

Assignment 3 (python) (15 points)

All solutions should be stored in a directory called `assignment3` in your private repository.

3.1: RPN calculator (4 points)

Make an interactive calculator which takes input in Reverse Polish Notation. This means that the calculator should have an internal stack on which numbers are pushed (added) and popped (removed). When a number is input to the calculator, it is pushed to the end of the stack, and when an arithmetic operation (+, *, /) is input, it should pop the last two numbers of the stack, compute their sum/product/quotient, and push that to the end of the stack.

Further, add options for multiple space-separated inputs on one line, to be parsed left to right. Also add a print-input `p` which prints the last number of the stack without popping it, and `v`, `sin`, `cos` for square root, sine and cosine functions. Finally, add the option for your script to be called with a string as command line input, in which case it should treat the string as a space-separated list of inputs.

Example usage:

```
$ python rp.py
> 1
> 2
> +
> p
3
> 5 2
> * + p
13
```

Name of file: `rp.py`

Hint: For making the calculator interactive, look into the `input` and/or `raw_input` methods.

3.2: wc (3 points)

Make a Python implementation of the standard utility `wc` which counts the words of a file. When called with a file name as command line argument, print the single line `a b c fn` where `a` is the number of lines in the file, `b` the number of words, `c` the number of characters, and `fn` the filename.

Further, extend your script so that it can be called as `wc *` to print a nice list of word counts for all files in the current directory, or `wc *.py` to print a nice list of word counts for all python scripts in the current directory.

Name of file: `wc.py`

3.3: Simple unit testing (5 points)

Write a simple unit testing framework. Specifically, take the `UnitTest` class stub from `unit_testing.py`, and implement `__init__` and `__call__` methods so that the example script `addition_testing.py` testing a silly reimplementaion of addition runs with the output “2/3 tests passed”. Write a (very) brief report of what test is failing, and explain what is wrong with `better_addition`.

Name of files: `my_unit_testing.py` (script), `report.txt` (report) If you do problem 3.4, feel free to store the script in the `my_unit_testing` directory.

3.4: Your first module (3 points)

Make a Python module of your solution of problem 3.3. (If you were unable to complete this, choose any of the other two problems and make a module of one of them instead.) Add a `setup.py` -file so it is easy to install for users. Name your module `my_unit_testing` so that the call `from my_unit_testing import UnitTest` works (even when not in the same directory as the `my_unit_testing.py` file). Make sure to properly document your code so that a user can get help while using it using e.g. docstrings.

Store your module in the directory `assignment3/my_unit_testing`.