Peer-review of assignment 5 for *INF3331-peterek*

oleksako, oleksako@uio.no
martimj, martimj@uio.no
andhols, andhols@uio.no

November 13, 2016

1 Review

The assignment was tested on an UiO machine (Red Hat Enterprise Linux Server 6.8) and Python 3.5.2.

General feedback

The code is clean and well structured. The names for functions and variables are very intelligible. All this makes the code easy to read.

What is good is that the program uses argparse to process the input arguments, which makes the processing of arguments very pleasant and simple at the same time. We can advise to have slightly more comments.

Assignment 5.1: Syntax highlightning

The program work as expected with the provided example (hello.py). However, it behaves weird with the examples in the tasks 5.2 and 5.3. All the lines of a similar type either disappear or become substituted with the first line of its type (see on the examples below). This might be a system specific, and behaves differently on a Windows machine. Furthermore, as was explained on piazza forum, the program should be able to process enclosed blocks, which it does not do.

```
#!/usr/bin/env python3
# Comments dissapear or become Comment 1 everywhere
# Comment 2
from typing import List
import argparse
# Comment 3

def equalize(arr1: List, arr2: List, element=" some string in here"):
    if size_orig < size_mod:
        for i in range(size_mod - size_orig):
            arr1 += [element]
    elif size_orig > size_mod:
        for_function():

    return arr1, arr2
# Comment 4 with an enclosed

definition to(test function definitioni):

"""""""

def compare(orig_file: str, mod_file: str):
        """ comparing lines of two files, printing and writing to file.

        """
        while a == True and b == False:
            orig_file = orig_file.split("\n")
            mod_file = mod_file.split("\n")

        orig_file, mod_file = equalize(orig_file, mod_file)
        with open("diff_output.txt", 'a') as out:
                for line_o, line_mod in zip(orig_file, mod_file):
                        if line_o == line_mod:
                                line = "0 " + line_o
                                out.write(line + '\n')
                                print("0", line_o)
                        elif line_o != line_mod:
                                line = "- " + line_o
                                line2 = "+ " + line_mod
                                out.write(line + '\n')
                                out.write(line2 + '\n')
                                print("-", line_o)
                                print("+", line_mod)
```

```
# Comments dissapear or become Comment 1 everywhere
# Comments dissapear or become Comment 1 everywhere
from typing from List
from argparse
# Comments dissapear or become Comment 1 everywhere

def equalize(arr1: List, arr2: List, element=" some string in here"):
    if size_orig < size_mod:
        for i in range(size_mod - size_orig):
            arr1 += [element]
    if size_orig > size_mod:
        for i in range(size_mod - size_orig):

    return arr1, arr2
# Comments dissapear or become Comment 1 everywhere

def equalize(arr1: List, arr2: List, element=" some string in here"):

" some string in here"

def equalize(arr1: List, arr2: List, element=" some string in here"):
        " some string in here"
        while a == True and b == True :
            orig_file = orig_file.split(" some string in here")
            mod_file = mod_file.split(" some string in here")

        orig_file, mod_file = equalize(orig_file, mod_file)
        with open(" some string in here", " some string in here") as out:
                for i in range(size_mod - size_orig):
                        if line_o == line_mod:
                                line = " some string in here" + line_o
                                out.write(line + " some string in here")
                                print(" some string in here", line_o)
                        if line_o != line_mod:
                                line = " some string in here" + line_o
                                line2 = " some string in here" + line_mod
                                out.write(line + " some string in here")
                                out.write(line2 + " some string in here")
                                print(" some string in here", line_o)
                                print(" some string in here", line_o)
```

Assignment 5.2: Python syntax

It was hard to test parts 5.2, 5.3 and 5.4 because of the implementation of "higlighter.py" from the part 5.1, which incorrectly substituted lines. However, these mistakes should not be considered as drawbacks of part 5.2, 5.3 and 5.4.

As for the regex implementation there just one thing to mention. In many places "*" (zero or more) is better to change on "+" (one or more):

```
1       "\b(def\s*.*\(.*\):)": def
```

```
1       "\b(def\s+.*\(.+\):)": def
```

With "zero or more", regex wrongly catches words even if they are just a starting part of other words. For example "definition to(test function definitioni):" will be treated as def. Same applies to the other regex rules.

Assignment 5.3: Syntax for your favorite language

We have assessed files for the programming language C in this section. Similar to the previous part, instead of 'zero or more' it is better to use 'one or more'. Other than that, all looks good.

Assignment 5.4: Syntax for your second favorite language

We have assessed files for the programming language Java in this section. In many regexes are missing the leading 'backslash b'. In all those cases the sought-for words become highlighted even if they occur at the end of bigger words. For example, in word "myclass", "mint", "sherif", the endings "class", "int" and "if" will be colorized. Besides that, all seems to work as expected.

Assignment 5.5: superdiff

As was mentioned at the piazza forum, the best way to implemet the task would be using "Longest common subsequence". Your implementation often mistakenly treat blocks without changes as if each line there was first deleted and then inserted.

Assignment 5.6: Coroling diff

The coloring looks good.