

F# Arithmetic Samples

Background

In this example, we demonstrate basic F# Arithmetic Samples in Light Syntax.

Integer Arithmetic

The example below demonstrates integer arithmetic:

```
#light
let SampleArithmetic1() =
    let x = 10 + 12 - 3
    let y = x * 2 + 1
    let r1,r2 = x/3, x%3
    printfn "x = %d, y = %d, r1 = %d, r2 = %d" x y r1 r2
```

x = 19, y = 39, r1 = 6, r2 = 1

Floating Point Arithmetic

The example below demonstrates floating point arithmetic:

```
#light
let SampleArithmetic2() =
    let x = 10.0 + 12.0 - 3.0
    let y = x * 2.0 + 1.0
    let r1 = x/3.0
    printfn "x = %g, y = %g, r1 = %g" x y r1
```

x = 19, y = 39, r1 = 6.33333

Numeric Type Conversion

The example below demonstrates how to convert between various numeric types:

```
#light
let SampleArithmetic3() =
    // Manipulating double-precision (64-bit) floating point numbers
    let pi1 = float 3 + 0.1415 // 'float' is an overloaded conversion operator
    let pi2 = double 3 + 0.1415 // identical - 'double' is a synonym for 'float'
    printfn "pi1 = %f, pi2 = %f" pi1 pi2;

    let i1 = int 3.1415
    let i2 = int64 3.1415
    let i3 = truncate 3.1415 // identical
    printfn "i1 = %d, i2 = %d, i3 = %d" i1 i2 i3;

    // Manipulating single-precision (32-bit) floating point numbers
    let f32a = 2.1415f + 1.0f // float32 (System.Single)
    let f32b = 2.1415f + float32 1 // float32 - identical
    printfn "f32a = %f, f32b = %G" (Float32.to_float f32a) (Float32.to_float f32b)
```

```
// Manipulating bytes
let byteA = byte (3+4)           // byte
let byteB = 255uy                // byte
let byteC = 0xFFuy              // byte
let byteD = byte 0xFF           // byte
printfn "byteA = %d, byteB = %d" (Byte.to_int byteA) (Byte.to_int byteB)
```

```
pi1 = 3.141500, pi2 = 3.141500
i1 = 3, i2 = 3, i3 = 3
f32a = 3.141500, f32b = 3.1415
byteA = 7, byteB = 255
```

Bitwise Integer Operations

The example below demonstrates bitwise integer operations:

```
#light
let Sample4() =
    // Operators over integers:
    let x1 = 0xAB7F3456 &&& 0xFFFF0000
    let x2 = 0xAB7F3456 ||| 0xFFFF0000
    let x3 = 0x12343456 ^^^ 0xFFFF0000
    let x4 = 0x1234ABCD <<< 1
    let x5 = 0x1234ABCD >>> 16

    // Also over other integral types, e.g. Int64:
    let x6 = 0x0A0A0A0A012343456L &&& 0x00000000FFFF0000L

    // Also over other integral types, e.g. unsigned Int64:
    let x6u = 0x0A0A0A0A012343456UL &&& 0x0000FFFF00000000UL

    // And bytes:
    let x7 = 0x13uy &&& 0x11uy

    // Now print the results:
    printfn "x1 = 0x%08x" x1;
    printfn "x2 = 0x%08x" x2;
    printfn "x3 = 0x%08x" x3;
    printfn "x4 = 0x%08x" x4;
    printfn "x5 = 0x%08x" x5;
    printfn "x6 = 0x%016x" x6;
    printfn "x6u = 0x%016x" x6u;
    printfn "x7 = 0x%02x" (Byte.to_int x7)
```

```
x1 = 0xab7f0000
x2 = 0xffff3456
x3 = 0xedcb3456
x4 = 0x2469579a
x5 = 0x00001234
x6 = 0x0000000012340000
x6u = 0x0000a0a000000000
x7 = 0x11
```

This concludes our sample.