

Informatics 1

Functional Programming Lecture 1

Monday 29 September 2008

Functions

Philip Wadler

University of Edinburgh

Any questions?

Tutorial, Labs, Lab week

ITO will assign you to tutorials, starting next week

Tuesday/Wednesday Computation and Logic

Thursday/Friday Functional Programming

Drop-in laboratories

Mondays 35pm West

Tuesdays 25pm West

Wednesdays 25pm West

Thursdays 25pm South

Fridays 35pm West

Computer Lab West and South, Appleton Tower, level 5 (not level 4)

Lab Week Exercise due 5pm Friday 3 October

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 Mon 29 Sep 2008.

Thompson, Chapters 4–5 (pp. 53–95): by Mon 7 Oct 2008.

Thompson, Chapters 6–7 (pp. 96–134): by Mon 15 Oct 2008.

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.”

Edsger Dijkstra, 1930–2002



JANE ST. CAPITAL

[About Jane Street](#) | [Careers](#) | [Contact Information](#)

Research and Technology Jobs at Jane Street

Jane Street Capital is a proprietary trading firm that operates around the clock and around the world. We bring a deep understanding of trading, a scientific approach, and innovative technology to bear on the problem of trading profitably on the world's highly-competitive financial markets. We run a small, nimble operation where technology and trading are tightly integrated.

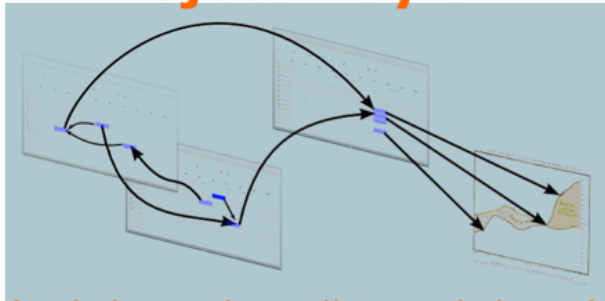
At Jane Street, there is room to get deeply involved in a number of areas at the same time. We are actively looking for people interested in software development, system administration, and quantitative research--potentially all on the same day.

The ideal candidate has:

- A commitment to the practical. One of the big attractions of our work is the opportunity to apply serious ideas to real-world problems.
- Experience with functional programming languages (OCaml, SML, Scheme, Haskell, Lisp, F#, Erlang, etc) is important. Applicants should also have experience with UNIX and a deep understanding of computers and technology.
- A strong mathematical background. This is a must for candidates interested in research, and includes a good understanding of probability and statistics, calculus, algorithms, etc. We draw on ideas from everywhere we can, so we value interest and experience in a range of scientific fields.
- Good second-order knowledge. In trading, understanding the boundary between what you do and don't know is as (or more) important than how much you know.

Internet Startup Jobs

Project Playfair



'websheets that talk to websheets'

Edinburgh-based start-up Project Playfair has won the prestigious Seedcamp Venture Capital Start-Up Competition and is now hiring smart young things. (These are equity positions).

You will have strong experience in functional programming (Erlang preferred but not necessary). Skills in the Ubuntu platform (6.06 LTS Dapper Drake) is also desirable.

We are looking for immediate starts and will consider current students on a part time basis – 20 hours a week. Interviews will be Friday 14th September 7pm in Edinburgh at 34a Howe St

To apply phone Gordon 07776 251669



Seedcamp is a VC funding competition organised by the following VC firms:



dfjesprit

wellingtonpartners



atlasventure

BENCHMARK CAPITAL Index VENTURES

ACCEL

Seedcamp backers have funded:



Why learn Haskell?

- Important to learn many languages over your career
- Learn to operate on data structures all at once rather than a piece a time
- Puts experienced and inexperienced programmers on a more equal footing

What is Haskell?

- A functional programming language
- Pure
- Lazy
- Uses type classes
- For use in education, research, and industry
- Designed by a committee
- Recently celebrated its 20'th birthday

“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.

Families of programming languages

- Functional (Erlang, F#, Haskell, Hope, Javascript, Miranda, O'Camel, Scala, Scheme, SML)
 - More powerful
 - More compact programs
- Object-oriented (C++, F#, Java, Javascript, O'Camel, Perl, Python, Ruby, Scala)
 - More widely used
 - More libraries

Functional programming in the real world

- Google MapReduce, Sawzall
- Ericsson AXE phone switch
- Jane Street Capital
- Credit Suisse
- Morgan Stanley
- Perl 6
- DARCS
- XMonad
- Yahoo
- Twitter
- Garbage collection
- F#

Functional programming is the next next thing

Features from functional languages are appearing in other languages

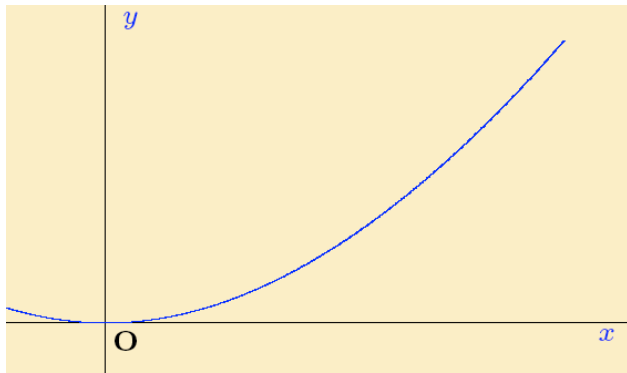
- Garbage collection (Java, C#, Python, Perl, Ruby, Javascript)
- Higher-order functions (Java, C#, Python, Perl, Ruby, Javascript)
- Generics (Java, C#)
- List comprehensions (C#, Python, Perl 6, Javascript)
- Type classes (C++ “concepts”)

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



Kinds of data

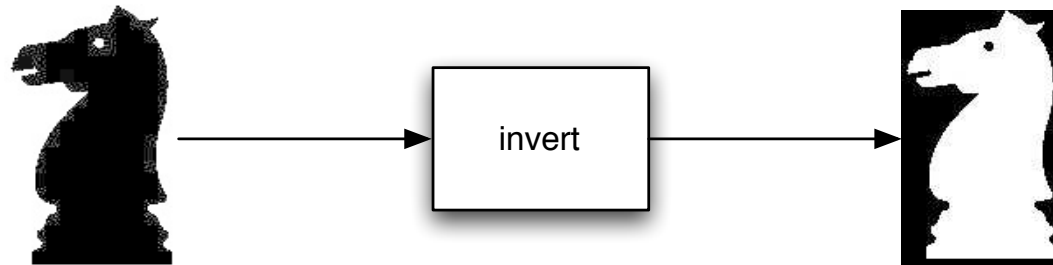
- Integers: 42, -69
- Floats: 3.14
- Characters: 'h'
- Strings: "hello"
- Pictures: 

Applying a function

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

```
knight :: Picture
```

```
invert knight
```



Composing functions

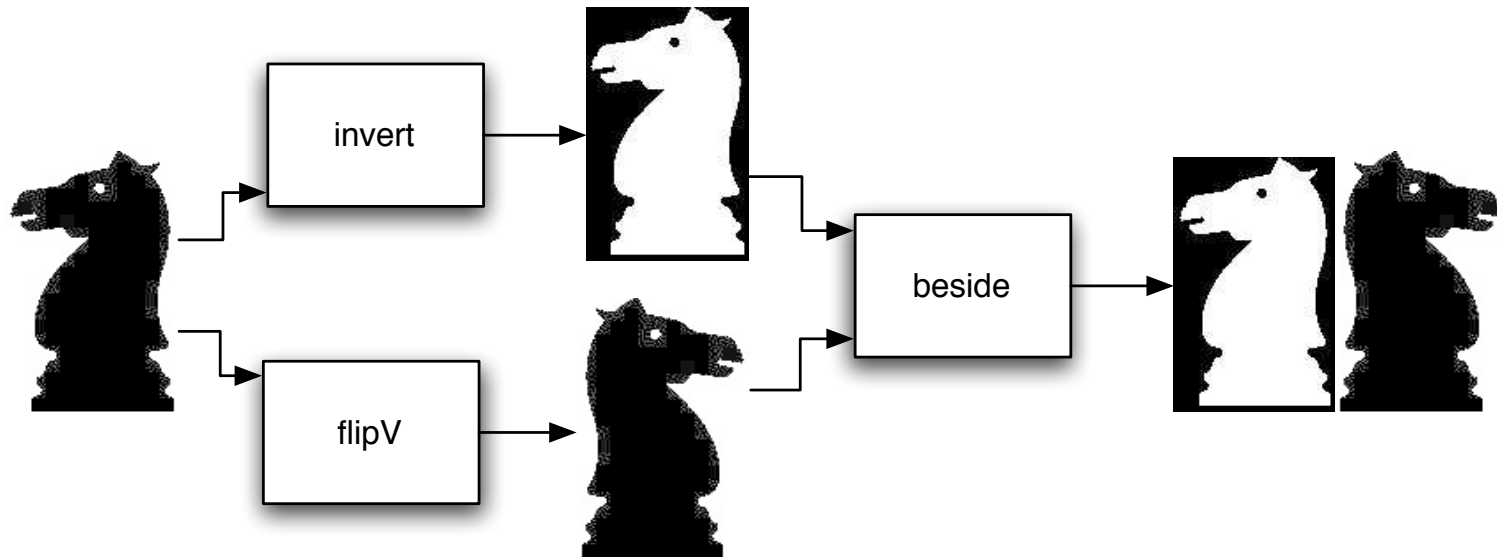
```
beside :: Picture -> Picture -> Picture
```

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

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

```
knight :: Picture
```

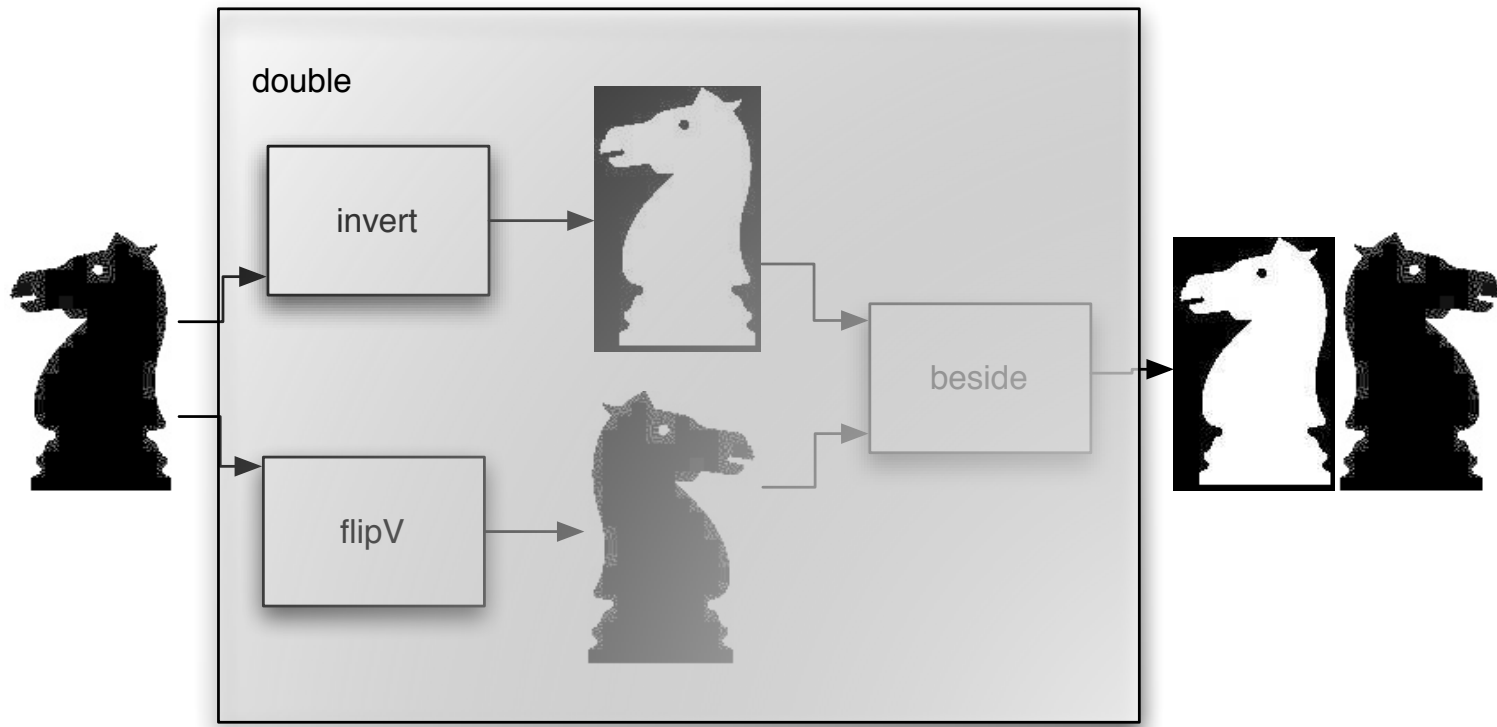
```
beside (invert knight) (flipV knight)
```



Defining a new function

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

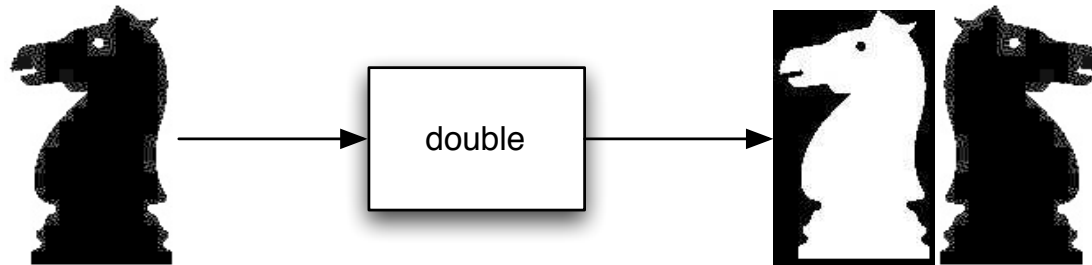
```
double knight
```



Defining a new function

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

```
double knight
```



Terminology

Type signature

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

Function declaration

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

The diagram illustrates the components of a function declaration. A blue dot is placed above the word 'makePicture' in the code. A vertical blue line descends from this dot to the text 'function name' below. A horizontal green line is drawn under the entire right-hand side of the declaration, 'sideBySide (invert p) (flipV p)'. A vertical green line descends from the center of this horizontal line to the text 'function body' below.

Terminology

