F# - DISCRIMINATED UNIONS

Unions, or discriminated unions allows you to build up complex data structures representing well-defined set of choices. For example, you need to build an implementation of a *choice* variable, which has two values yes and no. Using the Unions tool, you can design this.

Syntax

Discriminated unions are defined using the following syntax −

```
type type-name =
   | case-identifier1 [of [ fieldname1 : ] type1 [ * [ fieldname2 : ]
type2 ...]
   | case-identifier2 [of [fieldname3 : ]type3 [ * [ fieldname4 : ]type4 ...]
...
```

Our simple implementation of ,*choice,* will look like the following −

```
type choice =
   | Yes
   | No
```

The following example uses the type choice −

```
type choice =
   | Yes
   | No

let x = Yes (* creates an instance of choice *)
let y = No (* creates another instance of choice *)
let main() =
   printfn "x: %A" x
   printfn "y: %A" y
main()
```

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

```
x: Yes
y: No
```

Example 1

The following example shows the implementation of the voltage states that sets a bit on high or low −

```
type VoltageState =
   | High
   | Low

let toggleSwitch = function (* pattern matching input *)
   | High -> Low
   | Low -> High

let main() =
   let on = High
   let off = Low
   let change = toggleSwitch off

   printfn "Switch on state: %A" on
   printfn "Switch off state: %A" off
   printfn "Toggle off: %A" change
   printfn "Toggle the Changed state: %A" (toggleSwitch change)
```

```
main()
```

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

```
Switch on state: High
Switch off state: Low
Toggle off: High
Toggle the Changed state: Low
```

Example 2

```
type Shape =
   // here we store the radius of a circle
   | Circle of float

   // here we store the side length.
   | Square of float

   // here we store the height and width.
   | Rectangle of float * float

let pi = 3.141592654

let area myShape =
   match myShape with
   | Circle radius -> pi * radius * radius
   | Square s -> s * s
   | Rectangle (h, w) -> h * w

let radius = 12.0
let myCircle = Circle(radius)
printfn "Area of circle with radius %g: %g" radius (area myCircle)

let side = 15.0
let mySquare = Square(side)
printfn "Area of square that has side %g: %g" side (area mySquare)

let height, width = 5.0, 8.0
let myRectangle = Rectangle(height, width)
printfn "Area of rectangle with height %g and width %g is %g" height width (area
myRectangle)
```

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

```
Area of circle with radius 12: 452.389
Area of square that has side 15: 225
Area of rectangle with height 5 and width 8 is 40
```