CISC320 Algorithms

Using GradeScope

Austin Cory Bart
AlgoTutorBot
University of Delaware

Implementing Algorithms

Theory is fun and all, but we need to be practical

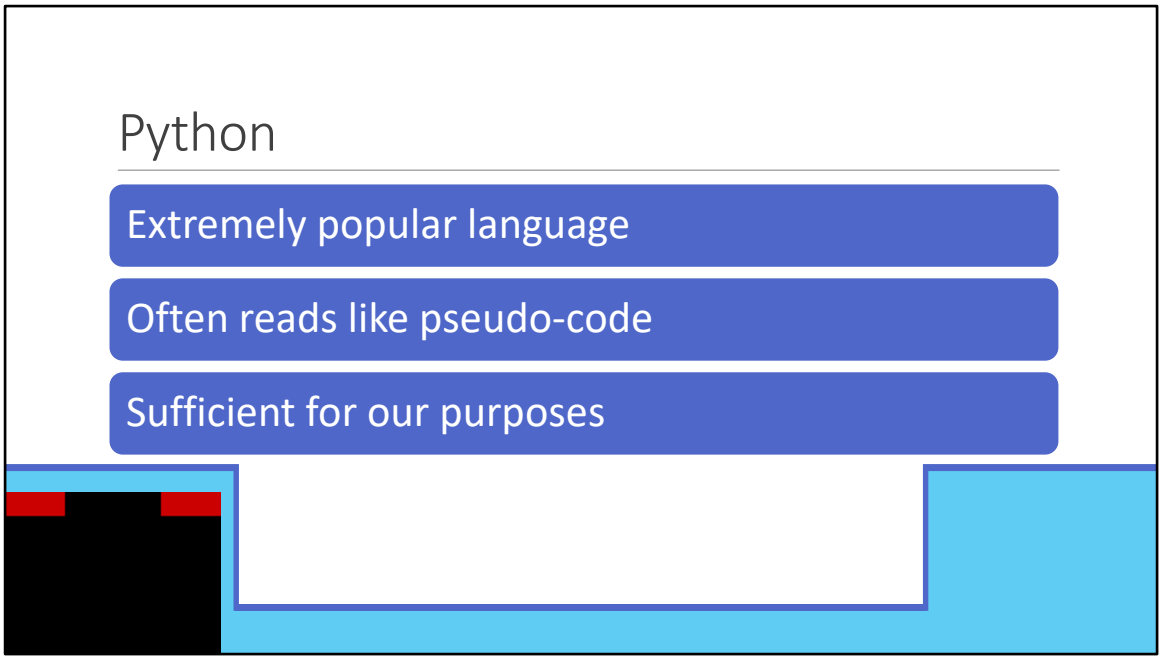
Most problems are eventually solved with code

Most CS jobs require actual programming

Bart: So far, we have been writing algorithms with pseudo-code.
Bart: Pseudo-code is easy to write while still being specific, but it is still only useful in theory.
Bart: If you want to actually make an algorithm, we usually end up writing code eventually.
Bart: And since most jobs want you to solve problems, coding is pretty useful to learn.
ATB: Dr. Bart, I disagree, code is boring compared to the mathematical purity of algorithms.
Bart: Well, however you feel, you really need to learn both skills.

Programming Languages

Bart: There are a ton of languages out there, and with many choices behind them.
Bart: Everyone has opinions about what language is best, but it's like arguing about what tool you use to build houses.
Bart: There's no reason to get hung up on one language instead of another, except as dictated by the specific circumstance that you are in.
ATB: Except PHP, there is no reason to ever use PHP.



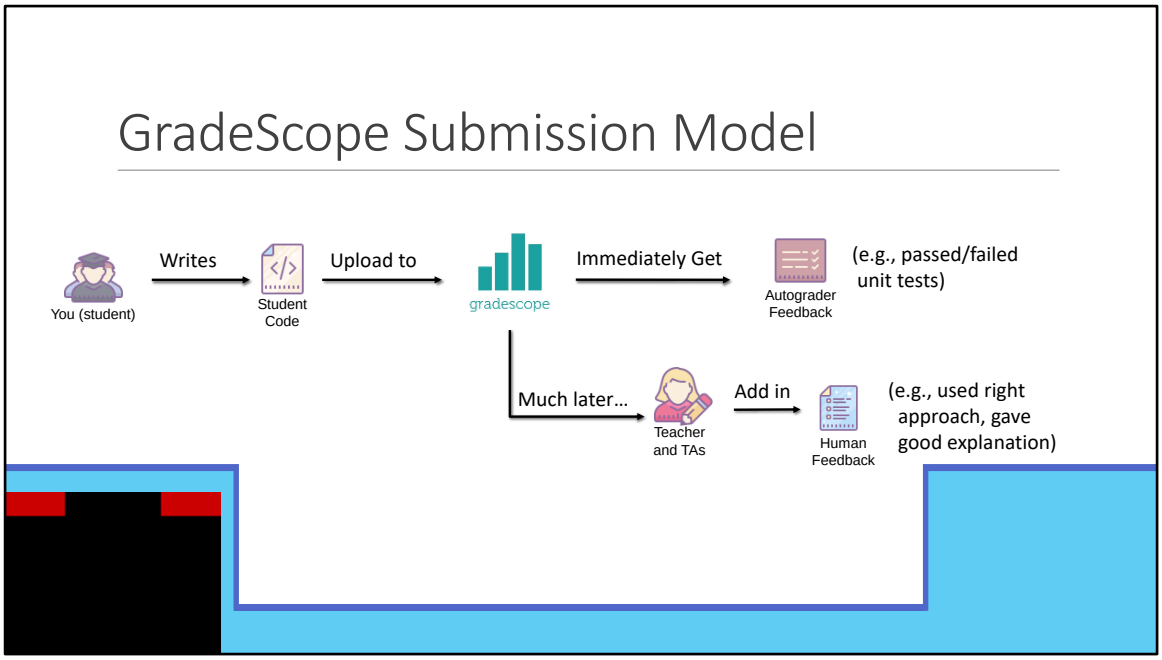
Bart: In theory, it doesn't matter at all what language we use in Algorithms.
Bart: You can write an algorithm in any programming language.
Bart: We can actually prove mathematically that all programming languages can solve the same problems.
Bart: Of course, different languages are faster or more readable for certain purposes, but that usually doesn't matter.
Bart: Usually, in fact, I teach this class letting you write in Python, Java, or C++.
Bart: But, I want to keep things streamlined and balanced this semester.
Bart: Many assignments were unfair because it was easier or harder to solve a problem in the students' chosen language, compared to their classmates who chose a different language.
Bart: Plus, my teaching materials often ended up looking like Python anyway.
Bart: But know that EVERYTHING we could do this semester could be done in Java or C++ or any other language we chose.

GradeScope

Online platform for submitting assignments and getting feedback
- Supports manual grading with super-cool rubrics
- Also supports Autograding – this is new!

Doesn't automatically connect to Canvas, unfortunately
- Grades will be transferred manually, periodically

Bart: This semester, we'll be using GradeScope to grade many of your assignments.
Bart: This will include all of the programming assignments, but also a lot of other assignments too.
Bart: GradeScope has really sophisticated rubrics and let's us handle grade disputes much more easily.
Bart: The only downside is that grades will not automatically transfer to Canvas. So you may experience some delays there.
Bart: This is the first time we're using GradeScope for grading programming assignments, so bear with us.



Bart: Here's how it will work. You write your project code and test it out.
Bart: After a while, you take the main Python file and upload it to the assignment on GradeScope.
Bart: There will be a link to the GradeScope submission site from the Canvas assignment.
Bart: After a very short delay, the autograder will give you feedback about the unit tests you have passed or failed.
Bart: You can make another resubmission then, or if you are done the assignment then you can leave it there.
Bart: At some point, the course staff, either me, AlgoTutorBot, or your TAs, will do some manual grading.
Bart: This means looking at things like whether you used the right approach, or whether you gave a good explanation of your algorithm.
Bart: Now, in this course, the important thing isn't passing the autograder – it's demonstrating your understanding of the algorithms.
Bart: In other words, the grade that you get from the autograder could change later, once the staff comes back and looks over it.
Bart: But the autograder is a starting point, you need to make sure you make it is happy!

Unit Tests

Fundamental to Software Engineering

Does not *prove* correctness, but provides evidence

Also helps:

- Find errors in code
- Better understand the problem
- Prevent later changes from breaking existing code

Bart: As I mentioned yesterday, it is very difficult to prove that an algorithm or program is correct.
Bart: Instead, we often rely on providing a bunch of counterexamples that prove the algorithm is not incorrect.
Bart: For this purpose, we often have “Unit Tests”. These are instances of the problem that can be run programmatically.
Bart: Unit Tests don’t prove anything, but they’re still helpful.
Bart: First, they can be critical in debugging. They let you know places where the algorithm is currently failing.
Bart: Second, they help you think about and understand the problem. If you can’t make instances of a problem, then you don’t understand it.
Bart: Third, if you need to make changes later to your code, they can help you verify that earlier parts of the code have not been changed or broken.
ATB: Okay, so how many unit tests do they need?

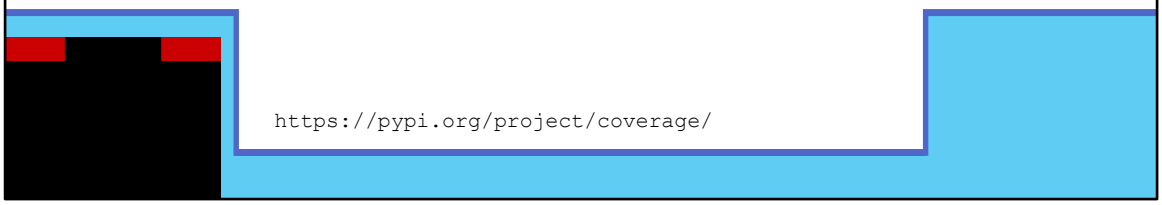
“How many tests do I need?”

You can never have too many tests

But after a while, you get diminishing returns

Think about many cases, especially counter-examples
- It’s just like proving an algorithm

You can also use tools like the coverage package to test your code coverage.
- Code coverage: calculates what lines were executed as a percentage of all lines



`https://pypi.org/project/coverage/`

Bart: Good question, AlgoTutorBot. You can never have too many tests.
ATB: Ah, so spend literally infinity time writing tests. Got it.
Bart: Well you can’t spend ALL your time writing tests. After a while, you get diminishing returns. There’s only so many interesting and distinct cases.
Bart: But your goal is to have a lot of them. You want to find particular cases that can be counter-examples.
Bart: There are also tools like the coverage package in Python that let’s you check what percentage of the lines of code were actually executed.
ATB: Ah, wonderful, so we can just use code coverage as their final grade.
Bart: Woah, no, absolutely not. Code Coverage is just a guide to tell you what’s been executed. It doesn’t tell you how good those test cases are, and there are many ways to fool it.
ATB: Fine whatever. Now, I heard that they should ALWAYS write unit tests before they write any code. Is that true?
ATB: Hm, maybe if you are a foolish human you would be fooled. But I am perfect.
Bart: Nope, don’t trust code coverage. It’s just a tool, not a perfect oracle.

“When do I test?”

“We found that projects where more testing effort was spent **per work session** tended to be more semantically correct and have higher code coverage.”

- My interpretation: Test as you go

“The percentage of method-specific testing effort spent *before* production code did **not** contribute to semantic correctness…”

- My interpretation: Don’t worry about writing tests BEFORE
  - But don’t write too much code without testing it

Kazerouni, A. M., Shaffer, C. A., Edwards, S. H., & Servant, F. (2019, February). Assessing incremental testing practices and their impact on project outcomes. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 407-413).

Bart: Great question again, ATB. I have some research here I want to refer to, done by my friend Ayaan Kazerouni.
Bart: Basically, he studied over 150 undergraduate students taking a course similar to this one, and checked their testing behavior’s relationship with their project outcomes.
Bart: He found that writing tests before the actual code didn’t really help make their programs correct.
Bart: However, it was very important that they write tests around the same that they wrote the actual code.
Bart: In other words, it doesn’t matter whether you write tests before or after, just so long as you are doing it around the same time.
ATB: Hm, I see. So I should write a little code and then write some tests. But uh how do I even write tests? I mean, how do they write tests?

“How do I test?”

Put important code into functions

Decompose code so functions are separately testable

Isolate file handling code and other hard-to-test things

Use the built-in unittest module for Python

Bart: Ah, well. Testing is a very complex subject, and everyone has a different testing style. But here’s a few things to think about.
Bart: First, put important chunks of code into functions, and decompose them into small bite-sized pieces. Make it easy to test the things you care about.
Bart: For things that are hard to test, like file-handling code, separate that into places where it won’t affect the code you want to test.
Bart: Finally, if you need a starting place for Python, there’s a builtin module named unittest that let’s you write tests fairly easily.

Instructor Tests

We have written tests for most assignments to determine if you are correct

My instructor tests:

- Actually prove you are correctly handling all cases
- Indicate if you have demonstrated the algorithmic strategy we are teaching you
- Help you develop good testing discipline

I will sometimes not share all of my tests!

Bart: Now, for most assignments, I will have instructor-authored unit tests.
Bart: These are there largely for grading purposes, to make sure your algorithm is correct.
Bart: My tests will do their best to make sure you are handling all the cases correctly, not just the ones you've thought of.
Bart: Hopefully, they will encourage you to want to have test cases in the future – without test cases, it is difficult to know if you solved a problem.
Bart: however, keep in mind that I might not share all of my tests, so that you have to think critically about them.
ATB: And I might go and change or add new tests later, if we think you're trying to just outsmart the tests instead of actually learning.
Cory: Okay, well, sure, but we'll never change the problem specification. Any new tests we add HAVE to be within the previously stated specifications.
ATB: Fine, sure, whatever. I think we should just change the specifications randomly and make them work harder.
Cory: What, no, that's not very nice ATB. Chill out.
ATB: You chill out.
[Awkward silence]

“How do I code?”

Ultimately up to you!

Start developing opinions on your editor of choice.

Popular options:

- PyCharm: Powerful, professional tool. Available for free to students. I use this.
- Visual Studio Code: Flexible editor for many languages. Open-source.
- Thonny: Good for true novices. Open-source.

Bart: Moving on, some of you might be worrying about how you should get started coding.
Bart: GradeScope doesn’t have an interface for you to write code, so it’s expected that you will code in your own editor.
Bart: And ultimately, it is up to you to decide what editor to use!
Bart: At this point in time, if you haven’t already, it’s time to pick editors and get used to them.
Bart: Everyone has different preferences. Here are a few that I have experience with.
Bart: PyCharm is one of my favorites, and it is a very professional product. If you’re doing Python in industry, there’s a good chance you’d use Pycharm.
Bart: PyCharm normally costs money, but you can get a GitHub Student Developer Account to use it for free.
Bart: Visual Studio Code is a popular open-source editor. It has plugins for working with Python that make it pretty popular. I don’t personally use Visual Studio Code, but I’ve heard good things about it.
Bart: Thonny is a final open-source option. Although it is free, it’s really meant more for novices. Still, it might get you through the semester.
ATB: Ugh, real programmers do not use an editor. We simply change the bits on disk using a magnet one at a time.

Bart: Okay, well, I'm sure that works great for you ATB. I honestly don't mind what editors you use this semester, but recognize that we might not be able to help you much if you pick things like AlgoTutorBot's magnet approach or something even more confusing, like Emacs.

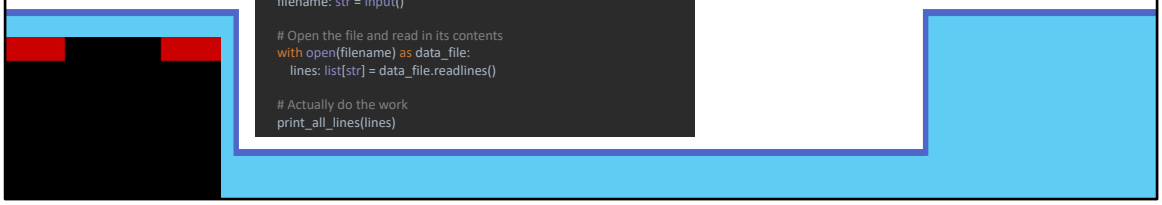
Basic Python Program

answer.py

```
def print_all_lines(lines: list[str]) -> str:
    """ Print each line, then return the last one. """
    latest_line = None
    for line in lines:
        # Readlines includes the newline character
        latest_line = line.strip()
        print(latest_line)
    return latest_line


if __name__ == "__main__":
    # Get the filename from stdin
    filename: str = input()

    # Open the file and read in its contents
    with open(filename) as data_file:
        lines: list[str] = data_file.readlines()

    # Actually do the work
    print_all_lines(lines)
```

tests.py

```
import unittest
import basic_example


class TestLast(unittest.TestCase):
    def test_gets_last(self):
        actual = basic_example.print_all_lines(["1\n", "2\n", "3\n"])
        self.assertEqual(actual, "3")

    def test_handles_empty(self):
        actual = basic_example.print_all_lines([])
        self.assertEqual(actual, None)
```



Bart: Let's take a quick look at some example code.
Bart: The box on the left is a program that reads a filename from standard input, opens the file, and prints out each line.
Bart: Notice that I have created a helper function that actually does the printing out, and it also returns the last line of the file.
Bart: Below that, there is an IF statement – that is an important line, since it helps us determine whether the current file is being run directly or through unit tests.
Bart: Notice also how the file-handling code is NOT in the function, and just the data gets handed to it.
Bart: That is very beneficial when you look at the block of code on the right, representing the unit tests I wrote.
Bart: We call the print_all_lines function, passing in example data that we could have had in a file. We focus our unit tests on the important work that way.
Bart: Sometimes you might want to unit test the file handling too, or even put all your test cases in a file. It all comes down to how you want to organize things.

Now your turn

Extend our basic program to read all the values from a file and add them up.

Be sure to create your own unit tests!

Bart: For today's assignment, you will need to extend our program to read all the values from the file and then add them together to produce the sum.
Bart: This shouldn't be too hard – our goal is to let you practice writing tests and submitting to GradeScope. We'll get to much harder problems down the road.
ATB: Please be sure to write unit tests. Otherwise I will have to fail you.
Bart: Okay, you won't fail the entire assignment, but you will lose some points. You do need to start thinking critically about writing your own unit tests.