~~Bed Bath &~~ Course Review & Beyond

6.1000 LECTURE 26

FALL 2025

Announcements

- Final exam logistics
  - Thursday, Dec 18 at 9 am in Johnson Track
  - Announcement email by end of this week
  - TAs may announce extra practice problems
- Extra office hours
  - Every day before the final
  - See calendar for times
- Scores and grades
  - Pset 7 scores will appear by end of this week
  - Letter grades are determined on an individual basis
- Course evaluations
  - Deadline is Monday, Dec 15 at 9 am
  - https://eduapps.mit.edu/subjeval/studenthome.htm

Today's outline

- High-level thematic review of course material
- In-class problems for discussion
- Next steps in learning programming and computer science

Course Review

Problem sets

- Pset 1 – Encryption cracking
- Pset 2 – Regression and plotting
- Pset 3 – Gerrymandering
- Pset 4 – MBTA simulation
- Pset 5 – Taxi and virus-spread simulations
- Pset 6 – Luggage optimization
- Pset 7 – Climate change and impacts

Objects and variables

- All computation transforms input data to output
- Objects give interpretation to bits in memory
  - **`int`**, **`float`**, **`bool`**
- Objects types are associated with valid operations
- Variables are names that allow us to hold onto objects

Control flow

- A Turing-complete programming language typically supports conditionals and iteration
- Conditionals (**`if`** statements) allow program behavior to react to different input
- Iteration (**`while`** and **`for`** loops) allows programs to handle inputs of indefinite size

- Python debugging tip: pay attention to indentation!

Compound data types

- Often convenient to operate on data as a group
  - examples from psets?
- Python's built-in compound types come and their operations
  - sequences: **`str`**, **`list`**, **`tuple`**, **`range`**
  - unordered: **`dict`**, **`set`**
- With the exception of **`str`**, all store only references to other objects
  - Raises issue of mutability and aliasing
- **`for`** loops support iteration on all of them

Encapsulated programs – functions

- Engineers build complex systems by reusing ideas and functionality
  - So does Nature
- Functions are just **named** programs, specifying **input** (parameters), **output** (return), and **implementation** (body)
- Separate ***defining*** a function from ***running/calling*** a function
- To run a function, need it's own context of names to be safe from overriding other names
- Hence the notion of a **frame** and how it is created and destroyed

Custom data types – classes

- We can achieve much with built-in data types, but we rarely think about what we do on computers in terms of those types
- Classes **group functionality together** to establish what one can do with a **custom-named** type of data
- Every language has its own conventions for syntax and object-oriented design
  - Python: **`self`**, object creation, underscore names

Exception handling – alternative control flow

- As we design more concepts (functions and classes) into our programs, we have more situations to consider at every point of connection
- Conditionals are sufficient for handling, but can get unwieldy
- Exceptions can be a more elegant mechanism for separating intended operation and out-of-specification situations
- Augments the rules of function call evaluation by potentially bypassing normal returns

Graphs and algorithms

- Graphs are a natural and fundamental way to express relationships between data, info, concepts, etc.
- Many problems can be expressed in terms of operating on graphs
  - examples from psets?
- Computation itself can often be modeled as a graph
  - **recursion** inherently forms a **tree** of function calls
  - **dynamic programming** identifies duplicates nodes in a recursion tree
  - compound data types and class hierarchies
- Graph search algorithms form the basis of traversing a graph's structure
  - **DFS, BFS, Dijkstra's**

Simulation and data

- Computational power makes practical certain iterative/recursive approaches to solving problems
  - **exhaustive enumeration** of solution space
- Especially effective for addressing uncertainty
  - randomness in modeling
  - randomness in an experiment's results
- Learning models from data
  - **regression** tools to fit/**train** polynomials to data
  - avoid overfitting with **validation** and **testing**
- Evaluating statistical significance from Monte Carlo simulation outcomes
  - many independent samples allow us to invoke the **Central Limit Theorem** to compute a **confidence interval**
  - **p-values** express likelihood of observed outcome under the null hypothesis

In-class Problems

Relevant sample problems from 6.101 exams

- Fall 2023 Midterm Question 3 – add and multiply polynomials
- Spring 2024 Midterm Question 3 – debug string processing
- Fall 2024 Final Question 11 – function frames, aliasing
- Fall 2023 Final Question 2.2 – class environment diagram
- Spring 2023 Final Question 3 – class inheritance rules
- Spring 2024 Final Question 3 – class environment diagram

Further Learning

Subsequent classes in EECS

- Programming, software skills
  - 6.101, 6.102, 6.104, 6.106
- Algorithms, discrete math, proofs
  - 6.120, 6.121, 6.122, 6.140
- Probability, statistics, machine learning, inference
  - 6.370, 6.380, 6.390
- Computer and systems architecture
  - 6.190, 6.191, 6.180, 6.18x, 6.16x
- More areas mapped out here!
  - https://www.eecs.mit.edu/academics/subject-numbering/

Other majors that depend on programming

- Math, physics
  - 18-C
- Biology
  - 6-7, 6-9, 9, 20
- Engineering
  - 1-12, 3, 3-C, 10-ENG, 16, 22, 22-ENG
- Urban planning
  - 11-6
- Economics, management
  - 6-14, 15-2
- https://catalog.mit.edu/degree-charts/

Projects/careers involving programming and CS

- Software
  - user interface (front-end) vs data processing (back-end)
  - design and architecture vs testing and quality
- Data and learning
  - inference and optimization research
  - robotics: planning, acting, sensing
  - ethics of data collection and usage
- Industries
  - aerospace, power systems, transportation
  - space and underwater exploration, environmental monitoring
  - chip fabrication, medical testing
  - finance, sports analytics

“Out-of-curriculum” programming skills

- Command line, shell scripting
  - Bash shell (Linux), Z shell (macOS), PowerShell (Windows)
- Version control – keep a history (graph!) of your work
  - Git, Mercurial, Subversion
- Virtual machines – run other OSes and programs within them
  - VMWare, Parallels, VirtualBox
  - Windows Subsystem for Linux (WSL), Docker
- Software packaging
  - Python virtual environments (venv), pip
  - OS packaging, apt/dpkg (Debian, Ubuntu), homebrew (macOS), winget (Windows)
- Other programming languages
  - C, C++, Java, Rust, Go
  - JavaScript, TypeScript, Ruby, Swift (iOS), Kotlin (Android)
  - Scheme, Common Lisp, Haskell
- Student-produced IAP course
  - https://missing.csail.mit.edu/

Thank You and Good Luck on the Final!