CS 61A SU26

Lecture 03: Higher-Order Functions (HOFs)

June 24, 2026

Rebecca Dang

Join pollev.com/rebeccadang831 (or scan QR code) on your phone or laptop

We'll begin at Berkeley time (10 minutes after), as per tradition!

Potpourri

5 min

- Announcements
- Quizzes
- Gateway Building
- Review: Control

Announcements

- Sometimes I'll change/update lecture slides/code after the fact - check the website!
- Lab 01, HW 01, and Hog (first project) are released!
- Projects in this course:
  - Checkpoint due date
  - Early due date (extra credit!)

Quizzes

- Highly recommend: Do orientation quiz ASAP if you haven't already
  - If you miss it, CBTF will release remote, async quiz next week
- Reserve Quiz 1 slot ASAP
  - DSP accommodations: **Please submit your letters AND login to PrairieTest (PT) ASAP** so we can process your accommodations **before** you reserve your slot
  - Accommodations do **not apply retroactively** if you already reserved
  - **If you already reserved, check that your accommodations were factored in on PT.** It should show up on the home page where it shows your reservation time.
- Quiz 1 logistics and practice quiz to be released (ideally) tomorrow

Gateway Building!

Gateway Building!

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Select ALL of the ways to fill in blanks A and B that would result in EQUIVALENT behavior when Variation 1 and 2 are executed independently (regardless of the value of x).

Fibonacci

- The Fibonacci sequence is defined as:
  - 0th Fibonacci number is 0
  - 1st Fibonacci number is 1
  - Fibonacci number n is the sum of the previous 2 numbers
- Ex: 0, 1, 1, 2, 3, 5, 8, 13, ...

Wikipedia: "**The Fibonacci numbers were first described in Indian mathematics as early as 200 BC in work by Pingala** on enumerating possible patterns of Sanskrit poetry formed from syllables of two lengths. They are **named after** the **Italian mathematician Leonardo of Pisa, also known as Fibonacci**, who **introduced the sequence to Western European mathematics** in his 1202 book *Liber Abaci*."

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Select ALL implementations of Fibonacci that are correct.

Higher-Order Functions

50 min

- Functions as data
- Lambda expressions
- HOFs
- Functions as inputs
- Functions as outputs
- Lambdas: WWPD
- Environments

Functions as data

- Unlike some other programming languages, in Python, functions can be treated as both **procedures** and **data**
  - Functions are **"first-class" objects** (more on objects later!)
- Ex: In lecture 1, we "stored" `min` and `max` in variables, `f` and `g`
- Demo: `<function object>`

Lambda expressions

- So far, we learned to use `def` to define our own functions
- Another way: **Lambda expressions**
  - `lambda <parameters>: <return expression>`
- Lambda functions are often called **anonymous functions** because they have no **intrinsic name** (unlike functions created with `def`)
- Demo
  - Lambda with 0 or more parameters
  - Storing and calling a lambda function

HOFs

- A **higher-order function (HOF)** is a function that takes in another function as input, returns a function as output, or both
- Motivation:
  - Generalization
  - Avoiding repetition
    - DRY: **D**on't **R**epeat **Y**ourself
  - "Factory" functions, e.g. `make_discount` (later slide)

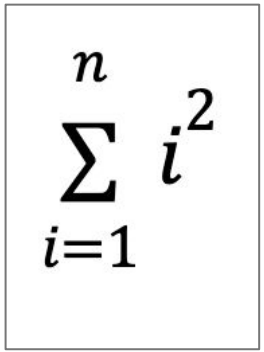
Functions as inputs (1 of 3)

```
def sum_of_squares(n):
    i = 1
    total = 0
    while i <= n:
        total += i ** 2
        i += 1
    return total
```

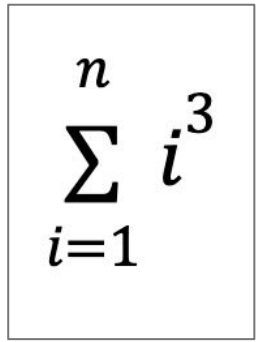
```
def sum_of_cubes(n):
    i = 1
    total = 0
    while i <= n:
        total += i ** 3
        i += 1
    return total
```

Functions as inputs (2 of 3)

```
def sum_of_squares(n):
    i = 1
    total = 0
    while i <= n:
        total += i ** 2
        i += 1
    return total
```




```
def sum_of_cubes(n):
    i = 1
    total = 0
    while i <= n:
        total += i ** 3
        i += 1
    return total
```



Functions as inputs (3 of 3)

```
def summation(n, term):
    i = 1
    total = 0
    while i <= n:
        total += term(i)
        i += 1
    return total
```

Functions as outputs (1 of 2)

```
def half_off(price):
    return price * 0.5

def ten_percent_off(price):
    return price * 0.9

def twenty_percent_off(price):
    return price * 0.8
```

```
def compute_new_price(discount, price):
    return discount(price)
```

Functions as outputs (2 of 2)

```
def make_discount(percent_off):
    def discount(price):
        return price * (1 - percent_off)
    return discount
```

```
def compute_new_price(discount, price):
    return discount(price)
```

```
>>> half_off = make_discount(0.5)
>>> compute_new_price(half_off, 20)
10.0
```

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Lambdas: What would Python display?

Lambdas WWPD: What happened? (1 of 5)

```
(lambda x, y: lambda z: x * y + z)(1, 2)(3)
```

Outer function called directly

Operands of outer function (x and y)

Lambdas WWPD: What happened? (2 of 5)

```
(lambda x, y: lambda z: x * y + z)(1, 2)(3)
```

Return expression of outer function

Lambdas WWPD: What happened? (3 of 5)

```
(lambda x, y: lambda z: x * y + z)(1, 2)(3)
```

Since the inner function was returned, this entire thing evaluates to that inner function

Lambdas WWPD: What happened? (4 of 5)

```
(lambda x, y: lambda z: x * y + z)(1, 2)(3)
```

Return expression of inner function (has access to x and y from its parent frame)

Operand of inner function (z)

Lambdas WWPD: What happened? (5 of 5)

```
(lambda x, y: lambda z: x * y + z)(1, 2)(3)
```

```
def outer(x, y):
    def inner(z):
        return x * y + z
    return inner
outer(1, 2)(3)
```

Tip: Rewrite `lambda` into `def` if you're confused on what it's doing! Or do the opposite if we ask you to write a nested `lambda`.

Environments

- In a previous slide, we talked about "parent frame"
  - But what's a parent? And what's a frame?
  - How do we figure out what data we have access to in a program? (e.g. **variable scope**)
- An **environment** is a sequence of **frames**
  - You might also hear programmers call it a "**stack**" of frames, hence the term **"stack trace"** from Lecture 1: Exceptions, or **"stack overflow"** (we'll talk about that next week!)
- A **frame** is a set of bindings between names and values
- All Python programs begin in the special **global frame**

Name Lookup Rules

- A function's **parent frame** is the **frame in which the function is <u>defined</u>** (not the frame in which it's called)
- Each time we call a function, we open a new **local frame**
- When we try to retrieve the value of a name, we first look in the current frame
  - If it's not there, we recursively check if it exists in the **parent frame**
  - If we go all the way up to the global frame and it's still not there, throw a `NameError`
- ⚠ **Caution:** You can't update a variable outside your local frame (results in `UnboundLocalError`)

When poll is active respond at **PollEv.com/rebeccadang831**

Activate this Poll with the Poll Everywhere Live app.

If video conferencing, you must be sharing your full screen to share the activated poll.

Environments: What would Python display?

5 min break

HOFs Practice

20 min

- Combination
- Multiply Triple

Practice: Combination

- Implement a higher-order function `combination(n, initial, combiner, term)` See the doctests for examples.
- Download starter code `03.py` from the course website under Lecture 3
- Run doctests with `python3 -m doctest 03.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?

Practice: Multiply Triple

- **Using only functions with a single argument**, implement the function `multiply_triple` so that you can use it to return the product of 3 numbers x, y, and z.
- Download starter code `03.py` from the course website under Lecture 3
- Run doctests with `python3 -m doctest 03.py`
- 5 min: Try implementing it yourself
- 5 min: Discuss with someone next to you and compare your implementations
  - What worked?
  - What didn't?
  - Why?