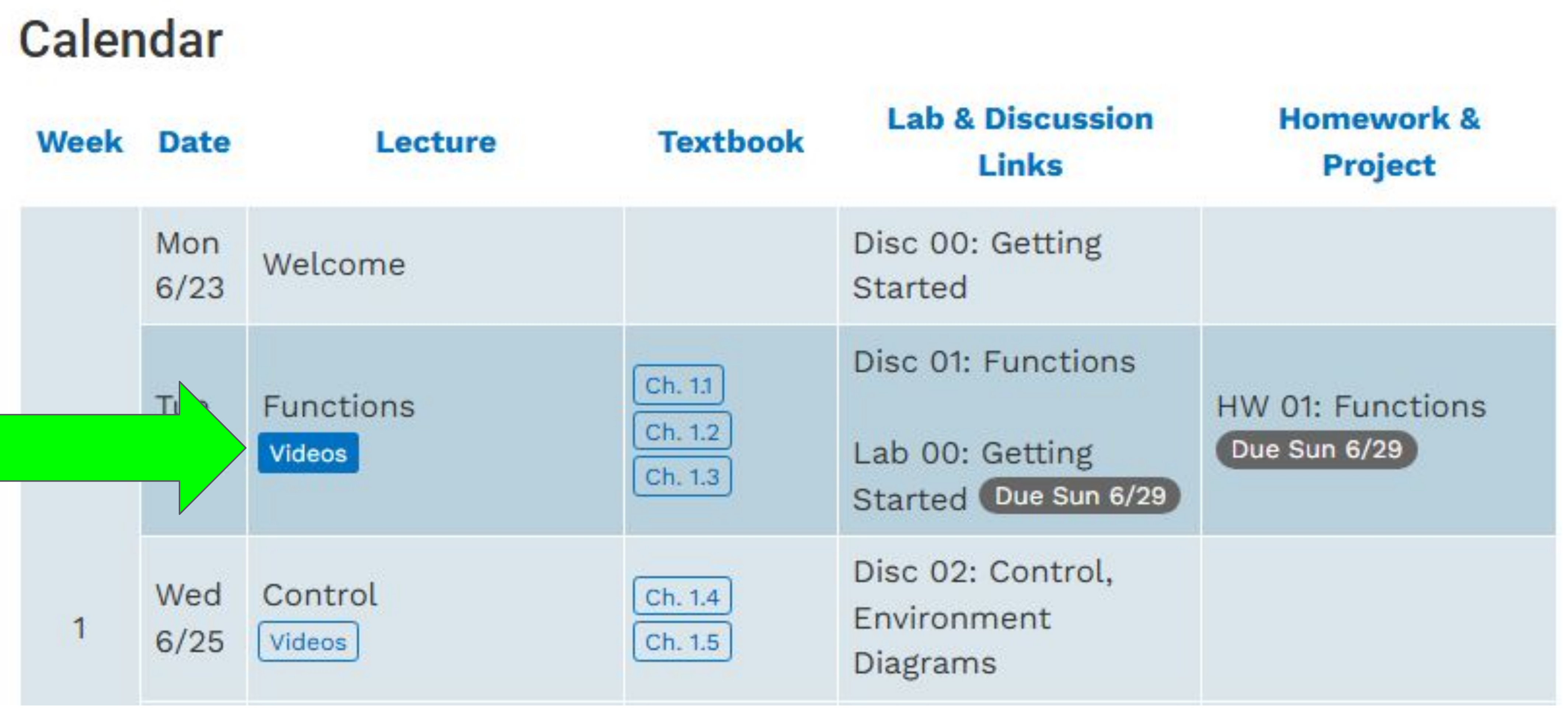
# Welcome to Data C88C!

**Lecture 03: Control**
Wednesday, June 25th, 2025
Week 1
Summer 2025
Instructor: Eric Kim (ekim555@berkeley.edu)

# Announcements

- Lab00
  - <u>Due</u>: Sun June 29th, 11:59 PM PST
- HW01
  - <u>Due</u>: Sun June 29th, 11:59 PM PST
- **Reminder**: watch YouTube video BEFORE lecture!
  - See course website for video link



Calendar

| Week | Date | Lecture | Textbook | Lab & Discussion Links | Homework & Project |
|---|---|---|---|---|---|
| | Mon 6/23 | Welcome | | Disc 00: Getting Started | |
| | Tue | Functions<br>Videos | Ch. 1.1<br>Ch. 1.2<br>Ch. 1.3 | Disc 01: Functions<br>Lab 00: Getting Started Due Sun 6/29 | HW 01: Functions<br>Due Sun 6/29 |
| 1 | Wed 6/25 | Control<br>Videos | Ch. 1.4<br>Ch. 1.5 | Disc 02: Control, Environment Diagrams | |

**Important**: watch these videos before lecture to maximize learning!

# Lecture Overview

- Control ("if" statements)
- While loops

# Print and None

(Demo)

# Example: Print Then Return

**Question**: which of these functions first prints, then returns, the value of f(x)?

```
def h1(x):
    return print(f(x))
```

(A)

```
def h2(x):
    print(f(x))
    return f(x)
```

(B)

```
def h3(x):
    y = f(x)
    print(y)
    return y
```

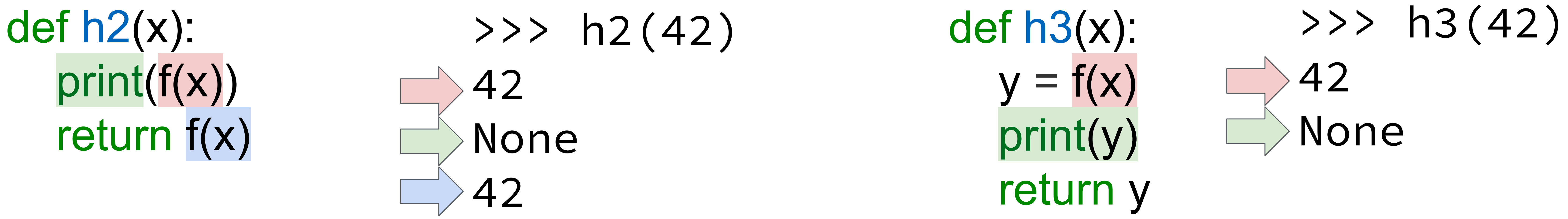
(C)

**Answer**: C

**Question**: what is a function \`f\` where (B) and (C) would have different behavior?

**Answer**: \`f = print\`.

Breaking down the output, color coded to match which part of the code generated it



```
def h2(x):
    print(f(x))
    return f(x)
```

```
>>> h2(42)
42
None
42
```

```
def h3(x):
    y = f(x)
    print(y)
    return y
```

```
>>> h3(42)
42
None
```

# Control

# Conditional Statements

Conditional statements (often called "If" Statements) contain statements that may or may not be evaluated.

| | | x=10 | x=1 | x=-1 |
|---|---|---|---|---|
| if x > 2:<br>    print('big')<br>if x > 0:<br>    print('positive') | Two separate (unrelated) conditional statements | big<br>positive | positive | |
| if x > 2:<br>    print('big')<br>elif x > 0:<br>    print(‘less big’) | One statement with two clauses: if and elif<br>Only one body can ever be executed | big | less big | |
| if x > 2:<br>    print('big')<br>elif x > 0:<br>    print(‘less big')<br>else:<br>    print(‘not pos') | One statement with three clauses: if, elif, else<br>Only one body can ever be executed | big | less big | not pos |

# While loops

- While loops let you repeat some code multiple times

```
def while_ex00():
    print(3)
    print(2)
    print(1)
    print("blast off!")

>>> while_ex00()
3
2
1
blast off!
```

```
def while_ex01():
    i = 3
    while i > 0:
        print(i)
        i = i - 1  # shorthand: i -= 1
    print("blast off!")

>>> while_ex00()
3
2
1
blast off!
```

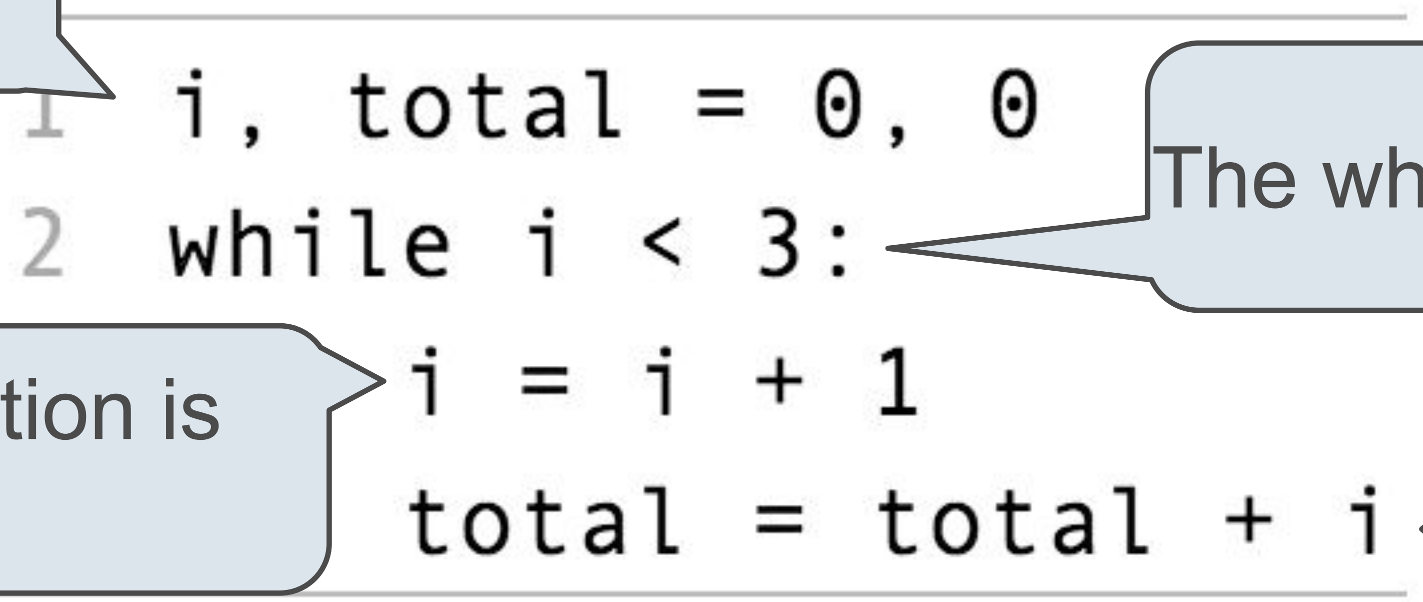
# While Statements

While statements contain statements that are repeated **as long as some condition is true**.

**Important considerations**:

- How many separate names are needed and what do they mean?
- The while condition **must eventually become a false value** for the statement to end (unless there is a return statement inside the while body).
- Once the while condition is evaluated, the entire body is executed.

Names and their initial values



```
i, total = 0, 0
while i < 3:
    i = i + 1
    total = total + i
```

The while condition is evaluated before each iteration

A name that appears in the while condition is changing

Executed even when is set to 3

# While loops: Caution

```
def while_ex02():
    i = 3
    while i > 0:
        print(i)
    print("blast off!")
```

```
>>> while_ex02()
3
3
3
3
3
3
3
3
3
3
3
3
3
...
```

**Question**: What Will Python Do?

**Answer**: print 3 forever! This is known as an "infinite loop". Very common bug.

Neat way to heat up your room though!

Tip: if you suspect your code is infinite-looping (eg runs for a long time without terminating), you can interrupt your program from the terminal by pressing **<CTRL> + C** (at the same time)

(Demo: 03.py:Demo00)

# Example: Prime Factorization

# Example: Prime Factorization

Each positive integer n has a unique set of prime factors whose product is n ("Fundamental Theorem of Arithmetic")

...
8  = 2 * 2 * 2
9  = 3 * 3
10 = 2 * 5
11 = 11
12 = 2 * 2 * 3
...

Challenge: how to calculate the prime factors of a number?

One approach: Find the smallest prime factor of n, divide by it, then repeat on the remaining integer

Example: $\underbrace{\mathbf{858}}_{\text{divisible by 2}} = 2 * \underbrace{\mathbf{429}}_{\text{divisible by 3}} = 2 * 3 * \underbrace{\mathbf{143}}_{\text{divisible by 11}} = 2 * 3 * 11 * \underbrace{\mathbf{13}}_{\text{divisible by 13}} = 2 * 3 * 11 * 13 * \underbrace{\mathbf{1}}_{}$

**Question**: how do we know that we are done?

**Answer**: when the remaining integer is 1

# Advice: problem solving and coding

- When faced with a coding problem: resist the urge to start writing code immediately! My advice:
- **Fully understand the problem statement**.
  - You'd be surprised how often people miss this step
- **Come up with an approach that solves the problem.**
  - Tip: work out (manually!) a few problem instances. Paper + pencil works wonders!
  - From these toy/small examples, you can flesh out the "general" approach
- Then, finally: **write the code that implements your approach**.
- What you don't want to do:
  - Dive straight into coding, and flail around because you don't know what the correct approach should be

Note: this is advice I generally give for technical coding
interviews, but also applies to C88C!

# Example: Prime Factorization

Challenge: how to calculate the prime factors of a number?

One approach: Find the smallest factor of n, divide by it, then repeat on the remaining integer

Example: $$\underbrace{\mathbf{858}}_{\text{divisible by 2}} = 2 * \underbrace{\mathbf{429}}_{\text{divisible by 3}} = 2 * 3 * \underbrace{\mathbf{143}}_{\text{divisible by 11}} = 2 * 3 * 11 * \underbrace{\mathbf{13}}_{\text{divisible by 13}} = 2 * 3 * 11 * 13 * \underbrace{\mathbf{1}}_{}$$

**Question**: how do we know that we are done?

**Answer**: when the remaining integer is 1

Phew, I have my **approach** down! Next, I'll start thinking about what my code should look like.

Approach: I want to repeatedly divide the "current active" integer via its smallest prime factor.

"repeatedly divide": `while` loop
"current active" integer: local variable that is updated within the while loop
"smallest factor": I should define a helper/utility function that computes this!

# Helper function: smallest_factor

`smallest_factor(x)` should, given an integer `x`, return the smallest factor of x.

Examples:

```
# 10 = 2 * 5
>>> smallest_factor(10)
2
# 15 = 3 * 5
>>> smallest_factor(15)
3
# 35 = 5 * 7
>>> smallest_factor(35)
5
# 13 = 13 (prime!)
>>> smallest_factor(13)
13
```

(Demo: 03.py:Demo01)

How to implement this?

Idea: start from k=2, and ask:

(k=2) is x divisible by k?
If Yes: return k
If No: increase k, and repeat

(k=3) is x divisible by k?
If Yes: return k
If No: increase k, and repeat
...

How to do "is x divisible by k" in Python?

modulo % operator!

```
>>> 6 % 1
0
>>> 6 % 2
0
>>> 6 % 3
0
>>> 6 % 4
2
>>> 6 % 5
1
>>> 6 % 6
0
```

6 % 2 = 0 means: 6 is divisible by 2

6 % 4 = 2 means: 6 is not divisible by 4, the remainder is 2