CS 105: Introduction to Computer Science

Prof. Thao Nguyen

Spring 2025

Materials adapted from Dave Wonnacott

Welcome!

- Please fill out a **notecard**

- Let me know if you can’t access **Piazza**

Why CS?

- The world runs on computers!

- Programming has many applications

- Develop and practice computational thinking

Course Staff

- **Professor:** Thao Nguyen (she/her)
  - From Hanoi, Vietnam
  - Ph.D. from Brown University
  - Research: Human-robot interaction and collaboration
  - Office: KINSC L303
  - Email: tnguyen3@haverford.edu

Course Staff

- **Lab Instructor:** Suzanne Lindell
  - Handles all lab assignments
  - Email: shlindel@haverford.edu

Course Staff

- **Teaching Assistants:**

Ahmed Haj Ahmed

Meghan Galban

Amy Saxon

Mateo Taylor

Expectations

- Little/no prior programming experience
- Attend all lectures and labs *on time*, and actively participate
  - Email me if you will be absent
- Work on lab projects each week
  - Start your work well before the deadline
  - It's good to arrive at lab already stuck on something, so we can help
- Take two in-class midterms
- Develop and present a final project
- Help your peers, but stay within the Honor Code and CS collaboration policy
- Treat others with respect and awareness
- **Let us know if you are struggling/scared/discouraged**

Resources

- Course webpage
- Office & TA hours
- Lab monitors in H110 Sunday – Thursday from 7-11pm
- Piazza: should be used for all content/logistic questions
- Email: allow at least 24 hours for a response (48 during weekends)

Collaboration Policy

- Allowed sources:
  - Textbook, lecture slides
  - Course staff
  - Your lab partner(s)
  - The Internet, though *it must be cited and only used for reference*

- Rule of thumb: **you may not copy code**
  - This includes copy-paste, manual copying, and AI coding
  - When in doubt, ask the instructors

Academic Accommodations

Haverford College is committed to providing equal access to students with a disability. If you have (or think you have) a learning difference or disability – including mental health, medical, or physical impairment - please contact the Office of Access and Disability Services (ADS) at **hc-ads@haverford.edu**. The Coordinator will confidentially discuss the process to establish reasonable accommodations.

Students who have already been approved to receive academic accommodations and want to use their accommodations in this course should share their verification letter with me and also make arrangements to meet with me as soon as possible to discuss their specific accommodations. Please note that accommodations are **not retroactive** and require advance notice to implement.

It is a state law in Pennsylvania that individuals must be given advance notice if they are to be recorded. Therefore, any student who has a disability-related need to audio record this class must first be approved for this accommodation from the Coordinator of Access and Disability Services and then must speak with me. Other class members will need to be aware that this class may be recorded.

https://www.haverford.edu/access-and-disability-services/accommodations/receiving-accommodations


Let's
Get
Started!

So, let's start!

Programming is about *problem solving* (specifically, problems involving *information*)

We distinguish

- Problem *instances* (e.g., what is .. *6*7*? *1337*1701*? *area of 12'x17' room*?)
- *General* problems (e.g., multiply numbers; find areas of rectangles)

We express solutions to general problems as *computer programs*

- Many specific notations, e.g. Python, Java, C++, C#, Haskell

Warning: Programs often contain "bugs"

So, let's start *programming*!

Solving a problem instance (size of a 12' by 17' room, in square feet):

```
12 * 17
```

Or, to be more helpful:

```
print("A 12' by 17' room is", 12*17, "square feet")
```

We can use variables to name things

```
area: float = 12*17
print("A 12' by 17' room is", area, "square feet")
```

Let's *really* start programming!

We express the solution to a *general problem* as a Python "function":

```
def rectangle_area(width: float, height: float) -> float:
    return width*height      # area is just width times height

rectangle_area(12, 17)
```

Or, to be more helpful:

```
print("area of 12' by 17' room is", rectangle_area(12, 17), "square feet")
```

Anatomy of a Python Function

```
def rectangle_area(width: float, height:float) -> float:
    return width*height      # area is just width times height

rectangle_area(12, 17)
```

- *parameters* (w and h): identify necessary information for the problem
- *type* (float): identifies kind of information (number, text, list, etc.)
  - Type names after w and h give type of information coming in
  - Type name after -> gives the type of information coming out
- *body* is made up of Python statements (such as return) to produce the result
- *return* identifies the information coming out of the function
- *comments* (with #) communicate to *other programmers,* not the computer
- *call* to function gives *actual arguments* (12 and 17), can be used in later calc.
  - rectangle_area(12, 17) + rectangle_area(12,5) # combined area of two rooms

Now, you start programming!

For reference, our previous Python function:

```
def rectangle_area(width: float, height: float) -> float:
    return width*height      # area is just width times height

rectangle_area(12, 17)       # function call
```

Group activity: Write either

- "right_triangle_area"
- "useful_room_area" (door-swing removes seven square feet of useful space)

Next example: useful_room_area

Let's explore the answer to that second exercise:

```
def useful_room_area(width: float, height: float) -> float:
    return ?????

useful_room_area(12, 17)
```

But, what if someone asks for?

```
useful_room_area(4, 1.2)
```

⇒ ***If*** time permits, Thao introduces "if" statements (otherwise, Thursday!)

Basic Uses of Conditional Statements ("if")

```
def useful_room_area(width: float, height:float) -> float:
    if width*height >= 7:
        return width*height - 7    # width times depth, minus door-swing
    else:
        return 0

useful_room_area(4, 1.2)
```

Anatomy of if/else statements:

- *test condition*: a boolean expression (True or False) telling which part to select
- *true-branch*: code indented below test is used when test condition is *true*
- *false-branch:* code indented below "else:" is used when test condition is *false*

Programming and CS: Back to the Big Picture

Computer Science involves reasoning about programs and computation

- How do we know if a program is *correct*?
- How do we understand the *cost* of running a program?
- How do we try to predict/understand the *social impact*?

Use techniques from many fields:

- Mathematics
- Engineering
- Social engagement (with customers; among programmers)