Let's go step by step. **JavaScript vs TypeScript** examples for each concept: variables, if/else, loops, functions, classes, and objects. I'll also explain **what TypeScript adds**.

---

1. Variables

JavaScript

```
var name = "Alice"; // string
var age = 25;       // number
var isStudent = true; // boolean
```

TypeScript

```
let name: string = "Alice";
let age: number = 25;
let isStudent: boolean = true;
```

 Explanation:

- TypeScript lets you **declare types explicitly** (`string`, `number`, `boolean`)
- Helps catch errors like:

```
age = "twenty-five"; //  Error in TypeScript, allowed in JS
```

---

2. If, Else If, Else

JavaScript

```
var age = 18;

if (age < 18) {
  console.log("Minor");
} else if (age === 18) {
  console.log("Just became adult");
} else {
  console.log("Adult");
}
```

TypeScript

```
let age: number = 18;

if (age < 18) {
  console.log("Minor");
```

```
} else if (age === 18) {
  console.log("Just became adult");
} else {
  console.log("Adult");
}
```

 Explanation:

- TypeScript checks **type safety** (`age` must be a number)
- JS allows runtime errors if wrong types are compared

---

3. Loops (for loop)

JavaScript

```
for (var i = 0; i < 5; i++) {
  console.log("Count:", i);
}
```

TypeScript

```
for (let i: number = 0; i < 5; i++) {
  console.log("Count:", i);
}
```

 Explanation:

- TypeScript adds **type safety** for loop counters (`i` is number)
- Helps prevent accidental assignment of a wrong type

---

4. Functions

JavaScript

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

console.log(add(5, 10));
```

TypeScript

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

console.log(add(5, 10));
```

 Explanation:

- `a: number, b: number` → parameters must be numbers
- `: number` → return type must be number
- TypeScript prevents `add("5", 10)` which would be allowed in JS

---

5. Classes and Objects

JavaScript

```
class Person {
  constructor(name, age) {
    this.name = name;
    this.age = age;
  }

  greet() {
    console.log("Hello, " + this.name);
  }
}

var person1 = new Person("Alice", 25);
person1.greet(); // Hello, Alice
```

TypeScript

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

```
  age: number;

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

  greet(): void {
    console.log("Hello, " + this.name);
  }
}

let person1 = new Person("Alice", 25);
person1.greet(); // Hello, Alice
```

 Explanation:

- TypeScript allows you to **define types for properties and methods**
- Prevents errors like assigning a number to `name`
- Helps with **autocomplete and better code suggestions**

---

6. Full Comparison Table

Concept	JavaScript	TypeScript	TypeScript Benefit
Variable	`var x = 5;`	`let x: number = 5;`	Catches type mistakes
If/Else	`if(a>5){}`	`if(a: number > 5){}`	Ensures correct type used in conditions
Loop	`for(var i=0;i<5;i++){}`	`for(let i:number=0;i<5;i++){}`	Type-checked loop counter
Function	`function add(a,b){}`	`function add(a:number,b:number):number {}`	Parameter & return type checking
Class/Object	`class Person{}`	`class Person{name:string; age:number}`	Property & method type safety

 Key Takeaways for Beginners

1. **TypeScript = JavaScript + Types** → safer and more predictable.
2. It **prevents common mistakes** like assigning wrong types.
3. Works for **all JS features** (functions, loops, classes, objects).
4. Great for **big projects** where errors can be costly.