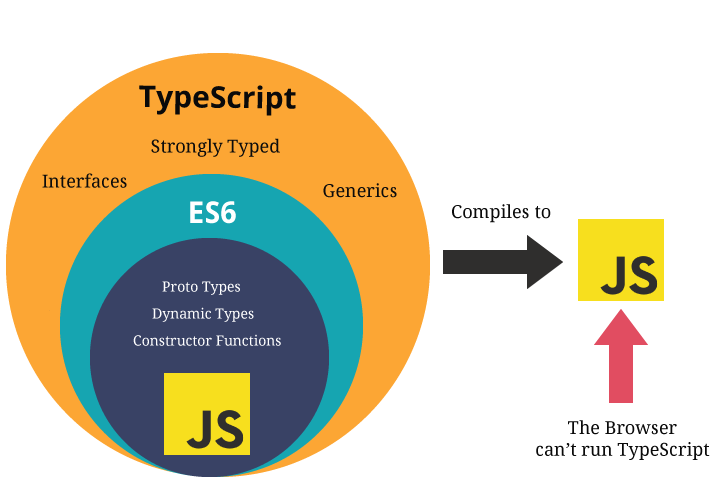
**TypeScript**

**What is TypeScript?**

****

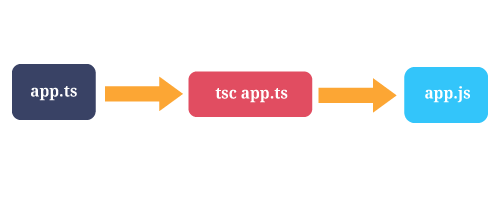
* TypeScript is wrapper around the JavaScript.
* TypeScript is Strongly Typed Language which means you have to be specific with Variable type.
* TypeScript doesn’t run in the browser. We need to compile it to JS to run.
* TypeScript is a typed superset of JavaScript and pure Object oriented.

**Features of TypeScript:**

* Due to the static typing, code written in TypeScript is more predictable, and is generally easier to debug.
* Makes it easier to organize the code for very large and complicated apps.
* You can use TypeScript for other JS libraries, Because TS compiles plain JS.
* TS is platform Independent.

**TypeScript Compilation**

* The file extension of TypeScript is “**.ts**”.
* The TSC (TypeScript Compiler) is a source-to-source compiler (transcompiler / transpiler).
* The TSC generates the equivalent JS code from the source TS code. This process is termed as transpilation.



* ***tsc first.ts*** This will create a new file named first.js in the same location. Keep in mind that if you already had a file named **first.js**, it would be overwritten.
* You can run multiple files at the time by using ***tsc first.ts second.ts third.ts***
* Initializes a watcher process that will keep main.js up to date.  
  ***tsc main.ts --watch***

**Node Introduction**

1. Node.js is a server-side platform built on Google Chrome's JavaScript Engine (V8 Engine).
2. Cross-platform runtime environment used for development of server-side web applications.
3. Node.js applications are written in JavaScript and can be run on a wide variety of operating systems.
4. Node.js is based on an event-driven.
5. Non-blocking Input/Output.
6. Node.js also has thousands of JavaScript modules which simplifies the development of web applications called as NPM.
7. Single thread process.

**Node js suits for ?**

Node js can do all the tasks which can be done by php, java, .net etc.,

For efficiently we can use it for the below applications,

1. Chat applications.
2. Game servers.
3. Streaming Servers.
4. Advertisement servers.
5. Single Page Applications.

**Module Types:**

Module in Node.js is a simple or complex functionality organized in single or multiple JavaScript files which can be reused throughout the Node.js application.

Node.js includes three types of modules:

Core Modules

Local Modules

Third Party Modules

**Node.js Core Modules:**

Node.js is a lightweight framework. The core modules include bare minimum functionalities of Node.js. These core modules are load automatically when Node.js process starts.

However, you need to import the core module first in order to use it in your application.

**Loading Core Modules:**

In order to use Node.js core or NPM modules, you first need to import it using require() function as shown below.

var module = require('module\_name');

As per above syntax, specify the module name in the require() function. The require() function will return an object, function, property or any other JavaScript type, depending on what the specified module returns.

**Node.js Third party Module:**

The third party module can be downloaded by NPM (Node Package Manager). These type of modules are developed by others and we can use that in our project. Some of the best third party module examples are listed as follows: express, gulp, lodash, async, socket.io, mongoose, underscore, pm2, bower, q, debug, react, etc.,

Third party modules can be install inside the project folder or globally.

**Loading Third Party Module:**

Third party Node.js module can be downloaded using NPM (node package manager) which you can download locally or globally. To download globally we use the following command.

**npm install  <module\_name>  -g**

here  we use -g to install package globally. If you want to install locally then use the following command.

**npm install  <module\_name>  --save**

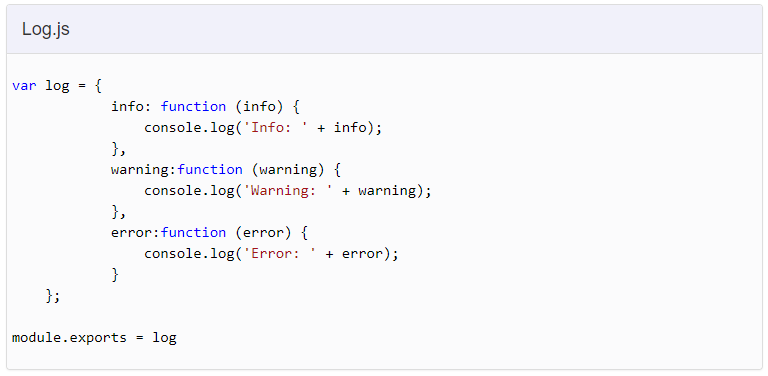
Above command will download node package inside node\_modules folder and then you can directly use require function to load node module.

**var module= require('module\_name');**

**Node.js Local Module:**

Local modules are modules created locally in your Node.js application. These modules include different functionalities of your application in separate files and folders. You can also package it and distribute it via NPM, so that Node.js community can use it.

In Node.js, module should be placed in a separate JavaScript file. So, create a Log.js file and write the following code in it.

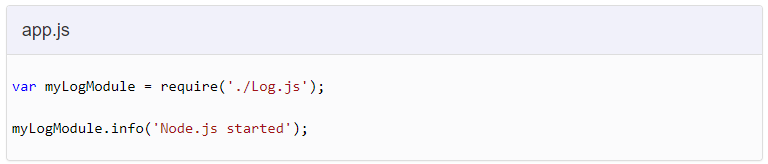


In the above example of logging module, we have created an object with three functions - **info(), warning() and error()**. At the end, we have assigned this object to **module.exports**. The module.exports in the above example exposes a log object as a module.

The module.exports is a special object which is included in every JS file in the Node.js application by default. Use module.exports or exports to expose a function.

**Loading Local Module:**

To use local modules in your application, you need to load it using require() function in the same way as core module. However, you need to specify the path of JavaScript file of the module.



In the above example, app.js is using log module. First, it loads the logging module using require() function and specified path where logging module is stored. Logging module is contained in Log.js file in the root folder. So, we have specified the path './Log.js' in the require() function. The '.' denotes a root folder.

The require() function returns a log object because logging module exposes an object in Log.js using module.exports. So now you can use logging module as an object and call any of its function using dot notation e.g **myLogModule.info()** or **myLogModule.warning()** or **myLogModule.error()**

**To know the Node and NPM versions**

**node -v and npm -v**

**Execute the TS files directly from Node cmd**

1. Install ts-node npm globally by using this command

*npm install typescript -g*

*npm install ts-node -g*

1. Run the below command

*ts-node fileName.ts*

**How to Identify Installed Package version**

*npm info* ***packageName*** *version*

**Variable declaration in TypeScript**

We can declare a variable in TS by using the following ways

i. var variableName : dataType = value;

ii. var variableName : dataType;

iii. var variableName = value;

iv. var variableName; (you can store any values here)

**Difference between *var* and *let:***

|  |  |
| --- | --- |
| **let** | **var** |
| let allows you to declare variables that are limited in scope to the block, statement, or expression on which it is used. | This is unlike the [var](https://developer.mozilla.org/en-US/docs/JavaScript/Reference/Statements/var) keyword, which defines a variable globally, or locally to an entire function regardless of block scope. |
| Variables declared by let have their scope in the block for which they are defined, as well as in any contained sub-blocks. | In this way, let works very much like var. The main difference is that the scope of a var variable is the entire enclosing function |

**Example:**

|  |
| --- |
| function varTest() {   var x = 1;   if (true) {     var x = 2;  // same variable!     console.log(x);     }   console.log(x);   } |

|  |
| --- |
| function letTest() {   let x = 1;   if (true) {     let x = 2;  // different variable     console.log(x);     }   console.log(x);   } |

**String:**

let name : string = “gopi”;

You can change the variable value later but not datatype.

**Template String:**

Template strings are surrounded by the backtick/backquote (`) character, and embedded expressions are of the form *${ expr }*.

**Example:**

|  |
| --- |
| let g:string = "gopi"; let age:number = 30; let msg = `Name is ${g}  age is ${age}`; console.log(msg); |

**Result:**

Name is gopi

age  is 30

**Number:**

var num : number = 20;

**Boolean:**

var boo : boolean = true;

**Array:**

var arr = [“string”, “string”]; => this is a string Array, you can store only string values not other data types.

var arr = [“string”, number]; => now the array only accept string and numbers only not other data types.

var arr : string[]; => this is a string Array, you can store only string values not other data types.

var arr : any[]; => now this array accept any kind of data types.

var xyz : string | number; // union method

var arr : (string|number|boolean)[];

**Tuple:**

The tuple data type allows you to create an array where the type of a fixed number of elements is known in advance. The type of the rest of the elements can only be one of the types that you have already specified for the tuple.

var tu : [number, string];

**Object:**

*var obj: {name:string, phone:number} = {name:”gopi”, phone:123};*

In the above code you can only use 2 properties which is name and phone. You can’t dynamically add more properties.

If you want to add more properties dynamically means just use,

*var obj : {[k:string]:string} = {};*

Now you can add more properties.

**Enum:**

Enum allows us to define a set of named numeric **constant**. By default, enum is zero-based but we can change its value according to the requirement.

***enum*** *Animals {cat, lion, dog, cow, monkey};*

*let c: Animals = Animals.cat; // 0*

feature of enums is that you can also go from a numeric value to the name of that value in the enum

*console.log(Animals[3]); // cow*

By default, the numbering of enums starts at 0, but you can also set a different value for the first or any other members manually. This will change the value of all the members following them by increasing their value by 1. You can also set all the values manually in an enum.

**any:**

The any type is a powerful way to work with existing JavaScript, allowing you to gradually opt-in and opt-out of type-checking during compilation.

*var x:any = “test”;*

**void:**

void is a little like the opposite of any: the absence of having any type at all. You may commonly see this as the return type of functions that do not return a value:

*function warnUser(): void {  
    alert("This is my warning message");  
}*

Declaring variables of type void is not useful because you can only assign undefined to them:

*let unusable: void = undefined;*

**Null and Undefined:**

*let u: undefined = undefined;  
let n: null = null;*

**Using Types in functions**

We can use the data types in functions also.

**Difference** **between *let* and *var:***

* Unlike var , let and const are block scoped.
* Once you bind a value/object to a variable using const, you can't reassign to that variable.

**Functions:**

Functions are the fundamental building block of any applications.

In typescript also we can write functions as named functions or anonymous function.