COMP105 Lecture 3

Getting Started with Haskell

ghci

Haskell comes with an **interpreter** called `ghci`

```
ghci> 2 + 15
17
ghci> 49 * 100
4900
ghci> 1892 - 1471
421
ghci> 5 / 2
2.5
ghci> 2^10
1024
ghci> 2**2.5
5.65685
```

Boolean expressions

Haskell uses **C** syntax for and and or

```
ghci> True && False
False
ghci> True && True
True
ghci> False || True
True
ghci> not False
True
ghci> not (True && True)
False
```

Equalities

```
ghci> 5 == 5
True
ghci> 1 == 0
False
ghci> 5 /= 5
False
ghci> 5 /= 4
True
ghci> "hello" == "hello"
True
```

Inequalities

```
ghci> 1 < 2
True
ghci> 1 <= 1
True
ghci> 100 > 101
False
ghci> 10 >= -10
True
```

Brackets

Order of operations is as we expect (BODMAS)

```
ghci> (50 * 100) - 4999
1
ghci> 50 * 100 - 4999
1
ghci> 50 * (100 - 4999)
-244950
```

Make sure to bracket negatives

- `5 * (-3)` rather than `5 * -3`

Evaluating a function

Haskell uses **special** syntax for function calls

```
ghci> min 9 10
9
ghci> min 3.4 3.2
3.2
ghci> max 100 101
101
```

The syntax is [function name][space][arg1][space][arg2]...

Compare to python `f(x, y, z)` becomes `f x y z`

- Commas $\rightarrow$ spaces, brackets $\rightarrow$ nothing

Bracketing of functions

Functions **bind more tightly** than any other operator

(BFODMAS)

```
ghci> max 2 1 + 3
5
ghci> (max 2 1) + 3
5
```

You will need to put brackets **around** arguments

```
ghci> min 28 100/4
7.0
ghci> min 28 (100/4)
25.0
```

Special syntax for two-argument functions

A function with two arguments can be made **infix**

```
ghci> mod 10 4
2
ghci> 10  mod  4
2
```

Here we surround the function by **backticks** (next to the 1 key)

- The function `mod` does the modulo function (% in other languages)

Special syntax for two-argument functions

Or we can take infix operators and make them normal functions by surrounding them by brackets

```
ghci> 1 + 1
2
ghci> (+) 1 1
2
ghci> (*) 49 22
1078
```

May seem useless now, but we will use this quite a bit when we talk about **higher order functions**

- Also, remember that everything is a function!

Exercise

The function `succ` adds 1 to its input, eg. `succ 4 = 5`

What is the answer for the following Haskell queries?

1. `succ 1 ^ succ 1`
2. `succ 1  min  succ (succ 1)`
3. `max ((/) 10 2) ((*) 2 2)`