Worksheet

To make work easier, write python modules that can be executed from the command line after saving them. Here I can only comment on how to do that in Linux, and I think it would be similar if not identical on MacOS, but should I learn from any of you how to do that in Windows I will include this in future revisions.

For me, such executable scripts will start with (Fedora 21):

```
#! /usr/bin/env python3
```

You need to make the file (e.g., `test.py`) executable to have it run as

```
script.py parameter1 parameter2 ...
```

or call, e.g.,

```
python3 script.py parameter1 parameter2 ...
```

Use detailed comments in your code. Also use "docstrings" for every function, class, and method. Proper documentation is an essential part of programming and python has been designed to help you write code that can be read later as well. Do read the guidelines on naming conventions for functions, classes, and methods, as well as for "docstrings." Use the modern `str.format` for formatting strings and output, and use context managers ("`with`" statement) to open files and close them automatically.

Executable python scripts will at the end have a section that starts with:

```
if __name__ == "__main__":
    ...
```

For development, however, I recommend to use IPython; only when all works, transform the script an executable version. Very useful is use of automatic reloading of modified modules. (IPython extension autoreload.) An exmple on how to set this up is given at `https://www.reddit.com/r/Python/comments/rsfsi/tutorial_spend_30_seconds_setting_up/`.

1. **Bare Python 3 Examples.**

   (a) **First Try at Shell Scrips**
   Write a code/script that takes as input two file names as the first two parameters, the first being a plain ASCII text file, and then as the third and fourth parameter two words, the script should replace all occurrences of the first word by the second word *and vice versa* then write the result to the second file name. If the second file already exists, it should be overwritten. As an example, if I were to call from the command line your script named `script.py`

   ```
   script.py infile.txt outfile.txt John Marry
   ```

   it reads `infile.txt`, replaces all occurrences of "John" by "Marry" and vice versa, and then writes the result to `outfile.txt`. It should not replace "marry" by "john" or "Johnson" by "Marryson". The reading of parameters can be done just using `sys.argv`, but more elegant would be to use the `argparse` module. The replacements can be done efficiently using the `re` module.

   (b) **Classes - Object Oriented Programming**
   Nuclei can be specified by a charge number Z and a mass number A. The number of neutrons is then given by $N = A - Z$. None of the number can be negative, and for $Z = 0$ only the neutron exists with $A = 1$. Obviously there can be many combinations of A and Z that do not exist in nature, but we will neglect this for now. You should write a class, "`Nucleus`," that represents these nuclei. The class is initialized by a string as parameter to the constructor, e.g.,

   ```
   nucleus = Nucleus('C12')
   ```

   where "C" stands for carbon ($Z = 6$). Your code should know about nuclei up to $Z = 20$ and represent the charge number internally as an integer number. You class should implement the `__str__` and `__repr__` functions to make nice output, e.g.,

```
>>> print(nucleus)
C12
>>> repr(nucleus)
"Nucleus('C12')"
```

Next, you class should implement the ability to add two nuclei together, using the `__add__` method, simulating nuclear reactions

```
>>> print(Nucleus('C12') + Nucleus('He4'))
O16
```

Finally, the script should allow to add ions from the command line and print the result:

```
script.py O16 H3
F19
```

2. **Numpy examples**

   (a) **Sorting, etc.**
   Write a code that reads in a column of floating point values from a text file. Each line contains one number, starting with the first line. The script should store the numbers in a `numpy.ndarray`, then sort the numbers by decreasing value and print out the sum of every second number starting with the largest. Your script should work equally with odd and even numbers of values. Example

   ```
   script.py datafile.txt
   123688.33
   ```

   (b) **Matrix Multiplication**
   Write a code that reads in two square matrices of arbitrary size from a file into `numpy.ndarray`s, multiples them, and prints out the result and the determinate of the result, separated by a blank line in exactly the format shown below. In the input file, each row goes into one line with the numbers separated by one or more spaces, and has a blank line between the two matrices. Your script should take the file name as parameter

   ```
   script.py matrixfile.txt
    3.87323606e+07     5.45177699e+08     6.73842193e+07
   -1.23476996e+14     5.05287360e+11    -2.71622025e+10
    1.31288962e+07     1.05936364e+11     5.33529626e+05

   -8.81286289499e+32
   ```

3. **Matplotlib Examples**

   The scripts should not end until the graph is closed or the enter button is pressed on the keyboard - so we can actually see the plot.

   (a) **Counting and Statistics**
   Write a script that reads in a text file and the produces a bar plot using `matplotlib` with the letter on the x-axis in alphabetical order and the total count for each letter in the file for the bar height. Do not distinguish between upper and lower case characters. Also plot letters with "zero" count. Ignore all non-letters. Label the x-axis "letter" and the y-axis "count".

   ```
   script.py mytextfile.txt
   ```

   (b) **Rubik's Cube**
   All transformation of the Rubik's cube can, e.g., be parametrized by a set of generators that are a counter-clockwise turn for each face. Let's call them **d**own, **u**p, **f**ront, **b**ack, **l**eft, and **r**ight. Write a class that represents the Rubik's cube. It is generated in the completely solved state (adopt the face colours and layout of the original Rubik's Cube). The class then has a method `transform` that takes the letter (in form of a string) as parameter and does the requested transformation. The class should also have a method that makes a 3D plot of the cube in its current state. Using this class, write a script that takes an input string of the letters, performs the corresponding transformations and then plots the resulting state of the cube using `matplotlib`.

   ```
   script.py bdullrf
   ```

Example Input Files

1. (a)
```
John and Marry Johnson have a house.
Marry and John did marry in June.
Marry's mother is John's boss.
This works out great for John.
```

2. (a)
```
1.234e5
234.33
12
134.667
-123.477e7
12000
23.45
42
```

   (b)
```
1.234e5 234.33 12
134.667 -123.477e7 12000
23.45 123.233 2333.4

123.9484 3. 546.
1.e5 32. 22
344 454.e5 222
```

3. (a)
```
John and Marry Johnson have a house.
Marry likes the house.
John works in the garden.
Marry's mother lives in the house as well.
```