Haskell
Lecture 3

Functions Over Lists

List Comprehensions

- *Describe a list in terms of another list*
- *General Form:*
  - *[expression on x | x <- list, tests on x value]*
- *See example comp.hs*

Pattern Matching

Consider the function below

```
alt x y
| x == 0        = y
|otherwise      = x
```

We can rewrite this using pattern matching

```
alt 0 y = y
alt x _ = x
```

Pattern Matching Rules

- *The patterns are examined sequentially*
- *The underscore '_' is a wildcard, use when the value is not to be used*
- *Can be used to name components, as in*
  *f (x, y) = 2*x + 3*y*
- *There is one argument to f above, and it is a tuple*

Patterns and Lists

- *A list can be empty, use the pattern []*
- *A list with a head and a tail, use the pattern (x:xs)*
- *A list with at least 2 elements, and you want to do something with the first 2, use pattern (x:y:ys)*

Writing Patterns for Functions

- *Do not write f x:xs*
  - *The above is the same as (f x):xs*
- *Use parentheses as in f (x:xs)*
- *Examples: mySum and myAnd in pat1.xs*

Case Expressions

Distinguish between various alternatives, revisit alt

```
alt x y
    case x of
        0 -> y
        _ -> x
```

In Class Examples

- *Write own versions of list functions:*
  - *take*
  - *reverse*
  - *concat*
  - *elem*
  - *and do selection sort*

Primitive and General List Recursion

- *Primitive, or tail, list recursion*
  - *base: what to do with empty list*
  - *recursion: given (x:xs), if you know answer from xs, how to add x to answer*
- *General recursion: multiple calls to recursive function at a step*
  - *Example: do quickSort*

String functions

- *A String is a list of Char*
- *words breaks a String into a list of Strings (words) separated from each other by white space in original String. unwords is the reverse*
- *lines and unlines is the same as words and unwords except that '\n' is the separator*