Week 2: Computing in python I, Week's Recap and Wrap up

MSc/MRes CMEE 2014-15

Samraat Pawar

Imperial College
London

October 17, 2014

RECAP OF TOPICS

- `python` data structures – lists: `[ ]`, tuples: `( )`, dictionaries: { }, `scipy` (or `numpy`) arrays: `scipy.array([])`

RECAP OF TOPICS

- `python` data structures – lists: `[ ]`, tuples: `( )`, dictionaries: { }, `scipy` (or `numpy`) arrays: `scipy.array([])`
- `ipython`

RECAP OF TOPICS

- `python` data structures – lists: `[ ]`, tuples: `( )`, dictionaries: { }, `scipy` (or `numpy`) arrays: `scipy.array([])`
- `ipython`
- Writing `python` functions (modules) and programs `http://www.diveintopython.net/getting_to_know_python/testing_modules.html`

RECAP OF TOPICS

- `python` data structures – lists: `[ ]`, tuples: `( )`, dictionaries: { }, `scipy` (or `numpy`) arrays: `scipy.array([])`
- `ipython`
- Writing `python` functions (modules) and programs `http://www.diveintopython.net/getting_to_know_python/testing_modules.html`
- Control flow (`if`, `while`, `for`)

RECAP OF TOPICS

- `python` data structures – lists: `[ ]`, tuples: `( )`], dictionaries: { }, `scipy` (or `numpy`) arrays: `scipy.array([])`)
- `ipython`
- Writing `python` functions (modules) and programs `http://www.diveintopython.net/getting_to_know_python/testing_modules.html`
- Control flow (`if`, `while`, `for`)
- Unit testing, Debugging, Profiling

RECAP OF TOPICS

- `python` data structures – lists: `[ ]`, tuples: `( )`, dictionaries: { }, `scipy` (or `numpy`) arrays: `scipy.array([])`
- `ipython`
- Writing `python` functions (modules) and programs `http://www.diveintopython.net/getting_to_know_python/testing_modules.html`
- Control flow (`if`, `while`, `for`)
- Unit testing, Debugging, Profiling
- Regex

RECAP OF TOPICS

- `python` data structures – lists: `[ ]`, tuples: `( )`, dictionaries: { }, `scipy` (or `numpy`) arrays: `scipy.array([])`
- `ipython`
- Writing `python` functions (modules) and programs `http://www.diveintopython.net/getting_to_know_python/testing_modules.html`
- Control flow (`if`, `while`, `for`)
- Unit testing, Debugging, Profiling
- Regex
- Running other applications from `python` (`subprocess`) – building a reproducible workflow

RECAP OF TOPICS

- `python` data structures – lists: `[ ]`, tuples: `( )`, dictionaries: { }, `scipy` (or `numpy`) arrays: `scipy.array([])`
- `ipython`
- Writing `python` functions (modules) and programs `http://www.diveintopython.net/getting_to_know_python/testing_modules.html`
- Control flow (`if`, `while`, `for`)
- Unit testing, Debugging, Profiling
- Regex
- Running other applications from `python` (`subprocess`) – building a reproducible workflow
- Numerical computation in `python` (`scipy`, `matplotlib`)

RECAP OF TOPICS

- `python` data structures – lists: `[ ]`, tuples: `( )`], dictionaries: { }, `scipy` (or `numpy`) arrays: `scipy.array([])`)
- `ipython`
- Writing `python` functions (modules) and programs `http://www.diveintopython.net/getting_to_know_python/testing_modules.html`
- Control flow (`if`, `while`, `for`)
- Unit testing, Debugging, Profiling
- Regex
- Running other applications from `python` (`subprocess`) – building a reproducible workflow
- Numerical computation in `python` (`scipy`, `matplotlib`)
- Over the next few weeks, go through `http://www.diveintopython.net` (Very accessible!)

PRACTICAL 0

1. Review and make sure you can run all the commands, code fragments, and functions we have covered today and get the expected outputs
2. Run `boilerplate.py` and `control_flow.py` from the terminal (try both python and ipython)
3. Run `boilerplate.py` and `control_flow.py` from within the python and ipython shells
4. Open and complete the tasks/exercises in `tuple.py`, `lc1.py`, `lc2.py`, `dictionary.py`
5. Keep all code files organized in `CMEECourseWork/Week2/Code`
6. Version control all your work; the updated bitbucket repository should contain: `boilerplate.py`, `control_flow.py`, `lc1.py`, `lc2.py`, `dictionary.py`, `tuple.py`

PRACTICAL 1, I

- Test and bring under version control: `basic_io.py`, `basic_csv.py`, `testout.csv`, `bodymass.csv`, `tesp.p`, `test_control_flow.py`, `debugme.py`, `profileme.py`

PRACTICAL 1, II

- Then:
  1. Open and run the code `test_oaks.py` — there's a bug, for no oaks are being found! (where's `TestOaksData.csv`?)
  2. Fix the bug (hint: `import ipdb; ipdb.set_trace()`)
  3. Now, write doctests to make sure that, bug or no bug, your `is_an_oak` function is working as expected (hint: `>>> is_an_oak('Fagus sylvatica')` should return `False`)
  4. If you write a good doctest, you will note that you found another error that you might not have just by debugging (hint: what happens if you try the doctest with 'Quercuss' instead of 'Quercus'?)
  5. How would you fix the new error you found using the doctest?

PRACTICAL 2.1, I

- Test and bring under version control: `scope.py`, `profileme.py`, `regexs.py`

PRACTICAL 2.1, II

- Then:
  - **Objective**: Align two DNA sequences such that they are as similar as possible
  - Start with the longest string and try to position the shorter string in all possible positions
  - For each position, count a “score” : number of bases matched perfectly over the number of bases attempted

PRACTICAL 2.1, III

- Your tasks:
  1. Open and run `../../Practicals/Code/Python/align_seqs.py` and make sure you understand what each line is doing (hint: you can use `pdb` to do this)
  2. Convert `align_seqs.py` to a Python function that takes the DNA sequences as an input from a `.csv` file and saves the best alignment to along with corresponding score in a single text file (hint: remember `pickle`!)
  3. Modify your `align_seqs.py` function such that in alignments with tied number of matches, one with highest proportion of matched pairs is returned. E.g., if alignment 1 matches 5 out of 9 bases, and alignment 2 matches 5 out of 6 bases, alignment 2 is returned.
  4. Make sure you provide a test `.csv` file that the script can call
  5. Extra Credit? Align the `.fasta` sequences from Week 1!

PRACTICAL 2.2, I

- Test and bring under version control: `LV1.py`

PRACTICAL 2.2, II

- Then, convert `LV1.py` into another script called `LV2.py` that does the following:
  1. Take arguments for the four LV model parameters `r, a, m ,e` from the commandline

     ```
     LV2.py arg1 arg2 ... etc
     ```

  2. Runs the Lotka-Volterra model with prey density dependence $rx(1 - \frac{x}{K})$
  3. Saves the plot as `.pdf` in an external results directory (`Week2/Results`)
  4. The chosen parameter values should show in the plot (e.g., $r = 1, a = .5$, etc)

- You change time length `t` too

- Also include a script called `run_LV2.py` in `Code` that will run `LV2.py` with appropriate arguments

PRACTICAL 2.2, III

- Extra credit if you also choose appropriate values for the paramaters such that both predator and prey persist under in model with prey density dependence
- Extra-extra credit if you can write a recursion version of the model in discrete time (what's this?) – it should do everything that `run_LV2.py` does
- Extra-extra-extra credit if you can write a recursion versiaon of the model in discrete time with a random gaussian fluctuation at each time-step (use `scipy.stats`)

PRACTICAL 2.2, IV

- Complete the code *blackbirds.py* that you find in the `CMEEMasteRepo` (necessary data file is also there)

PRACTICAL 2.2, V

- Keep `Code`, `Data`, `Results`, `Sandbox` at same level under `CMEECourseWork/Week2`!