30122

Wrapping Up

Project Questions

Large Files

Secrets

Extra Office Hours

Today: 1:15–3:15

Friday: by appt
& 3–5pm hybrid

Monday: by appt

Topics

Inheritance

Python Ecosystem

Web Scraping

APIs

Databases*

Record Linkage

Visualization

Shell Scripting

Decision Trees*

Data Structures

"Programming as Theory Building"

> "Non-trivial software changes over time. The requirements evolve, flaws need to be corrected, the world itself changes and violates assumptions we made in the past, or it just takes longer than one working session to finish. And all the while, that software is running in the real world. All of the design choices taken and not taken throughout development; all of the tradeoffs; all of the assumptions; all of the expected and unexpected situations the software encounters form a hugely complex system that includes both the software itself and the people building it. And that system is continuously changing."

via https://jenniferplusplus.com/losing-the-imitation-game/

writing code isn't the hard part-- building & updating your model is

This is the thing that AI does not help you with.

This time last year, I was much more impressed with AI, I've read a lot of AI generated code since then, and if I were running a team I'm pretty sure I'd mostly ban it.

Interfaces

Data Pipelines

Data Representation

Abstraction & Interfaces

One of the most important ideas in 121 was that of abstraction.

Abstraction gives us the ability to represent general patterns of behavior, i.e. functions instead of needing to write out specific steps.

By defining our interface, we opened up possibilities for having our code interact with already written code without making changes to that code.

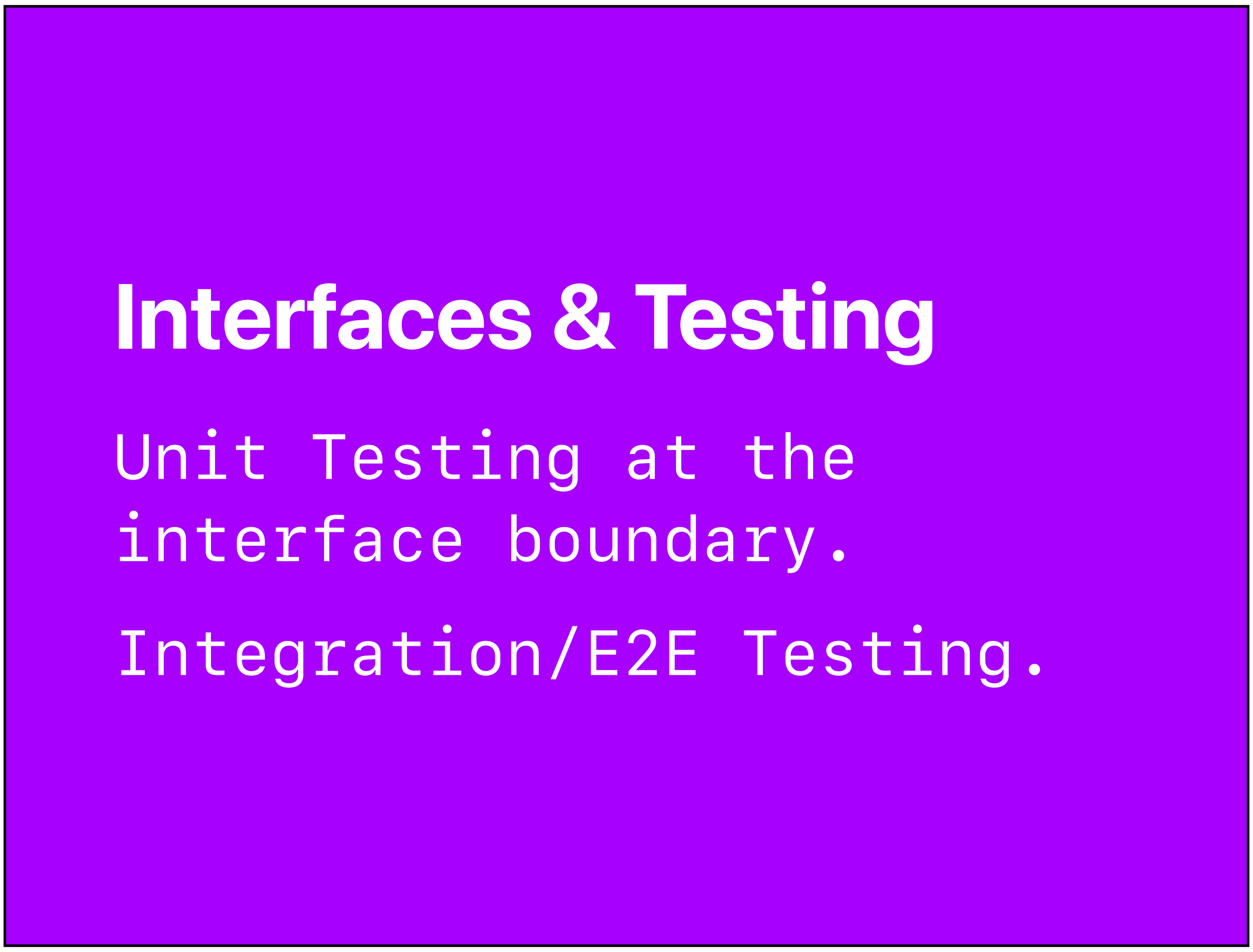
Interfaces & Encapsulation

A good interface hides implementation details.

This means we can work separately, or when we're working on a large project, we can focus on one portion.

Think about PAs like 4 and 5 where you had cleaning steps and processing steps. Hopefully once you finished one step you didn't have to think much about it again.

By breaking the problem into smaller parts, we ensure we can understand the relevant parts.



Interfaces & Testing

Unit Testing at the interface boundary.

Integration/E2E Testing.

Interfaces also form the natural boundary where we can write tests.

Instead of testing that each individual statement is exactly correct, we test that the entire interface (function or class) performs as expected.

Each unit should assume that the units it depends upon are correct. (Assuming they are controlled by us, otherwise we should *distrust* the input.)

Data Representation

Our choices affect the real world.

Being thoughtful about what your "final" data looks like can answer so many questions about what you need to do.

Once you have that, you can work backwards:

unit of analysis lets you know at what granularity you need data

Columns let you know what sources you need and if you are going to need to merge/calculate any of them.

You can also work forwards:

Mock the data.

Work on algorithms.

Key Skills

Decomposition
Debugging
Reading Documentation

These are the things I hope you got better at in this course.

They really form the bulk of what you do as a programmer.

Decomposition

Getting started.

statements -> functions -> (objects?) -> modules -> programs -> systems

Breaking a problem down into smaller parts is the name of the game.

We start by breaking down a problem into statements.

We group those statements into functions.

Debugging

programming errors: type confusion, logic issues, etc.

modeling errors: wrong representation of the real-world problem

As the complexity of our code grows, we lose the ability to look at a function or two and reason out what wrong.

As you've seen, by the time you're working on PA #4-5, TAs and I can't always find your bugs for you. Even more for your projects, as your code increases in complexity and differs more from code we've seen before, only the person that wrote it can effectively debug it.

When we ask you to help walk us through what you've tried, that serves multiple purposes, and it is not solely a pedagogical tool.

The process of debugging is an interactive one, you get better at it by accomplishing those goals.

Reading Documentation

`Learning to learn.`

In Practice

- Changes *fast*. Stable languages/libraries still have ~1 major update each year.
- Enormous range of OSS libraries & tools.
- Medium-size codebase: ~100k lines of code across ~300 files.

Really can't be emphasized how important this skill is in practice.

Technology changes fast.

Novice to Intermediate

— Learn Your Tools

— Read Lots of Code

— Find Community

Beyond just saying "time" and "practice" these are three things I think can really improve your life and skills.

In my experience, these are the people that others call "10x" developers.

Intermediate Python

- Generators
- Exceptions
- Decorators
- Comfort with OO Design
- `itertools`, `collections`, `functools`

https://docs.python.org/3/reference/index.html

https://realpython.com

Python Distilled – Beazley

Beyond Python

javascript
rust
web framework: Flask, Django, HTMX, React
ML framework: PyTorch,
pandas / polars / SQL

Not everyone needs all of these, but some things you might want to start exploring. Having a good subset of these in your toolbox is an incremental investment on top of what you've already done but adds a ton of power to your ability to create.

Pitfalls

- Remember that abstractions are abstractions
- Losing sight of *why* and *for who*
- Programming fads
- Techno-solutionism

Hazards

- Burnout
- Imposter Syndrome

Creativity

You have one of the most powerful creative tools available to you.

You can:

...build a website that can potentially reach thousands or even millions of people.

...combine two or more datasets to make a novel connection or argument.

...automate repetitive tasks to save yourself or others time & energy.

...make art.

Questions

Stay in Touch

Office Hours:
https://people.cs.uchicago.edu/~jturk/

Feedback – CAPP Camp Survey, Course Feedback Survey, Official Survey