Spring 2013

Q260

Programming for the Cognitive and Information Sciences
Aimed at little or no programming experience.

**Goals of Q260**

Improve your confidence and skills at:

- **Computational thinking:** Writing code.
- **Understanding code.**
- **Understand the abilities and limitations of computation.**
- **Map scientific problems into a computational frame (through to Q320).**
Outline

1. Administrivia
2. Thinking like a computer scientist
3. Why programming for cognitive science?
4. Course Overview
5. Python
Administrivia
Lectures
Monday and Wednesday 2:30pm - 3:45pm
Lindley Hall 030

Labs
Fridays 2:30pm - 3:45pm
Lindley Hall 030

Course website
http://mypage.iu.edu/~edizquie/q260
General references

No required textbook.

Suggested textbook:

Think Python: How to Think Like a Computer Scientist

Other resources on the website.

(If you find additional resources, please let me know so that I can share them with everyone else)
Participation

You are expected to attend.
You are encouraged to participate.
Note we are going to cover a lot of material that is not covered anywhere but in class.

Lectures

There are required readings. Do the readings before class.
The lectures will include computer demonstrations.
The rest of the time will be used to work together through additional problems on the topic.
There will be no lecture notes. Take your own notes.
My job is to help you learn to program. I'll need you to help me teach you. You are encouraged to ask questions always.
Assignments and Exam

There will be assignments given weekly, 7 total. And one final exam.

Assignments will be further developments of the topics covered that week in the class.

Assignments will start off easy and boring. We’ll work up the level of difficulty and interestingness as we go along.

Collaborations. You are welcome to collaborate on assignments with your peers. Each of you must turn in your own assignment. You should state in your report who you collaborated with and what each of your contributions were.

Late work. Assignments get 10% deducted automatically for each day that it is handed in late.
Hand-in procedure

Assignments will be handed via Email. Make sure to follow each of the following rules:

1. The email must be sent to both the AI and the Instructor. The subject of the email must be "Q260 H#" where # is the assignment number. For example, the subject line for the first homework should be: Q260 H1.

2. Text explanations, when necessary, should be sent in either the body of the email or as an attached Word or PDF document.

3. Code for each of the individual solutions must be attached in the email. Name each of the files with the individual solutions with your student ID, followed by the homework number, followed by the solution number. For example, the name of my Python file for solution to the second problem in the first homework will be: edizquiieH1S2.py

4. At the start of each Python file, in a comment, write down:
   1. The identification of the problem (# Homework 1, Solution 2).
   2. Your name (#Name: Ross Ashby).
   3. The number of hours (roughly) you spent on the problems in that part (#Time: 1h30mins).
   4. The names of the people you collaborated with (#Collaborators: Norbert Wiener) and any details about the collaboration.
Grading

- Final exam: 20%
- Homework assignments (10% each): 70%
- Participation: 10%
My job is to help you learn how to think like a computer scientist.

If you are stuck, if you are struggling, if you are not certain about something. Please ask.
Thinking like a computer scientist
What does it mean to do computation?

What is knowledge?

Declarative and Imperative knowledge.

**Declarative** - statement of a fact.

**Imperative** -
Sequence of specific instructions. Do in order. Along the way we do some tests. Depending on the value of the tests, we may change how to proceed.
How can we build a mechanical process to capture that set of computations?

We could build a little circuit to do this computation.

We would need:
Couple of elements that stored values in it.
Some wires to move things around.
Some elements to do addition, division.
Something to do the testing.
Fixed programmed computers

Turing Bombe, 1939.
Electromechanical machine.
Designed to decipher German enigma-machine encrypted signals during WWII.

Atanasoff, 1941.
First electronic digital non-programmable computer.
Designed to solve linear equations.
We want more general problem solvers.

Suppose you can build a circuit that takes as input another circuit diagram. The circuit would wonderfully reconfigure itself to act like that circuit.

A machine that can take a *recipe* - a description of a sequence of steps - as its input. And that machine will now act like what is described in the recipe.
Stored-program computer
(programmable computers)

The analytical engine, 1837.
First general-purpose computer, designed by Charles Babbage.
Never built.

Ada Lovelace.
Known as the first programmer.

ENIAC, 1946.
First electronic general-purpose computer, designed by Mauchly and Eckert from University of Pennsylvania.

They are everywhere!
What would that look like?
A program is a recipe - an algorithm - a sequence of well-defined instructions.

As with cooking, there are a fixed set of primitive instructions.

A good cook with a small set of primitives can create an unbounded number of great dishes.

What are the right primitives? Turing, 1936: with 6 simple primitives, anything can be computed.

Anything that you can do in one programming language, you can do in any other.
To describe a recipe we need a language.

There are many languages. They all have pluses and minuses. There is no best language. Some are more suited to some things than others.
High level vs. Low level (Layers of abstraction)

Easier and faster to write and read. Portable.

High-level languages

- Python
- Matlab
- Mathematica
- NetLogo

Low-level languages

Run faster (if you know what you are doing).

- FORTRAN
- C
- Pascal
- High-Level Language
- Assembly Language
- Machine Language
- Hardware
General vs. Targeted


*C++.* Big projects. Computationally intensive projects.

*Python.* Scripting. Faster development.
Compiled vs. Interpreter
(Processing high-level languages into low-level languages)

Compiled:
- Spends a lot of time analyzing and processing the program.
- The resulting executable is some form of machine-specific binary code.
- Program execution is fast.

Interpreted:
- Relatively little time is spent analyzing and processing the program.
- The resulting code is some sort of intermediate code.
- The resulting code is interpreted by another program.
- Program execution is relatively slow.
The process of designing, writing, testing, debugging, and maintaining the source code of computer programs.
Like mathematicians, you will have to use formal languages to denote ideas.

Like engineers, you will have to design things, assemble components into systems and evaluate tradeoffs among alternatives.

Like scientists, you will have to observe the behavior of complex systems, form hypothesis, and test predictions.

And like chefs,... you will have to be creative, and you will have to develop your own programming style.
Problem solving

The ability to formulate problems, think creatively about solutions, and express a solution clearly and accurately.

Philosophy of mind with a screwdriver

How do I write a program that will help me think about a specific phenomena and understand it better?
Why Programming for Cognitive Science?
“Cognitive science is the interdisciplinary study of mind and intelligence, embracing philosophy, psychology, artificial intelligence, neuroscience, linguistics, and anthropology.”

Stanford Encyclopedia of Philosophy
Programming is used in cognitive and information science in three main ways:

To organize and analyze large amounts of experimental data.

To generate systematic experiments.

To create computational models of cognitive phenomena - to test and validate existing hypotheses, and to generate further predictions.
Science may discover immortality, but it won't happen in the next eighty years.

You'll Never Find a Programming Language That Frees You From the Burden of Clarifying Your Ideas.

But I know what I mean!

You avoid your friend Mike because you're uncomfortably attracted to him.

Nice try, Mike. I get out of the well.

Auw.
Cognition as Computation

Most work in cognitive science assumes the mind has:

- Mental representations analogous to computer data structures,
- Computational procedures similar to computational algorithms.
Connectionist approaches

Inspired by the brain. Simple processing units.

Parallel, distributed, processing. Robust mechanisms.

Often unintuitive solutions to cognitive problems.

Less emphasis on mental representations, and computational process.

Many different levels of abstraction, down to increasing biophysics realism.
Bayesian approaches

Decision-making, generating predictions, reasoning with uncertainty.

Methods of inference that take into consideration probabilities.

Baye’s laws says how we should rationally update our beliefs in the presence of new information.
Embodied, situated, and dynamical system approaches

The brain is not in a vacuum. There's a body and an environment.

Adaptive behavior is the result of the interaction of all three components.

The biomechanical properties of the body are essential.

Also the time and timing of the actions make the difference in the success of behavior.
Course Overview
Part 1: Programming basics

Programming languages.

Python and IDLE.


Programming style.

Algorithmic complexity. Search algorithms.

Dynamic programming.

Object-oriented programming.
Part 2: Simple computational models

Random walks in biology.

Orientation in simple agents.

Self-organization in multi-agent systems (origins of language).

Evolutionary games. (Evolutionary dynamics, drift).

Interfaces. Getting data from people.

Bayesian networks.

Hodgkin-Huxley model.

Optimization algorithms.

Continuous-time recurrent neural networks.
Part 3: Final programming project

What are you interested in?
How can you see programming used?
Python
Even though we will use Python - this course is not about Python.

We will use it to learn to program.

How to design recipes.

Those tools should transfer to any other languages.

By the end of the course, you should acquire the confidence that you can learn to program in any language.
Why learn Python?

- Platform independent.
- Easy and clean syntax.
- Highly readable.
- Rapid development time.
- Scalable.
- Object-oriented.
- Dynamic type.
- Automated memory management.
- Extendable and embeddable.
- Widely used and large community.
- Free and open source.
I learned it last night! Everything is so simple!
Hello World is just print "Hello, world!"

I dunno...
Dynamic typing? Whitespace?
Come join us! Programming is fun again!
It's a whole new world up here!
But how are you flying?

I just typed
import antigavity
That's it?

...I also sampled everything in the medicine cabinet for comparison.

But I think this is the Python.
Debugging
Syntax.
The legal expressions.
"Boy dog clear."
Static semantics. What expressions are meaningful.
"My desk is Susan."
Full semantics. Does it mean what I want it to mean. Will it produce the results that I want it to produce?

0800 Andam started
1000 Andam stopped - Andam
13'06 (033) MP - MC
(033) PRO = 2.130476415
cond 2.130476415
Relays 6-2 in 033 failed special speed test
in relay
Relays changed
in relay
1100 Started Cosine Tape (Sine check)
1525 Started Multi Adder Test.
1545 Relay #20 Panel F
(Moth) in relay.
First actual case of bug being found.
1630 Andam started.
1700 closed down.
Debugging

Syntax. The legal expressions. “Boy dog clear.”

Static semantics. What expressions are meaningful. “My desk is Susan.”

Full semantics. Does it mean what I want it to mean. Will it produce the results that I want it to produce?