Title	: "Practical python programming for Big Data and the scientist"
Credits	: 2 ECTS for the basic part, 2 ECTS for the advanced part.
Language	: English.
Seats	: 30 for the basic part, 15 for the advanced part.
Faculty	: Science and Technology, Uppsala University.
Venue	: Room 1003 at the Evolution Biology Center (Norbyvägen 18D)
Price	: Free of charge.

Apply here: `https://goo.gl/AaiQEa`

Deadline to apply: 20th of May 2016 at 23h59

Dates

Part 1	: Basic part dates: 10th to 12th of October 2016 and 17th to 19th of October 2016.
Part 2	: Advanced part dates: 24th to 27th of October 2016.
Format	: Half day sessions (13h00 to 17h30) with exercises to solve on your own in the mornings.

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.

1 Motivation

The last decades have brought many changes to all the domains of science and technology. One trend that is steadily increasing in importance is the 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.

2 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 drawbacks 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.

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 much with 3.x.

3 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.

4 Schedule

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 will run over the course of two weeks and contain six half days as shown in figure 1. A half day includes four hours and a half running from 13h00 to 17h30 with three coffee breaks included. The mornings are used to complete homework-type exercices.

	Monday	Tuesday	Wednesday	Thursday	Friday
Week 41	Afternoon	Afternoon	Afternoon	no course	no course

	Monday	Tuesday	Wednesday	Thursday	Friday
Week 42	Afternoon	Afternoon	Afternoon	no course	no course

Figure 1: Schedule for basic part

The advanced includes three half days (an extra fourth day might be added if required) 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 2. A reasonably short deadline avoids that the personal projects become disproportionately ambitious and take the students too much time.

	Monday	Tuesday	Wednesday	Thursday	Friday
Week 43	Afternoon	Afternoon	Afternoon	Afternoon	no course

	Monday	Tuesday	Wednesday	Thursday	Friday
Week 44	no course	no course	no course	no course	p. deadline

Figure 2: Schedule for advanced part

5 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.

When you apply for the course, you should provide a short (200 words max.) motivation letter that will be used in case we get more applications than there are available spaces for the course.

6 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 of the python module. 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 a couple hours for a beginner. This enables everyone to have at least a faint idea of what python coding entails 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. For all the 30 accepted students, and 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.

7 Teachers

7.1 Lucas Sinclair

E-mail : <lucas@envonautics.com>

Position : Envonautics consultant

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.

7.2 Alexander Eiler

E-mail : <alex@envonautics.com>

Position : Envonautics consultant and docent

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.

7.3 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.