Informatics 1

Functional Programming Lectures 1 and 2

Monday 24–Tuesday 25 September 2012

Introduction, Functions

Don Sannella

University of Edinburgh

Welcome to Informatics 1, Functional Programming!

Informatics 1 course organiser: Paul Anderson

Functional programming (Inf1-FP)

Lecturer: Don Sannella

Teaching assistant: Chris Banks

Computation and logic (Inf1-CL)

Lecturer: Dave Robertson

Teaching assistant: Xi Bai

Informatics Teaching Organization (ITO):

Kirsten Belk

Where to find us

IF – Informatics Forum

AT – Appleton Tower

Inf1 course organiser: Paul Anderson `dcspaul@inf.ed.ac.uk` IF 1.24

Functional programming (Inf1-FP)

Lecturer: Don Sannella `Don.Sannella@inf.ed.ac.uk` IF 4.04

Teaching assistant: Chris Banks `C.Banks@ed.ac.uk` IF 3.50

Informatics Teaching Organization (ITO):

Kirsten Belk AT 4.02

Required text and reading

Haskell: The Craft of Functional Programming (Third Edition),
Simon Thompson, Addison-Wesley, 2011.

Reading assignment

Monday 24 September 2012	Chapters 1–3 (pp. 1–66)
Monday 1 October 2012	Chapters 4–7 (pp. 67–176)
Monday 8 October 2012	Chapters 8–9 (pp. 177–212)

The assigned reading covers the material very well with plenty of examples.

There will be no lecture notes, just the book. *Get it and read it!*

Lab Week Exercise and Drop-In Labs

Monday	3–5pm (demonstrator 3:30-4:30pm)	Computer Lab West
Tuesday	2–5pm (demonstrator 2:00-3:00pm)	Computer Lab West
Wednesday	2–5pm (demonstrator 2:00-3:00pm)	Computer Lab West
Thursday	2–5pm (demonstrator 2:00-3:00pm)	Computer Lab West
Friday	3–5pm (demonstrator 3:30-4:30pm)	Computer Lab West

Computer Lab West – Appleton Tower, fifth floor

Lab Week Exercise

submit by 5pm Friday 28 September 2012

do all the parts

Tutorials

ITO will assign you to tutorials, which start in Week 3.

Attendance is compulsory.

Tuesday/Wednesday	Computation and Logic
Thursday/Friday	*Functional Programming*

Contact the ITO if you need to change to a tutorial at a different time.

You *must* do each week's tutorial exercise! Do it *before* the tutorial!

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

You may *collaborate*, but you are responsible for knowing the material.

Mark of 0% on tutorial exercises means you have no incentive to *plagiarize*.

But *you will fail the exam if you don't do the tutorial exercises!*

Formative vs. Summative

0%	Lab week exercise
0%	Tutorial 1
0%	Tutorial 2
0%	Tutorial 3
10%	Class Test
0%	Tutorial 4
0%	Tutorial 5
0%	Tutorial 6
0%	Tutorial 7
0%	Mock Test
0%	Tutorial 8
90%	Final Exam

Course Webpage

See *http://www.inf.ed.ac.uk/teaching/courses/inf1/fp/* for:

- Course content
- Organisational information: what, where, when
- Lecture slides, reading assignment, *tutorial exercises*, solutions
- Course blog
- Past exam papers
- Programming competition
- Other resources

Any questions?

Any questions?

Questions make you *look good*!

Don's *secret technique* for asking questions.

Don's *secret goal* for this course

Part I

Introduction

Computational Thinking

> "In their capacity as a tool computers will be but a ripple on the surface of our culture. In their capacity as intellectual challenge, they are without precedent in the cultural history of mankind."
>
> Edsgar Dijkstra, 1930–2002

“Informatics” vs. “Computer Science”

> “Computer science is no more about computers than astronomy is about telescopes.”
>
> Edsgar Dijkstra, 1930–2002

Why learn Haskell?

- Important to learn many languages over your career
- Functional languages increasingly important in industry
- Puts experienced and inexperienced programmers on an equal footing
- Operate on data structure *as a whole* rather than *piecemeal*
- Good for concurrency, which is increasingly important

Linguistic Relativity

"Language shapes the way we think, and determines what we can think about."

Benjamin Lee Whorf, 1897–1941

"The limits of my language mean the limits of my world."

Ludwig Wittgenstein, 1889–1951

"A language that doesn't affect the way you think about programming, is not worth knowing."

Alan Perlis, 1922–1990

What is Haskell?

- A functional programming language
- For use in education, research, and industry
- Designed by a committee
- Mature—over 20 years old!

"A History of Haskell: being lazy with class",
Paul Hudak (Yale University),
John Hughes (Chalmers University),
Simon Peyton Jones (Microsoft Research),
Philip Wadler (Edinburgh University),
The Third ACM SIGPLAN History of Programming Languages Conference (HOPL-III),
San Diego, California, June 9–10, 2007.

Look at these web pages:

ICFP 2012

icfpconference.org/icfp2012/

Jane Street Capital

www.janestreet.com/technology/ocaml.php

Microsoft

www.microsoft.com/casestudies/
Case_Study_Detail.aspx?casestudyid=4000006794

Families of programming languages

- Functional

  Erlang, F#, Haskell, Hope, Javascript, Miranda, O'Caml, Racket, Scala, Scheme, SML

  - More powerful
  - More compact programs

- Object-oriented

  C++, F#, Java, Javascript, O'Caml, Perl, Python, Ruby, Scala

  - More widely used
  - More libraries

Functional programming in the real world

- Google MapReduce, Sawzall
- Ericsson AXE phone switch
- Perl 6
- DARCS
- XMonad
- Yahoo
- Twitter
- Facebook
- Garbage collection

Functional programming is the new new thing

Erlang, F#, Scala attracting a lot of interest from developers

Features from functional languages are appearing in other languages

- Garbage collection Java, C#, Python, Perl, Ruby, Javascript
- Higher-order functions Java, C#, Python, Perl, Ruby, Javascript
- Generics Java, C#
- List comprehensions C#, Python, Perl 6, Javascript
- Type classes C++ "concepts"

Part II

Functions

What is a function?

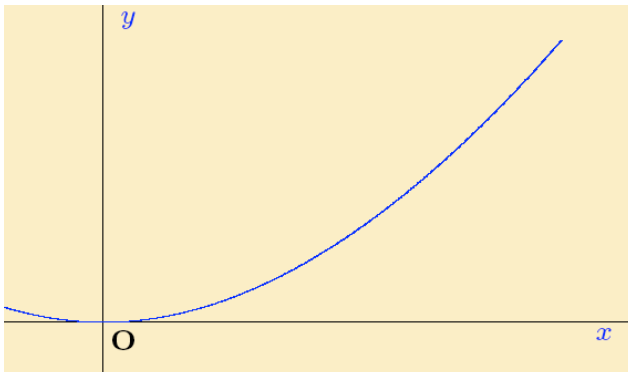
- A recipe for generating an output from inputs:
  "Multiply a number by itself"

- A set of (input, output) pairs:
  (1,1) (2,4) (3,9) (4,16) (5,25) ...

- An equation:

$$f\ x = x^2$$

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

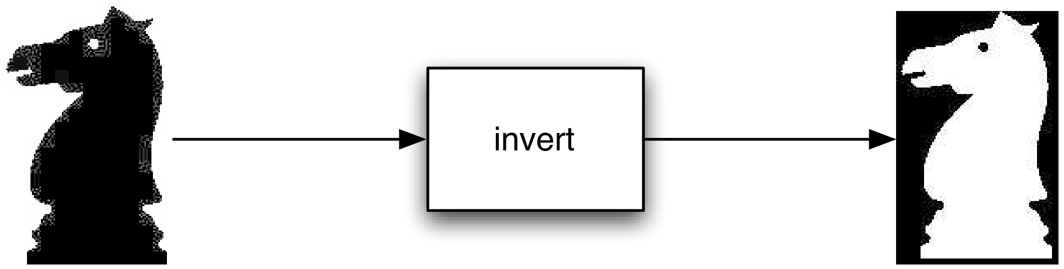


Kinds of data

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

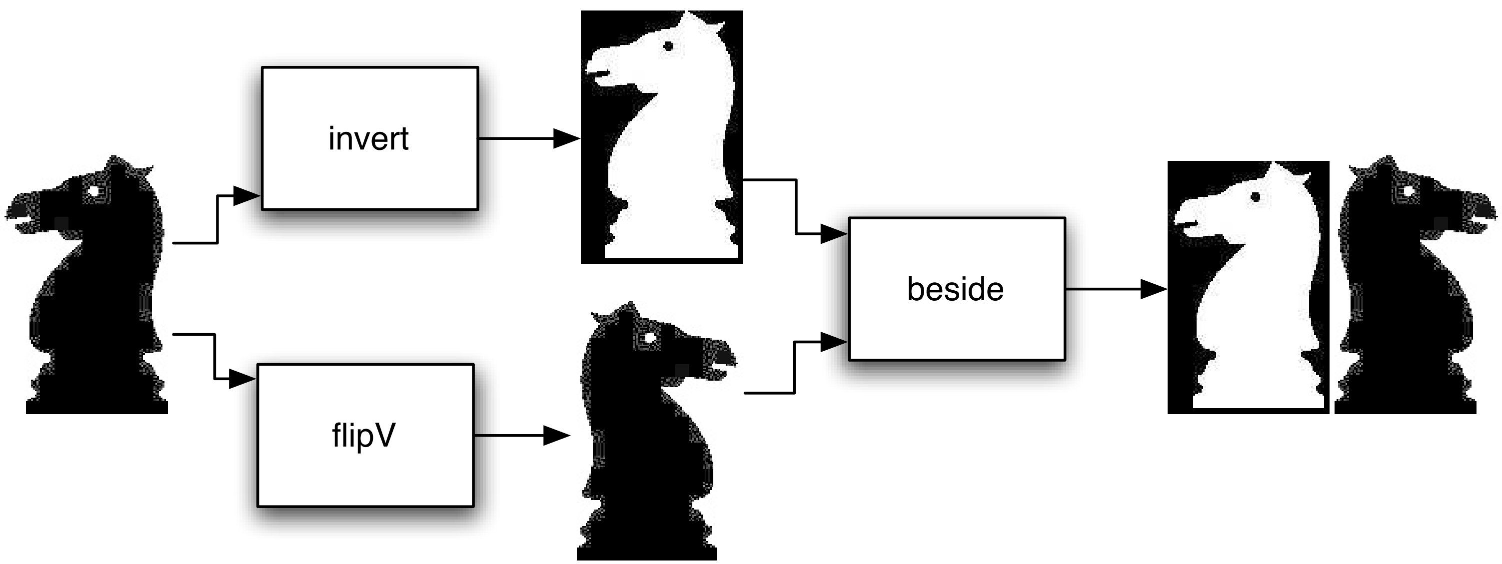
Applying a function

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

invert knight
```



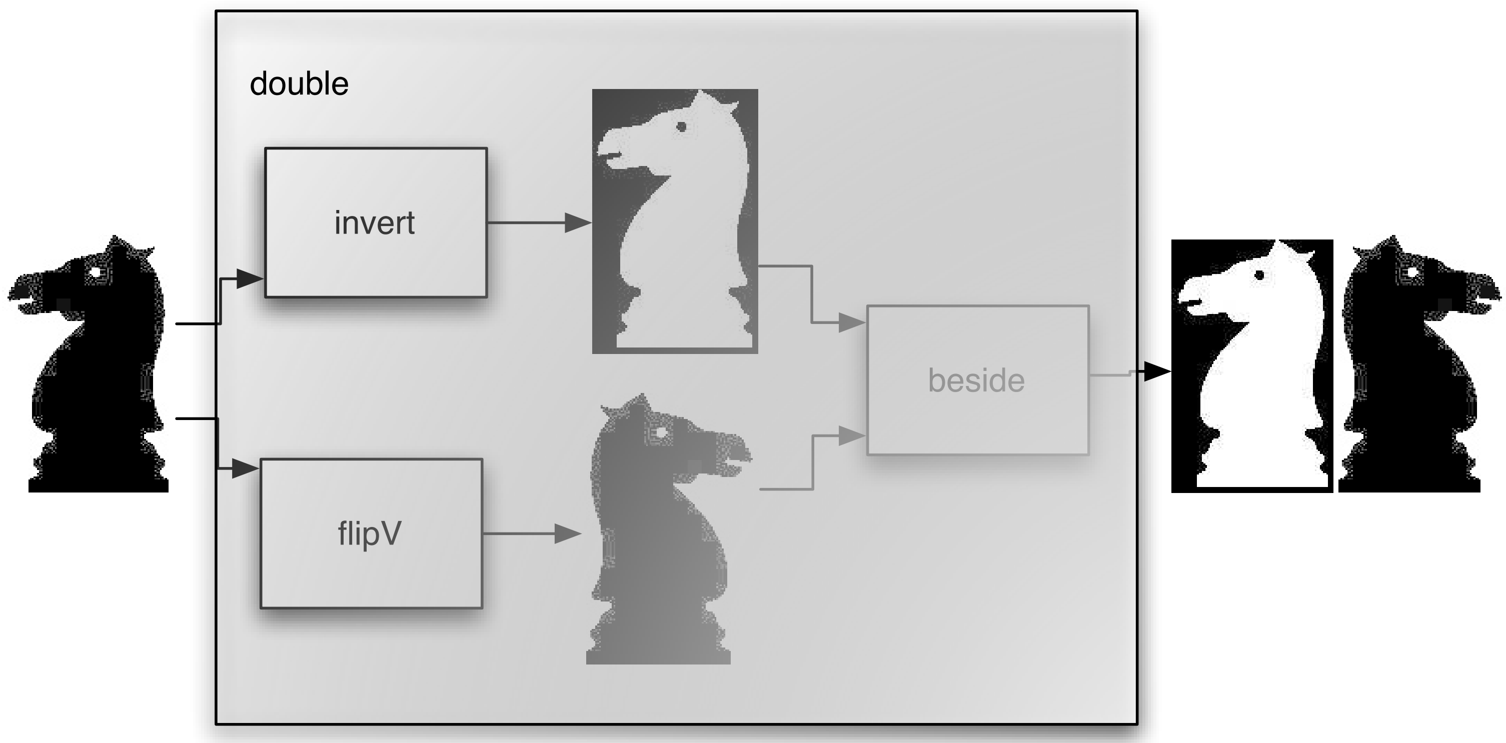
Composing functions

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

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



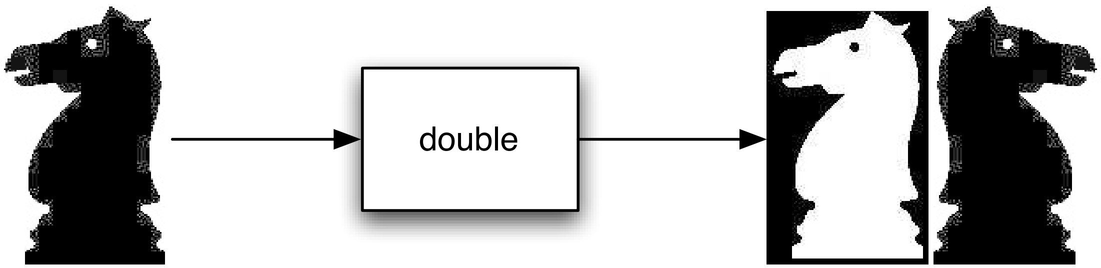
Defining a new function

```
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

```
double :: Picture -> Picture
```

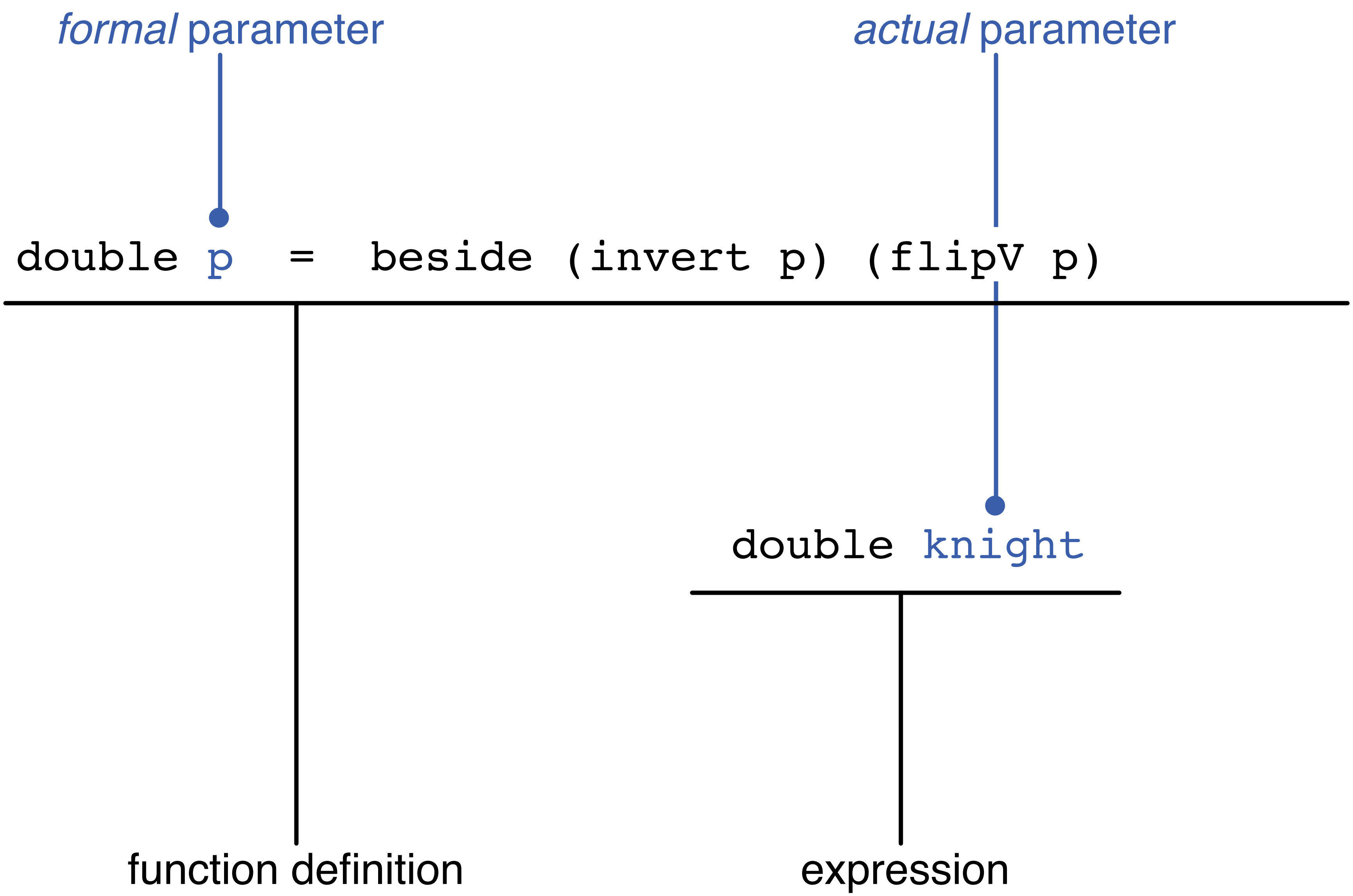
Function declaration

```
double p  =  beside (invert p) (flipV p)
```

function name — `double`

function body — `beside (invert p) (flipV p)`

Terminology



Part III

The Rule of Leibniz

Operations on numbers

```
[melchior]dts: 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

squares.hs

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

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

Testing our functions

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

Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( squares.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 squares.hs)

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

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

Evaluation

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

Loading package base-1.0 ... linking ... done.
Compiling Main             ( squares.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
```

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

The Rule of Leibniz

- Identity of Indiscernables: "No two distinct things exactly resemble one another." — Leibniz

  That is, two objects are identical if and only if they satisfy the same properties.

- "A difference that makes no difference is no difference." — Spock

- "Equals may be substituted for equals." — My high school teacher

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
  =
      (3 + 4) + 5
```

Part IV

QuickCheck

QuickCheck properties

squares_prop.hs

```
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 the program

```
[melchior]dts: ghci squares_prop.hs
GHCi, version 6.8.3: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( squares_prop.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 V

The Rule of Leibniz (reprise)

Gottfried Wilhelm Leibniz (1646–1716)

Gottfried Wilhelm Leibniz (1646–1716)

Anticipated symbolic logic, discovered calculus (independently of Newton), introduced the term "monad" to philosophy.

> "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."