

# TypeScript

The  
superset  
of  
javascript  
that  
TRANSPILES  
to  
Java Script

## Installation

Install Node JS (npm will be installed with it)

Check the version of node.

**node -v** ( mine is v6.11.1 )

then check the version of npm,

**npm -v** (mine is 3.10.10)

Install typescript

**npm install -g typescript**

then check the version of tsc (tsc is the TypeScript Transpiler)

**tsc -v** (mine is 2.4.2)

Create a folder named as tseg (for typescript examples) (wherever you want)

move to the tseg folder, use editor of your choice (vi, vim, edit, gedit or whatever)

create the following file

**eg1.ts**

```
class ExampleOne
{
  public static whatever(): number
  {
    console.log("God is great");
    console.log("Ujjain is the city of Gods");
    console.log("I live in Ujjain");
    return 0;
  }
}
ExampleOne.whatever();
```

---

To transpile the above code, type

**tsc eg1.ts**

the transpiler will create eg1.js, to execute, type

**node eg1.js**

If you are interested in using IDE,

Install **Visual Studio Code** ( The opensource editor created by Microsoft ). Please note that I am not talking about Visual Studio, I am talking about **Visual Studio Code**

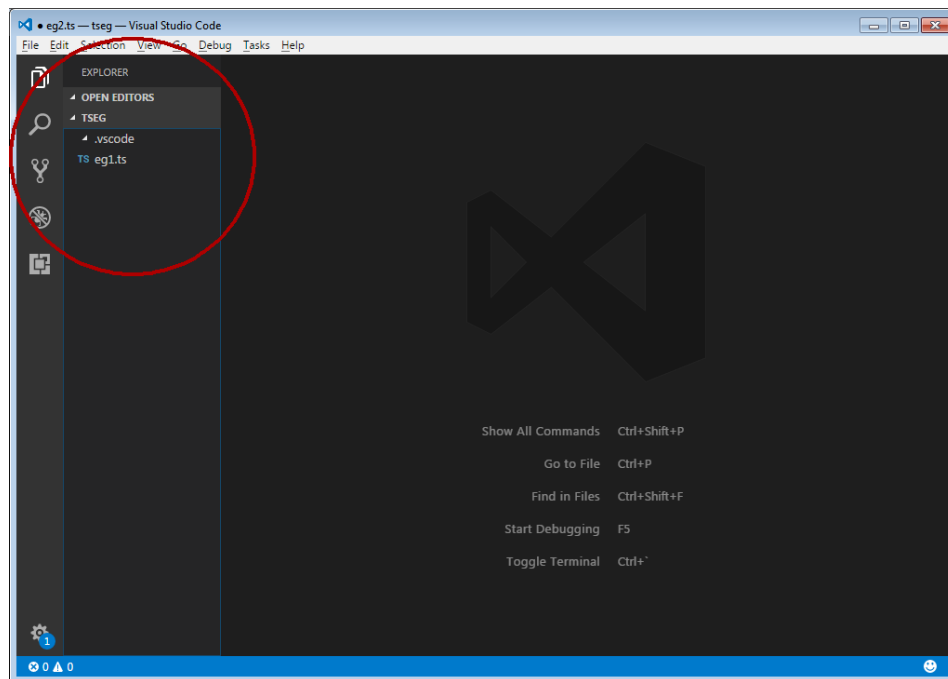
Download and install, it is free.

Now move to the tseg folder

Now while staying in the tseg folder type

**code** .

Visual studio code will be launched, with the tseg folder opened in it, something as follows



Now create a new file and save it as eg2.ts, then type the following code in it.

**eg2.ts**

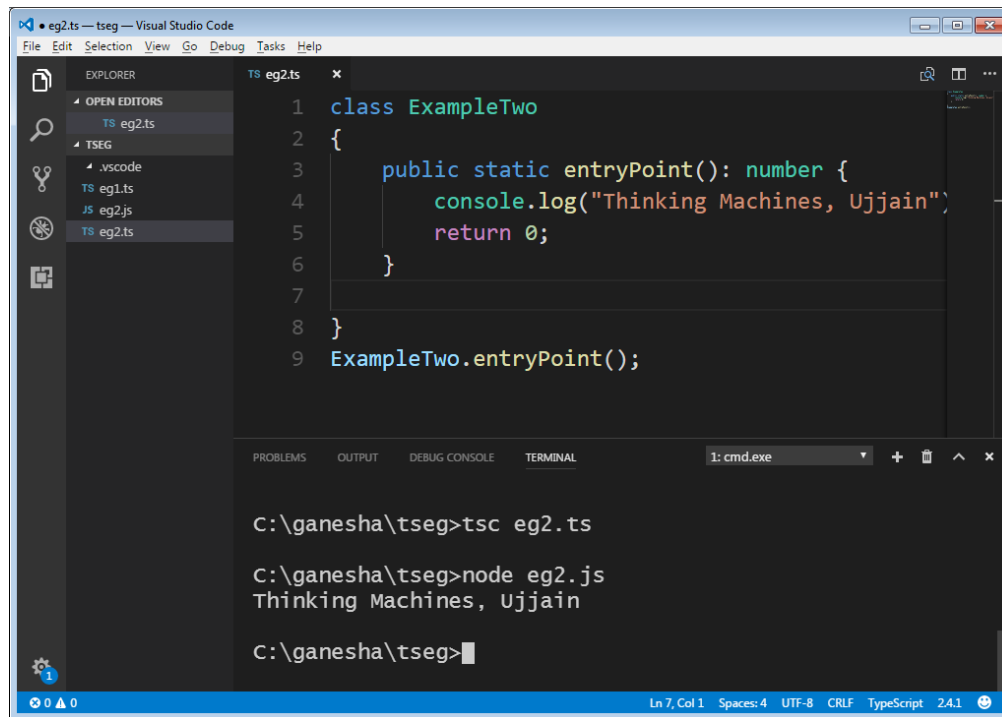
```
class ExampleTwo
{
    public static entryPoint(): number {
        console.log("Thinking Machines, Ujjain");
        return 0;
    }
}
ExampleTwo.entryPoint();
```

---

Now select the **View** options from menu bar followed by **Integrated terminal**

In the terminal type `tsc eg2.ts` to compile, and if there are no errors then type `node eg2.js` to execute. Right now I strongly suggest that you don't need to configure tasks and launch to debug from within

the Visual Studio Code Editor.



The screenshot shows the Visual Studio Code editor interface. The Explorer sidebar on the left displays the file structure with 'eg2.ts' selected. The main editor window shows the following TypeScript code:

```
1 class ExampleTwo
2 {
3     public static entryPoint(): number {
4         console.log("Thinking Machines, Ujjain");
5         return 0;
6     }
7
8 }
9 ExampleTwo.entryPoint();
```

Below the editor, the TERMINAL panel is active, showing the execution of the code:

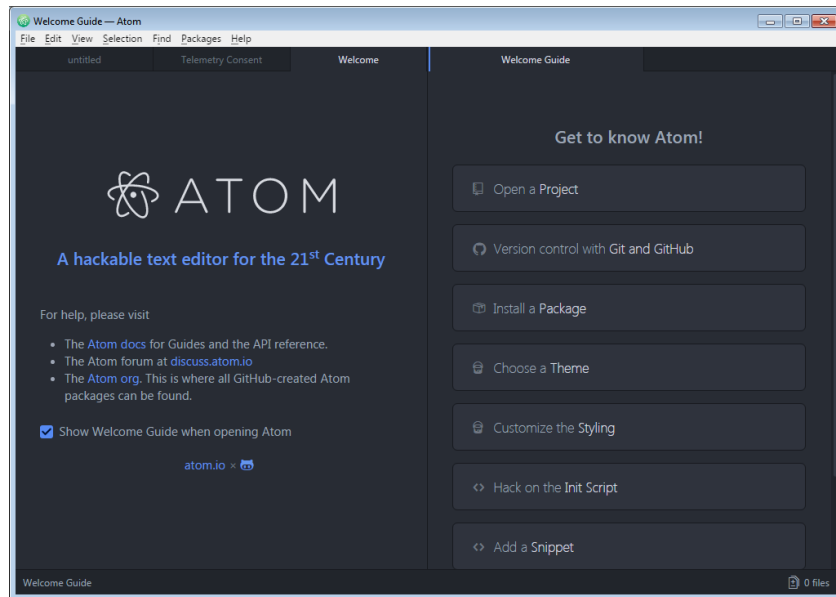
```
C:\ganesha\tseg>tsc eg2.ts
C:\ganesha\tseg>node eg2.js
Thinking Machines, Ujjain
C:\ganesha\tseg>
```

The status bar at the bottom indicates the current position is Line 7, Column 1, with 4 spaces, using UTF-8 encoding, CRLF line endings, and the TypeScript language.

Now let us see how we can use ATOM IDE instead of Visual Studio Code.

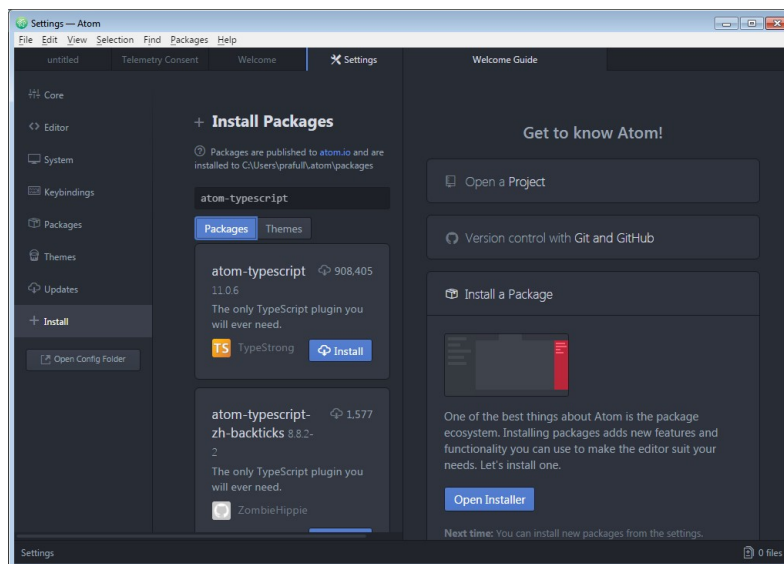
## Download and install ATOM ( A hackable text editor for the 21<sup>st</sup> Century)

Start ATOM IDE and this is what you should see

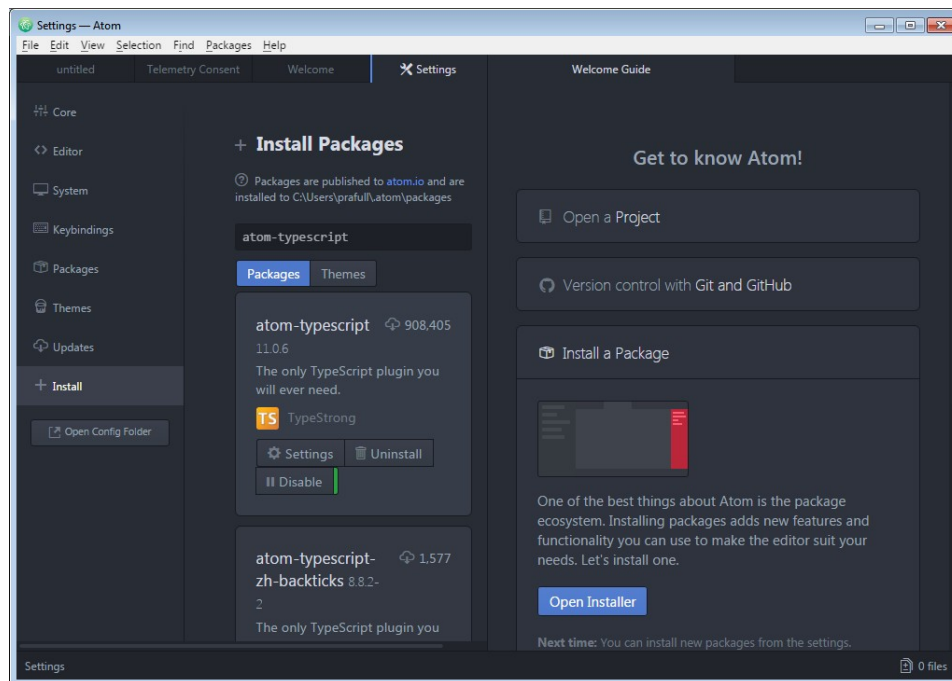


Select Install a package, then click the Open Installer button from the expanded view, then in the textbox that says search packages type **atom-typescript** and click the package button

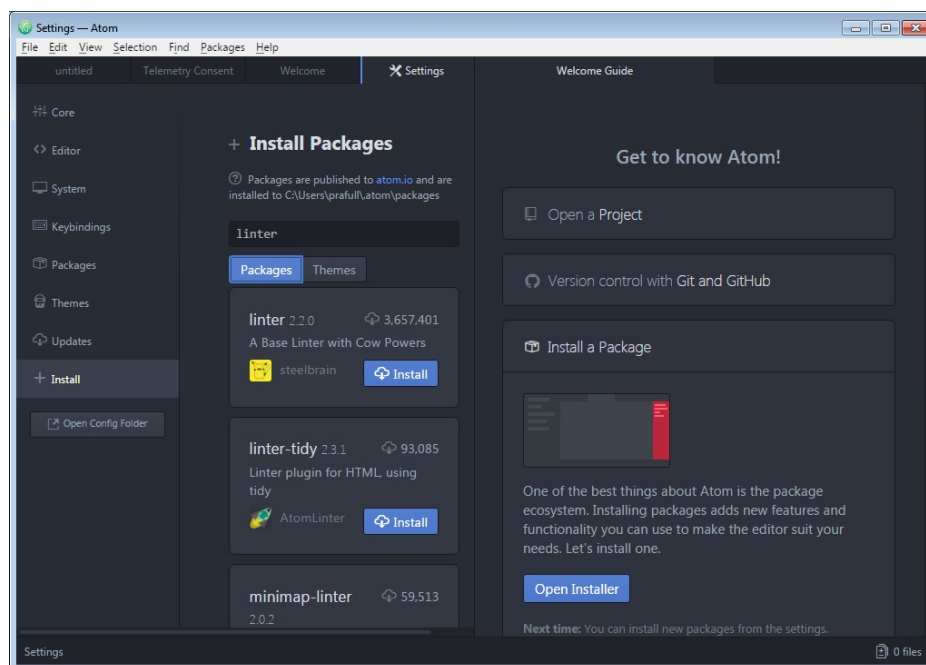
After a few seconds you should see the following



Click the Install button, after some time the following should appear, it means that the plugin was successfully installed.

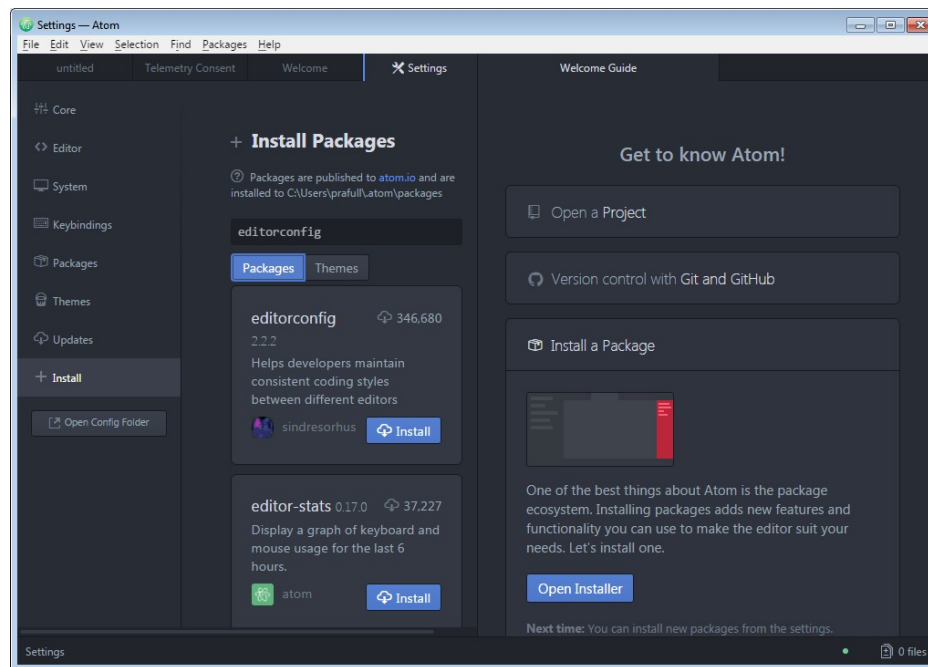


Now do the same for installing **linter**

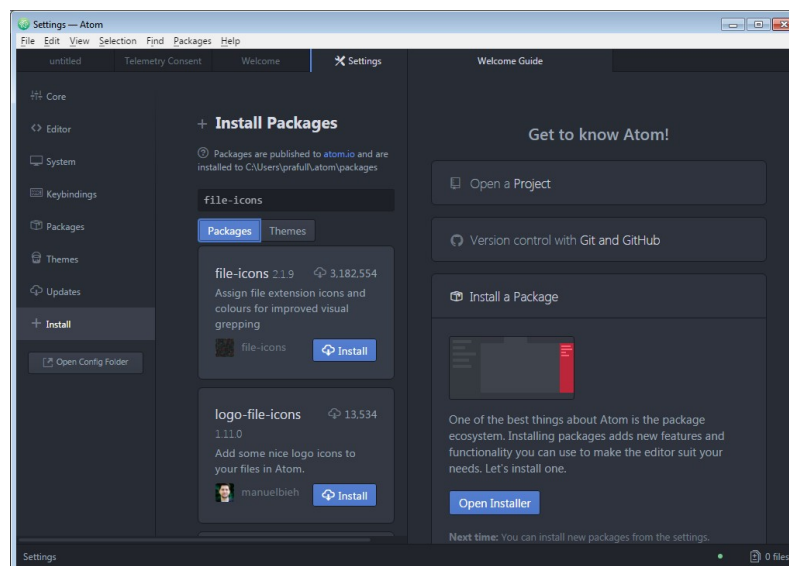


In between it will ask you permission multiple times to install dependencies, allow it to install.

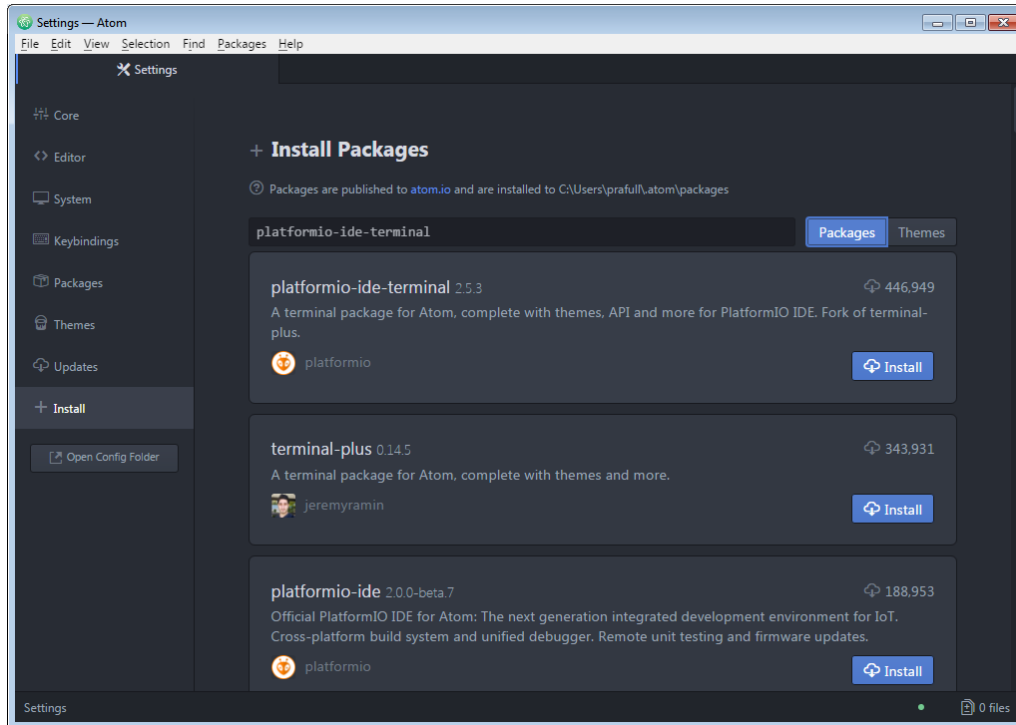
Now do the same for **editorconfig**



Right now it is of no use, but do the same for **file-icons**



Now close this window, we will learn to open the same installation UI from menu. Select **File** option from the top menu bar, follower by **Settings**, then click install and type the name of the package **platformio-ide-terminal**, see the following screenshot



Now open the tseg folder and create the following file

**eg3.ts**

```
class ExampleThree
{
    public static main(): number {
        console.log("You can get kicks out of programming.")
        return 0;
    }
}
```

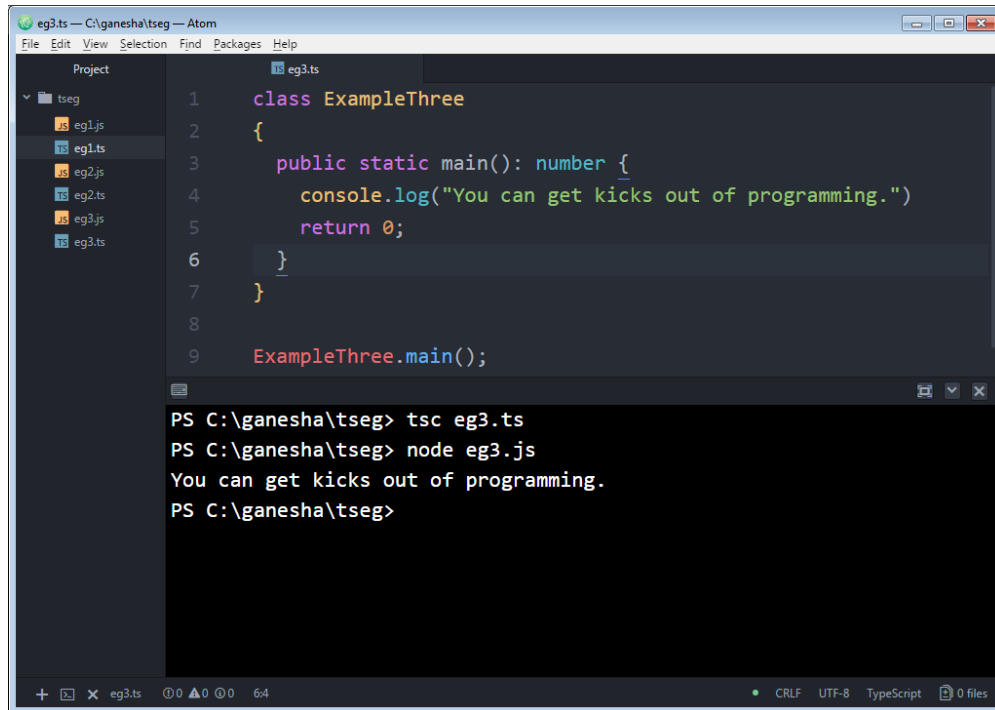
ExampleThree.main();

---



Now press **ctrl `**

The terminal panel will open as shown below, use **tsc eg3.ts** to transpile and **node eg3.js** to run as show below



```
eg3.ts — C:\ganesha\tseg — Atom
File Edit View Selection Find Packages Help

Project
└─ tseg
   ├── eg1.js
   ├── eg1.ts
   ├── eg2.js
   ├── eg2.ts
   ├── eg3.js
   └── eg3.ts

1  class ExampleThree
2  {
3      public static main(): number {
4          console.log("You can get kicks out of programming.")
5          return 0;
6      }
7  }
8
9  ExampleThree.main();

PS C:\ganesha\tseg> tsc eg3.ts
PS C:\ganesha\tseg> node eg3.js
You can get kicks out of programming.
PS C:\ganesha\tseg>
```

# Have fun.

eg4.ts

```
class Bulb
{
    private wattage:number;
    setWattage(wattage:number){
        if(wattage>=0 && wattage<=240)
            this.wattage=wattage;
        else
            this.wattage=0;
    }
    getWattage():number{
        return this.wattage;
    }
}
var b=new Bulb();
b.setWattage(60);
```

```
console.log("Wattage is : "+b.getWattage());
```

---

**eg5.ts**

```
class Bulb
{
  private wattage:number;
  constructor(wattage:number){
    this.setWattage(wattage);
  }
  setWattage(wattage:number){
    if(wattage>=0 && wattage<=240)
      this.wattage=wattage;
    else
      this.wattage=0;
  }
  getWattage():number{
    return this.wattage;
  }
}
var b=new Bulb(60);
console.log("Wattage is : "+b.getWattage());
```

---

Henceforth assume that I am validating input to keep the object in valid state, because I don't want to waste your and my time in writing if/else

**eg6.ts**

```
class Rectangle
{
  private length:number;
  private breadth:number;
  constructor(length:number,breadth:number)
  {
    this.length=length;
    this.breadth=breadth;
  }
  getLength():number
  {
    return this.length;
  }
  getBreadth():number
  {
    return this.breadth;
  }
}
class Box extends Rectangle
{
  private height:number;
  constructor(length:number,breadth:number,height:number)
  {
```

```

super(length,breadth);
this.height=height;
}
public getHeight()
{
    return this.height;
}
}
var b=new Box(10,3,15);
console.log("Length : "+b.getHeight());
console.log("Breadth : "+b.getBreadth());
console.log("Height : "+b.getHeight());

```

---

eg7.ts

```

class Bulb
{
    private w:number;
    public set wattage(w:number)
    {
        this.w=(w>=0 && w<=240)?w:0;
    }
    public get wattage()
    {
        return this.w;
    }
}
var b=new Bulb();
b.wattage=60;
console.log("Wattage is "+b.wattage);

```

---

Note : the example ts7.ts has access methods (set/get), hence to transpile such examples, you need to provide the target option as follow

**tsc --target ES5 eg7.ts**

then

**node eg7.js**

---

eg8.ts

```

class Bulb
{
    public static cnt:number=0;
    constructor()
    {
        Bulb.cnt++;
    }
    static release()
    {

```

```
Bulb.cnt--;  
return null;  
}  
}  
var b1=new Bulb;  
var b2=new Bulb;  
console.log("Number of Bulbs : "+Bulb.cnt);  
var b3=new Bulb;  
console.log("Number of Bulbs : "+Bulb.cnt);  
b1=Bulb.release(); // it is a trick, don't take it as destructor  
console.log("Number of Bulbs : "+Bulb.cnt);
```

---

**eg9.ts**

```
abstract class Car  
{  
    public abstract manual();  
}  
class HondaCity extends Car{  
    public manual(){  
        console.log("Blah blah blah about HondaCity")  
    }  
}  
abstract class Matiz extends Car{  
  
}  
class Herbie  
{  
}  
class HondaJazz  
{  
    manual(){  
        console.log("Blah blah blah about HondaJazz")  
    }  
}  
  
function doService(c:Car)  
{  
    c.manual();  
}
```

```
// var m=new Matiz; // Matiz cannot be instantiated  
var c:Car;  
// c=new Herbie; // won't compile as it is not implementing manual  
c=new HondaJazz; // will compile, even if it is not extending Car, since it is implementing manual  
c.manual();  
c=new HondaCity;
```

```
c.manual();
doService(new HondaCity);
doService(new HondaJazz);
// doService(new Herbie); // won't compile for the same reasons
```

---

## Fun starts here

eg10.ts

```
class Bulb
{
  constructor(private wattage:number){
  }
  getWattage(): number {
    return this.wattage;
  }
}
var b:Bulb;
b=new Bulb(60);
console.log("Wattage is : "+b.getWattage());
```

---

eg11.ts

```
// This code won't compile
function add(a:number,b:number)
{
  console.log(a+b);
}
function add(a:string,b:string)
{
  console.log(a+b);
}
add(10,20);
add("Thinking ", "Machines");
```

---

eg12.ts

```
function add(a:number,b:number);
function add(a:string,b:string);
function add(a:any,b:any){
  if(a && b && typeof a ==="number" && typeof b==="number")
  {
    console.log(a+b);
  } else if(a && b && typeof a ==="string" && typeof b==="string")
  {
    console.log(a+b);
  }
}

add(10,20);
add("Thinking ", "Machines");
```

```
//add(10,"Cool"); // won't compile  
// add(true,false); // won't compile
```

---

**eg13.ts**

```
interface Bulb  
{  
  wattage:number;  
  setWattage(number);  
  getWattage():number;  
}  
class Philips implements Bulb{  
  wattage:number;  
  setWattage(wattage:number)  
  {  
    this.wattage=wattage;  
  }  
  getWattage():number {  
    return this.wattage;  
  }  
}  
/* the following won't compile because it does not implement Bulb features  
class Silvania implements Bulb{  
  
}  
*/  
function lightIt(b:Bulb)  
{  
  console.log("The bulb of wattage : "+b.getWattage()+" has been turned on.");  
}  
  
var p=new Philips;  
p.setWattage(60);  
lightIt(p);
```

---

**eg14.ts**

```
module Inventory  
{  
  export class Item {  
    public getBrand(): string  
    {  
      return "Philips";  
    }  
  }  
}  
  
var k=new Inventory.Item;  
console.log(k.getBrand());
```

---

**eg15.ts**

```
var add=function(x:number,y:number):number {  
    return x+y;  
}
```

```
console.log(add(10,20));
```

---

**eg16.ts**

```
var add=function(x,y) {  
    return x+y;  
}
```

```
console.log(add(10,20));  
console.log(add("Thinking ","Machines"));
```

---

**eg17.ts**

```
var add:(x:number,y:number)=>number=function(x:number,y:number):number {  
    return x+y;  
}
```

```
console.log(add(10,20));
```

---

**eg18.ts**

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

```
console.log(add(10,20));  
console.log(add(10,20,30));
```

---

**eg19.ts**

**// the following code won't compile**

```
function add(a:number,b:number,c?:number,d:number):number  
{  
    if(c) return a+b+c+d;  
    return a+b+d;  
}
```

```
console.log(add(10,20));  
console.log(add(10,20,30));  
console.log(add(10,20,30,40));
```

---

**eg20.ts**

```
function add(a:number,b:number,c?:number,d?:number):number  
{  
    if(d) return a+b+c+d;  
    if(c) return a+b+c;  
    return a+b;  
}
```

```
console.log(add(10,20));
console.log(add(10,20,30));
console.log(add(10,20,30,40));
```

---

**eg22.ts**

```
function add(a:number=0,b:number,c:number):number
{
return a+b+c;
}
```

```
console.log(add(undefined,10,20));
console.log(add(10,20,30));
```

---

**eg23.ts**

```
function add(...numbers:number[]): number{
var x:number=0;
for(var i:number=0;i<numbers.length;i++)
{
x+=numbers[i];
}
return x;
}
console.log(add(10,20,30,4));
console.log(add(2,3,4));
```

---

**eg24.ts**

```
function add(...numbers:number[]): number{
var x:number=0;
for(var num of numbers)
{
x+=num;
}
return x;
}
console.log(add(10,20,30,4));
console.log(add(2,3,4));
```

---

**eg25.ts**

```
function doSomething(x:number)
{

    if(x==10)
    {
        var y:number=30;
    }
    console.log(y);
}
doSomething(10);
doSomething(20);
```

---



**eg26.ts**

```
// the following code won't compile as
// var keyword provides function level scope
// whereas let keyword provides block level scope
function doSomething(x:number)
{

    if(x==10)
    {
        let y:number=30;
    }
    console.log(y);
}
doSomething(10);
doSomething(20);
```

---

**eg27.ts**

```
var x=[10,20,30];
for(let y of x)
{
    console.log(typeof y);
}
for(let y in x)
{
    console.log(typeof y);
}
```

---

**eg28.ts**

```
function cool()
{
    console.log(this); // refers to instance of window
    console.log(typeof this);
}
class aaa
{
    public cool()
    {
        console.log(this); // refers to instance of aaa
        console.log(typeof this);
    }
}
cool();
console.log("-----");
var a=new aaa;
a.cool();
```

---

**eg29.ts**

// This code will compile, but will produce incorrect output

```
class Counter
{
  private x:number;
  constructor(x:number)
  {
    this.x=x;
  }
  public getCounter(){
    return function(){
      let j=this.x;
      this.x++;
      return j;
    }
  }
}
```

```
var c=new Counter(1);
var counter1=c.getCounter();
console.log(counter1());
console.log(counter1());
console.log(counter1());
```

---

**eg30.ts**

// This code will compile, but will produce incorrect output

```
class Counter
{
  private x:number;
  constructor(x:number)
  {
    this.x=x;
  }
  public getCounter(){
    return ()=>{
      let j=this.x;
      this.x++;
      return j;
    }
  }
}
```

```
var c1=new Counter(1);
var next1=c1.getCounter();
console.log(next1());
console.log(next1());
console.log(next1());
```

```
var c2=new Counter(1001);
var next2=c2.getCounter();
console.log(next2());
console.log(next2());
console.log(next2());
console.log(next1());
console.log(next2());
```

---

**eg31.ts**

```
var lambda1=(a:number,b:number)=>{
    return a+b;
}

var lambda2=(a:number,b:number)=>a+b;

console.log(lambda1(10,20));
console.log(lambda2(100,200));
```

---

**eg32.ts**

```
class Student {
    constructor(public rollNumber:number,public firstName:string,public lastName:string)
    {
    }
}

var s1=new Student(101,"Sameer","Gupta");
var {rollNumber,firstName}=s1;
console.log("Roll number : "+rollNumber);
console.log("First name : "+firstName);
var s2=new Student(102,"Rohit","Sharma");
({rollNumber,firstName}=s2);
console.log("Roll number : "+rollNumber);
console.log("First name : "+firstName);
```

---

**eg33.ts**

```
class Student {
    constructor(public rollNumber:number,public firstName:string,public lastName:string,public
age:number)
    {
    }
}

var s1=new Student(101,"Sameer","Gupta",24);
var {rollNumber,firstName}=s1;
console.log("Roll number : "+rollNumber);
console.log("First name : "+firstName);
var s2=new Student(102,"Rohit","Sharma",27);
var {firstName,...r}=s2;
console.log("First name : "+firstName);
```

```
console.log("Last name : "+r.lastName);  
console.log("Age : "+r.age);
```

---

**eg34.ts**

```
var e:number=100;  
var f:number=200;  
var g:number;  
g=e;  
e=f;  
f=g;  
console.log(e);  
console.log(f);  
var x:number=10;  
var y:number=20;  
[x,y]=[y,x];  
console.log(x);  
console.log(y);
```

---

**eg35.ts**

```
var x=[10,20,30,40];  
var [e,f,...r]=x;  
console.log(x);  
console.log(e);  
console.log(f);  
console.log(r);
```

---

**eg36.ts**

```
var x=[10,20,30,40];  
var [e,...r]=x;  
console.log(x);  
console.log(e);  
console.log(r);
```

---

**eg37.ts**

```
function add(a,b,c)  
{  
    return a+b+c;  
}  
var x=[10,20,30];  
console.log(add.apply(null,x));
```

---

**eg38.ts**

```
class Item  
{  
    public name:string;  
    public price:number;  
}  
function purchaseCheapest(...x)  
{  
    let cheapest=x[0];  
    for(let i=1;i<x.length;i++)
```

```

    {
      if(x[i].price<cheapest.price) {
        cheapest=x[i];
      }
    }
    this.name=cheapest.name;
    this.price=cheapest.price;
  }
  var items=[];
  items[0]=new Item;
  items[0].name="Compass";
  items[0].price=430;
  items[1]=new Item;
  items[1].name="Mouse";
  items[1].price=150;
  items[2]=new Item;
  items[2].name="Keyboard";
  items[2].price=220;
  var item=new Item;
  purchaseCheapest.apply(item,items);
  console.log("Purchased : "+item.name+" for Rs."+item.price);

```

---

**eg39.ts**

```

var a="Ujjain";
var b=`${a} is the city of Gods.`;
console.log(b);
var c=`Thinking Machines \
is in ${a}`;
console.log(c);
var d=`Ujjain is a
Cool place`;
console.log(d);

```

---

**eg40.ts**

```

// This is something special and cool
// Consider that we have a HTML file
// we need to present its contents as it is on WEBPage
// such kind of code will come in handy
var a="Thinking Machines";
var b="Ujjain";
var c=myHTMLProcessor`<div><B>${a}</b> is in <B><u>${b}</b></u>`;
function myHTMLProcessor(v,...p)
{
  let s="";
  console.log(v);
  console.log(p);
  console.log("-----");
  for(let i=0;i<p.length;i++)

```

```
{
  s=s+v[i];
  s=s+p[i];
  s=s.replace(/&/g,"&");
  s=s.replace(/"/g,"&quot;");
  s=s.replace(/'/g,"&#39;");
  s=s.replace(/</g,"&lt;");
  s=s.replace(/>/g,"&gt;");
  // we can put the key vals in somekind of DS and iterate that DS while replacing all occurrences of
  // the keys with values.
}
s=s+v[v.length-1];
return s;
}
console.log(c);
```

---

**eg41.ts**

```
interface Institute {
  name: string;
  city: string;
}
var k='{"name":"Thinking Machines","city":"Ujjain"}';
console.log(k);
var m:Institute=JSON.parse(k);
console.log(m);
console.log(m.name);
console.log(m.city);
```

---

**eg42.ts**

```
interface Student {
  rollNumber:number;
  name:string;
}
class SchoolStudent implements Student{
  rollNumber:number;
  name:string;
}
var s1:Student=new SchoolStudent;
s1.rollNumber=101;
s1.name="Sameer";
var k=JSON.stringify(s1);
console.log(s1);
console.log(k);
var m='{"rollNumber":200,"name":"Abhi","age":34}';
var s2:Student=JSON.parse(m);
console.log(s2.rollNumber);
console.log(s2.name);
// console.log(s2.age); // this won't compile
```

---