CS134 Lecture 34:
Wrapping Up
Announcements & Logistics

• **Lab 10** due Wed/Thus at 10 pm

• CS134 Scheduled Final: **Friday, May 17, 9:30 AM**
  • Room: **TCL 123 (all sections)**

• CS134 Review Session before Finals:
  • **Wednesday, May 15, 4:30-5:30 PM**
  • Room: **TCL 123**

Do You Have Any Questions?
Last Time: Sorting Wrap Up

- Discussed efficiency of selection and merge sort
  - You implemented and compared wall-clock time in Lab 10
- If you take CS136, you will see these algorithms and concepts again

![Graph showing time complexity](image)
Today and Friday

- Today we will wrap up the course (first 30 mins):
  - Overview of what we learned
  - Concepts vs programming language: discuss high level differences between Python vs Java, and why your CS134 skills will translate
  - How to do more CS stuff on your own/at Williams
- Last 15 or so mins: course evaluations
- Friday's class plan:
  - Jeopardy style review session!!
  - Form teams with your classmates and come up with team names!
  - CS has a long tradition of bad puns and obscure references...
CS134 in a Nutshell

• We have covered many topics this semester!

• We started out learning the basics of programming, and we used python as our medium to explore these building blocks

• Pre-midterm

  • Types & Operators (int, float, %, //, /, concatenation, etc)
  • Functions (variable scope, return vs print, defining vs calling functions)
  • Booleans and conditionals (if elif else, >, <, ==, not, and, or)
  • Iteration: for loops, while loops, nested loops, accumulation variables in loops
  • Sequences: strings (operators, in/not in, iteration, etc), lists (operators, indexing, slicing, etc), ranges, tuples, lists of lists
  • Mutability and aliasing
  • Built-in python data structures: lists, tuples and sets
CS134 in a Nutshell

• Then we moved on to more advanced CS topics

• **Post-midterm**
  
  • **New data structure**: dictionaries
  
  • **File reading**: with `open(…) as`, processing file lines in a loop
  
  • **Recursion**: recursive methods and classes
    
    • **Graphical recursion** with `turtle` graphics library
  
  • **Classes, Objects, and OOP**
    
    • attributes, special methods, getters, setters, inheritance
    
    • “Bigger” OOP Examples: Autocomplete, Tic Tac Toe, Boggle
    
    • Special methods as well as `sorted()` with optional key argument

• **Advanced topics**:
  
  • Efficiency (Big-O), Linked Lists, Searching and sorting
Takeaway: What is Computer Science?

• Computer science ≠ computer programming!

• Computer science is the study of what computers [can] do; programming is the practice of making computers do useful things.

• Programming is a big part of computer science, but **there is much more to CS** than just writing programs!

• A big part of CS (and CS134) is **computational thinking**

Biggest Takeaway: Computational Thinking

• Computational thinking allows us to develop solutions for complex problems. We present these solutions such that a computer, a human, or both, can understand.

• Four pillars of CT:
  • Decomposition - break down a complex problem into smaller parts
  • Pattern recognition – look for similarities among and within problems
  • Abstraction – focus on important information only, ignore irrelevant details
  • Algorithms - develop a step-by-step solution to the problem

• A computer can performs billion of operations per second, but computers only do exactly what you tell them to do!

• In this course we will learn **learned** how to 1) use CT to develop algorithms for solving problems, and 2) implement our algorithms through computer programs
CS134 Labs: Practice with Computational Thinking

• Labs were designed to make look at real life **commonplace** processes through a computational lens
These Concepts Carry Over

• We used Python as a way to practice fundamentals of CS
  • Decomposition, Pattern recognition, Abstraction and Algorithms
• Programming languages just give us a way to express our logic
  • If the language changes, this expression changes (syntax)
  • But the outline of the solution (the logical steps) stay the same!
• Adapting to a new language is just a matter of getting familiar with its syntax, main structure and quirks
• Let's discuss this through high level comparison of Python vs Java
Java AND Python both share ...

- Both languages support similar building blocks
  - Loops and conditionals (if/else, for loops and while loops)
  - Built-in data types for numbers, booleans, strings, arrays/lists
  - Classes and OOP
  - Function frame model and scope
  - Recursion
  - ...

The ideas we learned in Python carry over to Java, we just need to learn how to express them using new syntax!
Unlike Python, Java is ...

- Java is a **statically typed** language
  - In Java, each variable must specify a type which *cannot be changed*
  - In Python, types are not specified, and a variable's type can change

```
x = 3
x = "hello"
```

```
int x = 3;
string y = "hello";
```
Pros and Cons of Strict Typing

- **Python** is a "Loosey goosey" (technical term: **loosely typed**) language
  - Why good? Makes it easy to get started, less cumbersome / overhead
  - Why bad? Can lead to unexpected runtime errors, Python tries to "overcorrect" type issues whenever possible leading to unexpected behavior

- **Java** is a **strongly-typed** language: all variable types need to be declared at initialization and cannot change types
  - Why good? Can catch most type errors during compilation!
  - Why bad? Makes the code more verbose/requires more "boilerplate"
Example: Python's Loose Types

• Confusingly, Python tries to fix "type mismatches" by doing bizarre things

• Does this look familiar?

```python
>>> word1 = ['hello']
>>> word2 = 'world'

>>> word1 += word2  # calls.append secretly
>>> print(word1)
['hello', 'w', 'o', 'r', 'l', 'd']
```
Unlike Python, Java is ...

- Java is (in many senses) a compiled language
- Java code you write is translated into bytecode
- Bytecode is run in a Java virtual machine
  - There is no REPL (no equivalent of interactive python)
  - The Java virtual machine runs all the code in the "main method"

![JavaFlowDiagram]
Compiling Can Be Helpful

• One consequence of the compiler is that certain type of errors can be found at *compile time*

• This is almost like a round of debugging before there are even any bugs!

```
def add(x, y):
    return x + y
...
add("abc", 3)
```

```
public int add(int x, int y) {
    return x + y
}
...
add("abc", 3)
```
Python vs. Java

**Python**

- Powerful language used by many programmers
- Designed for making common programming tasks simple
- Good for new programmers, and for scientific computing

**Java**

- Powerful language used by many programmers
- Designed for building large-scale systems design
- Good fit for large, scalable reliable software projects

**Neither language is "better" than the other. They are each useful for different things.**
Python vs Java: Hello World

- Python has low overhead to get started
- Java has more overhead upfront (but we'll see why in CSCI 136)

```python
# hello.py
print("Hello, World!")
```

```java
# Hello.java
public class Hello {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}
```
Python vs Java: Running Our Code

- **Python** is an interpreted language: the *interpreter* runs through the code line by line and executes each line; this can also be done interactively!
- **Java** is a compiled language: code must be compiled first (converted to machine code) before it is executed.

```python
# hello.py
print("Hello, World!")

% python3 hello-simple.py
Hello, World!

% python3
>>> print("Hello World!")
Hello World!
```

```java
# Hello.java

public class Hello {
    public static void main(String args[]) {
        System.out.println("Hello, World!");
    }
}

% javac Hello.java
% java Hello
Hello, World!
```
What's Next?

• If this is the last CS course you take, you can use Python to solve real problems!

• A good way to practice is to use Python to accomplish interesting tasks (hobbies, course projects, ...)

• If you take CSCI 136, you will learn to write code that is reusable, maintainable, and scalable

  • More open-ended assignments that focus on design
  • Build your own data structures and learn to identify which data structure is the most appropriate for a given problem
  • Build on Big-Oh discussion and add mathematical rigor
What's Next?

• If you liked coming up with your own algorithms and you enjoyed the "puzzle" aspects of labs, CS 256 is for you!

• How to: apply different algorithmic paradigms and prove that algorithms are correct and efficient

• If you're curious how computers work, how data is represented in memory, how software and hardware interface, CS 237 is for you!

• How to: optimize the practical parts of your program, get the most out of your physical computing resources, become a "hacker"

• If you enjoyed the process of learning python and want to better understand the design choices of the language itself, CS 334 is for you!

• How to: program in different language paradigms and pick the best language for the job (or design your own!)
Takeaways

• You all should be proud of how much you’ve learned!

• Computer Science is all about breaking down the problem and figuring out how to put the pieces together
  • This problem-solving mindset transcends languages/majors, and will help you throughout your life!

• Thank you for your patience and enthusiasm throughout the course

WE MADE IT!
Course Evals Logistics

• Two parts: (1) SCS form, (2) Blue sheets (both online)

• Your feedback helps us improve the course and shape the CS curriculum
  • Your responses are confidential and we only receive anonymized comments after we submit our grades
  • We appreciate your constructive feedback

• SCS forms are used for evaluation, blue sheets are open-ended comments directed only to your instructor

To access the online evaluations, log into Glow (glow.williams.edu) using your regular Williams username and password (the same ones you use for your Williams email account). On your Glow dashboard you’ll see a course called “Course Evaluations.” Click on this and then follow the instructions you see on the screen. If you have trouble finding the evaluation, you can ask a neighbor for help or reach out to ir@williams.edu.
Beyond CS134

• For those interested in continuing on the CS path:
  • Take CS136 or MATH 200
  • Practice Java over summer break: redo CS134 labs in Java
• In general, if you enjoy puzzles and programming, you can practice these skills on your own:
  • Project Euler (Math + CS puzzles)
  • LeetCode (Coding Interview Prep, Python/Java examples)
  • MIT online course: The Missing Semester of Your CS Education
• CS courses as non-majors: can still take CS136, Math 200 etc, winter study courses (Video games, Lida's winter study, etc)