COMP105 Lecture 4

Let

Let expressions

Sometimes we want to use the same **expression more than once**

```
(x * x - 4) + sqrt (x * x - 4)
```

Why do we have to write out `x * x - 4` twice?

The solution: use a `let` expression

```
let s = (x * x - 4) in s + sqrt s
```

Let syntax

The syntax for a let expression is:

```
let <bindings> in <expression>
```

Where

- `<bindings>` gives names to some expressions
- `<expression>` uses those bindings

You can bind more than one name in a let expression

```
let a = 1; b = 2 in a + b
```

Lets vs variables

A let expression **does not** create variables

- Think of the bindings as names for particular expressions
- You can't change a binding once it is bound

The following code is an **error**

```
let a = x * x; a = a + 1 in a
```

Let in ghci

Let can be used to bind names in ghci

```
ghci> let a = 1
ghci> let b = 2
ghci> max a b
2
```

Let across multiple lines

Usually it is clearer to write a let across multiple lines

```
f x y = let a = x * x
            b = y * y
        in
            a * a + b * b
```

Here we don't need to use ; to separate the bindings

- Just put each one on its own line

Let examples

```
let a = 100; b = 200; c = 300 in a*b*c
```

```
4 * (let a = 9 in a + 1) + 2
```

```
cylinder r h =
    let sideArea = 2 * pi * r * h
        topArea = pi * r ** 2
    in  sideArea + 2 * topArea
```

Haskell's layout rule

Each definition at the same level should start on **exactly** the column

```
f x y z = let
              a = x * x
              b = y * y
              cc = z * z
          in
              a * a + b * b
```

Watch out for **tabs** and **spaces**!

- I suggest that you only use spaces when coding in Haskell

Haskell's layout rule

Failure to follow the layout rule will give **an error**

Both of these examples will fail because they are not aligned

```
let
    a = x * x
   b = y * y
in
    a + b
```

```
let
    a = x * x
     b = y * y
in
    a + b
```

Ignoring the layout rule

You can ignore the layout rule if you use {} and ;

```
let {a = x * x;
  b = y * y;
      c = z * z}
in
    a + b
```

Exercises

Use `let` to solve the following exercises

1. Write a function `exercise1` that takes one argument `x` and computes `a = x * x + 1` and returns `a * a * a`

2. Write a function `exercise2` that takes one argument `x` and computes `a = x + 1`, `bb = a + 2`, `ccc = a + bb` and returns `ccc*ccc`