

CSCI 145 · Introduction to Computing · Fall 2026

Course Introduction

Lecture 1 — course overview, algorithms, variables & expressions

Outline

- What is computer science?
-

- Syllabus overview
-

- Algorithms
-

- Variables & expressions
-

programming theory

building computers

networking machine learning

artificial intelligence

games

algorithms

What is computer science?

making art

operating systems

human-computer interaction

databases

robotics

bioinformatics

What is computer science?

Creating solutions to problems

- identifying problems
- figuring out if an efficient process to solve them exists
- designing processes that solve them
- evaluating the solutions for correctness and complexity

What do computer scientists do?

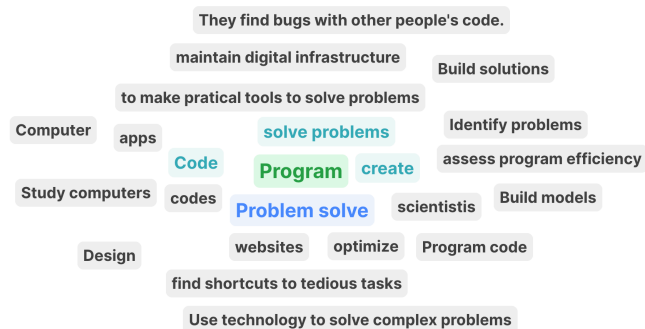
Go to **slido.com** and enter the code:

3674841

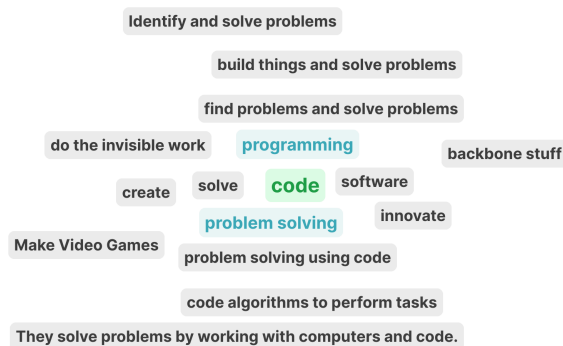
Or [join directly here](#)



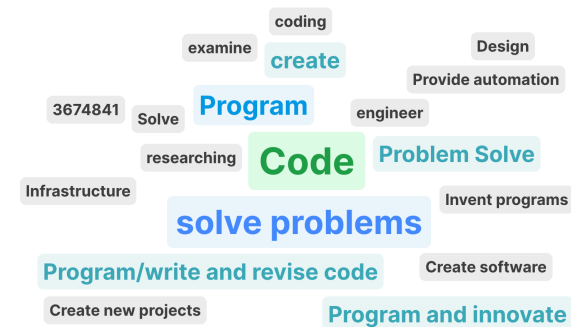
What do computer scientists do?



Section A



Section B



Section C

What do computer scientists do?

A computer scientist understands how to design and analyze **algorithms**, how to store and retrieve information, how computers function, and how to develop software systems that solve complex problems.

— *adapted from cse.umich.edu*

Part 2 of 4 — the course

Syllabus overview

About the course

Logistics

- 75-minute class meetings
- 50-minute lab or exam on Fridays
- Homework and labs are programming
- Exams and quizzes are paper-and-pencil

Purpose

- Introductory computing course
- An entry point to the major/minor
- Expects no prior programming experience

You will learn the fundamentals of computer science, gain experience with Python programming, create new things, and build your problem-solving skills.

Learning objectives

- Describe computer science as a discipline and give examples of how programs, algorithms, and data structures are used in different application areas
- Solve computational problems with procedural statements (assignment, operators, conditionals, loops), functions, basic data structures (list, set, map), objects, and I/O
- Design, implement, document, and debug medium-sized programs in Python
- Analyze and apply iterative and recursive solutions to computational problems
- Informally explain and evaluate asymptotic complexity of algorithms
- Explain how data is represented in the computer, especially numbers, text, and images

Assessment

Homework - 15%

Programming in Python
Feedback on correctness
and style

Labs - 10%

7 Fridays throughout the
semester
Work in pairs, submit by
Sunday night

Quizzes - 15%

Most Mon/Tues during
class, 10-15 minutes each
In person, on paper

Exams - 30%

4 Fridays throughout the
semester, 50 minutes
each
In person, on paper

Final exam - 25%

During finals week
In person, on paper

Participation - 5%

Attendance and
engagement
Surveys and reflections

Course site: go.middlebury.edu/cs145 (middcs.github.io/csci0145-f26)

Course materials

- All course materials are at go/cs145/
- Free online textbooks — nothing to purchase
- Python, using the **Thonny** IDE (Integrated Development Environment)
- At the end of class or after class today: install Thonny from thonny.org

Course site: go.middlebury.edu/cs145 (middcs.github.io/csci0145-f26)

Getting help

- Office hours — see [go/briggs/](#), [go/candrews/](#), and [go/smith/](#)
- Drop-in hours — see [go/cshelp/](#)
- Campuswire — ask questions about assignments, see what others are asking

Course site: go.middlebury.edu/cs145 (middcs.github.io/csci0145-f26)

Disability Access and Accommodation

- If you have a letter of accommodation, please send it to me as soon as possible
- If you think you may need one, contact the Disability Resource Center
- For more information, email ada@middlebury.edu

Collaboration and Honor Code

- **Lab assignments:** collaborate as partners with a classmate
- **Homework:** discuss with classmates, but your code must be your own work
- **Exams and quizzes:** no discussion with classmates, no outside resources
- **All assignments:** outside resources (Google, AI tools) may aid your learning, but may not answer the assignment for you

Part 3 of 4

Algorithms

What is an algorithm?

An unambiguous sequence of steps that, if followed exactly, will solve the problem under consideration.

- Has a **finite** number of steps
- Each step is **well-defined** (unambiguous)
- Eventually **halts**
- Solves a **general class** of problems

Where have you seen algorithms before?

Banana Bread

Ingredients

3 or 4 ripe bananas
1/3 c melted butter
3/4 c sugar
1 egg, beaten
1 t vanilla
1 t baking soda
pinch salt
1 1/2 c flour

Instructions

Preheat oven to 350.
Use wooden spoon and mix butter into mashed bananas in large mixing bowl. Mix in sugar, egg and vanilla. Sprinkle baking soda and salt over the mixture and mix in. Add flour last and mix. Pour mixture into buttered 4x8 in loaf pan. Bake 1 hour. Cool on rack.

Recipes



Origami



Rubik's Cube

Expressing an algorithm as pseudocode

```
if tired           # you'll want a coffee if you're tired  
    fill cup with coffee  
else  
    fill cup with water  
  
while thirsty      # keep drinking as long as you are thirsty  
    drink from cup
```

Pseudocode mixes natural language and familiar control structures
(**if** / **else** , **while**) without committing to a real language's exact syntax.

One thing to expect

Algorithms don't always work the first time — a **bug** is when your code doesn't do what you intended. Finding and fixing bugs (**debugging**) is a normal, central part of programming, and we'll build real tools and habits for it later in the course.

For example, what about bugs in the pseudocode of the previous slide!?

What happens when the “cup” becomes empty?

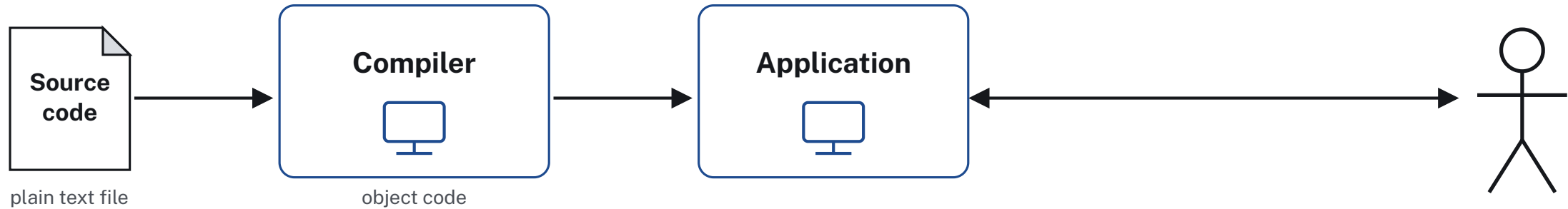


Logical error

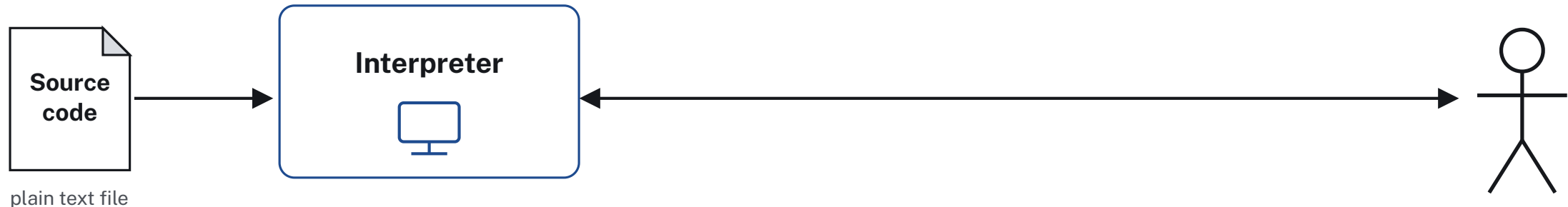
It will no longer be possible to drink from the cup if it is empty.

How does your code run?

Compiled code C, C++, Java, etc.



Interpreted code Python, JavaScript, Ruby, PHP, etc.



Python is **interpreted** — there's no separate compile step. The interpreter reads and runs your code directly, which is why typing a line into the shell shows a result immediately.

Part 4 of 4

Variables & expressions

What is a variable?

Variables store data to use later.

```
numerator = 12  
denominator = 3  
print(numerator // denominator)
```

← // is floor division (more on this soon)

numerator:

12

denominator:

3

Running this prints **4**.

How do we create a variable?

numerator = 12

variable name
on left

value or expression
on right

numerator: 12

Expressions and operators

```
>>> 3 + 4  
7
```

```
>>> 3 + 4 * 2  
11
```

```
>>> (3 + 4) * 2  
14
```

```
>>> 2 ** 5  
32
```

An **expression** combines values with **operators** to produce a new value. Python follows the usual order of operations — parentheses, exponents, multiplication and division, then addition and subtraction.

Three ways to divide

```
>>> 7 / 2  
3.5
```

```
>>> 7 // 2  
3
```

```
>>> 7 % 2  
1
```

/ always gives a decimal answer

// (floor division) rounds down to the nearest whole number

% (modulo) gives just the remainder

Note:

-7 // 2 is -4

7.0 // 2 is 3.0

Variables can appear in expressions

```
radius = 4  
area = 3.14159 * radius ** 2  
print(area)
```

Python runs this top to bottom: `radius` is created first, then used to compute `area`. Change `radius` and rerun — nothing else in the program has to change.

Which type should you use to represent...

Your name

`str` — a sequence of characters

Days in a week

`int` — a whole number

Are you a dog person?

`bool` — True or False

Distance to 75 Shannon, in km

`float` — a decimal number

Figuring out types

`type()` tells you a value's type.

```
>>> type("hello")  
<class 'str'>
```

```
>>> type(2.0)  
<class 'float'>
```

```
>>> type("2")  
<class 'str'>
```

Converting between types

```
>>> int(2.5)
2
>>> float(2)
2.0
>>> str(2)
'2'
>>> int("2")
2
```

`int()`, `float()`, and `str()` convert between types. We'll give `type` and its relatives a formal name — **functions** — next class.

To do

- Review the course site, syllabus, these slides, and today's notes: go/cs145/
- Install the Thonny IDE from thonny.org
- Accept the invitation to join [Gradescope](#)
- Accept the invitation to join [Campuswire](#)
- Begin perusing [How to Think Like a Computer Scientist](#)

Next time

Lecture 2 — more expressions and variables, and our first functions