**Non-Primitive Data Type**

* These are Mutable types.
* The variable and its state can be changed after defining.
* The memory allocated for a variable can change dynamically.
* Non-Primitive types are allocated with memory heap.
* The range of value varies according to the memory available.
* They don’t have any fixed range for storing value.
* TypeScript non-primitive types are
  + Array
  + Object
  + Regular Expression

**Array Type**

* Arrays in computer programming are used to **reduce overhead** and **complexity**.
* Arrays can reduce overhead by storing values in sequential order.
* Array can reduce complexity by storing multiple values under one name.
* Array is a type of arrangement that stores in sequential order, but can be accessed random.
* Array is a collection of various types of values. Some technologies have issues with allocating different types of memory in sequential order. Hence, they restrict array to store same type values.
* Array size can be changed dynamically, which is not supported in various technologies.
* TypeScript Array can be used for both similar type of values and for various types of values.
* It can restrict array to handle similar type to satisfy certain situations.
* It can allow array to handle different types of values to satisfy other situations.
* TypeScript Array have the behaviour of Array as well as Collection [Stack, Queue, HashTable etc.]
* Array is a data structure.
* **John von Neumann – 1945**
* **1957 FORTRAN**
* **1960 COBOL, ALGOL 60**
* **1972 C**
* **1983 C++**
* Array is used to implement data structures such as
  + Lists
  + Heaps
  + Hash tables
  + Queues
  + Deques
  + Stacks
  + Strings
  + Vlists

**Declaring Array in TypeScript**

**let variableName:string[];**

Meta Characters

? zero or one occurance

+ one or more occurances

\* zero or more occurances

[] multiple , random, range

* After declaring Array, you have to **initialize or render** memory for array.
* Values can’t be stored into Array if memory is not **Initialized** or **Rendered**.

Ex:

let products:string[];

products[0] = "TV"; // invalid – [0] – is not defined

products[1] = "Mobile";

console.log(products.toString());

**Initialize memory for Array**

* You can render or initialize memory for Array by using
  + Meta Character “[ ]”
  + Array Constructor “Array()”

**Using Meta Character:**

Ex:

let products:string[] = []; // Initialization

products[0] = "TV";

products[1] = "Mobile";

console.log(products.toString());

Ex:

let products:string[];

products = []; //rendering

products[0] = "TV";

products[1] = "Mobile";

console.log(products.toString());

**Using Array Constructor:**

Ex:

let products:string[] = new Array();

products[0] = "TV";

products[1] = "Mobile";

console.log(products.toString());

Ex:

let products:string[];

products = Array();

products[0] = "Samsung TV";

products[1] = "Mobile";

console.log(products.toString());

* **new is** dynamic memory allocating operator.

**FAQ: What is difference between [ ] and Array()?**

1. Array() will not allow to **initialize** different types of values even when the data type is “any”. It can handle only the type of value which is first loaded into memory.

Array [ ] will allow to initialize or render.

**FAQ: How typescript array can store different types of values?**

1. By using “any” as data type.

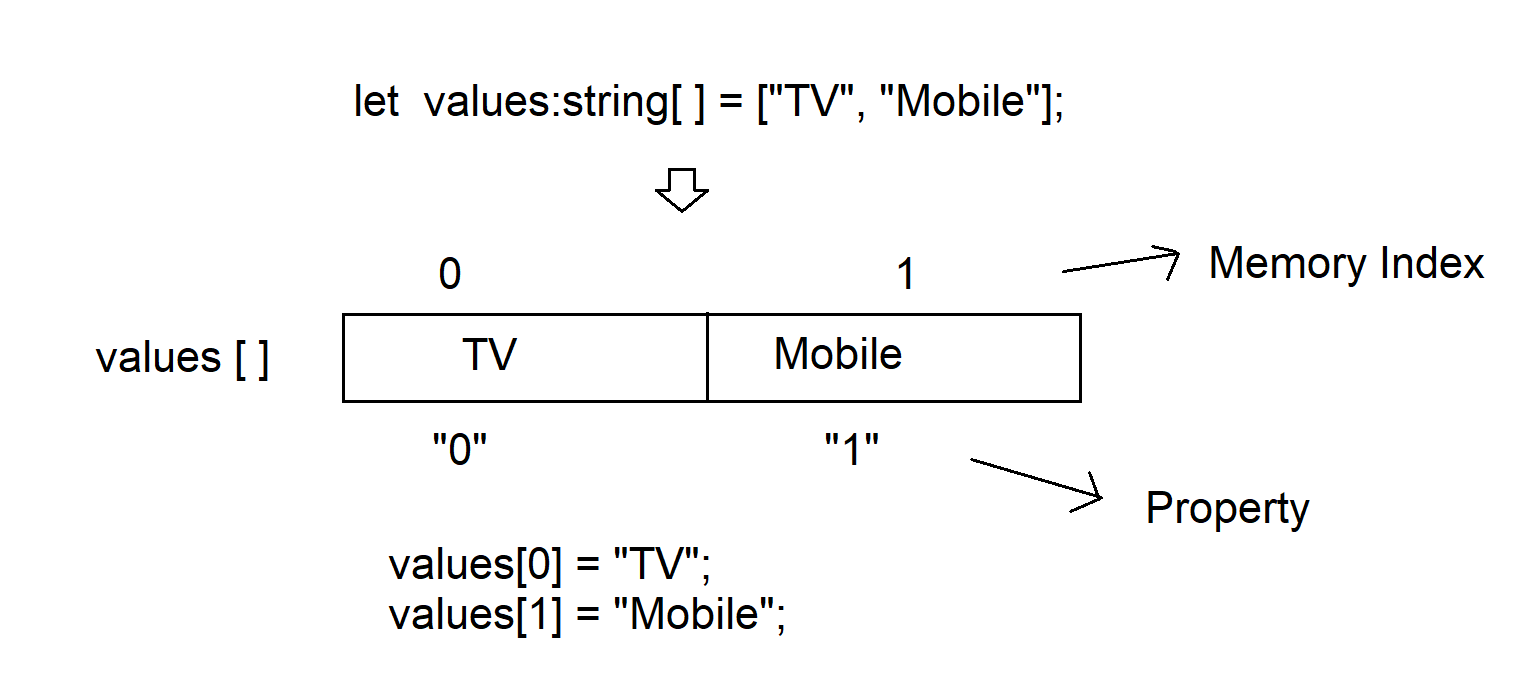
Ex:

let products:any[] = [10, "A", true];

**Note: If array is allowing different types of values then it is called as “Tuple”.**

**Storing Values into Array**

* You can initialize values or render values into memory allocated for Array.
* The memory allocated for array comprises of multiple blocks, which are accessed by using a **property name**.



* The property of array is “**string**” type which maps to same name **index**.

values[0] **0** is property and maps to **0** index.

Ex:

let values:string[] = ["TV", "Mobile"];

for(var property in values)

{

console.log(`${property} : [${typeof property}] - ${values[property]}`);

}

O/P:

0 : [string] - TV

1 : [string] - Mobile

Ex:

let values:string[] = [];

values["0"] = "Samsung TV";

values["1"] = "Nike Casuals";

console.log(values.toString());

**Manipulating Array Elements**

* Array object is derived from “Array” interface that provides a set of properties and methods which are used to manipulate Array.

**Reading Array Elements:**

|  |  |
| --- | --- |
| **Member / Technique** | **Description** |
| toString() | It reads all elements from an array and returns a string that concat elements with [,] delimiter. |
| join() | It reads all element from an array and returns a string that concat elements with custom delimiter.  Ex:  let products:string[] = ["TV", "Mobile", "Shoe"];  console.log(products.toString());  console.log(products.join("-->")); |
| Loop | * TypeScript can use “for, while, do while” loops for reading array elements. * Looping control defines initializer, condition and counter.   Ex:  let products:string[] = ["TV", "Mobile", "Shoe"];  for(var i=0; i<products.length; i++)  {  console.log(products[i]);  } |
| Iterator | * TypeScript can use “for-in, for-of” iterators * Iterator is a design pattern used to access elements from a collection in sequential order without condition, initializer and counter.   Ex: Iterate over values “**for-of”**  let products:string[] = ["TV", "Mobile", "Shoe"];  for(var item of products)  {  console.log(item);  }  Ex: Iterate over properties “**for-in”**  let products:string[] = ["TV", "Mobile", "Shoe"];  for(var property in products)  {  console.log(property);  }  Ex: Iterate over properties and values “**for-in”**  let products:string[] = ["TV", "Mobile", "Shoe"];  for(var property in products)  {  console.log(`${property} : ${products[property]}`);  } |
| find() | It can search for elements based on any specific condition and return only the first occurrence element that matches the condition.  Syntax:  arrayName.find(function(param){ return condition; });  Ex:  let sales:number[] = [20000, 42000, 13000, 45000, 52000, 12000];  let result:number = sales.find(function(val){  return val>40000;  });  console.log(result);  Ex:  let sales:number[] = [20000, 42000, 13000, 45000, 52000, 12000];  function getvalue(val){  return val>40000;  }  let result:number = sales.find(getvalue);  console.log(result); |
| filter() | It is similar to “find()” but can return all values that match the give condition.  Ex:  let sales:number[] = [20000, 42000, 13000, 45000, 52000, 12000];  function getvalue(sale){  return sale>40000;  }  let result:number[] = sales.filter(getvalue);  for(var item of result)  {  console.log(item);  } |
| slice() | It can return only the elements between specified index.  Ex:  let sales:number[] = [20000, 42000, 13000, 45000, 52000, 12000];  console.log(sales.slice(1,3));  console.log(slaes.slice(1)); 1 to end |

**Adding values into an Array**

|  |  |
| --- | --- |
| push() | It adds a value(s) into array as the last element.  Ex:  let products:string[] = ["TV","Mobile"];  function PrintValues()  {  for(var property in products) {  console.log(`[${property}] : ${products[property]} `);  }  }  products.push("Shoe","Watch");  PrintValues(); |
| unshift() | It adds a value(s) into array as first element.  Ex:  let products:string[] = ["TV","Mobile"];  function PrintValues()  {  for(var property in products) {  console.log(`[${property}] : ${products[property]} `);  }  }  products.unshift("Shoe","Watch");  PrintValues(); |
| splice() | It adds a value(s) into array at specified index location.  Syntax:  arrayName.splice(startIndex, deleteCount, “items..”)  Ex:  let products:string[] = ["TV","Mobile"];  function PrintValues()  {  for(var property in products) {  console.log(`[${property}] : ${products[property]} `);  }  }  products.splice(1,0,"Watch","Shoe");  PrintValues(); |

**Removing Array Elements**

|  |  |
| --- | --- |
| **Method** | **Description** |
| pop() | It removes and returns the last element.  Ex:  let products:string[] = ["TV","Mobile","Watch"];  function PrintValues()  {  for(var property in products) {  console.log(`[${property}] : ${products[property]} `);  }  }  PrintValues();  let removedElement:string = products.pop();  console.log(`${removedElement} Removed.`);  PrintValues(); |
| shift() | It removes and returns the first element.  Ex:  let products:string[] = ["TV","Mobile","Watch"];  function PrintValues()  {  for(var property in products) {  console.log(`[${property}] : ${products[property]} `);  }  }  PrintValues();  let removedElement:string = products.shift();  console.log(`${removedElement} Removed.`);  PrintValues(); |
| splice() | It removes specified element(s) based on index number and returns an array of removed elements.  Ex:  let products:string[] = ["TV","Mobile","Watch","Shoe"];  function PrintValues()  {  for(var property in products) {  console.log(`[${property}] : ${products[property]} `);  }  }  PrintValues();  let removedElement:string[] = products.splice(1,2);  console.log(`${removedElement} Removed.`);  PrintValues(); |

**Searching for Element in Array**

|  |  |
| --- | --- |
| **Method** | **Description** |
| find() | It can search for elements based on any specific condition and return only the first occurrence element that matches the condition. |
| filter() | It is similar to “find()” but can return all values that match the give condition. |
| indexOf() | It can search for your value in array and return its index number. It finds the first occurrence.  Ex:  let products:string[] = ["TV","Watch","Shoe"];  let searchString:string = "Mobile";  if(products.indexOf(searchString)==-1) {  console.log(`${searchString} Not Found`);  } else {  console.log(`${searchString} found at ${products.indexOf(searchString)}`);  } |
| lastIndex() | It finds and returns the last occurrence index number. The index functions return “-1” if not found. |
| includes() | It is a boolean method that determines whether an array contains a specified element.  **New in ES7.**  Syntax:  arrayName.includes(“searchValue”, startAtIndex)  Ex:  let products:string[] = ["TV","Watch","Mobile", "Shoe"];  let searchString:string = "Mobile";  if(products.includes(searchString)==true) {  console.log(`${searchString} found at ${products.indexOf(searchString)}`);  } else {  console.log(`${searchString} Not Found`);  } |

**New into Array**

|  |  |
| --- | --- |
| keys() | * It can iterate over keys in array and return all keys in the forms of iterator. * It returns an iterator object that contains collection of all keys. * TypeScript requires “Array.from()” method to read values from an iterator. * **ES6**   Ex:  let products:string[] = ["TV","Watch","Mobile", "Shoe"];  let result:any = products.keys();  for(var item of Array.from(result))  {  console.log(item);  } |
| entries() | It can iterate over key and value pair.  **ES6**  Ex:  let products:string[] = ["TV","Watch","Mobile", "Shoe"];  let result:any = products.entries();  for(var item of Array.from(result))  {  console.log(item);  } |
| every() | **ES5 – It is similar to find and filter.**  It checks for values that match given condition and return boolean result.  It is not an iterator.  It requires an explicit iterator to verify every value from a collection. |
| forEach() | **It is similar to entries.**  Ex:  let products:string[] = ["Mobile", "TV", "Shoe", "Mobile"];  function CheckValues(product, index){  console.log(`[${index}] : ${product}`);  }  products.forEach(CheckValues); |

**Array Destruction**

* It is a technique used to destruct the elements from array type memory and store them in individual references.

Ex:

let products:string[] = ["Nike Casuals", "Samsung TV"];

// Previous Technique

let product1 = products[0];

let product2 = products[1];

// Destruction

let [shoe, tv] = products;

console.log(`${shoe} \n ${tv}`);

Ex:

let math:any[] = [function(a,b){return a+b}, function(a,b){return a\*b}];

let [add, mul] = math;

console.log(`Addition= ${add(10,20)}`);

console.log(`Multiplication=${mul(2,5)}`);

**Object Type**