F# - MAPS

In F#, a map is a special kind of set that associates the values with key. A map is created in a similar way as sets are created.

Creating Maps

Maps are created by creating an empty map using Map.empty and adding items using the Add function. The following example demonstrates this −

Example

```
(* Create an empty Map *)
let students =
   Map.empty. (* Creating an empty Map *)
      Add("Zara Ali", "1501").
      Add("Rishita Gupta", "1502").
      Add("Robin Sahoo", "1503").
      Add("Gillian Megan", "1504");;
printfn "Map - students: %A" students

(* Convert a list to Map *)
let capitals =
   [ "Argentina", "Buenos Aires";
      "France ", "Paris";
      "Chili", "Santiago";
      "Malaysia", " Kuala Lumpur";
      "Switzerland", "Bern" ]
   |> Map.ofList;;
printfn "Map capitals : %A" capitals
```

When you compile and execute the program, it yields the following output −

```
Map - students: map
[("Gillian Megan", "1504"); ("Rishita Gupta", "1502"); ("Robin Sahoo", "1503
");
("Zara Ali", "1501")]
Map capitals : map
[("Argentina", "Buenos Aires"); ("Chili", "Santiago"); ("France ", "Paris");
("Malaysia", " Kuala Lumpur"); ("Switzerland", "Bern")]
```

You can access individual elements in the map using the key.

Example

```
(* Create an empty Map *)
let students =
   Map.empty. (* Creating an empty Map *)
      Add("Zara Ali", "1501").
      Add("Rishita Gupta", "1502").
      Add("Robin Sahoo", "1503").
      Add("Gillian Megan", "1504");;
printfn "Map - students: %A" students

(*Accessing an element using key *)
printfn "%A" students.["Zara Ali"]
```

When you compile and execute the program, it yields the following output −

```
Map - students: map
[("Gillian Megan", "1504"); ("Rishita Gupta", "1502"); ("Robin Sahoo", "1503
");
```

```
("Zara Ali", "1501")]
"1501"
```

Basic Operations on Maps

Add module name

The following table shows the basic operations on maps −

Member	Description
Add	Returns a new map with the binding added to the given map.
ContainsKey	Tests if an element is in the domain of the map.
Count	The number of bindings in the map.
IsEmpty	Returns true if there are no bindings in the map.
Item	Lookup an element in the map. Raises KeyNotFoundException if no binding exists in the map.
Remove	Removes an element from the domain of the map. No exception is raised if the element is not present.
TryFind	Lookup an element in the map, returning a **Some** value if the element is in the domain of the map and **None** if not.

The following example demonstrates the uses of some of the above functionalities −

Example

```
(* Create an empty Map *)
let students =
   Map.empty. (* Creating an empty Map *)
      Add("Zara Ali", "1501").
      Add("Rishita Gupta", "1502").
      Add("Robin Sahoo", "1503").
      Add("Gillian Megan", "1504").
      Add("Shraddha Dubey", "1505").
      Add("Novonil Sarker", "1506").
      Add("Joan Paul", "1507");;
printfn "Map - students: %A" students
printfn "Map - number of students: %d" students.Count

(* finding the registration number of a student*)
let found = students.TryFind "Rishita Gupta"
match found with
| Some x -> printfn "Found %s." x
| None -> printfn "Did not find the specified value."
```

When you compile and execute the program, it yields the following output −

```
Map - students: map
[("Gillian Megan", "1504"); ("Joan Paul", "1507"); ("Novonil Sarker", "1506"
);
("Rishita Gupta", "1502"); ("Robin Sahoo", "1503");
("Shraddha Dubey", "1505"); ("Zara Ali", "1501")]
Map - number of students: 7
Found 1502.
```