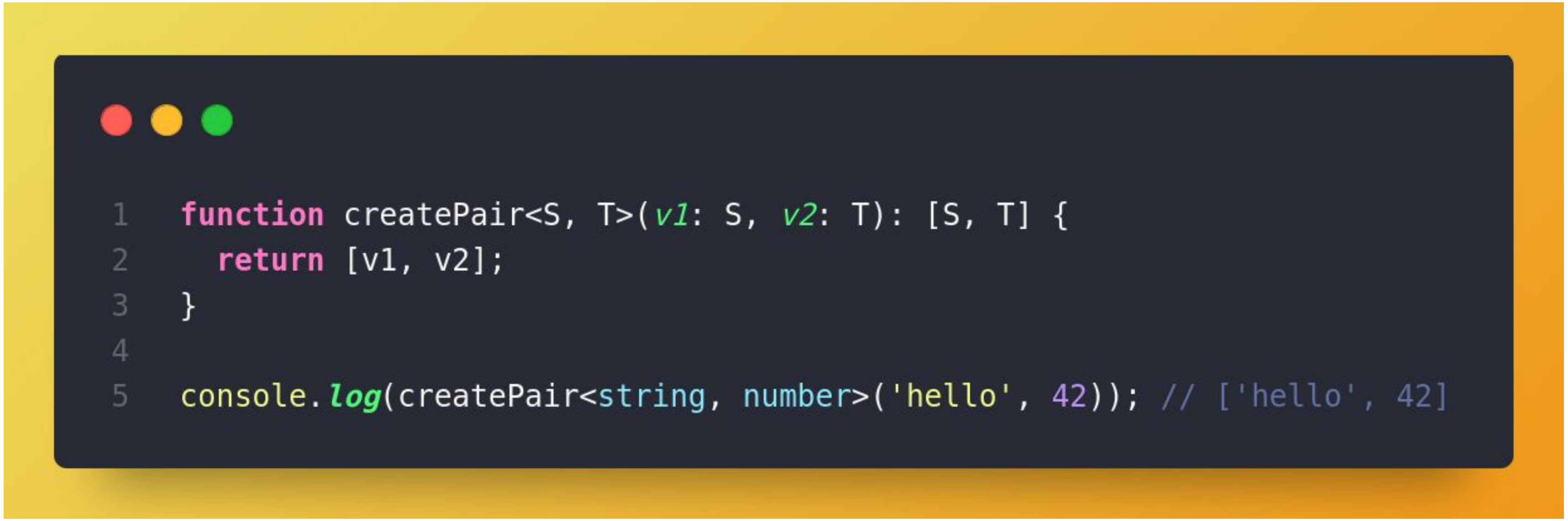


```typescript
class Person {
  private name: string;

  public constructor(name: string) {
    this.name = name;
  }

  public getName(): string {
    return this.name;
  }
}

const person = new Person("Jane");

// person.name isn't accessible outside
console.log(person.getName());
```

TypeScript Classes - Key Ideas

- We can use TypeScript to define and use classes
- Classes are the same as JavaScript, except types add type definition to class members and visibility modifier like private, protect and etc.
- All concept of Object Oriented programming can be applied to typescript classes, like polymorphism, inheritance and so on.

TypeScript Basic Generics - Example



```
function createPair<S, T>(v1: S, v2: T): [S, T] {
  return [v1, v2];
}

console.log(createPair<string, number>('hello', 42)); // ['hello', 42]
```

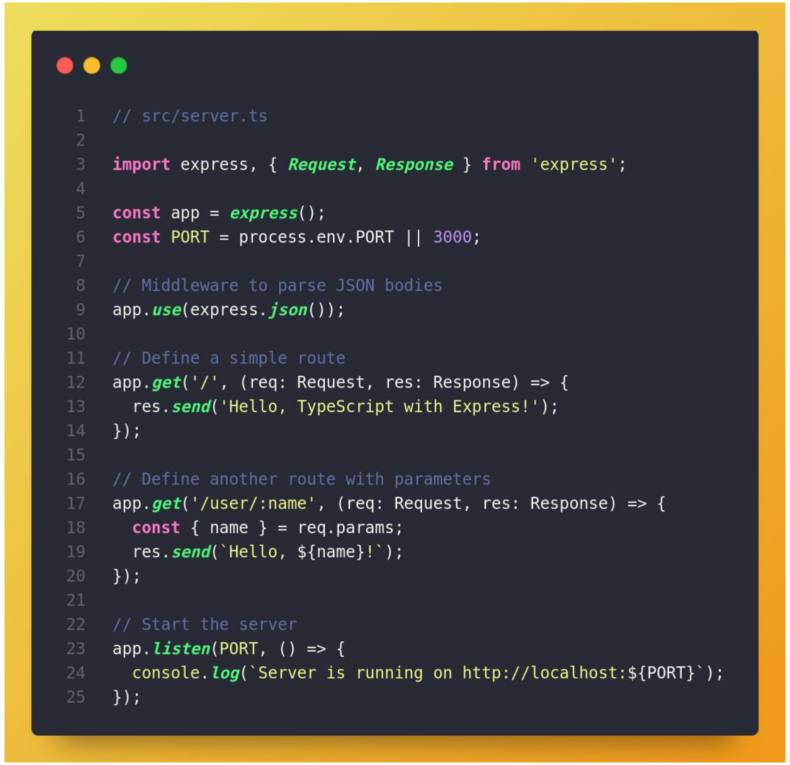
TypeScript Basic Generics - Key Ideas

- Generics makes it easier to write reusable code.
- Generics can be used to create variables, functions, classes and interfaces that can work with different types with same code
-

Recap of Key Points

- Typescript helps us write better and well structured code.
- TypeScript is a superset of JavaScript which means it is an addition to JavaScript that enables JavaScript to have types.
- Types helps us avoid most of the error during compilation that could occur in production environment.
- Once we wrote TypeScript we should use TypeScript compiler to make the code runnable.
- We need to install TypeScript compiler using npm for TypeScript projects.
- We can think of TypeScript completely identical to JavaScript except that it has Types in it

Basic Node Server in TypeScript - Example



```typescript
// src/server.ts

import express, { Request, Response } from 'express';

const app = express();
const PORT = process.env.PORT || 3000;

// Middleware to parse JSON bodies
app.use(express.json());

// Define a simple route
app.get('/', (req: Request, res: Response) => {
  res.send('Hello, TypeScript with Express!');
});

// Define another route with parameters
app.get('/user/:name', (req: Request, res: Response) => {
  const { name } = req.params;
  res.send(`Hello, ${name}!`);
});

// Start the server
app.listen(PORT, () => {
  console.log(`Server is running on http://localhost:${PORT}`);
});
```

Basic Node Server in TypeScript - Steps

- Initialize a node project
- Install `express, typescript, @types/node, @types/express`
- Initialize typescript configuration file using `npx tsc –init`
- Create `server.ts` file in src directory
- Compile your `server.ts` using `npx tsc`
- Run the generated `server.js` file using `node src/server.js`
- Now the server should be up and running like in you did it in JavaScript

Good luck!