Task documentation SYN: Syntax Highlighting in Python 3 for IPP 2015/2016
Name and surname: Jan Ondruch
Login: xondru14

# IPP Project 2 (SYN) Documentation

## Assignment

The goal of this project was to create a script in a programming language Python 3, that automatically highlights various parts of a text. Data selection for highlighting is specified by a format file. Using additional program argument, it is possible to modify the final output data.

## Program Logic

Following paragraphs describe the main program logic.

### Argument parsing

For argument parsing I used *argparse* module, that saves the input arguments into an object. The module implicitly generates help option and covers most of the forbidden parameters. However, *sys.argv* - list of command-line arguments was used as well to meet all of the parameter's synopsis requirements. A list of input arguments is created and passed for the further processing.

### Format file

Format file comprises of lines, each containing 2 parts. The first one is a regular expression describing a part in the input file to be modified. The second part includes the *HTML codes* standing for *HTML tags*, that after application wrap the selection from the first part. Each line (if not empty) is worked out in a `parse_formatfile()` method and saved as a *FormatLine* class. In this step, partial validity is checked and the two main parts of the lines are separated and stored as object instance attributes.

### Format line

Using `convert_regex()` method, pseudo-regular expressions are converted into a python regular expression. The process of the conversion is executed in a loop iterating character by character of the expression. If successful, compiled regular expression is stored as a `regex` variable.

The main logic of the program revolves around index positions of the expressions, that are to be edited in the input file. Initially, index spans representing a string in the input file that matches the given regular expression are saved into a list `spans`.

```
[(9, 11)]
[(2, 3), (3, 4)]
[(6, 7), (13, 14)]
[(0, 1)]
...
```

Thereafter, the program searches through the input file string data once again, but this time actually wrapping the matched expressions with the corresponding *HTML tag*. Converting the data, `apply_regex()` utilizes 2 dictionaries for *HTML code* to *HTML tag* conversion and for resolution of the tag's length.

As tags are inserted for each span (for one span, there can be many tags), all the other spans have to be recalculated. The difference – number to be added, varies depending on the position of the span relative to the modified one. Tag overlapping is implicitly dealt with by this solution. After all of the spans in every line of the format file are worked out, the whole input file data is modified as `inputFileData`. If `--br` parameter is given, `<br />` tag is inserted at the end of each line. Eventually, the modified input data is outputted.

**Support functionality**

*FormatLine* class implementation requires interconnection of the instances (within the class itself), therefore *weakref* module along with `getinstances(cls)` class method were used to support the functionality.

```
self._instances.add(weakref.ref(self))
```

**Errors**

Errors are handled by try-except mechanism. When an error is found, an exception is raised and the program terminates with a corresponding error code.