EMERGENCY HASKELL CRASH COURSE PART 1

Declarations and expressions

A simple haskell program is just a sequence of **declarations**. A declaration is either:

- A constant declaration `myConst = <exp>`, or
- A function declaration `myFun param1 param2 = <exp>`,

where `<exp>` is an **expression**. An expression is anything that evaluates to a value.

Example:

```
x = 5
add x y = x + y
avg3 a b c = (a + b + c) / 3
```

> *Q:* What's the difference between an expression and a value?
>
> *Q:* In Haskell, a function with 0 parameters is the same as a constant. Is the same true for Python or Java?

Some basic expressions:

- Numbers: `5`, `-5`, `2.0`
- Booleans: `True`, `False`
- Linked lists: `[1, 2, 3]`
- Operators: `True && False`, `10 * (-2) + 3`, `f >>= m`, `1 : [2, 3]`
- String `"hello"`
- Char `'h'`
- Tuples: `(1, 'a', "hello")`
- If-expressions: `if x > 3 then ... else ...`

Example:

```
customGreeting fname lname =
  if fname == "Mary" && lname == "Sue" then
    "Hey Mary!"
  else
    "Hi " ++ fname ++ " " ++ lname
```

Q: In imperative languages (like Python, C(++), Java) `if` is a **statement**. What's the difference between a statement and an expression?

Want to declare local variables? We have expressions for that too:

- `let`-expressions: `let <decls> in <expr>`
- `where`-expressions: `<expr> where <decls>`

Example:

```
customGreeting fname lname nationality =
  let title = if nationality == "France" then "Monsieur" else "Mister"
      fullName = fname ++ " " ++ lname
  in
  "Hi " ++ title ++ " " ++ fullName

-- or equivalently,

customGreeting fname lname nationality =
  "Hi " ++ title ++ " " ++ fullName
  where
    title = if nationality == "France" then "Monsieur" else "Mister"
    fullName = fname ++ " " ++ lname
```

Once a variable is declared, you can't change its value. There's no way to do something like this:

```
def foo():
    x = 1
    if ...:
      x = 5    # set x to 5
    return x
```

However, **variable shadowing** is allowed:

```
foo a =
    let x = 1 in
    if x < a then
      let x = 5 in
      x
    else
      x
```

Q: What does `foo` do?

Q: What is variable shadowing? How does it differ from mutation?

I left off the most important expression of all: **function application**.

In most languages, function application looks like this:

```
<funexp>(<exp1>, <exp2>, ...)
```

In Haskell, we write `<funexp> <exp1> <exp2> ....`

So `f("hello", "world")` becomes `f "hello" "world"`.

Q: How do you write `f(x, g(y,z), h(w))` in Haskell? Use parentheses for grouping.

Q: If you wrote `f x g y z h w`, how do you think the compiler would interpret it?

Q: If you wrote `f(1,2)`, how do you think the compiler would interpret it? (Hint: Haskell supports tuples.)

Q: What's the difference between an argument and a parameter?

Q: Here's Python code for merge sort. Translate it into Haskell.

```
def sort(xs):
  if len(xs) < 2:
    return xs
  half = len(xs) / 2
  first = xs[:half]
  second = xs[half:]
  return merge(sort(first), sort(second))
```

You'll need to use the functions `length`, `take`, `drop` and the `++` operator. You can find what they do in the documentation for the `Data.List` module.

Hello world

So if Haskell doesn't have statements and all functions are pure, how do you print "Hello World"?

You can't.

Instead, you *create a special type of value* containing some commands, and give that value to the runtime.

One way to create this value is using the `print` function. It takes a value to print (e.g. a String or Integer) and produces an `IO ()` value. This result works just like any other value; you can pass it to a function or stick it in a data structure. You can combine two `IO ()` values using the `>>` operator.

If your `main` function returns an `IO ()` value, the Haskell runtime will "run" the value.

Example:

```
main =
  let
    x = print "Hello"
    y = print "World"
  in
    x >> print " " >> y

-- Running the program prints "Hello World"
```

For the next few weeks, we won't use `IO` at all. We'll revisit it in the "monads" lecture.

Q: What's the difference between a language and a compiler? Can a language without exist if it has no compilers? What exactly *is* a language?