# CISC 260: Programming Paradigms
# Lecture 2

Joshua Dunfield
Queen's University
Fall 2018

# Functional programming in the “real world”

- http://cufp.org/2015/fighting-spam-with-haskell-at-facebook.html
- http://cufp.org/2016/yaron-minsky-keynote.html (OCaml at Jane Street)
- http://cufp.org/2017/using-functional-programming-to-accelerate-translational-research-at-pfizer.html
- (CUFP = Commercial Users of Functional Programming conference)

## Functional programming in the “real world”

- also, parts of Firefox (if we count Rust)

# Some examples of stepping

- Also called “calculation” (the textbook) or “tracing”

$$2 + 2$$

$\Rightarrow$

$$(2 + 2) + (3 + 4)$$

$\Rightarrow$

$\Rightarrow$

$\Rightarrow$

1 / 3

$\Rightarrow$

$2 + 3 * 4$

$\Rightarrow$

$\Rightarrow$

# Checking our expectations in GHCi

# GHCi

- "read-eval-print loop"
  - also found in Python
  - not found in Java, C, C++

1. **read** input
2. **evaluate**
3. **print** result
4. loop back to 1

# GHCi

- To load definitions from a file (.hs, “Haskell script”)

```
:load filename.hs
```

- To reload previous definitions:

```
:reload
```

# Functions

- An anonymous function
  ("lambda" λ; \ looks sort of like λ)

```
\x -> x + 1
```

But we can call this function:

```
(\x -> x + 1) 6
```

or use it inside a definition

# Terminology

`\x -> body` is a lambda

`x` is the *bound variable*, `body` is the *body*

A lambda is a function.

If `f` is a function, then

```
f arg
```

is a **function call** or **function application:**

"apply `f` to `arg`"

# Stepping rule for function application

If

`f` is a function,

and the bound variable of `f` is `x`,

and the body of `f` is `body`,

then

`f arg` ⇒ `body` with `arg` substituted for `x`

# Stepping rule for function application

If

`f` is a function,

and the bound variable of `f` is `x`,

and the body of `f` is `body`,

then

`f arg` ⇒ `body` with `arg` substituted for `x`

Example:

```
    (\x -> x + 1) 6
⇒   x + 1  with  6  substituted for  x
=   6 + 1
⇒   7
```

# Stepping rule for function application

If

`f` is a function,

and the bound variable of `f` is `x`,

and the body of `f` is `body`,

then

`f arg` ⇒ `body` with `arg` substituted for `x`

Example:

```
    (\yow -> yow + 1) 6
⇒   yow + 1  with  6  substituted for  yow
=   6 + 1
⇒   7
```

# Stepping rule for function application

If

`f` is a function,

and the bound variable of `f` is `x`,

and the body of `f` is `body`,

then

`f arg` ⇒ `body` with `arg` substituted for `x`

Example:

```
    (\y -> y + y) 6
⇒   y + y  with  6  substituted for  y
=   6 + 6
⇒   12
```

# Defining functions

- Most functions are **named**

```
functionname boundvar = body
```

# Stepping rule for function application

If

`f` is a function,

and the bound variable of `f` is `x`,

and the body of `f` is `body`,

then

`f arg` ⇒ `body` with `arg` substituted for `x`

```
double x = x + x
```