Method Parameters

- Every Parameter is mandatory.
- You can define optional parameters by using “null reference character [?]”.
- You can use “undefined” type to verify, value defined or not.

  Ex:

```
class Product
{
   public Details(Name:string, Price:number, Stock?:boolean){
       if(Stock==undefined){
console.log(`Name=${Name}\nPrice=${Price}`);
       } else {
console.log(`Name=${Name}\nPrice=${Price}\nStock=${Stock}`);
       }
   }
}
let tv = new Product;
tv.Details("Samsung TV",56000.55, true);
```

Note: A required parameter can’t follow after optional parameter.

Syntax:

method(**param?**:string, param:number); // invalid

- All optional parameters must be last parameters in formal list.
- A method supports maximum 1024 params.
- Method parameter can be any type
    - Array
    - Object
    - Function
    - String
    - Number
    - Boolean etc.

Ex: Object as Parameter

```
class Product
{
   public Details(product:any){
     for(var property in product) {
        console.log(`${property} : ${product[property]}`);
     }
   }
}
```

```
let tv = new Product;
tv.Details({Name: "Samsung TV", Price: 45000.55, Stock:true});
console.log(`------------------`);
let shoe = new Product;
let nike = {
    Name: "Nike Casuals",
    Price: 45000.55
}
shoe.Details(nike);
```

Ex: Array as parameter

```
class Demo
{
    PrintList(list:string[]){
        for(var item of list) {
            console.log(item);
        }
    }
}
```

```
let obj = new Demo;
obj.PrintList(new Array("TV", "Mobile"));
obj.PrintList(["Nike Casuals", "Lee Boot"]);
let fashion = ["Shirt", "Jeans"];
obj.PrintList(fashion);
```

Ex: Function Parameter

```
class Login
{
   public VerifyDetails(password:string, success:any,
failure:any){
      if(password=="admin12") {
         success();
      } else {
         failure();
      }
   }
}
let obj = new Login;
```

```
obj.VerifyDetails("admin123",function(){console.log('Login Success')}, function(){console.log('Invalid Password')});
```

Ex: Array of Objects

```
class Product
{
    public Details(products:any[]) {
        for(var item of products) {
            console.log(`${item.Name} - ${item.Price}`);
        }
    }
}
let tv = new Product;
tv.Details([{Name:"TV", Price: 45000.55}, {Name:"Mobile", Price:6000.55}]);
```

ES5 introduced “rest” parameters

- Single formal parameter can handle multiple actual values.
- It is defined by using “…paramName”
- Every method can have only one rest parameter.

- Rest parameter must be the last parameter in formal list.

Ex:

```
class Product
{
  public PrintList(...list:any){
    for(var item of list) {
       console.log(item);
    }
  }
}
let obj = new Product;
obj.PrintList("Samsung TV", "Mobile", "Nike Casual", "Lee Boot");
```

Method with Return Type

- Why we need a method to return value?
  To build an expression.
- Expression performs specified operation and returns a value.
- You can build dynamic expressions by using method with return value.

- You can use the method reference memory for storing a value.

FAQ:

What is “void”?

- It is used to discard the returned value.
- It will not configure method as a reference to store value.

Can a void method use “return” keyword?

- Yes

What is the purpose of return keyword in void method?

- To terminate the execution

Can a method have multiple return types defined?

- Yes [Type Script support union of types]

Ex:

```
class Product
{
  public Print(value:string|number):string | number {
    if((typeof value)=="string") {
      return `Hello ! ${value}`;
    } else {
```

```
        return  value;
      }
  }
}
```

Ex:

```
class Service
{
    public Captcha():string {
        var a = Math.random() * 10;
        var b = Math.random() * 10;
        var c = Math.random() * 10;
        var d = Math.random() * 10;
        var e = Math.random() * 10;
        var f = Math.random() * 10;
        var code = `${Math.round(a)} ${Math.round(b)}
${Math.round(c)} ${Math.round(d)} ${Math.round(e)}
${Math.round(f)}`;
        return code;
    }
}
let obj = new Service;
```

```
console.log(obj.Captcha());
```

Lambda Notation for Method

- It is a short hand technique of configuring the method.
- It allows to minify your code.
- Lambda expression comprises of entities

  ( ) - Defines the parameters

  => - Return value or statements to execute

  { } - Configures multiple statements

Traditional Approach

```
public Print(username:string):string {
  return “Hello ! “ + username;
}
```

LAMBDA Approach

```
var Print = username:string => “Hello !” + username;
```

Ex:

```
class Service
{
```

```
  public Hello(username:string):string {
     return `Hello ! ${username}`;
  }
  public Welcome = (username:string) => `Welcome !
${username}`;
  public Print = () => console.log("This is print
method");
  public list = (data:any[]) => {
     for(var item of data) {
        console.log(item);
     }
  }
}
let obj = new Service;
console.log(obj.Welcome("john"));
obj.Print();
obj.list(["TV", "Mobile"]);
```

Ex:

```
let products = [
```

```
    {Name: "Samsung TV", Category:"Electronics", Price: 45000.55},
    {Name: "Nike Casuals", Category: "Footwear", Price: 3000.44},
    {Name: "Earpods", Category: "Electronics", Price: 4200.44}
];
let electronics:any[] = products.filter(function(product){
    return product.Category=="Electronics"
});
let footwear:any[] = products.filter(product=> product.Category=="Footwear");

let electronicsCount:number = products.filter(product=>product.Category=="Electronics").length;

console.log(electronics);
console.log(footwear);
console.log(`Total No of Electronics: ${electronicsCount}`);
```

Constructor in a Class

- Constructor is a design pattern
- It is a process of constructing an object for class.
- Constructor is used for instantiation. [Creating of Object]
- You can prevent instantiation by using a private constructor.
- Constructor is a special type of subroutine [method], which executes automatically.
- Constructor is loaded into memory at the time of loading class into memory.
- Constructor is anonymous, without name.
- Class name is used for constructor.