Kursnamn “Practical python programming for Big Data and the scientist”
Name of course

Omfattning (högskolepoäng) 4, divided into beginners and advanced parts (2+2)
ECTS credits

Tidsperiod Autumn 2016
Course period

Antal platser first part 30 and second 15
Maximum number of participants

Undervisningsspråk English
Language of instruction

Kursens syfte samt motivering till varför den bör vara fakultetsgemensam

The main goal of the course is to provide sufficient know-how in the craft of programming to solve the easy and intermediate computational problems a scientist will be confronted with, by themselves. The course will give them a basic practical proficiency in the python programming language. After the course the student should be able to:

* Write simple and intermediate programs to process, filter, clean, analyze, and visualize biological data, such as, sequencing data, mass spectrometry data, etc.

* Understand the paradigm of object-oriented programming.
* Understand the advantages of using test-driven development.
* Learn best practices in programming.
* Basic proficiency with a revision control solution.
* Ability to quickly debug.
* Ability to read understand and communitate with other programmers.

Acquiring programming skills is, by essence, multidisciplinary since one is learning a tool that can be applied in so many different areas. Programming is useful in Physics, Math, Chemistry, Biology.

Kursinnehåll, kursens uppläggning samt examinationsform

Scientists nowadays require strong computer and data analysis skills. With this course we will give the practical knowledge that a scientist needs in his endeavors when using big data. The students will receive the required theory about how a computer operates and how a programming language is structured. They will also learn in a practical fashion, by applying each concept in problem-oriented computer labs. This hands-on experience is essential in being able to reproduce these skills once the student is alone. They will also complete a project of their choosing, typically something that they need or wanted in their current projects. For instance, the development of an automated analysis procedure, or the creation of a series of programs to process the large quantities of data acquired or made available in public repositories. Examination will consist of participation in the lectures, the computer-labs, the home assignments and the practical project. A detailed course outline is attached.

Målgrupp/er (specifiera ämnen/inriktningar) samt rekommenderade förkunskaper
Target group/s (specify, if possible, subject/specialization) and recommended background
PhD students and post-doctoral scientists from science and technology branches (especially biology) are targeted by this course. No Previous programming experience is need, but basic computer skills are required.

Huvudansvarig institution Departement of Ecology and Genetics
Department with main responsibility

Andra inblandade institutioner (specifiera hur).
Other departments involved (specify how).
n/a

Kontaktperson/er (namn, e-postadress)
Contact person (name, e-mail address)
alexander.eiler@ebc.uu.se and lucas.sinclair@me.com

Anmälan om kursdeltagande till
Application from course participants should be sent to
lucas.sinclair@me.com and alexander.eiler@ebc.uu.se

Senast 31st of August 2016
Not later than

Kursen har tidigare givits (ange när) n/a med n/a deltagare (ange antal)
The course has previously been given (specify when and number of participants)

Page

1 Introduction

This document includes the information relative to the creation and subsequent running of a python programming course for scientists in natural science oriented towards PhD-level and postdoc-level students at Uppsala University in the fall semester of 2016.

2 Motivation

The last decades have brought many changes to all the domains of science and technology. One trend is the steadily increasing importance and necessity of using large quantities of data to answer scientific questions. For instance, the field of physics has almost entirely switched to data-driven research and, indeed, students in physics learn programming very early nowadays. An other field that has been heavily affected is that of biology. There, "Big Data" is also changing the fundamental ways in which science is conducted, the cost of DNA sequencing having dropped by five orders of magnitude in the last ten years. Yet, not all scientists are ready or have been trained for this sweeping change.

Everywhere, enormous amounts of data are now being produced and the computer operators required to analyze and process them have largely become the bottleneck in many research programs. Sadly, the computational skills needed are often not included in the standard university curricula, yet many scientists are now exposed to such big data and spend half or more of their time behind a keyboard instead of in the laboratory. In this course, we will give the practical knowledge that a scientist needs to address the common challenges of large-scale data analysis. This course won't make them professional information technology specialists, but it should make the participants literate enough to solve a large portion of their problems by teaching them how to program in a modern, widely-used, efficient and user-friendly language: python. They will also a learn a bit about programming tools around python that can help them organize and reuse.

In essence, we see many of our peers that are either stuck at a low level of proficiency and only venture into Excel for basic problem solving or, in the second case, are a bit more proficient but have been hired as de-facto IT scientists while lacking hard programming skills. After this course, the students will become versed in python and they will be able to solve day-to-day computational problems by themselves. At the same time, the heavier users will be able to produce more readable, maintainable and shareable code, while automating large parts of their analysis and becoming leaps and bounds more productive.

Programming courses aimed at the modern scientist that missed this type of training in his undergraduate path are scarce. Some courses do exist but they tend to be much shorter and devote, in our opinion, too little time to hands-on exercise which are critical to the learning of a new skill.

3 Why Python

There is a plethora of programming languages that have been developed over the last fifty years and the variety of choice can seem overwhelming at first. We should maybe start by asking ourselves: why should we even focus on only one language and not learn all the best ones? Well, the answer to that is quite simple. Firstly, it takes a significant amount of time investment to learn the modalities and syntax rules of a programming language, even for the ones that are designed to be quickly assimilated. Secondly, your productivity will be much greater if you have a deep mastery of a single multi-purpose programming language when compared to the situation where you have but a surface knowledge in many different programming languages.

It is therefore optimal in the context of answering scientific questions to select one particular language and to become good at it. The language should be modern, widely used (i.e. large community of support), easy to learn, and must be able solve a wide-range of problems (i.e. not a domain specific language.).

In addition, when choosing a language, there is a balance to be made between the speed of execution of the final product in terms of processor-hours and the time it takes to create the final product in terms of programmer-hours. Languages that are "close to the machine" such as Assembly will create programs that can run extremely fast and in an extremely resource efficient manner, however they will take months or years for the programmer to create. Languages that are "far from the machine" and are abstracted away from the bare metal will run in a slower fashion but will enable the programmer to create them in minutes or hours. The current situation in scientific computation is that most of the challenges commonly faced can be resolved without the need of highly efficient programming. It is thus desirable to choose a high-level or "scripting" language to promote the best productivity. In other words, letting your script run a couple hours or days is never a problem and you can work on other things in the mean time.

This still leaves us with many choices that could be more or less adequate. We believe python is the best candidate for this day and age in science and technology. It presents many advantages and also posses several complementary tools developed for it, whether for biology, physics, chemistry or mathematics.

We can highlight some of the problems that we have identified with the other candidate languages:

Perl - Perl is somewhat of an old tradition in scientific computation (especially biology) but has been strongly declining in recent years. Many legacy code exists and it can be useful to know about perl when one is tasked with debugging old projects. Some groups still use it for new projects, but overall it is being abandoned in modern genomics. For instance, the latest version, Perl 6 was never adopted.

R - This is easily one of the most popular languages in science (especially for statistics) alongside python. It has two strong points which are applying models to multi-factorial data and creating graphs or visualizations. Unfortunately, outside of these very specific tasks it is very lacking and is a typical example of a domain-specific language that is only good for solving a particular class of problems. Its design is odd, possessing several object models and being a cocktail of imperative and functional features [2].

Matlab - Though popular in many fields such as electrical engineering or meteorology, this is a commercial product which forces the programmer to be bound to MathWorks®. Buying a license is very expensive. There is no reason today not to use free and open source solutions.

Java - Overall it is pretty close to the machine and requires lots of boiler plate code. Though there are scientific libraries for some fields, it is rarely used in real-world science projects.

C - Very close to the machine and only useful for a scientist in the cases where computational performance is absolutely crucial. Which is almost never.

Julia - A very interesting upcoming language that would suit very well the kind of problems in science. But it's still in its infancy and not suitable to be taught yet.

Bash - Not a proper programming language. Only really good at interactive use and browsing file systems. Avoid.

This specific course will teach python as it is found in the 2.7.x version and will not concern itself with 3.x.

4 Content

In the first half, the students will receive the required theory about how a computer operates and how a programming language is structured. Most importantly, they will learn the python language in a practical fashion, by applying each new concept in problem-oriented computer sessions. This hands-on experience is essential in being able to reproduce these skills once the student is alone again with his/her own *Big Data* and scientific questions. Then, in the second half, the students will complete a project of their choosing, typically something that they need or want in their current and future academic projects. For instance, the development of an automated analysis procedure, or the creation of a series of programs to process the large quantities of data acquired in their laboratories. There is a substantial degree of freedom in choosing the course project and we encourage the students to build something that will be useful for them.

In summary, the students will get the basic training and the core skills needed to be productive in a data-oriented scientific research team. In addition to understanding how to produce code in a semi-professional manner, this will include introduction to standard tools such as those for archiving, backuping, distributing and installing the code they write. Practically, this will be taught with short lectures and tutorials alternating with many practical exercises.

5 Learning Outcomes

The main goal of the course is to provide sufficient knowledge in the craft of programming to enable scientists to solve the easy and intermediate computational problems they will be confronted with independently. The course will give them a intermediate practical proficiency in the python programming language. After the course the student should be able to:

- Write simple and intermediate programs in python to process, filter, clean, analyze, and visualize scientific data.
- Ability to automate much of the computational tasks that are used in their day-to-day work.
- Understand the universally used paradigm of object-oriented programming as it is implemented in python.
- Understand the necessity and advantage of using test-driven development.
- Understand the best practices in python programming, such as the advantages of style guides.
- Basic proficiency with the revision control solution that is "`git`" to archive and distribute the programs or scripts created.
- Acquire experience with understanding and quickly debugging errors within the code.
- Ability to read and understand programs written by one's pears, to review them or modify them.

6 Demand

A recent survey run in 2015 in the Uppsala-Stockholm academic area found more than 130 people saying that they were interested in a "Python for genomics" course. The survey was only targeting biologists but since our course is addressing a wider audience, we can assume that the interest is even larger.

In total, 169 responses were recorded. A few select questions and the answers offered by the participants are shown below in figures 1 and 2.

Response	Count	Fraction
Not interested	**2**	*1.2%*
It depends a bit on the content	**29**	*17.2%*
Interested	**79**	*46.7%*
Count me in!	**58**	*34.3%*
Other	**1**	*0.6%*

Figure 1: Question: "How much would you be interested in a course teaching how to use python for bioinformatics?"

Response	Count	Fraction
I never used it	**64**	*38.3%*
I create basic small scripts	**81**	*48.5%*
I create my own classes and modules	**14**	*8.4%*
I am an advanced user	**8**	*4.8%*

Figure 2: Question: "Do you have any experience with python?"

7 Duration and time plan

The course will run over a month and be split into two parts, one basic part and one advanced part. The second part will be the natural continuation of the first part and will be focused around the personal project. The participants can inscribe to only the first part, only the second part, or to both.

The basic part would run over the course of two weeks and contain six half days as shown in figure 3. A half day would include four hours and a half running from 13h00 to 17h30 with three coffee breaks included. Courses would be split evenly between lectures/seminars and exercise/practical sessions.

	Monday	Tuesday	Wednesday	Thursday	Friday
Week 1	Afternoon	Afternoon	Afternoon	no course	no course
Week 2	Afternoon	Afternoon	Afternoon	no course	no course

Figure 3: Schedule for basic part

The advanced part would include three half days of lectures and exercises. The rest of time is devoted to the personal project with the hand-in deadline placed on the last day of the fourth week as shown in figure 4. A reasonably short deadline avoids that the personal projects become disproportionately ambitious and take the students too much time.

8 Working title

"Practical python programming for Big Data and the scientist"

	Monday	Tuesday	Wednesday	Thursday	Friday
Week 3	Afternoon	Afternoon	Afternoon	no course	no course

	Monday	Tuesday	Wednesday	Thursday	Friday
Week 4	no course	no course	no course	no course	p. deadline

Figure 4: Schedule for advanced part

9 Credits

The estimated worth of this course in its current form is 4 ECTS. They are split evenly between the two parts. 2 ECTS for the basic part and 2 ECTS for the advanced part.

10 Semester period

The planned date for this course to be given is in October 2016. Ideally starting on the Monday of week 41. But this is flexible.

11 Target group

We are aiming for PhD-level and post-doc level students of all backgrounds. The course will not be open to Master students due to the limited number of participants. Another run of this course for Master students can be possible at an other time.

12 Number of seats

- 30 students for the basic part.
- 15 students for the advanced part.

The limited number of seats in the advanced part is due to the fact that we estimate that each teacher can only realistically take on the supervision and correction of five personal projects each.

13 Venue

It depends on availability naturally, but we would like to use room 1003 at the Evolution Biology Center. This room can be used without paying any renting fee. It is found at Norbyvagen 18D, Uppsala, on the first floor to the right, next to the fika room.

14 Entry requirements

The only knowledge requirement is basic computer usage which everybody must have to be admitted to a PhD anyways. Some Unix/Bash experience is recommended but not required for course participants. Previous knowledge in other programming languages is a plus.

The participants are expected to bring their own portable computers. No computers will be offered on-site. There is no requirement on the operating system installed on the participant's laptop. Both *OS X* and *Linux* come with python pre-installed, and for *Windows* it's not too hard to install it yourself.

We think it's best to teach students on their own personal computers so that they may continue to use the tools and technologies they have learned later on. When putting students in front of university provided computers, they are often unable to reproduce the same setup or installation on their own machines afterwards and stop using the skills they practiced once the course is over.

15 Pre-work before attendance

A certain amount of work is required before coming to the first day of the class. We would like all students to log on to `https://www.codecademy.com/learn/python` and complete the first five units. These are the following:

Unit 1 - Python Syntax

Unit 2 - Strings and Console Output

Unit 3 - Conditionals and Control Flow

Unit 4 - Functions

Unit 5 - Lists and dictionaries

This should take about six hours for a beginner. This enables everyone to have at least a faint idea of what python coding entails and looks like before starting the course. By spending some time alone at first, the students will start to be habituated in python and will be able to focus on the more interesting programming aspects of the course, being already familiar with the very basics such as fundamental syntax. To confirm their inscription, we would like participants to send us a screen-shot showing their achievements on the web site at the latest on the Friday of the week before the course start.

We also would like each student to have `ipython` installed on his/her laptop as well as his/her favorite text editor. Because, for some students, even this is a challenging task, we propose to organize an "Install Fest" prior to the course start where anyone can come to set up his/her computer properly. After this event a social activity is planned to get to know each other better.

Optionally, we recommend that the students also add `pyflakes` to their favorite text editor (`pyflakes` is a simple extension which checks Python source files for errors and warnings).

16 Daily schedule of basic part

The teaching consists of lectures and computer exercises which are intertwined. Each day, the session runs over four and a half hours with three coffee breaks included. We will put a strong emphasis on being interactive and will include constant feedback from teacher to student as well as from student to teacher. We want to make sure that the level of engagement is appropriate and that students are stimulated and challenged while still comfortable with what they are learning.

16.0 Day 0

An optional install fest where anyone can come to us to check he has a working copy of `ipython` and a good text editor on his/her computer.

16.1 Day 1

Interactive lecture

- Examples of why programming is absolutely essential for your life as a scientist. From simple to complex examples. Reproducible science means reproducible programs, some journals require you to publish code.
- Why did we select python as a language (mention examples with competition). But also what is python (imperative and not functional, duck-typed, high-level). Why did we choose version 2.7.x.
- *Theory*: The fundamentals of a modern computer: processing power in terms of flops, basic number representation, the effect of the cascading memory speeds from hard drive to L1 cache.
- A brief introduction to bash: a simple interface to interact with your operating system and file system, and why it should be avoided.
- *Theory*: The three realms to being a good programmer in any language: data, instructions and syntax.
- The two modes we will be using python in: the interactive prompt for live exploring against files for storing finished code as well as code that we are working on.
- Review of the built-in types: `int`, `float`, `str`, `list`, `tuple`, `dict`, `set`.
- Review of control flows: `if`, `else`, `for`, `while`
- Defining functions.

Exercises

- Figure out how long the computer waits in terms of clock cycles when requesting a resource (i) form a local 7200rpm and 10ms seek time hard drive (ii) on a server in California.
- Reflect upon the syntax of a programming language: should it be close to English or not ?
- Simple exercise that involves casting different types and avoiding `TypeError` exceptions. (just a level above the codeacademy ones).
- Simple exercise that involves using control flows (just a level above the codeacademy ones).
- Exercise that makes use of several functions.
- The students give us feedback on the course.

16.2 Day 2

Interactive lecture

- *Theory*: A brief introduction to algorithms: why and when does the order of computation really matter (simple example when comparing elements to each other = N^2, and example of De Bruijn's graph).
- Mutable and immutable types. Pass by copy or pass by reference.
- List comprehensions.

- File I/O and text manipulation. What is parsing and serializing. Unicode vs ASCII.
- The built-in modules: `os`, `sys`, `re`, etc.
- Regular expressions.

Exercises

- An exercise showing why code duplication is bad.
- An exercise showing reference passing.
- A few exercises on text manipulation and file I/O. Experiment with encodings.
- An exercise using regex patterns.

16.3 Day 3

Interactive lecture

- Linear versus structured programming: advantages and techniques.
- Object oriented programming as the dominating paradigm since it took over C's `structs`.
- How objects link to data structures (DS). How a problem is often solved by simply identifying the right DS.
- Using python packages: how to download and install most of them.
- Packages studied in more depth:
  1. `matplotlib`
  2. `statsmodels`
  3. `pandas`

Exercises

- Example that uses objects and show basic inheritance and basic composition.
- Example that shows the right selection of data structures.
- Example that combines all the three packages: load a dataset with pandas, run a stat model on it, then plot a visualization using matplotlib.

16.4 Day 4

Interactive lecture

- Control your code and share it: introducing `git` and subsequently github.
- From now on, assignments are to be uploaded on github (use the GUI preferably at first).
- Make your own modules that can be imported.
- Introducing PEP8 and coding style guides.
- Comments and docstrings.

Exercises

- Make your account on github and start making simple commits. Use either command line or GUI.
- Given a TSV file that has a table where columns are identifiers and rows are different samples. Also given an index where each identifier is associated with a classification. Some identifiers might belong to the same classification ! Instruction: make a new table linking all different classes to different samples and save it in CSV-UTF8 format.
- Write a simple parser (e.g. FASTA) that responds to certain requirements and passes certain given tests.
- Students comment on each other's code and try to establish a personal coding style guide.

16.5 Day 5

Interactive lecture

- The different licenses you can put on your code: MIT, BSD, GPL, LGPL, Creative Commons.
- *Theory*: Introduction to the Zen of python and some philosophical remarks.
- How test driven development is the only safe form of development, especially in science where trust is so important. Shocking example: e.g. blog post in QIIME.
- How python can be used to do functional programming. Iterators and generators.

Exercises

- Example showing the advantage of lazy processing data, such as an iterative dataset that can't hold in RAM.
- Write a program that identifies where proteins are hiding in nucleotide sequences (three difficulty steps: simple visual print out, auto frame detection, multi-protein).
- Given a class structure where methods are left blank, automate the processing of N samples through a small series of tools.
- Add automated test to past exercises or other objects.
- Second round of evaluation.

16.6 Day 6

Interactive lecture

- Things that the students said were not clear enough. This lecture can be designed according to the feedback and the questions we will have gotten. It is a buffer zone of sorts and in any case can be just used for a Q&A session.

Questionnaires

- Evaluation of student's theoretical knowledge. Questions on the lectures that were given. Written questionnaire, no computer. One hour.
- Practical knowledge assessment. Exercise like the ones they have been doing in the past two weeks. Computer and internet available. Two hours.

17 Daily schedule of advanced part

Here, the students will develop their own project: a specific analysis he/she wants to program or automate, a tool she/he wants to create to process some data, or even a research-project on programming. The students are free to choose the direction of their project, but it must be doable in no more than about four days of programming.

17.7 Day 7

Interactive lecture

- Example of how to turn a function oriented script into an object-oriented script and make it better, more readable, more maintainable, and more powerful.
- Nice uses for decorators.
- SQLite3 databases or HDF5 storage for different types of data.
- Premature optimization is the root of all evil. But when you do optimize, run a profiler. Examples of profilers.
- Following the optimization theme: what is garbage collection and how it helps you.

Exercises

- Without googling them, try to come up with one example for each of the points of the "Zen of Python".
- Example where the student must identify the slow parts of a program.

17.8 Day 8

Interactive lecture

- A few examples of the dark magic that python can do and why it should be avoided. Introspection such as touching `self.__dict__` or even `__module__`, the few cases where `__new__` is justified, metaclasses, monkey patching. Readability is important, don't use black magic.
- *Theory*: The slightly different pythons: CPython, RPython, PyPy, IronPython, Jython, StacklessPython, etc
- *Theory*: The more different pythons: Python 2 and Python 3. What are the usage stats ? Why should I care ?
- Python and speed, when it can actually be useful to write a loop in C.
- Python and concurrency: why it is awful and a brief mention of the Global Interpreter Lock.
- Making user-friendly command-line utilities in python.

Exercises

- Deciphering a script with too-much black magic.
- Small exercise with a few command-line parameters.
- Teachers offer help and guide the students in defining the skeleton of their project.
- Students discuss their personal project draft with other students and try to finalize the blueprint of their course project.

17.9 Day 9

Interactive lecture

- Making pipeline or workflows in python. Libraries or roll your own.
- Examples of previous projects and pipelines. Good and bad sides of design. Stories form the past.
- More complete ways of documenting your code. UML diagrams, sphinx autogeneration, `readthedocs.org`, github's issues and wiki.
- Methods for collaborating on code and team management. Concepts such as Agile, Sprints, Scrum and Extreme.

Exercises

- Each teacher takes on 5 students and three groups are created.
- Within groups, short 5 minute presentations for each student on their final project blueprint.
- The scope and draft of each personal project is validated by the teachers and students have until the Friday of the next week to complete them. They may come to us in between and each one has a certain amount of meeting time they can use live or via Skype.

17.10 Day 10

Deadline: after the equivalent of four days of individual work, the project must be handed-in. The projects are evaluated by the teachers. Corrections and comments are noted for the students benefit. In the following week, the teachers write up a summary applauding the student's success and also bringing a constructive criticism showing where they may still develop.

17.11 Day 11

Sometime after the course is over, the student will receive a link by e-mail to participate in an anonymous course evaluation where they have the possibly to grade the teachers according to the same scale they are evaluated with: Fail (U), Pass (G), Pass with distinction (VG).

18 Simple example of practical usage

Every now and then, someone comes up to us asking for our expertise in the subject of programming to solve a particular problem he has with some data analysis. This generally includes things such as the loading of text files, some kind of processing and production of other text files as output. The tasks are often very simple from an algorithmic point of view, but are sufficiently

subtle to prevent their implementation in Excel, the data tool that the scientist is usually familiar with. They ask for help because the execution of the desired task without such intervention would imply spending between minutes to hours doing repetitive tasks in a graphical interface. Not only is this incredibly dull for the intellectual mind, it is also error prone and unscientific.

One day, a post-doc came up to me with the following problem at hand: my colleague had a few hundred text files in a directory that each contained some type of fluorescence measure. The files all contained two columns, tab separated, with the wavelength on the left and the absorbency coefficient on the right. They all looked something like this:

```
123 12344
456 3444
135 12344
... continues for many lines
```

He wanted to pick out every entry in every file that had a wavelength greater or equal to 400. The solution was a matter of only eleven lines of code, but it requires knowledge to write them.

The solution can be viewed at `https://gist.github.com/e8307e8569ee33ec7b46`

This is the kind of things the course would attempt to teach in the very first lectures. Of course, the tasks at hand are usually more complex and this is only a simple representative of the type of problems which we see several faced with around us, without any appropriate means of solving.

19 Complex example of practical usage

A more advanced example of a data processing problem that a scientist might encounter and would have to solve with by programming could be that of a biological phylogenomics analysis, such as the one that was recently implemented by us in a publication about metabolic potential in freshwater habitats [1]. Here, starting from 29 reference genomes (as FASTA files) originating from multiple aquatic microbial clades, various analysis are carried out and a final phylogenetic tree is constructed to bring an answer to our scientific question. First, genomes clusters are made according to homology measures on predicted protein regions. Then, using a selection of concatenated coding regions, a multiple sequence alignment is generated. Next, a bootstrapped tree is built using a specific evolutionary model. In the end, that tree is visualized and a PDF containing the result is outputted. This result can be seen in figure 5.

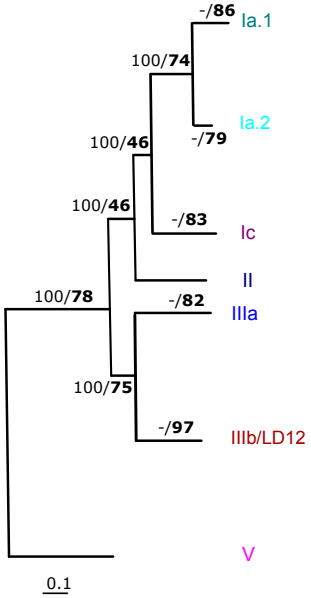

Figure 5: Excerpt from Eiler et al. ISME J. 2016

The process described above is implemented in a "pipeline" where, once the inputs are defined and loaded, it can proceed without user intervention down to the final output. One of the main advantages of having automatized the generation of this figure is that a variety of methodologies can be explored quickly. Indeed, it becomes easy to vary one of the parameters of the analysis such as the homology threshold or the minimum length of coding regions and regenerate the figure in a short time. In this way, a hundred or a thousand such figures can be generated overnight within an interval of possible parameters – greatly increasing the ability of the scientist to explore his/her data!

The code making up this pipeline was written in python in a object oriented fashion and can be viewed at: `https://bitbucket.org/xapple/ld12/src`

This is the kind of project that the students would attempt to develop in the advanced part of the course (though this one took a bit more than four days). Of course, the problems at hand can be very diverse and this represents only an example from our own research.

NB: If you are interested in the science behind this project, briefly, the figure details evolutionary relationships between closely related marine and freshwater bacteria. This is an unrooted maximum likelihood phylogenetic tree of concatenated ribosomal protein sequences from single-cell genomes and isolate genomes of the SAR11 clade. In addition to bootstrap values as inferred by maximum likelihood, the second (large and bold) number at the branching points show the proportion of protein trees with a corresponding branching pattern. These proportions were generated by using the RobinsonFoulds metric summarizing 518 orthologous protein trees that had at least one representative in each of the seven subclades.

20 Reading material

No specific book is associated to this course, and the students will receive different written materials coming from various sources. However, if the students want to have such a book to follow and help them, they can use this free one:

`http://openbookproject.net/thinkcs/python/english2e/index.html`

If this book is read by the student, it is our opinion that the chapters after number 17 are less interesting in the aspect of python programming for the scientist.

21 Assessment

21.1 Basic part

- Participation (attendance sheet) in mandatory lectures (0.5 credits), where essential knowledge is acquired.
- Participation (attendance sheet) in mandatory computer-lab practicals (0.5 credits), where hands-on experience is gained.
- Participation in the knowledge assessment questionnaire (0.5 credits) dealing with general programming concepts and knowledge about how a computer operates (Day 6).
- Participation in the practical knowledge-check session (0.5 credits) dealing with specific practical programs that the student must solve on his/her computer (Day 6).

21.2 Advanced part

- Participation in lectures and exercises of the advanced part of the course (1 credit).
- Satisfactory report and results based on a student-chosen research project or tool creation project (1 credit).

To pass the course in full, all assignments and exams must be completed and approved. A maximum of two missed mandatory attendances can be compensated by make-up work.

22 Grading

The grading system is the following: Fail (U), Pass (G), Pass with distinction (VG).

The final grade is a weighted mean based on the examinations and attendance detailed above.

23 Pedagogy

23.1 Multi-modal

One of the keys to success in a new course where the student's expectations are high is obviously the skills and pedagogy displayed by the teachers. We are armed to face this challenge and are accustomed to adapt ourselves to the different learning style that each student can prefer. Our lectures and materials are designed to be adequate for visual learning, auditory learning and read/write type learning.

23.2 Formative assessment

It's quite important in a course like this to be able to quickly adapt our lessons as the days pass and the needs of the student become better understood. We need to receive feedback from the students as early as possible to see if the pace is too fast or, on the contrary too slow. We need to estimate if the content we cover is optimal for the background of our students. We need to know if the level of complexity of the course is too high for the participants already on the first day. For this we intent to have several feedback and evaluation questionnaires along the course. At the same time, we will glean information in other more informal contexts such as at the coffee breaks.

23.3 Active exercises

We plan to have many activating exercises to cut up the long lectures. Students will be asked to perform quick exercises that all require them to write a few sentences on small cards that are subsequently picked up by the teachers:

- **Paraphrasing**: The student should recount the concept/concepts learned as if he was explaining them to a particular group. The target group could be a fellow student not having attended the course, his/her grandma, his/her boss, etc.
- **Application creation**: The student is tasked to imagine and invent an application for the concept or method he just was taught. In other words, a practical scenario where the knowledge gained could be useful.
- **Questionnaire prognostics**: The student has the opportunity to come up with a knowledge test question that he/she thinks up. This forces him/her to conceptualize and summarize the notions he/she has apprehended. No promise is made that we will use them, of course.

In addition, it is also planned to run several sessions where we will use the voting clickers (Turing technologies ResponseCard) that can be borrowed at the teaching department. This will be combined with interactive blackboard activities (e.g. `https://padlet.com`).

24 Teachers

Three teachers, two hired externally.

24.1 Ludovic Dutoit

E-mail : <ludovic.dutoit@ebc.uu.se>

Position : PhD Student at EBC

Department : Department of Ecology and Genetics, Evolutionary Biology

Statement : Currently a PhD student at the department of evolutionary biology, I work with large genomics on questions relative to the evolution of genomes and the creation of species. I am a biologist by training but I have gradually become accustomed to programming and handeling large datasets through my academic training.

Experience : Python and R user. I have been involved in teaching courses in evolutionary genetics for undergraduates students. At the post-graduate level, I have been teaching the European workshop in Genomics in Czech republic, 2015.

24.2 Alexander Eiler

E-mail : <alex@envonautics.com>

Position : Envonautics consultant and docent (currently also P.I. at EBC)

Showcase : `https://github.com/alper1976`

Statement : I am a biologist with a large interest in the technological aspects of genomics. Thus, I have acquired scientific programming skills along my career, and not only out of the necessity to take an active part in the big data revolution. I also greatly enjoyed this endeavor. I am now able to collaborate efficiently with bioinformaticians, computer scientists and programmers on multiple projects. This is a very valuable skill that I can transfer to the students. Having learned these skills "the hard way", I am better able to understand the hurdles and difficulties that other scientists will encounter.

Experience : Python and R user, led and taught multiple courses at Uppsala University, including introductory courses into genomics, years of experience in leading data heavy research projects.

24.3 Lucas Sinclair

E-mail : <lucas@envonautics.com>

Position : Envonautics consultant (currently also PhD student at EBC)

Showcase : `https://github.com/xapple`

Statement : I have always been interested in programming and started at a relatively young age by making small video games. This has helped me in my under-graduate training as an engineer in life sciences and enabled me to easily specialize in bioinformatics. As I have now joined a more "classical" wet-lab and field biology department, today, I feel that my programming skills are one of the most valuable skills I could transfer to my peers. Overall, it's a subject I feel very comfortable in and enjoy teaching.

Experience : Veteran python user, taught a python course at the SIB (Swiss Institute of Bioinformatics), followed the Academic Teacher Training course, contributed to the open-source python projects.

25 Budget

Funds are required to finance the three teachers that will work on this course for four weeks. All three organizers are currently employed by Uppsala University, two contracts will be terminated by summer 2016. They will thus need to be hired and teach on a consultant or temporary-employed basis. Out of the total amount, 123'780SEK will go to these two teachers (Docent Alexander Eiler, future PhD Lucas Sinclair) and 27'500 SEK to the UU employed teacher (PhD candidate Ludovic Dutoit).

The current suggested rates for a PhD student, a post-doc and a docent are 550, 820 and 960 SEK/h, respectively. As a promotional offer we are ready to accept hourly rates of only 550, 720 and 850 SEK/h, respectively. Finally, we add a total of 3000 SEK in the case a room needs to be booked or other unforeseeable costs arise. The total budget of the course is detailed in figure 6.

	Basic part					
Name	per hour	lectures	lab/seminars	supervision	total hours	SEK
Ludovic Dutoit	550	1	12	2	30	16'500
Alexander Eiler	850	3	12	2	38	32'300
Lucas Sinclair	720	8	12	2	58	41'760
	Advanced part					
Name	per hour	lectures	lab/seminars	supervision	total hours	SEK
Ludovic Dutoit	550	0	6	8	20	11'000
Alexander Eiler	850	2	6	8	28	23'800
Lucas Sinclair	720	4	6	8	36	25'920
Extra costs						3000
					Total:	**154'280**

Figure 6: Detailed budget for each teacher.

The following funds are applied for:

- Faculty: Science and Technology - 80'000SEK
- Institution: Department of Ecology and Genetics - 74'280SEK

26 References

[1] Alexander Eiler, Rhiannon Mondav, Lucas Sinclair, Leyden Fernandez-Vidal, Douglas G Scofield, Patrick Schwientek, Manuel Martinez-Garcia, David Torrents, Katherine D McMahon, Siv GE Andersson, Ramunas Stepanauskas, Tanja Woyke, and Stefan Bertilsson. Tuning fresh: radiation through rewiring of central metabolism in streamlined bacteria. *The ISME Journal*, January 2016.

[2] Floréal Morandat, Brandon Hill, Leo Osvald, and Jan Vitek. Evaluating the Design of the R Language. In *ECOOP 2012 – Object-Oriented Programming*, pages 104–131. Springer Berlin Heidelberg, Berlin, Heidelberg, June 2012.