Advanced Python for Biologists - Detailed syllabus

1. Data structures in Python

In this session we will briefly recap Python's basic data structures, before looking at a couple of new data types — tuples and sets — and discussing where each should be used. We will then see how we can combine these basic types to make more complex data structures for solving specific problems. We'll finish our discussion by looking at specialized data types that are found in the Python core library. This session will also be our first introduction to benchmarking as we talk about the relative performance of different data types. In the practical session we'll learn how to parse an input file into a complex data structure which we can then use to rapidly query the data. Core concepts introduced: tuples, sets, higher-order data structures, default dicts, Counters, big-O notation.

2. Recursion and trees

In this session we will cover two very closely related concepts: trees (i.e. the various ways that we can store hierarchical data) and recursive functions (the best way to operate on treelike data). As recursion is inherently confusing, we'll start with a gentle introduction using biological examples before moving on to consider a number of core tree algorithms concerning parents, children, and common ancestors. In the practical session we'll look in detail at one particular way of identifying the last common ancestor of a group of nodes, which will give us an opportunity to explore the role of recursion. Core concepts introduced: nested lists, storing hierarchical data, recursive functions, relationship between recursion and iteration.

3. Classes and objects

In this session we will introduce the core concepts of object-oriented programming, and see how the data types that we use all the time in Python are actually examples of classes. We'll take a very simple example and use it to examine how we can construct our own classes, moving from an imperative style of programming to an object-oriented style. As we do so, we'll discuss where and when object-orientation is a good idea. In the practical we will practise writing classes to solve simple biological problems and familiarize ourselves with the division of code into library and client that object-oriented programming demands. Core concepts introduced: classes, instances, methods vs. functions, self, constructors, magic methods.

4. Object-oriented programming

Following on from the previous session, we will go over some advanced ideas that are common to most object-oriented programming languages. For each idea we'll discuss the basic concept, the scenarios in which it's useful, and the details of how it works in Python. This overview will also allow us to consider the challenges involved in designing object-oriented code. In the practical we will work on a simulation which will involve multiple classes working together. Core concepts introduced: inheritance and class hierarchies, method overriding, superclasses and subclasses, polymorphism, composition, multiple inheritance.

5. Functional programming in Python

This session will start with a look at a few different concepts that are important in functional programming, culminating in a discussion of the idea of state and its role in program design.

We will see how functional programming is, in many ways, the complement of object-oriented programming and how that realization informs our decision about when to use each approach. We'll take a quick tour of Python's built-in tools that take advantage of functional programming and see how we can build our own. We'll finish with a brief look at how functional programming can vastly simplify the writing of parallel code. In the practical, we'll practise using Python's built-in functional tools, then implement one of our own. Core concepts introduced: state and mutability, side effects, first-class functions, declarative programming, lazy evaluation, parallelism, higher-order functions.

6. Iterators, comprehensions and generators

We'll start this session with a discussion of Python's iteration mechanism, focussing particularly on the behaviour of the functional methods from the previous session. Next, we'll introduce the idea of comprehensions as a way to concisely define lists and generators as a way to produce those lists efficiently. We'll see how to extend the same idea to sets and dicts, leaving us with comprehensions as a powerful tool in our programming toolbox. We'll finish with a look at how we can use iterators inside our own classes, tying together the ideas of object-oriented and functional programming. In the practical, we'll re-examine some of the problems from previously in the course using the new tools. Core concepts introduced: iteration, interfaces, comprehensions, generators, eager vs. lazy sequences.

7. Exception handling

This session will start with a reminder of the difference between syntax errors and exceptions, after which we will explore the syntax involved in catching and handling exceptions. We'll then examine the way that exceptions can be handled in multiple places and the consequences for program design. We'll finish this session by learning how we can take advantage of Python's built-in exception types to signal problems in our own code, and how we can create custom exception types to deal with specific issues. In the practical we'll modify existing code to make use of exceptions. Core concepts introduced: exception classes, try/except/else/finally blocks, context managers, exception bubbling, defining and raising exceptions.

8. Packaging and distribution

We'll start this session by looking at our options for reusing code in Python and seeing how the methods differ depending on whether we want to share code between files in a program, between many programs on the same system, or between many programmers on different systems. This leads into a discussion about packaging and distribution, in which we'll discuss the roles of Python's package management tools and package repositories. In the practical session we'll turn existing code into modules and packages. Core concepts introduced: modules, namespaces, dependencies, executing modules, packages, metadata.

9. & 10.

Programming workshop based on progress of the course and delegates' interests/problems and own data