Curried Functions

Curry

A Tasty dish?

Haskell Curry!

Curried Functions

- Currying is a functional programming tech-nique that takes a function of N arguments and produces a related one where some of the arguments are fixed

A tasty dish?

- Currying was named after the Mathematical logician Haskell Curry (1900-1982)
- Curry worked on combinatory logic …
- A technique that eliminates the need for variables in mathematical logic …
- and hence computer programming!
  - At least in theory
- The functional programming language Haskell is also named in honor of Haskell Curry

Functions in Haskell

- In Haskell we can define *g* as a function that takes two arguments of types *a* and *b* and returns a value of type *c* like this:
  - g :: (a, b) -> c
- We can let *f* be the curried form of *g* by
  - f = curry g
- The function f now has the signature
  - f :: a -> b -> c
- *f* takes an arg of type *a* & returns a function that takes an arg of type *b* & returns a value of type *c*

Functions in Haskell

- *All* functions in Haskell are curried, i.e., *all Haskell functions take just single arguments.*
- This is mostly hidden in notation, and is not apparent to a new Haskeller
- Let's take the function div :: Int -> Int -> Int which performs integer division
- The expression div 11 2 evaluates to 5
- But it's a two-part process
  - div 11 is evaled & *returns a function* of type Int -> Int
  - That function is applied to the value 2, yielding 5

Curried functions

- Functions with multiple arguments are possible by returning functions as results:

```
add'    :: Int → (Int → Int)
add' x y = x+y
```

- **add'** takes an integer **x** and returns a function **add' x**. in turn, this function takes an integer **y** and returns the result **x + y.**

Note:

- add and add' produce the same final result, but add takes its two arguments at the same time, whereas add' takes them one at a time:

```
add  :: (Int,Int) → Int

add' :: Int → (Int → Int)
```

- Functions that take their arguments one at a time are called curried functions, celebrating the work of Haskell Curry on such functions.

Curried functions with multiple argument

- Functions with more than two arguments can be curried by returning nested functions:

```
mult      :: Int → (Int → (Int → Int))
mult x y z = x*y*z
```

- **mult** takes an integer **x** and returns a function **mult x**, which in turn takes an integer **y** and returns a function **mult x y**, which finally takes and integer **z** and returns the result **x * y *z.**

Why is currying useful?

- Curried functions are more flexible than functions on tuples, because useful functions can often be made by partial applying a curried function.

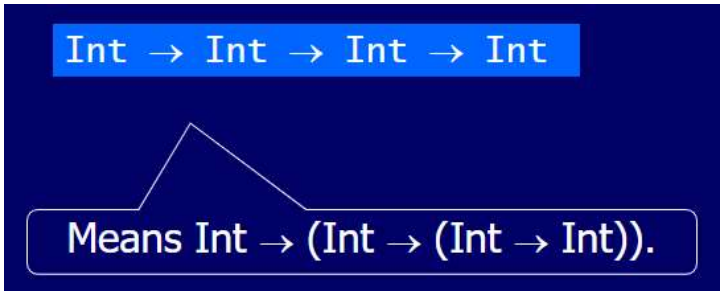
- For example

```
add' 1 :: Int → Int

take 5 :: [Int] → [Int]

drop 5 :: [Int] → [Int]
```

Currying Conventions

- To avoid excess parentheses when using curried functions, two simple conventions are adopted:

1. The arrow → associated to the right.



2. As a consequence, it is natural for function application to associate to the **left**.