Conclusion

Recap

A summary of topics

- Programming primitives
- Derived programming structures
- Programming-language concepts, design, and implementation
- Programming “Paradigms”
- Software engineering
- Analysis
- Side excursions

Programming Primitives

- Recursion: the all-encompassing repetitive construct; recursive think- ing
- Pairs: A universal data-structuring tool.
- Functions as data values, functions on functions
- Exceptions: Dealing with errors.
- Classes

Derived Programming Structures

- Can build almost anything from primitives.
- Although Python also has specialized implementations of some important data structures.
- Sequences:
  - Lists: traversals, searching, inserting, deleting (destructive and non-destructive)
  - Trees: traversals, binary search trees, constructing, inserting, deleting
- Maps.
- Iterators, generators
- Trees: uses, traversing, and searching.

Programming Language Concepts, Design, Implementation

- Python was developed largely as a teaching language, and is simpler in many ways than other "production" languages...
- And yet, it is a good deal more powerful (as measured by work done per line of code) than these same languages.
- Still, as you've seen, there are problems, too: dynamic vs. static discovery of errors.
- Big item: scope (what instance of what definition applies to evaluation of an identifier). This is what environment diagrams are intended to model.
  - – Alternative: dynamic scoping.
- Implementing a language [CS164]:
  - Interpreters
  - Trees as an intermediate language
  - Relationship of run-time environment representation to scope rules.
  - "Little" languages as a programming tool

Paradigms

- Functional programming: expressions, not statements; no side-effects; use of higher-order functions.
- Data-directed and object-oriented programming:
  - Organize program around types of data, not functions
  - Inheritance
  - Interface vs. implementation
- Declarative programming:
  - State goals or properties of the solution rather than procedures.
  - Regular Expressions: Describe text with patterns; system figures out how to match them.
  - BNF: Describe languages with simple rules; system figures out how to parse them.
  - Syntax-Driven Translation: Hook BNF with rules that produce results. We saw calculators, language translators.

Software Engineering

- Biggest ideas: Abstraction, separation of concerns
- Specification of a program vs. its implementation
  - Syntactic spec (header) vs. semantic spec (comment).
  - Example of multiple implementations for the same abstract behavior
- Testing: for every program, there is a test.
  - In "Extreme Programming" there is a test for every module.
- Software engineering implicit in all our software courses, explicit in CS169.

Analysis

What we can measure when we measure speed:

- Raw time.
- Counts of selected representative operations.
- Symbolic expressions of running time.
- Looking at worst cases simplifies the problem (and is useful).

Application of asymptotic notation ($\Theta(\cdot)$, etc.) to summarizing symbolic time measurements concisely.

Side excursions

- Computability [CS172]: Some functions cannot be computed. Problems that are “near” such functions often cannot be computed quickly.
- SQL [CS186]: A widely used language for accessing and updating databases.
- Prolog: A somewhat extreme example of a declarative programming language involving logical inference.

What's next?

What’s Next (Course-Wise)?

- CS61B: (conventional) data structures, statically typed production languages.
- CS61C: computing architecture and hardware as programmers see it.
- CS70: Discrete Math and Probablilty Theory.
- CSC100: Data Science
- CS170, CS171, CS172, CS174: “Theory”—analysis and construction of algorithms, cryptography, computability, complexity, combinatorics, use of probabilistic algorithms and analysis.
- CS161: Security
- CS162: Operating systems.
- CS164: Implementation of programming languages
- CS168: Introduction to the Internet
- CS160, CS169: User interfaces, software engineering
- CS176: Computational Biology

What’s Next (Course-Wise)?

- CS182, CS188, CS189: Neural networks, Artificial intelligence, Machine Learning
- CS184: Graphics
- CS186: Databases
- CS191: Quantum Computing
- CS195: Social Implications of Computing
- EECS 16A, 16B: Designing Information Systems and Devices
- EECS 126: Probabilty and Random Processes
- EECS149: Embedded Systems
- EECS 151: Digital Design
- CS194: Special topics. (E.g.) computational photography and image manipulation, cryptography, cyberwar.
- Plus graduate courses on these subjects and more.
- And please don’t forget CS199 and research projects.

There’s Also Electrical Engineering

- EE105: Microelectronic Devices and Circuits.
- EE106: Robotics
- EE118, EE134: Optical Engineering, Photovotalaic Devices.
- EE120: Signals and Systems.
- EE123: Digital Signal Processing.
- EE126: Probability and Random Processes.
- EE130: Integrated Circuit Devices.
- EE137A: Power Circuits.
- EE140: Linear Integrated Circuits (analog circuits, amplifiers).
- EE142: Integrated Circuits for Communication.
- EE143: Microfabrication Technology.
- EE147: Micromechanical Systems (MEMS).
- EE192: Mechatronic Design.

What's next? (Otherwise)

- Programming contests
- Hackathons
- More paradigms and languages: the web
- The open-source world: Go out and build something!
- Above all: Have fun!

Fun with Python 🎉🐍

What can you do with Python?

Almost anything!

- Webapp backends
- Web scraping
- Natural Language Processing
- Data analysis
- Machine Learning
- Scientific computing
- Games
- Procedural generation - L Systems, Noise, Markov

*Except you should be careful when you use recursion...

What can you do with Python?

Almost anything! Thanks to libraries!

- Webapp backends (Flask, Django)
- Web scraping (BeautifulSoup)
- Natural Language Processing (NLTK)
- Data analysis (Numpy, Pandas, Matplotlib)
- Machine Learning (FastAi, PyTorch, Keras)
- Scientific computing (SciPy)
- Games (Pygame)
- Procedural generation - L Systems, Noise, Markov

*Except you should be careful when you use recursion...

Web scraping & Markov chains

👉🏽Demo: Composing Gobbledygooks

Web scraping: Getting data from webpages by traversing the HTML.

Markov chain: A way to generate a sequence based on the probabalistic next token.

Further learning: urllib2 module, BeautifulSoup documentation, CS70 and EECS126 for Markov chains

Turtle & L-systems

👉🏽Demo: L Trees!

Turtle: A library for drawing graphics (as if a pen is controlled by a turtle).

L-system: A parallel rewriting system and a type of formal grammar, developed originally by a biologist to model the growth of plants.

Example: Axiom: `A`, Rules: `A → AB`, `B → A`

```
n = 0 : A
n = 1 : AB
n = 2 : ABA
n = 3 : ABAAB
```

Further learning: turtle module, Tutorial: Turtles and Strings and L-Systems, Algorithmic Botany: Graphical Modeling using L-systems, L-system examples

Natural Language Processing

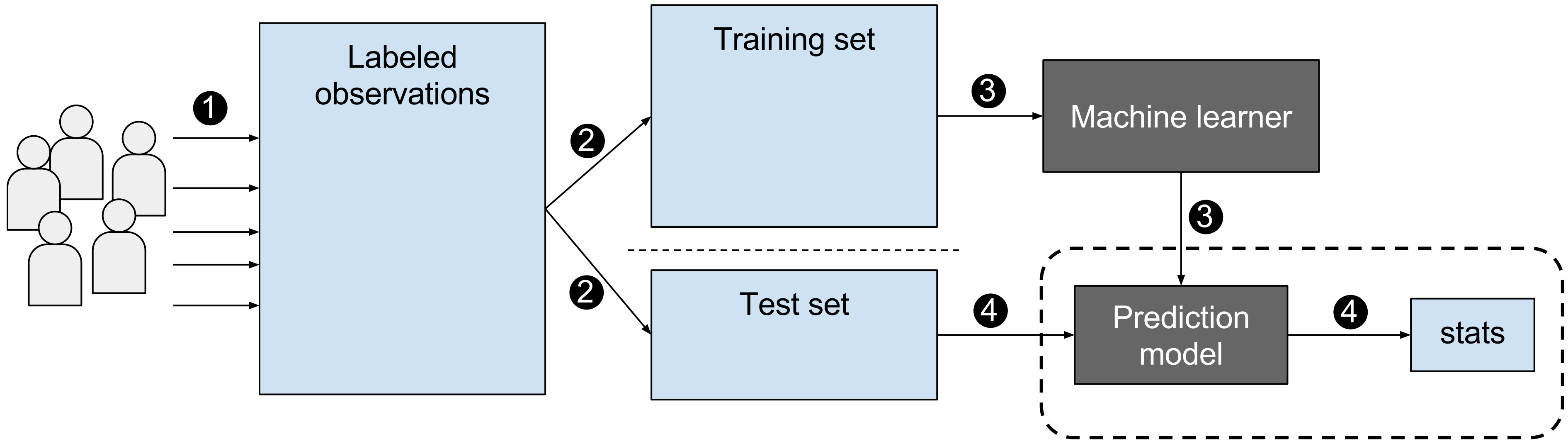
👉🏽Demo: Sentence trees!

NLP includes language modeling, spelling correction, text classification, sentiment analysis, information retrieval, relation extraction, recommendation systems, translation question answering, word vectors, and more.

Further learning: NLTK Book, NLTK Sentiment Analysis, Dan Jurafsky's lectures and books, Berkeley classes: INFO 159, CS 288

Demo: Supervised Machine Learning

👉🏽Demo: Bee vs. Wasp?



Further learning: FastAI Documentation, Kaggle ML tutorial, Bias in ML, Berkeley classes: CS182, CS188, CS189