## Introduction to TypeScript

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. TypeScript is pure object-oriented with classes, interfaces and statically typed programming languages like [C#](https://www.edureka.co/blog/c-sharp-tutorial/) or [Java](https://www.edureka.co/blog/java-tutorial/). It requires a compiler to compile and generate in JavaScript file. Basically, TypeScript is the ES6 version of JavaScript with some additional features.

A TypeScript code is written in a file with **.ts extension** and then compiled into JavaScript using the compiler.

You can write the file in any code editor and the compiler needs to be installed on your platform. After the installation, the command **tsc <filename>.ts** compiles the TypeScript code into a plain JavaScript file.

**Syntax:**

var message:string = "Welcome to Edureka!"  
console.log(message)

On compiling, it generates the following JavaScript code:

//Generated by typescript 1.8.10  
var message = "Welcome to Edureka!";  
console.log(message);

### Features of TypeScript

**features - typescript tutorial - edureka**



* **Cross-Platform:**  The TypeScript compiler can be installed on any Operating System such as Windows, MacOS and Linux.
* **Object-Oriented Language**: TypeScript provides features like [Classes](https://www.edureka.co/blog/javascript-classes/), Interfaces, and Modules. Thus, it can write object-oriented code for client-side as well as server-side development.
* **Static type-checking**: TypeScript uses static typing and helps type checking at compile time. Thus, you can find errors while writing the code without running the script.
* **Optional Static Typing**: TypeScript also allows optional static typing in case you are using the dynamic typing of JavaScript.
* **DOM Manipulation**: You can use TypeScript to manipulate the DOM for adding or removing elements.
* **ES 6 Features**: TypeScript includes most features of planned ECMAScript 2015 (ES 6, 7) such as class, interface, Arrow functions, etc.

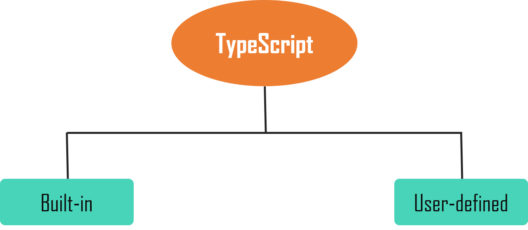
### Advantages of Using TypeScript

* TypeScript is **fast, simple, easy to learn** and runs on any browser or JavaScript engine.
* It is **similar** to **JavaScript** and uses the same syntax and semantics.
* This helps backend developers write front-end **code faster**.
* TypeScript code can be called from an **existing JavaScript code**. Also, it works with existing JavaScript frameworks and libraries without any issues.
* The Definition file, with .d.ts extension, provides support for existing JavaScript libraries like **Jquery, D3.js**, etc. So, TypeScript code can add [JavaScript libraries](https://www.edureka.co/blog/javascript-libraries/) using type definitions to avail the benefits of type-checking, code autocompletion, and documentation in existing dynamically-typed JavaScript libraries.
* It includes features from **ES6** and **ES7** that can run in ES5-level JavaScript engines like [Node.js](https://www.edureka.co/blog/node-js-installation/).

Now that you have understood what is TypeScript, let’s move on with this TypeScript Tutorial and have a look at the different types.

## TypeScript Types

The Type System represents the different types of values supported by the language. It checks the **validity** of the supplied **values** before they are stored or manipulated by the program.



It can be classified into two types such as:

* **Built-in**: This includes number, string, boolean, void, null and undefined.
* **User-defined**: It includes enums, classes, interfaces, arrays, and tuple.

Now let’s move on with this TypeScript Tutorial and understand more about variables.

## TypeScript Tutorial: Variables

A variable is a named space in the memory which is used to store values.

The type syntax for declaring a variable in TypeScript includes a colon (:) after the variable name, followed by its type.

Similar to JavaScript, we use the [var keyword](https://www.edureka.co/blog/javascript-variable/) to declare a variable.

There are four options when we declare a variable:

var [identifier] : [type-annotation] = value;

var [identifier] : [type-annotation];

var [identifier] = value;

var [identifier];

**Example:**

var name:string = "Daisy";

var empid:number = 1001;

console.log("name"+name)

console.log("employee id "+empid)

On compiling, it will generate following JavaScript code:

//Generated by typescript 1.8.10

var name = "Daisy";

var empid = 1001;

console.log("name" + name);

console.log("employee id: " + empid);

**Output:**

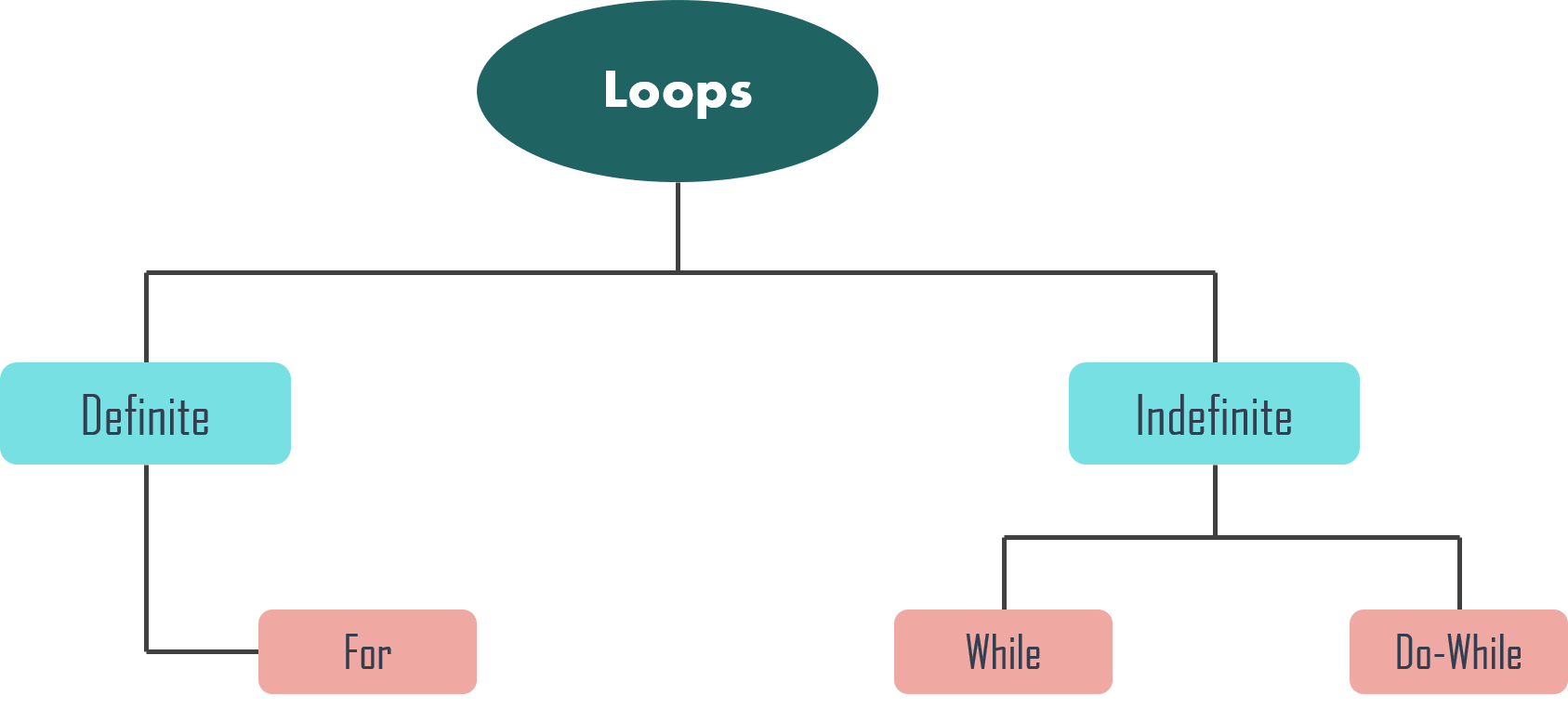
name:Daisy  
employee id:1001

Now let’s move on to the next topic of our TypeScript Tutorial.

## Loops

There may be situations when a block of code needs to be executed several numbers of times. A [loop](https://www.edureka.co/blog/javascript-loops/) statement allows us to execute a statement or group of statements multiple times.

The TypeScript loops can be classified as:



### For Loop

The [for loop](https://www.edureka.co/blog/javascript-foreach/) is an implementation of a definite loop.

**Syntax:**

for (first expression; second expression; third expression ) {

// statements to be executed repeatedly

}

Here, the first expression is executed before the loop starts. The second expression is the condition for the loop to execute. And the third expression is executed after the execution of every code block.

**Example:**

for (let i = 0; i < 2; i++) {

console.log ("Execute block statement" + i);

}

**Output:**

Execute block statement 0 Execute block statement 1

### While Loop

The while loop executes the instructions each time the condition specified evaluates to true.

**Syntax:**

while (condition expression) {  
// code block to be executed  
}

**Example:**

let i: number = 1;  
  
while (i < 3) {  
console.log( "Block statement execution no." + i )  
i++;  
}

**Output:**

Block statement execution no.1 Block statement execution no.2

## Functions

In JavaScript, [functions](https://www.edureka.co/blog/javascript-functions/) are one of the most important part as it is a functional programming language. Functions ensure that the program is maintainable and reusable, and organized into readable blocks. While TypeScript provides the concept of classes and modules, functions still are an integral part of the language.

### Named Functions

A named function is used to declare and call a function by its given name.

**Example:**

function display() {  
console.log("TypeScript Function");  
}  
  
display();

**Output:**

TypeScript Function

### Anonymous Function

An anonymous function is one which is defined as an expression. This expression is stored in a variable. These functions are invoked using the variable name that the function is stored in.

**Example:**

let greeting = function() {  
console.log("TypeScript Function");  
};  
  
greeting();

**Output:**

TypeScript Function

### Arrow Function

Fat arrow notations are used for anonymous functions i.e for function expressions. They are also called lambda functions in other languages.

**Syntax:**

(param1, param2, ..., paramN) => expression

Using the fat arrow (=>) drops the need to use the ‘function’ keyword. Parameters are passed in the angular brackets <>, and the function expression is enclosed within the curly brackets {}.

**Example:**

let sum = (x: number, y: number): number => {  
return x + y;  
}  
  
sum(10, 30); //returns 40

### Function Overloading

TypeScript provides the concept of function overloading. Thus, you can have multiple functions with the same name but different parameter types and return type.

**Example:**

function add(a:string, b:string):string;  
function add(a:number, b:number): number;  
return a + b;  
}  
add("Hello ", "Edureka"); // returns "Hello Edureka"  
add(10, 10); // returns 20

In the above example, we have the same function add() with two function declarations and one function implementation. The first signature has two parameters of type string, whereas the second signature has two parameters of the type number.

These are the different types of functions. Now, let’s move on with our TypeScript Tutorial and understand strings in TypeScript.

## TypeScript Tutorial: Strings

The [string](https://www.edureka.co/blog/javascript-string-functions/) is another primitive data type that is used to store text data. String values are surrounded by single quotation marks or double quotation marks.

**Syntax:**

var var\_name = new String(string);

There are different properties of methods available in String object such as:

* **Constructor** – It returns a reference to the String function that created the object
* **Length** – This returns the length of the string
* **Prototype** – This property allows you to add properties and methods to an object

**Example:**

let name = new String("Welcome to Edureka!");  
console.log("Message: " +name);  
console.log("Length: "+name.length);

**Output:**

Message: Welcome to Edureka!  
Length: 19

## Arrays in TypeScript

An [array](https://www.edureka.co/blog/javascript-array/) is a special type of data type that stores multiple values of different data types sequentially using a special syntax. Array elements are identified by a unique integer called the subscript or index of the element.

**Syntax:**

var array\_name[:datatype]; //declaration  
array\_name = [val1,val2,valn..] //initialization

**Example:**

let names: Array<string>;  
names = ['John', 'Daisy', 'Rachel'];  
  
let ids: Array<number>;  
ids = [101, 700, 321];

## TypeScript Interfaces

The interface is a structure that defines the contract in your application. It defines the syntax for classes to follow. It contains only the declaration of the members and it is the responsibility of the deriving class to define the members.

**Example:**

interface Employee {  
empID: number;  
empName: string;  
getSalary: (number) => number; // arrow function  
getManagerName(number): string;  
}

In the above example, the **Employee** interface includes two properties **empID** and **empName**. It also includes a method declaration **getSalaray** using an **arrow function** which includes one number parameter and a number return type. The **getManagerName** method is declared using a normal function.

## TypeScript Classes

TypeScript introduced classes so that they can avail the benefits of object-oriented techniques like encapsulation and abstraction. The class in TypeScript is compiled to plain JavaScript functions by the TypeScript compiler to work across platforms and browsers.

A class includes the following:

* **Constructor**
* **Properties**
* **Methods**

**Example:**

class Employee {  
empID: number;  
empName: string;  
  
constructor(ID: number, name: string) {  
this.empName = name;  
this.empID = ID;  
}  
  
getSalary() : number {  
return 40000;  
}  
}

### Inheritance

TypeScript supports [Inheritance](https://www.edureka.co/blog/inheritance-in-javascript/) as it is the ability of a program to create new classes from an existing class. The class that is extended to create newer classes is called the parent class or the super class. The newly created classes are called the child or sub classes.

A class inherits from another class using the ‘extends’ keyword. Child classes inherit all properties and methods except private members and constructors from the parent class. But, TypeScript does not support multiple inheritance.

**Syntax:**

class child\_class\_name extends parent\_class\_name

**Example:**

class Person {  
name: string;  
  
constructor(name: string) {  
this.name = name;  
}  
}  
  
class Employee extends Person {  
empID: number;  
  
constructor(empID: number, name:string) {  
super(name);  
this.empID = empid;  
}  
  
displayName():void {  
console.log("Name = " + this.name + ", Employee ID = " + this.empID);  
}  
}  
  
let emp = new Employee(701, "Jason");  
emp.displayName(); // Name =Jason, Employee ID = 701

Now that you know about classes, let’s move ahead with this TypeScript Tutorial and learn about Objects.

## Objects in TypeScript

An object is an instance that contains a different set of key-value pairs. The values can be scalar values or functions or even an array of other objects.

**Syntax:**

var object\_name = {  
key1: “value1”, //scalar value  
key2: “value”,  
key3: function() {  
//functions  
},  
key4:[“content1”, “content2”]

An object can contain scalar values, functions and structures like arrays and tuples.

**Example:**

var person = {  
firstname:"Danny",  
lastname:"Green"  
};  
//access the object values  
console.log(person.firstname)  
console.log(person.lastname)

On compiling, it will generate the same code in JavaScript.

**Output:**

Danny  
Green

These were the different important elements of TypeScript. Now, let’s move on with this TypeScript Tutorial and have a look at an example to understand the use case.