LEC 00

CSE 123

Welcome & Syllabus

sli.do #cse123

Raise hand or ask questions here!

BEFORE WE START

Talk to your neighbors:

Introduce yourself to your neighbor!

What is your name? Major? What have you been up to the past week?

Instructor: Ziao Yin

TAs: Nichole Packard Trien Eeshani Chris

Lecture Outline

- Introductions
- About this Course
  - Course Components & Tools
  - Making the Most of this Class
- Assignments and Grading
- Comparable

Course Staff

- Instructor: Ziao Yin

- Teaching Assistants: 5 Terrific TAs!
  - Available in section, office hours, and discussion board
  - Invaluable source of information & help in this course

- We're excited to get to know you!
  - Our goal is to help you succeed ☺

What is this Class?

CSE 121 – Computer Programming I

- Data types (int, String, boolean)
- Methods / Functions
  - Parameters, Returns
- Control structures
  - Loops, Conditionals
- Arrays & 2D arrays
- **Computational Thinking**
  (language agnostic)

CSE 122 – Computer Programming II

- Functional Decomposition
- File I/O
- Using data structures
  - List, Stacks / Queues, Sets, Maps
- Object Oriented Programming
  - Interfaces

CSE 123 – Computer Programming III

- Advanced Object Oriented Programming
  - Comparable, Inheritance/Polymorphism, Abstract Classes
- Implementing data structures
  - ArrayLists, LinkedLists, Trees
- Recursion
- Critical analysis of design

Why 123?

1. To solve more complex problems by leveraging more complex programming structures / patterns

2. To better rationalize specific design decisions
   - How to “best” structure programs
   - Which data structures are “most” appropriate to use

Be a better programmer

3. To understand and critically analyze intersections between Computer Science and society
   - Search engines, algorithmic art, machine learning, etc.
   - Developing informed opinions on current issues

Be a better person

Prerequisite Knowledge

- Comfort with control structures
  - loops, conditionals, methods/functions
- Experience with using basic data structures
  - arrays, lists, sets, maps
- Experience with console and file input/output
- Exposure to simple object-oriented programming
  - classes, interfaces
- Programming experience *in Java*
  - Or willingness to *pick it up on your own*

What do you want to get out of this course? Why brings you to this course?

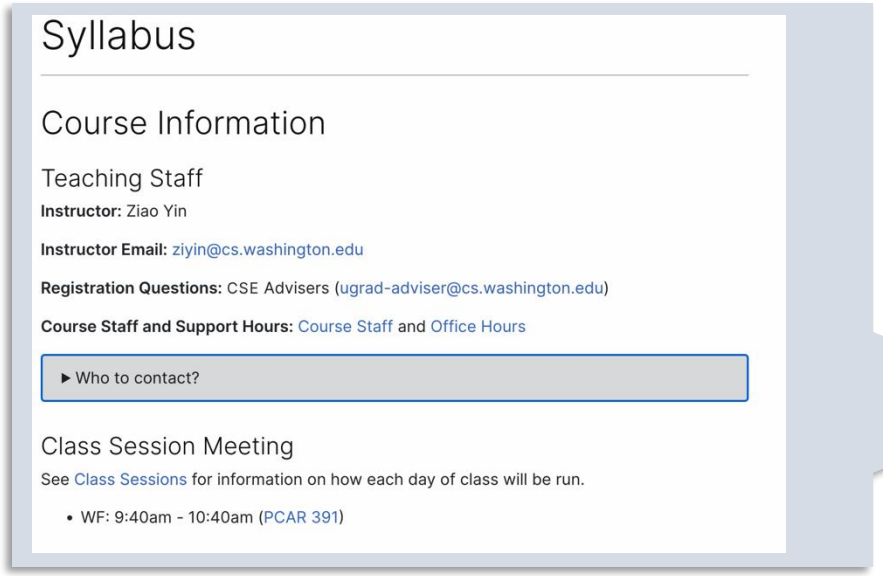
sli.do #cse123

Lecture Outline

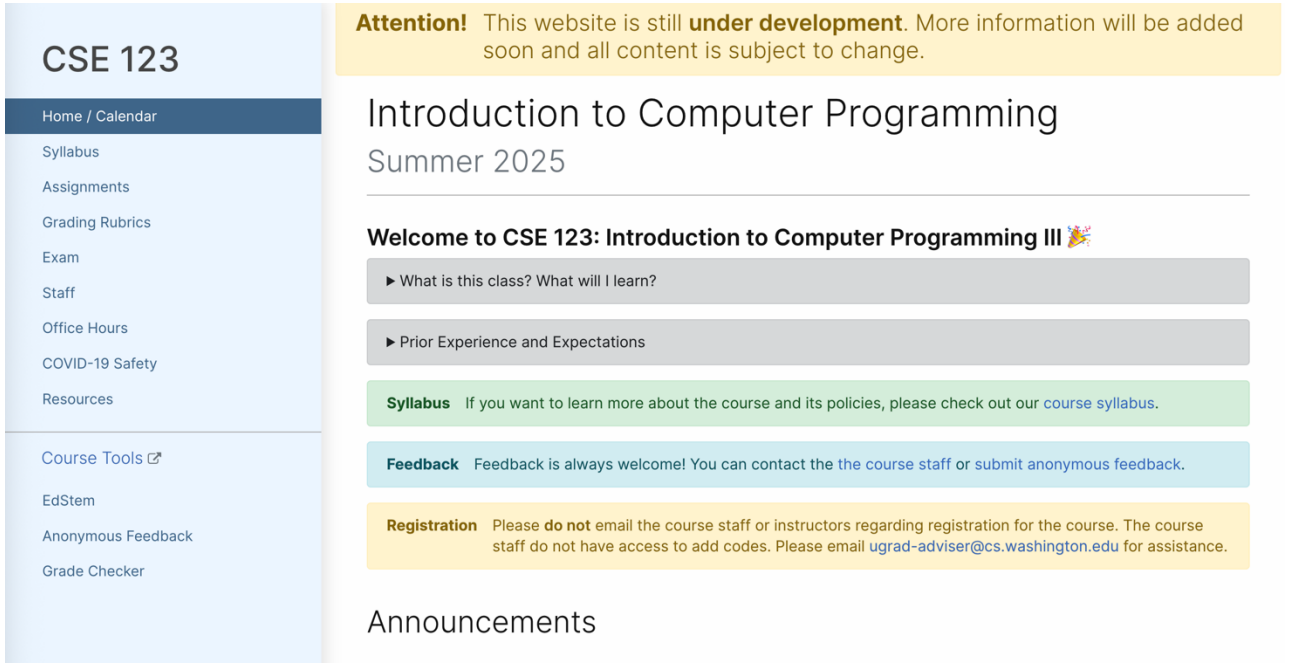
- Introductions
- About this Course
  - Course Components & Tools
  - Making the Most of this Class
- Assignments and Grading
- Comparable

Course Website

cs.uw.edu/123



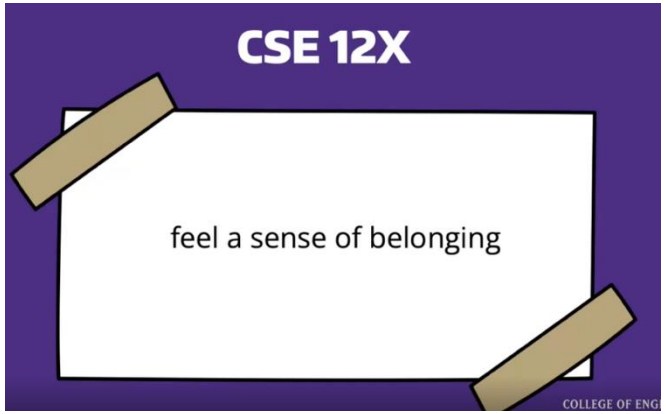
Review the syllabus!



Contains most course info – check frequently!
Announcements, Calendar, Lecture Slides, Office Hours schedule, Staff Bios, Important Links

Creating an inclusive environment

Video



- This is a more professional environment than hanging out with friends
- Think about the impact your words can have.
- Collaboration, Support, and Empathy
- Check your own biases and communicate thoughtfully
- Challenge unacceptable behaviors

Other Course Tools

Ed

- Community & Information
  - Discussion Board
    (please ask & answer!; anonymous option)
  - Announcements
- Pre-Class Materials / Section Handouts
- Assignments
  - Online IDE
  - Submit assignments
  - View Feedback

My Digital Hand

- Queueing in office hours

VSCode

- Develop offline
- Visual debugger

Canvas

- Lecture recordings

Sli.do

- In-class activities
  (ungraded)
- No account needed

Lecture Outline

- Introductions
- **About this Course**
  - Course Components & Tools
  - **Making the Most of this Class**
- Assignments and Grading
- Comparable

Digression: One of my hobbies

How Learning Works

- Learning requires **active participation** in the process. It's not as simple as sitting and listening to someone talk at you.
  - Requires **deliberate practice** in **learning by doing**
  - Benefits from **collaborative learning**

- Hybrid classroom model
  - Asks you to do some preparation before class in the form of readings and practice problems.
    - Should take ~30 minutes outside of class per lesson
  - Class will start with brief recap, then pick up where the reading and practice problems leave off.
  - Attendance isn't graded, but showing up and trying is the first step in succeeding in the class!

- Pre-class materials are ungraded, but…
  - It's okay if you find them challenging! That means you are learning!

Learning Pattern

- Pre-class Work
  - Your first introduction to a topic — Instruction
  - Interactive components — Practice
- In-class lessons
  - Recap of pre-class work — Instruction
  - Further instruction from where it left off — Instruction
  - Interactive components — Practice
- Section
  - Review, restate, rephrase content — Instruction
  - Very interactive — Practice
- Assignments
  - Learn by applying! — Practice
  - We're here to help! — Instruction
- Quizzes/Final Exam
  - Show us what you've learned — Assessment

Getting Help

- Discussion Board
  - Feel free to make a public or private post on Ed
  - We encourage you to answer other peoples' questions! A great way to learn
- Introductory Programming Lab (Office Hours)
  - TAs can help you face to face in office hours, and look at your code
  - You can go to the IPL with **any** course questions, not just assignments
- Section
  - Work through related problems, get to know your TA who is here to support you
- Your Peers
  - We encourage you to form study groups! Discord or Ed are great places to do that
- Email
  - We prefer that all content and logistic questions go on the Ed discussion board (even if you make them private). Many more students than staff!
  - For serious personal circumstances, you can email me directly. It never hurts to email us, but if it's a common logistic question, we may politely ask you to post on the discussion board instead.

Lecture Outline

- Introductions
- About this Course
  - Course Components & Tools
  - Making the Most of this Class
- **Assignments and Grading**
- Comparable

Assignments and Grading

- Our goal in the course is for you to **gain proficiency the concepts and skills** we teach
- We assess your proficiency by asking you to apply the concepts and skills on tasks or problems
- By necessity, we are assessing your *work* as a proxy for your proficiency

Assignments

- Your learning in this course will be assessed in four ways:
  - Programming Assignments (4 total)
    - Structured programming assignments to assess your proficiency of programming concepts
  - Creative Projects (4 total)
    - Smaller, more open-ended assignments to give you space to explore
  - Quizzes (3 total, in section)
    - Series of problems covering all material up to that point
  - Final Exam (Wednesday+Friday, August 20+22, 9:40-10:40)
    - Final, culminating assessment of all your skills and knowledge

Resubmission and Ignored Quiz Problems

Learning takes time, and doesn't always happen on the first try

- One previous Programming Assignment or Creative Project can be **resubmitted** each week
  - Must be accompanied by a write-up describing changes (via Google Form)
  - Grade on resubmission will replace original grade
  - An assignment can be resubmitted in the 3 cycles after feedback has been published
  - *Tip: Resubmit as early as possible!*
- We will ignore your **two lowest quiz problem grades**
  - No special action required– we'll do this automatically

- See the syllabus for more details

Grading

Grades should reflect your proficiency in the course objectives

- All assignments will be graded **E (Excellent), S (Satisfactory), N (Not yet), or U (Unassessable)**
- Final grades will be assigned based on the **amount of work at each level**

- See the syllabus for more details

Collaboration Policy

- When we assess your work in this class, we need to know that it's *yours*.
- Unless otherwise specified, **all graded work must be completed individually.**

Some specific rules to highlight:

- do not share your own solution code or view solution code from any source – including but not limited to other students, tutors, or the internet
- do not use AI tools (e.g. ChatGPT) on graded work in any capacity

See the syllabus for more details (this is *very* important to understand).

Lecture Outline

- Introductions
- About this Course
  - Course Components & Tools
  - Making the Most of this Class
- Assignments and Grading
- **Comparable**

Comparable

- `Comparable<E>` is an `interface` that allows implementers to define an ordering between two objects
  - Used by `TreeSet`, `TreeMap`, `Collections.sort`, etc.

- One required method:

```
public int compareTo (E other);
```

- Returned integer falls into 1 of 3 categories
  - `< 0`: `this` is "less than" `other`
  - `= 0`: `this` is "equal to" `other`
  - `> 0`: `this` is "greater than" `other`

```
a.compareTo(b);
```

this other

- "Sorted" usually means from least to greatest
  - Think sorted alphabetically, numerically, chronologically

Comparable

- `compareTo` implementation when comparing two integers (`a`) ascending:

```
if (this.a < other.a)      -> negative number
else if (this.a > other.a) -> positive number
else                       -> 0
```

- This is just subtraction!

```
this.a - other.a
```

- What if we wanted to sort descending?

```
other.a - this.a
```

- **Warning**: this only works for integers! Doubles have issues with truncation.

Comparable

Consider a `Course` class that has the following fields: an `int` enrollment and a `String` courseCode.

We want to sort so that **higher** enrollment classes come **first**, and when enrollment is tied, we sort by the **course code alphabetically**

```
public int compareTo(Course other) {
    if (this.enrollment != other.enrollment) {
        return other.enrollment - this.enrollment; // More students first
    }
    return this.courseCode.compareTo(other.courseCode);
}
```

Comparable

We want to sort so that **higher** enrollment classes come **first**, and when enrollment is tied, we sort by the **course code alphabetically**

```
public int compareTo(Course other) {
    if (this.enrollment != other.enrollment) {
        return other.enrollment - this.enrollment; // More students first
    }
    return this.courseCode.compareTo(other.courseCode);
}
```

We are sorting first based on enrollment, then by courseCode, so we first have to check inequally of enrollment

Comparable

We want to sort so that **higher** enrollment classes come **first**, and when enrollment is tied, we sort by the **course code alphabetically**

```
public int compareTo(Course other) {
    if (this.enrollment != other.enrollment) {
        return other.enrollment - this.enrollment; // More students first
    }
    return this.courseCode.compareTo(other.courseCode);
}
```

Subtraction trick since `enrollment` is an int!

Notice that we are doing `other.enrollment - this.enrollment`

Comparable

We want to sort so that **higher** enrollment classes come **first**, and when enrollment is tied, we sort by the **course code alphabetically**

```
public int compareTo(Course other) {
    if (this.enrollment != other.enrollment) {
        return other.enrollment - this.enrollment;
    }
    return this.courseCode.compareTo(other.courseCode);
}
```

this.enrollment	other.enrollment	this-other	other-this	**expected**
550	300	250	-250	negative (sorted earlier)
120	500	-380	380	positive (sorted later)
65	65	0	0	equal

Comparable

We want to sort so that **higher** enrollment classes come **first**, and when enrollment is tied, we sort by the **course code alphabetically**

```
public int compareTo(Course other) {
    if (this.enrollment != other.enrollment) {
        return other.enrollment - this.enrollment;
    }
    return this.courseCode.compareTo(other.courseCode);
}
```

this.enrollment	other.enrollment	this-other	other-this	expected
550	300	250	-250	negative (sorted earlier)
120	500	-380	380	positive (sorted later)
65	65	0	0	equal

Comparable

We want to sort so that **higher** enrollment classes come **first**, and when enrollment is tied, we sort by the **course code alphabetically**

```
public int compareTo(Course other) {
    if (this.enrollment != other.enrollment) {
        return other.enrollment - this.enrollment; // More students first
    }
    return this.courseCode.compareTo(other.courseCode);
}
```

Since we are wanting to compare `Strings`, we can utilize `String`'s built in `compareTo()`! (compares alphabetically)

Coming up…

- 👦🏫 Go to your first quiz section and meet your TAs tomorrow!
  - Make sure to double check MyUW for the location.
- 💻 Complete Pre-Class Work 1 before class on Friday!
  - It will be posted and linked from the course calendar later today.
- ❓ Complete the Introductory Survey
  - This helps us gather data about the students taking our classes and their backgrounds, to inform future offerings.