Concepts of programming languages

My title here

Author name here

Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Please use Markdown to write your slides.

This makes sure that slides will be consistent – and easy for me to edit in the future.

Start a new slide with by beginning a new line three dashes `---`.

For example:

```
---

My contents

---
```

Titles

You can use the hash symbol # to make the title of a slide.

Subtitle

You can use more than one hash symbol ## to have subtitles on your slide.

- Bullet lists
- are pretty easy
- too!

Emphasis

You can include a word in asterisks to add *emphasis* or two asterisks to make it **bold**.

That is:

```
*emphasis* vs **bold**
```

Images

Please include any images in the `img` subdirectory.

You can refer to images using the usual markdown syntax:

Figure 1: My caption

Staged builds

This is quite easy

Staged builds

This is quite easy

Just insert `. . .` on a new line when you want the slide to appear incrementally.

Code

You can use backticks to include inline code such as `x` or `y`.

Use three backticks to introduce a code block:

```
main = print "Hello world!"
```

Syntax highlighting

There are syntax highlighting options for the most widely used languages.

```
foo y = let x = 4 in x + z
  where
  z = 12
```


Making slides

I've included a Makefile to build slides.

You will need to have the Haskell tool `pandoc` installed:

```
> cabal install pandoc
> make
```

Working with markdown

You may want to install the markdown mode for emacs (or some other editor of choice).

I've included some file local variables at the bottom of this file – you may find them useful.

Inline latex

You can always use *inline* LaTeXcommands if you want.

But try to avoid this if you can.

Most Markdown commands should suffice.

LaTeXis useful for formula's

$$\tau + x = \sigma \tag{1}$$

Or inline formulas, enclosed in dollar symbols like so $\tau + x$.

Questions

If you can’t get things to work, don’t hesitate to get in touch!

Janus

A reversible programming language

Not turing complete!

Reversibility

Every statement can be reverted. No history is stored.

```
x += y * 3
```

Reversibility

Every statement can be reverted. No history is stored.

```
x += y * 3
```

```
x -= y * 3
```

Injective functions

Reversible languages can only compute injective functions.

$$\forall x, y : f(x) = f(y) \implies x = y \tag{2}$$

Every output has only a single input.

Injective functions

Reversible languages can only compute injective functions.

$$\forall x, y : f(x) = f(y) \implies x = y \tag{2}$$

Every output has only a single input.

$$h(x) = (x, g(x)) \tag{3}$$

Turing completeness

Turing machines can compute non-injective functions.

Reversible languages are not turing complete.

Reversible Turing complete.

Turing machines

Infinite tape of memory

Finite set of states

Transition function

- Current state
- Current symbol on tape
- Write symbol
- Move tape pointer
- Next state

Turing machines

Forward deterministic: given any state and tape, there is at most one transition *from* that state.

Backward deterministic: given any state and tape, there is at most one transition *to* that state.

P is the class of forward deterministic turing machines, NP of non-deterministic turing machines.

Reversible Turing complete: a language that can simulate forward and backward deterministic turing machines.

What do reversible languages compute

Given a forward deterministic turing machine that computes $f(x)$,

There exists a reversible turing machine that computes $x \rightarrow (x, f(x))$.

More memory.