# Assignment 4 (regex) (30/40 + 5 points)

All solutions should be stored in a directory called `assignment5` in your private repository.

## 5.0: Background info on syntax highlighting (0 points)

When you use a good editor to edit a file with Python code, it will color it to make it easier to read for humans. Generally speaking comments will have a color to make them stand out from the rest of the text, so it is easier for us to separate them from the "active" parts of the program.

```
from my_unit_testing import UnitTest

def better_addition(a, b, num_rechecks=2):
    """Returns sum of a, b, but double checks answer several times."""
    sum_computations = [a + b for n in range(num_rechecks)]

    for n in range(num_rechecks):
        if sum_computations[n] != sum_computations[n-1]:
            print("Hang on, let me recheck that")
            return better_addition(a, b, num_rechecks)

    return sum_computations[0]  # if all computations match, return whichever
```

In the first part of this assignment, you will create a program which does that, and make it extensible to use different color schemes and support different languages. It should read a regex dictionary from a file which in a sense specifies what parts of the language should be colored, and a color theme from another file which specifies what colors should be used.

For examples, take a look at the example files `naython.syntax`, `naython1.theme`, and `naython2.theme` which specifies the syntax of a very simple "programming" language and gives two different color themes to color it by. When your program is used to color `hello.ny`, your output should look like one of the two below, depending on which theme file you used:

```
NNNN Naython is a very simple programming language.
NNNN The only statement is the print statement.
NNNN Comments are started with four Ns.

print(hello)          NNNN prints hello
```

```
NNNN Naython is a very simple programming language.
NNNN The only statement is the print statement.
NNNN Comments are started with four Ns.

print(hello)          NNNN prints hello
```

## 5.1: Syntax highlighting (7 points)

Create a program which takes as input a regex , a color theme and a file and outputs the file with appropriate colors to standard out. Your program should

be callable from the command line as `python3 highlighter.py syntaxfile themefile sourcefile_to_color` in the manner described above.

For more information on how to print in color, see clarifications.

Name of file: `highlighter.py`.

## 5.2: Python syntax (INF3331: 10 points) (INF4331: 15 points)

Create a regex dictionary and color theme for Python. Your files should be usable by your program and follow the format specified in 5.0 - look at the naython examples for guidance. Choose at least 7 (INF4331: at least 10) pieces of Python syntax from the below list and include them in your dictionary. Then create a second color theme. The color themes should be different, but you are free to use whichever colors you think look nicest.

- Comments
- Function definitions
- Class definitions
- Strings
- Imports
- "Special" statements `None, True, False`
- Variable assignments
- Decorators
- Try/except
- for-loops
- while-loops
- if/elif/else blocks
- Something else you feel is missing (be reasonable)

Name of files: `python.syntax`, `python.theme`, `python2.theme`

## 5.3: Syntax for your favorite language (INF4331 only, 5 points)

Create a regex dictionary and color theme for your favorite language.

Name of files: `favorite_language.syntax`, `favorite_language.theme`

## 5.4: Syntax for your second favorite language (INF3331 and INF4331, up to 5 bonus points)

For 5 bonus points, repeat 5.3 for a different language.

Name of files: `favorite_language.syntax`, `favorite_language.theme`

## 5.5: superdiff (10 points)

The standard utility `diff` takes two files as input, and outputs a file containing all changes which have to be made to the first file to make it into the second file. In this assignment, you will create your own implementation of the standard `diff` utility. For convenience, your implementation should take two filenames from the command line, and treat the first as the "original" version of a file, and the second as a modified version.

It should then go through the files line by line, and if a line has not been modified, print it with a `0` in front, if it has been added, print it with a `+` in front, and if it has been deleted, print it with a `-` in front. If a line has been modified, treat it as being a deletion of the original line, and then an addition of the modified line.

Name of file: `my_diff.py`

## 5.6: Coloring diff (3 points)

In this assignment, you will take the output from `my_diff.py` and color it so that additions become green, deletions become red and no-change lines aren't colored in any special way. As you might notice, this is similar to what you did before, so instead of making a new script, make your program as a syntax file and color theme for your syntax highlighter.

Name of files: `diff.syntax`, `diff.theme`

# Clarifications

## 5.0 (general clarifications)

- Your `.syntax` files should have lines of the form `regex:  name` where regex is a quoted string specifying a regex, and name is some arbitrary alphabetical string giving some name to the thing specified by the regex. See `naython.syntax` for an example.

- Your `.theme` files should have lines of the form `name:  color_sequence` where `name` is one of the names specified in the matching `.syntax` file, and `color_sequence` is some bash color sequence. (i.e. something which would be valid if you did `"\033[{}m".format(color_sequence)`

## 5.1

- Information on printing in color can be found in `http://misc.flogisoft.com/bash/tip_colors_and_formatting`, and an example in `coloring_example.py` in the student resources repository.

  Note that you need to be careful with printing in color when two regexes "overlap". For example, if you for example have decided to color Python strings blue and newline characters green, coloring `''Line 1 \n Line 2''` properly means printing first the color sequence for blue, then `''Line 1 ` , then the color sequence for green, then `\n`, then the color sequence for blue, then `Line 2''`. So your program needs to "notice" that it needs to re-insert the blue color sequence in the middle of the string.

## 5.3, 5.4

- Use your best judgment on how much work you should expect to do here. If you make a complete syntax highlighter for whitespace, that is very cute, but you should not expect to get 5 points. On the other hand, if you manage to color a decent (but not all) subset of Java, that is probably enough.

## 5.5

- There is some ambiguity inherent to the problem. For example, if the original has the lines "A", "B" and the modified version has the lines "B", "A", was "A" deleted and later inserted, or was "B" inserted and later deleted? There are a lot of strange edge cases, so don't be too worried about making an implementation which handles absolutely all of them.

- Try to make sure that your program is "reasonable" (i.e. if the original is "A", "B", "C", "D", "E", "F" and the modified file is "B", "C", "D", "E", "F", "G", your program should ideally output that "A" has been

deleted and "F" has been inserted at the end, and not say "everything in the first file was deleted, then everything in the second file was inserted".

### 5.6

- Your syntax and theme should work so that if the output from `my_diff.py` (5.6) is stored in the file `diff_output.txt`, the call `python3 highlighter.py diff.syntax diff.theme diff_output.txt` should produce the desired colored diff.