Welcome to Computer Science 211

Road map

What’s it all about?

Road map

What’s it all about?

Course Topics

Road map

What’s it all about?

Course Topics

Policies & grades

Road map

What’s it all about?

Course Topics

Policies & grades

Academic honesty

Road map

What’s it all about?

Course Topics

Policies & grades

Academic honesty

Help & advice

Up next

Welcome to Computer Science 211

What’s it all about?

Course Topics

Policies & grades

Academic honesty

Help & advice

What CS 211 is all about (1/2)

From the course abstract:

What CS 211 is all about (1/2)

From the course abstract:

- CS 211 teaches foundational software design skills at a small-to-medium scale.

What CS 211 is all about (1/2)

From the course abstract:

- CS 211 teaches foundational software design skills at a small-to-medium scale. We will grow from writing single functions to writing interacting systems of several components.

What CS 211 is all about (1/2)

From the course abstract:

- CS 211 teaches foundational software design skills at a small-to-medium scale. We will grow from writing single functions to writing interacting systems of several components.
- We aim to provide a bridge from the student-oriented *HtDP* languages

What CS 211 is all about (1/2)

From the course abstract:

- CS 211 teaches foundational software design skills at a small-to-medium scale. We will grow from writing single functions to writing interacting systems of several components.
- We aim to provide a bridge from the student-oriented *HtDP* languages (that is, CS 111)

What CS 211 is all about (1/2)

From the course abstract:

- CS 211 teaches foundational software design skills at a small-to-medium scale. We will grow from writing single functions to writing interacting systems of several components.
- We aim to provide a bridge from the student-oriented *HtDP* languages (that is, CS 111) to real, industry-standard languages and tools.

What CS 211 is all about (1/2)

From the course abstract:

- CS 211 teaches foundational software design skills at a small-to-medium scale. We will grow from writing single functions to writing interacting systems of several components.
- We aim to provide a bridge from the student-oriented *HtDP* languages (that is, CS 111) to real, industry-standard languages and tools. Like C11, the UNIX shell, Make, C++14, and CLion.

What CS 211 is all about (1/2)

From the course abstract:

- CS 211 teaches foundational software design skills at a small-to-medium scale. We will grow from writing single functions to writing interacting systems of several components.
- We aim to provide a bridge from the student-oriented *HtDP* languages (that is, CS 111) to real, industry-standard languages and tools. Like C11, the UNIX shell, Make, C++14, and CLion.
- In the first half…

What CS 211 is all about (2/2)

From the course abstract:

- In the first half, you’ll learn the basics of imperative programming and manual memory management using the C programming language.

What CS 211 is all about (2/2)

From the course abstract:

- In the first half, you'll learn the basics of imperative programming and manual memory management using the C programming language. This will help you form connections between the high-level programming concepts you learned in CS 111 and the low-level machine concepts you will learn in CS 213.
- In the second half, we'll transition to C++, which provides abstraction mechanisms such as classes and templates that we use to express our design ideas.

What CS 211 is all about (2/2)

From the course abstract:

- In the first half, you'll learn the basics of imperative programming and manual memory management using the C programming language. This will help you form connections between the high-level programming concepts you learned in CS 111 and the low-level machine concepts you will learn in CS 213.
- In the second half, we'll transition to C++, which provides abstraction mechanisms such as classes and templates that we use to express our design ideas. We'll learn how to define our own, new types that act like the built-in ones.

What CS 211 is all about (2/2)

From the course abstract:

- In the first half, you'll learn the basics of imperative programming and manual memory management using the C programming language. This will help you form connections between the high-level programming concepts you learned in CS 111 and the low-level machine concepts you will learn in CS 213.
- In the second half, we'll transition to C++, which provides abstraction mechanisms such as classes and templates that we use to express our design ideas. We'll learn how to define our own, new types that act like the built-in ones.
- Topics include...

Up next

Topics

- Language mechanisms
- Design techniques
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming
- Design techniques
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
- Design techniques
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
- Design techniques
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
- Design techniques
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers

- Design techniques

- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization
- Engineering practices

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization
- Engineering practices
  - Testing

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization
- Engineering practices
  - Testing: for gaining confidence in our software

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization
- Engineering practices
  - Testing: for gaining confidence in our software
  - Debugging

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization
- Engineering practices
  - Testing: for gaining confidence in our software
  - Debugging: to see what's happening in memory

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization
- Engineering practices
  - Testing: for gaining confidence in our software
  - Debugging: to see what's happening in memory
  - The Unix shell

Topics

- Language mechanisms
  - New syntax for functional programming: expressions, values, conditionals, variables, functions
  - Imperative programming
    - Statements: sequencing, iteration
    - Mutation: objects, assignment
  - Memory allocation on the stack and the heap
  - Representing information with structs, arrays, pointers
  - Static types, type erasure, generics
- Design techniques
  - Data abstraction: defining our own types
  - Memory management via ownership and borrowing
  - RAII: Resource Acquisition Is Initialization
- Engineering practices
  - Testing: for gaining confidence in our software
  - Debugging: to see what's happening in memory
  - The Unix shell: a compositional user interface

Up next

Welcome to Computer Science 211

What’s it all about?

Course Topics

Policies & grades

Academic honesty

Help & advice

Grade composition

what	**%**	**when**	**#**
programming homeworks	65.0%	Tuesdays	6
lab quizzes	5.0%	Sundays*	8
final project prose	7.5%	Thursdays	1
final project code	22.5%	Tu, Mar 9	1

* Except Lab1, which is due this Thursday (Jan 14)!

Homework policies

- Two will be done on your own (HW1 & HW5)

Homework policies

- Two will be done on your own (HW1 & HW5)
- Most will be pair-programmed with a registered partner

Homework policies

- Two will be done on your own (HW1 & HW5)
- Most will be pair-programmed with a registered partner
- Don’t get behind!

Homework policies

- Two will be done on your own (HW1 & HW5)
- Most will be pair-programmed with a registered partner
- Don't get behind!
- You'll need to do a self evaluation for each

Homework policies

- Two will be done on your own (HW1 & HW5)
- Most will be pair-programmed with a registered partner
- Don’t get behind!
- You’ll need to do a self evaluation for each
- No cheating…

Up next

Welcome to Computer Science 211

- What’s it all about?
- Course Topics
- Policies & grades
- Academic honesty
- Help & advice

Levels of collaboration

We define three levels of collaboration*:

1. **Partner Collaboration**
    - Your code and the other student's code are identical because you share it and work on it together
    - Only for registered partners on specified homeworks

* Refer to the syllabus for the official version of this policy.

Levels of collaboration

We define three levels of collaboration*:

1. **Partner Collaboration**
   - Your code and the other student's code are identical because you share it and work on it together
   - Only for registered partners on specified homeworks
2. **Close Collaboration**
   - You communicate about code however you see fit
   - Only acceptable when working on labs

* Refer to the syllabus for the official version of this policy.

Levels of collaboration

We define three levels of collaboration*:

1. **Partner Collaboration**
   - Your code and the other student's code are identical because you share it and work on it together
   - Only for registered partners on specified homeworks
2. **Close Collaboration**
   - You communicate about code however you see fit
   - Only acceptable when working on labs
3. **Arms-length Collaboration**
   - You discuss problems and solutions at a high level
   - MAY NOT read, write, look at, record, or transcribe code in question
   - MAY NOT have the code up on screen during collaboration
   - MUST submit a file named `COLLABORATION.txt` that lists your arms-length collaborators

* Refer to the syllabus for the official version of this policy.

Academic honesty

In CS 211, we take cheating very seriously.

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
    - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
    - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
    - Obtain code from an outside resource, such as Stack Overflow

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow
- **Please don't do these things,** because:

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow
- **Please don’t do these things,** because:
  - If you don’t write code, you won’t learn; try to embrace the struggle!

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow
- **Please don't do these things,** because:
  - If you don't write code, you won't learn; try to embrace the struggle!
  - All suspected cheating will be reported to the relevant dean for investigation

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow
- **Please don’t do these things,** because:
  - If you don’t write code, you won’t learn; try to embrace the struggle!
  - All suspected cheating will be reported to the relevant dean for investigation
- If unsure about your particular situation, ask the instructor or other course staff

Academic honesty

In CS 211, we take cheating very seriously.

- Cheating is when you:
  - Share (give or receive) homework code with anyone who is neither course staff nor your official, registered partner
  - Obtain code from an outside resource, such as Stack Overflow
- **Please don't do these things,** because:
  - If you don't write code, you won't learn; try to embrace the struggle!
  - All suspected cheating will be reported to the relevant dean for investigation
- If unsure about your particular situation, ask the instructor or other course staff
- *For the official policy, please see the Collaboration and Academic Integrity section of the syllabus*

Up next

Getting help

- **Synch.** Your course staff has online office hours:

Instructors:	Jesse Tov	(M/W, 10–11:30 AM CT)
	Branden Ghena	(Tu/Th, 3:30-5 PM CT)

Getting help

- **Synch.** Your course staff has online office hours:

Instructors:	Jesse Tov	(M/W, 10–11:30 AM CT)
	Branden Ghena	(Tu/Th, 3:30-5 PM CT)
Grad TA:	Yihan Zhang	

Getting help

- **Synch.** Your course staff has online office hours:

Instructors:	Jesse Tov	(M/W, 10–11:30 AM CT)
	Branden Ghena	(Tu/Th, 3:30-5 PM CT)
Grad TA:	Yihan Zhang	
Peer TAs:	Ben Caterine, Nicole Chen, Naythen Farr, Ben Fisk, Jules Gilligan, David Jin, Mae Mastin, Alex Saavedra	

The schedule and Zoom links are on Canvas.

Getting help

- **Synch.** Your course staff has online office hours:

Instructors:	Jesse Tov	(M/W, 10–11:30 AM CT)
	Branden Ghena	(Tu/Th, 3:30-5 PM CT)
Grad TA:	Yihan Zhang	
Peer TAs:	Ben Caterine, Nicole Chen, Naythen Farr, Ben Fisk, Jules Gilligan, David Jin, Mae Mastin, Alex Saavedra	

  The schedule and Zoom links are on Canvas.

- **Asynch.** Ask questions on Campuswire:

  `https://campuswire.com/c/G0D1CAAD8`

Advice

- If you’re considering dropping, please talk to me first.

Advice

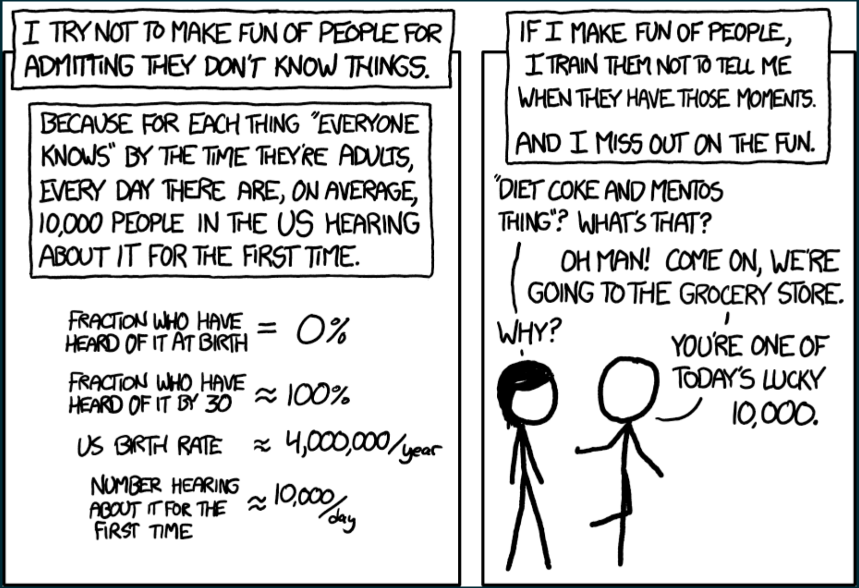
- If you're considering dropping, please talk to me first.
- The only prereq is CS 111, so if you succeeded there then you absolutely do belong here.

Advice

- If you’re considering dropping, please talk to me first.
- The only prereq is CS 111, so if you succeeded there then you absolutely do belong here.
- If you find the course difficult, that’s because it *is* difficult.

Advice

- If you're considering dropping, please talk to me first.
- The only prereq is CS 111, so if you succeeded there then you absolutely do belong here.
- If you find the course difficult, that's because it *is* difficult.
- Be kind to each other.


I TRY NOT TO MAKE FUN OF PEOPLE FOR ADMITTING THEY DON'T KNOW THINGS.
BECAUSE FOR EACH THING "EVERYONE KNOWS" BY THE TIME THEY'RE ADULTS, EVERY DAY THERE ARE, ON AVERAGE, 10,000 PEOPLE IN THE US HEARING ABOUT IT FOR THE FIRST TIME.
FRACTION WHO HAVE HEARD OF IT AT BIRTH = 0%
FRACTION WHO HAVE HEARD OF IT BY 30 ≈ 100%
US BIRTH RATE ≈ 4,000,000/year
NUMBER HEARING ABOUT IT FOR THE FIRST TIME ≈ 10,000/day
IF I MAKE FUN OF PEOPLE, I TRAIN THEM NOT TO TELL ME WHEN THEY HAVE THOSE MOMENTS.
AND I MISS OUT ON THE FUN.
"DIET COKE AND MENTOS THING"? WHAT'S THAT?
OH MAN! COME ON, WE'RE GOING TO THE GROCERY STORE.
WHY?
YOU'RE ONE OF TODAY'S LUCKY 10,000.

Relative homework difficulties

HW	Difficulty
1	3

(On a scale from 1 to 10)

Relative homework difficulties

HW	Difficulty
1	3
2	5

(On a scale from 1 to 10)

Relative homework difficulties

HW	Difficulty
1	3
2	5
3	7

(On a scale from 1 to 10)

Relative homework difficulties

HW	Difficulty
1	3
2	5
3	7
4	11

(On a scale from 1 to 10)

Relative homework difficulties

HW	Difficulty
1	3
2	5
3	7
4	11
5	6

(On a scale from 1 to 10)

Relative homework difficulties

HW	Difficulty
1	3
2	5
3	7
4	11
5	6
6	8

(On a scale from 1 to 10)

Relative homework difficulties

HW	Difficulty
1	3
2	5
3	7
4	11
5	6
6	8
FP	10ish*

(On a scale from 1 to 10)

* But really it's up to you