

Informatics 1
Functional Programming Lectures 1 and 2

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: Karoliina Lehtinen

Computation and logic (Inf1-CL)

Lecturer: Michael Fourman

Teaching assistant: ???

Informatics Teaching Organization (ITO):

Gregor Hall

Where to find us

IF – Informatics Forum

FH – Forrest Hill

Inf1 course organiser: [Paul Anderson](mailto:dcspaul@inf.ed.ac.uk) dcspaul@inf.ed.ac.uk IF 1.24

Functional programming (Inf1-FP)

Lecturer: [Don Sannella](mailto:Don.Sannella@ed.ac.uk) Don.Sannella@ed.ac.uk IF 5.12

Teaching assistant: [Karoliina Lehtinen](mailto:M.K.Lehtinen@sms.ed.ac.uk) M.K.Lehtinen@sms.ed.ac.uk IF
5.34

Informatics Teaching Organization (ITO):

[Gregor Hall](#) FH 1.B15

Required text and reading

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

or

Learn You a Haskell for Great Good!
Miran Lipovača, No Starch Press, 2011.

Reading assignment

Monday 21 September 2015

Thompson: parts of Chap. 1-3

Lipovača: parts of intro, Chap. 1-2

Monday 28 September 2015 etc.

See the course web page

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

There will be no lecture notes, just the books. *Get one of them and read it!*

Linux / DICE Tutorial

Tuesday	22 September 2015	2–4pm	Forrest Hill Drill Hall
Wednesday	23 September 2015	2–4pm	Forrest Hill Drill Hall
Thursday	24 September 2015	2–4pm	Forrest Hill Drill Hall

Forrest Hill Drill Hall, FH 1.B30

Lab Week Exercise and Drop-In Labs

Monday	3–5pm (demonstrator 3:00-4:00pm)	Forrest Hill Drill Hall
Tuesday	2–5pm (demonstrator 3:00-4:00pm)	Forrest Hill Drill Hall
Wednesday	2–5pm (demonstrator 3:00-4:00pm)	Forrest Hill Drill Hall
Thursday	2–5pm (demonstrator 3:00-4:00pm)	Forrest Hill Drill Hall
Friday	3–5pm (demonstrator 3:00-4:00pm)	Forrest Hill Drill Hall

Forrest Hill Drill Hall, FH 1.B30

Lab Week Exercise

submit by 5pm Friday 2 October 2015

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!*

Automated Feedback

CamlBack (UCLA) will give automated feedback on tutorial exercises.

Use is optional, but it's good to get immediate feedback.

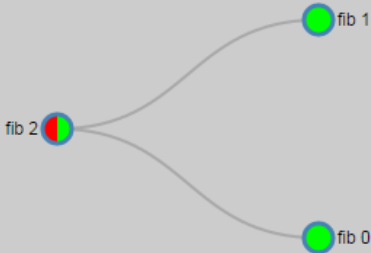
Problems

Feedback

incorrect function calls

correct function calls; incorrect result

correct function calls and result



A call graph showing a node labeled 'fib 2' on the left. Two curved lines branch out from 'fib 2' to two nodes on the right labeled 'fib 1' (top) and 'fib 0' (bottom). The 'fib 2' node has a red, green, and blue segment. The 'fib 1' and 'fib 0' nodes are entirely green.

Problem 1

```
1 fib :: Integer -> Integer
2 fib 0 = 0
3 fib 1 = 1
4 fib n = fib (n-1) * fib (n-2)
```

Check Function

Reset

Failed a test. Hover over the call nodes above for more information.

Automated Feedback

CamlBack (UCLA) will give automated feedback on tutorial exercises.

Use is optional, but it's good to get immediate feedback.

Problems

Feedback

incorrect function calls

correct function calls; incorrect result

correct function calls and result

fib 2

fib 1

fib 0

argument(s): 2

returned: 0

expected: 1

Results of child function calls are not combined properly

Problem 1

```
1 fib :: Integer -> Integer
2 fib 0 = 0
3 fib 1 = 1
4 fib n = fib (n-1) * fib (n-2)
```

Check Function

Reset

Failed a test. Hover over the call nodes above for more information.

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 Exam
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
- Past exam papers
- Programming competition
- Other resources

Any questions?

Questions make you *look good*!

Don's *secret technique* for asking questions.

Don's *secret goal* for this course

Piazza

Use the Piazza online Inf1-FP forum:

- For *asking questions* outside lectures
- For *reading answers* to questions asked by others
- For *writing answers* to questions asked by others

See the course webpage for the link and for sign-up instructions

Part I

Introduction

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

Operating on data structure as a whole rather than piecemeal:

In an imperative language, you often use a loop to operate on the items in a data structure, one at a time.

In a functional language, you tend to operate on the whole data structure: Whoosh!

This leads to higher-level thinking about algorithms.

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

Look at these web pages:

ICFP 2015

icfpconference.org/icfp2015/

Jane Street Capital

www.janestreet.com/technology/

Facebook

www.wired.com/2015/09/

facebook.com/new-anti-spam-system-hints-future-coding/

Families of programming languages

- Functional

Erlang, F#, Haskell, Hope, Javascript, Miranda, OCaml, Racket, Scala, Scheme, SML

- More powerful
- More compact programs

- Object-oriented

C++, F#, Java, Javascript, OCaml, 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?

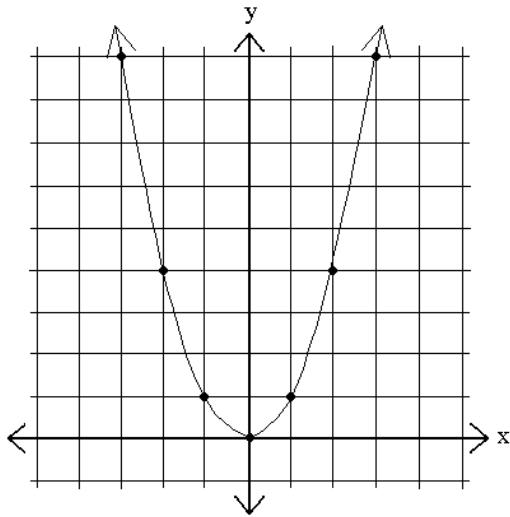
- A recipe for generating an output from inputs:
“Multiply a number by itself”

- 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

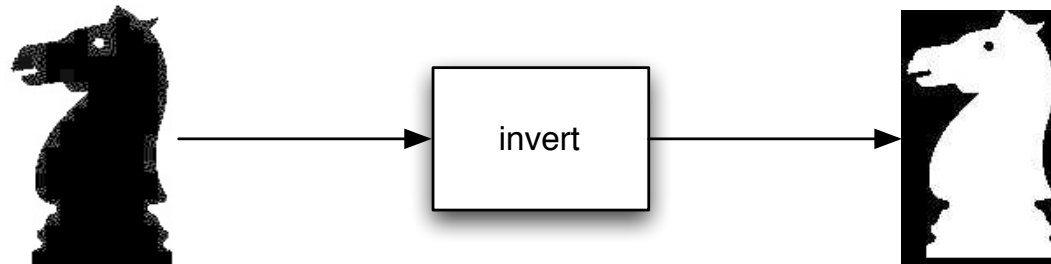
- Integers: 42, -69
- Floats: 3.14
- Characters: 'h'
- Strings: "hello"
- Booleans: True, False
- Pictures: 

Applying a function

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

```
knight :: Picture
```

```
invert knight
```



`invert` is a function. Every value in Haskell has a type, maybe more than one. We write `value :: type`.

A type is a category of values. Types of functions contain arrows.

When we write an expression (example: `invert knight`) then Haskell will complain if it can't make sense of the types.

Composing functions

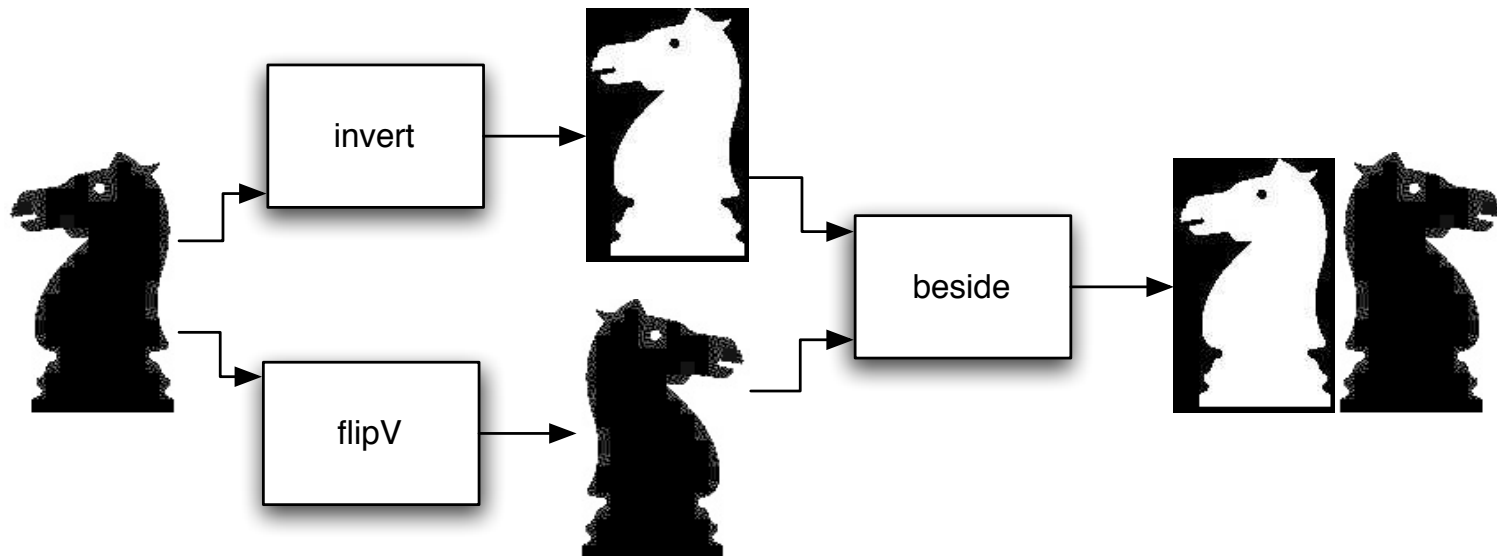
```
beside :: Picture -> Picture -> Picture
```

```
flipV :: Picture -> Picture
```

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

```
knight :: Picture
```

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

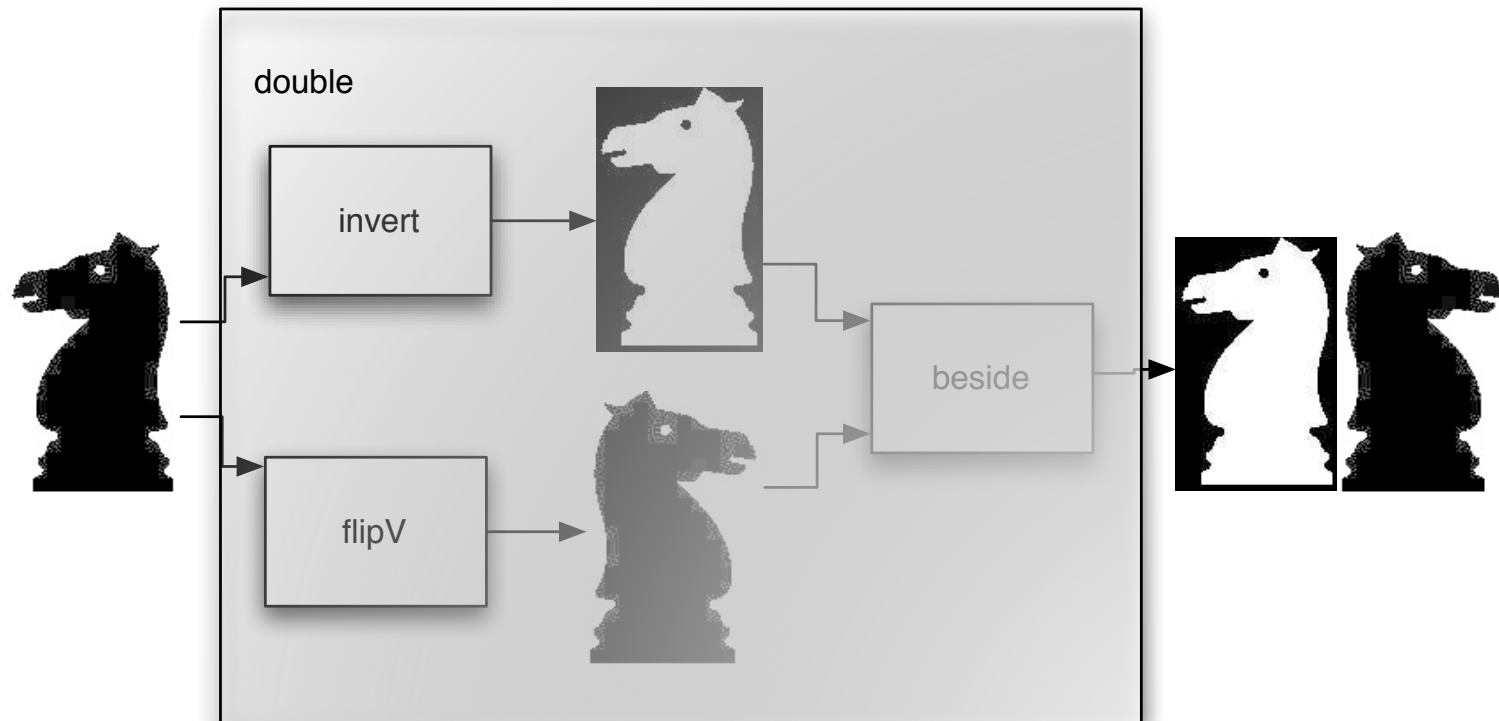


beside is a function with two arguments. There is a reason for writing the type this way, to be explained later.

Defining a new function

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

```
double knight
```

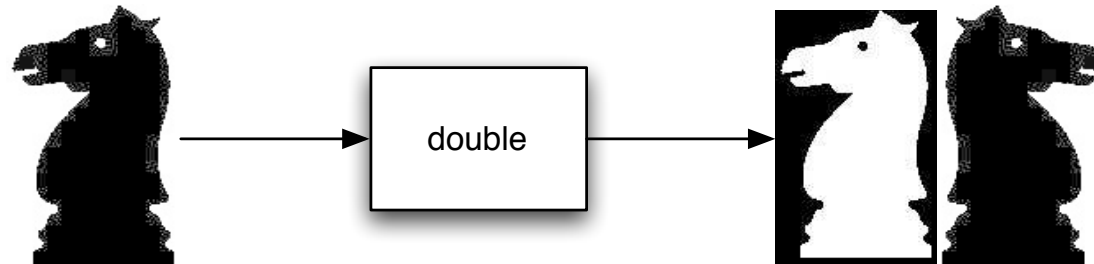


Functions are defined using equations. The variable name (`p`) is irrelevant - we could use `pic` or `x` instead.
`double` produces the picture we had before, but packaged to work on any picture, not just knight.

Defining a new function

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

```
double knight
```



We could write beside as an infix function instead:

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

Any function can be written as infix by enclosing it in backquotes.

Terminology

Type signature

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

Function declaration

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

function name

function body

Terminology

