1. Basic Types:

TypesScript supports basic types such as `number`, `string`, `boolean`, `null`, and `undefined`.

Example 1 (Basic types):

typescript
Copy code

```
let age: number = 30; // number type
let name: string = "John"; // string type
let isActive: boolean = true; // boolean type
let emptyValue: null = null; // null type
let notAssigned: undefined = undefined; // undefined type
```

In this example:

- `age` is a number.
- `name` is a string.
- `isActive` is a boolean (true or false).
- `emptyValue` is explicitly `null`.
- `notAssigned` is `undefined` (no value assigned).

Example 2 (Boolean check):

```
let isFinished: boolean = false;
if (isFinished) {
  console.log("The task is finished.");
} else {
  console.log("The task is not finished yet.");
}
```

Here, `isFinished` is a boolean variable. Based on its value (`false`), the `else` statement runs, printing: "The task is not finished yet."

---

2. Object Types (Arrays, Tuples, Interfaces):

Object types include collections such as arrays, tuples, and structured types like interfaces.

Example 1 (Array):

```
let fruits: string[] = ["Apple", "Banana", "Mango"];
console.log(fruits[0]); // Output: "Apple"
```

In this array example:

- `fruits` is an array of strings.

Example 2 (Tuple and Interface):

```
// Tuple Example
let user: [number, string] = [1, "John"];
console.log(user[1]); // Output: "John"

// Interface Example
interface Person {
  name: string;
  age: number;
}

let employee: Person = { name: "Alice", age: 28 };
console.log(employee.name); // Output: "Alice"
```

Here:

- `user` is a tuple with a number and string.
- `Person` is an interface describing the shape of the object `employee`.

---

3. Type Inference:

TypeScript automatically infers types based on the value assigned to the variable.

Example 1 (Type Inference):

```
let city = "New York"; // TypeScript infers city as string
let population = 8000000; // population is inferred as number
```

TypeScript infers `city` as a `string` and `population` as a `number` based on their values.

Example 2 (Array Inference):

```
let colors = ["red", "green", "blue"]; // inferred as string[]
colors.push("yellow");
console.log(colors); // ["red", "green", "blue", "yellow"]
```

TypeScript infers `colors` as an array of strings.

---

4. Type Assertions:

Type assertions allow you to manually specify the type when TypeScript is unable to infer it.

Example 1 (Type Assertion):

```
let someValue: any = "Hello TypeScript!";
let strLength: number = (someValue as string).length;
console.log(strLength); // Output: 16
```

Here, we assert that `someValue` is a string, so we can use `.length`.

Example 2 (Assertion with DOM Element):

```
let inputElement = document.getElementById("input-field") as
HTMLInputElement;
inputElement.value = "New value!";
```

We assert that `inputElement` is an `HTMLInputElement`, so we can safely access its `value`.

---

5. Declaring Variables and Constants:

Example 1 (`let` and `const`):

```
let count = 10; // This value can be changed
count = 20;

const pi = 3.14; // A constant value
// pi = 3.1415; // Error: constant cannot be reassigned
```

Here, `let` allows changing `count`, but `const` prevents changing `pi`.

Example 2 (Block Scoping):

```
if (true) {
  let message = "Inside block";
  console.log(message); // Output: "Inside block"
}
// console.log(message); // Error: message is not defined
```

The `let` keyword ensures `message` is scoped to the block and not accessible outside.

---

6. Functions:

Example 1 (Defining Functions and Parameters):

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

console.log(add(5, 3)); // Output: 8
```

This function `add` takes two `number` parameters and returns their sum.

Example 2 (Optional and Default Parameters):

```
function greet(name: string = "Guest", age?: number): string {
  return `Hello ${name}, you are ${age ? age : "unknown"} years
old.`;
}

console.log(greet()); // Output: Hello Guest, you are unknown years
old.
console.log(greet("John", 25)); // Output: Hello John, you are 25
years old.
```

---

7. Classes:

Example 1 (Class and Constructor):

```
class Car {
  brand: string;
  constructor(brand: string) {
    this.brand = brand;
  }

  displayBrand() {
    console.log("This car is a " + this.brand);
  }
}

let myCar = new Car("Toyota");
myCar.displayBrand(); // Output: This car is a Toyota
```

Example 2 (Inheritance):

```
class Vehicle {
  constructor(public type: string) {}

  start() {
    console.log(`${this.type} is starting.`);
  }
}

class Motorcycle extends Vehicle {
  constructor() {
    super("Motorcycle");
  }
}

let myBike = new Motorcycle();
myBike.start(); // Output: Motorcycle is starting.
```

---

8. Modules:

Example 1 (Exporting and Importing):

```
// module.ts
export function sayHello() {
  console.log("Hello from module");
}

// main.ts
import { sayHello } from './module';
sayHello(); // Output: Hello from module
```

Example 2 (Default Exports):

```
// mathUtils.ts
export default function add(a: number, b: number) {
  return a + b;
}

// app.ts
import add from './mathUtils';
console.log(add(5, 3)); // Output: 8
```

---

9. Generics:

Example 1 (Generic Function):

```
function identity<T>(arg: T): T {
  return arg;
}

console.log(identity<string>("Hello")); // Output: Hello
console.log(identity<number>(123)); // Output: 123
```

Example 2 (Generic Class):

```
class Box<T> {
  content: T;
```

```
  constructor(content: T) {
    this.content = content;
  }

  getContent(): T {
    return this.content;
  }
}

let numberBox = new Box<number>(100);
console.log(numberBox.getContent()); // Output: 100
```