Teaching F#

From numerical expressions to 3D graphics

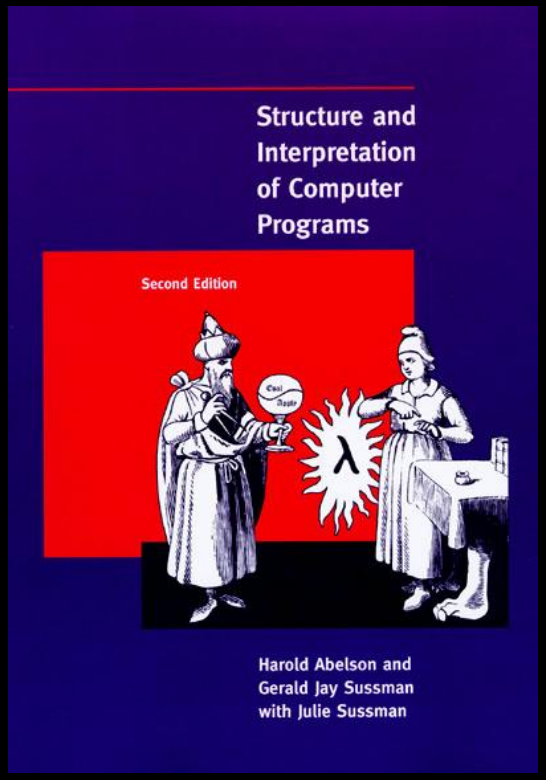
Tomáš Petříček

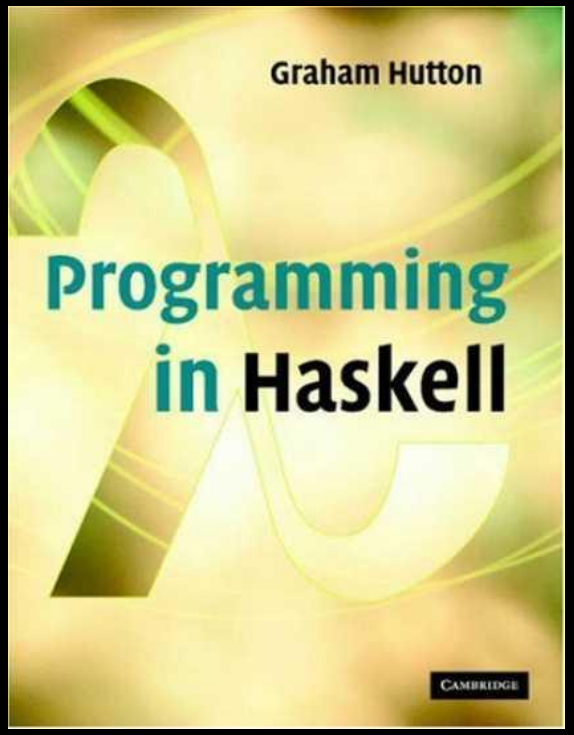
F# enthusiast and F# book author
PhD student at University of Cambridge

Target audience

- Freshman students
  - Basic knowledge of math principles (Not too much – they may not like it)
  - Earlier in high-school or later at university should be fine too
- Possibly first programming course
  - Controversial topic, but F# has many benefits

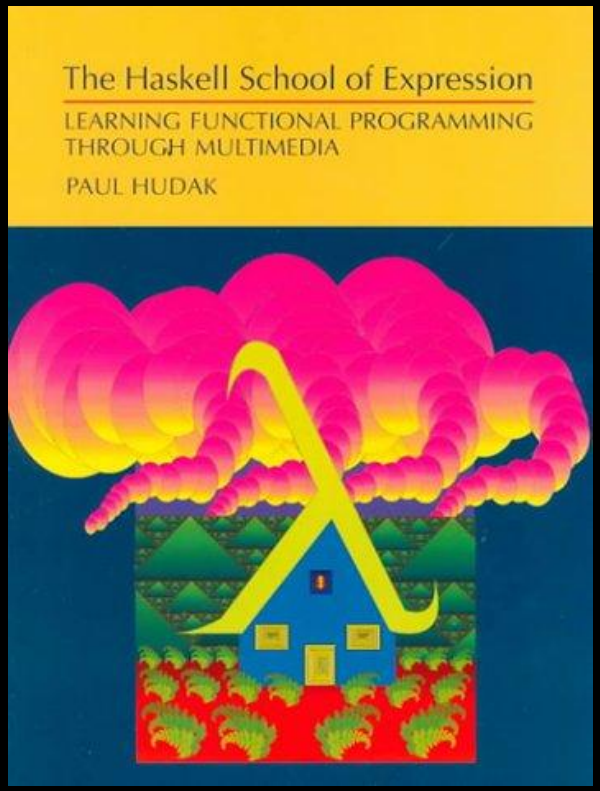
Related functional textbooks #1

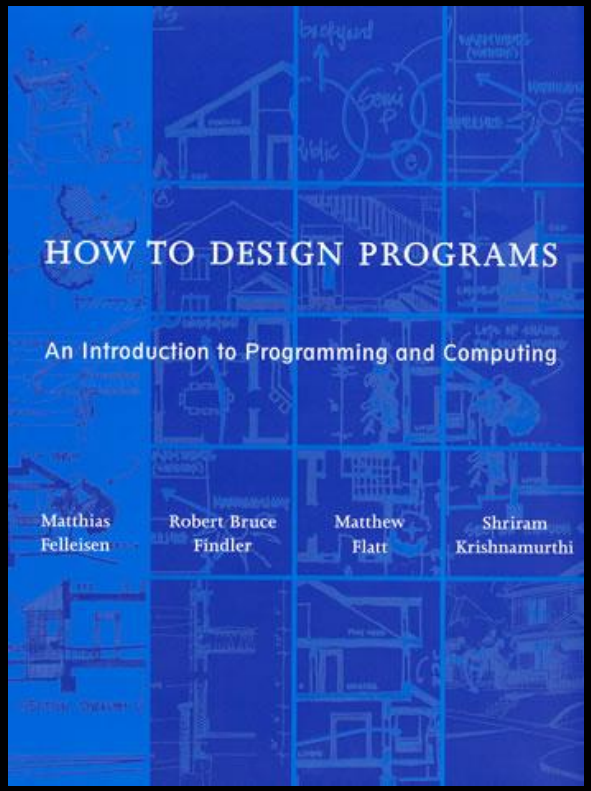




- Emphasize theory and abstract thinking

Related functional textbooks #2





- Functional programming with fun demos

Why F# as a first language?

- Mathematically oriented language
  - Supplements math and theory courses
  - Eliminates differences between students
- Open-source with cross-platform support
  - F# for MonoDevelop on Linux/Mac
- Practical language with some market
  - .NET/Mono skills are valued by industry
  - Very easy transition to C# and other languages

First functional programming steps

Expressions in F#

What F# shares with math?

- Language is exact – we need to precisely say what we want to get
- Composition does not break things – we can mix various correct facts
- Things do not change – theorems will always be true, π will not change

Pythagorean theorem

- Math has equations and expressions

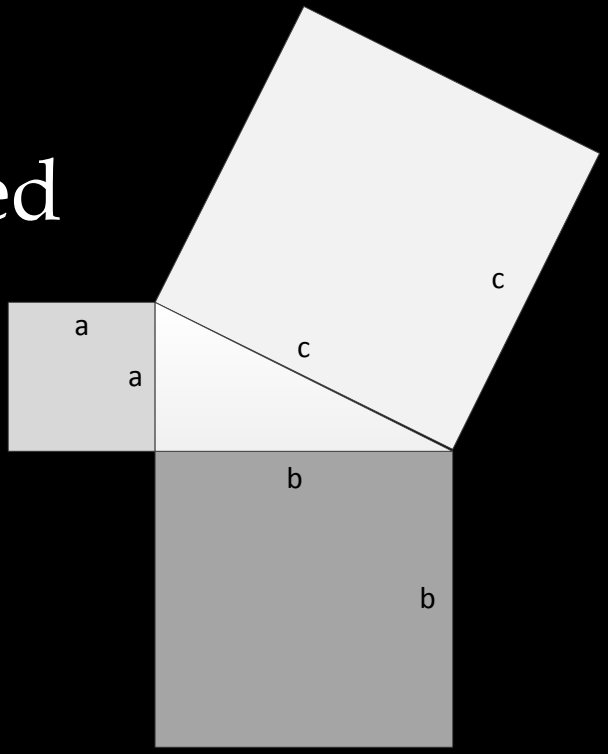
$$c^2 = a^2 + b^2 \qquad \sqrt{a^2 + b^2}$$

- Expressions can be evaluated

$$\sqrt{3^2 + 4^2} = \sqrt{9 + 16} = \sqrt{25} = 5$$

- Using F# Interactive

```
> sqrt((pown 3.0 2) + (pown 4.0 2));;
val it : float = 5.0
```



*Introducing **let** declarations and **if***

- Solving quadratic equations

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \qquad \frac{-b \pm \sqrt{D}}{2a} \quad \text{where} \quad D = b^2 - 4ac$$

- Script for solving written in F#

```
let d = pown b 2 - 4.0 * a * c;;

if d < 0.0 then "No real solution"
elif d > 0.0 then "Two solutions"
else "One solution";;

let x1 = (-b + sqrt d) / 2.0 * a
let x2 = (-b - sqrt d) / 2.0 * a;;
```

Working with F# expressions

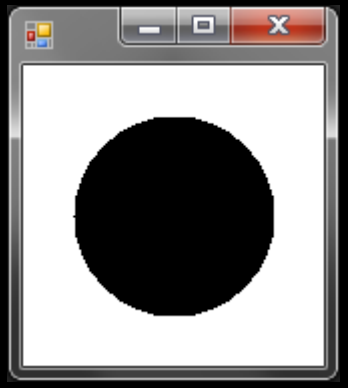
DEMO

Fun examples for students to play with

Composing graphics in F#

Composing graphics

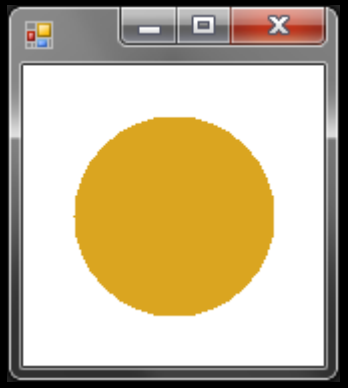
- Expressions that have *drawing* as the result
  - Same concepts as mathematical expressions



- Basic shapes

```
> Fun.circle 100.0f;;
val it : Drawing = (Drawing 100x100)
```

- Functions for creating new shapes

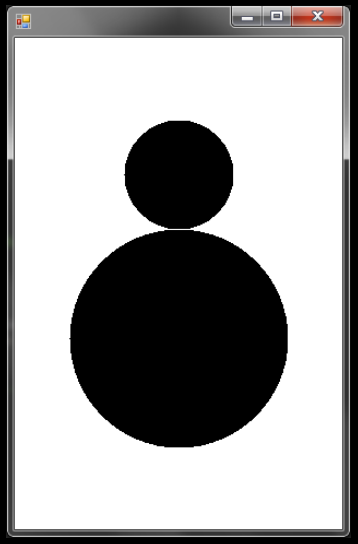


```
> Fun.fillColor Color.Goldenrod
    (Fun.circle 100.0f);;
val it : Drawing = (Drawing 100x100)
```

Working with shapes

- Composing drawings using custom operator ($)

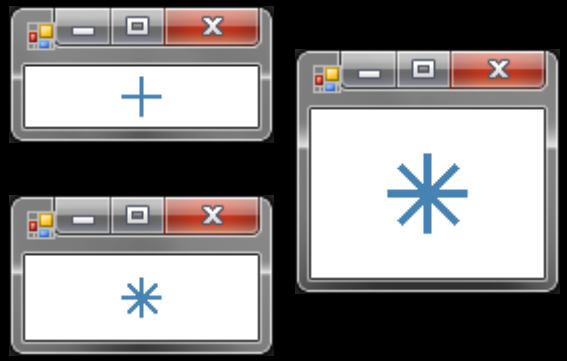
```
Fun.circle 200.0f $
  Fun.move 0.0f 150.0f (Fun.circle 100.0f)
```



- Let declarations to reuse drawings

```
let plus =
  Fun.lineStyle 2.0f Color.SteelBlue
    ( Fun.line -10.0f 0.0f 10.0f 0.0f $
      Fun.line  0.0f -10.0f 0.0f 10.0f )

let star = plus $ (Fun.rotate 45.0f plus)
let bigStar = Fun.scale 2.0f 2.0f star
```



Creating drawings using F# expressions

DEMO

Difficult concepts explained using graphics

Introducing recursion and 3D

Explaining recursion

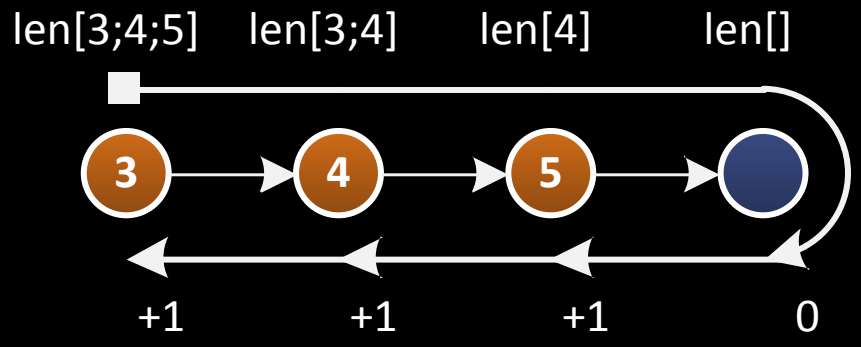
- Recursion is tricky concept
- Two types of recursions (from HTDP)
  - Structural – follows recursive data structure
    For example lists, trees, expressions etc.
    Such functions must terminate in F#
  - General – arbitrary recursive calls
    May not terminate (if it is not well written)
    For example generating fractals

General recursion for lists

- Not tail-recursive list length

```
let rec length list =
    match list with
    | [] -> 0
    | head::tail -> (length tail) + 1
```

- Easy visualization of execution



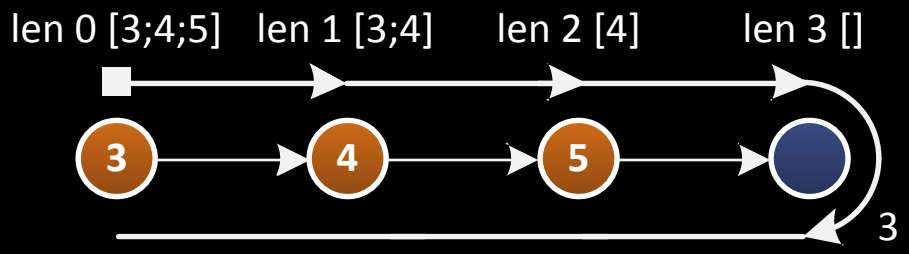
  - Not tail-recursive – work done on the "way back"

General recursion for lists

- Tail-recursive list length

```
let rec length acc list =
    match list with
    | head::tail ->
        let newacc = acc + 1 in length newacc tail
    | [] -> acc;;
```

- All work done on the way "forward"
  - This is what *tail-recursive* means!



Generating drawings from lists

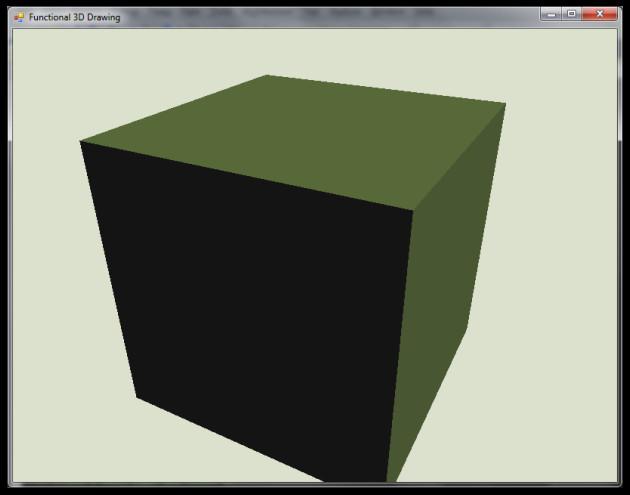
DEMO

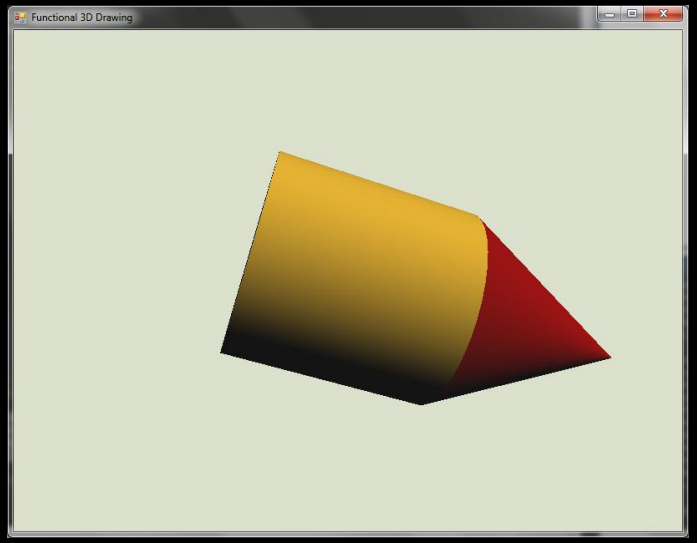
Adding new dimension to drawings

- Expressions that define 3D objects
- Just like math and 2D graphics

```
> Fun.cube;;
val it : Drawing3D = (...)
```

```
Fun.color Color.DarkRed Fun.cone $
( Fun.color Color.Goldenrod
    (Fun.translate (0.0, 0.0, 1.0)
       Fun.cylinder) );;
val it : Drawing3D = (...)
```





Introducing general recursion using 3D objects

DEMO

Links and Q & A

- Functional variations Web Site

  Cross-platform F# support
  Teaching materials for F#, etc…

  - http://functional-variations.net

- If you're interested in more, get in touch!
  - http://tomasp.net | tomas@tomasp.net
  - http://twitter.com/tomaspetricek