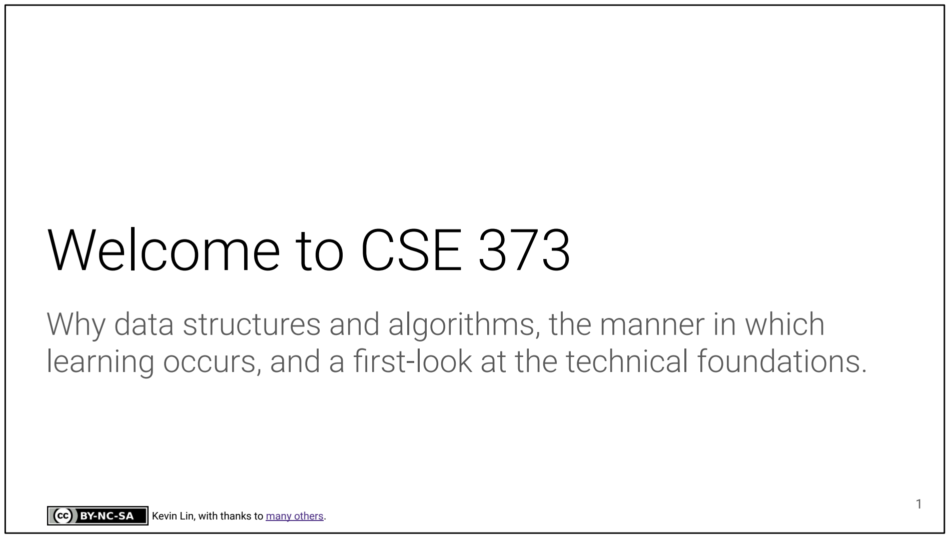
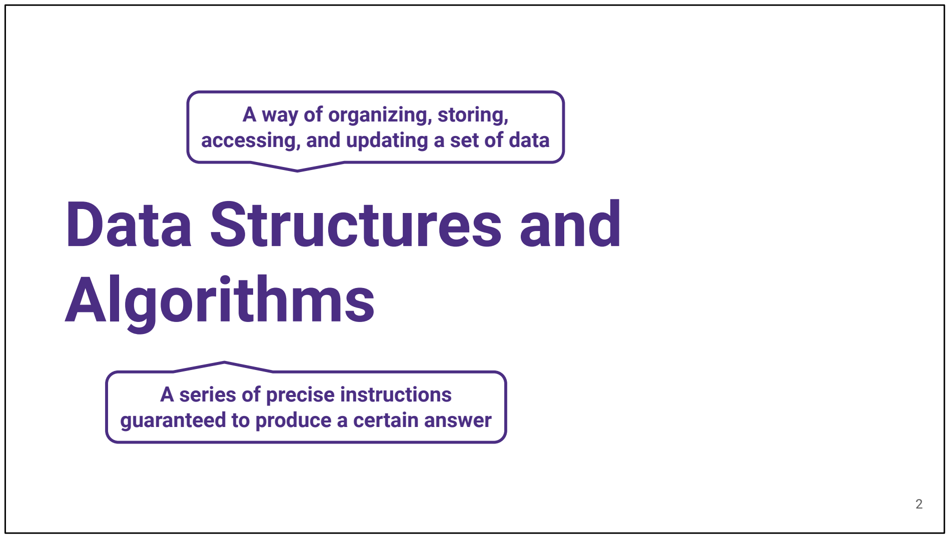


Welcome to CSE 373

Why data structures and algorithms, the manner in which learning occurs, and a first-look at the technical foundations.



A way of organizing, storing, accessing, and updating a set of data

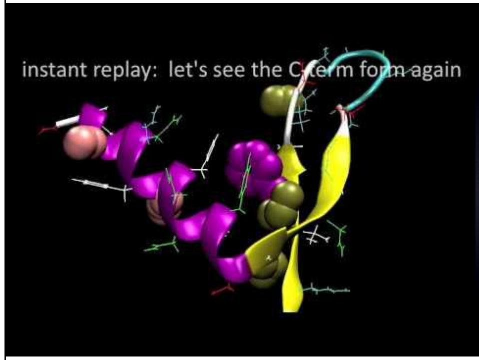
Data Structures and Algorithms

A series of precise instructions guaranteed to produce a certain answer

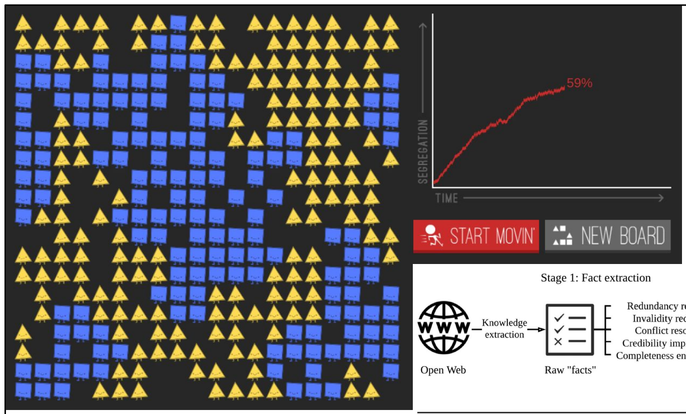
Ask questions anonymously on Piazza. Look for the pinned Lecture Questions thread.

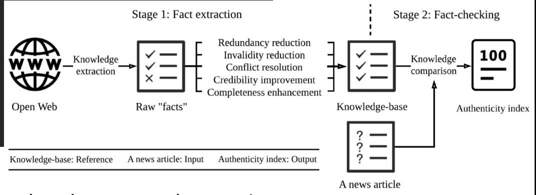
Each week, there will be scaffolded lecture handouts with whitespace for note-taking like this one. Towards the end of this handout, there are some questions associated with each slide to help guide your thinking. These handouts won't be collected, and I encourage you to jot down whatever notes benefit you. I'll suggest some learning practices that work best for this course in about 20 minutes.

Drive current progress (?) in society



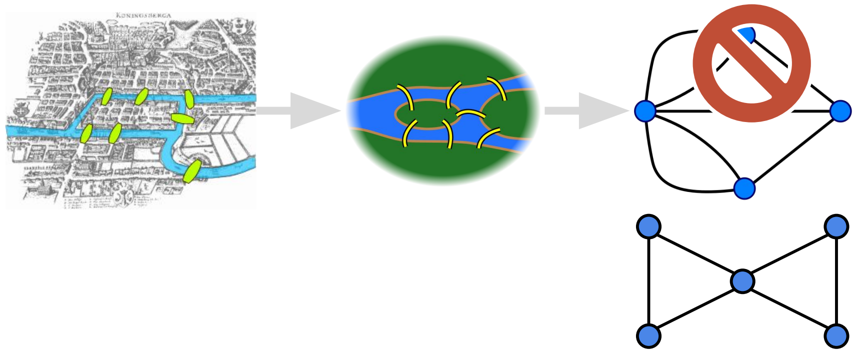
Discover new knowledge and advance the sciences





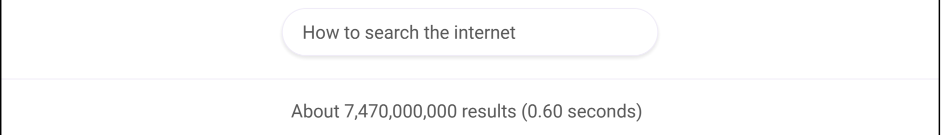
Discover new knowledge and advance the sciences

Parable of the Polygons (Vi Hart, Nicky Case); Fake News: A Survey of Research, Detection Methods, and Opportunities (Xinyi Zhou, Reza Zafarani/arXiv:1812.00315)



Understand different disciplines and problems in CS

Konigsberg Bridges (Bogdan Giuşcă/Wikimedia), Diagram of Seven Bridges (Chris Martin/Wikimedia), Konigsberg Graph (Riojajar~commonswiki/Wikimedia)

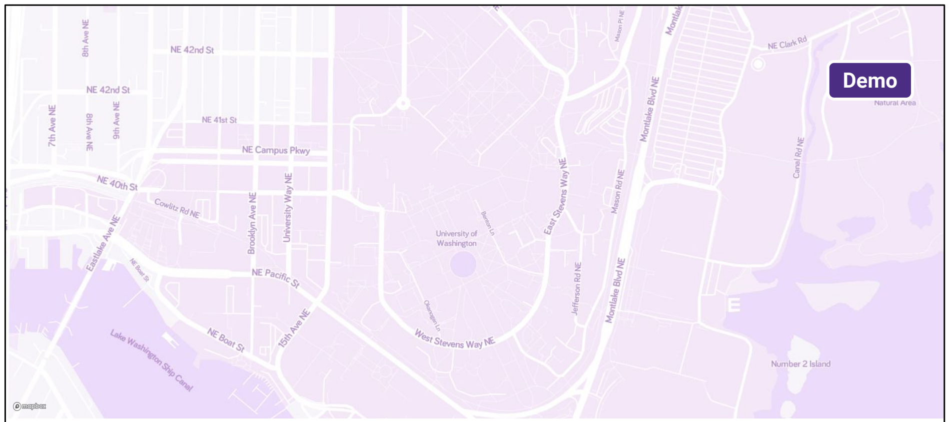


How to search the internet

About 7,470,000,000 results (0.60 seconds)

Support daily life

7

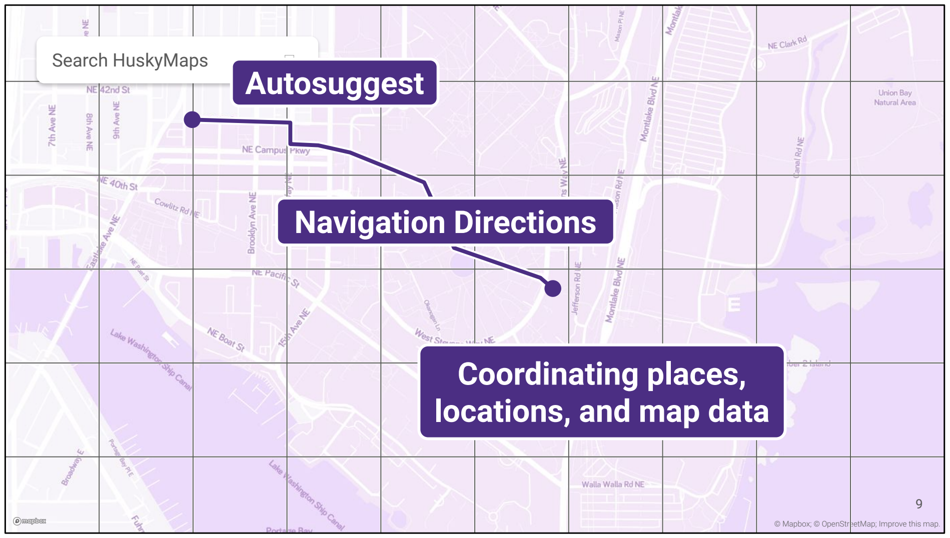


Demo

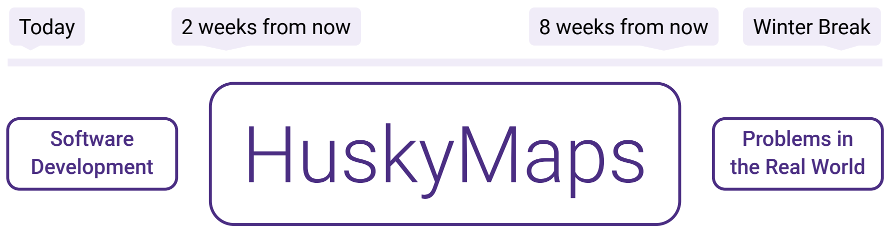
Support daily life

8

© Mapbox; © OpenStreetMap; Improve this map.

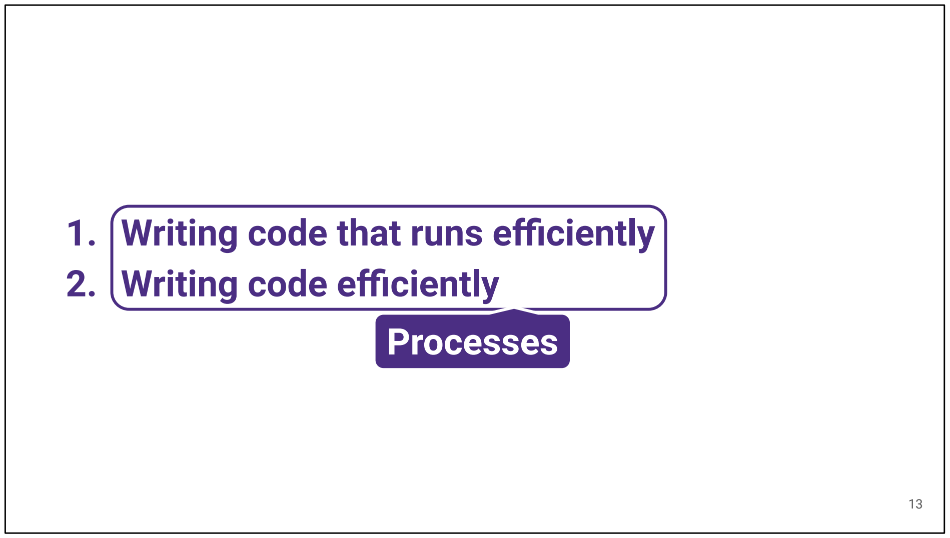

Search HuskyMaps
Autosuggest
Navigation Directions
Coordinating places, locations, and map data
© Mapbox; © OpenStreetMap; Improve this map

1. **Writing code that runs efficiently**
2. **Writing code efficiently**



Course Overview

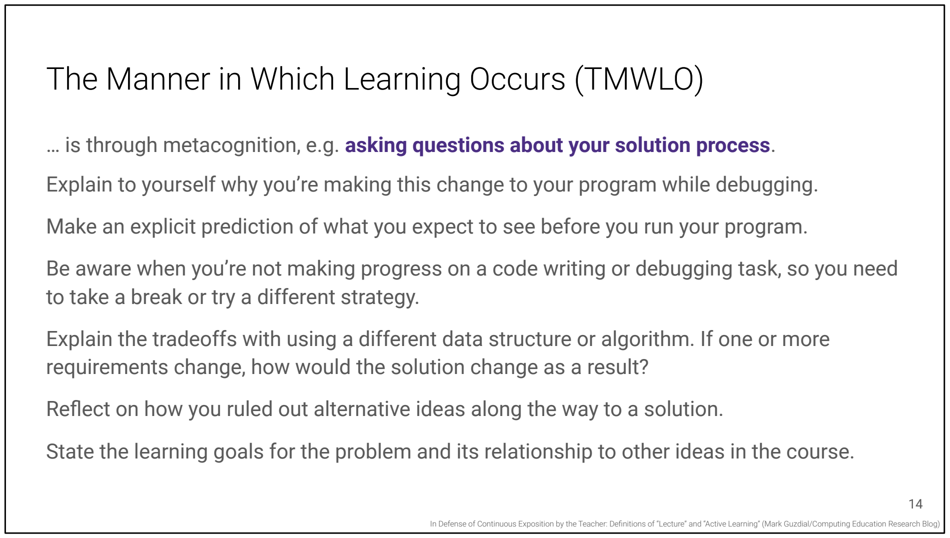
What do you hope to learn in CSE 373?



1. Writing code that runs efficiently
2. Writing code efficiently

Processes

13



The Manner in Which Learning Occurs (TMWLO)

… is through metacognition, e.g. **asking questions about your solution process**.

Explain to yourself why you're making this change to your program while debugging.

Make an explicit prediction of what you expect to see before you run your program.

Be aware when you're not making progress on a code writing or debugging task, so you need to take a break or try a different strategy.

Explain the tradeoffs with using a different data structure or algorithm. If one or more requirements change, how would the solution change as a result?

Reflect on how you ruled out alternative ideas along the way to a solution.

State the learning goals for the problem and its relationship to other ideas in the course.

14

In Defense of Continuous Exposition by the Teacher: Definitions of "Lecture" and "Active Learning" (Mark Guzdial/Computing Education Research Blog)

Here's the note-taking strategy that I think works best for this course (once we get to the technical content).

- Jot down any questions, insights, or realizations that come to mind. See the above questions to get started.
- Jot down the relationships between new ideas and old ideas. How does a new piece of information strengthen, weaken, or otherwise complicate earlier ideas?
- Sketch visualizations of your mental diagrams and small examples. What general principles or patterns seem to govern the visualizations or examples?

I see active thinking and engagement as being the most important part of learning, with note-taking as a means to this end. Give this strategy a try, but do what helps you learn the best.

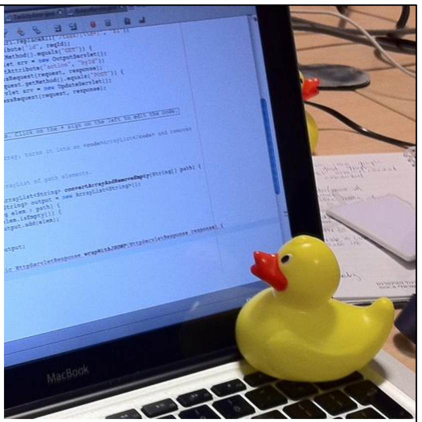
?: How do your own learning strategies (in, say, previous courses) compare to this metacognitive approach?

Real world analogues

Minimal working example

Rubber duck debugging

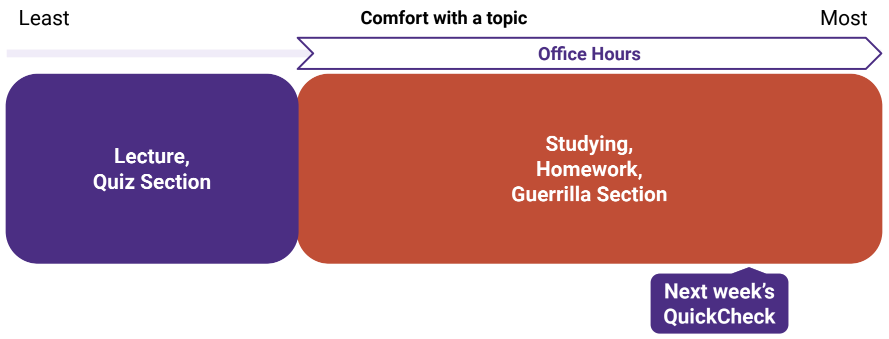
Scientific method



Rubber duck assisting with debugging (Tom Morris/Wikimedia)

Experienced programmers can sometimes seem to solve problems almost intuitively because of how much practice they've had getting unstuck. While they get stuck just as often as you or me, they've exercised their problem-solving muscle enough to have an idea of which debugging strategy to try next. This learning doesn't come immediately: it takes a lot of practice to develop these metacognitive skills.

Basic Learning Workflow



This is the barebones (sad) version of CSE 373. We'll talk about a better learning workflow soon.

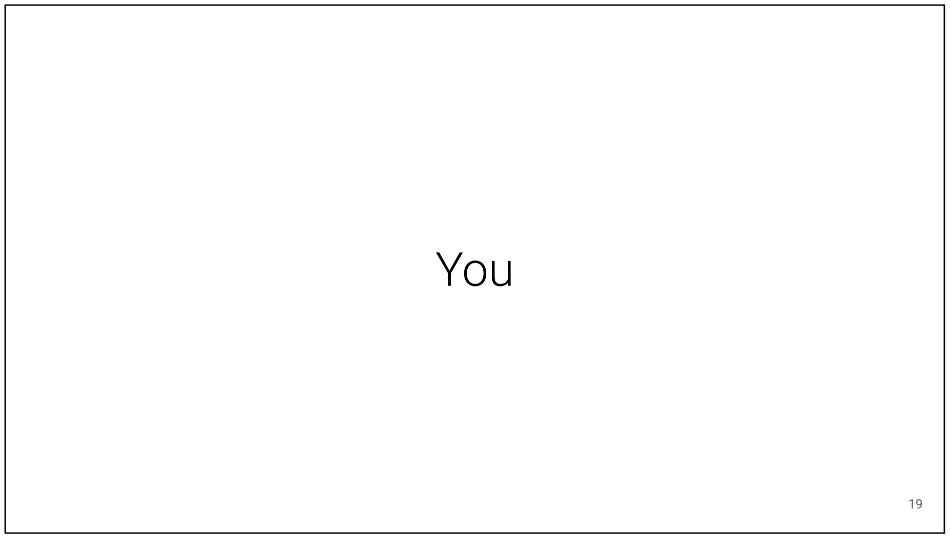
cs.uw.edu/373

Everything is posted to the course website

Course schedule, policies, and staff introductions. The course website is your one-stop shop, but you'll also receive major announcements via Piazza email notifications.

1. Learning
2. Community
3. Course Staff

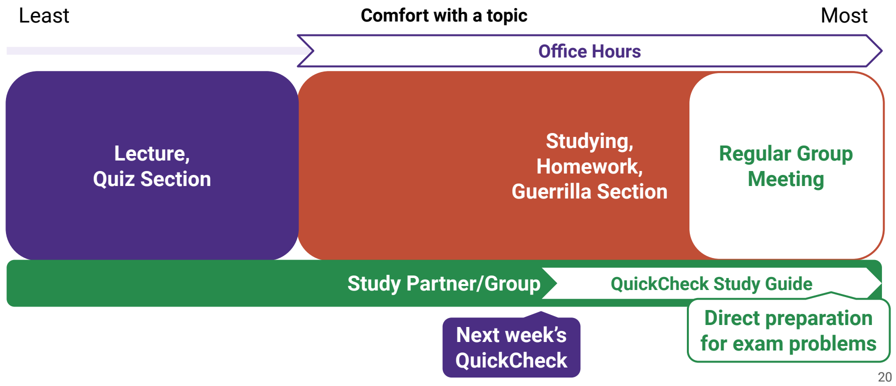
Regardless of whether you want to work in industry, academia, non-profits, or contribute to the world in another way, any project of interesting scale will involve other people. Part of the experience of this course is about engaging productively with members of the course community.



You

The reality is that learning is not so simple that we'll always feel comfortable with a topic even after seeing it in class, working through it in section, and solving problems on the homework. It's important to have a study partner or group for this reason, but also to improve the efficiency of your learning. Efficiency matters because the most precious and limited resource in this course is your 120 hours of attention divided over 10 weeks. It's important to make the most of those hours.

TMWLO, CSE 373 Edition



Protip 1. For the regular group meeting, pick a time that overlaps with office hours so that if you're stuck your entire group can discuss in office hours without having to reschedule.

Protip 2. Office hours around assignment due dates like Tuesday tend to be crowded.

Limits of collaboration

Do not claim to be responsible for work that is not yours.

We really do catch people who violate the rule, because:

- We also know how to search the internet for solutions.
- We use data structures and algorithms to check your work.

All code you submit should be your own work, with a few permissions:

- Receiving significant conceptual ideas towards a solution.
- Using small snippets of code that you find online for solving tiny problems.

These must be cited with comments in your code.

Collaboration is strongly encouraged

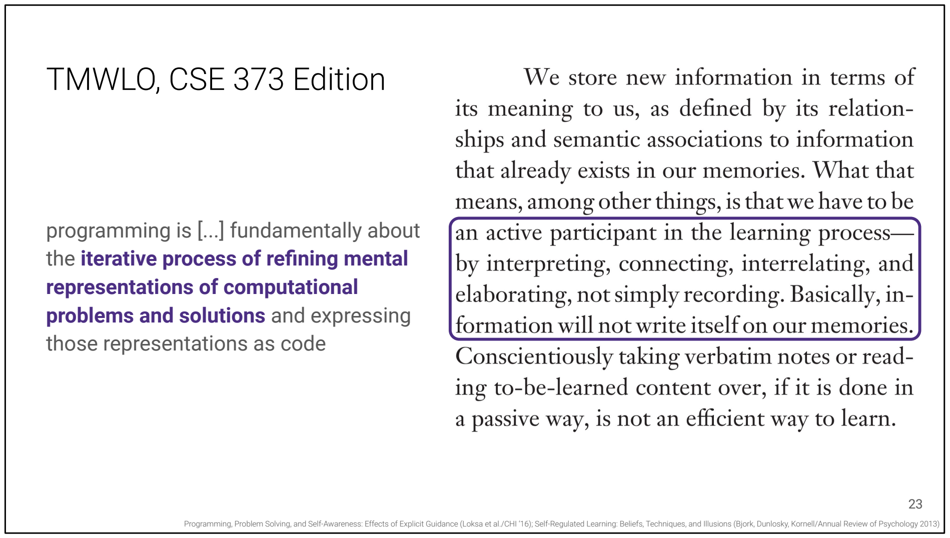
Discuss everything with each other. Teaching is the best way to learn!

Form study groups with your peers in lecture, quiz section, or group study.

Final grades are **not curved**, i.e. they are not based on your relative performance.

Effort	Attending office hours, making progress on every homework, reading Piazza
Participation	Engaging in discussion in lecture or section, asking Piazza questions
Altruism	Helping other students, answering Piazza questions

EPA is optional and can provide a slight grade boost, typically a small percent of your grade.



TMWLO, CSE 373 Edition

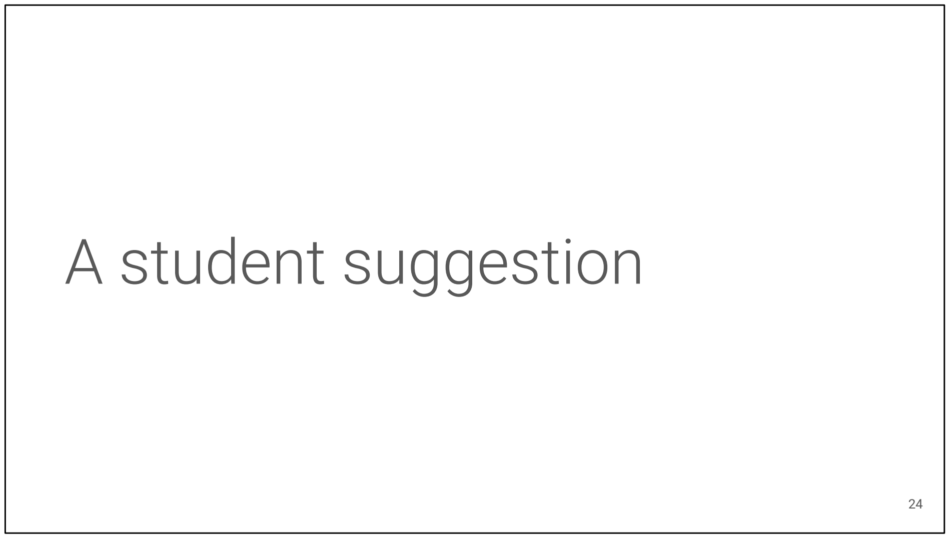
programming is […] fundamentally about the **iterative process of refining mental representations of computational problems and solutions** and expressing those representations as code

We store new information in terms of its meaning to us, as defined by its relationships and semantic associations to information that already exists in our memories. What that means, among other things, is that we have to be an active participant in the learning process—by interpreting, connecting, interrelating, and elaborating, not simply recording. Basically, information will not write itself on our memories. Conscientiously taking verbatim notes or reading to-be-learned content over, if it is done in a passive way, is not an efficient way to learn.

23

Programming, Problem Solving, and Self-Awareness: Effects of Explicit Guidance (Loksa et al./CHI '16); Self-Regulated Learning: Beliefs, Techniques, and Illusions (Bjork, Dunlosky, Kornell/Annual Review of Psychology 2013)

The average human can only hold 7 ± 2 chunks of information in their working memory. One of the meta-challenges for learning in this course is determining what chunks make the most most effective use of that space: which mental representations best model a particular problem. Learning happens when you refine and update your mental representations, so we should embrace the mistakes we make along the way. That's when we know learning is happening.



A student suggestion

24

WORKING WITH A FRIEND. This is in caps because I think it is so so important and worked so so well for me. I want to add that it is best if this friend is about at the same level as you (with respect to grades or intellect or however I should say that). Often, we'd both be stuck on a problem. I've seen people studying alone sit and stare at the solution trying to make sense of it, or post on Piazza. But my friend and I often just went with "well, obviously the answer key must be right. So let's try to figure it out". And we did. We came up with a fool-proof algorithm for how to know if adding a node with change the original MST, came up with tricks to figure out which sorting gets assigned to which column, etc. Also, it's best if this person is someone you're close to. My friend and I often told each other straight up "nah that's definitely not going to work". **I find it difficult being this straightforward with an acquaintance.** Also we weren't embarrassed of throwing completely bizarre ideas at each other, provided we started it off with "I have no idea if this leads to anything, but…"

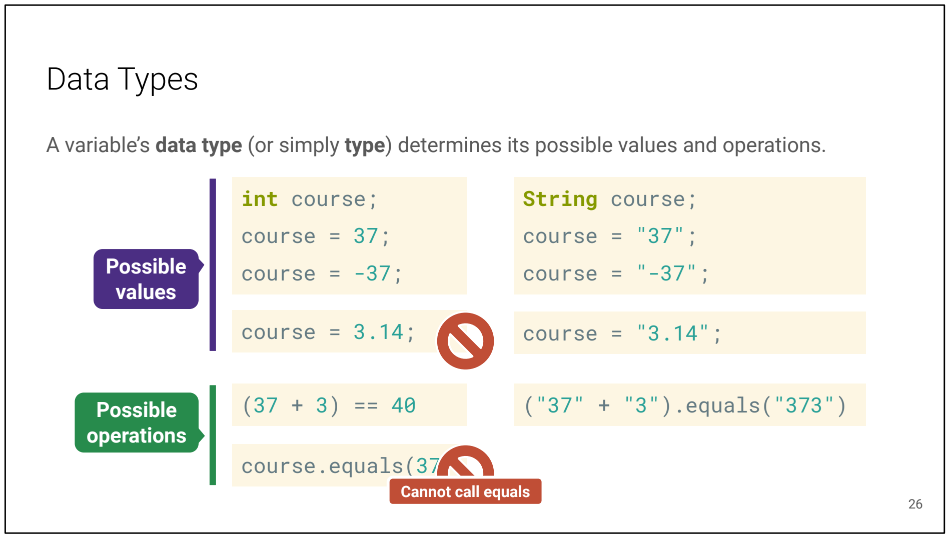
In short, this course is all about asking questions–to yourself, to your peers, to the staff–about what you're thinking, how you arrived at that moment of thinking, and how that thinking complicates your understanding of previous ideas.

Abstract Data Types

25

All programs work with data in one way or another. A function takes input data (arguments) and transforms them into some output data (return value). How we organize this data governs the way we write programs.

There are a lot of questions in the note space for the first few lectures. Think of them as a tool to guide your thinking rather than questions that need to be answered concretely or written down.



Data Types

A variable's **data type** (or simply **type**) determines its possible values and operations.

Possible values

```
int course;
course = 37;
course = -37;
```

```
String course;
course = "37";
course = "-37";
```

```
course = 3.14;
```

```
course = "3.14";
```

Possible operations

```
(37 + 3) == 40
```

```
("37" + "3").equals("373")
```

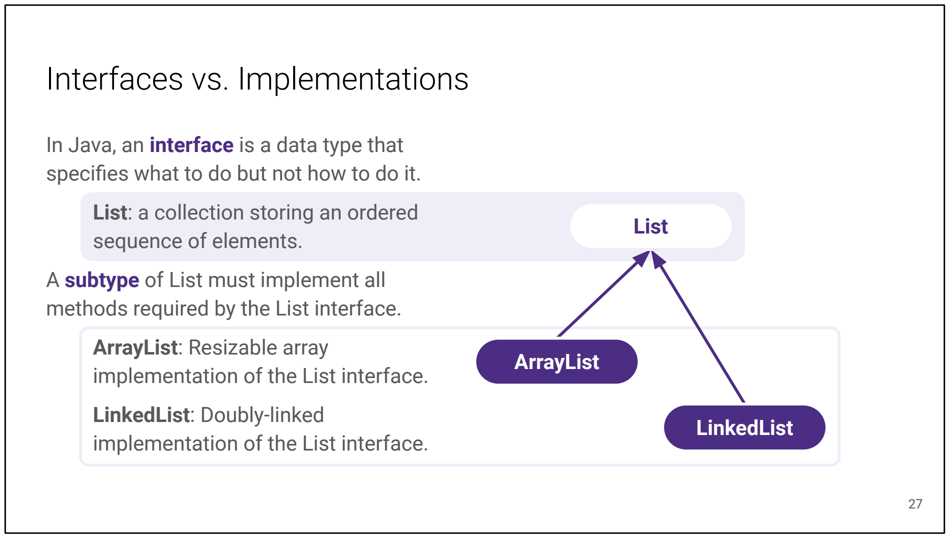
```
course.equals(37
```

Cannot call equals

26

?. What is an example of an impossible value for String?

?. What is an example of an impossible operation for String?



Interfaces vs. Implementations

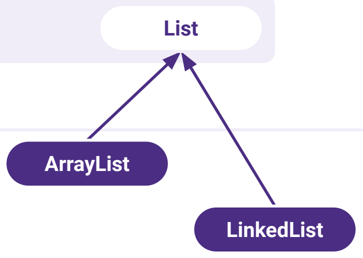
In Java, an **interface** is a data type that specifies what to do but not how to do it.

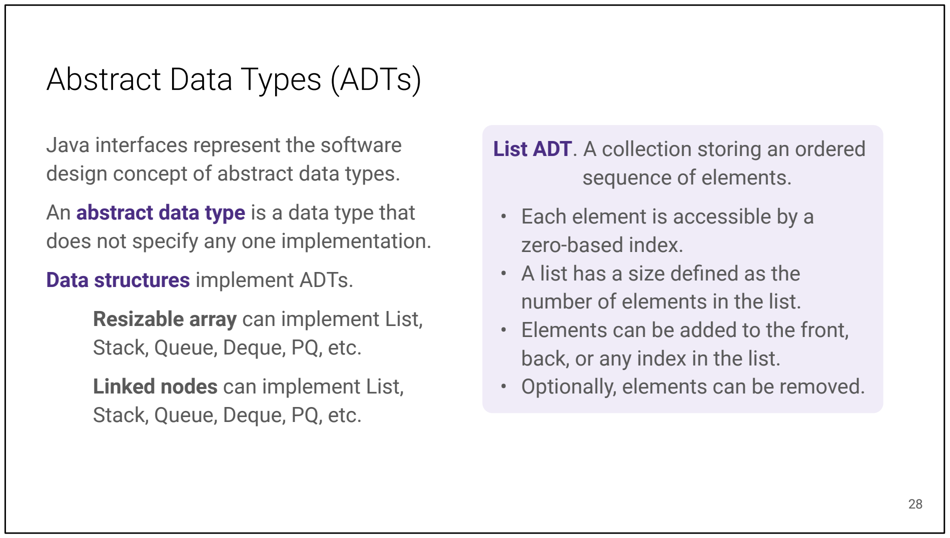
List: a collection storing an ordered sequence of elements.

A **subtype** of List must implement all methods required by the List interface.

ArrayList: Resizable array implementation of the List interface.

LinkedList: Doubly-linked implementation of the List interface.





Abstract Data Types (ADTs)

Java interfaces represent the software design concept of abstract data types.

An **abstract data type** is a data type that does not specify any one implementation.

Data structures implement ADTs.

Resizable array can implement List, Stack, Queue, Deque, PQ, etc.

Linked nodes can implement List, Stack, Queue, Deque, PQ, etc.

List ADT. A collection storing an ordered sequence of elements.

- Each element is accessible by a zero-based index.
- A list has a size defined as the number of elements in the list.
- Elements can be added to the front, back, or any index in the list.
- Optionally, elements can be removed.

Data types determine possible values and operations.

?. What differentiates interfaces from int and String data types?

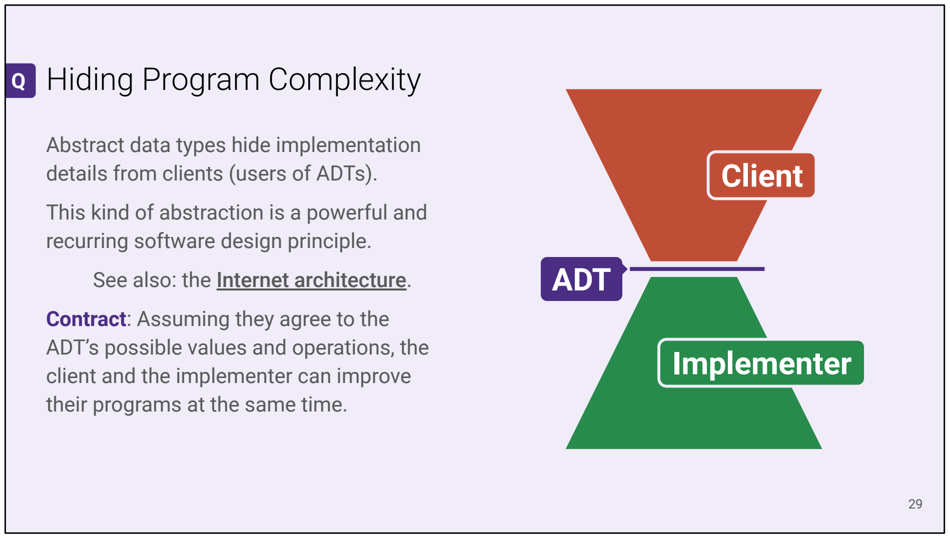
?. In Java, how do we declare that ArrayList is a subtype of the List interface?

?. What does Java do to check that ArrayList is indeed a subtype of the List interface?

?. What's the difference between interfaces and ADTs?

?. What are the practical benefits of separating behavior (ADT) from implementation (data structure)?

?. Why not just use ArrayList all the time?



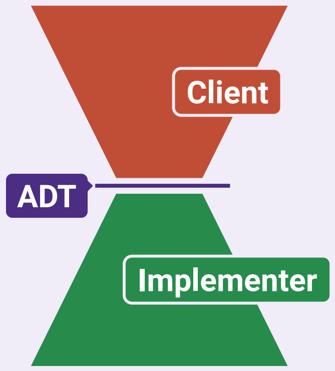
Q Hiding Program Complexity

Abstract data types hide implementation details from clients (users of ADTs).

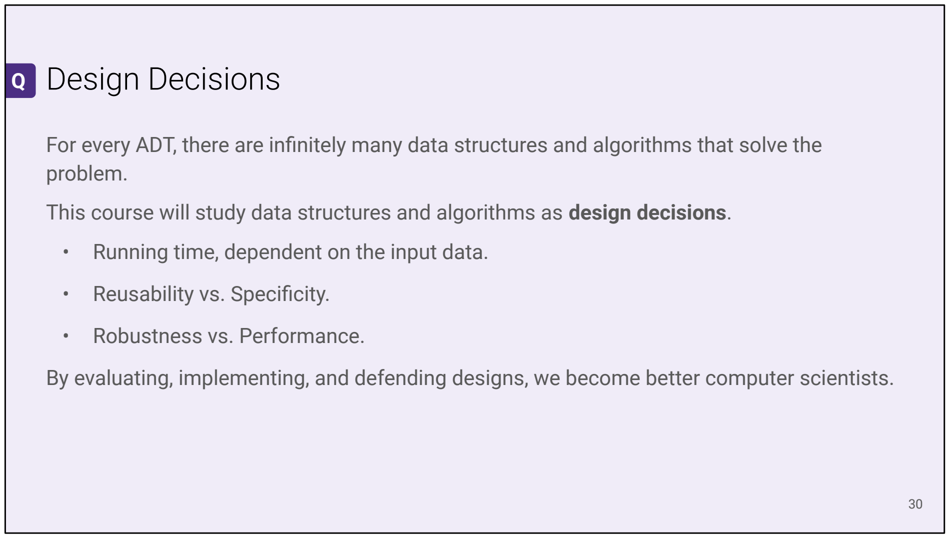
This kind of abstraction is a powerful and recurring software design principle.

See also: the Internet architecture.

Contract: Assuming they agree to the ADT's possible values and operations, the client and the implementer can improve their programs at the same time.



29



Q Design Decisions

For every ADT, there are infinitely many data structures and algorithms that solve the problem.

This course will study data structures and algorithms as **design decisions**.

- Running time, dependent on the input data.
- Reusability vs. Specificity.
- Robustness vs. Performance.

By evaluating, implementing, and defending designs, we become better computer scientists.

30

The area of each shape in the diagram represents the relative complexity.

Q1. Describe an (imaginary) scenario where the contract does not hold. What are the consequences of breaking the contract?

Q2. Are there times when it would be useful to know the implementation details of an ADT's values or operations? Why?

?. Beyond simply computing the correct result, what criteria make one program any better or worse than another program?

Practice with the List ADT coming up in section.

Extra Design Question. Dub Street Burgers is implementing a new system for ticket (i.e. food order) management. When a new ticket comes in, it is placed at the end of the line of tickets. Food is prepared in about the order requested, but some food orders take less time to prepare than others. As a result, some tickets may be fulfilled earlier than other tickets.

Let's represent tickets as a list. Should we use an ArrayList or a LinkedList? Why?