Data Structures and Algorithms

Spring 2024, BSc EFDS, Imperial College Business School

Module Lead: Ryan Cory-Wright, r.cory-wright@imperial.ac.uk

Tutorial Lead: Jay DesLauriers, j.deslauriers@imperial.ac.uk

Note: the below syllabus is indicative and may be edited during the term. Please refer to Insendi to download the latest version.

Contents

Before you get started

Welcome to Data Structures and Algorithms! The course is organized into ten sessions over ten weeks. Each session consists of a live lecture, asynchronous online materials, and a set of exercises.

We will be doing programming exercises in Python in every session, so **please install Python on your machine before starting out on the first session**. There are a few different versions of Python around. We will be using the **Anaconda distribution of Python**.

To install Python:

1. Download the Getting Started Tutorial from the Files tab on Insendi.
2. The tutorial will guide you through installation instructions and the first steps of running Python.

Objectives

This module will introduce you to computational problem solving through the design of algorithms and data structures. Devising efficient methods for sifting through large datasets is at the core of most modern technological innovation, ranging from search engines and social networks to healthcare, energy and finance. Our objective is learning to build algorithms: moving from a possibly ambiguous problem statement to formulating a computational solution method. We will introduce the basic principles of algorithm design and analysis through classic problems such as searching and sorting. We will also study algorithms on graphs, with wide applications in areas such as social networks.

We will implement our algorithms in Python, a versatile open-source language widely used in the economics and data science communities for tasks like data analysis, machine learning, web scraping, and natural language processing. Python is designed to be both powerful and easy to read, which makes it a great language for beginners to learn. Our objective is to build a strong foundation in the basic concepts of programming – such as variables, objects, loops, and functions – and learn to apply them in practical problem solving. We will also learn to read and build on other people's code, and introduce some key analytics and business use cases of Python: writing scripts to automate tasks, parsing and interpreting data, and scraping the web.

Prerequisites and materials

This module assumes no previous programming experience. However, we will cover a lot of material, so if you have not done any programming before, some extra preparation may be helpful. We believe that the following strategies will be useful for most students:

- If you would like to get started on Python before the module, spend a few hours working through an online tutorial, for example the first two chapters of *Automate the Boring Stuff with Python* (see the materials below).
- During the module, make sure to go through the assigned readings and exercises before each lecture.
- If you get stuck in coding, ask for help on Ed!
  Mastering Python will require practice and hard work, but it's no use being stuck on a problem for hours.

Although the module materials are mostly self-contained, we recommend the

textbook

- *Introduction to Computation and Programming Using Python* (Third edition, 2021), John Guttag, MIT Press.
  - We will cover approximately the first half of the book.

We will assign readings for each session from the Guttag book, but will also point to alternative readings that are freely available online. These will cover roughly the same topics, but may not always perfectly match the book, so we recommend you get the book.

Other Python books

- *Starting Out with Python* (Third edition, 2014), Tony Gaddis, Pearson.
  - Thorough on basic Python topics with lots of exercises.
- *Automate the Boring Stuff with Python*, Al Sweigart.
  - Free online book with lots of practical applications, available at https://automatetheboringstuff.com. Includes great video material too.

Other algorithms books

- *Grokking Algorithms* (2016), Aditya Bhargava, Manning.
  - Probably the best algorithms book to go with this module. Covers many basic algorithms in an intuitive way, though skipping mathematical details.
- *Introduction to Algorithms* (Third edition, 2009), Thomas Cormen et al., MIT Press.
- *The Algorithm Design Manual* (Second edition, 2008), Steven Skiena, Springer.
  - These are in-depth computer science textbooks on algorithms, including mathematical proofs.
- Algorithm Design by Jon Kleinberg and Eva Tardos. Pearson, 2006.
  - Harder than what we will see in this class, but good, e.g., if you would like to study up before a summer internship interview

Structure

The module is divided into ten sessions over ten weeks. Each session consists of self-study online materials and exercises supported by live classes. You will work through each session in three parts:

1. **Preparation** (asynchronous). You will go through preparatory readings,

videos, and exercises.
2. **Live class**. We will discuss and review the key ideas of the session in a live class. A part of the class will be devoted to working on exercises, supported by teaching assistants.
3. **Review** (asynchronous). You will complete a set of exercises to practice and review the learnings from the session.

The details for working through each session are provided in Session details and on the business school Hub. Note that most sessions contain an assessed component to be submitted. See the Assessment section below for details.

The module is very hands-on, with many Python exercises in the sessions and homework assignments. Based on our experience, **you will likely spend more time working on exercises than reading and watching content**. See session details below for indicative timings per session.

If you have any questions on the materials, get stuck working on an exercise, or would like to discuss a topic, the easiest ways to get help are:

- Post a question on the module chat on Ed, which will be regularly monitored by tutors.
- Ask the instructor in a live class. There will be time devoted to open questions in the end of each class.

Lecture schedule

Please see below for session details and preparation instructions.

Session	Live class time	Class type
Session 1: Introduction to Python	16/01/2024	Lecture
Session 2: Algorithms and computational thinking	23/01/2024	Lecture
Session 3: Sequential data	30/01/2024	Review
Session 4: Computational complexity and search	06/02/2024	Lecture
Session 5: Sorting algorithms	13/02/2024	Lecture
Session 6: Python data structures	20/02/2024	Review
Session 7: Data structures and OOP	27/02/2024	Lecture
Session 8: Graphs and breadth-first search	05/03/2024	Lecture
Session 9: Weighted graphs and Dijkstra's algorithm	12/03/2024	Lecture
Session 10: Greedy algorithms; analytics with Python	19/03/2024	Lecture

When the class type is *review*, the *preparation* part of the session is longer

than usual: you are expected to work through Python exercises before the start of the class.

Assessment

Exam (70 %)

The exam will take place at the beginning of the summer term (May 7 to May 14). It will cover both theoretical questions and ask you to write out small segments of Python code on paper as if it were to be compiled on a computer. A mock exam will be made available toward the end of the module, and we will provide a review of the class during the second hour of lecture 10, to help you study.

Session hand-ins (10 %)

Most sessions have a small graded component. They are typically released the day before the session and due the Monday following the session at 9am. You don't need to finish all exercises; you will get full credit for reasonable effort on most of them. In the event that you need help on these tutorials, there will be time during the tutorial to ask for it.

Homework assignments (20 %)

There will be two individual homework assignments intended to recap, explore and combine ideas from the course materials.

Assessment schedule (tentative)

Name	Date assigned	Date due	Results due	Grade weight
Homework 1	23/01/2024	06/02/2024	27/02/2024	10%
Homework 2	13/02/2024	01/03/2024	22/03/2024	10%
Exam	Early May 2024	Early May 2024	TBC (May 2024?)	70%
Tutorial 1	N/A	N/A	N/A	0%
Tutorial 2	23/01/2024	29/01/2024	TBC	*
Tutorial 3	30/01/2024	05/02/2024	TBC	*
No tutorial	06/02/2024	12/02/2024	TBC	0%
Tutorial 5	13/02/2024	19/02/2024	TBC	*
Tutorial 6	20/02/2024	26/02/2024	TBC	*
Tutorial 7	27/02/2024	04/03/2024	TBC	*
Tutorial 8	05/03/2024	11/03/2024	TBC	*

Name	Date assigned	Date due	Results due	Grade weight
Tutorial 9	12/03/2024	18/03/2024	TBC	*
Tutorial 10	N/A	N/A	TBC	*

* Sessions 2-3 and 5-9 will each have a small assessed component. They will be worth 10 % of the grade in total. They will each be worth 1.43%. Note that in week 4 there will not be tutorials on the Friday; we will upload an exercise that you can complete if you like, but it will not be marked.

Assessment marking

After each session, you will submit a small number of exercises, which will be marked for correctness. That is, you will typically implement Python functions, and we will check whether they work correctly with different inputs. The individual homeworks 1 and 2 will be similarly marked for correctness only.

Collaboration guidelines

We strongly encourage you to work together and discuss the course materials and activities with your classmates. In our experience, collaborating with classmates is both a great way to learn and much more fun than working alone.

However, the limit of this cooperation is sharing code on the individual homework assignments. You may discuss ideas and approaches for the homework, but **you should write your own code.** In brief, you should NOT:

- Copy code from your colleagues, or let others see your code
- Look for solutions online, or post your solutions online
- Use GPT-4 or similar large language models to write code

The College takes plagiarism very seriously and it can result in severe penalties. The College's Academic Integrity policy can be found here. Please note that the autograding software we use can run plagiarism checks against both other submissions and online sources.

Week-by-week details

Week 1: Introduction to Python

Back to schedule

We will start introducing basic concepts of the Python programming language: syntax, calculations, variables and assignment, and conditional statements.

Preparation (1h)

1. Install Python on your machine before the first session. There are a few different versions of Python around. We will be using the *Anaconda distribution of Python 3.9*. Download Getting Started Tutorial from module Files on the Hub for installation instructions and guidance on starting out with Python.
2. Complete the readings below and think about how you would explain to your classmates:

- What different ways are there for writing and running Python code on your computer?
- What is a variable? What is the difference between `value = 2` and `value == 2` in Python?
- What is a conditional statement, and for what do we use them?

Readings:

Guttag: Chapters 1, 2.0-2.5.

OR

Sweigart, Al. Automate the Boring Stuff with Python.

- Chapter 1 – Basics https://automatetheboringstuff.com/chapter1/
- Chapter 2 – Control flow (until for loop) https://automatetheboringstuff.com/chapter2/

Lecture (2h)

Introductions. We will discuss the learning goals for the module and how to achieve them and start out with Python.

Review (1h)

1. Work through the rest of session 1. This includes a number of Python exercises to complete on your machine, contained in the session zip file.

Session 2: Algorithms and computational thinking

Back to schedule

We will talk about the role of algorithms and abstraction in problem solving and managing complexity. We will also introduce functions in Python, and start solving computational problems.

Preparation (1h)

1. Work through the screens up to the lecture.
2. Complete the readings below and think about how you would explain to your classmates:

- What is a function?
- Why do we use functions?
- What is the difference between "defining" and "calling" a function?

Readings:

Guttag: Chapters 3.1, 4.

OR

Sweigart, Al. Automate the Boring Stuff with Python.

- Chapter 3 – Functions https://automatetheboringstuff.com/chapter3/

Optional Readings:

Ford, Paul. What is code?

- https://www.bloomberg.com/graphics/2015-paul-ford-what-is-code/

Lecture (2h)

In this lecture we will start devising and implementing algorithms to solve computational problems. We will also discuss the use of functions to manage program complexity.

Review (1h)

1. Work through the mandatory exercises.
2. (Optional) Work through the optional exercises.

Session 3: Sequential data

Back to schedule

We will cover more key Python concepts, including for loops, lists, and string operations. We will also work more on functions.

Preparation (2h)

1. Complete the readings below and think about how you would explain to your classmates:

- How does a `for` loop differ from a `while` loop?
- What does the `range` function do?
- A Python `list` is an *ordered*, *mutable* data *collection* with many *built-in methods*. What do we mean by these terms?

2. Work through the screens up to the lecture.

Readings:

Guttag: Chapters 2.6, 5.1-5.5.

OR

Sweigart, Al. Automate the Boring Stuff with Python.

- Chapter 2 – Control flow https://automatetheboringstuff.com/chapter2/
- Chapter 4 – Lists https://automatetheboringstuff.com/chapter4/

Review Lecture (2h)

In this review lecture, we will recap the Python concepts we have covered so far, in particular those you learned in the main part of this session: for loops, lists, and string operations. As our problems are starting to get more involved, we will also talk about how to debug our code to fix errors.

Review (1h)

1. (Optional) Learn more about functions and work through an application on studying Donald Trump's tweets.

Session 4: Computational complexity and search algorithms

Back to schedule

We will learn to analyse the complexity of algorithms, and apply these ideas to a classic problems of computer science: search. We will also continue to practice looping and functions.

Preparation (1h)

1. Work through the screens up to the lecture.
2. Complete the readings below and think about how you would explain to your classmates:

- If computers keep getting faster, why do we care about computational complexity?
- Why do we not measure complexity simply by the time it takes a program to run?
- What is Big-O notation, and why is it useful for measuring complexity?

Readings:

Guttag: Chapters 11, 12.1.

OR

Miller and Ranum. Problem Solving with Algorithms and Data Structures.

- https://runestone.academy/runestone/books/published/pythonds/index.html
- Sections 3.1-3.3 (Complexity)
- Sections 6.1-6.4. (Search algorithms)

Lecture (2h)

We will discuss the concept of computational complexity and Big-O notation used to describe it. We will apply these ideas to study ubiquitous search algorithms.

Review (1h)

1. Work through the mandatory exercises for the session.
2. (Optional) Work through the optional exercises.

Session 5: Sorting algorithms

Preparation (1h)

1. Work through the screens up to the lecture.
2. Complete the readings below and think about how you would explain to your classmates:

- How do the merge sort and selection sort algorithms work?
- Why is merge sort more computationally efficient than selection sort?

Readings:

Guttag: Chapters 6, 12.2.

OR

Miller and Ranum. Problem Solving with Algorithms and Data Structures.

- https://runestone.academy/runestone/books/published/pythonds/index.html
- Sections 6.6, 6.8, 6.11 (Selection sort and merge sort)

Lecture (2h)

We will study another canonical computer science problem: sorting. We will focus on understanding the logic behind two sorting algorithms, as well as their computational complexity.

Review (1h)

1. Work through the mandatory exercises.
2. (Optional) Work through the optional exercises.

Session 6: Python data structures and libraries

Back to schedule

We will introduce more core elements of Python: dictionaries and sets, file operations, exceptions, and libraries. In the optional part of the session, we will also introduce an important and fun use case of Python: scraping data from the web and interacting with APIs.

Preparation (2h)

1. Complete the readings below and think about how you would explain to your classmates:

- When is a dictionary useful compared to a list?
- When and why do we use exceptions?

2. Work through the screens up to the lecture.

Readings:

Guttag: Chapters 5.6-5.8, 7, 8.2, 9.

OR

Sweigart, Al. Automate the Boring Stuff with Python.

- Chapter 5 – Dictionaries https://automatetheboringstuff.com/chapter5/
- Chapter 8 – Files https://automatetheboringstuff.com/chapter8/
- Chapter 10 – Debugging https://automatetheboringstuff.com/chapter10/

Optional Readings:

Miller and Ranum. Problem Solving with Algorithms and Data Structures.

- https://runestone.academy/runestone/books/published/pythonds/index.html
- Section 6.5: Hashing (how dictionaries work)

Sweigart, Al. Automate the Boring Stuff with Python. - Chapter 11 - Web scraping https://automatetheboringstuff.com/chapter11/

Murray, Scott. Interactive Data Visualization on the Web. - http://chimera.labs.oreilly.com/books/1230000000345/index.html - Chapter 3 only, until Javascript

Berlind, David. What Are APIs and How Do They Work? - https://www.programmableweb.com/api-university/what-are-apis-and-how-do-they-work

Lecture (2h)

In this lecture, we will first review some of the material you learned in the first part of the session, in particular dictionaries. We will then introduce an important and fun use case of Python: scraping data from the web and

interacting with APIs. We will also learn to work with Jupyter Notebook, a browser-based Python interface that combines code with narrative text.

Review (1h)

1. (Optional) Complete exercises on Trump tweets and web scraping.

Session 7: Object-oriented programming (OOP) and data structures; numerical Python

Back to schedule

Everything in Python is an "object". But what are objects? We will introduce the fundamental concept of object-oriented programming, and talk about how this links to the design of data structures. We will introduce key linear data structures, and learn to define our own data structures.

We will also introduce some essential Python libraries: numpy for numerical computing and matplotlib for plotting.

Preparation (1h)

1. Work through the screens up to the lecture.
2. Complete the readings below and think about how you would explain to your classmates:

- What is object-oriented programming, and what are its main benefits?
- What are classes, objects, and instances?
- What do leading underscores signify in Python?

Readings:

Guttag: Chapter 10.1.

OR

Chithul, Swaroop. A Byte of Python.

- Object Oriented Programming
- https://python.swaroopch.com/oop.html

AND (for numpy and matplotlib)

VanderPlas, Jake. Python Data Science Handbook.

- https://github.com/jakevdp/PythonDataScienceHandbook

- Chapter 2

Optional readings:

Rougier, Nicolas. Numpy tutorial.

- http://www.labri.fr/perso/nrougier/teaching/numpy/numpy.html

Lecture (2h)

We will discuss the concept of objects, and how and why we might define our own objects. We will also talk about the workings of linear data structures, lists in particular.

Review (1h)

1. Work through the mandatory exercises.
2. (Optional) Work through optional exercises on the queue data structure.

Session 8: Graphs and breadth-first search

Back to schedule

Graphs are ubiquitous, and efficiently solving graph problems is crucial to companies like Google or Facebook. In this session we will study graph search in unweighted graphs. We will build on our work on algorithms and data structures to study the breadth-first search algorithm and the queue data structure, seeing how the design of algorithms and data structures go in lockstep. We will also apply the algorithm to a real-world social network problem.

Preparation (1h)

1. Work through the screens up to the lecture.
2. Complete the readings below and think about how you would explain to your classmates:

- What are three examples of everyday things that can be represented as unweighted graphs?
- What are assumptions made in the breadth-first search algorithm? What is the output of the algorithm?

Readings:

Guttag: Chapter 14.2.

OR

Kun, Jeremy. Depth- and Breadth-First Search.

- https://jeremykun.com/2013/01/22/depth-and-breadth-first-search/

Moore, Karleigh. Breadth-first search (BFS).

- https://brilliant.org/wiki/breadth-first-search-bfs/

Lecture (2h)

We will discuss the notion of graphs and introduce breadth-first search in a social-network context. We will discuss the algorithm and how it makes use of the queue data structure.

Review (1h)

1. Work through the mandatory exercises.
2. (Optional) Work through the optional exercises.

Session 9: Weighted graphs and Dijkstra's algorithm

Back to schedule

We will continue to work on graphs by studying Dijkstra's algorithm for finding shortest paths in weighted graphs.

Preparation (1h)

1. Work through the screens up to the lecture.
2. Complete the readings below and think about how you would explain to your classmates:

- What are the assumptions made in Dijkstra's algorithm, and what is the output?
- How does the algorithm work?

Readings

Wikipedia. Dijkstra's algorithm.

- https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Abiy, Thaddeus, et al. Dijkstra's shortest path algorithm.

- https://brilliant.org/wiki/dijkstras-short-path-finder/

Hazzard, Erik. Dijkstra's Algorithm - Shortest Path

- http://vasir.net/blog/game_development/dijkstras_algorithm_shortest_path

Lecture (2h)

We will discuss the how Dijkstra's algorithm works as well as its complexity and the importance of choosing the right data structure for the algorithm.

Review (1h)

1. Work through the mandatory exercises.
2. (Optional) Work through the optional exercises.

Session 10: Greedy algorithms; data analytics with Python

Back to schedule

We will finish the course with a brief discussion of the limits of computation. Even with all the computational resources at our disposal some problems turn out to be *intractable*: difficult to solve accurately at large scale. We will talk about intractability and the use of approximations and greedy algorithms. Examples include the knapsack problem and the traveling salesman problem. We will also introduce data analytics using the popular `pandas` library.

Preparation (1h)

1. Work through the screens up to the lecture.
2. Complete the readings below and think about how you would explain to your classmates:

- What are the assumptions of the (0-1) knapsack problem, and what is the output? What is the computational complexity of the problem?
- There is a greedy algorithm to the knapsack problem. What is a greedy algorithm and what are their main benefits and problems?

Readings:

Guttag. Chapter 14.1.

VanderPlas, Jake. Python Data Science Handbook.

- https://github.com/jakevdp/PythonDataScienceHandbook
- Chapter 3

Reda, Greg. Intro to pandas data structures.

- http://www.gregreda.com/2013/10/26/intro-to-pandas-data-structures/

Evans, Julia. Pandas cookbook.

- https://github.com/jvns/pandas-cookbook
- Chapters 1-2.

Optional Readings:

Moffitt, Chris. Common Excel Tasks Demonstrated in Pandas.

- http://pbpython.com/excel-pandas-comp.html

10 Minutes to pandas

- A succinct reference for key tools in the package.
- http://pandas.pydata.org/pandas-docs/stable/10min.html

Rougier, Nicolas. Matplotlib tutorial.

- https://www.labri.fr/perso/nrougier/teaching/matplotlib/

Augsburger, Tom. Modern pandas.

- Advanced material on best practices.
- https://tomaugspurger.github.io/modern-1-intro

Scikit-learn tutorial.

- http://scikit-learn.org/stable/tutorial/basic/tutorial.html

Lecture (2h)

We will discuss intractability and greedy algorithms applied to the knapsack problem. We will conclude with some ideas for next steps with Python.

Review (2h)

1. Work through the mandatory exercises.
2. (Optional) Work through the optional exercises.