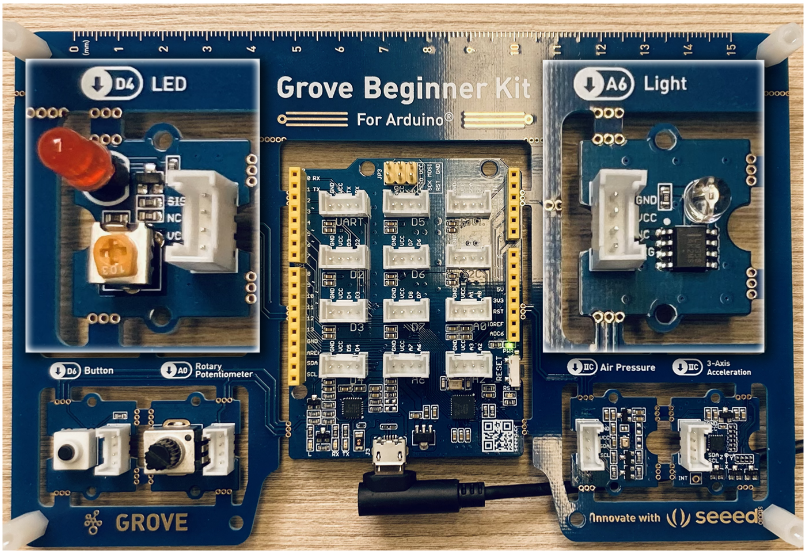
Suppose we want to create a warning light for a photographic darkroom that indicates when there is too much light present (not very much!).



We have this board with a light sensor and a red LED that might be suitable for use in a darkroom. Could we build a warning sensor for this application?

Recent pain

A lab-inspired example:

- read
- write
- repeat

Slightly less pain

A script we can run repeatedly:

```python
from engi1020.arduino.api import *

# Read the current light level (port A6)
light = analog_read(6)

# Turn the LED (port D4) if light is bright
if light > 400:
    digital_write(4, True)
else:
    digital_write(4, False)
```

... but there must be a better way!

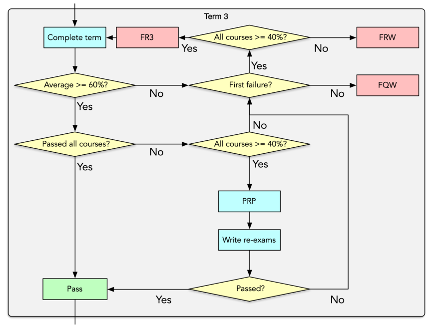
We could simplify this `if` statement as follows:

```python
digital_write(4, light > 400)
```

Remember flowcharts?

Term 3 promotion:

- different from Engineering One (see exercise 3)
- also not the whole story!

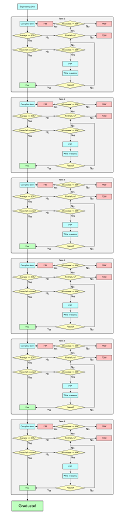


In the previous two lectures, we saw that the *control flow* of a program can be changed through the use of `if` statements. We used the example of a student's progression through Term 3 of an Engineering discipline to show how we can choose to follow one path of control flow or another based on a *condition*. However, there is more to control flow than simply choosing between two (or more) alternative paths!

A bigger flowchart

What's different?

- lots of terms!
- each is essentially the same
- can we describe this process more *abstractly*?



Previously, we considered how a flowchart can be used to model a student's promotion decision in a single term. However, the Engineering program is more than just one term! The flowchart to the right shows something a bit more realistic: a student's progression through all of the academic terms of their Engineering discipline.

Notice that the only difference between each term is *which term it is*. That is, if we could *abstract away* the detail of which term we are currently in, calling it something like term n instead, we could treat every term in exactly the same way. We could then write a description of "how a term works" without having to know which term it is, and then we could *repeat* this procedure for each term from three to eight.

Recall:

Conditional control flow

either do this or else do that, based on a condition

Looping

do this over and over while a condition is satisified

We said that there are two major ways of directing control flow in a program:

- we can choose to execute one bit of code or another, based on some condition (*conditional control flow*) and
- we can execute a bit of code over and over while a condition is met (*looping*).

In this portion of the course, we will see the second form of control flow: *looping* while a condition is met.

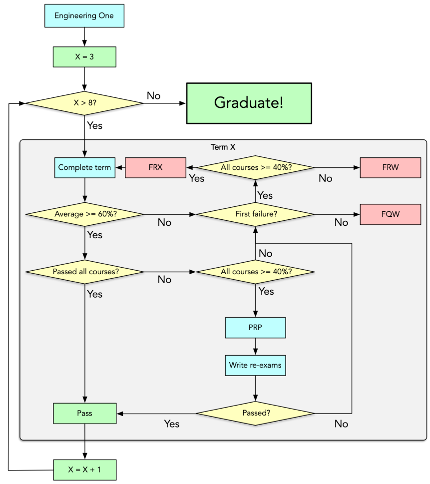
A more compact representation

The same process!

What's different?

- *abstract* term description
- *repeating* control flow

A *loop!*



If we change the description of each term to use the variable n instead of an explicit term number, we can represent the whole program as shown here. Now there is one description of "a term", and we simply repeat it over and over. This is called *looping*, and there are ways of doing this in ______________________________.

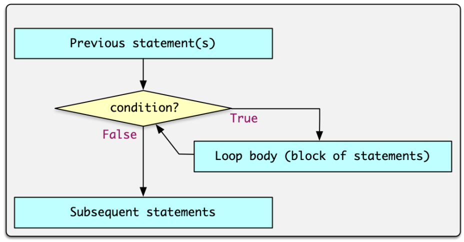
Note, in particular, three key aspects:

1. we **set up** the loop by initializing the value n (start in Term 3),
2. we **check a condition** every time we go through the loop to see if we're done yet and
3. we **execute** the loop and **update the value of** n every time we go through it.

The `while` loop

Around and around...

```
while condition:
    # execute some statements!
    x = an_assignment()
    print(something)
```



- condition: still a Boolean expression
- beware the infinite loop

Example: factorial

Concrete examples:

$$3! = 3 \times 2 \times 1$$

$$5! = 5 \times 4 \times 3 \times 2 \times 1$$

Abstract definition:

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

$$0! = 1$$

These equations are very *concrete*: we can actually evaluate both sides of the equation and check that it's true. However, such concrete statements of truth are not very widely applicable! We would like to have a more *abstract* solution: how can we find the factorial of ____________ positive integer?

This definition has two parts:

1. the *general case*: how to compute a factorial abstractly for most numbers and
2. the *base case*: what to do in a specific, concrete case.

This is also how *proof by induction* works: you prove that something is generally true for n as long as it's true for $n-1$, then you find an example of an n where you can prove it using other means. Then, you've proved your theorem from that value of n up to infinity! These kinds of problems --- with a general case and a base case --- are also very amenable to implementation in a ______________________________________.

Factorial pseudocode

The factorial of n is:

- let f = 1
- as long as $n > 0$:
  - $f = f * n$
  - $n = n - 1$

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

$$0! = 1$$

Can you convert this into Python?

Another example

I'm thinking of a number