ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Programming Techniques for Scientific Simulations Exercise 13

HS 15
Prof. M. Troyer

Problem 13.1 Linear algebra with Python

This exercise should help you get acquainted with the NumPy[1] module, which is essential for doing numerical calculations in Python.
Your task is to repeat the calculations done in exercise 11.1 (eigenvalues of the harmonic chain) using Python. You can use the following resources to help you along:

- **NumPy Tutorial:**
  `https://docs.scipy.org/doc/numpy-dev/user/quickstart.html`
- **Array creation routines:**
  `http://docs.scipy.org/doc/numpy/reference/routines.array-creation.html`
- **Linear algebra:**
  `http://docs.scipy.org/doc/numpy/reference/routines.linalg.html`

Problem 13.2 Preparation: Python Modules

Modularising code will be useful when writing simulations in Python. With the following directory structure:

```
src/
    main.py
    hello.py
    include/
            world.py
```

You can include your own Python libraries via relative paths (assuming an object named *world* exists in `world.py`) just like the system-provided modules in `main.py`:

```
import hello
from include.world import world
```

Note: older Python versions require *namespace packages* to have an empty file named `__init__.py` in and leading up to any directory you wish to include modules from.

You can read a more detailed description of modules in the official documentation:
`https://docs.python.org/3/tutorial/modules.html`

Problem 13.3 PyPenna

Write a simulation of the Penna model in Python.
Create an extensible directory structure parallel to your C++ implementation and write the simulation into `main.py`. Design and import a Fish class from `penna/fish.py`.
The code should produce results similar to the C++ version, but the output doesn't have to be identical due to differences in the RNG libraries.

[1] `http://www.numpy.org/`

Problem 13.4 Python Golf Challenge (optional)

Let's have a game of golf. The task is to replicate the following output exactly, using as little code as possible:

- First, print the alphabet in capitalised and non-capitalised form adjacently:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz
```

- Next, your program should print the letters interleaved:

```
AaBbCcDdEeFfGgHhIiJjKkLlMmNnOoPpQqRrSsTtUuVvWwXxYyZz
```

- And again, but wrapped at every 10 characters, in a box (with spaces on both sides of each character):

```
| A a B b C c D d E e |
| F f G g H h I i J j |
| K k L l M m N n O o |
| P p Q q R r S s T t |
| U u V v W w X x Y y |
| Z z                 |
```

- Now we want to annotate the alphabet in a Python list at four specific points:

```
['a: ', 'apple']
['b']
['c']
['d']
['e']
['f']
['g: ', 'grape']
['h']
['i']
['j']
['k']
['l: ', 'lemon']
['m']
['n']
['o: ', 'olive']
['p']
['q']
['r']
['s']
['t']
['u']
['v']
['w']
['x']
['y']
['z']
```

- And finally we only want to see the relevant parts as a string:

```
apple begins with a, grape begins with g, lemon begins with l, olive begins with o
```

Write a correct Python 3 implementation in a single file **without** using the `import` statement, executing any external code or reading from a file. The output should be **exactly** the same as in the `golf_reference.txt` file that is provided.

The goal of the challenge is to implement this using fewer than 450 characters (including newlines). This is equivalent to having a source file of less than 450 bytes in size. The submission with the smallest size (in bytes) wins the challenge. To check the size of your source file, you can use the *wordcount* command line utility (`wc -c filename`).

Hint: You can write the body of a `for`, `while`, `if`, etc. on the same line. Also, both newlines and semicolons are valid delimiters for expressions. However, make sure **never** to use this kind of syntax in a serious code.