1 Introduction

This module will provide expertise for finding diffraction images in a directory.

1.1 Glossary

The following exact terms will be used in this document.

- Template - a string which represents the names for a sequence of diffraction images, like `foo_bar_1_####.img`. The `###` will be replaced with a sequence of three-digit numbers.
- Directory - the location of these files.
- Prefix, Extension - - strings representing the two halfs of the template around `###`. e.g. foo_bar_1 and img above.

2 Use Cases

2.1 UC 1: Derive Template and Directory from Image Name

Action: Derive from a (full path) image name, return a likely template and directory for these files.

Function: image2template_directory(image), image2template(image)

2.2 UC 2: Finding Images from Template and Directory

Action: a template and directory are provided. This directory is searched for files which have a matching name, and the list of matching image numbers returned as a sorted list of integers.

Function: find_matching_images(template, directory)

2.3 UC 3: Constructing Full Path

When provided with a template, directory and image, construct the full path to the image.

Function: template_directory_number2image(template, directory, number)

3 Implementation

3.1 UC 1

This will use the following regular expression to match the image name:

```
(.*)_([0-9]*)\.(.*)
```

which means (whatever) (underscore) (some digits) (dot) (whatever). This will not match files called foo001.img or foo.001 etc. Is this a problem?? Look in to this - could the (underscore) and (dot) be optional? Should be doable. Adding ? after the offending tokens could do it - add this to the unit test. This would make the expression:

```
(.*)_?([0-9]*)\.?(.*)?
```

Oh - this et's stuck on the greediness of things - better off trying to match a number of patterns in sequence and see which works best... Yes, this works though it makes for notty code, including dictionarys of how to put the template back together. I ended up with:

```
def image2template(filename):
    '''Return a template to match this filename.'''

    # the patterns in the order I want to test them

    pattern_keys = [r'(.*)_([0-9]*)\.(.*)',
                    r'([^\.]*)\.([0-9]+)',
                    r'(.*?)([0-9]*)\.(.*)']

    # patterns is a dictionary of possible regular expressions with
    # the format strings to put the file name back together

    patterns = {r'(.*)_([0-9]*)\.(.*)':'%s_%s.%s',
                r'([^\.]*)\.([0-9]+)':'%s.%s%s',
                r'(.*?)([0-9]*)\.(.*)':'%s%s.%s'}

    for pattern in pattern_keys:
        match = re.compile(pattern).match(filename)

        if match:
            prefix = match.group(1)
            number = match.group(2)
            try:
                exten = match.group(3)
            except:
                exten = ''

            for digit in string.digits:
                number = number.replace(digit, '#')

            return patterns[pattern] % (prefix, number, exten)

    raise RuntimeError, 'filename %s not understood as a template' % \
          filename
```

Still - it works!

3.2 UC 2, 3

Implemented.