Fall 2012

Q530

Programming for Cognitive Science
Aimed at little or no programming experience.

Improve your confidence and skills at:

Writing code.

Reading code.

Understand the abilities and limitations of programming.

Map scientific problems into a computational frame.
Outline

1. Thinking like a computer scientist
2. Why programming for cognitive science?
3. A brief survey of computational models in cognitive science
4. Course overview
5. Administrivia
6. Fundamentals of programming
Thinking like a computer scientist
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.

Like artists,... you will have to be creative, and you will have to develop your own programming style.
Problem solving

How do I write a piece of code that will help me solve a specific problem?

Philosophy of mind with a screwdriver

How do I write a piece of code that will help me think about a specific phenomena and understand it better?
The process of designing, writing, testing, debugging, and maintaining the source code of computer programs.
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.
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!
Before computers there were algorithms.

An algorithm is a recipe - a sequence of well-defined instructions.

There are a fixed set of primitives. (Alan Turing: with 6 simple primitives, anything can be computed.)

Anything that you can do in one programming language, you can do in any other.
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
Cognitive science aims to explain the processes that underlie intelligent behavior:

- How do we categorize objects and relations between them?
- How do we update our knowledge based on new experiences?
- How do brains process information?
- How do we make decisions with uncertain information?
- How does behavior arise from brain-body-environment interactions?
- How can design artifacts that operate under their own control?
A lot of cognitive science is experimental.

But experiments without theory are blind.

Designing, building, and experimenting with mathematical models is central to cognitive science.
Programming is used in cognitive science in three main ways:

To organize and analyze large amounts of experimental data.

To generate systematic experiments.

To create computational models.
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.
Goals of this course:

Provide you with a skill set of basic programming tools, and

An appreciation for how these tools can be used in the study of cognition.
Brief Survey of Computational Models in Cognitive Science
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.
Evolutionary games.
Bayesian networks.
Hodgkin-Huxley model.
Optimization algorithms.
Continuous-time recurrent neural networks.
Part 3: Programming project

What are you interested in?
What is your research on?
How can you see programming used?
Administrivia
Lectures
  Tuesday and Thursday 2:30pm - 3:45pm
  Psychology PY226

Course website
  http://mypage.iu.edu/~edizquie/q530
General references

No assigned textbook.

Resources will be open access.

Main ones will be:

1. Think Python: How to Think Like a Computer Scientist
2. Python Programming.
3. The Python Tutorial.

(If you find additional resources, please let me know so that I can share them with everyone else)
Participation
Attendance will not be graded. You are expected to participate.

Lectures
Sessions will begin with a presentation of the topic. You are expected to have done the readings for that week and participate in a discussion of the topic. The lecture and discussion 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. You are encouraged to ask questions always.
Assignments

There will be assignments given weekly for (roughly) the first 12 weeks.

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

Assignments shall be handed via Email (see details on the assignments).

Assignments will not be accepted late for any reason. Instead, the lowest homework grade will be dropped, so it would be wise to plan on using this drop in the case of illness, emergencies, etc.

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

If you have problems with the homework, seek help.
Final project

The final weeks of the semester will be dedicated to working on your final project on a topic of interest and relevance to your graduate research.

Proposal

500 word abstract proposing how you will use programming to study an aspect of cognition.

Written report

You will be asked to write a short paper (over 1000 words, four or more pages in length, 1.5 spaced, 12 point font) reporting on the motivation for your project, the methods, a justification of the model and the programming, the results, and a discussion. The report should include figures and references. You will also be asked to turn in the programming code, well commented.

Oral presentation

15 minute presentation to share your final project and to get feedback from your peers.
Grading

- **Homework assignments** (5% each, \( \times 11 \), lowest dropped)
- **Final project** (5% written proposal, 15% oral presentation, 15% written report)
- **Participation**
My job is to help you learn.

If you are stuck, if you are struggling, if you are not certain about something. Please ask.
Fundamentals of Programming
Fundamentals of a good program

Reliability.

Robustness.

Usability.

Portability.

Maintainability.

Efficiency.
Layers of abstraction

Easier and faster to write and read. Portable.

High-level languages

Low-level languages
Run faster (if you know what you are doing).
Processing high-level languages into low-level languages

Spends a lot of time analyzing and processing the program.

The resulting executable is some form of machine-specific binary code.

The computer hardware interprets (executes) the resulting code.

Program execution is fast.

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.
Programming paradigms

Non-structured programming.

Procedural or imperative programming (Fortran, Basic).

Structured programming (C, Ada, Pascal).

Object-oriented programming (C++, Java).

Multi-paradigm (Python)
Programming languages commonly used in Cognitive Science


C++. Big projects. Computationally intensive projects.

Python. Scripting. Faster development.
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 antigravity
That's it?

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

But I think this is the Python.
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.
Language structure

**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?
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.