CCPS 109 Computer Science I

This course is an introduction to programming and computer science using the Python 3 programming language, taught by Ilkka Kokkarinen for the Chang School of Continuing Education, Ryerson University, Toronto. I do not have an office at Ryerson, so there are no office hours. Most issues can be solved by email or in the lab, and should something come up that could not, we can set up a short meeting in the Chang School student instruction area before the class. You can contact me using my email at `ilkka.kokkarinen@gmail.com`.

Grading (CHANGED MARCH 16):

Take-home Python graded labs. Solving **ten graded lab problems** so that they pass the tester gives you the passing mark 50% for the course, for a letter grade of D minus. After that, every successfully solved lab problem gives you one more point to the course grade, after which the letter grade is awarded according to the official Ryerson conversion table. To get the highest possible grade of A plus of 90%, you therefore need to solve fifty of these 109 lab problems.

Important: the graded labs are effectively a take-home exam, and **therefore all the Ryerson rules for academic integrity are in full effect** for the students officially taking this course in Ryerson! For students taking this course for Ryerson University, **your lab submissions must be fully of your own original work. You may not share actual code with anybody inside or outside the course, although it is perfectly fine to discuss any problem at the level of ideas with other people.**

Material used:

- Official Python Tutorial.
- Python Lecture Notes by Ilkka Kokkarinen.
- Example programs on GitHub (on repl.it).
- Graded lab problems on GitHub (on repl.it).
- CodingBat Python for practice before you move on to solve graded problems.

Reference material to be kept open in another browser tab while you are coding to be consulted as needed, but never to be memorized:

- Python standard library.
- Stack Overflow compilation of Python notes for Professionals.
- PEP 8, the official Python Style Guide.

Doubleplusgood extra reading material heartily endorsed by the instructor for those students looking for further learning:

- Composing Programs in Python
- Python 3 Module of the Week
- Python 3 Tutorial
- Python solutions to Project Euler problems by "nayuki".
- Python tips
- Dan Bader's Python Tips
- Jeff Knupp blog archives
- Peter Norvig's Pytudes
- John D. Cook's categories on Python and Math, for those of you who are inclined towards recreational mathematics. (The math posts usually contain additional Python code to further investigate and numerically confirm the mathematical analysis.)

Software to install:

This course uses **Python 3.7+ everywhere**, and **the older Python 2 will not be able to run most of the examples and labs**. To download and install Python 3 on your own computer, the mainstream way is to go straight to the source www.python.org and get it from there. However, a much better way to install not only the Python environment but a whole host of utilities and additional Python modules would be the Anaconda distribution. It gives you extra goodies such as the Spyder IDE and important Python frameworks such as `numpy` right out of the box, batteries included. (You don't have to create yourself an Anaconda Cloud account to be able to download and use Anaconda locally on your own computer.)

As with most things these days, it is also possible to simply not download and install anything at all, but do all your work inside the web browser with all your data available on the cloud wherever you have a working Internet connection. The excellent site repl.it is used by the instructor to interactively demonstrate the language during the lectures.

Another excellent site Python Tutor allows you to visualize the execution of your Python code one step at a time, making it easier to find out what exactly your function is doing right or wrong during its execution. Many students have found this particular site to be an excellent learning tool to debug their functions that do not return the expected correct results for some parameter values.

Part One: Core Language

The lessons of Part One of this course teach you the fundamentals of Python language and its data types. All final exam questions and graded lab problems can be answered based on only this core language.

Week 1: Interactive and scripted Python

Python interactive environment. Data objects. Python expressions. Integers and floating point numbers. Arithmetic operators. Assigning names to objects. Namespaces. Names as references, aliasing. Types and type conversions. Importing new names from Python modules. Invoking function objects assigned to names. Writing and executing Python scripts. Textual input and output on data.

Examples: `first.py`
Advanced: `referencedemo.py`

Week 2: Functions that make decisions

Writing functions in Python. Indentation in Python syntax. Functions as objects. Parameters and arguments. Keyword and optional arguments. Returning a result from a function. The temporary local namespace created for the duration of function execution. The if-else ladders in Python. Building conditions from comparison operators and logical connectives. Chained order comparisons. Other data types treated as truth values.

Examples: `conditions.py` `timedemo.py`
CodingBat practice: Warmup-1, Logic-1, Logic-2

Week 3: Sequence types string, list and tuple

Python text strings. Unicode characters. Raw strings. Some useful methods defined inside string objects. Heterogeneous sequences in Python. Converting between strings and lists. Sequence indexing and slicing. Handy built-in functions that operate on sequences. Creating numerical sequences with `range`. Creating lists with list comprehensions. Tuples as immutable lists. Multiple assignments with tuple syntax.

Examples: `listdemo.py` `stringdemo.py`
Advanced: `tuples.py` `comprehensions.py` `bytearraydemo.py`
CodingBat practice: Warmup-2, String-1, List-1

Week 4: Iterating through sequences, sets and dictionaries

Iterating through the elements of a sequence with the simple `for`-loop. The powerful data types of sets and dictionaries of Python. Using sets and dictionaries to store previously computed results and other data for fast access later. Reading and processing text from a file.

Examples: `defdemo.py` `setsanddicts.py` `wordcount.py`
CodingBat practice: String-2, List-2

Week 5: Unlimited iteration

General purpose while-loops for repetition for an initially unknown number of rounds. Nesting different types of loops inside each other. Achieving practical goals and solving problems with loops. Breaking out of a loop with `break`, or merely skipping the current round with the `continue` statement. The optional `else`-branch after a loop.

Examples: `mathproblems.py` `cardproblems.py` `hangman.py`
Advanced: `listproblems.py` `primes.py` `exceptiondemo.py`

This week, you should complete whatever CodingBat practice exercises you have remaining so that you feel comfortable enough with the language to start solving our graded lab problems and earning the two points per each correctly solved problem to your course grade.

Part Two: Solving Problems with Python

The lessons of Part Two use the core language features and techniques you learned in Part One to solve interesting problems, and introduce additional advanced language features such as iterators, generators and recursion that make solving many kinds of otherwise complex problems a breeze.

Week 6: Some educational example scripts

Demonstrate the language structures and programming techniques seen in the previous weeks with larger example programs that actually solve real problems. Operations on text. Reading structured data from JSON files into Python lists and dictionaries for computations on that data. Maintaining a dictionary of counters to compute multiple properties of data simultaneously. Reading and writing text files in Python. Creating anonymous functions with lambdas. Sorting lists with arbitrary key functions.

Examples: `stringproblems.py` `countries.py` `mountains.py` `specialnumbers.py`
Advanced: `dissociated.py`

Week 7: Lazy sequences

Iterators as lazy sequences. Iterator operations. Generators. Writing your own arbitrary computational iterators as Python generators. Generator comprehensions. Iterator decorators. Python `itertools` module and its powerful operations for building up additional iterators.

Examples: `generators.py` `reservoir.py`

(Also, you can check out the example implementations of all the neat functions in the `itertools` module, plus the recipes on that page for equally useful additional functions that are not already included in this module.)

Week 8: Recursion and some functional programming techniques

Subdividing problems to self-similar smaller parts. Recursion terminology. Base cases of recursion. Pros and cons of using recursion. Building new functions from existing functions. Currying a function with partial evaluation. Memoization of previously computed results of recursive functions. Other useful function decorators.

Examples: `recursion.py` `functional.py`

Week 9: String processing

Interesting example problems on strings and words drawn from simple computational linguistics, to illustrate the power of the Python core language to solve such problems.

Examples: `wordproblems.py` `morse.py` `wordfill.py`
Advanced: `wordlayers.py` `genetic.py`

Part Three: Special Topics

The contents of these last three lectures are not part of the final exam, nor are they needed to solve any of the graded labs. However, these topics are taught because of their general importance and usefulness. For example, *numpy* and *scipy* and their derivatives such as pandas and scikits are pretty much the real reasons why people from all walks of life use Python to solve problems in the real world.

Week 10: Introduction to numpy, scipy and matplotlib

Accessing the underlying computer and its hierarchical file system with the `sys` and `os` modules. Introspection of the internals of Python virtual machine and objects in the heap memory. The homogeneous multidimensional numerical arrays of numpy. Reshaping numpy arrays. Universal functions in numpy and their element-wise operation on numpy arrays. Some common handy functions. A general overview of the scientific functions of scipy. Processing images and sounds as numpy arrays. Powerful image processing operations of convolution and filtering. Some more pretty fractals rendered as pixel images.

Examples: `sysandos.py` `numpydemo.py` `imagedemo.py` `sounddemo.py`
Advanced: `fractals.py` `matplotdemo.py` `scipywords.py`

For those who go on to do more of your own programming with numpy and scipy, keep the Numpy Reference Manual and Scipy Reference open in some browser tabs for quick lookup of whichever of their multitude of functions you actually need. 100 Numpy Exercises is a nice little cookbook of the clever Numpy functionality, both basic and more esoteric, whereas SciPy Lecture Notes are another good tutorial for those who wish to learn to use this powerful framework.

Week 11: Classes and objects in Python

Rationale of defining your own data types as classes in Python, as opposed to writing functions that operate on low-level representations. Object methods and attributes versus class methods and attributes. The "dunder methods" that make your objects work seamlessly with Python's built-in functions and arithmetic operators. Public and private names inside a class. Python decorators for objects. Extending more specific subclasses from existing classes. Managed attributes implemented with properties. Abstract superclasses and abstract methods. "Monkey patching" an entire class or only one individual object while the program is running.

Examples: `cards.py` `temperature.py` `shape.py`

Week 12: Turtle graphics and fractals

Geometric primitives on the integer plane. Basic relative and absolute operations of turtle graphics. Rendering complex shapes as a series of small line segments. Colours and filling. Creating branching graphical structures with recursion. Iterative and recursive fractal shapes. Subdividing line segments to create interesting fractal paths. (screenshots) Spillover of any previous example programs that were skipped due to lack of time.

Examples: `geometry.py` `turtledemo.py` `circlefill.py`
Advanced: `polysub.py` `paramcurve.py`

Final Exam

Week 13: Final exam (in class, open book, no electronics allowed)

~~The final exam will consist of eight questions, each question asking you to write one function that fulfills the given specification. Of these eight questions, you can solve~~ ~~**any five questions of your choice**~~~~, each of these five worth the same 8 out of the 40 points of your final exam mark.~~

Old final exams: Fall 18 (model answers), Winter 19 (sorry, no model answers for this one), Fall 19 (model answers)