

# Haskell – Crib Sheet

## Arithmetic Operators

		<pre>Prelude&gt; 7 + 2 59 Prelude&gt; 3 * 4 12 Prelude&gt; 19 - 18 1 Prelude&gt; 4 / 2 2.0 Prelude&gt; ( 19 - 18 + 1 ) * 2 4</pre>
<code>^</code>	Power	<pre>Prelude&gt; 2 ^ 3 8</pre>
<code>`div`</code>	Integer division	<pre>Prelude&gt; 5 `div` 2 2</pre>
<code>`mod`</code>	Remainder	<pre>Prelude&gt; 5 `mod` 2 1</pre>

## Boolean and Relational Operators

<code>&amp;&amp;</code>	AND operation	<pre>Prelude&gt; True &amp;&amp; False False</pre>
<code>  </code>	OR operation	<pre>Prelude&gt; True    False True</pre>
<code>not</code>	Not operation	<pre>Prelude&gt; not False True</pre>
<code>==</code>	Is equal to	<pre>Prelude&gt; 5 == 4 False</pre>
<code>/=</code>	Is not equal to	<pre>Prelude&gt; 5 /= 2 True</pre>
<code>&gt;=</code>	Greater than or equal to	<pre>Prelude&gt; 5 &gt;= 2 True</pre>
<code>&gt;</code>	Greater than	<pre>Prelude&gt; 5 &gt; 2 True</pre>
<code>&lt;=</code>	less than or equal to	<pre>Prelude&gt; 5 &lt;= 2 False</pre>
<code>&lt;</code>	less than	<pre>Prelude&gt; 5 &lt; 2 False</pre>

## Built-in Functions

sqrt - Square Root	<pre>Prelude&gt; sqrt 25 5.0</pre>
even - If integer is even return True if not return False	<pre>Prelude&gt; even 12 True</pre>
odd - If number is odd return True if not return False	<pre>Prelude&gt; odd 12 False</pre>
round - rounds to nearest integer	<pre>Prelude&gt; round 5.7 6</pre>

min - returns the minimum value if a pair of numbers	<b>Prelude&gt;</b> min 5 6 7 6
max - returns the maximum value of a pair of numbers	<b>Prelude&gt;</b> max 5 6 7 7
succ - returns the (succeeding) next integer	<b>Prelude&gt;</b> succ 12 13

### Create Variables

Integer	<b>Prelude&gt;</b> let a = 10 <b>Prelude&gt;</b> a 10
String	<b>Prelude&gt;</b> let b = "Hello" <b>Prelude&gt;</b> b "Hello"

### Create Lists

Create Integer list	<b>Prelude&gt;</b> let nums = [18,17,16,13,12,19]
Create String list	<b>Prelude&gt;</b> let li = ["a","b","c","d"]
Create Empty list	<b>Prelude&gt;</b> let x = []
As a shorthand, sequences of numbers can be given by a pair of dots (..) between the first and last element	<b>Prelude&gt;</b> let nums = [1..10] <b>Prelude&gt;</b> nums [1,2,3,4,5,6,7,8,9,10]
If the second element is present, this gives the interval for the number sequence	<b>Prelude&gt;</b> let nums = [2,4..12] <b>Prelude&gt;</b> nums [2,4,6,8,10,12]

### List processing

Returns the first element in a list	<b>Prelude&gt;</b> head [3,6,9,1,2] 3 <b>Prelude&gt;</b> head ["a","b","c","d"] "a"
Returns all but the first element in a list	<b>Prelude&gt;</b> tail [3,6,9,1,2] [6,9,1,2]
returns the last element in a list	<b>Prelude&gt;</b> last [3,6,9,1,2] 2
Returns all but the last element in a list	<b>Prelude&gt;</b> init [3,6,9,1,2] [3,6,9,1]
Return the length of a list	<b>Prelude&gt;</b> length [3,6,9,1,2] 5
Check to see if a list is empty	<b>Prelude&gt;</b> let x = [] <b>Prelude&gt;</b> null x True <b>Prelude&gt;</b> let a = [1,2,3,4] <b>Prelude&gt;</b> null a False
Return an element from the list (starting from zero)	<b>Prelude&gt;</b> [3,6,9,1,2] !! 1 6
Select the first 2 elements from a list	<b>Prelude&gt;</b> take 2 [3,6,9,1,2] [3,6]
Remove the first 2 elements from a list	<b>Prelude&gt;</b> drop 2 [3,6,9,1,2] [9,1,2]

Append to a list	<b>Prelude&gt;</b> [1,2,3,4] ++ [5] [1,2,3,4,5]
Prepend to a list	<b>Prelude&gt;</b> [1] ++ [2,3,4,5] [1,2,3,4,5] <b>Prelude&gt;</b> 1:[2,3,4,5] [1,2,3,4,5]
Append 2 lists	<b>Prelude&gt;</b> [3,6,9] ++ [1,2] [3,6,9,1,2]
Reverse a list	<b>Prelude&gt;</b> reverse[3,6,9,1,2] [2,1,9,6,3]
Calculate the product of all values in a list	<b>Prelude&gt;</b> product [3,6,9,1,2] 324
Calculate the sum of all values in a list	<b>Prelude&gt;</b> sum [3,6,9,1,2] 21

### Higher order functions

Higher-order functions are function that can take a function as an argument and/or return a function as a result.

<b>Map</b> applies a function to each element of a list and returns the results in a list. Eg multiply all elements by 2	<b>Prelude&gt;</b> map (*2) [1,2,3,4,5] [2,4,6,8,10]
<b>Filter</b> applies a function to select items from a list. Eg select all values greater than 2	<b>Prelude&gt;</b> filter (>2) [1,2,3,4,5] [3,4,5]
<b>Fold</b> reduces a list to single value using recursion. Eg sum the numbers in a list	<b>Prelude&gt;</b> foldl (+) 6 [1,2,3,4,5] 15

### Functions

<b>Function application</b> - Function applied to its arguments	<b>Prelude&gt;</b> let add x y = x + y <b>Prelude&gt;</b> add 3 4 7
<b>Function Type</b> -Functions have argument data types and an result data types  function :: argument_type -> result_type	double :: Integer -> Integer double x = x * 2
<b>Composition of functions</b> - Functions can be combined to create a new function	<b>Prelude&gt;</b> let f x = 8 * 5 <b>Prelude&gt;</b> let f x = x * 5 <b>Prelude&gt;</b> let g x = 3 + x^2 <b>Prelude&gt;</b> f (g 4) 95 <b>Prelude&gt;</b> g (f 4) 403
<b>Partial function application</b> - When a function is applied to fewer than the total number of parameters.	<b>Prelude&gt;</b> let add x y = x + y <b>Prelude&gt;</b> let add3 = add 3 <b>Prelude&gt;</b> add3 4 7

## First class object

In functional programming languages functions can be first class objects. This mean they can be:

assigned to variables	
Passed as arguments to other functions	<b>Prelude&gt;</b> let double x = x * 2 <b>Prelude&gt;</b> map double [1,2,3] [2,4,6]
returned from other functions	<b>Prelude&gt;</b> let add x y = x + y <b>Prelude&gt;</b> let add3 = add 3
stored in data structures	<b>Prelude&gt;</b> let add x y = x + y <b>Prelude&gt;</b> let a = [add 3 4, 5, 6]