

## Comp 205: Comparative Programming Languages

Functional Programming Languages:

### More Haskell

Lecture notes, exercises, etc., can be found at:  
[www.csc.liv.ac.uk/~grant/Teaching/COMP205/](http://www.csc.liv.ac.uk/~grant/Teaching/COMP205/)

## Tuple Types

Haskell allows composite types to be built up by "tupling", e.g.

- `(Integer,Integer)` is the type of pairs of `Integer`s; similarly
- `(Integer,Char,Bool)`, etc.

In general, `(A,B)` represents the type of pairs whose first component has type `A` and whose second component has type `B`.

## Tuples

Elements of tuple types are also written using the `(_,_)` notation; for example, `(24,'a')` is of type `(Integer,Char)`.

The `(_,_)` notation is an example of a special kind of operator called a **constructor**:

- all tuples can be built using this operator, and
- the operator is not evaluated:

```
Prelude> (24,'a')
(24,'a')
```

## Pattern-Matching #2

Constructors can be used in patterns:

```
-- add a pair of numbers
add :: (Integer,Integer) -> Integer
add (x,y) = x + y
```

```
fst (x,y) = x
snd (x,y) = y
```

## Hot Stuff: Currying

```
curriedAdd :: Integer -> Integer -> Integer
curriedAdd m n = m + n
```

```
Prelude> :l arithmetic
...
Main> :t curriedAdd
curriedAdd :: Integer -> Integer -> Integer
Main> :t curriedAdd 3
curriedAdd 3 :: Integer -> Integer
Main> curriedAdd 3 5
8
```

## Currying

There is a one-to-one correspondence between the types `(A,B) -> C` and `A -> (B -> C)`.

Given a function `f :: (A,B) -> C`, its curried equivalent is the function

```
curriedF :: A -> B -> C
curriedF a b = f (a,b)
```

## Curried Max

```
maxOf2 :: Integer -> Integer -> Integer
```

A possible definition of `maxOf2` is:

```
maxOf2 m n
| m >= n = m
| m < n  = n
```

## Or...

Another possible (equivalent) definition of `maxOf2` is:

```
maxOf2 m n
| m >= n = m
| otherwise = n
```

## Or Even...

Another possible (equivalent) definition of `maxOf2` is:

```
maxOf2 m n
| m >= n = m
| m <= n = n
```

## Nested Definitions

"Local" definitions can be made using the `where` keyword:

```
maxOf3 :: Int -> Int -> Int -> Int
maxOf3 x y z = maxOf2 u z
               where
                 u = maxOf2 x y
```

which is equivalent to:

```
maxOf3 x y z = maxOf2 (maxOf2 x y) z
```

## The Fibonacci Sequence

```
-- nth number in the Fibonacci sequence
fib n = fib1
      where
        (fib1, fib2) = fibs n

-- (nth, (n+1)th) in Fib. seq.
fibs 0 = (1,1)
fibs n = (f2, f1 + f2)
        where
          (f1,f2) = fibs (n-1)
```

## ... Or

```
fib n = fib1
      where
        (fib1, fib2) = fibs n

fibs n
| n <= 0 = (1,1)
| n > 0  = (f2, f1 + f2)
          where
            (f1,f2) = fibs (n-1)
```

## The $F$ sequence where $F$ = Fibonacci

```
-- file: fib.hs

fib n = fib1
  where
    (fib1, fib2) = fibs n
    fibs m
      | m < 1  = (1,1)
      | 0 < m  = (f2, f1 + f2)
      where
        (f1,f2) = fibs (m-1),
```

## The $F$ sequence where $F$ = Fibonacci

```
-- file: fib.hs

fib n = f1
  where
    (f1, f2) = fibs n
    fibs n
      | n < 1  = (1,1)
      | 0 < n  = (f2, f1 + f2)
      where
        (f1,f2) = fibs (n-1),
```

## Local is Hidden

```
Main> :l fib.hs
...
Main> fibs 5

ERROR - Undefined variable "fibs"
Main>
```

## Summary

Key topics:

- Tuples and Currying
- Guards (`| <Test> = ..`)
- Local definitions (`where`)

*Next: Lists*