

Syllabus

NBN Advanced Python 2007
Aims & Assessment

Aims

- Pragmatic
- Compressed
- Actual programming
- Exercises & solutions
- Handouts and materials

The aims of class are pragmatic, less concerned with the shibboleths of computer science and more about letting scientists get their work done, quickly and correctly, learning via actual programming experience and solving practical problems. Due to time constraints, the lecture program is compressed more than is optimal. Each session will attempt to cover one practical topic (e.g. a given module or problem space), a shorter section on a more abstract or more general topic (e.g. design, debugging) as well as time for exercises concerning those topics.

All handouts and materials will be available for upload from the course website and from [my own](#).

Four broad topics

- Advanced Python programming
- Software engineering
- Biocomputing relevant libraries
- Other useful modules

Possible topics are listed but suggestions about others are welcome, as well as the order in which we cover them.

Topic 1: Advanced Python programming

- Modules, exceptions, functional programming, iterators, I/O
- Installation
- Object-oriented programming
- Text manipulation (regular expressions & Unicode)
- Mathematical computing (math, Decimal and NumPy)

Topic 2: Software engineering

- Useful software tools (IPython)
- Design, defensive programming, debugging techniques
- Code documentation & testing (epydoc, doctest)
- Structured data (XML, YAML, JSON)

Topic 3: BioPython et al.

- Because it sucks
- ... and other bioscience libraries
- CoreBio
- Others?

Topic 4: Other Python modules

- Data visualisation and graphics (Matplotlib, PIL, Sping)
- Networks & graphs (NetworkX) and simulation (SimPy)
- Simple GUIs with PythonCard
- Introduction to databases and associated libraries (the Python DB-API, SqlObject & SQLAlchemy)
- Web presentation and frameworks (HTML, TurboGears)
- Restructured text and LaTeX (Docutils)
- PDF generation (ReportLab)?

Detailed outline

- To be announced ...
- Week 1: James does everything for me

This should not be taken as the actual sequence of lectures, but more as how items from the above list will be realised. While there will end up being something like 25 sessions, some of these will be devoted towards your own projects and presentations.

Week 2: objects, imports, tools, numbers

1.
 - Abstraction. Type of languages. Programming language features?
 - Data structures. Built in. Objects & object-orientation.
2.
 - IPython and object introspection.
 - Objects and “magic” (special) methods. Separating interface and implementation.
3.
 - More about objects. Duck-typing. Subclasses. `object`.
4.
 - More about subclasses. Class design. docstrings.
 - Multiple inheritance. Mixins. Class introspection.
5.
 - Names and memory. Importing and modules.
 - “like” types. `yield` and iterators.

The main objectives for week one will be familiarisation with IPython, and learning about importing / modules and object-orientation. To an extent the last depends on grasping functions and namespaces, so those will be covered too. A large part of day 5 will be devoted to summary, self-study and exercises.

Week 3: Databases and numbers

1.
 - Storage options: simple databases, pickling, shelve.
 - Using `csv`.
2.
 - Databases.
 - The Python DB-API.
3.
 - SQLAlchemy.
4.
 - SQLAlchemy.

As databases will be required by some of the projects, this topic is being bought forward. Projects have to be fixed by mid-week!

Week 4: GUIs

1.
 - Pythoncard
 - Using `csv`.
2.
 - Databases.
 - The Python DB-API.
3.
 - `SQLObject`.
4.
 - SQLAlchemy.

This week will be devoted towards things that people need for their projects: simple GUIs with PythonCard and image handling via PIL. As the course progresses, we will allow more time for your own work on projects. By the end of the week, everyone should have some code to show me for a check on how their projects are going.

Week 6: Dogends

This week we will mostly be panicking, trying to finish projects. As a consequence, the topic load will be light and devoted to those subjects that can be left until the end without harm.

1.
 - Restructured text
 - LaTeX
2.
 - PDF generation with ReportLab
3. ?
4. ?
5. Presentations

Assessment

- An approved project, of your own choice or from suggestions
- Non-trivial
- Complete
- May have to self-teach
- Pairs okay

Schedule:

- Topic by mid week 3
- Progress reports mid week 4 & 5
- Presentation & code-check, week 6

Marking will be via completion of a programming project of your own choosing. Ideally it should be something relevant to your study or research, but a small list of ideas are provided below. Some guidelines:

- It should be non-trivial.
- It should be complete. That is, while it can be written so as to be extended or incorporated into a larger project *later*, what is presented must be functional *now*. It can be a prototype, but it has to work.
- Extensions or add-ons to other frameworks (e.g. BioPython) are acceptable, as long as their use is demonstrated.
- The resultant software doesn't have to be pretty, but it should be usable and reusable. As a rule of thumb, a script that will only be run once isn't acceptable.
- Given the time constraints of the course, you will probably have to work out some material before it is covered in class. For example, if you're using a web framework, be aware we may not cover them until late in the course.
- All project proposals have to be vetoed by myself and should be fixed by mid week 3. If need be, project aims can be renegotiated along the way. Presentations will be given on progress in weeks 4 and 5 and the marking based on a final presentation and examination of the code in week 6. Unfortunately no extensions are possible.
- You can work in pairs if need be.

Project suggestions 1

- A web service storing, visualising and search some type of biological data.

Turbogears and Django are probably the web toolkits that are quickest to get up to speed on.)

- (For Java programmers only.) Java has some useful libraries and features that Python doesn't (e.g JDBC). Make one of these services accessible from Python.

Fortunately, there are several ways to wrap Java or call it from Python. See Jepe and Jype.

- Many types of biological data are graph-like in nature (e.g metabolic networks, gene networks). Build a framework for visualising and manipulating these in Python.

It may make sense to build this on top of an existing framework, like NetworkX.

Project suggestions 2

- BioPython sucks. Find an item on BioPython's bug list, or part of it that is poorly designed and fix it.

CoreBio could use some of BioPython's functions so design it for inclusion in that library.

- Design a module for representing phylogenies, rooted and unrooted, with attendant functions.

Of course, any library should be documented, tested and demonstrated.

- Design a module for drawing phylogenies.

Maybe you can work with someone on the previous project. The SPNG module may be a good drawing toolkit.

- Write a GUI or parser for creating and editing one of the many complicated biological data formats: NEXUS, BioEdit etc.

Advanced Python, NBN 2007. P-M Agapow (agapow@bbsrc.ac.uk)