

**FUNCTIONAL**

**PROGRAMMING**

**WHAT ARE  
FUNCTIONS?**

- ▶ **A relation from a set to a set**

$\{(true, false), (false, true), (false, false)\}$

- ▶ **One output object for every input**

$\{(true, false), (false, true)\}$

# ACTUAL FUNCTIONS

```
boolean not(boolean b) {  
    return !b;  
}
```

```
String hello(String s) {  
    return "Hello " + s;  
}
```

**NOT**  
**FUNCTIONS**

```
System.out.println("Hello world");
```

```
System.currentTimeMillis();
```

```
x = x + 1;
```

```
boolean update(int j) {  
    i += j;  
    return j + 1;  
}
```

... functional programming is a restriction on how we write programs, but not on what programs we can express.

– **Rúnar Bjarnason and Paul Chiusano, Functional Programming in Scala, 2014**



# EQUATIONAL REASONING

```
main :: IO ()
main = print $ sum [1, 2, 3]
```

```
sum :: [Int] -> Int
sum = foldl (+) 0
```

```
foldl :: (Int -> Int -> Int) -> Int -> [Int] -> Int
foldl f z xs = go z xs
  where go z [] = z
        go z (x:xs) = go (f z x) xs
```

```
main :: IO ()  
main = print $ sum [1, 2, 3]
```

```
sum :: [Int] -> Int
```

```
sum xs = go 0 xs
```

```
  where go z [] = z
```

```
        go z (x:xs) = go (z + x) xs
```

```
main :: IO ()
```

```
main = print sum'
```

```
sum' :: Int
```

```
sum' = go 0 [1, 2, 3]
```

```
  where go z [] = z
```

```
        go z (x:xs) = go (z + x) xs
```

```
main :: IO ()
```

```
main = print sum'
```

```
sum' :: Int
```

```
sum' = go (((0 + 1) + 2) + 3) []
```

```
  where go z [] = z
```

```
main :: IO ()  
main = print sum'
```

```
sum' :: Int
```

```
sum' = (((0 + 1) + 2) + 3)
```

```
main :: IO ()  
main = print sum'
```

```
sum' :: Int
```

```
sum' = 6
```

```
main :: IO ()
```

```
main = print 6
```



**NOT**  
**EQUATIONAL**  
**REASONING**

```
val x = { println("Hello"); 1 }
```

```
x + x
```

```
// Hello
```

```
// 2
```

```
{ println("Hello"); 1 } +  
  { println("Hello"); 1 }
```

```
// Hello
```

```
// Hello
```

```
// 2
```

# PRINCIPLE OF COMPOSITIONALITY

**Meaning of a complex statement is the combination of  
the meanings of its parts.**

**COMPROMISING**

Can we write **partially functional** programs?

**TOOLS**



▶ **Haskell**

▶ **ML**

▶ **Scala**

▶ **F#**

**GO WRITE FUNCTIONS!**