1 Quick Sample!

Might as well start with factorial up top as a sample.

$fact\ 1 = 1$
$fact\ n = n * fact\ (n - 1)$

2 On with the first chapter!

Ok, so preliminaries. Order of precedence is as you would expect, except function application come before anything else and doesn't require parens.

$f\ 10$ -- means apply f to 10
$f\ 10 + 2$ -- is f of 10, then add 2
$f\ (10 + 2)$ -- is f of 12.
$f\ 10 + 2 \not\equiv f\ (10 + 2)$ -- because function application is highest precedence

Be careful with negative numbers! You may want to surround them with parentheses to play it safe. $f\ (-3)$

Boolean Algebra is simple:

$(True, False)$ -- are defined and work as expected in Haskell.
$\wedge$ -- `&&` is for conjunction.
$\vee$ -- `||` is for disjunction.
$\neg$ -- `not` is for negation.

$True \wedge False \equiv False$
$False \vee True \equiv True$

Equality testing is `==` (AKA: $\equiv$) and `/-` (AKA: $\not\equiv$) It is a common mistake to forget `/=` is not equals. Also equality must be between items of the same type! Get accustomed to strong typing. Learn to *love* it.

Let's list some basic functions in Prelude, which is imported by GHCI on startup and you can generally count on being there in your programs (unless you specifically choose not to import it, which you will know how to do if you ever need to make that decision):

$succ\ x$ -- is the successor function on x.
$min\ x\ y$ -- returns the smaller of x and y.
$max\ x\ y$ -- returns the larger of x and y.

Since function precedence is highest, these formulae are equivalent:

$succ\ 9 + max\ 5\ 4 + 1$
$(succ\ 9) + (max\ 5\ 4) + 1$

The backtick ‘ is in the upper right corner of your keyboard, above the squiqley tilde ~ sign. We can make any function infix with backticks ‘

$10\ `max`\ 9 \equiv max\ 10\ 9$
$92\ `div`\ 10 \equiv div\ 92\ 10$

3 Baby's First Functions

$doubleMe\ x = x + x$
$doubleUs\ x\ y = x * 2 + y * 2$
$doubleSmallNumber\ x = \textbf{if}\ x > 100$
  $\textbf{then}\ x$
  $\textbf{else}\ x * 2$
$doubleSmallNumber'\ x = (\textbf{if}\ x > 100\ \textbf{then}\ x\ \textbf{else}\ x * 2) + 1$

Remember in Haskell's `if` statement the `else` is mandatory! The expression must evaluate to something of the appropriate type.

’ (apostrophe) is valid in function names, which is pretty cool.

$conanO'Brien =$ `"It's a-me, Conan O'Brien!"`

4 An Intro to Lists

Haskell lists are homogenous data structures. They store several elements of the same type.

$lostNumbers = [4, 8, 15, 16, 23, 43]$

Just so you know, if we had typed the previous in GHCI we would need `let`

$\textbf{let}\ lostNumbers' = [4, 8, 15, 16, 23, 43]$

Lists are concatenated with the `++` (AKA: $+\!\!+$) operator.

$[1, 2, 3, 4] +\!\!+ lostNumbers$
$[1, 2, 3, 4] +\!\!+ [5, 6, 7, 8]$
`"hello"` $+\!\!+$ `" "` $+\!\!+$ `"world"` $\equiv$ `"hello world"`

Yes, strings are just lists (of type *Char*)

$+\!\!+$ is a fairly expensive operation. The whole list must be walked through.
: on the other hand (cons) is cheap. Just tack something on the front.

`'A' : " SMALL CAT"` $\equiv$ `"A SMALL CAT"`
$5 : [1, 2, 3, 4] \equiv [5, 1, 2, 3, 4]$

Actually, [1,2,3] is syntactic sugar for 1:2:3:[]
`!!` is the index operator.

`"Joe" !! 2` $\equiv$ `'e'`

Lists are indexed from 0, like anyone sane would expect. Lists can be nested (contain other lists). They can be different lengths but not different types.

Lists are compared using $<$,$>$,$\leqslant$,$\geqslant$,$\equiv$,$\not\equiv$ in lexicographic order.

$[3, 2, 1] > [1, 2, 3] \equiv \mathit{True}$

Useful list operators: *head*, *tail*, *last*, *init*, *length*, *null*, *reverse*, *maximum*, *minimum*, *sum*, *product*, *elem x xs* (AKA: `x 'elem' xs` or $x \in xs$)

init $[1, 2, 3] \equiv [1, 2]$
null $[\,] \equiv \mathit{True}$

`take x xs` (where x is an Int, takes x elements from xs)
`drop x xs` (where x is an Int, drops the first x elements from xs)

take 3 $[1, 2, 3, 4, 5] \equiv [1, 2, 3]$
drop 3 $[1, 2, 3, 4, 5] \equiv [4, 5]$

`..` is the range operator and is mega-useful.

$[1\,..\,10] \equiv [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]$
`['a'..'z']` $\equiv$ -- the list of the lowercase alphabet –

There is a special form with step values.

$[\mathit{FIRST}, \mathit{SECOND}\,..\,\mathit{LAST}]$

$[2, 4\,..\,20] \equiv [2, 4, 6, 8, 10, 12, 14, 16, 18, 20]$
$[3, 6\,..\,20] \equiv [3, 6, 9, 12, 15, 18]$
$[4, 10\,..\,20] \equiv [4, 10, 16]$ -- step by 6
$[20, 19\,..\,10] \equiv [20, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10]$ -- step is necessary here!

Infinite lists are both fine and really cool.

$[1\,..]$ -- is the natural numbers (if we begin at 1) or positive integers
$[0\,..]$ -- is the natural numbers if you consider 0 natural

$[13, 26..]$ -- is the postive multiples of 13
take 24 $[13, 26..]$ -- is the first 24 positive multiples of 13.

`cycle` takes a list and replicates its elements infinitely

take 10 (*cycle* $[1, 2, 3]$) $\equiv [1, 2, 3, 1, 2, 3, 1, 2, 3, 1]$
take 11 (*cycle* `"LOL "`) $\equiv$ `"LOL LOL LOL"`