

F# Tutorial

Pipe-Forward Operator

February 5, 2018

1 Syntax, variables, functions

1.1 Key concepts:

1. Having a good text editor helps you code much easier.
2. (a) Once defined, a variable in F# cannot change value (unless "mutable" is used)
(b) If you need an updated value, create a new one.
3. Different datatypes (e.g. integer and decimal-numbers) do not combine easily.
4. Defining and using functions in F# is slightly different from math notation/ other languages.
 - (a) F# automatically detects the type of the variables (e.g. integer, double, etc.) for a function.
 - (b) The variable types for a function will be enforced.

1.2 Introduction:

1.2.1 Comments

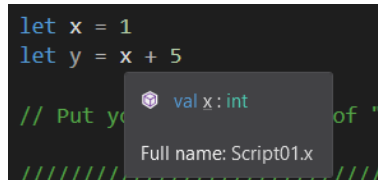
You can use double-slash `//`, triple-slash `///`, or star-bracket `(* *)` to make comments.

```
// These words are ignored.  
/// These words are ignored.  
(* These words are ignored. *)  
let x = 1  
let y = x + 5
```

1.2.2 Intellisense

If you are using Visual Studio or Visual Studio Code, you can put your mouse on top of the variable name `x` or `y`, and see that it is an `int` or integer.

This feature will help you identify what is each variable/function, and make coding easier for you.



1.2.3 Common data types and printing

Some of the common types in F# are:

Keyword	Description	Print in output:
<code>int</code>	Integer	<code>%i</code>
<code>double</code> or <code>float</code>	Decimal numbers	<code>%f</code>
<code>string</code>	Words/Sentences	<code>%s</code>
<code>bool</code>	True/False	<code>%b</code>
<code>-</code>	Other objects	<code>%A</code> or <code>%O</code>

```
let name = "John"
let age = 21
let height = 170.5

printfn "My name is: %s" name
// Output:
// My name is: John

printfn "Name: %s. Age: %i. Height: %f." name age height
// Output:
// Name: John. Age: 21. Height: 170.500000

printfn "His height is: %.2f" height
// Output:
// His height is: 170.50
///// Show only two decimal.
```

For example, in the second example, inside the string-format, there are `%s`, `%i`, `%f`. And so, we expect a string, integer, and decimal (in that order) after the string-format specification in order to completely print the result to the output console.

1.2.4 Equality and simple if-else

The `let ... = ...` combination is used to assigned a value to a variable. Other than this situation, the equal sign `=` is used for equality testing. `=`, `<>` are used for equality/inequality testing.

```
let valueToTest = 20
let isValueEqualToTwenty = (valueToTest = 20)

if isValueEqualToTwenty then
    printfn "Yes, the value is Twenty"
else
    printfn "No, the value is not Twenty"
// Output: "Yes, the value is Twenty"
////////////////////////////////////

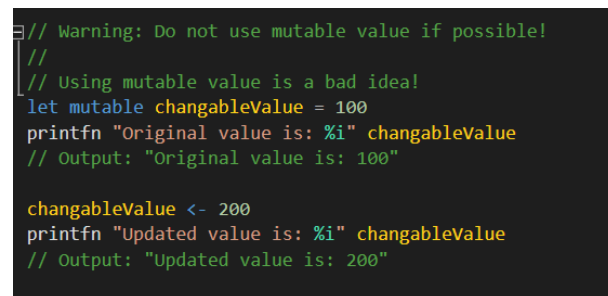
let inputUserName = "Jack"

if inputUserName = "John" then
    printfn "Welcome back, John"
else
    printfn "Access denied."
// Output: "Access denied."
```

In Java/C++, `==`, `!=` are used for comparison, and in Javascript, `===`, `!==` are used.

1.2.5 Immutability

In F#, variables are by default immutable/unchangable. Once defined, the value of a variable cannot be changed. You can make a variable changable/mutable using the keyword `mutable` and symbol `<-`, but this is highly discouraged. (If you use VisualStudio, then the color of the variable name will change color, warning you of potential mutable values)



```
// Warning: Do not use mutable value if possible!
//
// Using mutable value is a bad idea!
let mutable changableValue = 100
printfn "Original value is: %i" changableValue
// Output: "Original value is: 100"

changableValue <- 200
printfn "Updated value is: %i" changableValue
// Output: "Updated value is: 200"
```

In addition, if you try to update an immutable/unchangable value using `<-`, you will get an error.

```
// Uncomment the code below to see an error:

let immutableValue = 100
immutableValue <- 300
```

This value is not mutable. Consider using the mutable keyword, e.g. 'let mutable immutableValue = expression'.

Why do we recommend immutable/unchangeable values:

Imagine you have a code below, where you have defined a mutable value `x`, and after thousands of lines of code later, you used `x`'s value again:

```
let mutable x = 100
//
// Thousands of lines of code later.....
// You have many lines of code in between.....
// It is hard to keep track.....
// Have you changed/updated x's value?
// Did you accidentally call any function that modify x?
// Can you guarantee x's value stay unchanged?
//
//
let y = x + 1
// What is the value of y?
//
// That depends on what happens between y's definition
// and x's definition.
```

On the other hand, if `x` is immutable/unchangeable:

```
let x = 100
//
// Thousands of lines of code later.....
// You have many lines of code in between.....
// But because x is immutable/unchangeable.....
// We can be sure that x stays constant.....
// And we can safely conclude that.....
//
let y = x + 1
// y = 101
```

1.2.6 (+) Operator on the same type of variable

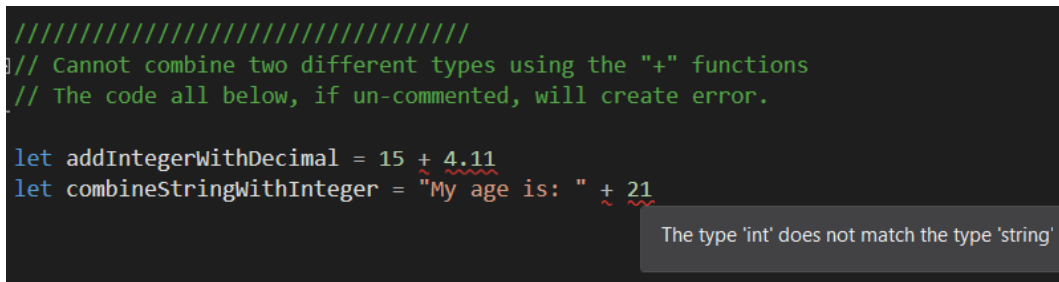
Integers, double, and string support the (+) operation:

```
let number1 = 40
let number2 = 55
let addTwoNumbers = number1 + number2

// Remark: "float" and "double" mean the same thing in F#.
let sqrtTwoApprox = 1.414
let piApprox = 3.1415926
let addTwoDecimals = sqrtTwoApprox + piApprox

let sentenceStart = "My school is "
let schoolName = "National University of Singapore"
let combinedSentence = sentenceStart + schoolName
```

However, you cannot add an integer with a decimal in F# directly using (+), and you cannot add/concatenate a string with a number directly using (+). If you use VisualStudio, then you may see an error similar to the one below.



```
////////////////////////////////////
// Cannot combine two different types using the "+" functions
// The code all below, if un-commented, will create error.

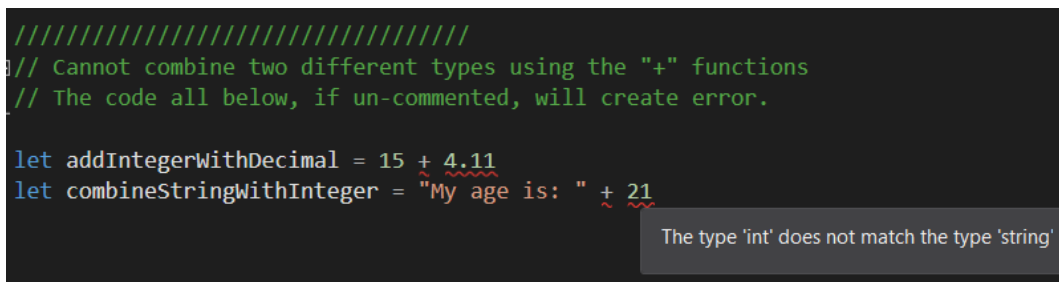
let addIntegerWithDecimal = 15 + 4.11
let combineStringWithInteger = "My age is: " + 21
```

The type 'int' does not match the type 'string'

Furthermore, some functions, like the square root `sqrt` and math exponent (`**`) only accepts decimal numbers:

```
let sqrtRootOfNine = sqrt 9.0
let twoToPowerOfFive = 2.0 ** 5.0
```

And it will cause error if you use them with integer input instead.



```
////////////////////////////////////
// Cannot combine two different types using the "+" functions
// The code all below, if un-commented, will create error.

let addIntegerWithDecimal = 15 + 4.11
let combineStringWithInteger = "My age is: " + 21
```

The type 'int' does not match the type 'string'

1.2.7 Functions