**Generic**

In TypeScript, Generics are basically a kind of tool that enables you to create reusable code components that work with a number of types instead of a single type. Using Generics is a very safe way to empower classes, types and interfaces to act the way parameters act so that the developers are able to easily reuse them for all kinds of input.

In languages like C# and Java, one of the main tools in the toolbox for creating reusable components is generics, that is, being able to create a component that can work over a variety of types rather than a single one. This allows users to consume these components and use their own types.

The main reason to use generics in TypeScript is to enable types, classes or interfaces to act as parameters. It helps us reuse the same code for different types of input since the type itself is available as a parameter.

**Some benefits of generics are:**

* Defining a relationship between input and output parameters types. For example, function test <T>(input: T[]): T { … } allows you to make sure input and output use the same type, though input as an array.
* Stronger type checks at compile time will be available.
* You can remove some unnecessary type casts. For example, when you have const list: Array<Item> = [], going over array elements, you will have access to all the Item members.

**Example:**

function identity<T>(arg: T): T {

return arg;

}

We’ve now added a type variable T to the identity function. This T allows us to capture the type the user provides (e.g. number), so that we can use that information later. Here, we use T again as the return type. On inspection, we can now see the same type is used for the argument and the return type. This allows us to traffic that type information in one side of the function and out the other.

We say that this version of the identity function is generic, as it works over a range of types. Unlike using any, it’s also just as precise (ie, it doesn’t lose any information) as the first identity function that used numbers for the argument and return type.

Once we’ve written the generic identity function, we can call it in one of two ways. The first way is to pass all of the arguments, including the type argument, to the function:

let output = identity<string>("myString");

let output: string

Here we explicitly set T to be string as one of the arguments to the function call, denoted using the <> around the arguments rather than ().