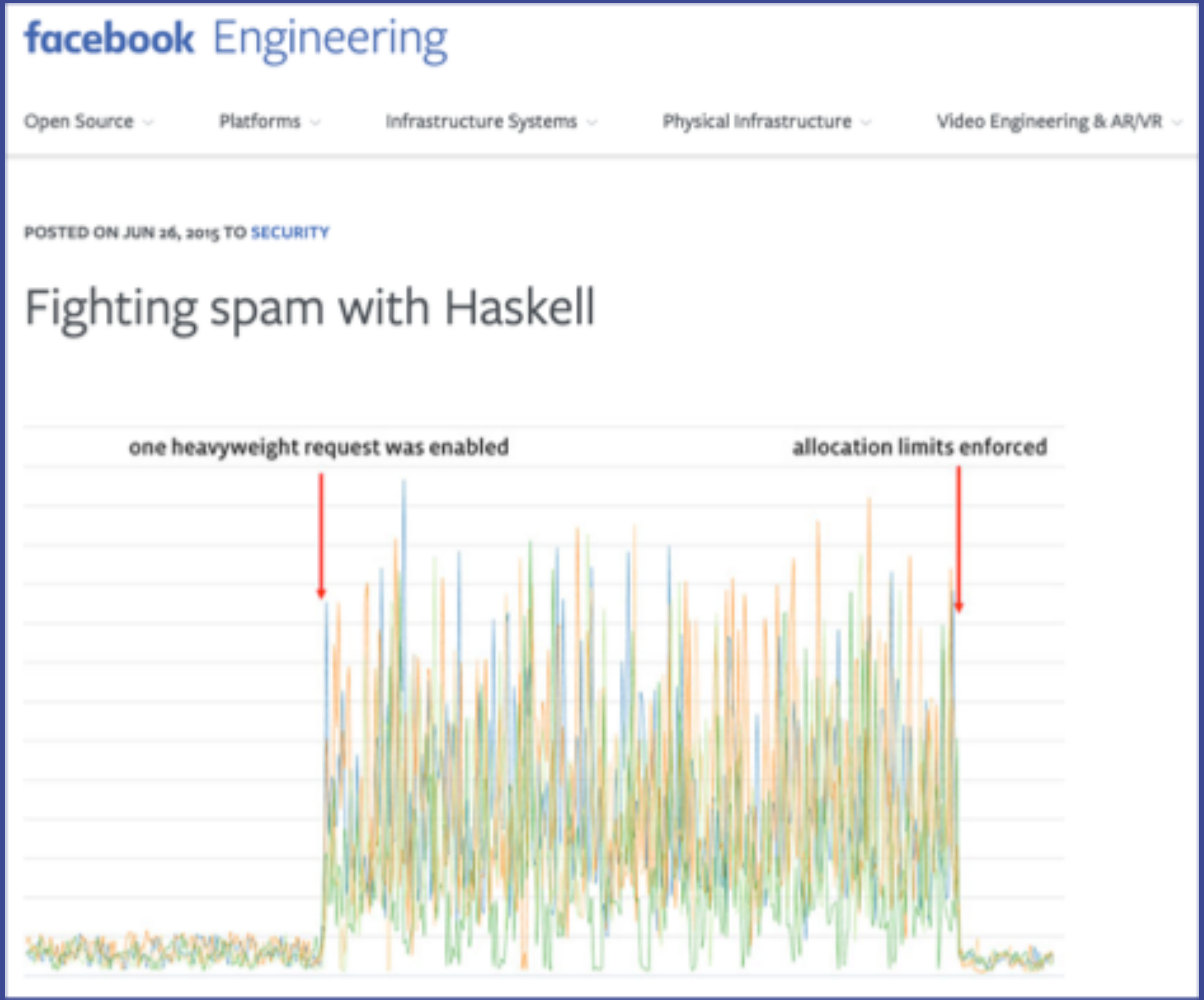
CS 152

Programming Paradigms

Haskell

https://xkcd.com/1312/

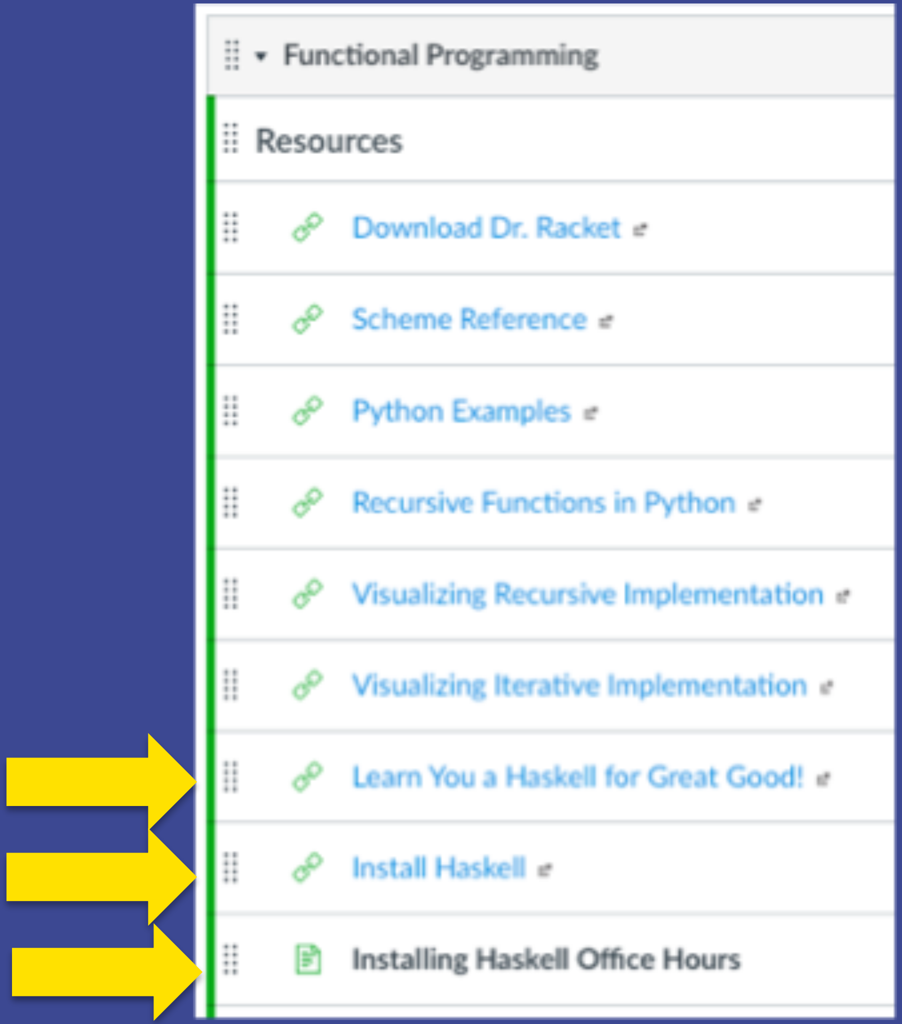
Fighting spam with Haskell



Today

- Install Haskell
- Basics of Haskell
- Haskell vs Scheme:
  - Type checking and type inference
- Exam 1

New Resources on Canvas



Installing Haskell

- Link on Canvas under Resources
- MAC OS:
  - use ghcup to install ghc and cabal-install
  - no need to install stack
  - may need to update you PATH.
- Windows:
  - Open PowerShell as an administrator (elevated prompt) – Right click
  - Install Chocolatey
  - Install haskell-dev
- Issues? Get help on Friday, September 18 at 1PM.

Haskell

- Use any editor to create Haskell programs with hs extension.
- In a terminal window, navigate to the directory containing your Haskell programs
- ghci to invoke the interpreter
- :load yourprogram.hs
- :reload to reload after making changes
- :quit to exit the interpreter

The Basics: Comments

```
{- A comment  that
can span multiple lines -}

-- a single line comment

third = first + second --   an inline comment
```

The Basics: Comments and Bindings

```
{- Lecture 8
Basics Examples
September 2020 -}


-- Bindings
first = 1
second = 4
third = first + second –– bind third to the sum
```

The Basics: Bindings

```
{- Lecture 8
Basics Examples
September 2020 -}

-- Bindings
first = 1
second = 4
first = 0 -- changed my mind
third = first + second –– bind third to the sum
```

```
example1.hs:8:1: error:
    Multiple declarations of ‘first’
    Declared at: example1.hs:6:1
                 example1.hs:8:1
  |
8 | first = 0 -- changed my mind
  | ^^^^^
Failed, no modules loaded.
```

The Basics: Boolean Values

True and False

The Basics: Conditionals

if <condition> then <true_result> else <false_result>

IMPORTANT: <true_result> and <false_result> must have the same (or compatible) type.

if x == 0 then 1 else 1/x ✓

if x == 0 then 1 else False => Error

The Basics: Types

- At the interpreter prompt

  :t or :type to get the type of an expression

  Prelude> :t True

  True :: Bool

- :: means has type

The Basics: Functions

Named functions:

```
square x = x * x
```

```
Prelude> :l example1.hs
[1 of 1] Compiling Main             ( example1.hs, interpreted )
Ok, one module loaded.
*Main> square 4
16
*Main> :t square
square :: Num a => a -> a
```

The Basics: Functions

Named functions with types:

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

Prelude> :l example1.hs
[1 of 1] Compiling Main            ( example1.hs, interpreted )
Ok, one module loaded.
*Main> square 4
16.0
*Main> :t square
square :: Double -> Double
```

The Basics: Recursive Functions

```
factorial 0 = 1
factorial 1 = 1
factorial x = x * factorial (x - 1)
```

```
factorial 5
120
```

The Basics: Anonymous Functions

```
Prelude> (\x -> x * x) 4  -- \ stands for lambda
16
Prelude> (\x y -> x * x + y * y) 3 4
25
```

The Basics: Lists

- Lists in Haskell are homogeneous:

```
xs = [1, 2.4, "Hello"]  => Error
xs = [1, 2.4, 4]
xs
[1.0,2.4,4.0]
 :t xs
xs :: Fractional a => [a]
```

The Basics: Lists

- No cars and cdrs!
- head and tail instead.

```
xs = [1, 2, 3]
head xs
1
tail xs
[2,3]
init xs -- all-but-last
[1,2]
last xs
3
```

The Basics: Lists

- : instead of *cons*

  xs = [1, 2, 3]
  0:xs
  [0,1,2,3]
  'a':xs
  error
  1.2:xs
  [1.2,1.0,2.0,3.0]
  ys:: [Integer]
  ys = [1, 2, 3]
  2.1:ys
  Error

The Basics: Lists

++ To concatenate (append in Scheme)

[1, 2, 3] ++ [4.3, 5, 6]

[1.0,2.0,3.0,4.3,5.0,6.0]

Haskell vs Scheme: Similarities

Haskell and Scheme are both functional languages

- Functions are first-class: they can be used in exactly the same ways as any other value (higher order functions).
- Programs are centered around evaluating expressions rather than executing instructions.
- Referential transparency: no side effects, no mutation: Haskell more pure than Scheme

Haskell vs Scheme: Differences

- Syntax:
  - no parentheses, function applications
  - infix notation
  - if … then … else…
  - lists
  - Pattern matching

- Semantics:
  - Type checking (static vs dynamic)
  - Haskell: type inference
  - Evaluation order: lazy vs applicative order
  - Haskell: function overloading
  - Haskell: fully curried

Type Checking

- Type checking: the process a translator (interpreter or compiler) goes through to determine whether type information in a program is consistent

Dynamic vs Static

- Types of type checking:
  - Dynamic: type information is checked at runtime
  - Static: types are determined from the text of the program and checked by the translator prior to execution

Strong vs Weak Typing

- Python and JavaScript are both dynamically typed… but

Strong vs Weak Typing

- Python is strongly typed. There is no implicit type conversion.

```
amount = 5
currency = "$"
print(currency+amount)
TypeError: Can't convert 'int' object to str implicitly
```

- Compare to JavaScript, which is weakly typed:

```
var amount = 5;
var currency = "$";
console.log(currency+amount);
$5
```

Type Checking in Scheme

- Scheme is a strongly and dynamically typed language: types are rigorously checked at runtime
- Type errors cause program termination
- No types in declarations and no explicit type names
- Variables have no predeclared types, but take on the type of the value they possess

Type Checking in Scheme

```
(define (confuse-me x y)
  (if x
      y
      (+ y "Hello")))

> (confuse-me #t 3)
3
> (confuse-me #f 3)
. . +: contract violation
  expected: number?
  given: "Hello"
  argument position: 2nd
  other arguments...:
```

Type Checking in Haskell

- Haskell is a statically typed language. Types are checked before execution
- Type errors cause compilation errors
- Type inference: Haskell infers the type of expressions that have not been explicitly associated with a type

Type Checking in Haskell

```
simple x y =
  if x
    then y
    else y + 1
```

```
:t simple
simple :: Num a => Bool -> a -> a
```

Type Checking in Haskell

```
confuseMe x y =
  if x
  then y
  else y + "Hello"
```

- error

Exam 1 Logistics

- Wednesday September 23 at 10:30 AM: please start on time
- On Canvas with LockDown Browser and Webcam Monitor
- 60 minutes total
- 6-8 questions for 15 points
- 1 handwritten cheat sheet (front and back)
- Scratch paper
- 15% of grade

Topics

- Historical Overview
- Language Design Criteria
- Functional Programming
  - Characteristics, advantages
  - Referential transparency
  - Scheme
  - Haskell – until today

Exam 1

Writing code:

- Scheme

Reading code:

- Haskell
- Python

How to prepare

- Review lectures
- Review iClicker quizzes
- Homework assignments
- Cheat sheet

Scheme Question Examples

- What does the following expression evaluate to?
- Write a Scheme function that ...
- Is this Scheme function tail recursive?
- What is the space complexity of this Scheme function?
- Turn a Scheme function into a tail recursive function
- Turn a Scheme function into a function that runs in constant space

Reminders

- Homework 4: install and use Haskell
  - Install before Friday
  - Issues? Office hours with TA on Friday at 1PM.
- Exam 1: September 23
  - Take the practice quiz if you have not done so yet
- Next: More Haskell