http://xkcd.com/1312/

CODE WRITTEN IN HASKELL IS GUARANTEED TO HAVE NO SIDE EFFECTS.
...BECAUSE NO ONE WILL EVER RUN IT?

Tony Hoare’s

“Hints on Programming Language Design”

CS 252:

Advanced Programming Language Principles

Introduction to Haskell

Prof. Tom Austin
San José State University

Key traits of Haskell

1. Purely functional
2. Lazy
3. Statically typed
4. Type inference
5. Fully curried functions

Interactive Haskell

```
$ ghci
GHCi, version 7.6.3
...
Prelude> 3 + 4
7
Prelude> let f x = x + 1
Prelude> f 3
4
Prelude>
```

Needed in ghci, but not in a script

Running Haskell from Unix command line

```
$ cat helloWorld.hs
main :: IO ()
main = do
  putStrln "Hello World"
$ runhaskell helloWorld.hs
Hello World
$
```

Haskell Base Types

- Int – bounded integers
- Integer – unbounded
- Float
- Double
- Bool
- Char

Lists

- Comma separated, as in Java.
- Some useful operators:
  - `++` concatenation
  - `:` prepend an item
  - `!!` get an element at the given index
  - `head` first item
  - `tail` rest of the list
  - `last` last item
  - `init` the beginning part of the list

List examples

```
Prelude> "I hate the homeless" ++
  "ness problem that plagues our city"
"I hate the homelessness problem that
plagues our city"
Prelude> let s = "bra" in
 s !! 2 : s ++ 'c' : last s : "da" ++ s
"abracadabra"
Prelude>
```

Ranges

```
Prelude> [1..15]
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15]
Prelude> ['a'..'z']
"abcdefghijklmnopqrstuvwxyz"
Prelude> [1,3..27]
[1,3,5,7,9,11,13,15,17,19,21,23,25,27]
Prelude> let evens = [2, 4..]
Prelude> take 5 evens
[2,4,6,8,10]
```

List Comprehensions

- Based on set notation:

$$S = \{\, 2 \cdot x \mid x \in \mathbb{N},\ \ x \le 10 \,\}$$

- The equivalent in Haskell is:

```
[2*x | x <- [1..10]]
```

- What does this give us?

```
[(a,b,c) | a<-[1..10],
           b<-[1..10],
           c<-[1..10],
           a^2 + b^2 == c^2 ]
```

A "tuple"

A Simple Function

```
> let inc x = x + 1
> inc 5
6
> inc 'c'
```

```
<interactive>:9:1:
  No instance for (Num Char)
arising from a use of `inc'
  Possible fix: add an instance
declaration for (Num Char)
  In the expression: inc 'c'
  In an equation for `it': it =
inc 'c'
```

```
> inc 2.5
3.5
```

Is this the behavior that we want?

Adding a *type signature*.

Now we will get a compilation error on `inc 2.5`

```
inc :: Int -> Int
inc x = x + 1
```

```
> inc (-5)
-4
```

Note the parens:
`inc -5`
would be a syntax error

Is *this* the behavior that we want?

Using *pattern matching*.

```
inc :: Int -> Int
inc x | x < 0 =
  error "no negative nums"
inc x = x + 1
```

This is a *guard condition*

Can't reassign variables in Haskell

"If you say that a is 5, you can't say it's something else later because you just said it was 5.

What are you, some kind of liar?"

Recursion

- Base case
  - tells us when to stop
- Recursive step
  - calls the function with a *smaller version* of the same problem

Recursive Example

```
addNums :: [Integer] -> Integer
addNums [] = 0
addNums (x:xs) = x + addNums xs
```

Lab: parts 1 & 2 (groups of 2-3)

Starter code is available at http://www.cs.sjsu.edu/~austin/cs252-fall17/labs/lab1/. Implement:

1. `maxNum`
2. "fizzbuzz" game

```
> fizzbuzz 15
"1 2 fizz 4 buzz fizz 7 8
fizz buzz 11 fizz 13 14
fizzbuzz"
```

Types

Haskell Types

- `:type` tells you the types for different values.
- `:t` is a shortcut.

```
Prelude>:t 'A'
'A' :: Char
```

'A' is a Char

What is the Type?

```
Prelude>:t "Hello"
"Hello" :: [Char]
```

"Hello" is an array of Chars (i.e. a String)

What is the Type?

```
Prelude>:t head

head :: [a] -> a
```

What is `a`?
Is `a` a type?

`a` is a *type variable;* it stands in place of other types.

What is the Type?

```
Prelude>:t (==)

(==) :: Eq a => a -> a -> Bool
```

This symbol indicates that Eq is a typeclass

`Eq a` indicates that `a` may be any type, provided that it satisfies the expected behavior. (Think of Java interfaces)

Some Typeclasses

- **`Eq`** – Support equality testing
- **`Ord`** – Can be ordered
- **`Show`** – Representable as strings
- **`Read`** – Buildable from a string representation
- **`Enum`** – Sequentially ordered
- **`Bounded`** – Upper and lower bound

JSON example
(in class)

Lab 1, part 3: JSON pretty printer

Download JSON.hs and jsonDriver.hs

In JSON.hs, implement the JObject case in `toString`

HW1: implement a BigNum module

HW1 explores how you might support big numbers in Haskell if it did *not* support them.

- Use a list of 'blocks' of digits, least significant block first. So 9,073,201 is stored as:
  [201,73,9]
- Starter code is available on the course website. **NOTE: YOU MAY NOT CHANGE THE TYPE SIGNATURES.**

Read "Learn you a Haskell"
chapters 4 & 5