Informatics 1

Functional Programming Lecture 2

Tuesday 29 September 2009

# The Rule of Leibniz

Philip Wadler

University of Edinburgh

Rosh Hashanah

Saturday 19 September 2009 / 1 Tishri 5770

May you be inscribed in the book of life for a sweet year!

Yom Kippur

Monday 28 September 2009 / 10 Tishri 5770

# Required text and reading

*Haskell: The Craft of Functional Programming*, Second Edition, Simon Thompson, Addison-Wesley, 1999.

Reading assignment:

Thompson, Chapters 1–3 (pp. 1–52) by Friday 25 September 2009.

Thompson, Chapters 4–5 (pp. 53–95) by Monday 5 October 2009.

Thompson, Chapters 6–7 (pp. 96–134) by Monday 12 October 2009.

# Labs and Lab week

## Drop-in laboratories

Computer Lab West, Appleton Tower, level 5

| | |
|---|---|
| Mondays | 3–5pm |
| Tuesdays | 2–5pm |
| Wednesdays | 2–5pm |
| Thursdays | 2–5pm |
| Fridays | 3–5pm |

Lab Week Exercise due *5pm Friday 2 October*.

# Tutorials

ITO will assign you to tutorials, starting next week

| | |
|---|---|
| Tuesday/Wednesday | Computation and Logic |
| Thursday/Friday | Functional Programming |

Do the tutorial work *before* the tutorial!

(You do not do the tutorial work *during* the tutorial!)

Bring a *printout* of your work to the tutorial!

# Part I

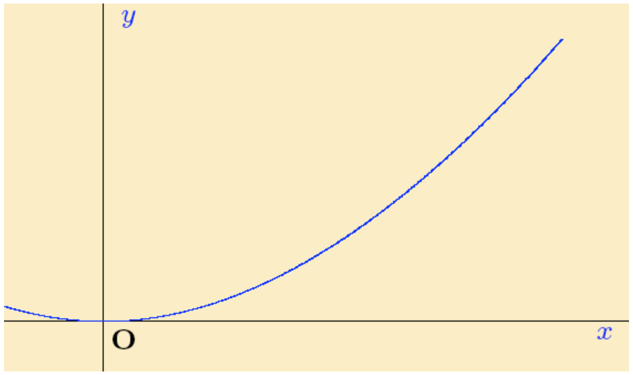
# The Rule of Leibniz

# What is a function?

- A recipe for generating an output from inputs:
  "Multiply a number by itself"
- A set of (input, output) pairs:
  (1,1) (2,4) (3,9) (4,16) (5,25) $\cdots$
- An equation:

$$f\ x = x^2$$

- A graph relating inputs to output (for numbers only):



# Operations on numbers

```
[culross]wadler: ghci
   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /  | |      GHC Interactive, version 6.7
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base ... linking ... done.
Prelude> 3+3
6
Prelude> 3*3
9
Prelude>
```

# Functions over numbers

lect02.hs

```
square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth a b =  square a + square b
```

# Testing our functions

```
[culross]wadler: ghci lect02.hs
   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /  | |      GHC Interactive, version 6.7
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base ... linking ... done.
[1 of 1] Compiling Main              ( lect02.hs, interpreted )
Ok, modules loaded: Main.
*Main> square 3
9
*Main> pyth 3 4
25
*Main>
```

# A few more tests

```
*Main> square 0
0
*Main> square 1
1
*Main> square 2
4
*Main> square 3
9
*Main> square 4
16
*Main> square (-3)
9
*Main> square 10000000000
100000000000000000000
```

# Declaration and evaluation

## Declaration (file lect02a.hs)

```
square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth a b  =  square a + square b
```

## Evaluation

```
% ghci lect02a.hs
   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /  | |      GHC Interactive, version 6.7
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base-1.0 ... linking ... done.
Compiling Main                 ( lect2.hs, interpreted )
Ok, modules loaded: Main.
*Main> pyth 3 4
25
*Main>
```

# The Rule of Leibniz

```
    square :: Integer -> Integer
    square x  =  x * x

    pyth :: Integer -> Integer -> Integer
    pyth a b  =  square a + square b
```

“Equals may be substituted for equals.”

— Gottfried Wilhelm Leibniz (1646–1716)

```
    pyth 3 4
  =
    square 3 + square 4
  =
    3*3 + 4*4
  =
    9 + 16
  =
    25
```

# Numerical operations are functions

```
   (+) :: Integer -> Integer -> Integer
   (*) :: Integer -> Integer -> Integer
```

```
Main*> 3+4
7
Main*> 3*4
12
```

```
   3 + 4   stands for   (+) 3 4
   3 * 4   stands for   (*) 3 4
```

```
Main*> (+) 3 4
7
Main*> (*) 3 4
12
```

# Precedence and parentheses

Function application takes *precedence* over infix operators.
(Function applications *binds more tightly than* infix operators.)

```
    square 3 + square 4
  =
    (square 3) + (square 4)
```

Multiplication takes *precedence* over addition.
(Multiplication *binds more tightly than* addition.)

```
    3*3 + 4*4
  =
    (3*3) + (4*4)
```

# Associativity

Addition is *associative*.

```
     3 + (4 + 5)
  =
     3 + 9
  =
     12
  =
     7 + 5
  =
     (3 + 4) + 5
```

Addition *associates to the left*.

```
     3 + 4 + 5
```

*stands for*

```
     (3 + 4) + 5
```

# Part II

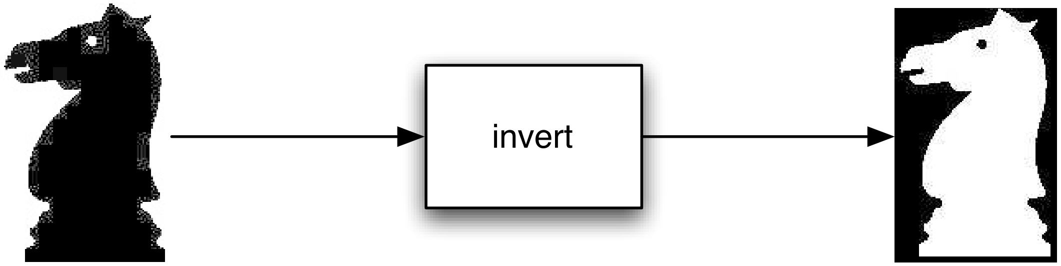
# Chess

# Kinds of data

- Integers: `42`, `-69`
- Floats: `3.14`
- Characters: `'h'`
- Strings: `"hello"`
- Pictures:

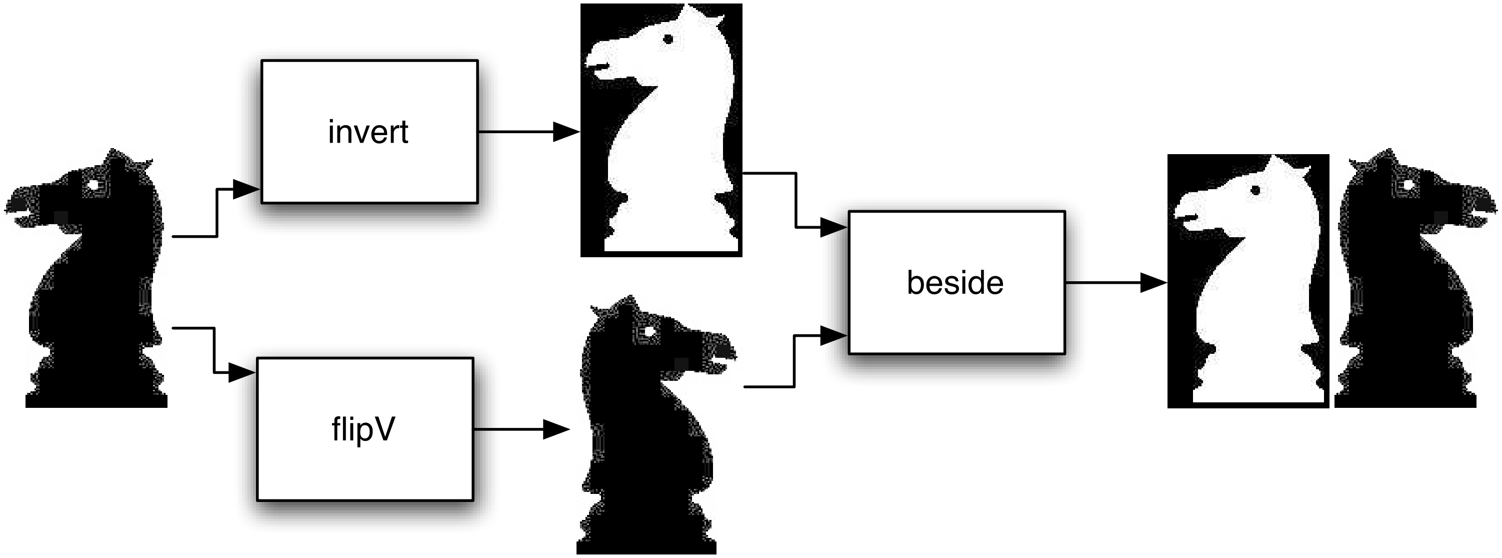
# Applying a function

```
invert :: Picture -> Picture
knight :: Picture

invert knight
```



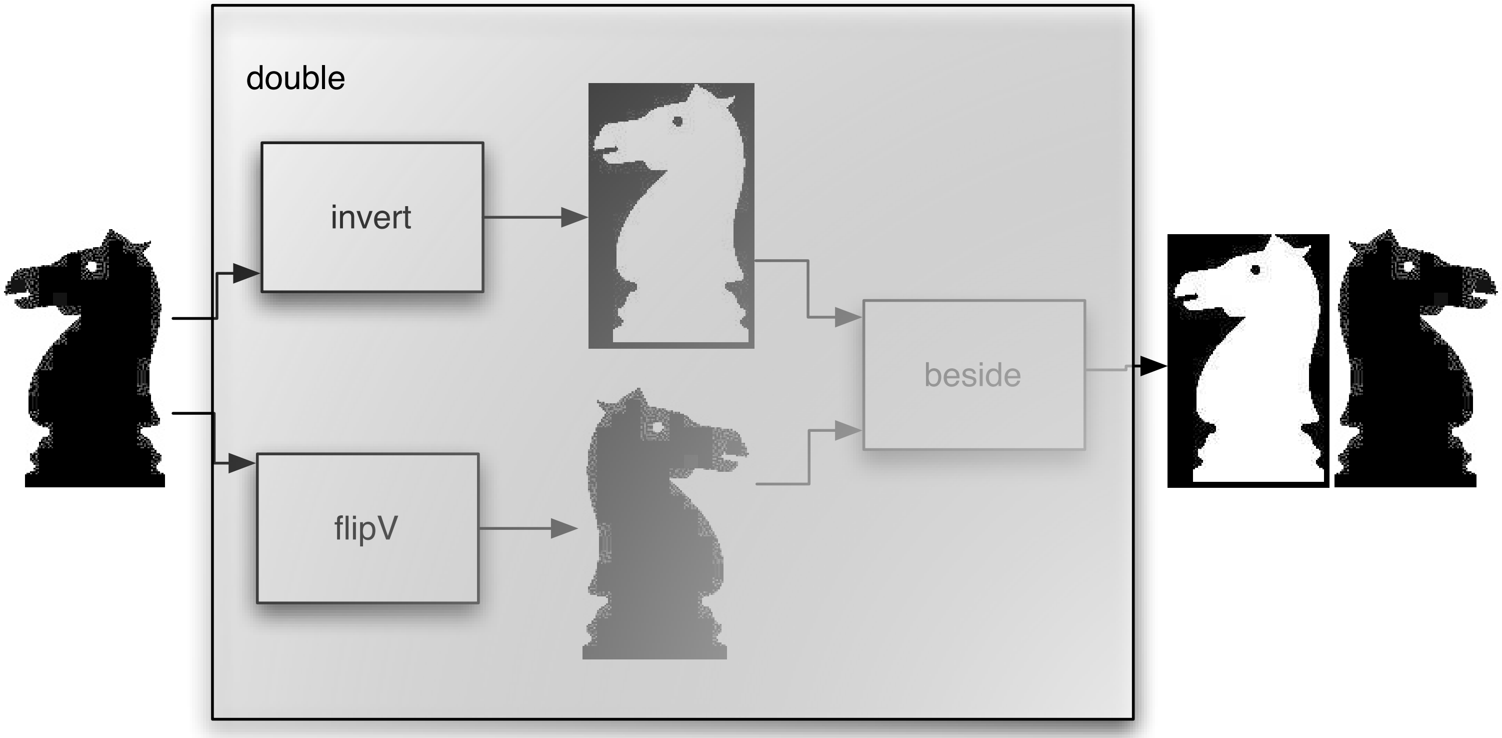
# Composing functions

```
beside :: Picture -> Picture -> Picture
flipV :: Picture -> Picture
invert :: Picture -> Picture
knight :: Picture

beside (invert knight) (flipV knight)
```



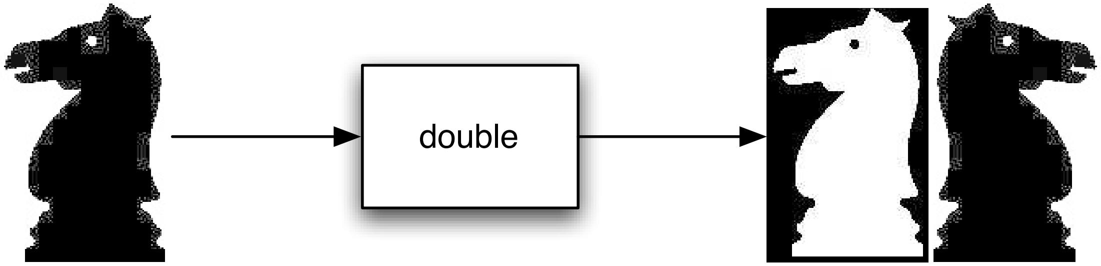
# Defining a new function

```
double :: Picture -> Picture
double p  =  beside (invert p) (flipV p)

double knight
```



# Defining a new function

```
double :: Picture -> Picture
double p  =  beside (invert p) (flipV p)

double knight
```



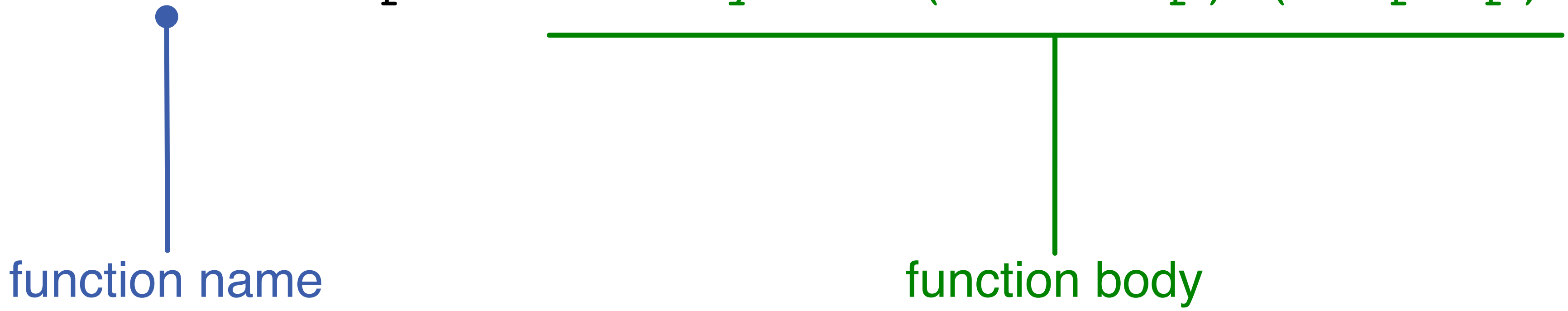
# Terminology

## Type signature

```
makePicture :: Picture -> Picture
```

## Function declaration

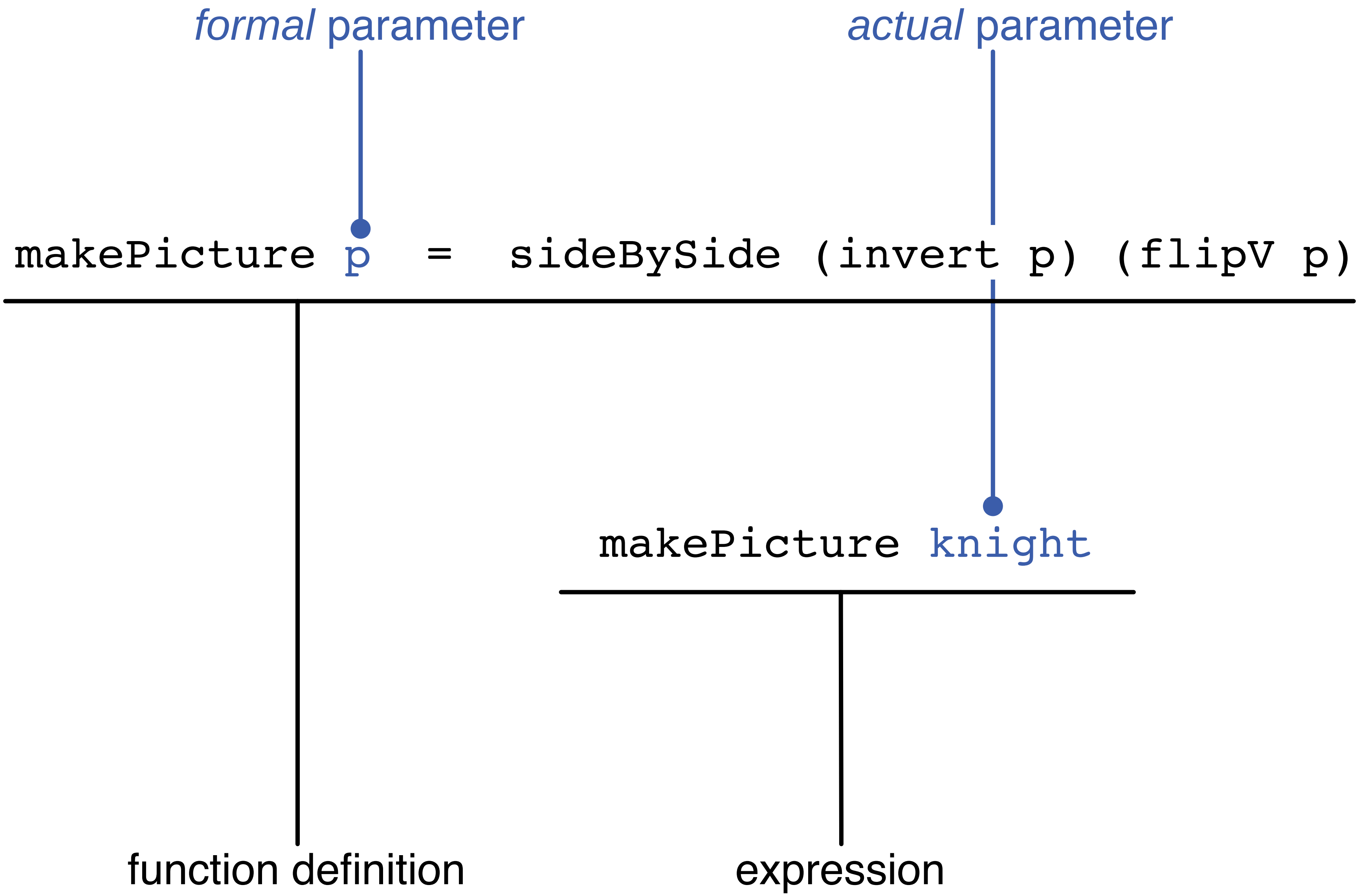
```
makePicture p  =  sideBySide (invert p) (flipV p)
```



function name

function body

# Terminology



*formal* parameter

*actual* parameter

```
makePicture p  =  sideBySide (invert p) (flipV p)
```

```
makePicture knight
```

function definition

expression

# Part III

# QuickCheck

# A program (file `lect02a.hs`)

```
square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth a b =  square a + square b
```

# Running a program

```
[culross]wadler: ghci lect02a.hs
GHCi, version 6.8.3: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
[1 of 1] Compiling Main          ( lect02a.hs, interpreted )
Ok, modules loaded: Main.
*Main> square 3
9
*Main> pyth 3 4
25
*Main>
```

# Another program (file `lect02b.hs`)

```haskell
import Test.QuickCheck

square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth a b =  square a + square b

prop_square :: Integer -> Bool
prop_square x  =
  square x >= 0

prop_squares :: Integer -> Integer -> Bool
prop_squares x y  =
  square (x+y) == square x + 2*x*y + square y

prop_pyth :: Integer -> Integer -> Bool
prop_pyth x y  =
  square (x+y) == pyth x y + 2*x*y
```

# Running another program

```
[culross]wadler: ghci lect02b.hs
GHCi, version 6.8.3: http://www.haskell.org/ghc/  :? for help
Loading package base ... linking ... done.
[1 of 1] Compiling Main          ( lect02b.hs, interpreted )
*Main> quickCheck prop_square
Loading package old-locale-1.0.0.0 ... linking ... done.
Loading package old-time-1.0.0.0 ... linking ... done.
Loading package random-1.0.0.0 ... linking ... done.
Loading package mtl-1.1.0.1 ... linking ... done.
Loading package QuickCheck-2.1 ... linking ... done.
+++ OK, passed 100 tests.
*Main> quickCheck prop_squares
+++ OK, passed 100 tests.
*Main> quickCheck prop_pyth
+++ OK, passed 100 tests.
```

# Part IV

# The Rule of Leibniz (reprise)

# Gottfried Wilhelm Leibniz (1646–1716)

Anticipated symbolic logic, and discovered calculus (independently of Newton).

> “The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate, without further ado, to see who is right.”

> “In symbols one observes an advantage in discovery which is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then indeed the labor of thought is wonderfully diminished.”