|  |
| --- |
| TYPESCRIPT |

Implementing a JavaScript-based application is more error-prone as it supports dynamic typing and supports non-strict mode with global variables.

TypeScript makes such implementations easy, as it supports static typing and structured code with the help of modules and object-oriented concepts.

This course introduces various features and programming constructs of TypeScript which enables you to develop a highly structured, less error-prone JavaScript code using TypeScript features.

JavaScript is the language used for client-side scripting to do client-side validations, DOM manipulation, Ajax calls, etc. using JavaScript. Also, JavaScript frameworks can be used for writing complex business logic that runs at the client-side.

As the complexity of the JavaScript code increases, it gradually becomes difficult in coding and maintaining. This is because of the limitations of the JavaScript language. There is no option to change the language for the client-side scripting as the browser understands only JavaScript.

The solution is to choose a language that is rich in features and the code can be converted to JavaScript. This process of converting the code written in one language into another language is called Transpilation.

TypeScript is one such language where its code can get transpiled to JavaScript.

* **Dynamic Typing**: It decides the data type of the variable dynamically at run time.
* **Interpreted Language**: It is a language in which the code instructions are executed directly without prior compilation to machine-language instructions.
* **Minimal Object Oriented support**: Object-Oriented Programming (OOP) is a programming methodology based on the concept of objects. Object-Oriented concepts like classes, encapsulation, inheritance help in the readability and reusability of the code.
* **Minimal IDE support**: Integrated Development Environment (IDE) is a software application that provides all necessary options like code refactoring, intelliSense support, debugging support to software programmers for software development.

Let us discuss each of them in detail.

Consider the below JavaScript function:

1. function calculateTotalPrice(quantity, unitPrice) {
2. return quantity \* unitPrice;
4. }

We can invoke this function using the below code:

1. console.log(calculateTotalPrice(3, "500"));

Since JavaScript is dynamically typed, we can invoke the above function using the below code as well.

1. console.log(calculateTotalPrice('three', "500"));

Even though the above code will not throw any error, it will return **NaN** as output, since the expected number type value is not passed to the quantity argument of the calculateTotalPrice function.

To avoid the above runtime error we can use static typing in TypeScript, wherein we will add data type to the function argument while defining it. Consider the same JavaScript function written using TypeScript as below:

1. function calculateTotalPrice(quantity:number, unitPrice:number) {
2. return quantity \* unitPrice;
4. }

In the above code, we are adding a **number** as the data type to both the arguments of the function. Hence when we invoke the function using the below code:

1. console.log(calculateTotalPrice('three', "500"));

We will get a compilation error since we are invoking the function with the first parameter as a **string** type. Hence we can detect error early at compilation time itself.

JavaScript is an interpreted language. The advantages are:

* It takes less amount of time to analyze the source code
* Memory efficient as no intermediate object code will get generated.

The disadvantage is most of the errors can be identified only at run time.

This can be overcome by using TypeScript which will be transpiled to JavaScript and most of the errors will be fixed during the transpilation time itself.

Therefore, TypeScript saves the application development time for a JavaScript developer.

Consider the below JavaScript code:

1. var \_\_extends = (this && this.\_\_extends) || (function () {
2. var extendStatics = Object.setPrototypeOf ||
3. ({ \_\_proto\_\_: [] } instanceof Array && function (d, b) { d.\_\_proto\_\_ = b; }) ||
4. function (d, b) { for (var p in b) if (b.hasOwnProperty(p)) d[p] = b[p]; };
5. return function (d, b) {
6. extendStatics(d, b);
7. function \_\_() { this.constructor = d; }
8. d.prototype = b === null ? Object.create(b) : (\_\_.prototype = b.prototype, new \_\_());
9. };
10. })();
11. var Product = (function () {
12. function Product() {
13. }
14. return Product;
15. }());
16. var Gadget = (function (\_super) {
17. \_\_extends(Gadget, \_super);
18. function Gadget() {
19. return \_super !== null && \_super.apply(this, arguments) || this;
20. }
21. Gadget.prototype.getProduct = function () {
22. };
23. return Gadget;
24. }(Product));

The equivalent Typescript code is as below:

1. class Product{
2. protected productId:number;
3. }
4. class Gadget extends Product{
5. getProduct():void{
6. }
7. }

You can observe from the above code that:

* The readability of the JavaScript code is minimal.
* Abstraction is done using closures or self-invoking functions wherein the number of lines of code is more compared to public, private, protected access modifiers.
* Classes are created using a constructor function, which leads to confusion compared to using the class keyword.
* Javascript supports OOP through prototypal inheritance which is complex for people with classical inheritance background to understand.

All the above-mentioned features are supported by TypeScript. In addition to that TypeScript also supports interface and generic concepts which is not supported by JavaScript.

A few IDEs have code refactoring, IntelliSense support for JavaScript application development.

* IntelliSense support helps in writing the code quickly.
* Refactoring support helps in changing the variable or function names throughout the application quickly.

Most of the IDEs has good support for TypeScript, some are listed below:

* Visual Studio with versions 2015, 2013, and so on
* Sublime Text
* Atom
* Eclipse
* Emacs
* WebStorm
* Vim

 JavaScript application development has become easier with the help of the following tools:

* **npm**can be used to download packages
* **webpack**can be used to manage the complexity of applications.
* **Babel**can be used to fetch the latest features of the language.
* Tools like **rollup**and **uglifyjs**can be used to optimize application payloads.
* **prettier**and **eslint**can be used to uphold code with consistent style as well as quality.
* IDE like **Visual Studio Code**with **Node.js**environment can be used to run JavaScript code everywhere.

There is browser support challenge for the latest ES6 version of JavaScript. You can use ES6 transpilers like Babel to address this.

**TypeScript**can be another preferred option which is a superset of JavaScript and transpiles to the preferred version of JavaScript.

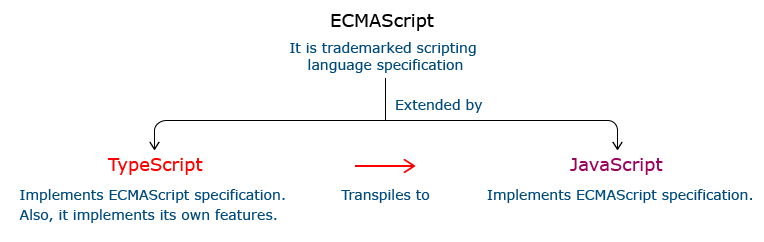
TypeScript can be considered as a typed superset of JavaScript, that transpiles to JavaScript.

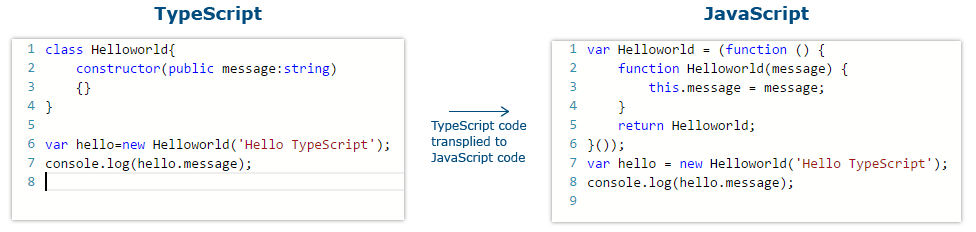
* Transplier converts the source code of one programming language to the source code of another programming language.
* TypeScript makes the development of JavaScript more of a traditional object-oriented experience.
* TypeScript is based on ECMAScript 6 and 7 proposals.
* Any valid JavaScript is TypeScript.

TypeScript implements EcmaScript specification. Apart from the EcmaScript specification, TypeScript has its own features as well.

JavaScript also implements EcmaScript.

TypeScript code must be transpiled to JavaScript code to use it in an application.





From the above code, the TypeScript class Helloworld is converted to a self-invoking function in JavaScript when transpiled.

You can use TypeScript's online playground editor to see how TypeScript gets converted into JavaScript.

**Static Typing:** It adds static typing to JavaScript, due to which the readability of the code improves and helps in finding more early compilation errors than run time errors.

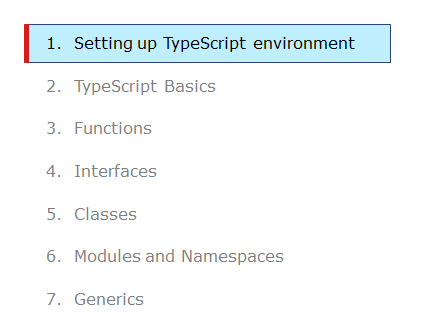
**Modules support:** TypeScript provides an option to create modules to modularize the code for easy maintenance. Modules help in making the application scalable.

**Object-Oriented Programming:** TypeScript supports Object-Oriented Programming features such as class, encapsulation, interface, inheritance and so on which helps in creating highly structured and reusable code.

**Open Source:** TypeScript is open source. The source code of TypeScript can be downloaded from Github.

**Cross-Platform:** It works across the platform.

**Tooling Support:** TypeScript works extremely well with Sublime Text, Eclipse, and almost all major IDEs compared to JavaScript.



To install TypeScript, go to the official site of TypeScript ( http://www.typescriptlang.org/ ) and follow the steps mentioned there to download TypeScript.

As mentioned on the official site, you need to install Node.js.

Install Node.js from the official site of Node.js ( https://nodejs.org/en/) or Software Center.

Open a Node.js command prompt and check whether node and npm are installed in your machine by using **"node -v "**and **"npm -v"**commands.

npm is a command-line tool that comes along with Node.js installation with which you can download node modules. TypeScript is also such a node module that can be installed using npm.

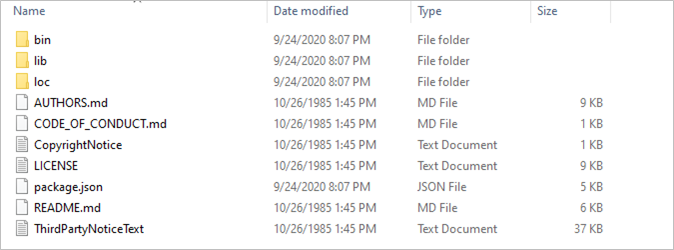
Open Node.js command prompt, execute the below commands as shown below to download the TypeScript node module from the local Infosys repository.

**Setup the registry:**

1. npm config set registry https:*//infyartifactory.ad.infosys.com/artifactory/api/npm/npm/*

In the same Node.js command prompt, type the **"npm i –g typescript"**command to download the TypeScript module from the repository.

On successful execution of above command, the TypeScript module will get downloaded under folder C:\Users\<<username>>\AppData\Roaming\npm\node\_modules\typescript as shown below.



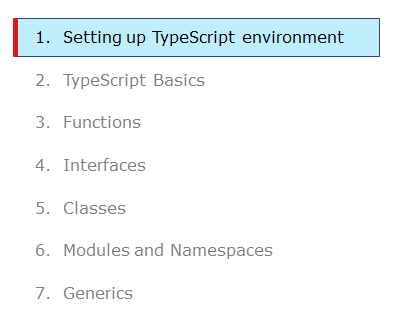
In the same command prompt check for TypeScript installation as below:

1. tsc -v
2. *//or*
3. tsc --version

Output showing version number indicates the successful installation of the TypeScript module.

Alternatively, you can also use TypeScript online playground editor in this case, you should be always connected to good Internet.

To configure TypeScript with different IDEs. Here are a few links for IDE configuration for TypeScript:

* **Visual Studio Code**: https://code.visualstudio.com/Docs/languages/typescript
* **Eclipse IDE**: https://github.com/palantir/eclipse-typescript
* **Visual Studio 2015**: https://angular.io/guide/visual-studio-2015
* 

To install TypeScript, go to the official site of TypeScript and follow the steps mentioned there to download TypeScript.

As mentioned on the official site, you need to install Node.js.

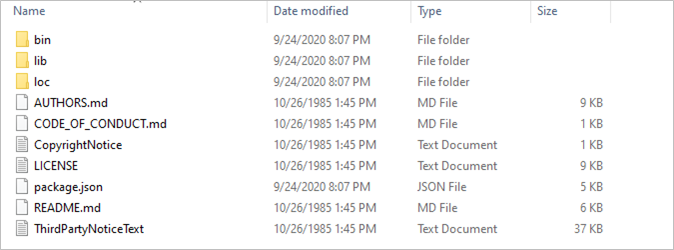
Install Node.js from the official site of Node.js.

Open a Node.js command prompt and check whether node and npm are installed in your machine by using **"node -v"** and **"npm -v"** commands.

npm is a command-line tool that comes along with Node.js installation with which you can download node modules. TypeScript is also such a node module that can be installed using npm.

Open a Node.js command prompt and use the command **"npm i –g typescript"** to download the TypeScript module from the npm repository.

Latest typescript module will get downloaded under folder C:\Users\<<username>>\AppData\Roaming\npm\node\_modules\typescript as shown below.



In the same command prompt check for TypeScript installation as below:

1. tsc -v
2. *//or*
3. tsc --version

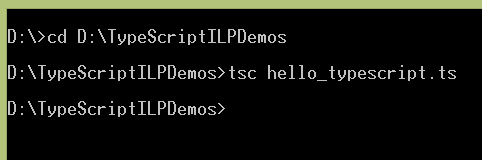
Output showing version number indicates the successful installation of the TypeScript module.

Alternatively, you can also use TypeScript online playground editor in this case you should be always connected to the Internet.

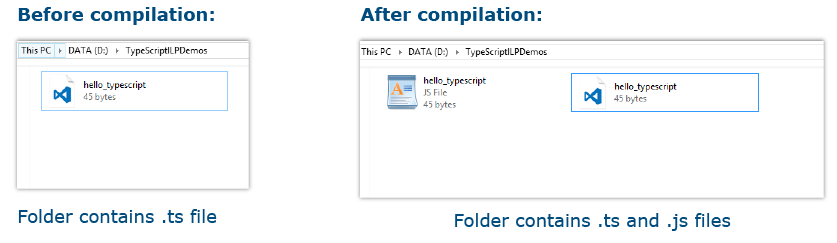
To configure TypeScript with different IDEs. Here are some of the popular IDEs for working with TypeScript:

* Visual Studio code
* Eclipse IDE
* Visual Studio 2015
* To start with the first application in TypeScript, create a file hello\_typescript.ts under a folder. In the ts file give a console.log statement and save it.


* 
* From the Node.js command prompt, navigate to the directory in which the ts file resides and compile the ts file using the **tsc** command.

* 

* After compilation of the TypeScript file, the corresponding JavaScript file gets generated.

* 

* To run the generated JavaScript file, use the node command from the command line or include it in an HTML page using the script tag and render it in the browser.

**Highlights:**

* Helloworld using TypeScript
* Execute First TypeScript program

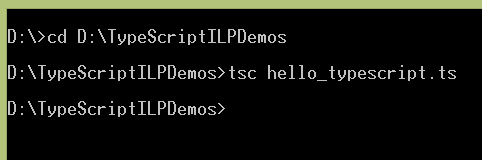
**Demo steps:**

Step 1: Create a file hello\_typescript.ts

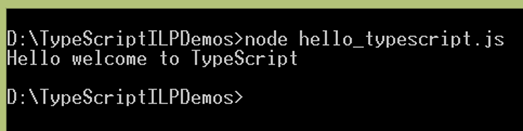
**hello\_typescript.ts**

1. console.log('Hello welcome to TypeScript');

Step 2: From the command prompt navigate to the folder in which the ts file resides and compile the ts file using the tsc command.

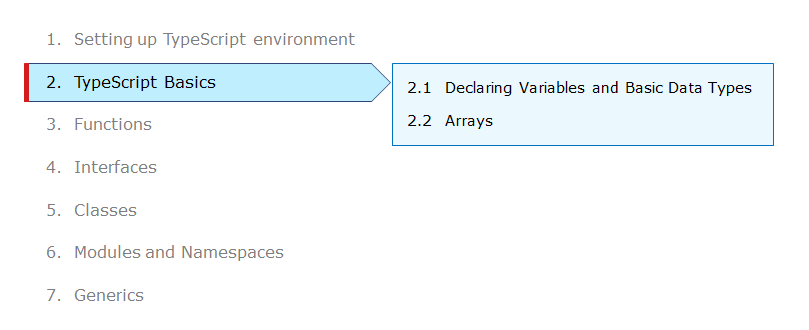


Step 3: Run the compiled js file using node command as below:

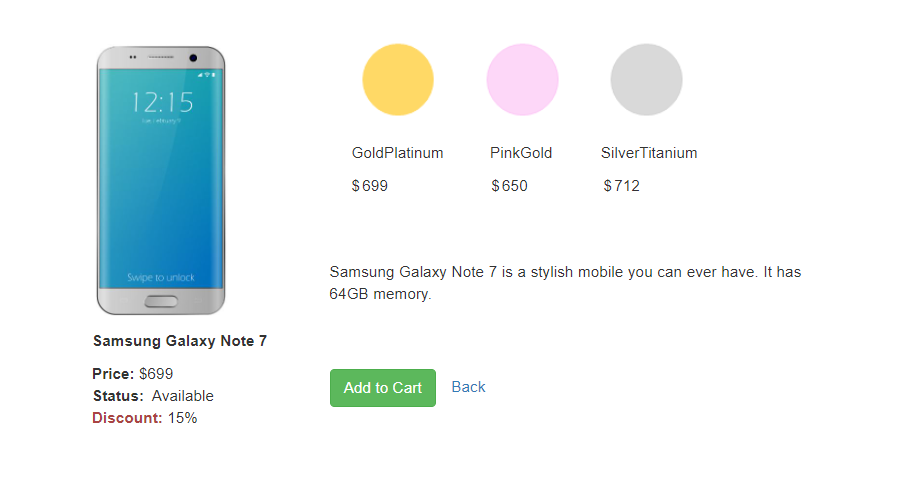


At the end of this module, you have learnt:

* Importance of TypeScript.
* How to install TypeScript node modules?
* How to execute a TypeScript Demo>



Consider the below page of the Mobile Cart application. The information such as mobile phone name, price, status on the availability of the mobile, discounts is displayed on the page. It also displays different price ranges as GoldPlatinum, PinkGold, SilverTitanium.



Each of the information getting displayed has a specific type. For example, the price can only be numeric and the mobile name can only be a string.

There should be a mechanism using which you can restrict the values being rendered on the page.

Sometimes it is preferred to have a text instead of numeric values to represent some information. For example, on the above page instead of categorizing the mobiles based on their prices, you can prefer to remember them as some text like GoldPlatinum, PinkGold, etc.

TypeScript is a static typed language that allows us to declare variables of various data types so that you ensure only the desired type of data being used in the application.

Let us discuss declaring variables and basic data types supported by TypeScript.

**Declaring Variables:**

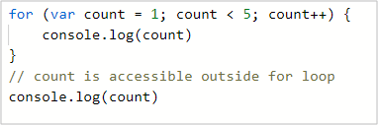
Declaration of a variable is used to assign a data type and an identifier to a variable. Declaration of variables in TypeScript is like JavaScript variable declaration.

Below are the three declaration types supported in TypeScript.

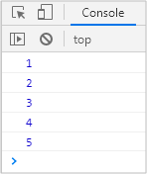
|  |  |
| --- | --- |
| **Data Type** | **Explanation** |
| var | * Variable declared with this type would have function scope. * They can be re-assigned and re-defined. * When a variable declared outside the function, it would have global scope and automatically attaches itself to the window object. |
| let | * Variable declared with this type would have a block-level scope. * They can be re-assigned and cannot be redefined. |
| const | * Variable declared with this type would have a block-level scope. * They can be neither re-assigned nor re-defined. |

**Problem with var declaration:**

In the below code, you will observe a strange behavior of the variable count. You can access this variable even outside the loop although it is defined inside the loop. This is due to the global scope nature of the **var**data type declaration.



Output:



Some of the other problems with global scope variable are:

* When the var declared variable is defined outside a function it is attached with the window object which has only one instance.
* This is a bad practice and causes an overridden of a variable when you try to use a third-party library working with the same variable.
* Generally, var variables have a function-scope and if they are not declared within a function, they have a global scope.

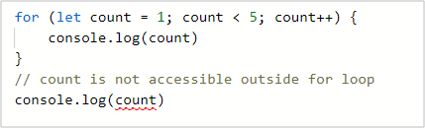
To overcome this issue, in ES2015. JavaScript has introduced **let**and **const**as two new declaration types and it is considered as standard because declaring variables by these two data types are safer than declaring a variable using the var keyword.

Let us learn about these declarations.

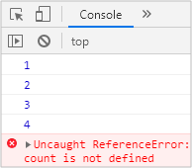
**Declaring Variables using let:**

When a developer wants to declare variables to live only within the block, they can achieve a block-scoped variable using **let**declaration.

Let us replace the var with the let keyword in the previous for loop code-snippet and observe the difference.



Output:

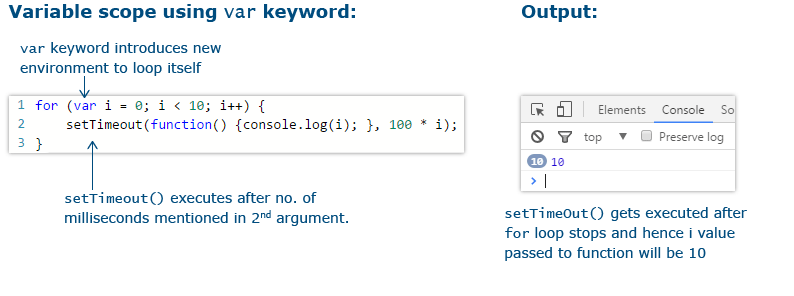


Since the count variable is a **block-scoped**, it is not accessible outside the for loop hence results in the error as not defined.

**Capturing Variable in the loop**

Variable declared using **var**keyword will have function scope. So, even if you declare a variable using **var**inside a loop, it will have function scope.

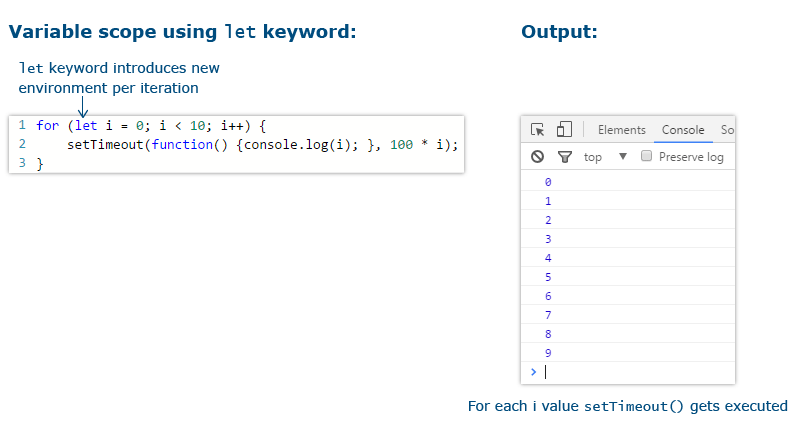
In the below example, variable **i** has been declared inside the for loop using the **var**keyword, but it will have the scope of the function. Thus, by the time **setTimeout**function executes, the value of **i** has already reached 10.



Therefore, every invocation of **setTimeout**function gets the same value of i as 10.

Variable declared using **let**keyword will have block scope. Therefore, once the block is terminated, the scope also is lost.

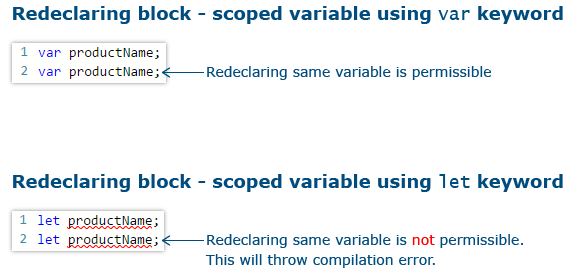
In the below example, variable **i** has been declared using **let**inside the for loop. So, its scope will be limited to one iteration of the loop.



In this scenario, every invocation of the **setTimeout**function will get the value of **i** from that iteration scope.

**Redeclaring block-scoped variable:**

* The let declared variable cannot be redeclared within the same block.
* The var declared variable can be redeclared within the same block.

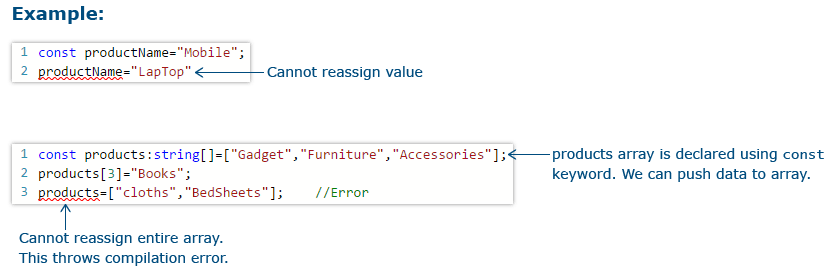
****

Due to these reasons let declaration is preferred over var declaration.

Now let us see what is const declaration meant for.

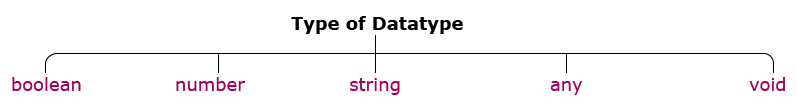
The const declaration is similar to the let declaration except that the value cannot be re-assigned to the variable.

* const declared variables are mutable if declared as an array or as an object literal.
* this declaration should be used if the value of the variable should not be reinitialized.



Data type mentions the type of value assigned to a variable. It is used for variable declarations.

Since TypeScript supports static typing, the data type of the variable can be determined at the compilation time itself.



There are variables of different types in TypeScript code based on the data type used while declaring the variable.

**boolean:**

boolean is a data type that accepts only true or false as a value for the variable it is been declared.

In a shopping cart application, you can use a boolean type variable to check for the product availability, to show/hide the product image, etc.



**number:**

number type represents numeric values. All TypeScript numbers are floating-point values.

In a shopping cart application, you can use number type to declare the Id of a product, price of a product, etc.

1

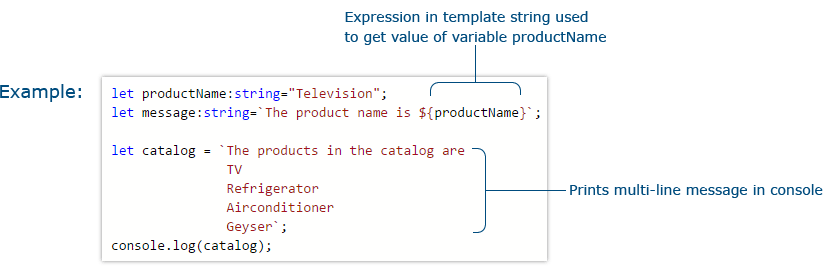
**string:**

A string is used to assign textual values or template strings. String values are written in quotes – single or double.

In a shopping cart application, you can use string type to declare variables like productName, productType, etc.



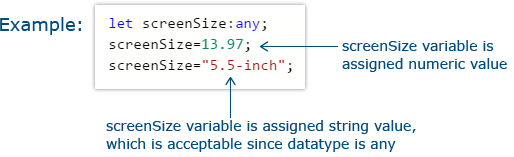
Template strings are types of string value that can have multiple lines and embedded expressions. They are surrounded by the backquote\backtick (`) character and embedded expressions of the form ${ expr }.



**any:**

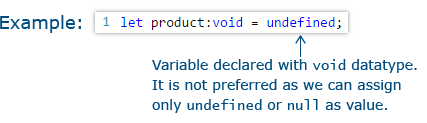
any type is used to declare a variable whose type can be determined dynamically at runtime with the value supplied to it. If no type annotation is mentioned while declaring the variable, any type will be assigned by default.

In a shopping cart application, you can use any type to declare a variable like screenSize of a Tablet.

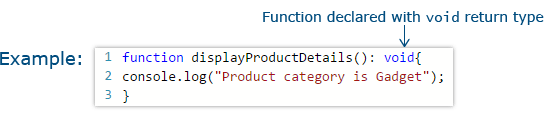


**void:**

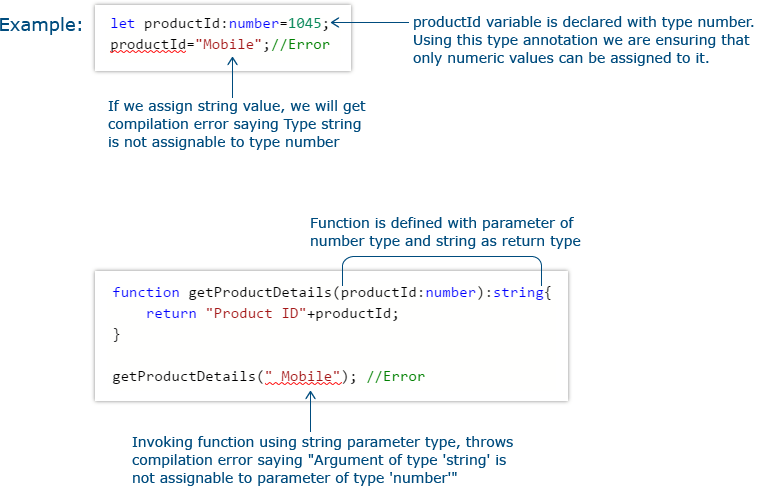
void type represents undefined as the value. the undefined value represents "no value".  
A variable can be declared with void type as below:



void type is commonly used to declare function return type. The function which does not return a value can be declared with a void return type. If you don’t mention any return type for the function, the void return type will be assigned by default.



Type Annotation is a way to enforce type restriction to a specific variable or a function.   
If a variable is declared with a specific data type and another type of value is assigned to the variable, a compilation error will be thrown.

 Basic Types

Problem Statement:

As a part of Mobile application development consider that you need to declare five variables namely productId, productDescription, productName, productAvailable, and discountPercentage of Number, String, Boolean and Constant data types as well as populate the respective type of the declared variable on the console along with discount percentage for that product then below mentioned code-snippet would fit into your requirement.

**Activity:**

* Modify the productName variable data type as a number at line no 7 and re-execute the code.
* Reassign productAvailable variable value as false and discountPercentage variable value as 30 after line no 20, populate the update the new value on the console, and re-execute the code.

// Declaring variables with basic types using let

let productId: number = 1045; // Declaring a numeric variable

let productDescription: string = '16GB, Gold'; // Declaring a string variable

let productName: string = 'Samsung Galaxy J7';

let productAvailable: boolean = true; // Declaring a boolean variable

console.log('The type of productId is ' + typeof productId);

console.log('The type of productAvailable is ' + typeof productAvailable);

console.log('The type of productName is ' + typeof productName);

// Declaring variables using const

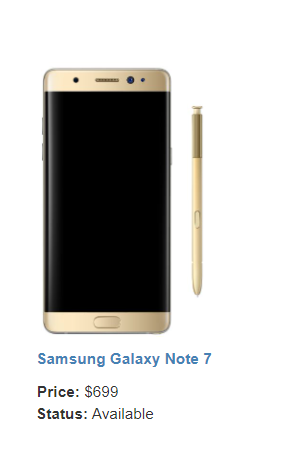
const discountPercentage: number = 15;

console.log('Discount percentage is:' + discountPercentage);

**Problem Statement:**

Create the below page of the Mobile Cart application:

Design the below screen:



To solve this problem, you will need HTML code to show the image, text, and labels.

For the value of Mobile name, price, and status information, write TypeScript code.

In the TypeScript code, use the below statement to update the designated place in the HTML page.

**document.getElementById("pName").innerHTML=mobileName;**

Here, mobileName is a TypeScript variable and pName is the id of the HTML element where the mobile name has to be displayed.

Similarly, price and status information need to be rendered on the HTML page.

Below is the complete HTML code for the page, write the TypeScript code for updating the information of mobile name, price, status, and discount.

1. <!doctype html>
2. <html>
3. <head>
4. <title>Mobile Cart</title>
5. <meta charset="UTF-8">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
8. <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
9. <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
10. <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
12. <style>
13. .navbar-inverse {
14. background-color:#005580;
15. background-image: none;
16. background-repeat: no-repeat;
17. color:#ffffff;
18. }
19. .navbar-inverse .navbar-nav > .active > a {
20. color: *#ffffff;*
21. background-color:transparent;
23. }
24. .navbar-inverse .navbar-nav > li > a:hover{
25. text-decoration: none;
26. color: *#ffffff;*
27. }
28. </style>
29. </head>
30. <body>
31. <nav class="navbar navbar-inverse navbar-fixed-top">
32. <div class="navbar-header">
33. <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
34. <span class="sr-only">Toggle navigation</span>
35. <span class="icon-bar"></span>
36. <span class="icon-bar"></span>
37. <span class="icon-bar"></span>
38. </button>
39. <a class="navbar-brand" href="#">Mobile Cart</a>
40. </div>
41. <div id="navbar" class="collapse navbar-collapse">
42. <ul class="nav navbar-nav">
43. <li><a href="#">Home</a></li>
45. </ul>
46. *<!--/.nav-collapse -->*
47. <ul class="nav navbar-nav navbar-middle" style="color:white; margin-right:30px;">
48. <li><a href="Cart.html"><span class="glyphicon glyphicon-shopping-cart" style="color:white"></span></a></li>
49. </ul>
50. </div>
51. </nav>
52. <div style="margin-top:7%">
53. <center> <h2>Your Favorite Online Mobile Shop!</h2> </center>
54. </div>
55. <div class="container" style="padding-top:5%">
56. <div class="row">
57. <div class="col-md-4">
58. <div style="text-align: center;">
59. <img src="Images/Part 1/SamsungGalaxy\_Gold.jpg" height="250px">
60. </div>
61. <div style="padding-top:10px;">
62. <div style="cursor:pointer;color:Steelblue;text-align: center;"><b>
63. <span id="pName"></span></b></div>
64. <div style="padding-top:10px;padding-left: 101px;"><b>Price:</b>&nbsp;&dollar;<span id="pPrice"></span></div>
65. <div style="padding-left: 100px;"><b>Status:</b><span id="pAvailable"></span></div>
66. </div>
67. </div>
69. </div>
70. </div>
71. </body>
72. *<!--Adding the converted js file -->*
73. <script src="productlist.js"></script>
74. </html>

**Attach the transpiled TypeScript file to the HTML page as highlighted in the above code.**

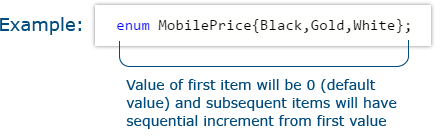
Enum in TypeScript is used to give more friendly names to a set of numeric values.

For example, if you need to store a set of size variables such as Small, Medium, Large with different numeric values, group those variables under a common enum called Size.

By default, enum’s first item will be assigned with zero as the value and the subsequent values will be incremented by one.

1

In a shopping cart application, you can use a MobilePrice enum to store different prices of the mobile depending on the mobile color.

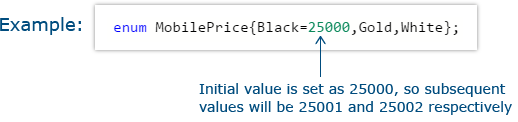


To get the value from an enum use one of the following:

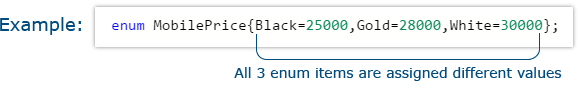


Enum items can also be initialized with different values than the default value. If you set a different value for one of the variables, the subsequent values will be incremented by 1.



You can even set different values to different enum items



Tryout : Enum

Problem Statement

Consider that a developer needs to declare an enum variable named MobilePrice and assign three different parameters like Black, Gold, and White with 250, 280, and 300 dollars respectivtely as default values and it also consists of the function named calculateAmount which would help the developer in calculating totalAmount variable based on different parameter like discount and color of Mobile.

**Activity:**

* Modify line no 2 as given below and re-execute the code.

enum MobilePrice {Black, Gold, White}

* Modify line no 2 as given below and re-execute the code and populate the actual and final price of White color mobile.

enum MobilePrice {Black, Gold=30000, White}

// declaring enum variable and assigning default values

enum MobilePrice {Black= 250, Gold= 280, White= 300}

// functon to calculate final amount

function calculateAmount(price: number): number {

let discount: number;

let totalAmount: number;

if (price === MobilePrice.Gold) {

discount = 5;

} else if (price === MobilePrice.White) {

discount = 8;

} else {

discount = 10;

}

totalAmount = price - price \* discount / 100;

return totalAmount;

}

// lines to populate the Actual and Final price of Black color Mobile

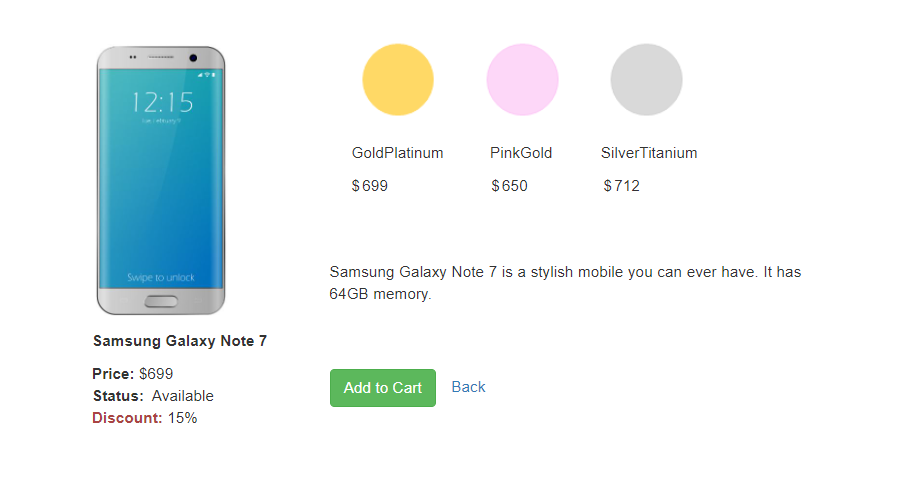
console.log('Actual price of the Mobile is $' + MobilePrice.Black);

console.log('The final price after discount is $' + calculateAmount(MobilePrice.Black));

**Problem Statement:**

Create the below page of the Mobile Cart application:

Design the below screen:



In the previous Mobile Cart exercise page, you have rendered the Mobile phone name, price, status on the availability of the mobile, and discount details.

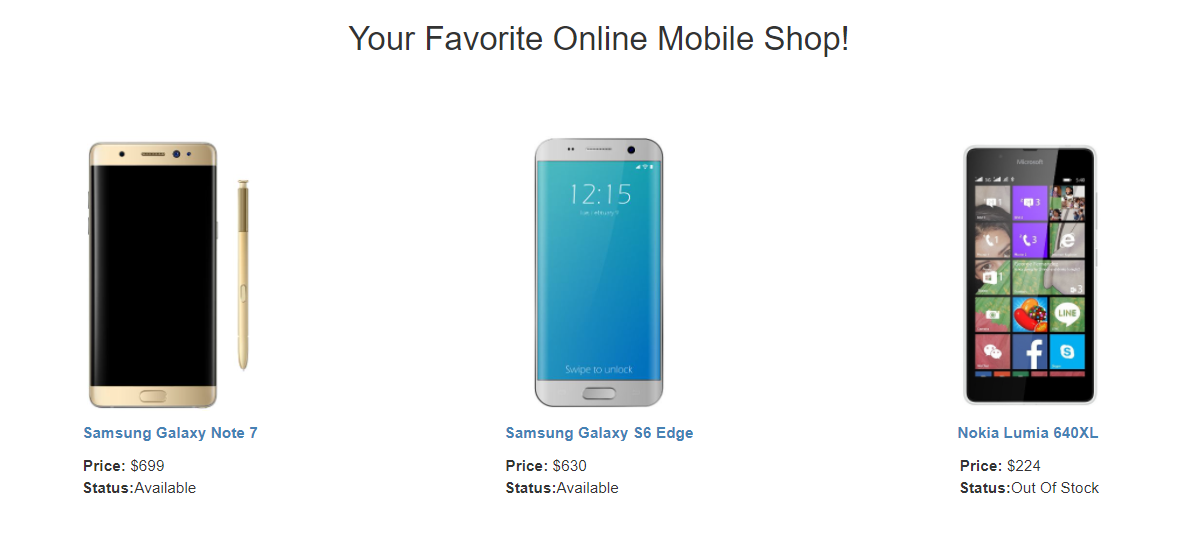
On the page, display the price of the mobile-based in three different colors. Instead of using the number in our code, represent them by string values like GoldPlatinum, PinkGold, SilverTitanium.

Below is the complete HTML code for the page, write the TypeScript code to declare enum variable to represent the price of mobiles and update the page using these enum values.

1. <!doctype html>
2. <html>
3. <head>
4. <title>Mobile Cart</title>
5. <meta charset="UTF-8">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
8. <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
10. <style>
11. .navbar-inverse {
12. background-color:#005580;
13. background-image: none;
14. background-repeat: no-repeat;
15. color:#ffffff;
16. }
17. .navbar-inverse .navbar-nav > .active > a {
18. color: *#ffffff;*
19. background-color:transparent;
21. }
22. .navbar-inverse .navbar-nav > li > a:hover{
23. text-decoration: none;
25. }
26. </style>
27. </head>
28. <body>
29. <nav class="navbar navbar-inverse navbar-fixed-top">
30. <div class="navbar-header">
31. <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
32. <span class="sr-only">Toggle navigation</span>
33. <span class="icon-bar"></span>
34. <span class="icon-bar"></span>
35. <span class="icon-bar"></span>
36. </button>
37. <a class="navbar-brand" href="#">Mobile Cart</a>
38. </div>
39. <div id="navbar" class="collapse navbar-collapse">
40. <ul class="nav navbar-nav">
41. <li ><a href="Index.html">Home</a></li>
42. </ul>
43. *<!--/.nav-collapse -->*
44. <ul class="nav navbar-nav navbar-middle" style="color:white; margin-right:30px;">
45. <li><a href="Cart.html"><span class="glyphicon glyphicon-shopping-cart" style="color:white"></span></a></li>
46. </ul>
47. </div>
48. </nav>
50. <div class="container" style="margin-top:7%">
51. <div class="row">
52. <div class="col-sm-4">
53. <div style="margin-left:40%;padding-top:15px;">
54. <div style="padding:15px;">
55. <div>
56. <img id="phoneImage" src ="Images/Part 1/samsung\_edge\_silver.jpg" height="250px" >
57. </div>
58. <div style="padding-top:10px;">
59. <div><b><span id="pName"></span></b></div>
60. <div style="padding-top:10px;"><b>Price:</b>&nbsp;&dollar;<span id="pPrice"></span></div>
61. <div><b>Status:</b>&nbsp;<span id="pAvailable"></span></div>
62. <div><b class="text-danger">Discount:</b>&nbsp;<span id="pDiscount"></span></div>
63. </div>
64. </div>
65. </div>
66. </div>
67. <div></div>
68. <div style="padding-top:15px;">
69. <div>
70. <img src ="Images/Part 1/goldmobile.png" style="padding-left:15px;cursor:pointer;" onclick="getMobileByColor('gold');" >
71. <img src ="Images/Part 1/pinkmobile.png" style="cursor:pointer;padding-left:15px;" onclick="getMobileByColor('pink');" >
72. <img src ="Images/Part 1/silvermobile.png" style="cursor:pointer;padding-left:15px;" onclick="getMobileByColor('silver');" >
74. </div>
75. <div style="padding-top:10px;">
76. <span style="padding-left:20px;">
77. GoldPlatinum
78. </span>
79. <span style="padding-left:38px;">
80. PinkGold
81. </span>
82. <span style="padding-left:40px;">
83. SilverTitanium
84. </span>
85. </div>
86. <div style="padding-top:10px;">
87. <span style="padding-left: 20px;">&dollar;</span><span id="gold" style="padding-left: 2px;"></span>
88. <span style="padding-left: 90px;">&dollar;</span><span id="pinkgold" style="padding-left: 2px;"></span>
89. <span style="padding-left: 65px;">&dollar;</span><span id="silver" style="padding-left: 2px;"></span>
90. <div id="productDescription" style="padding-top:5%;width:70%;text-align:justify">
91. Samsung Galaxy Note 7 is a stylish mobile you can ever have. It has 64GB memory.
92. </div>
93. <div style="padding-top:5%;padding-right:10%">
94. <button class="btn btn-success" onclick="addtoCart();">Add to Cart</button>
95. <a style="padding-left:10px;" onclick="backHome();">Back</a>
96. </div>
97. </div>
98. </div>
99. </div>
100. </div>
101. </body>
102. <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
103. <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
104. *<!-- Adding converted js code-->*
105. <script src="productdetail.js"></script>
106. </html>

**Attach the transpiled TypeScript file to the HTML page as highlighted in the above code.**

Consider the below page of the Mobile Cart application that displays the list of mobile phones available. For each mobile in the list, you can see its image, name, price, and availability status.

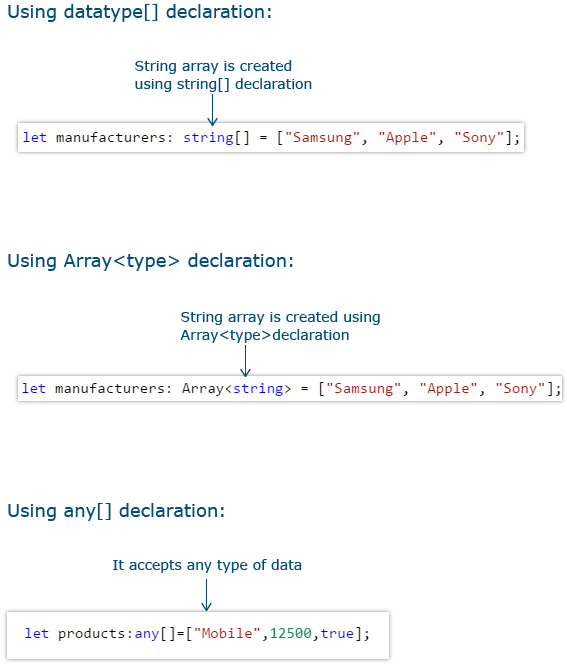


This requirement is implemented using the concept of **arrays**of TypeScript. Let us discuss arrays in TypeScript.

An Array is used to store multiple values in a single variable. You can easily access the values stored in an array using the index position of the data stored in the array.

TypeScript array is an object to store multiple values in a variable with a type annotation. They are like JavaScript arrays.

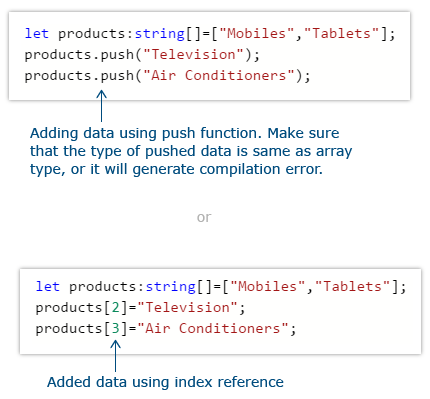
Arrays can be created using one of the below:



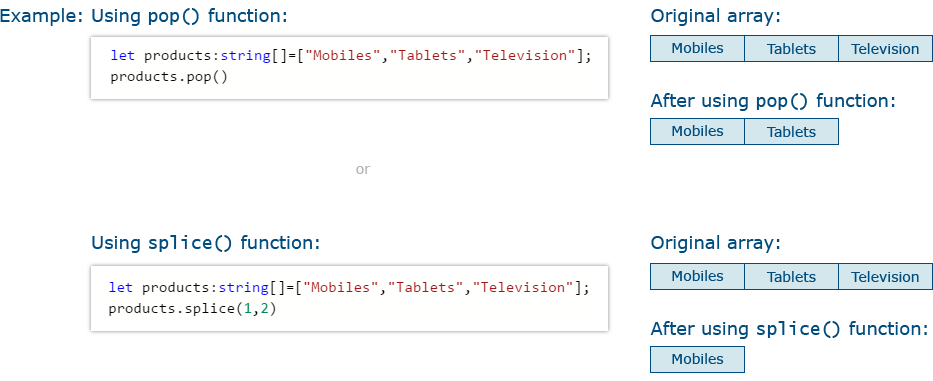
A TypeScript array defined with a specific type does not accept data of different types. TypeScript arrays can be accessed and used much like JavaScript arrays.

JavaScript Arrays has several useful properties and methods to access or modify the given array. The same is supported in TypeScript.

To add a dynamic value to an array, you can either use the push function or use the index reference.



Data can be removed from an array using the pop function or splice function



The splice function removes the item from a specific index position.

Tryout : Arrays

Problem Statement

Consider that a developer needs to declare an Array named manufacturers of any datatype which would hold 4 different objects having id - string and checked - boolean as parameters and the developer need to populate the declared array id value on the console. Then below-mentioned code-snippet would fit into the requirement.

**Activity:**

* Modify the datatype of the manufacturer's array as a string and re-execute the code.
* Modify for loop logic to populate the checked parameter value when manufacturers array id is "Apple".

// declaring an array of any datatype

const manufacturers: any[] = [{ id: 'Samsung', checked: false },

{ id: 'Motorola', checked: false },

{ id: 'Apple', checked: false },

{ id: 'Sony', checked: false }

];

console.log('Available Products are: ');

// logic to populate the above declared array's id value

for (const item of manufacturers) {

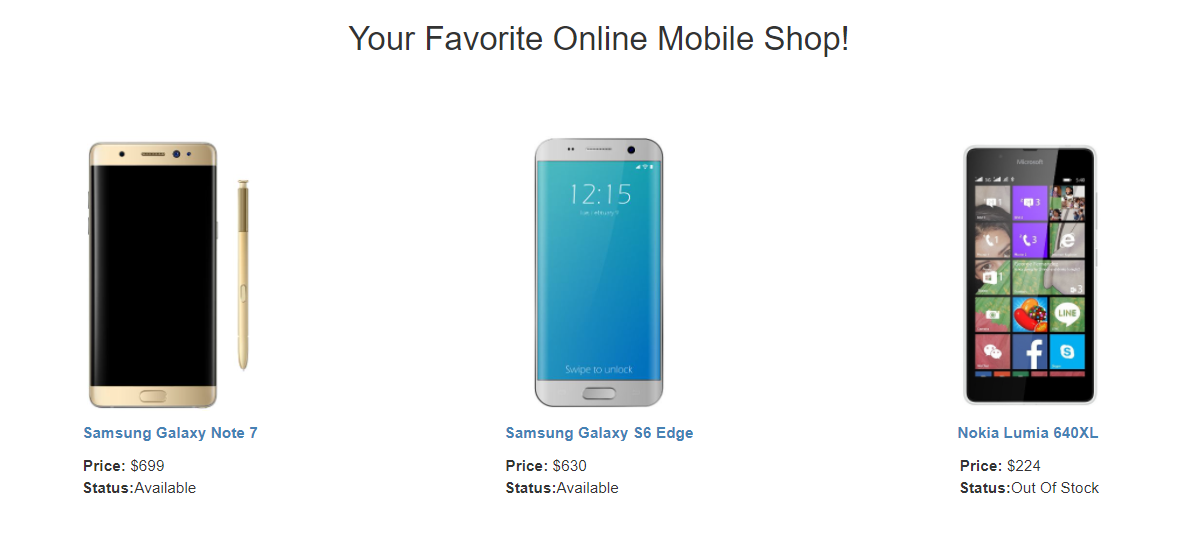
console.log(item.id);

}

**Problem Statement:**

Consider the below requirement of the Mobile Cart application:

Design the below screen:



The above page has objects which represent data about the mobile which will be used to render the details of the mobiles on the page.

**Example of a Product object:**

1. {pId:432,productName:"Samsung Galaxy Note 7",productPrice:59000,productAvailable:true,imageUrl:"SamsungGalaxy\_Gold.jpg",
2. productDescription:"Samsung Galaxy Note 7 is a stylish mobile you can ever have. It has 64GB memory."
3. }

Create such product objects and place them into a productlist array. Render the above screen with product details from this array.  
Write the TypeScript code to solve the above requirement. The complete HTML code is given below:

1. <!doctype html>
2. <html>
3. <head>
4. <title>Mobile Cart</title>
5. <meta charset="UTF-8">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
8. <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
9. <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
10. <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
12. <style>
13. .navbar-inverse {
14. background-color:#005580;
15. background-image: none;
16. background-repeat: no-repeat;
17. color:#ffffff;
18. }
19. .navbar-inverse .navbar-nav > .active > a {
20. color: *#ffffff;*
21. background-color:transparent;
23. }
24. .navbar-inverse .navbar-nav > li > a:hover{
25. text-decoration: none;
26. color: *#ffffff;*
27. }
28. </style>
29. </head>
30. <body>
31. <nav class="navbar navbar-inverse navbar-fixed-top">
32. <div class="navbar-header">
33. <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar" aria-expanded="false" aria-controls="navbar">
34. <span class="sr-only">Toggle navigation</span>
35. <span class="icon-bar"></span>
36. <span class="icon-bar"></span>
37. <span class="icon-bar"></span>
38. </button>
39. <a class="navbar-brand" href="#">Mobile Cart</a>
40. </div>
41. <div id="navbar" class="collapse navbar-collapse">
42. <ul class="nav navbar-nav">
43. <li><a href="#">Home</a></li>
44. </ul>
45. *<!--/.nav-collapse -->*
46. <ul class="nav navbar-nav navbar-middle" style="color:white; margin-right:30px;">
47. <li><a href="Cart.html"><span class="glyphicon glyphicon-shopping-cart" style="color:white"></span></a></li>
49. </ul>
50. </div>
51. </nav>
52. <div style="margin-top:7%">
53. <center> <h2>Your Favorite Online Mobile Shop!</h2> </center>
54. </div>
55. <div class="container" style="padding-top:5%">
56. <div class="row">
57. <div class="col-md-4">
58. <div style="text-align: center;">
59. <img src="Images/Part 1/SamsungGalaxy\_Gold.jpg" height="250px">
60. </div>
61. <div style="padding-top:10px;">
62. <div onclick="getMobileDetails();" style="cursor:pointer;color:Steelblue;text-align: center;"><b><span id="pName0"></span></b></div>
63. <div style="padding-top:10px;padding-left: 101px;"><b>Price:</b>&nbsp;&dollar;<span id="pPrice0"></span></div>
64. <div style="padding-left: 100px;"><b>Status:</b><span id="pAvailable0"></span></div>
65. </div>
66. </div>
67. <div class="col-md-4">
68. <div style="text-align: center;">
69. <img src="Images/Part 1/samsung\_edge\_silver.jpg" height="250px">
70. </div>
71. <div style="padding-top:10px;">
72. <div onclick="getMobileDetails('samsungedge',231);" style="cursor:pointer;color:Steelblue;text-align: center;"><b><span id="pName1"></span></b></div>
73. <div style="padding-top:10px;padding-left: 95px;"><b>Price:</b>&nbsp;&dollar;<span id="pPrice1"></span></div>
74. <div style="padding-left: 94px;"><b>Status:</b><span id="pAvailable1"></span></div>
75. </div>
76. </div>
77. <div class="col-md-4">
78. <div style="text-align: center;">
79. <img src="Images/Part 1/lumia\_640xl.jpg" height="250px">
80. </div>
81. <div style="padding-top:10px;">
82. <div onclick="getMobileDetails('lumia',875);" style="cursor:pointer;color:Steelblue;text-align: center; "><b><span id="pName2"></span></b></div>
83. <div style="padding-top:10px;padding-left: 118px;"><b>Price:</b>&nbsp;&dollar;<span id="pPrice2"></span></div>
84. <div style="padding-left: 117px;"><b>Status:</b><span id="pAvailable2"></span></div>
85. </div>
86. </div>
87. </div>
89. </div>
91. </body>
92. *<!--Adding the converted js file -->*
93. <script src="arrays.js"></script>
94. </html>

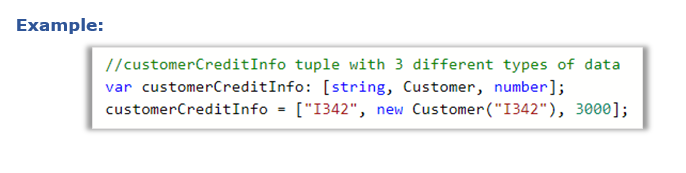
**Attach the transpiled TypeScript file to the HTML page as highlighted in the above code.**

Tuple type is a kind of array which accepts more than one predefined type of data. Arrays are used to represent a collection of similar objects, whereas tuples are used to represent a collection of different objects.

Let us consider an example, where customerCreditId, Customer object, and customerCreditLimit must be represented in a data type.

The choice could be to define a class, with these properties. If it is represented using the class, then there will be a requirement to instantiate the class and then the properties can be accessed.

Tuples provides an easy way to implement the same scenario with an array-like data structure, which is easy to access and manipulate.



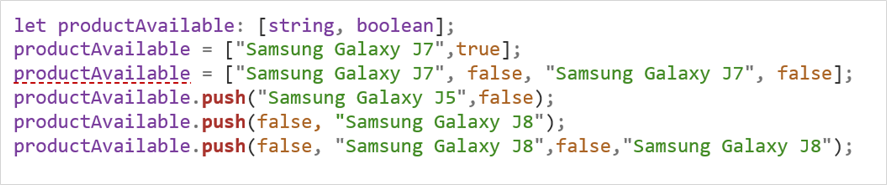
In order to access the customerCreditId, you can use customerCreditInfo[0].

In order to access the customer object, you can use customerCreditInfo[1].

In a shopping cart application, you can use tuples to store product details that should require more than one type of data.

* The order of the first set of data entries while initializing a tuple variable should be the same as the order in which the type is declared in a tuple.
* A developer can initialize only one entry as per TypeScript data restriction length policy.
* A compilation error will be thrown in below two cases:
  + if you are trying to assign multiple entries in the first initialization.
  + if you try to initialize different datatypes directly to the tuple declared variable.
* In order to overcome the above-mentioned compilation errors, you can use the push() method.

**Example:**



In the above example, the underlined error is due to multiple declarations in the first initialization which violates the length restriction policy. To avoid this, the push method can be used as shown in the code.

 Tryout : Tuple

Problem Statement

Consider that a developer needs to declare a Tuple variable named productAvailable consisting of two parameters like string and boolean and needs to populate the message "The product <<productName value>> is available" when availability parameter is true and populate the message "The product <<productName value>> is not available" when availability parameter is false and need have appropriate logic to assign productName respectively.  Below mentioned code-snippet would fit into this requirement.

**Activity:**

* Modify line no 11 as given below and re-execute the code.

productAvailable.push("Samsung Galaxy J5",false,1);

* Modify line no 11 as given below and re-execute the code.

productAvailable.push(false,"Samsung Galaxy J5");

// declaring a Tuple

let productAvailable: [string, boolean];

let productName;

let availability;

// assigning value to Tuple

productAvailable = ['Samsung Galaxy J7', true];

// Adding new value to Tuple

productAvailable.push('Samsung Galaxy J5', false);

// logic to check product availability based on datatype of declared Tuple variable

for (const item of productAvailable) {

if (typeof item === 'string') {

productName = item;

} else if (typeof item === 'boolean') {

availability = item;

if (availability === true) {

console.log(`The product ${productName} is available`);

} else if (availability === false) {

console.log(`The product ${productName} is not available`);

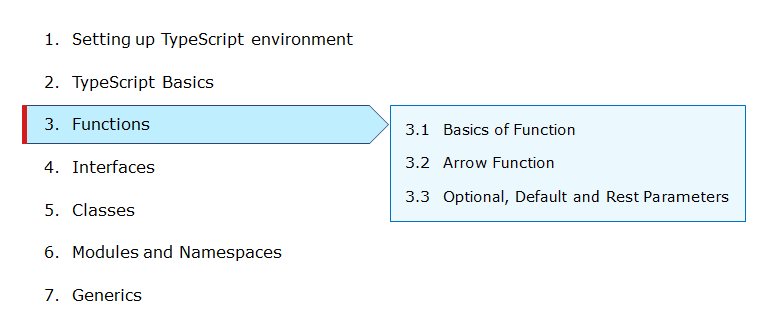
}

}

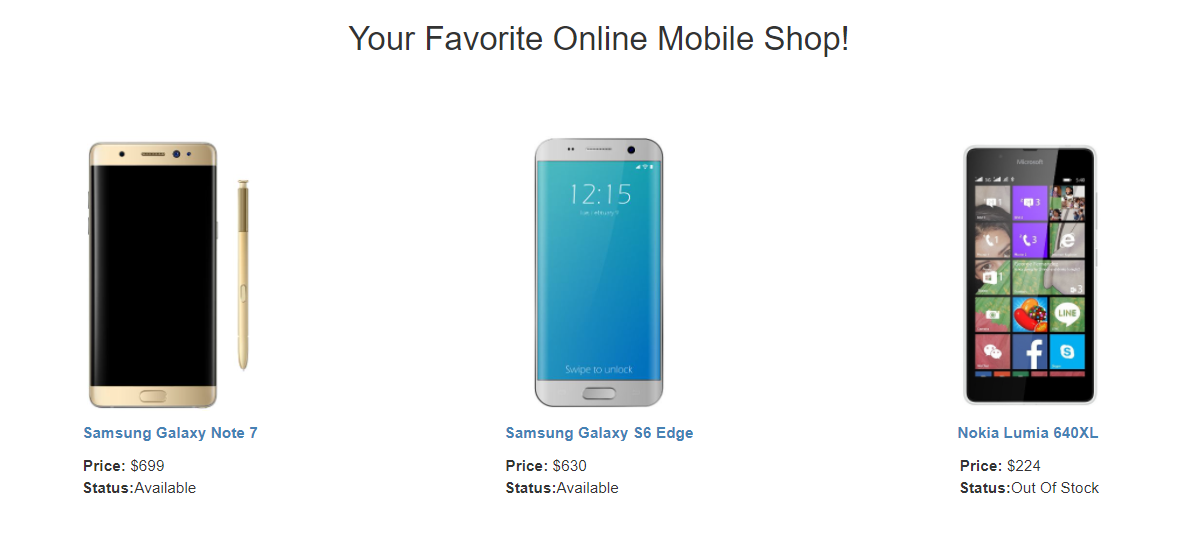
}

At the end of this module, you have learnt:

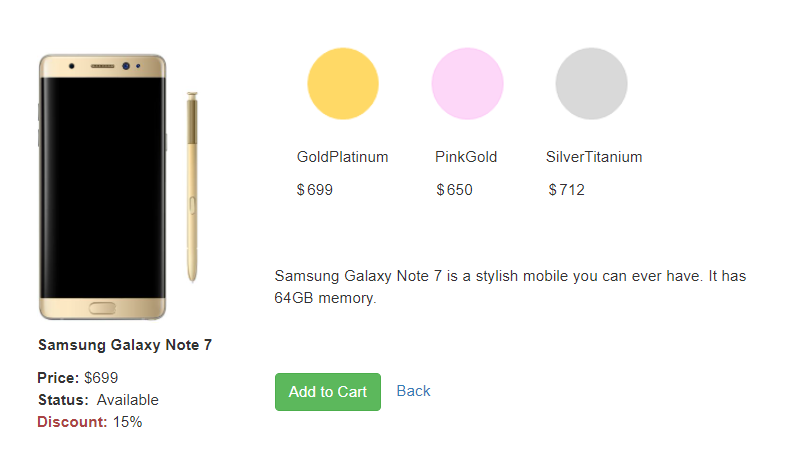
* How to declare variables in TypeScript?
* How to work with Enum objects using TypeScript?
* Implement the Tuple and Arrays concepts through TypeScript.



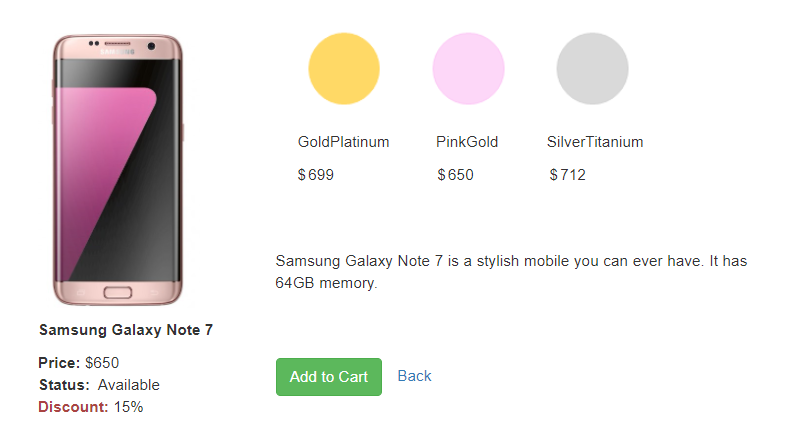
Consider the below page that displays the list of mobile phones available. For each mobile in the list, you can see its image, name, price, and availability status. The property ‘name’ of the mobile is clickable.



When the mobile name link is clicked, the user is navigated to the next screen which shows the details specific to the phone selected. The details such as different colors in which the phone is available, price of mobile according to the color selected, description of the phone, availability status, and discounts if any.



Users can click on each color to view the mobile image for that color. The Price corresponding to the color selected can also be shown. For example: On click of PinkGold the below screen should be rendered:



This requirement is implemented using the **function**concept in TypeScript that helps us to implement business logic and event handlers. Let us discuss functions in TypeScript.

**Function in TypeScript Vs JavaScript:**

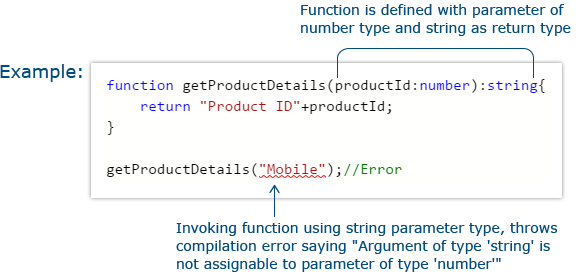
* A function is a block of statements to perform a particular task.
* A sequence of statements written within function forms function body.
* Functions are executed when it is invoked by a function call. Values can be passed to a function as function parameters and the function returns a value.
* Functions in TypeScript are like functions in JavaScript with some additional features.

|  |  |  |
| --- | --- | --- |
|  | TypeScript | JavaScript |
| Types: | Supports | Do not support |
| Required and optional parameters: | Supports | All parameters are optional |
| Function overloading: | Supports | Do not support |
| Arrow functions: | Supports | Supported with ES2015 |
| Default parameters: | Supports | Supported with ES2015 |
| Rest parameters: | Supports | Supported with ES2015 |

The parameter type is the data type of the parameter and the return type is the data type of the returned value from the function.

With the TypeScript function, you can add types to the parameter passed to the function and the function return types.

While defining the function, return the data with the same type as the function return type. While invoking the function you need to pass the data with the same data type as the function parameter type, or else it will throw a compilation error.



Tryout : Parameter Types and Return Types

Problem Statement

Consider that developer needs to declare a function - getMobileByManufacturer which accepts string as input parameter and returns the list of mobiles then below mentioned code-snippet fits into requirement.

**Activity:**

* Comment the line nos 9, 13 and 17 and re-execute the code.
* Modify line no 22 as below and re-execute the code.

console.log("The available mobile list:"+getMobileByManufacturer("Nokia"));

// declaring a function which accepts string datatype as parameter and returns string array

function getMobileByManufacturer(manufacturer: string): string[] {

let mobileList: string[];

if (manufacturer === 'Samsung') {

mobileList = ['Samsung Galaxy S6 Edge', 'Samsung Galaxy Note 7',

'Samsung Galaxy J7 SM-J700F'];

return mobileList;

} else if (manufacturer === 'Apple') {

mobileList = ['Apple iPhone 5s', 'Apple iPhone 6s ', 'Apple iPhone 7'];

return mobileList;

} else {

mobileList = ['Nokia 105', 'Nokia 230 Dual Sim'];

return mobileList;

}

}

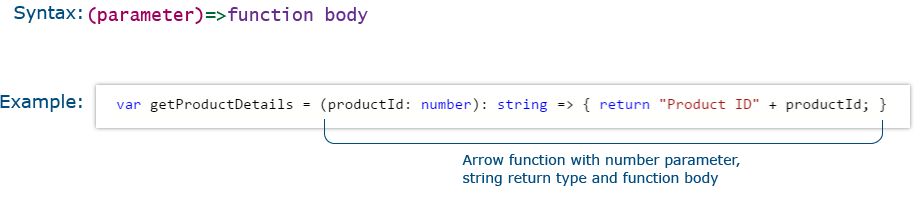
// logic to populate the Samsung manufacturer details on console

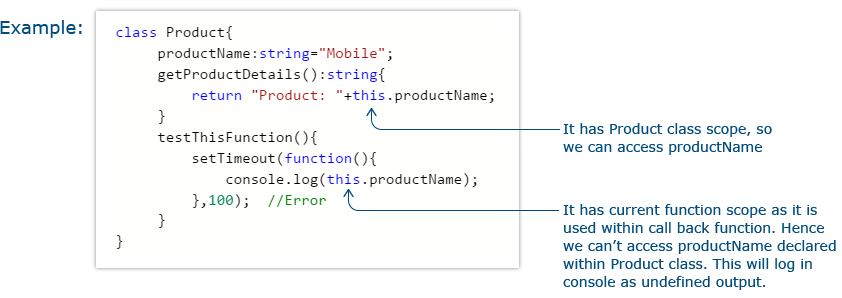
console.log('The available mobile list: ' + getMobileByManufacturer('Samsung'));

Arrow function is a concise way of writing a function. Whenever you need a function to be written within a loop, the arrow function will be the opt choice.

Do not use the function keyword to define an arrow function.

In a shopping cart application, you can use the arrow function to perform filtering, sorting, searching operations, and so on.



****

In a class, if a method wants to access the property of the class it should use this keyword.

For a particular object, this keyword will help to access the properties of the current object. This is possible because all the methods and properties are within the same scope.

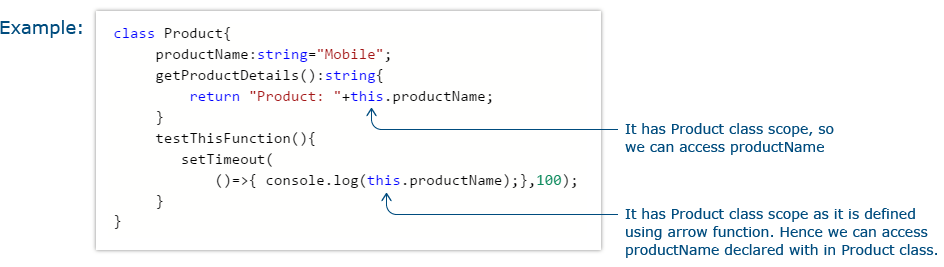
In the above example, when you use this.productName inside the getProductDetails method, getProductDetails method, and productName variable are in the same scope. Also, you get the desired result.

But when you use this.productName inside the setTimeout function, instead of directly using it in testThisFunction method, the scope of this.productName will be inside the setTimeout's callback function and not the testThisFunction method. That is the reason you are unable to access the value of productName for that particular object.

If you need to access the class scope with this keyword inside the callback function then use the arrow function.

Arrow function lexically captures the meaning of this keyword.

Rewrite the same logic using the arrow function as below:

****

In the above code, this.productName is written inside an arrow function. Since the callback function of setTimeout is implemented using the arrow function, it does not create a new scope and it will be in the same scope as the testThisFunction method.

Tryout : Arrow Function

Problem Statement

Consider that developer needs to declare a manufacturer's array holding 4 objects with id and price as a parameter and needs to implement an arrow function - myfunction to populate the id parameter of manufacturers array whose price is greater than or equal to 200 dollars then below mentioned code-snippet would fit into this requirement.

**Activity:**

* Modify the arrow function logic to populate the price value when the id is Apple and re-execute the code.
* Modify the for loop logic to populate only Microsoft manufacture id and price details and re-execute the code.

// declaring an Array with 3 objects

const manufacturers = [{ id: 'Samsung', price: 150 },

{ id: 'Microsoft', price: 200 },

{ id: 'Apple', price: 400 },

{ id: 'Micromax', price: 100 }

];

let test;

// Arrow function to populate the details of Array whose price is greater than 200

function myFunction() {

test = manufacturers.filter((manufacturer) => manufacturer.price >= 200);

}

// self-invoking an arrow function

myFunction();

console.log('Details of Manufacturer array are : ');

// logic to populate the manufacturer array details based on id value

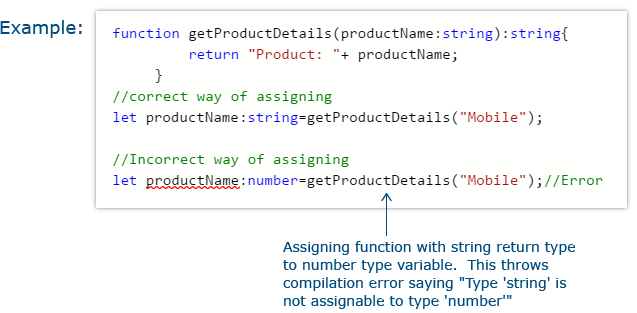
for (const item of test) {

console.log(item.id);

}

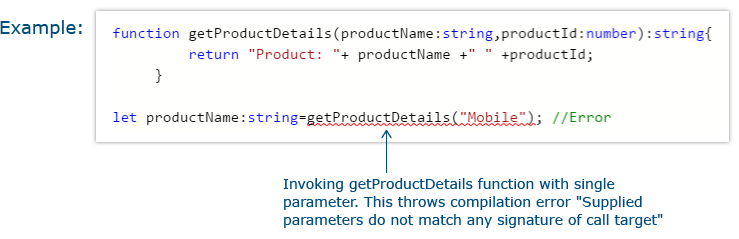
Function types are a combination of parameter types and return type. Functions can be assigned to variables.

While assigning a function to a variable make sure that the variable declaration is the same as the assigned function’s return type.



TypeScript treats every function parameter as mandatory. So when a function is compiled, the compiler will check whether there is a value supplied to all the parameters of the function, or else it will throw a compilation error.

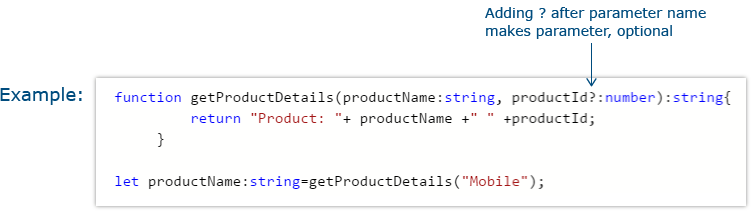
Consider the code below:



In the above example, you have tried to invoke a function with only a single parameter, whereas the definition of the function accepts two parameters. Hence, it will throw a compilation error. Also, optional parameter can be used to tackle this issue.

The Optional parameter is used to make a parameter, optional in a function while invoking the function.

If you rewrite the previous code using an optional parameter, it looks like the below:

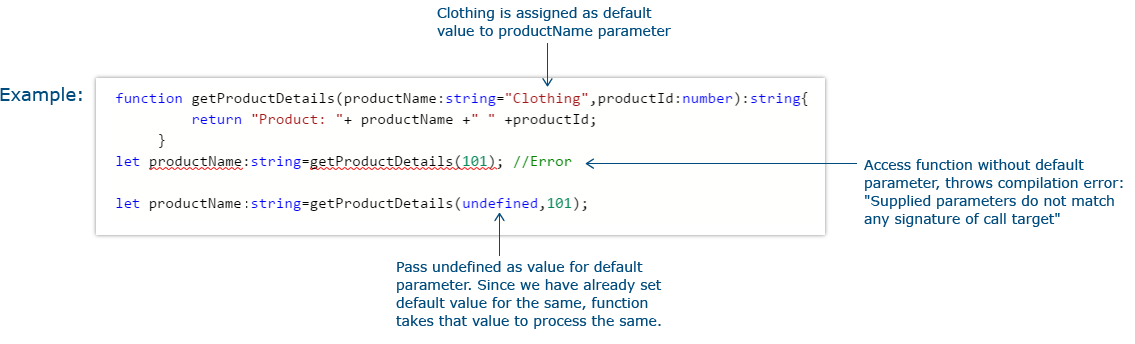


An Optional parameter must appear after all the mandatory parameters of a function.

Default parameter is used to assign the default value to a function parameter.

If the user does not provide a value for a function parameter or provide the value as undefined for it while invoking the function, the default value assigned will be considered.

If the default parameter comes before a required parameter, you need to explicitly pass undefined as the value to get the default value.



Tryout : Optional and Default Parameters

Problem Statement

Consider that developer needs to declare a function - getMobileByManufacturer with two parameters namely manufacturer and id, where manufacturer value should passed as Samsung and id parameter should be optional while invoking the function, if id is passed as 101 then this function should return Moto mobile list and if manufacturer parameter is either Samsung/Apple then this function should return respective mobile list and similar to make Samsung as default Manufacturer. Below mentioned code-snippet would fit into this requirement.

**Activity:**

* Modify the function declaration line no 2 as below and re-execute the code.

function getMobileByManufacturer(manufacturer,id?:number): string[]

* Modify the line no 35 as below and re-execute the code.

console.log("The available mobile list : " + getMobileByManufacturer(undefined));

console.log("The available mobile list : " + getMobileByManufacturer(101));

console.log("The available mobile list : " + getMobileByManufacturer(undefined,102));

// declaring a function with optional parameter

function getMobileByManufacturer(manufacturer: string = 'Samsung', id?: number): string[] {

let mobileList: string[];

// logic to be evaluated if id parameter while invoking above declared function

if (id) {

if (id === 101) {

mobileList = ['Moto G Play, 4th Gen', 'Moto Z Play with Style Mod'];

return mobileList;

}

}

// logic to return mobileList based on manufacturer category

if (manufacturer === 'Samsung') {

mobileList = [' Samsung Galaxy S6 Edge', ' Samsung Galaxy Note 7',

' Samsung Galaxy J7 SM-J700F'];

return mobileList;

} else if (manufacturer === 'Apple') {

mobileList = [' Apple iPhone 5s', ' Apple iPhone 6s', ' Apple iPhone 7'];

return mobileList;

} else {

mobileList = [' Nokia 105', ' Nokia 230 Dual Sim'];

return mobileList;

}

}

// statement to invoke function with optional parameter

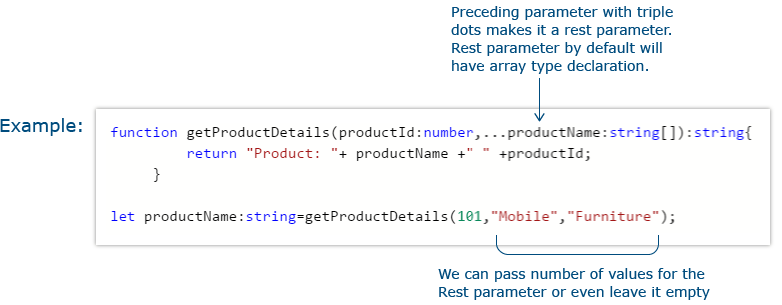
console.log('The available mobile list : ' + getMobileByManufacturer('Apple'));

// statement to invoke function with default parameter

console.log('The available mobile list : ' + getMobileByManufacturer(undefined, 101));

Rest Parameter is used to pass multiple values to a single parameter of a function. It accepts zero or more values for a single parameter.

* Rest Parameter should be declared as an array.
* Precede the parameter to be made as rest parameter with triple dots.
* Rest parameter should be the last parameter in the function parameter list.



Tryout : Rest Parameter

Problem Statement

Consider that developer needs to implement business logic for adding multiple Product values into a cart variable which is type of string array. Below mentioned code-snippet fits into the requirement.

**Activity:**

* Modify line no 2 as given below and re-execute the code.

let cart: string=[];

* Modify line no 8 as given below and re-execute the code.

function addtoCart(productName: string[]): string[]

* Modify line no 18 as given below and re-execute the code.

console.log("Cart Items are:"+addtoCart("Moto G Play, 4th Gen","Apple iPhone 5s",1));

// declaring a empty string array

const cart: string[] = [];

// arrow function logic to push values into cart array

const pushtoCart = (item: string) => { cart.push(item); };

// logic to add items into cart

function addtoCart(...productName: string[]): string[] {

for (const item of productName) {

pushtoCart(item);

}

return cart;

}

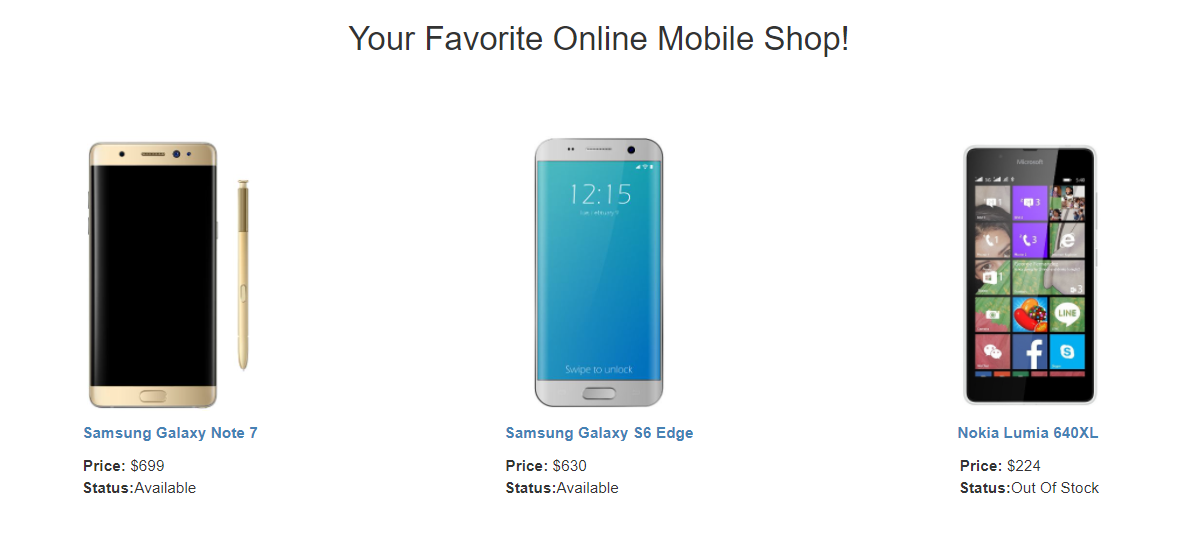
// to populate value on console

console.log('Cart Items are:' + addtoCart(' Moto G Play, 4th Gen', ' Apple iPhone 5s'));

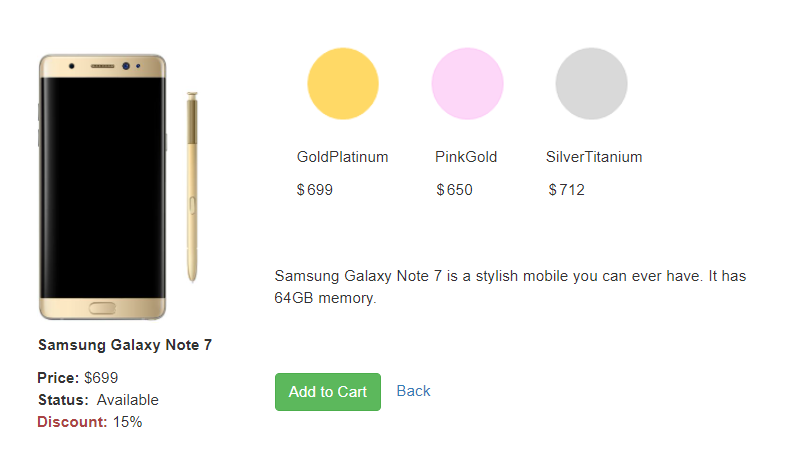
**Problem Statement:**

Consider the below requirement of the Mobile Cart application:

Design the below screen:



On click of the mobile name, it should navigate to the next screen as below:



To implement this, make the click event handler function to accept productName and productId as parameters.

Define an arrow function inside the event handler to filter the product array with the selected product object using the productId received by the function. Pass the selected product object to the next screen.

[

**Hint:**

Use localStorage to store the product object as below:

localStorage.setItem("Product",JSON.stringify(filteredList[0]));

Here filteredList is the array holding the selected product object which needs to be converted to string representation using JSON.stringify function, so that you can access the properties in the next page.

In the ProductDetail page, get the selected product object from the localStorage as below:

 let Product=JSON.parse(localStorage.getItem("Product"));

]

Write a TypeScript code to solve the above requirement. The Complete HTML code is given below:

HomePage.html:

1. <!doctype html>
2. <html>
3. <head>
4. <title>Mobile Cart</title>
5. <meta charset="UTF-8">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
7. <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
8. <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
9. <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
10. <style>
11. .navbar-inverse {
12. background-color: *#005580;*
13. background-image: none;
14. background-repeat: no-repeat;
15. color: *#ffffff;*
16. }
17. .navbar-inverse .navbar-nav>.active>a {
18. color: *#ffffff;*
19. background-color: transparent;
20. }
21. .navbar-inverse .navbar-nav>li>a:hover {
22. text-decoration: none;
23. color: *#ffffff;*
24. }
25. </style>
26. </head>
27. <body>
28. <nav class="navbar navbar-inverse navbar-fixed-top">
29. <div class="navbar-header">
30. <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar"
31. aria-expanded="false" aria-controls="navbar">
32. <span class="sr-only">Toggle navigation</span>
33. <span class="icon-bar"></span>
34. <span class="icon-bar"></span>
35. <span class="icon-bar"></span>
36. </button>
37. <a class="navbar-brand" href="#">Mobile Cart</a>
38. </div>
39. <div id="navbar" class="collapse navbar-collapse">
40. <ul class="nav navbar-nav">
41. <li><a href="#">Home</a></li>
42. </ul>
43. *<!--/.nav-collapse -->*
44. <ul class="nav navbar-nav navbar-middle" style="color:white; margin-right:30px;">
45. <li><a href="Cart.html"><span class="glyphicon glyphicon-shopping-cart" style="color:white"></span></a>
46. </li>
47. </ul>
48. </div>
49. </nav>
50. <div style="margin-top:7%">
51. <center>
52. <h2>Your Favorite Online Mobile Shop!</h2>
53. </center>
54. </div>
55. <div class="container" style="padding-top:5%">
56. <div class="row">
57. <div class="col-md-4">
58. <div style="text-align: center;">
59. <img src="images/SamsungGalaxy\_Gold.jpg" height="250px">
60. </div>
61. <div style="padding-top:10px;">
62. <div onclick="getMobileDetails();" style="cursor:pointer;color:Steelblue;text-align: center;">
63. <b><span id="pName0">Samsung Galaxy Note 7</span></b></div>
64. <div style="padding-top:10px;padding-left: 101px;"><b>Price:</b>&nbsp;&dollar;<span
65. id="pPrice0">699</span></div>
66. <div style="padding-left: 100px;"><b>Status:</b><span id="pAvailable0">Available</span></div>
67. </div>
68. </div>
69. <div class="col-md-4">
70. <div style="text-align: center;">
71. <img src="images/samsung\_edge\_silver.jpg" height="250px">
72. </div>
73. <div style="padding-top:10px;">
74. <div onclick="getMobileDetails('samsungedge',231);"
75. style="cursor:pointer;color:Steelblue;text-align: center;"><b><span id="pName1">Samsung Galaxy
76. S6 Edge</span></b></div>
77. <div style="padding-top:10px;padding-left: 95px;"><b>Price:</b>&nbsp;&dollar;<span
78. id="pPrice1">630</span></div>
79. <div style="padding-left: 94px;"><b>Status:</b><span id="pAvailable1">Available</span></div>
80. </div>
81. </div>
82. <div class="col-md-4">
83. <div style="text-align: center;">
84. <img src="images/lumia\_640xl.jpg" height="250px">
85. </div>
86. <div style="padding-top:10px;">
87. <div onclick="getMobileDetails('lumia',875);"
88. style="cursor:pointer;color:Steelblue;text-align: center; "><b><span id="pName2">Nokia Lumia
89. 640XL</span></b></div>
90. <div style="padding-top:10px;padding-left: 118px;"><b>Price:</b>&nbsp;&dollar;<span
91. id="pPrice2">224</span></div>
92. <div style="padding-left: 117px;"><b>Status:</b><span id="pAvailable2">Out of Stock!</span></div>
93. </div>
94. </div>
95. </div>
96. </div>
97. </body>
98. *<!-- Adding the converted js file -->*
99. <script src="productlist.js"></script>
100. </html>

**Attach the transpiled TypeScript file to the HTML page as highlighted in the above code.**

ProductDetail.html:

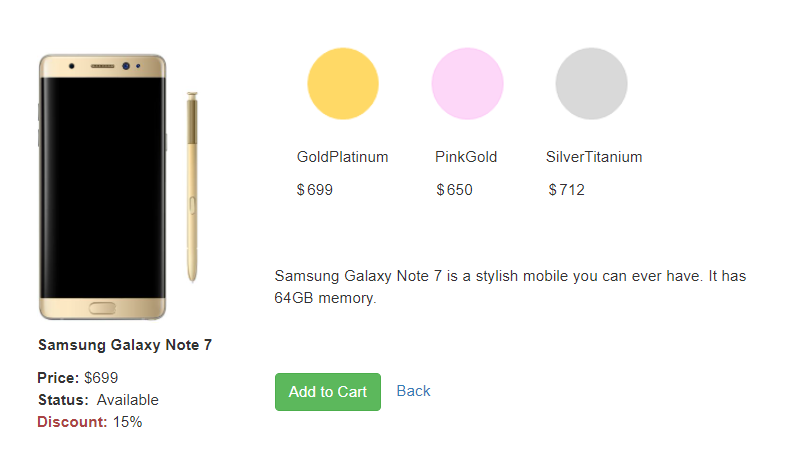
1. <!doctype html>
2. <html>
3. <head>
4. <title>Mobile Cart</title>
5. <meta charset="UTF-8">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
7. <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
8. <style>
9. .navbar-inverse {
10. background-color: *#005580;*
11. background-image: none;
12. background-repeat: no-repeat;
13. color: *#ffffff;*
14. }
15. .navbar-inverse .navbar-nav>.active>a {
16. color: *#ffffff;*
17. background-color: transparent;
18. }
19. .navbar-inverse .navbar-nav>li>a:hover {
20. text-decoration: none;
21. }
22. </style>
23. </head>
24. <body>
25. <nav class="navbar navbar-inverse navbar-fixed-top">
26. <div class="navbar-header">
27. <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar"
28. aria-expanded="false" aria-controls="navbar">
29. <span class="sr-only">Toggle navigation</span>
30. <span class="icon-bar"></span>
31. <span class="icon-bar"></span>
32. <span class="icon-bar"></span>
33. </button>
34. <a class="navbar-brand" href="#">Mobile Cart</a>
35. </div>
36. <div id="navbar" class="collapse navbar-collapse">
37. <ul class="nav navbar-nav">
38. <li><a href="Index.html">Home</a></li>
39. </ul>
40. *<!--/.nav-collapse -->*
41. <ul class="nav navbar-nav navbar-middle" style="color:white; margin-right:30px;">
42. <li><a href="Cart.html"><span class="glyphicon glyphicon-shopping-cart" style="color:white"></span></a>
43. </li>
44. </ul>
45. </div>
46. </nav>
47. <div class="container" style="margin-top:7%">
48. <div class="row">
49. <div class="col-sm-4">
50. <div style="margin-left:40%;padding-top:15px;">
51. <div style="padding:15px;">
52. <div>
53. <img id="phoneImage" src="Images/Part 1/SamsungGalaxy\_Gold.JPG" height="250px">
54. </div>
55. <div style="padding-top:10px;">
56. <div><b><span id="pName"></span></b></div>
57. <div style="padding-top:10px;"><b>Price:</b>&nbsp;&dollar;<span id="pPrice"></span></div>
58. <div><b>Status:</b>&nbsp;<span id="pAvailable"></span></div>
59. <div><b class="text-danger">Discount:</b>&nbsp;<span id="pDiscount"></span></div>
60. </div>
61. </div>
62. </div>
63. </div>
64. <div></div>
65. <div style="padding-top:15px;">
66. <div>
67. <img src="Images/Part 1/goldmobile.png" style="padding-left:15px;cursor:pointer;"
68. onclick="getMobileByColor('gold');">
69. <img src="Images/Part 1/pinkmobile.png" style="cursor:pointer;padding-left:15px;"
70. onclick="getMobileByColor('pink');">
71. <img src="Images/Part 1/silvermobile.png" style="cursor:pointer;padding-left:15px;"
72. onclick="getMobileByColor('silver');">
73. </div>
74. <div style="padding-top:10px;">
75. <span style="padding-left:20px;">
76. GoldPlatinum
77. </span>
78. <span style="padding-left:38px;">
79. PinkGold
80. </span>
81. <span style="padding-left:40px;">
82. SilverTitanium
83. </span>
84. </div>
85. <div style="padding-top:10px;">
86. <span style="padding-left: 20px;">&dollar;</span><span id="gold" style="padding-left: 2px;"></span>
87. <span style="padding-left: 90px;">&dollar;</span><span id="pinkgold"
88. style="padding-left: 2px;"></span>
89. <span style="padding-left: 65px;">&dollar;</span><span id="silver"
90. style="padding-left: 2px;"></span>
91. <div id="productDescription" style="padding-top:5%;width:70%;text-align:justify">
92. Samsung Galaxy Note 7 is a stylish mobile you can ever have. It has 64GB memory.
93. </div>
94. <div style="padding-top:5%;padding-right:10%">
95. <button class="btn btn-success" onclick="addtoCart();">Add to Cart</button>
96. <a style="padding-left:10px;" onclick="backHome();">Back</a>
97. </div>
98. </div>
99. </div>
100. </div>
101. </div>
102. </body>
103. <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
104. <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
105. *<!-- Adding converted js code-->*
106. <script src="productdetail.js"></script>
107. </html>

**Attach the transpiled TypeScript file to the HTML page as highlighted in the above code.**

Modify the event handler function to make productId as optional parameter and productName as default parameter (Samsung Galaxy Note7)

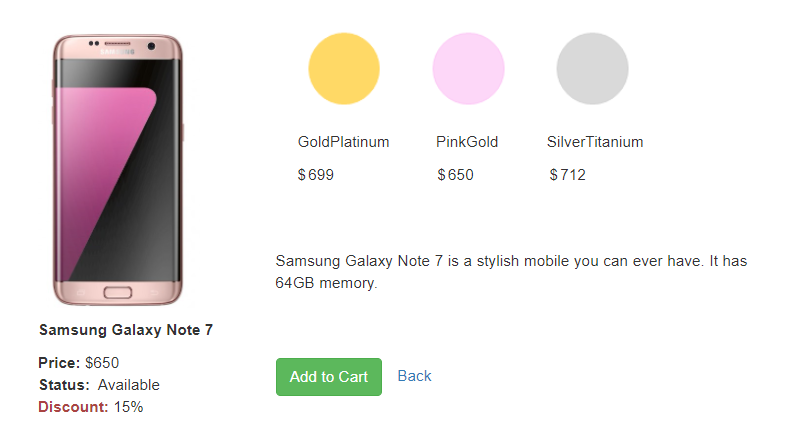
**Problem Statement:**

In the previous example we have navigated from the home page to the product detail page given below:



In this page, on click of each color, the corresponding mobile image, and price have to be updated on the screen.

**Example:** On click of PinkGold the below screen should be rendered:



Write an event handler that accepts color as the string parameter. Depending on the color value update the product details. The image and price of the product should be changed based on the color which has been clicked.

Below is the complete HTML code, write TypeScript code to solve the requirement.

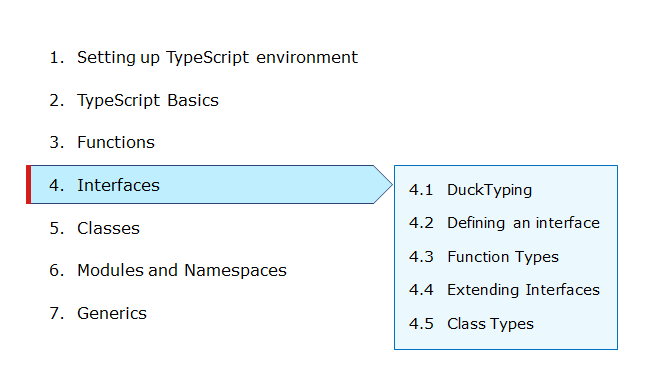
ProductDetail.html

1. <!doctype html>
2. <html>
3. <head>
4. <title>Mobile Cart</title>
5. <meta charset="UTF-8">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
7. <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
8. <style>
9. .navbar-inverse {
10. background-color: *#005580;*
11. background-image: none;
12. background-repeat: no-repeat;
13. color: *#ffffff;*
14. }
15. .navbar-inverse .navbar-nav>.active>a {
16. color: *#ffffff;*
17. background-color: transparent;
18. }
19. .navbar-inverse .navbar-nav>li>a:hover {
20. text-decoration: none;
21. }
22. </style>
23. </head>
24. <body>
25. <nav class="navbar navbar-inverse navbar-fixed-top">
26. <div class="navbar-header">
27. <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar"
28. aria-expanded="false" aria-controls="navbar">
29. <span class="sr-only">Toggle navigation</span>
30. <span class="icon-bar"></span>
31. <span class="icon-bar"></span>
32. <span class="icon-bar"></span>
33. </button>
34. <a class="navbar-brand" href="#">Mobile Cart</a>
35. </div>
36. <div id="navbar" class="collapse navbar-collapse">
37. <ul class="nav navbar-nav">
38. <li><a href="Index.html">Home</a></li>
39. </ul>
40. *<!--/.nav-collapse -->*
41. <ul class="nav navbar-nav navbar-middle" style="color:white; margin-right:30px;">
42. <li><a href="Cart.html"><span class="glyphicon glyphicon-shopping-cart" style="color:white"></span></a>
43. </li>
44. </ul>
45. </div>
46. </nav>
47. <div class="container" style="margin-top:7%">
48. <div class="row">
49. <div class="col-sm-4">
50. <div style="margin-left:40%;padding-top:15px;">
51. <div style="padding:15px;">
52. <div>
53. <img id="phoneImage" src="Images/Part 1/SamsungGalaxy\_Gold.JPG" height="250px">
54. </div>
55. <div style="padding-top:10px;">
56. <div><b><span id="pName"></span></b></div>
57. <div style="padding-top:10px;"><b>Price:</b>&nbsp;&dollar;<span id="pPrice"></span></div>
58. <div><b>Status:</b>&nbsp;<span id="pAvailable"></span></div>
59. <div><b class="text-danger">Discount:</b>&nbsp;<span id="pDiscount"></span></div>
60. </div>
61. </div>
62. </div>
63. </div>
64. <div></div>
65. <div style="padding-top:15px;">
66. <div>
67. <img src="Images/Part 1/goldmobile.png" style="padding-left:15px;cursor:pointer;"
68. onclick="getMobileByColor('gold');">
69. <img src="Images/Part 1/pinkmobile.png" style="cursor:pointer;padding-left:15px;"
70. onclick="getMobileByColor('pink');">
71. <img src="Images/Part 1/silvermobile.png" style="cursor:pointer;padding-left:15px;"
72. onclick="getMobileByColor('silver');">
73. </div>
74. <div style="padding-top:10px;">
75. <span style="padding-left:20px;">
76. GoldPlatinum
77. </span>
78. <span style="padding-left:38px;">
79. PinkGold
80. </span>
81. <span style="padding-left:40px;">
82. SilverTitanium
83. </span>
84. </div>
85. <div style="padding-top:10px;">
86. <span style="padding-left: 20px;">&dollar;</span><span id="gold" style="padding-left: 2px;"></span>
87. <span style="padding-left: 90px;">&dollar;</span><span id="pinkgold"
88. style="padding-left: 2px;"></span>
89. <span style="padding-left: 65px;">&dollar;</span><span id="silver"
90. style="padding-left: 2px;"></span>
91. <div id="productDescription" style="padding-top:5%;width:70%;text-align:justify">
92. Samsung Galaxy Note 7 is a stylish mobile you can ever have. It has 64GB memory.
93. </div>
94. <div style="padding-top:5%;padding-right:10%">
95. <button class="btn btn-success" onclick="addtoCart();">Add to Cart</button>
96. <a style="padding-left:10px;" onclick="backHome();">Back</a>
97. </div>
98. </div>
99. </div>
100. </div>
101. </div>
102. </body>
103. <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
104. <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
105. *<!-- Adding converted js code-->*
106. <script src="productdetail.js"></script>
107. </html>

**Attach the transpiled TypeScript file to the HTML page as highlighted in the above code.**

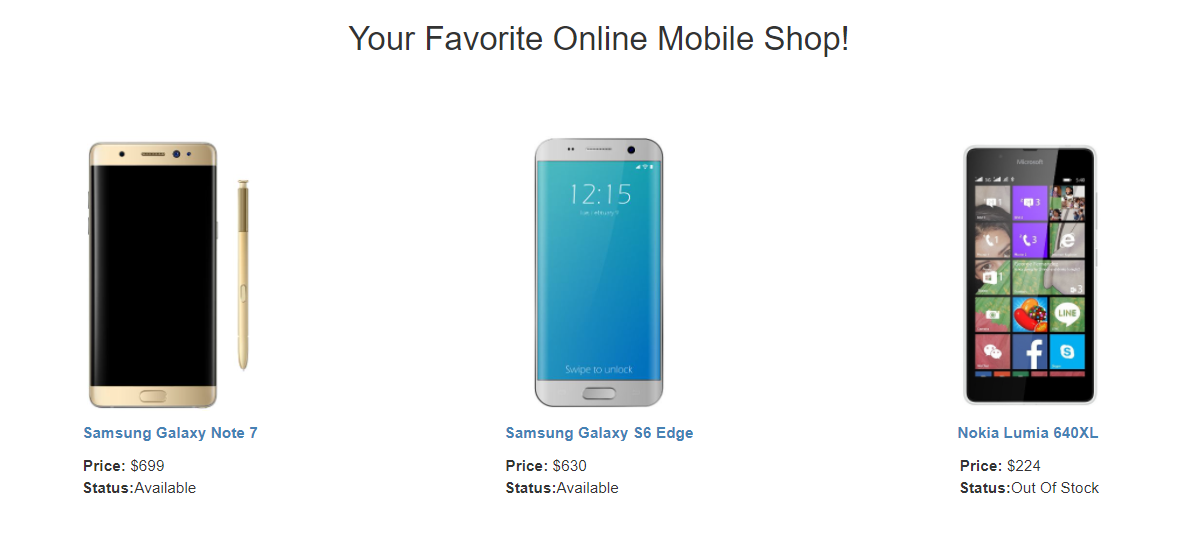
At the end of this module, you have learnt:

* Importance of Functions.
* Different Function Parameter types and their implementation.
* How to work with Arrow Functions and Rest Parameter using TypeScript Syntax?



* Interfaces can be used to impose consistency among various TypeScript classes.
* Any class which implements an interface should implement all the required members of that interface.
* Interfaces can be used to ensure that proper values are being passed into functions, properties as well as constructors.
* Interfaces can be used to achieve additional flexibility as well as loosely coupling in an application.
* Any object which implements an interface can be passed as a parameter to a function whose parameter type is declared the same as the interface.

Consider the below screen of the Mobile Cart application:

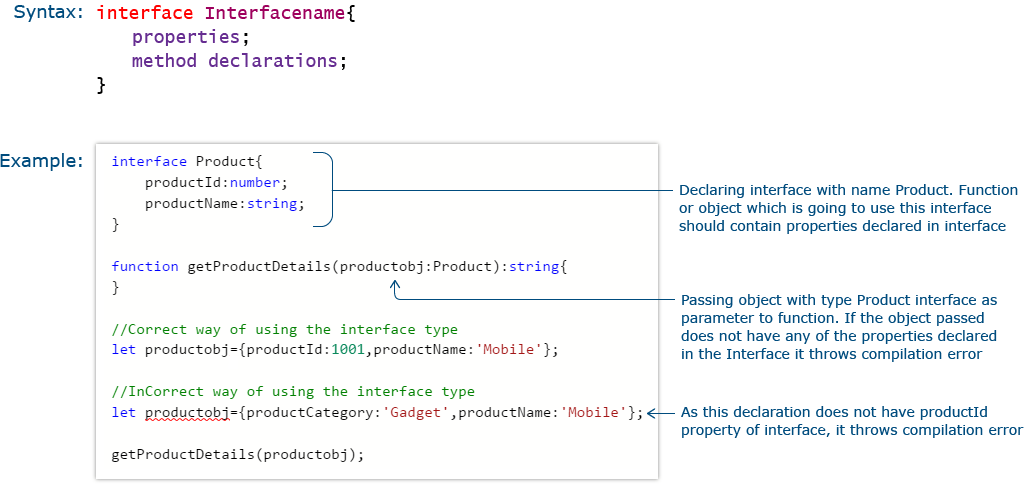


Here an array is used to store the product information. But not restricted the type of object to be stored in the array. You can restrict the array which contains only a particular type of object. For this, use Interface.

Let us discuss more on Interface in TypeScript.

An interface in TypeScript is used to define contracts within the code.

* Interfaces are used to enforce type checking by defining contracts.
* It is a collection of properties and method declarations.
* Interfaces are not supported in JavaScript and will get removed from the generated JavaScript.
* Interfaces are mainly used to identify issues upfront as we proceed with coding.


* 

Tryout : How to create an Interface

Problem Statement

Consider that a developer needs to declare an interface named - Product with two properties like productId and productName with a number and string datatype and need to implement logic to populate the Product details using this interface then this code-snippet would work fine.

**Activity:**

* Modify line no 13 as given below and re-execute the code.

let prodObject={productName:'Mobile', productPrice:10000};

* Replace the below-mentioned code at line no 13 and modify line no 9 to populate all the details of the Product onto the console.

let prodObject={productId: 1001, productName:'Mobile', productPrice:10000};

// declaring an interface

interface Product {

productId: number ;

productName: string ;

}

// logic to display the Product details with interface object as parameter

function getProductDetails(productobj: Product): string {

return 'The product name is : ' + productobj.productName;

}

// declaring a variable having interface properties

const prodObject = {productId: 1001, productName: 'Mobile'};

// declaring variable and invoking Product details function

const productDetails: string = getProductDetails(prodObject);

// line to populate the created product on console

console.log(productDetails);

Duck-Typing is a rule for checking the type compatibility for more complex variable types.

TypeScript compiler uses the duck-typing method to compare one object with the other by comparing that both the objects have the same properties with the same data types.

TypeScript interface uses the same duck typing method to enforce type restriction. If an object that has been assigned as an interface contains more properties than the interface mentioned properties, it will be accepted as an interface type and additional properties will be ignored for type checking

Let us rewrite the previous example to a pass additional parameter.

Tryout : Duck Typing

Problem Statement

Consider that a developer needs to declare an interface named - Product with two properties like productId and productName with the number and string datatype and need to implement logic to populate the Product details using this interface then this code-snippet would work fine.

**Activity:**

* Comment lines 3 and 4 and re-execute the code.
* Add one more property as productCategory into the declared Product interface and display all the Product details on the console.

// declaring an interface

interface Product {

productId: number;

productName: string;

}

// logic to display the Product details with interface object as parameter

// tslint:disable-next-line:adjacent-overload-signatures

function getProductDetails(productobj: Product): string {

return 'The product name is : ' + productobj.productName;

}

// declaring a variable along with interface properties

const prodObject = {productId: 1001, productName: 'Mobile', productCategory: 'Gadget'};

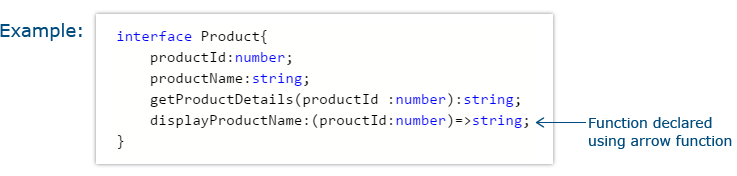
// declaring variable and invoking Product details function

const productDetails: string = getProductDetails(prodObject);

// line to populate the created product variable on console

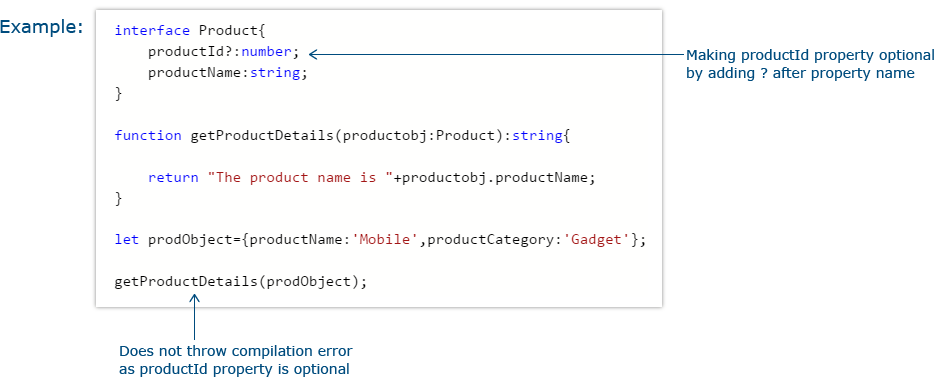
console.log(productDetails);

* Interface keyword is used to declare an interface.
* An interface should have properties and method declarations.
* Properties or methods in an interface should not have any access modifiers.
* Properties cannot be initialized in a TypeScript interface.



In a TypeScript interface, if certain properties need to be made optional, you can make them optional by adding ‘?’ after the property name.

Let us rewrite the duck typing example with the optional property.



Tryout : Interfaces Optional Property

Problem Statement

Consider that a developer needs to declare an interface named - Product with two properties like productId and productName with a number and string datatype, where productId should be declared as an optional parameter and need to implement logic to populate the Product details using this interface then this code-snippet would work fine.

**Activity:**

* Add one more property as productCategory as an optional parameter into the declared Product interface and display all the Product details on the console.

// declaring an interface with optional parameter

interface Product {

productId?: number;

productName: string;

}

// logic to display the Product details with interface object as parameter

function getProductDetails(productobj: Product): string {

return 'The product name is ' + productobj.productName;

}

// declaring a variable along with interface properties

const prodObject = {productName: 'Mobile', productCategory: 'Gadget'};

// declaring variable and invoking Product details function

const productDetails: string = getProductDetails(prodObject);

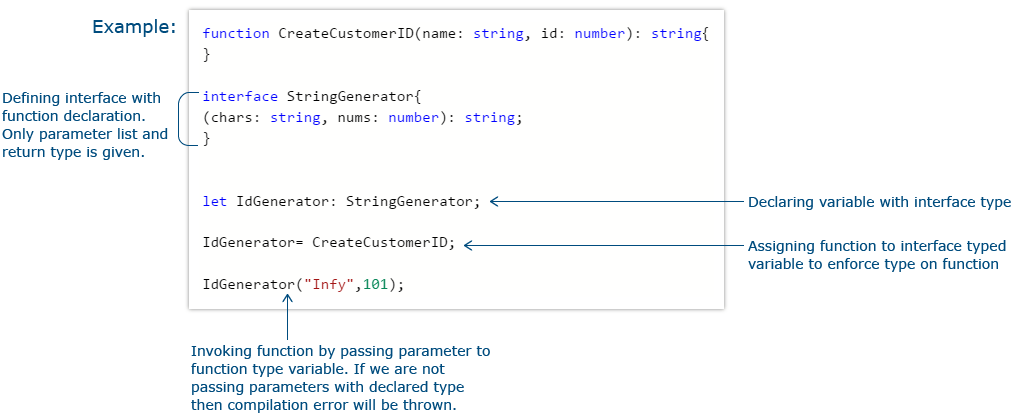
// line to populate the created product variable on console

console.log(productDetails);

Interfaces can be used to define the structure of functions like defining structure of objects.

Once the interface for a function type is declared, you can declare variables of that type and assign functions to the variable if the function matches the signature defined in the interface.

Function type interface is used to enforce the same number and type of parameters to any function which is been declared with the function type interface.

Tryout : Function Types

Problem Statement

Consider that a developer needs to declare an interface with function type and access its value below mentioned code-snippet would work fine.

**Activity:**

* Modify the line no 8 as below mentioned and re-execute the code.

(chars: string): string;

* Add below mentioned code at line no 19 and populate the respective details after line no 21.

let id2: string = IdGenerator("Mr.Sam", 102);

// declaring a function

function CreateCustomerID(name: string, id: number): string {

return 'The customer id is ' + name + ' ' + id;

}

// declaring a interface with function type

interface StringGenerator {

(chars: string, nums: number): string;

}

// creating reference variable of above declared interface

let idGenerator: StringGenerator;

// assignment of function to interface reference variable

idGenerator = CreateCustomerID;

// declaring a string parameter to hold return value of function type interface

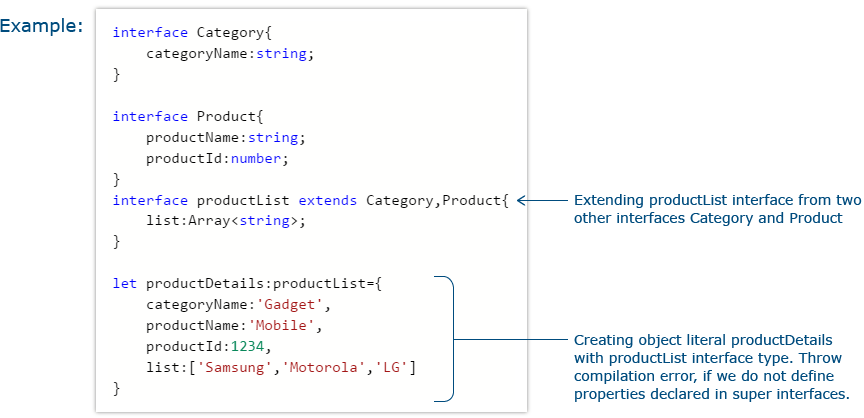
const customerId: string = idGenerator('Mr.Tom', 101);

// line to populate the Customer Details

console.log(customerId);

An interface can be extended from an already existing one using the extends keyword.

In the code below, extend the productList interface from both the Category interface and Product interface.



Tryout : Extending Interfaces

Problem Statement

Consider that a developer needs to declare a productList interface which extends properties from two other declared interfaces like Category,Product as well as implementation to create a variable of this interface type.

**Activity:**

* Comment line nos 20 and 21 and re-execute the code.
* Declare one more variable of type productList interface and populate its details on console.

// declaring a Category interface

interface Category {

categoryName: string;

}

// declaring a Product interface

interface Product {

productName: string;

productId: number;

}

// declaring a ProductList interface which is extends from Category and Product interfaces

interface ProductList extends Category, Product {

list: Array<string>;

}

// declaring a variable which is type of ProductList interface

const productDetails: ProductList = {

categoryName: 'Gadget',

productName: 'Mobile',

productId: 1234,

list: ['Samsung', 'Motorola', 'LG']

};

// assigning list value of productDetails variable into another variable

const listProduct = productDetails.list;

// assigning productName value of productDetails variable into another variable

const pname: string = productDetails.productName;

// line to populate Product name

console.log('Product Name is ' + pname);

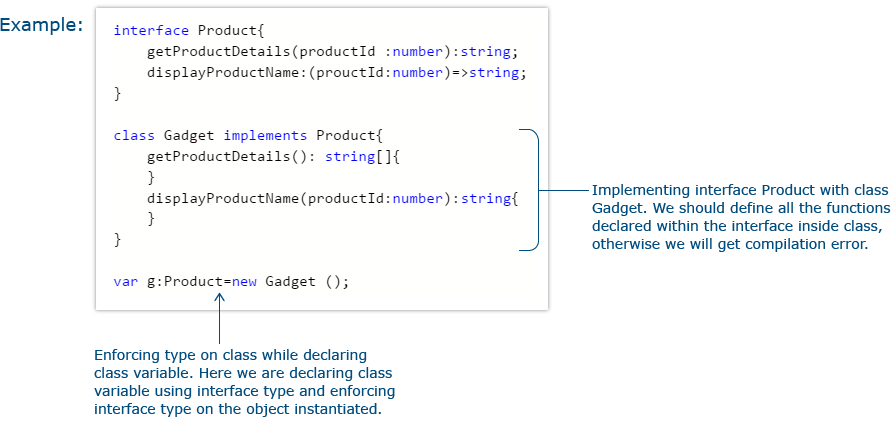
// line to populate Product list

console.log('Product List is ' + listProduct);

Make use of the interface to define class types to explicitly enforce that a class meets a particular contract. Use implements keyword to implement interface inside a class.

To enforce interface type on a class, while instantiating an object of a class declare it using the interface type.

The only interface declared functions and properties will be available with the instantiated object.

****

Tryout : Class Types

Problem Statement

Consider that a developer needs to declare a class named Gadget which implements the two methods named - getProductDetails and displayProductName from a Product interface and consists of its own method named getGadget as well as displays the Product name on console.

**Activity:**

* Comment line no 4 and re-execute the code.
* Create a variable named productDetails and hold return value of getProductDetails method and populate the details on console.

// declaring a Product interface

interface Product {

displayProductName: (prouctId: number) => string;

getProductDetails(): string[];

}

// declaring Gadget class which implements Product interface

class Gadget implements Product {

getProductDetails(): string[] {

return ['Samsung', 'LG', 'Moto'];

}

displayProductName(productId: number): string {

if (productId === 101) {

return 'Product Name is Mobile';

} else if ( productId === 201) {

return 'Product Name is Tablet';

}

}

getGadget(): string[] {

return ['Mobile', 'Tablet', 'iPad', 'iPod'];

}

}

// creating instance of class of interface type

const gadget: Product = new Gadget();

// creating variable to hold return value of displayProductName function

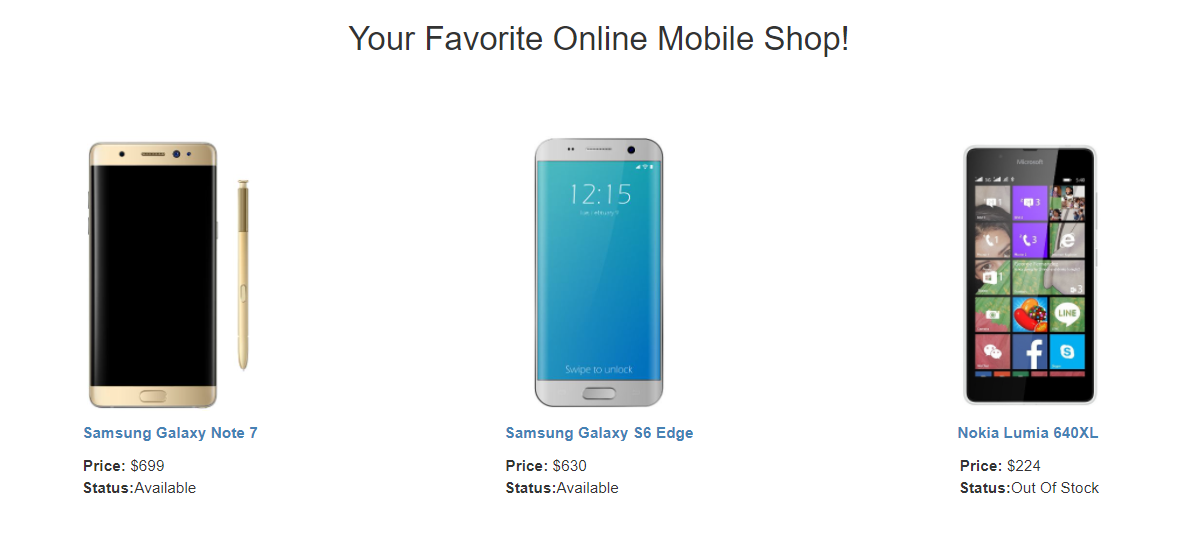
const productName: string = gadget.displayProductName(101);

// line to populate Product name on console

console.log(productName);

**Problem Statement:**

Consider the below screen of the Mobile Cart application which you have designed in the previous exercise:



The product array objects used in this example are not restricted to a specific type. Restrict the objects of the product array to a specific Interface type.

**The object must have the below properties:**

    pId:number;  
    productName: string;  
    productPrice: number;  
    productAvailable: boolean;  
    imageUrl:string;  
    productDescription:string;

Update the TypeScript code with the above requirement and render it in the HTML page.

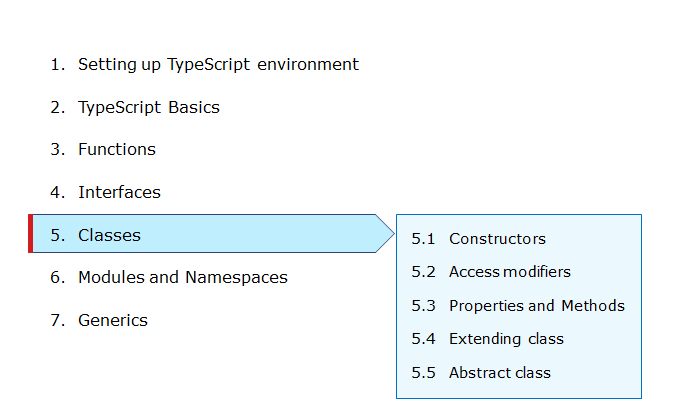
The complete HTML code is given below:

1. <!doctype html>
2. <html>
3. <head>
4. <title>Mobile Cart</title>
5. <meta charset="UTF-8">
6. <meta name="viewport" content="width=device-width, initial-scale=1">
7. <link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/css/bootstrap.min.css" rel="stylesheet">
8. <script src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>
9. <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.7/js/bootstrap.min.js"></script>
10. <style>
11. .navbar-inverse {
12. background-color: *#005580;*
13. background-image: none;
14. background-repeat: no-repeat;
15. color: *#ffffff;*
16. }
17. .navbar-inverse .navbar-nav>.active>a {
18. color: *#ffffff;*
19. background-color: transparent;
20. }
21. .navbar-inverse .navbar-nav>li>a:hover {
22. text-decoration: none;
23. color: *#ffffff;*
24. }
25. </style>
26. </head>
27. <body>
28. <nav class="navbar navbar-inverse navbar-fixed-top">
29. <div class="navbar-header">
30. <button type="button" class="navbar-toggle collapsed" data-toggle="collapse" data-target="#navbar"
31. aria-expanded="false" aria-controls="navbar">
32. <span class="sr-only">Toggle navigation</span>
33. <span class="icon-bar"></span>
34. <span class="icon-bar"></span>
35. <span class="icon-bar"></span>
36. </button>
37. <a class="navbar-brand" href="#">Mobile Cart</a>
38. </div>
39. <div id="navbar" class="collapse navbar-collapse">
40. <ul class="nav navbar-nav">
41. <li><a href="#">Home</a></li>
42. </ul>
43. *<!--/.nav-collapse -->*
44. <ul class="nav navbar-nav navbar-middle" style="color:white; margin-right:30px;">
45. <li><a href="Cart.html"><span class="glyphicon glyphicon-shopping-cart" style="color:white"></span></a>
46. </li>
47. </ul>
48. </div>
49. </nav>
50. <div style="margin-top:7%">
51. <center>
52. <h2>Your Favorite Online Mobile Shop!</h2>
53. </center>
54. </div>
55. <div class="container" style="padding-top:5%">
56. <div class="row">
57. <div class="col-md-4">
58. <div style="text-align: center;">
59. <img src="images/SamsungGalaxy\_Gold.jpg" height="250px">
60. </div>
61. <div style="padding-top:10px;">
62. <div onclick="getMobileDetails();" style="cursor:pointer;color:Steelblue;text-align: center;">
63. <b><span id="pName0">Samsung Galaxy Note 7</span></b></div>
64. <div style="padding-top:10px;padding-left: 101px;"><b>Price:</b>&nbsp;&dollar;<span
65. id="pPrice0">699</span></div>
66. <div style="padding-left: 100px;"><b>Status:</b><span id="pAvailable0">Available</span></div>
67. </div>
68. </div>
69. <div class="col-md-4">
70. <div style="text-align: center;">
71. <img src="images/samsung\_edge\_silver.jpg" height="250px">
72. </div>
73. <div style="padding-top:10px;">
74. <div onclick="getMobileDetails('samsungedge',231);"
75. style="cursor:pointer;color:Steelblue;text-align: center;"><b><span id="pName1">Samsung Galaxy
76. S6 Edge</span></b></div>
77. <div style="padding-top:10px;padding-left: 95px;"><b>Price:</b>&nbsp;&dollar;<span
78. id="pPrice1">630</span></div>
79. <div style="padding-left: 94px;"><b>Status:</b><span id="pAvailable1">Available</span></div>
80. </div>
81. </div>
82. <div class="col-md-4">
83. <div style="text-align: center;">
84. <img src="images/lumia\_640xl.jpg" height="250px">
85. </div>
86. <div style="padding-top:10px;">
87. <div onclick="getMobileDetails('lumia',875);"
88. style="cursor:pointer;color:Steelblue;text-align: center; "><b><span id="pName2">Nokia Lumia
89. 640XL</span></b></div>
90. <div style="padding-top:10px;padding-left: 118px;"><b>Price:</b>&nbsp;&dollar;<span
91. id="pPrice2">224</span></div>
92. <div style="padding-left: 117px;"><b>Status:</b><span id="pAvailable2">Out of Stock!</span></div>
93. </div>
94. </div>
95. </div>
96. </div>
97. </body>
98. *<!-- Adding the converted js file -->*
99. <script src="productlist.js"></script>
100. </html>

**Attach the transpiled TypeScript file to the HTML page as highlighted in the above code.**

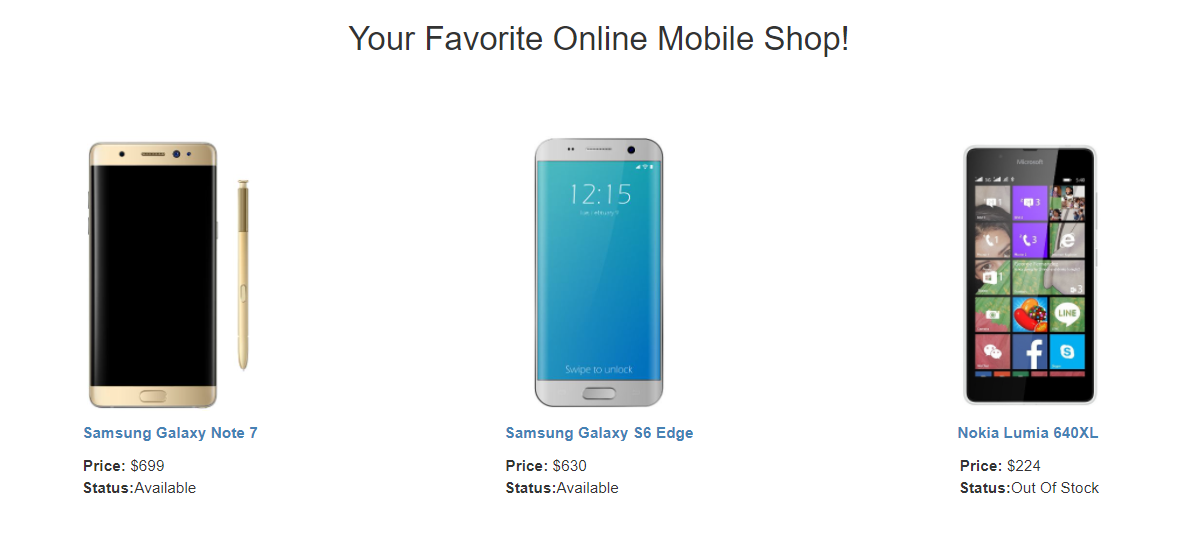
At the end of this module, you have learnt :

* The need for Interface.
* How to work with Function and Class Types Interface.

* 

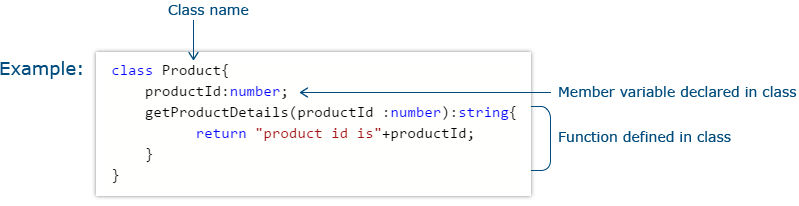
Classes are used to create reusable components. Till the ES5 version of JavaScript, you do not have a class concept as such. For implementing reusable components, use functions and prototype-based inheritance. TypeScript provides an option for the developers to use object-oriented programming with the help of classes.

In the Mobile Cart application, you can use a class to define the product and create various objects of different products. In the below screen, creating different objects of product and rendering the details



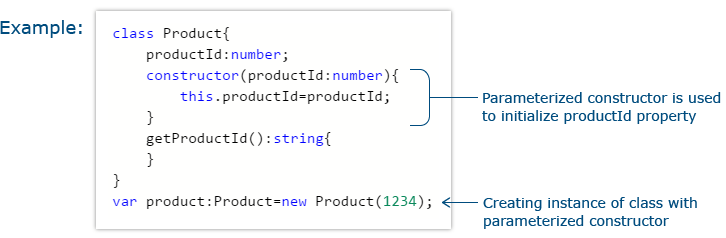
Let us discuss more on classes in TypeScript.

* Class is a template from which objects can be created.
* It provides behavior and state storage.
* It is meant for implementing reusable functionality.
* Use a class keyword to create a class.



A constructor is a function that gets executed automatically whenever an instance of a class is created using a new keyword.

* To create a constructor, a function with the name as a "constructor" is used.
* A class can hold a maximum of one constructor method per class.
* You can use optional parameters with a constructor function as well.



# **Tryout : Constructor**

Problem Statement

Consider that a developer needs to declare a class named - Product with the below-mentioned declarations:

* productId as number property
* Constructor to initialize this value
* getProductId method to return the message "Product id is <<id value>>".

**Activity:**

* Create another object of Product class after line no 15 and populate its details on console.
* Modify the declared Product class by adding another property named productPrice of number datatype, declare a method named getProductPrice, and with the help of product object, you need to populate the message "Product price is << value>>" on the console.

// declaring a Product class

class Product {

static productPrice: string;

productId: number;

// constructor declaration

constructor(productId: number) {

this.productId = productId;

}

getProductId(): string {

return 'Product id is : ' + this.productId;

}

}

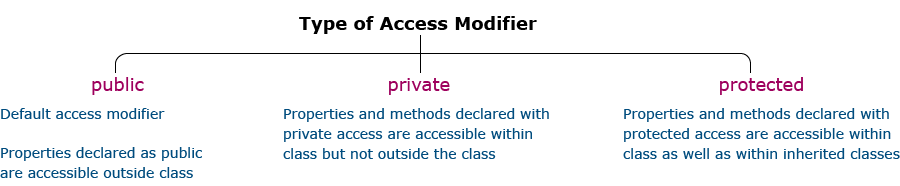
// creation of Product class object

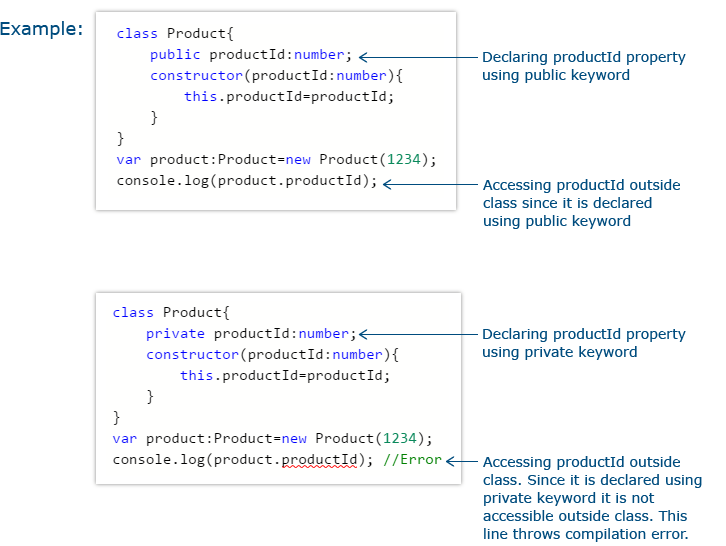
const product: Product = new Product(1234);

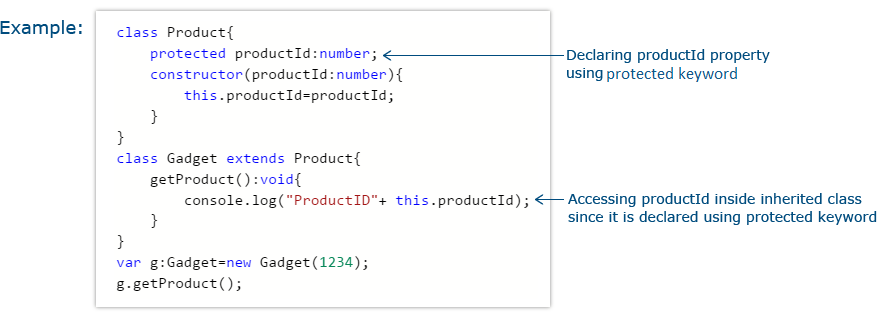
// line to populate the product id details

console.log(product.getProductId());

Access modifiers are used to provide certain restriction of accessing the properties and methods outside the class.

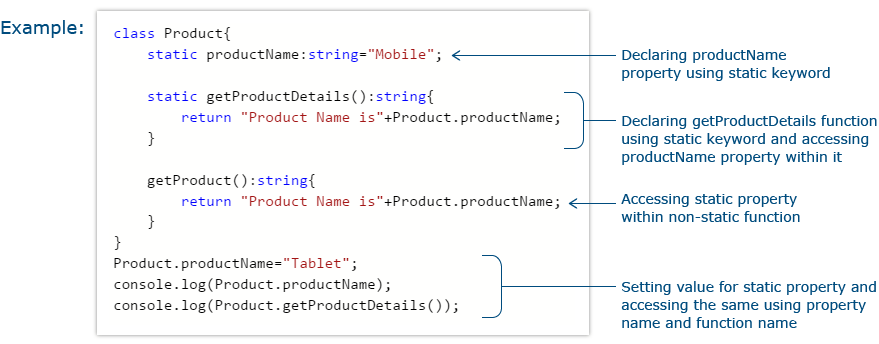






TypeScript provides an option to add a static keyword. This keyword can be used to declare a class variable or method.

* A static variable belongs to the class and not to the instance of the class.
* A variable or function declared with a static keyword can be accessed using the class name instead of the instance of the class.
* Static variables are initialized only once, at the start of the execution.
* A single copy of the static variables would be shared by all instances of the class.
* A static variable can be accessible inside the static function as well as the non-static function.
* A static function can access only static variables and other static functions.


* 

# **Tryout : Access Modifiers**

Problem Statement

Consider that a developer needs to create a Product class with 4 properties namely productId, productName, productPrice, productCategory with private, public, static, and protected access modifiers and accessing them through Gadget class and its methods.

**Activity:**

* Modify the line nos 5 and 6 as given below and re-execute the code.

productPrice: number=150;

private productCategory: string;

* Create another object of Gadget class and populate the product id details on the console.

// declaring a Product class with access modifiers

class Product {

static productPrice = 150;

private productId: number;

public productName: string;

protected productCategory: string;

// declaration of constructor with 3 parameters

constructor(productId: number, productName , productCategory) {

this.productId = productId;

this.productName = productName;

this.productCategory = productCategory;

}

// method ot display product id details

getProductId() {

console.log('The Product id is : ' + this.productId);

}

}

// declaring a Gadget class extending the properties from Product class

class Gadget extends Product {

// method to display productCategory property

getProduct(): void {

console.log('Product category is : ' + this.productCategory);

}

}

// Gadget Class object creation

const g: Gadget = new Gadget(1234, 'Mobile', 'SmartPhone');

// invoking getProduct method with the help of Gadget object

g.getProduct();

// invoking getProductId method with the help of Gadget object

g.getProductId();

// line to populate product name property with Gadget object

console.log('Product name is : ' + g.productName);

// line to populate static property product price directly with Product class name

console.log('Product price is : $' + Product.productPrice);