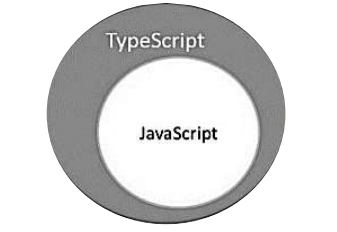
1. JavaScript was introduced as a language for the client side.
2. as JavaScript code grows, it tends to get messier, making it difficult to maintain and reuse the code
3. What is TypeScript? 🡺By definition, “TypeScript is JavaScript for application-scale development.”
4. TypeScript is a strongly typed, object oriented, compiled language.
5. It was designed by Anders Hejlsberg (designer of C#) at Microsoft.
6. In other words, TypeScript is JavaScript plus some additional features.
7. Features of TypeScript
8. **TypeScript is just JavaScript**🡺Hence; you only need to know JavaScript to use TypeScript.
9. **TypeScript supports other JS libraries**🡺Compiled TypeScript can be consumed from any JavaScript code. TypeScript-generated JavaScript can reuse all of the existing JavaScript frameworks, tools, and libraries.

22/5/2017

1. TypeScript is a typed superset of JavaScript that compiles to plain JavaScript.
2. TypeScript is pure object oriented with classes, interfaces and statically typed like C# or Java.

Note🡺 **what is the difference between Client side Scripting language and Server side Scripting language?**

1. 

What is TypeScript?

TypeScript is a strongly typed, object oriented, compiled language. It was designed by **Anders Hejlsberg** (designer of C#) at Microsoft. TypeScript is both a language and a set of tools. TypeScript is a typed superset of JavaScript compiled to JavaScript.

## Features of TypeScript

1. **TypeScript is just JavaScript**.
2. **TypeScript supports other JS libraries**.
3. **JavaScript is TypeScript**. This means that any valid **.js** file can be renamed to **.ts** and compiled with other TypeScript files.
4. **TypeScript is portable**. TypeScript is portable across browsers, devices, and operating systems. It can run on any environment that JavaScript runs on. Unlike its counterparts, TypeScript doesn’t need a dedicated VM or a specific runtime environment to execute.

### TypeScript and ECMAScript

1. The ECMAScript specification is a standardized specification of a scripting language. There are six editions of ECMA-262 published.

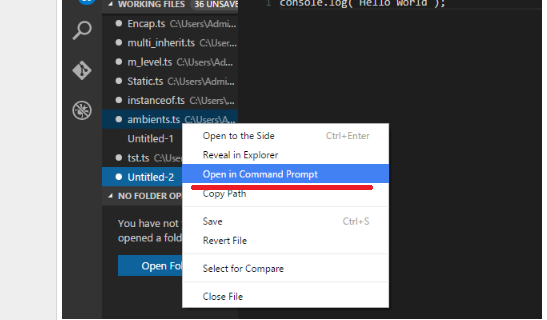


1. **TypeScript adopts its basic language features from the ECMAScript5 specification, i.e., the official specification for JavaScript. TypeScript language features like Modules and class-based orientation are in line with the EcmaScript 6 specification. Additionally, TypeScript also embraces features like generics and type annotations that aren’t a part of the EcmaScript6 specification.**
2. Why Use TypeScript?
3. **Compilation**
4. **Strong Static Typing**
5. **TypeScript supports type definitions for existing JavaScript libraries. TypeScript Definition file (with .d.ts extension) provides definition for external JavaScript libraries. Hence, TypeScript code can contain these libraries.**
6. **TypeScript supports Object Oriented Programming concepts like classes, interfaces, inheritance, etc.**

### Declaration Files

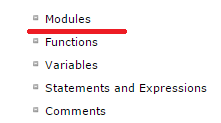
1. When a TypeScript script gets compiled, there is an option to generate a **declaration file** (with the extension **.d.ts**) that functions as an interface to the components in the compiled JavaScript.
2. The declaration files (files with **.d.ts** extension) provide intellisense for types, function calls, and variable support for JavaScript libraries like jQuery, MooTools, etc.

**Intellisense🡺Intelligent code completion**[**[1]**](https://en.wikipedia.org/wiki/Intelligent_code_completion#cite_note-1)[**[2]**](https://en.wikipedia.org/wiki/Intelligent_code_completion#cite_note-2)**is a context-aware**[**code completion**](https://en.wikipedia.org/wiki/Code_completion)**feature in some programming environments that speeds up the process of coding applications by reducing typos and other common mistakes**

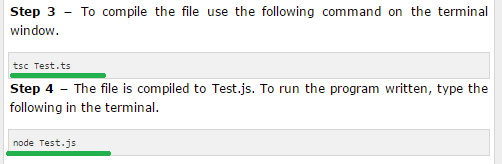


# TypeScript - Basic Syntax

1. A TypeScript program is composed of –

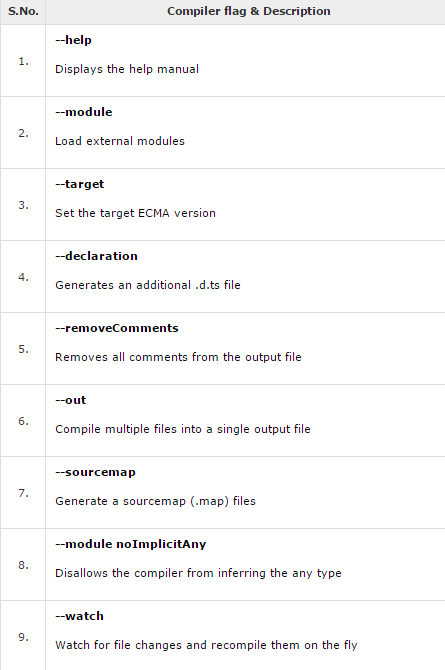


1. Steps to Compile and Run the TS file in Command Prompt is🡺



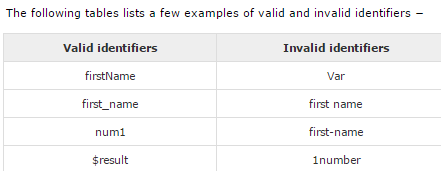
## 10. Compiler Flags

Compiler flags enable you to change the behavior of the compiler during compilation. Each compiler flag exposes a setting that allows you to change how the compiler behaves.



1. Multiple files can be compiled at once🡺
2. **Identifiers in TypeScript**
3. Identifiers are names given to elements in a program like variables, functions etc. The rules for identifiers are −

* Identifiers can include both, characters and digits. However, the identifier cannot begin with a digit.
* Identifiers cannot include special symbols except for underscore (\_) or a dollar sign ($).
* Identifiers cannot be keywords.
* They must be unique.
* Identifiers are case-sensitive.
* Identifiers cannot contain spaces.

1. 

## TypeScript ─ Keywords

1. 

### Whitespace and Line Breaks 🡺TypeScript ignores spaces, tabs, and newlines that appear in programs. You can use spaces, tabs, and newlines freely in your program

### TypeScript is Case-sensitive

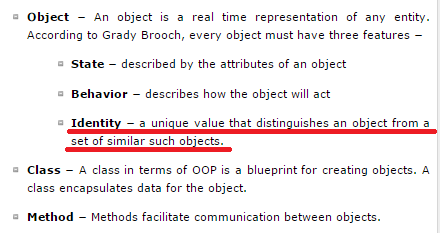
### Semicolons are optional

**Note🡺 a single line can contain multiple statements. However, these statements must be separated by a semicolon.**

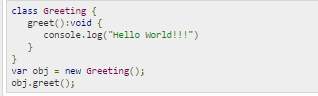
### Comments in TypeScript🡺

## TypeScript and Object Orientation🡺

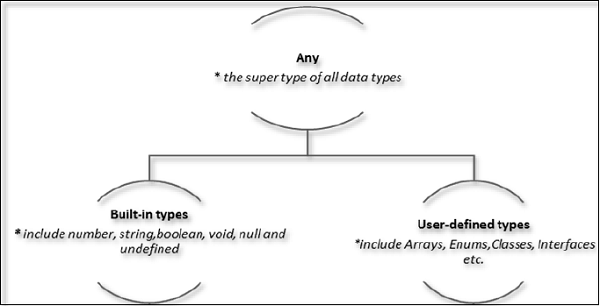
TypeScript is Object-Oriented JavaScript. Object Orientation is a software development paradigm that follows real-world modelling. Object Orientation considers a program as a collection of objects that communicate with each other via mechanism called methods. TypeScript supports these object oriented components too.



1. A simple type Script Example🡺



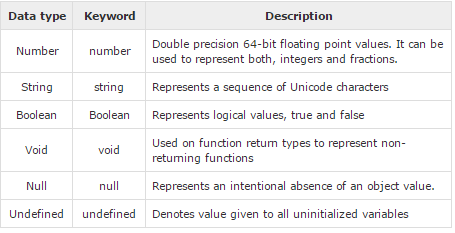
# TypeScript - Types

1. The Type System represents the different types of values supported by the language. The Type System checks the validity of the supplied values, before they are stored or manipulated by the program. This ensures that the code behaves as expected.
2. 

## The Any type

* 1. **The any data type is the super type of all types in TypeScript. It denotes a dynamic type. Using the any type is equivalent to opting out of type checking for a variable.**

## Built-in types

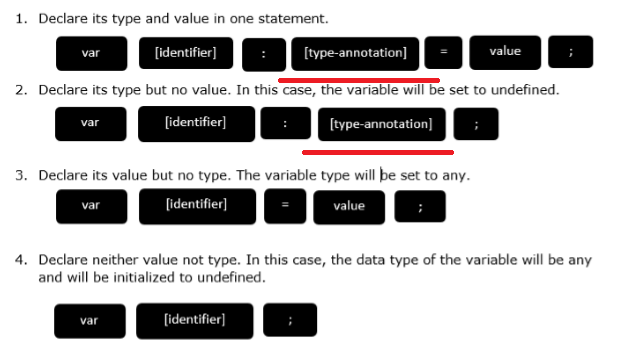
1. 
2. **Note − There is no integer type in TypeScript and JavaScript.**

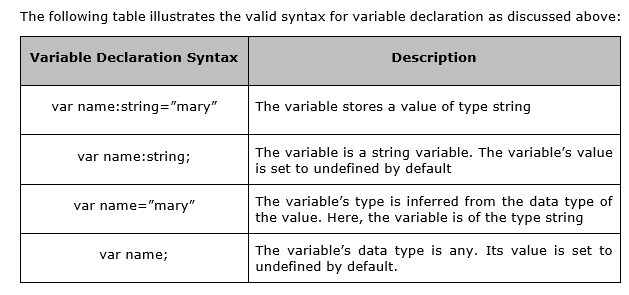
### Null and undefined ─ Are they the same?

1. The null and undefined cannot be used to reference the data type of a variable. They can only be assigned as values to a variable.
2. However, *null and undefined are not the same*. A variable initialized with undefined means that the variable has no value or object assigned to it while null means that the variable has been set to an object whose value is undefined.

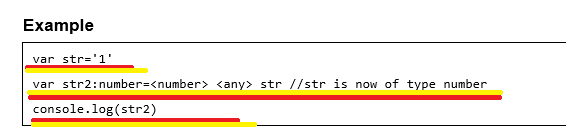
# TypeScript - Variables

1. A variable, by definition, is “a named space in the memory” that stores values. In other words, it acts as a container for values in a program
2. Variable Declaration in TypeScript🡺



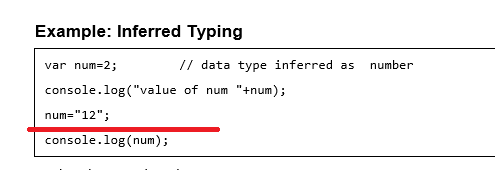
1. 
2. 

# Type Assertion in TypeScript

1. **TypeScript allows changing a variable from one type to another. TypeScript refers to this process as Type Assertion.**
2. The syntax is to put the target type between < > symbols and place it in front of the variable or expression
3. 

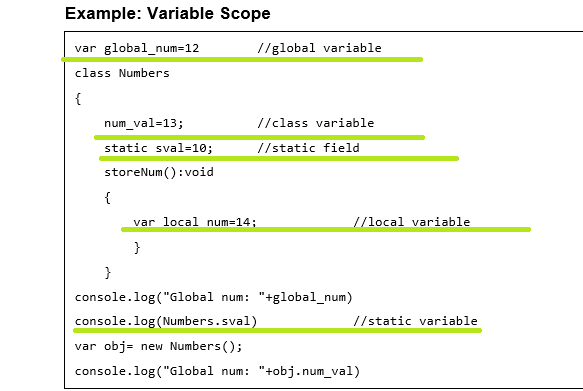
# Inferred Typing in TypeScript

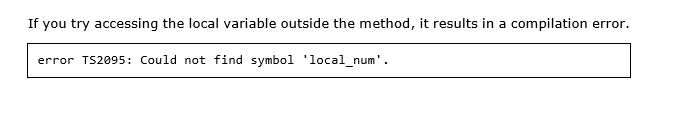
1. TypeScript also encourages dynamic typing of variables. This means that, TypeScript encourages declaring a variable without a type. In such cases, the compiler will determine the type of the variable on the basis of the value assigned to it. TypeScript will find the first usage of the variable within the code, determine the type to which it has been initially set and then assume the same type for this variable in the rest of your code block.



1. TypeScript Variable Scope🡺The availability of a variable within a program is determined by its scope.

* Global Scope ─ Global variables are declared outside the programming constructs. These variables can be accessed from anywhere within your code.
* Class Scope ─ These variables are also called fields. Fields or class variables are declared within the class but outside the methods. These variables can be accessed using the object of the class. Fields can also be static. Static fields can be accessed using the class name.
* Local Scope ─ Local variables, as the name suggests, are declared within the constructs like methods, loops etc. Local variables are accessible only within the construct where they are declared.

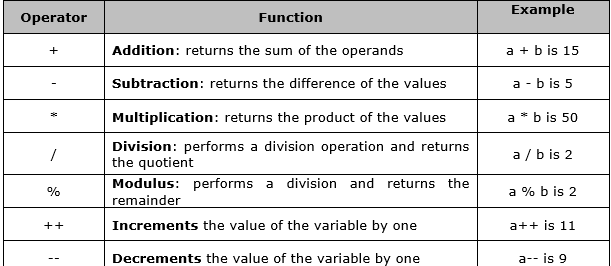
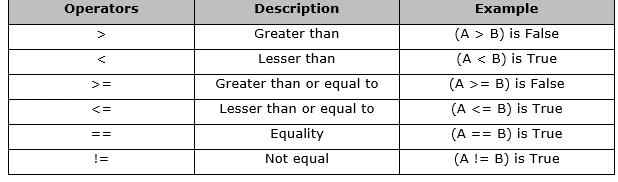
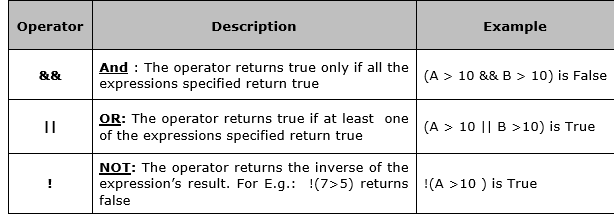


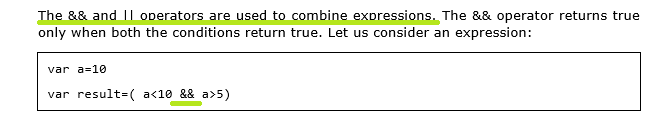
1. 

# TypeScript ─ Operators

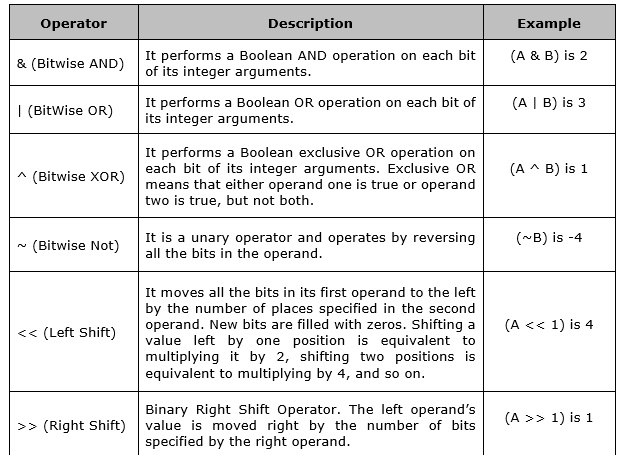
1. **An operator defines some function that will be performed on the data.** The data on which operators work are called operands.
2. The major operators in TypeScript can be classified as:

* Arithmetic operators
* Logical operators
* Relational operators
* Bitwise operators
* Assignment operators
* Ternary/conditional operator
* String operator
* Type Operator

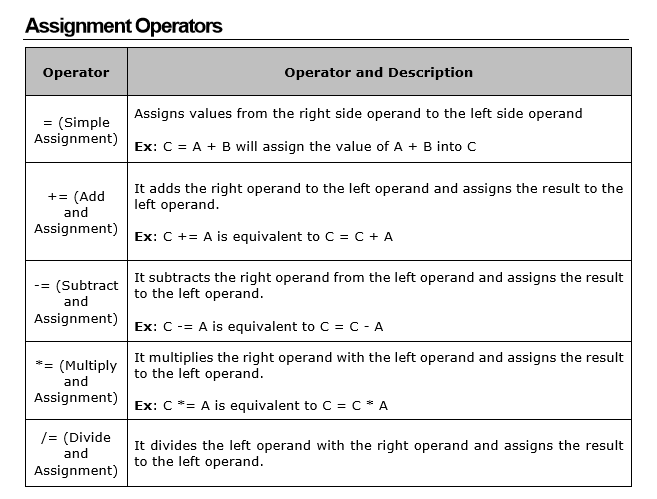
1. 
2. Relational Operators🡺
3. Logical Operators🡺
4. Short-circuit Operators (&& and ||) 🡺



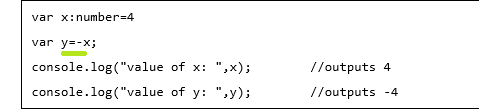
1. Bitwise Operators

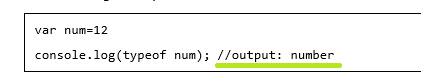


1. Assignment Operators



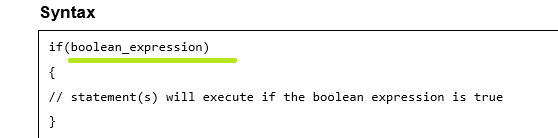
1. Miscellaneous Operators
2. The negation operator (-)

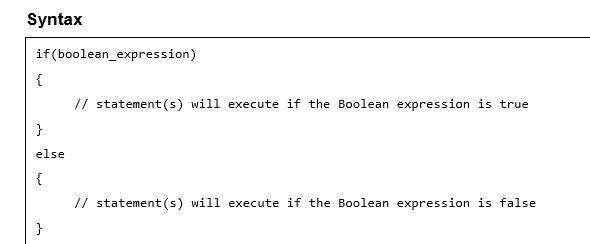


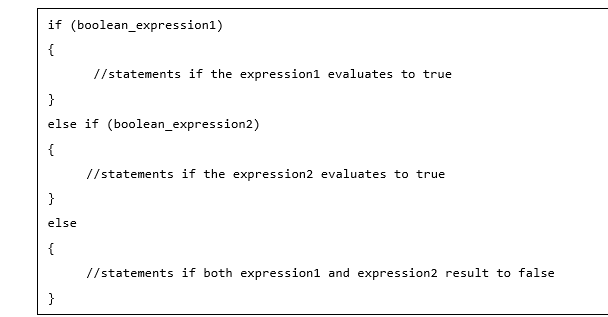
1. String Operators: Concatenation operator (+)
2. Conditional Operator (?) 🡺 Test\_Condition ? expr1 : expr2
3. Type Operators
4. typeof operator 🡺 It is a unary operator. This operator returns the data type of the operand. Take a look at the following example:
5. instanceof 🡺**This operator can be used to test if an object is of a specified type or not. The use of instanceof operator is discussed in the chapter classes.**

# TypeScript ─ Decision Making

1. **The if Statement** 🡺

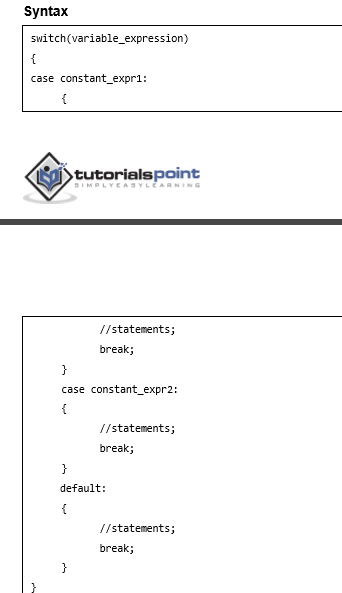


1. **The if…else Statement** 🡺
2. The else…if Ladder 🡺

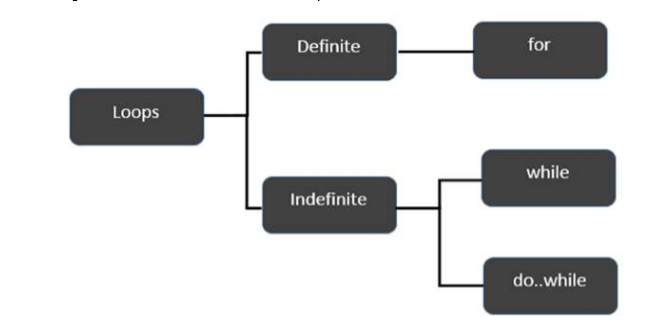
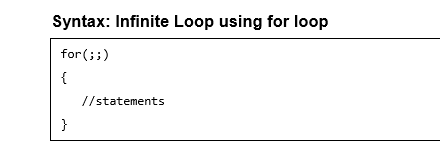
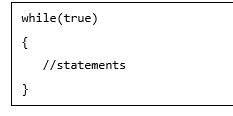


1. **The switch…case Statement**

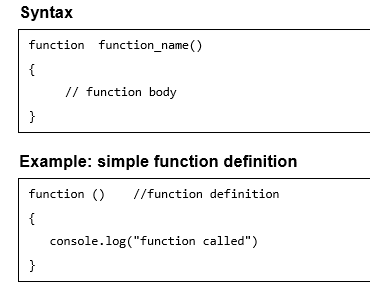
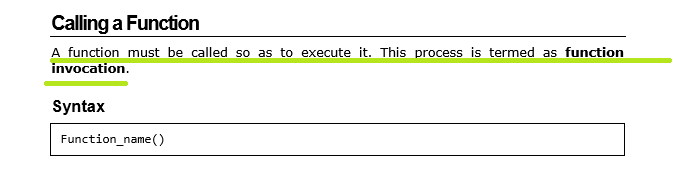
The switch statement evaluates an expression, matches the expression’s value to a case clause, and executes statements associated with that case.



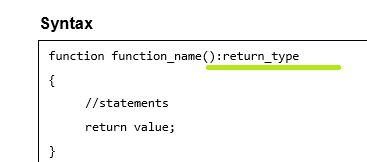
# TypeScript ─ Loops

1. 
2. The break Statement
3. The continue Statement
4. The Infinite Loop🡺An infinite loop is a loop that runs endlessly. The for loop and the while loop can be used to make an endless loop.
5. Example: Infinite loop using while loop🡺 

# TypeScript – Functions

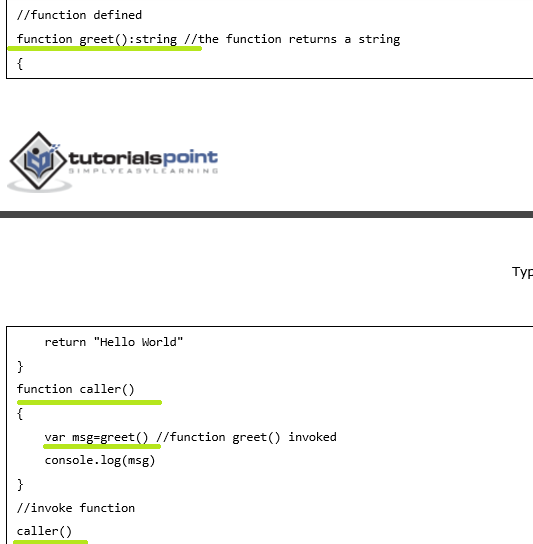
1. Functions are the building blocks of readable, maintainable, and reusable code.
2. A function is a set of statements to perform a specific task.
3. **A function declaration tells the compiler about a function's name, return type, and parameters. A function definition provides the actual body of the function.**
4. 
5. 

Returning Functions

1. Functions may also return value along with control, back to the caller. Such functions are called as returning functions.
2. 

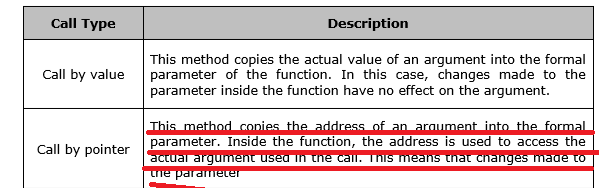
Syntax Explanation🡺

* The return\_type can be any valid data type.
* A returning function must end with a return statement.
* A function can return at the most one value. In other words, there can be only one return statement per function.
* The data type of the value returned must match the return type of the function.

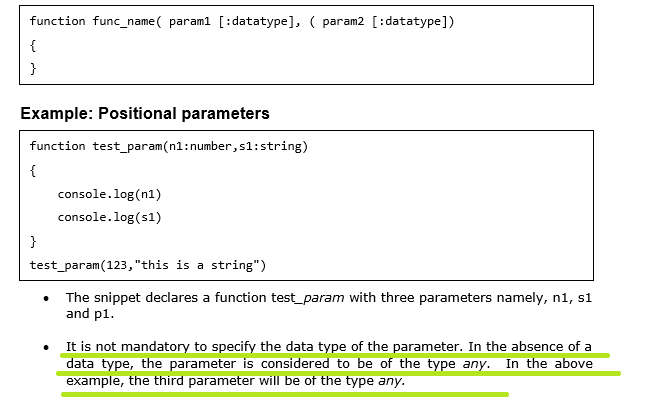
1. 

Parameterized Function

1. While calling a function, there are two ways that arguments can be passed to a function:

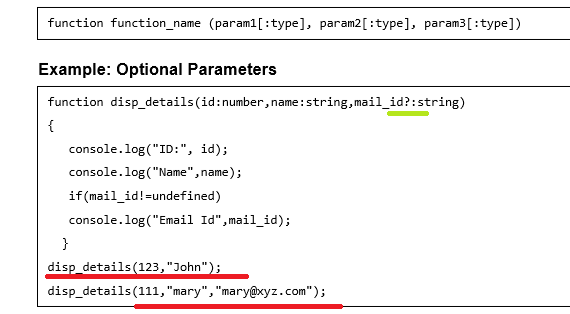


1. Following are the ways in which parameters can be used by functions:
   1. Positional Parameters



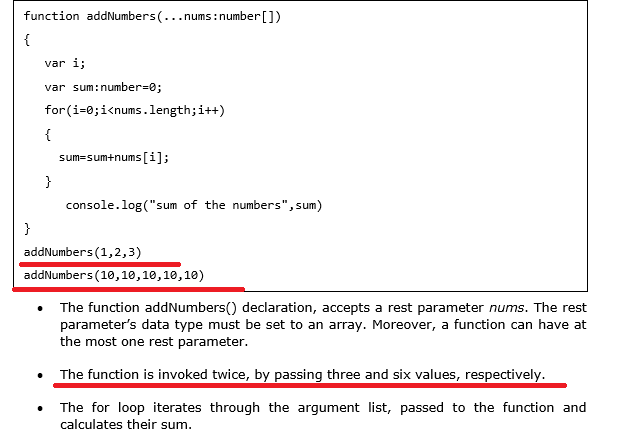
Note 🡺 In case of Positional parameters 🡺**The data type of the value passed must match the type of the parameter during its declaration. In case the data types don’t match, the compiler throws an error.**

* 1. **Optional Parameters**
* **Optional parameters can be used when arguments need not be compulsorily passed for a function’s execution. A parameter can be marked optional by appending a question mark to its name**
* **The optional parameter should be set as the last argument in a function.**
* **The syntax to declare a function with optional parameter is as given below**

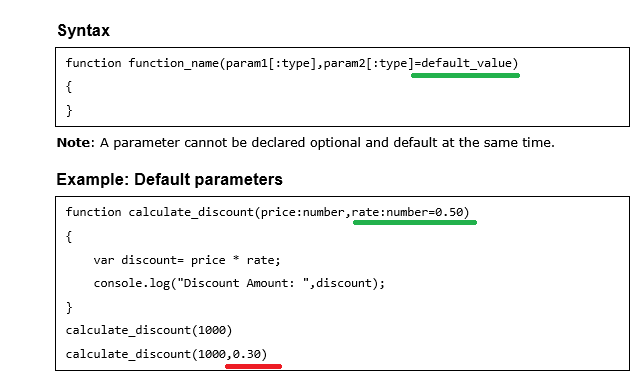


Explanation

* The above example declares a parameterized function. Here, the third parameter, i.e., mail\_id is an optional parameter.
* **If an optional parameter is not passed a value during the function call, the parameter’s value is set to undefined.**
* The function prints the value of mail\_id only if the argument is passed a value.
  1. **Rest Parameters**
* **Rest parameters are similar to variable arguments in Java. Rest parameters don’t restrict the number of values that you can pass to a function.** However, the values passed must all be of the same type. In other words, rest parameters act as placeholders for multiple arguments of the same type
* Example🡺

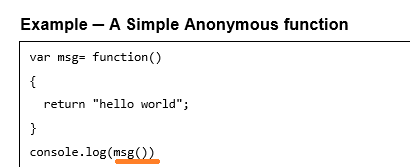


* 1. **Default Parameters**
* **Function parameters can also be assigned values by default.** However, such parameters can also be explicitly passed values.
* Example

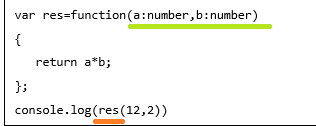


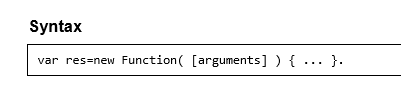
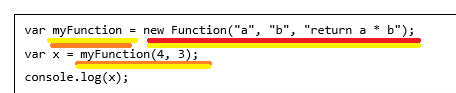
Explanation🡺**The value of the parameter rate is set to 0.50 by default. The same function is invoked, but with two arguments. The default value of rate is overwritten and is set to the value explicitly passed.**

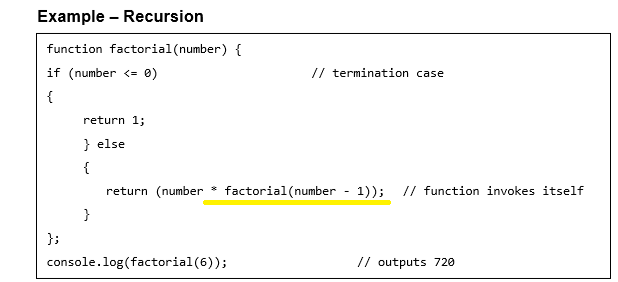
* 1. **Anonymous Function**
* Functions that are not bound to an identifier (function name) are called as anonymous functions. These functions are dynamically declared at runtime.
* Example🡺



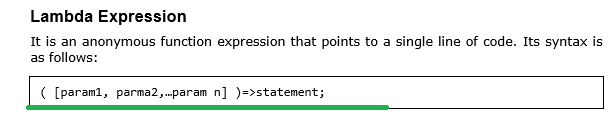
* Example 🡺Anonymous function with parameters



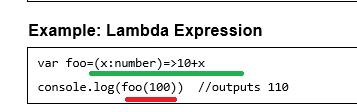
* 1. **The Function Constructor**
* **TypeScript also supports defining a function with the built-in JavaScript constructor called Function ()**
* 
* Example🡺
  1. **Recursion and TypeScript Functions**
* **Recursion is a technique for iterating over an operation by having a function call to itself repeatedly until it arrives at a result**
* Example

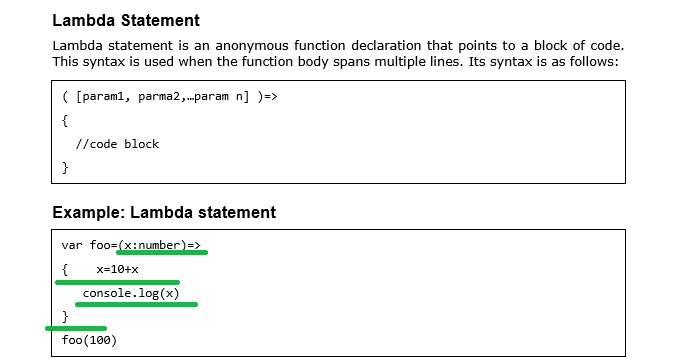


* **Example: Anonymous Recursive Function🡺we Will see this later**
  1. **Lambda Functions**
* **Lambda refers to anonymous functions in programming.**
* These functions are also called as Arrow functions.
* **Lambda Function - Anatomy**
* Parameters: A function may optionally have parameters
* **The fat arrow notation/lambda notation (=>): It is also called as the goes to operator**
* Statements: represent the function’s instruction set
* Syntax



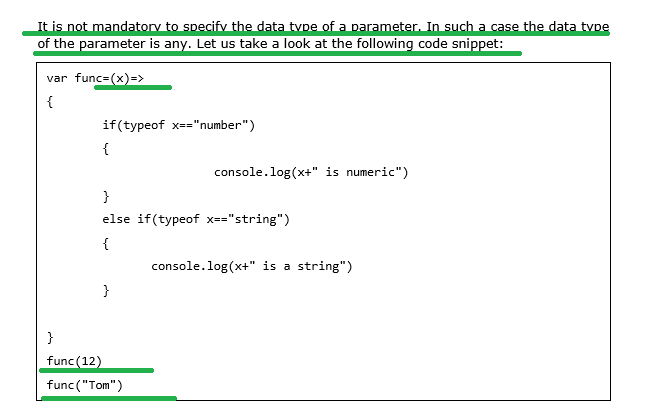
* **Example: Lambda Expression**





**Syntactic Variations**

1. Parameter type Inference



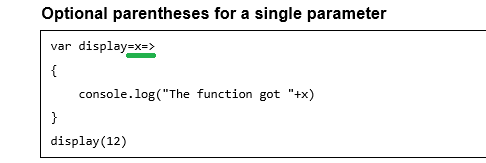
Output🡺

Its output is as follows:

12 is numeric

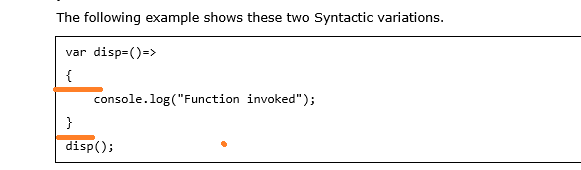
Tom is a string

1. Optional parentheses for a single parameter

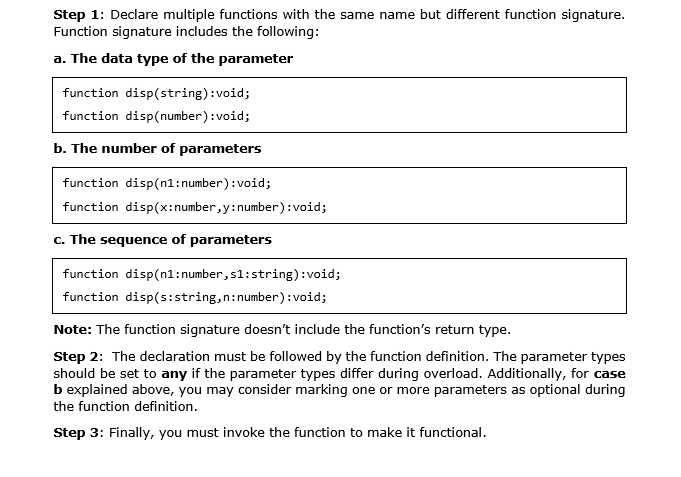


Output🡺

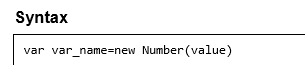
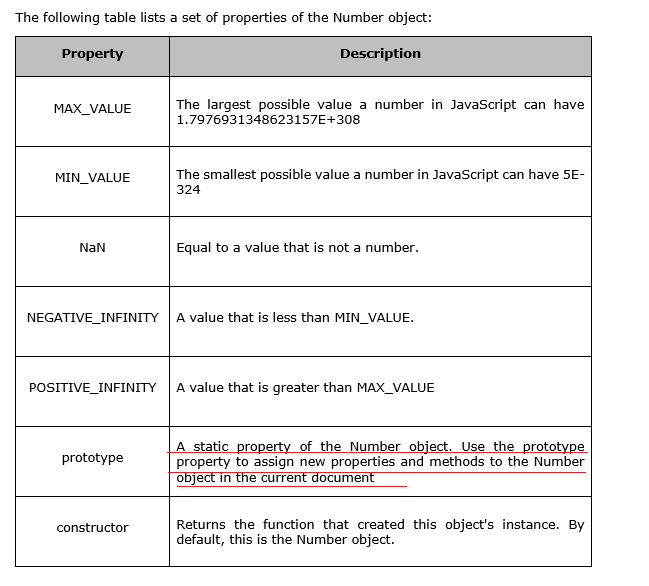
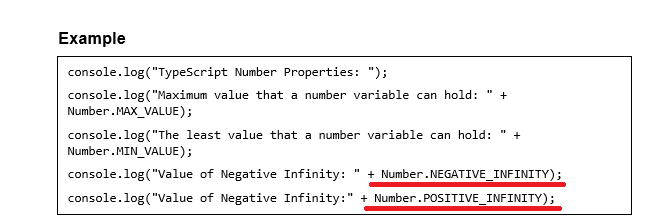
1. Optional braces for a single statement, Empty parentheses for no parameter



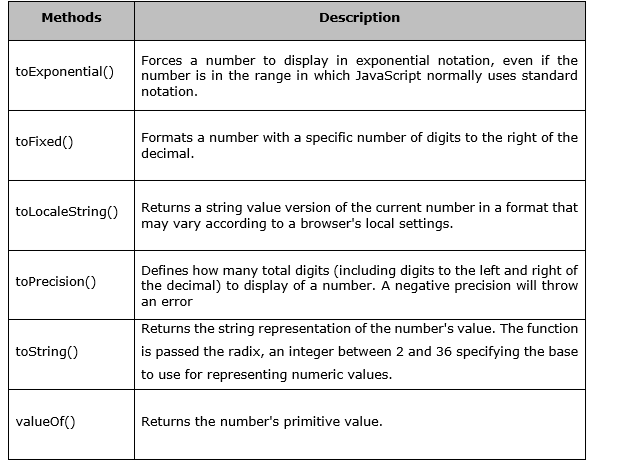
1. Function Overloads



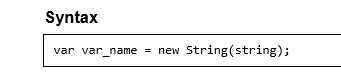
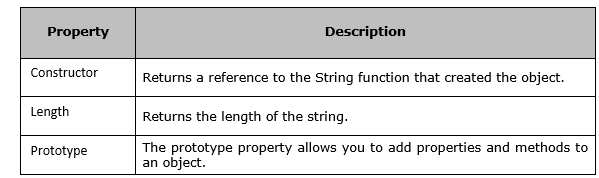
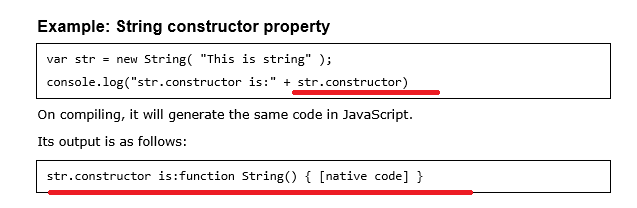
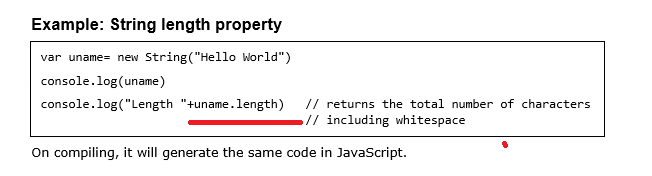
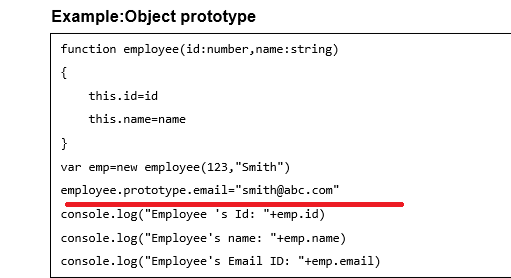
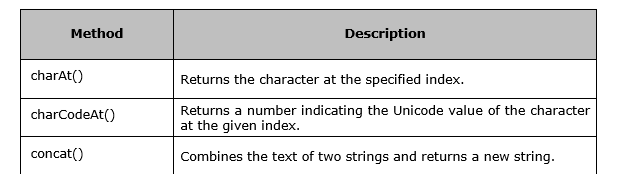
**TypeScript ─ Numbers**

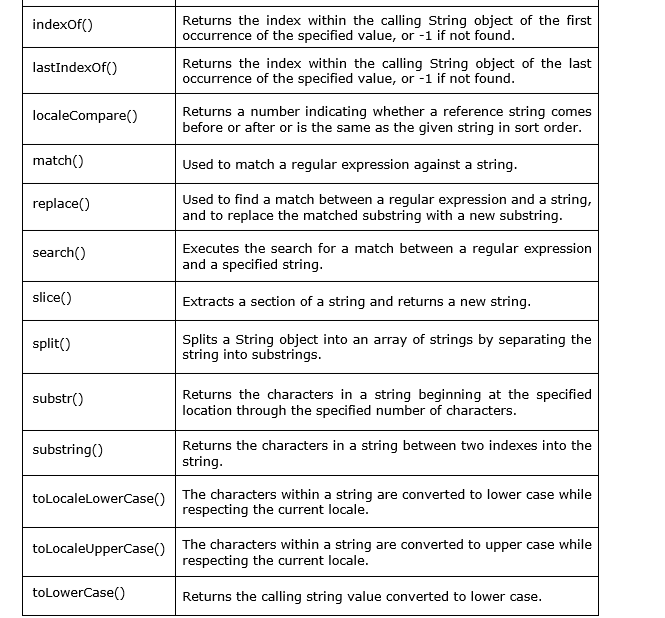
1. A number object converts numeric literal to an instance of the number class. The Number class acts as a wrapper and enables manipulation of numeric literals as they were objects.
2. 
3. **In case a non-numeric argument is passed as an argument to the Number’s constructor, it returns NaN (Not–a–Number)**
4. 
5. 
6. NEEDED A CLEAR EXAMPLE and explanation ON NAN and prototype🡺 we will see this later

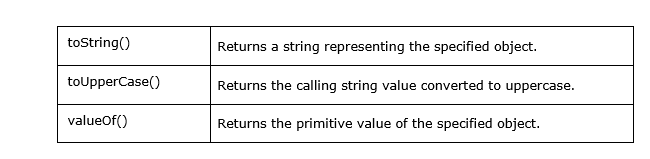
Number Methods



**TypeScript ─ Strings**

1. The String object lets you work with a series of characters.
2. 
3. 
4. 
5. 
6. 
7. String Methods
8. 



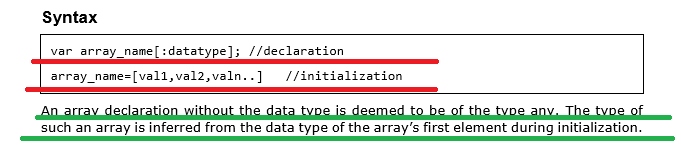


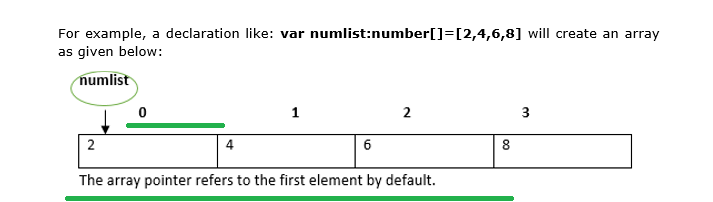
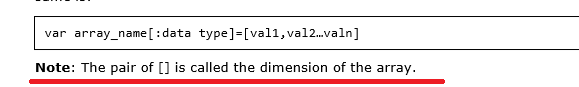
**TypeScript – Arrays**

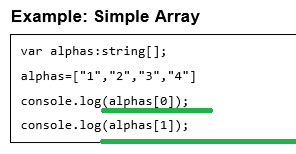
1. Features of an Array

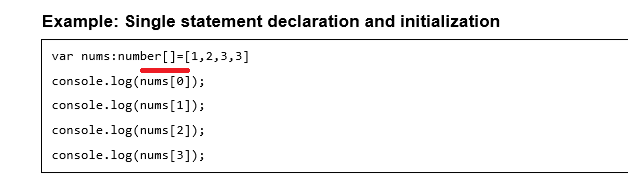
* An array declaration allocates sequential memory blocks.
* Arrays are static. This means that an array once initialized cannot be resized.
* Like variables, arrays too, should be declared before they are used. Use the var keyword to declare an array.
* Array element values can be updated or modified but cannot be deleted.

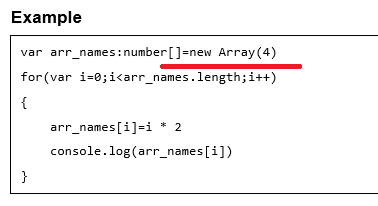
1. Declaring and Initializing Arrays

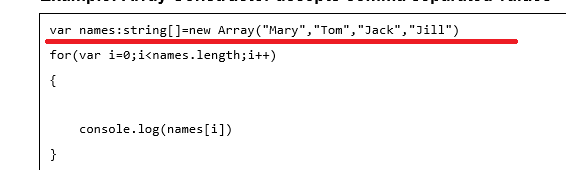


1. Example:
2. Arrays may be declared and initialized in a single statement. The syntax for the same is:
3. Accessing Array Elements example🡺

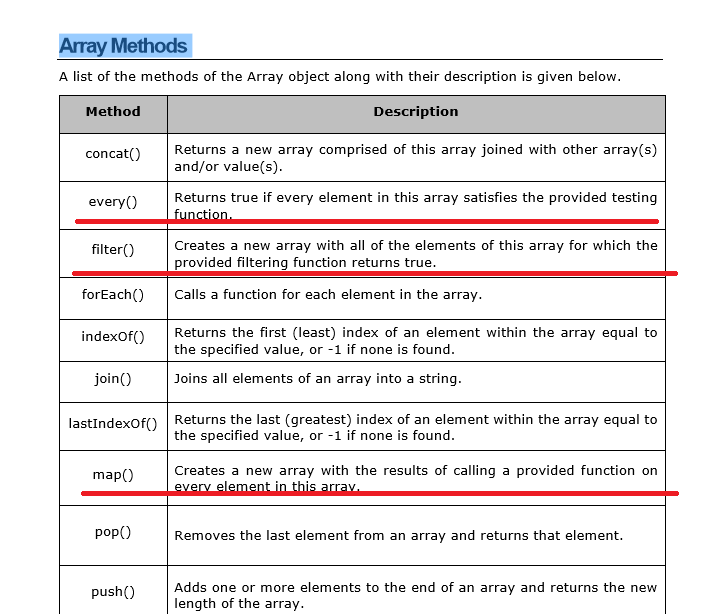


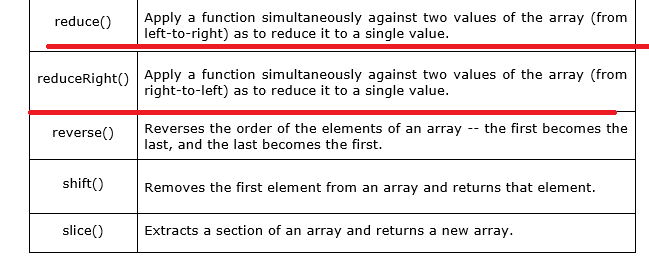
1. 
2. **Array Object**
   1. An array can also be created using the Array object. The Array constructor can be passed

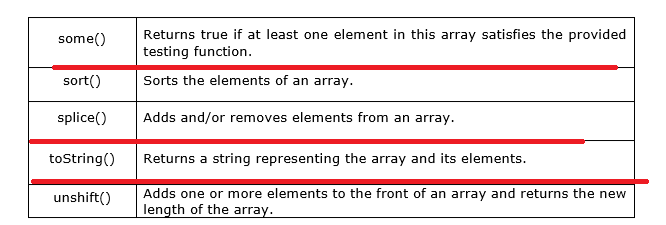
* A numeric value that represents the size of the array or
* A list of comma separated values.
  1. 
  2. Now let’s see the example for list of comma separated values



1. Array Methods



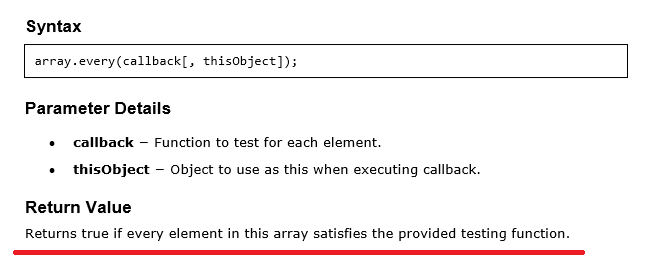


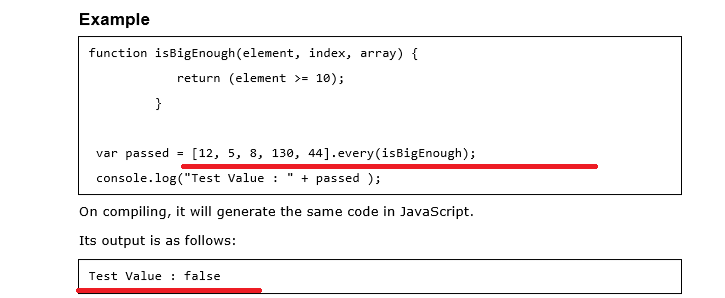


Now let’s see the example for the underlined functions

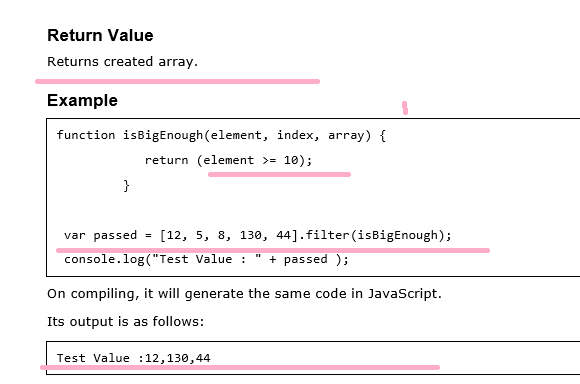
1. Every🡺 every() method tests whether all the elements in an array passes the test implemented by the provided function.

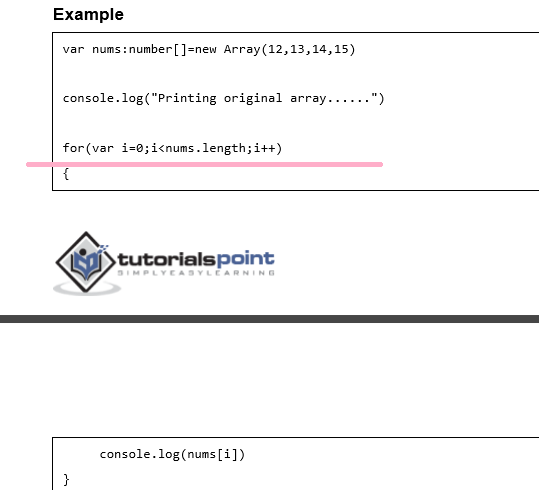
Example🡺Ms🡺Here all the elements of an array is greater than 10 or not is tested

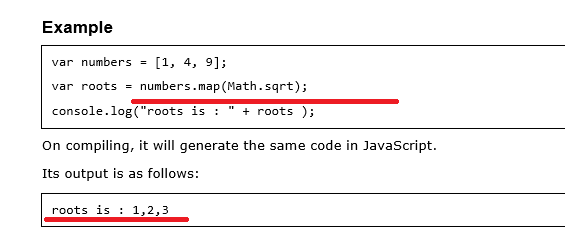




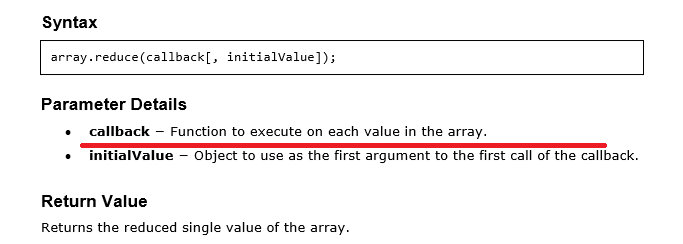
1. Filter🡺 filter() method creates a new array with all elements that pass the test implemented by the provided function.

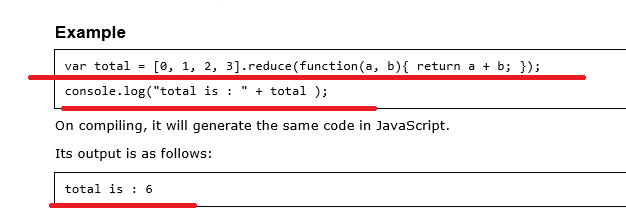


1. Foreach🡺
2. Map🡺 map() method creates a new array with the results of calling a provided function on every element in this array.

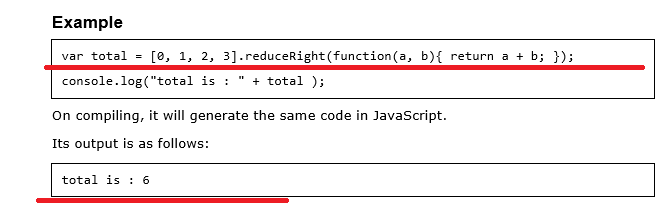


1. Reduce🡺 reduce() method applies a function simultaneously against two values of the array (from left-to-right) as to reduce it to a single value.

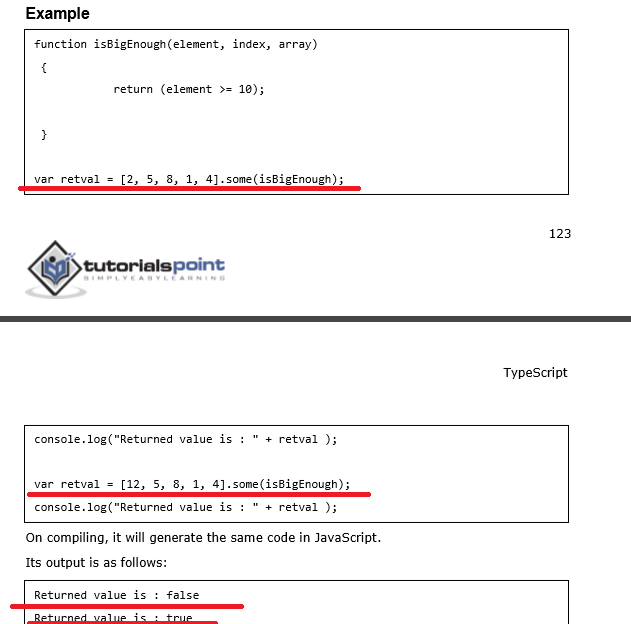




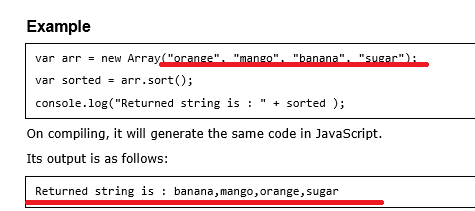
1. reduceRight🡺 reduceRight() method applies a function simultaneously against two values of the array (from right-to-left) as to reduce it to a single value



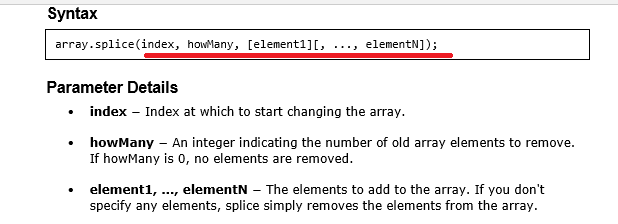
1. some 🡺 some() method tests whether some element in the array passes the test implemented by the provided function.

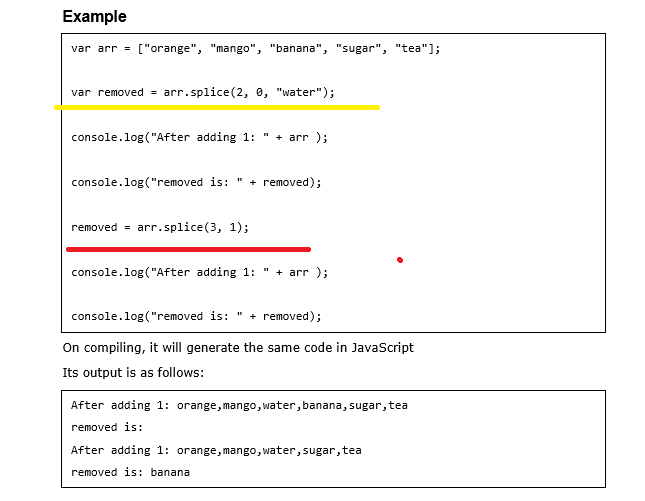


1. sort🡺

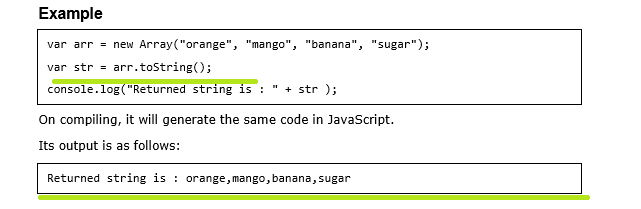


1. splice🡺 splice() method changes the content of an array, adding new elements while removing old elements

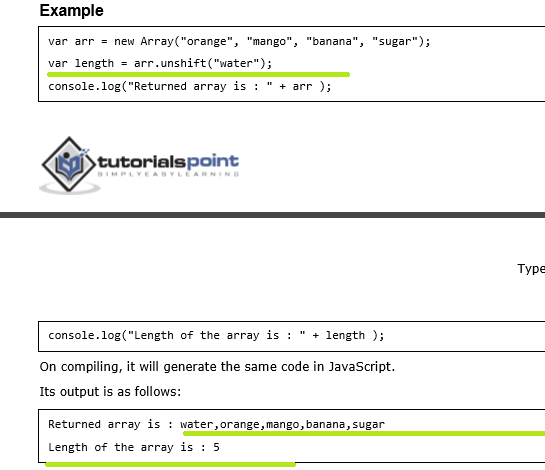




1. toString🡺

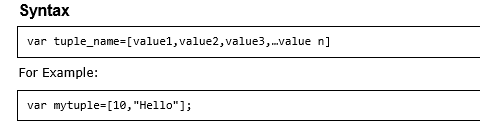


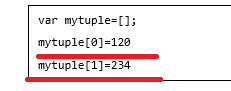
1. unshift🡺 unshift() method adds one or more elements to the beginning of an array and returns the new length of the array.



**Multidimensional Arrays🡺LATER**

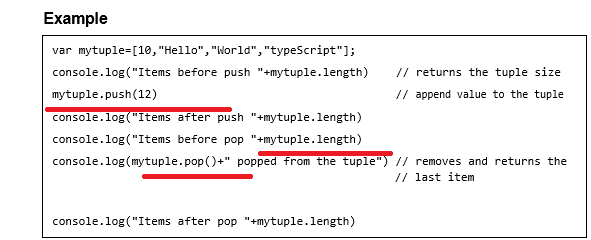
**TypeScript ─ Tuples**

1. **At times, there might be a need to store a collection of values of varied types. Arrays will not serve this purpose. TypeScript gives us a data type called tuple that helps to achieve such a purpose.**
2. It represents a heterogeneous collection of values. In other words, tuples enable storing multiple fields of different types. Tuples can also be passed as parameters to functions.
3. 
4. You can also declare an empty tuple in Typescript and choose to initialize it later.

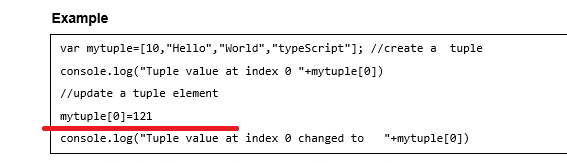


Tuple Operations

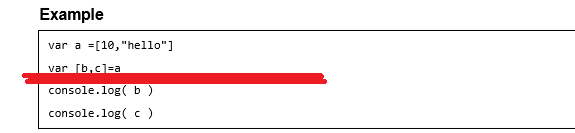
1. Tuples in TypeScript supports various operations like pushing a new item, removing an item from the tuple, etc.

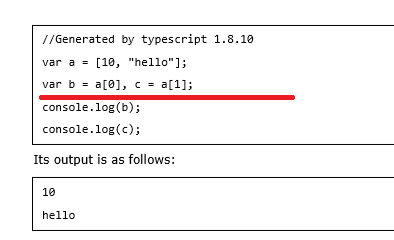


1. Updating Tuples 🡺 Tuples are mutable which means you can update or change the values of tuple elements.



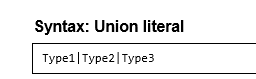
1. Destructuring a Tuple🡺 Destructuring refers to breaking up the structure of an entity. TypeScript supports destructuring when used in the context of a tuple.



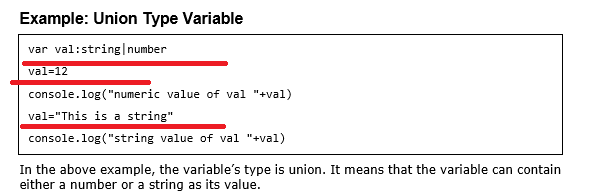
On compiling the following equivalent JS code is generated,

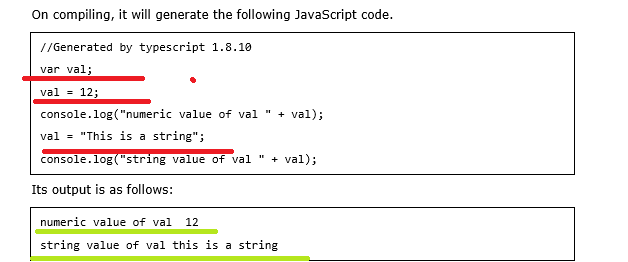
TypeScript – Union

1. TypeScript 1.4 gives programs the ability to combine one or two types. Union types are a powerful way to express a value that can be one of the several types
2. Two or more data types are combined using the pipe symbol (|) to denote a Union Type. In other words, a union type is written as a sequence of types separated by vertical bars.
3. Syntax🡺



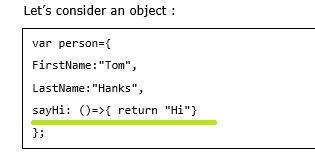
1. Example: Union Type Variable

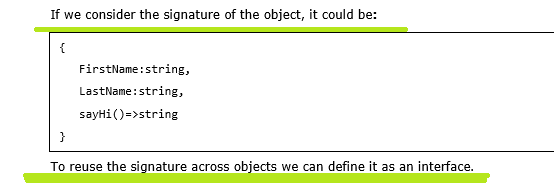


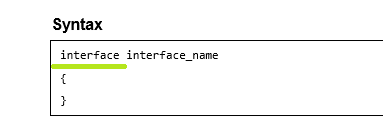


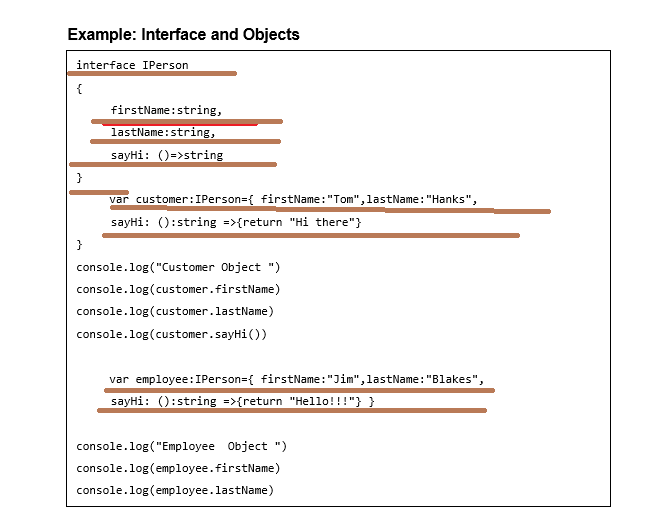
1. **I had covered only basics in this chapter**

TypeScript – Interfaces

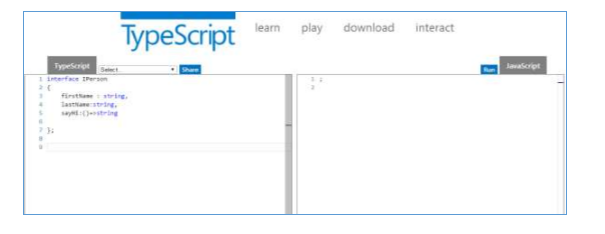
1. **An interface is a syntactical contract that an entity should conform to. In other words, an interface defines the syntax that any entity must adhere to.**
2. Interfaces define properties, methods, and events, which are the members of the interface.
3. 



1. **Declaring Interfaces**
   1. The interface keyword is used to declare an interface. Here is the syntax to declare an interface
   2. 

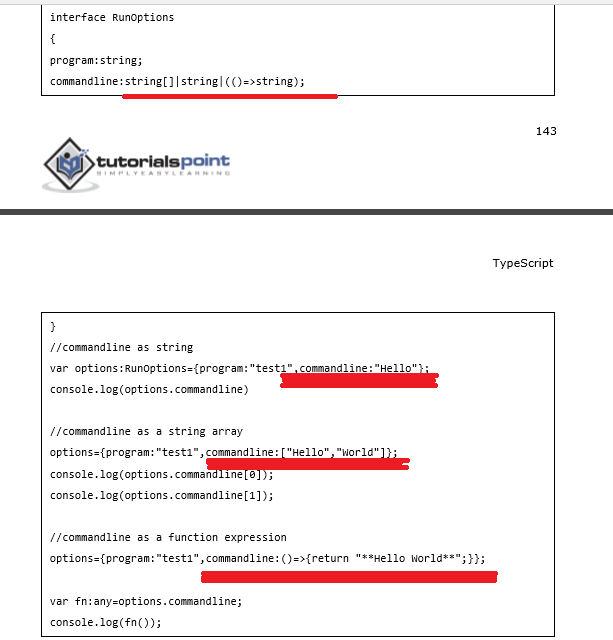


1. **Interfaces are not to be converted to JavaScript. It’s just part of TypeScript. If you see the screen shot of TS Playground tool there is no java script emitted when you declare an interface unlike a class. So interfaces have zero runtime JavaScript impact.**

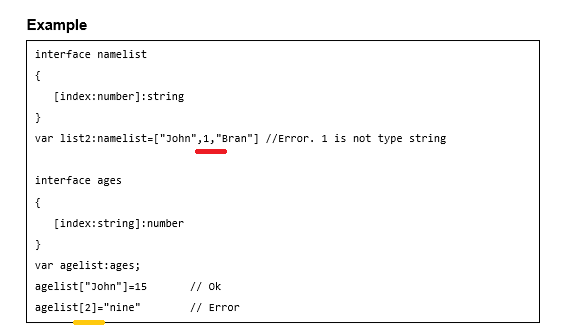


**Union Type and Interface**

1. The following example shows the use of Union Type and Interface:
2. Example🡺

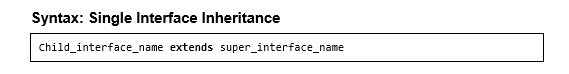


Interfaces and Arrays

1. **Interface can define both the kind of key an array uses and the type of entry it contains. An array Index can be of type string or type number.**
2. Example🡺

**Interfaces and Inheritance**

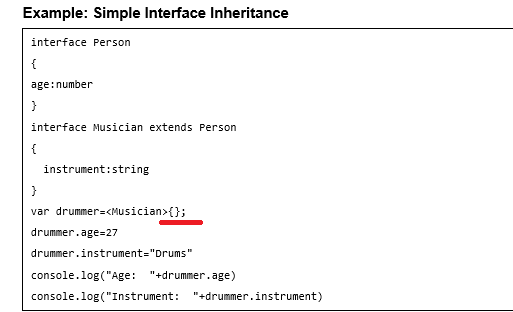
1. **An interface can be extended by other interfaces. In other words, an interface can inherit from other interface. Typescript allows an interface to inherit from multiple interfaces.**
2. Syntax: Single Interface Inheritance



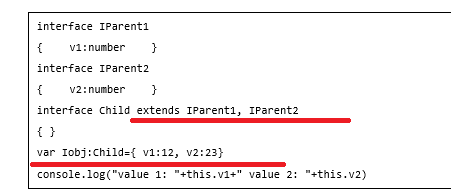
1. Syntax: Multiple Interface Inheritance



1. Example: Simple Interface Inheritance



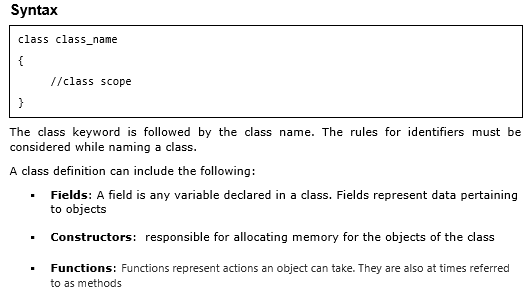
1. Example: Multiple Interface Inheritance



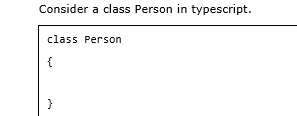
**TypeScript – Classes**

1. **TypeScript is object oriented JavaScript.**
2. **TypeScript supports object-oriented programming features like classes, interfaces, etc. A class in terms of OOP is a blueprint for creating objects. A class encapsulates data for the object. Typescript gives built in support for this concept called class. JavaScript ES5 or earlier didn’t support classes. Typescript gets this feature from ES6.**

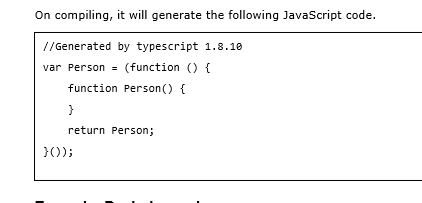
Creating classes



1. Example🡺

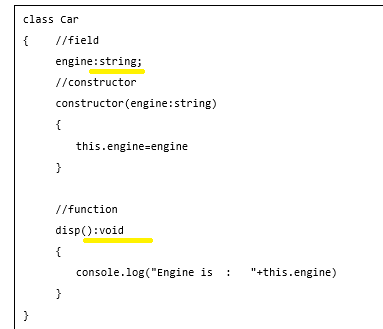


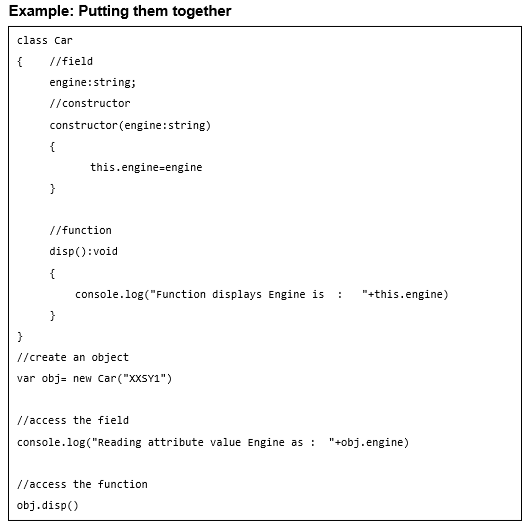
The equivalent JS code for the above code is



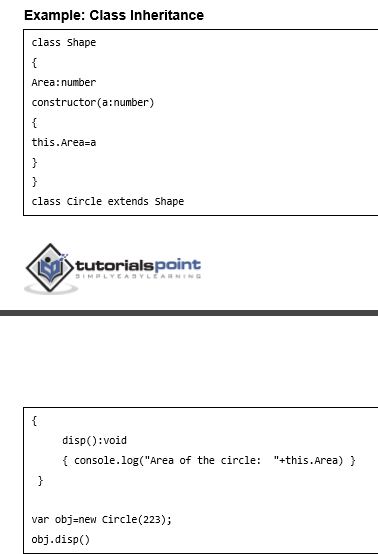
NOTE JS CREATES EQUIVALENT CODE FOR CLSS BUT IN CASE OF INTERFACE THERE IS NO EQUIVALENT CODE IN JS

1. Example: Declaring a class

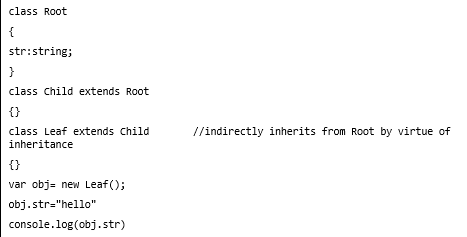


1. 

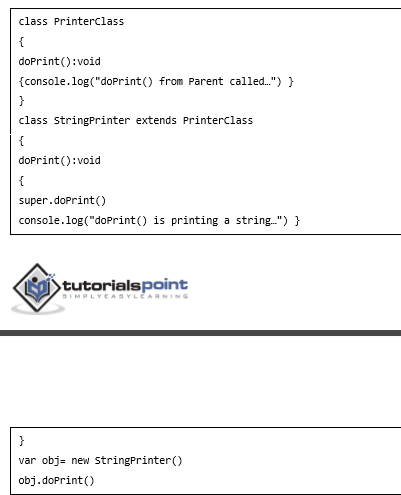
Class Inheritance

1. **TypeScript supports the concept of Inheritance. Inheritance is the ability of a program to create new classes from an existing class**
2. 
3. Inheritance can be classified as:

* Single: Every class can at the most extend from one parent class
* Multiple: A class can inherit from multiple classes. TypeScript doesn’t support multiple inheritance.
* Multi-level: The following example shows how multi-level inheritance works:

1. 

**TypeScript ─ Class inheritance and Method Overriding**

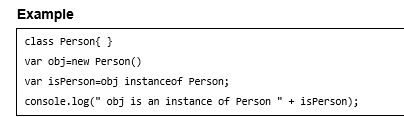
1. 

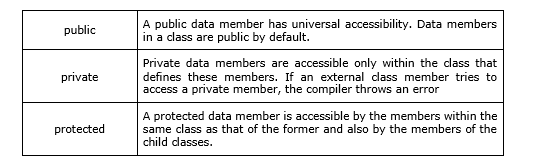
Output🡺

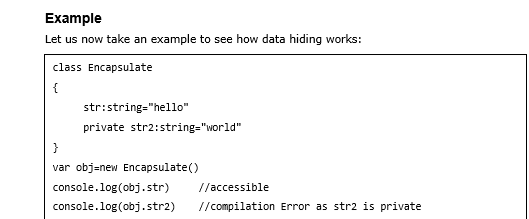
1. The static Keyword 🡺 The static keyword can be applied to the data members of a class. A static variable retains its values till the program finishes execution. Static members are referenced by the class name. 🡺Ms🡺 static can be used for variable, methods and etc…

**instanceof operator**

1. The instanceof operator returns true if the object belongs to the specified type

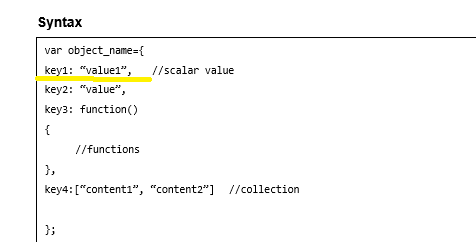


1. Data Hiding **🡺 A class can control the visibility of its data members to members of other classes. This capability is termed as Data Hiding or Encapsulation.**
2. 

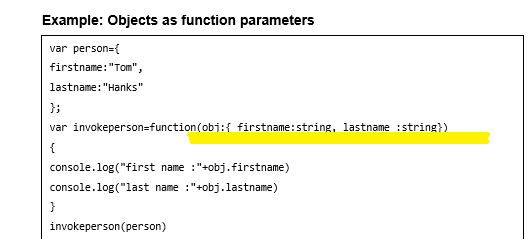


**TypeScript ─ Objects**

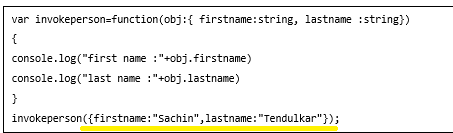
1. **An object is an instance which contains set of key value pairs. The values can be scalar values or functions or even array of other objects. The syntax is given below:**



1. **Objects as function parameters**



1. **Anonymous Object**



1. Duck-typing🡺 we will see this later

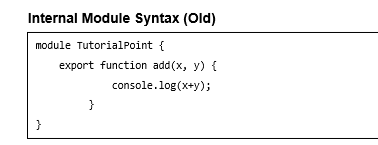
TypeScript ─ Namespaces🡺 later we will see this

**TypeScript – Modules**

1. Module is used to organize code written in TypeScript
2. Modules are broadly divided into

* Internal Modules
* External Modules

1. Internal modules came in earlier version of Typescript. This was used to logically group classes, interfaces, functions into one unit and can be exported in another module. This logical grouping is named namespace in latest version of TypeScript. So internal modules are obsolete instead we can use namespace. Internal modules are still supported, but its recommended to use namespace over internal modules.



NOTE 🡺 WE WILL SEE THE NAMESPACE AND MODULES LATER ONCE DONE WITH OUR BASIC CONCEPT