Conclusion

What did we learn?

Programming paradigms

Imperative programming: using statements to change a program's state.

```
nums = [1, 2, 4]
for i in range(0, len(nums)):
    nums[i] = nums[i] ** 2
```

Functional programming: expressions, not statements; no side-effects; use of higher-order functions.

```
list(map(lambda x: x ** 2, [1, 2, 4]))
```

```
(map (lambda (n) (expt n 2)) '(1 2 4))
```

Programming paradigms #2

Object-oriented and **data-centric** programming.

```
innocent_bee = Bee(5)
horrible_ant = Ant(10)
innocent_bee.fend_off(horrible_ant)
```

```
(define t
    (tree 3
            (list (tree 1 nil)
                (tree 2 (list (tree 1 nil) (tree 1 nil))))))
(map label (branches t))
```

Declarative programming: State goals or properties of the solution rather than procedures.

```
(.+)@(.+)\.(.{3})
```

```
calc_op: "(" OPERATOR calc_expr* ")"
```

Programming concepts

- **Data storage**:
  - Primitive/simple types: booleans, numbers, strings
  - Compound types: lists, linked lists, trees
- **Environments**: rules for how programs access and modify named objects
- **Higher-order functions**: Functions as data values, functions on functions
- **Recursion**: approaching a problem recursively, general recursive patterns
- **Mutability**: mutable objects, mutation operations, dangers of mutation
- **Exceptions**: Dealing with errors
- **Efficiency**: Different programs have different time/space needs

Software engineering

- Abstractions, separation of concerns
- Specification of a program vs. its implementation
  - Syntactic spec (header) vs. semantic spec (docstring).
  - Example of multiple implementations for the same abstract behavior
- Testing: for every program, there is a test.

Software engineering

- Abstractions, separation of concerns
- Specification of a program vs. its implementation
    - Syntactic spec (header) vs. semantic spec (docstring).
    - Example of multiple implementations for the same abstract behavior
- Testing: for every program, there is a test.

Remember: code isn't just read by computers, it's also read by humans.

What's next?

Practice programming

- Programming puzzles (HackerRank, LeetCode, Euler)
- Programming contests (Advent of Code, Kaggle)
- Hackathons
- More paradigms and languages (Web dev, Embedded)
- The open-source world: Go out and build something!
- Personal projects
- Above all: Have fun!

Future CS courses

- 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

Future CS courses #2

- 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.

Plus EE courses...

- 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.

Learn more Python!

- More built-in data types: sets, deques, datetime
- Generator expressions
- Threading, multiprocessing, queues
- Nonlocal/global
- More Python standard library modules: datetime, math, functools, urllib, etc.

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

What can you do with Python?

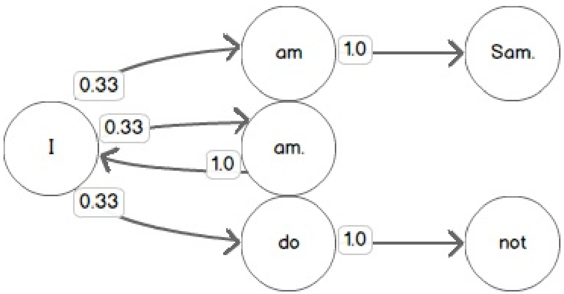
Almost anything! Thanks to libraries!

- 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

Web scraping & Markov chains

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

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



👩🏽‍💻 Demo: Composing Gobbledygooks

Further learning: urllib2 module, BeautifulSoup docs, N-Gram modeling with Markov chains, CS70/EECS126 for Markov chains

Web APIs

API (Application Programming Interface): A way to access the functionality or data of another program.

Web APIs: A way to access the functionality or data of an online web service. Typically over HTTP or via JavaScript.

▯▯ Demo: Movie Mashups

Further learning: urllib2 module, The Movie DB API, ProgrammableWeb

Turtle & L-systems

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
```

□□ Demo: L Trees!

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

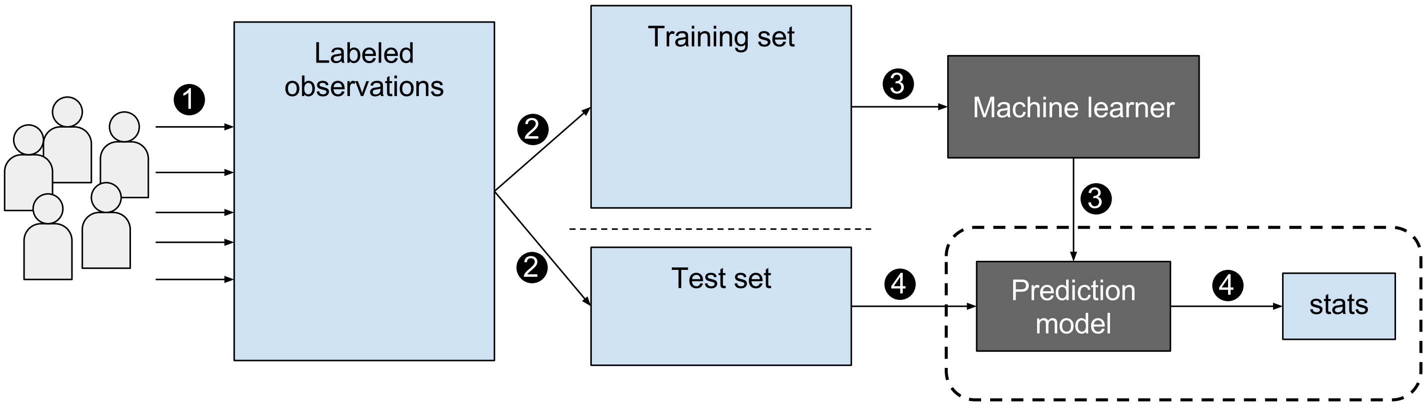
Natural Language Processing

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

□□ Demo: Sentence trees!

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

What do you want to do?

There are so many possible programs that haven't been made yet. What will you make?