Let's finish up functions!

Recall:

Keyword arguments

Default arguments

Today:

Recursion

Recall: keyword arguments

Using the colour LCD screen:

```
rgb_lcd_colour(255, 0, 255)
```

(aside: what colour is this?)

Easier to tell now:

```
rgb_lcd_colour(red=255, green=0, blue=255)
```

No *positional* arguments after the first *keyword* argument

Recall: default arguments

Passed to the parameter if no argument in the call

```
def get_user_input(prompt='Input? '):
    return input(prompt)
```

One way to print:

```
print('these', 'words', 'go', 'on', 'one', 'line')
print('these', 'words', 'go', 'on', 'the', 'next', 'line')
```

Another way:

```
print(1, 2, 3, sep='*', end=' + ')
print(4, 5, 6, sep='*')
```

Recursion

A function calling itself (??)

```
def factorial(n):
    return n * factorial(n-1)
```

- each **call** has its own variables
- beware of **infinite recursion** (as in the example above!)
- need a **base case**: when do we stop recursing?

There are lots of interesting problems in computing whose solutions can be expressed most elegantly via a recursive function. We won't require you to write a lot of those — that's more for a Data Structures and Algorithms course in Term 4 — but you do need to be at least somewhat familiar with the concept of recursion. It's an elegant tool, but like a lot of interesting concepts, it has subtleties to be aware of.

Like a loop with a perpetually-true condition, recursion can lead to a program that never stops running (at least until it runs out of memory for all of the function calls' memory!).

A _____ for recursion is when the recursion _____. In the example of a factorial, the factorial is actually defined in two parts:

$$n! = n \times (n-1)! \mid n > 0$$

$$0! = 1$$

The base case for our factorial function, therefore, is that _____`n factorial`_____.

Problems

Let's work on some problems!

Summary

More fun with functions:

- Special arguments:
  - keyword arguments
  - default arguments
- Recursion

Next: modules!