F# - MUTABLE DATA

 Copyright © tutorialspoint.com

Variables in F# are **immutable,** which means once a variable is bound to a value, it can't be changed. They are actually compiled as static read-only properties.

The following example demonstrates this.

Example

```
let x = 10
let y = 20
let z = x + y

printfn "x: %i" x
printfn "y: %i" y
printfn "z: %i" z

let x = 15
let y = 20
let z = x + y

printfn "x: %i" x
printfn "y: %i" y
printfn "z: %i" z
```

When you compile and execute the program, it shows the following error message −

```
Duplicate definition of value 'x'
Duplicate definition of value 'Y'
Duplicate definition of value 'Z'
```

Mutable Variables

At times you need to change the values stored in a variable. To specify that there could be a change in the value of a declared and assigned variable in later part of a program, F# provides the **mutable** keyword. You can declare and assign mutable variables using this keyword, whose values you will change.

The **mutable** keyword allows you to declare and assign values in a mutable variable.

You can assign some initial value to a mutable variable using the **let** keyword. However, to assign new subsequent value to it, you need to use the **<-** operator.

For example,

```
let mutable x = 10
x <- 15
```

The following example will clear the concept −

Example

```
let mutable x = 10
let y = 20
let mutable z = x + y

printfn "Original Values:"
printfn "x: %i" x
printfn "y: %i" y
printfn "z: %i" z

printfn "Let us change the value of x"
```

```
printfn "Value of z will change too."

x <- 15
z <- x + y

printfn "New Values:"
printfn "x: %i" x
printfn "y: %i" y
printfn "z: %i" z
```

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

```
Original Values:
x: 10
y: 20
z: 30
Let us change the value of x
Value of z will change too.
New Values:
x: 15
y: 20
z: 35
```

Uses of Mutable Data

Mutable data is often required and used in data processing, particularly with record data structure. The following example demonstrates this −

```
open System

type studentData =
   { ID : int;
      mutable IsRegistered : bool;
      mutable RegisteredText : string; }

let getStudent id =
   { ID = id;
      IsRegistered = false;
      RegisteredText = null; }

let registerStudents (students : studentData list) =
   students |> List.iter(fun st ->
      st.IsRegistered <- true
      st.RegisteredText <- sprintf "Registered %s" (DateTime.Now.ToString("hh:mm:ss"))

      Threading.Thread.Sleep(1000) (* Putting thread to sleep for 1 second to simulate
processing overhead. *))

let printData (students : studentData list) =
   students |> List.iter (fun x -> printfn "%A" x)

let main() =
   let students = List.init 3 getStudent

   printfn "Before Process:"
   printData students

   printfn "After process:"
   registerStudents students
   printData students

   Console.ReadKey(true) |> ignore

main()
```

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

```
Before Process:
{ID = 0;
IsRegistered = false;
RegisteredText = null;}
{ID = 1;
IsRegistered = false;
RegisteredText = null;}
{ID = 2;
IsRegistered = false;
RegisteredText = null;}
After process:
{ID = 0;
IsRegistered = true;
RegisteredText = "Registered 05:39:15";}
{ID = 1;
IsRegistered = true;
RegisteredText = "Registered 05:39:16";}
{ID = 2;
IsRegistered = true;
RegisteredText = "Registered 05:39:17";}
```