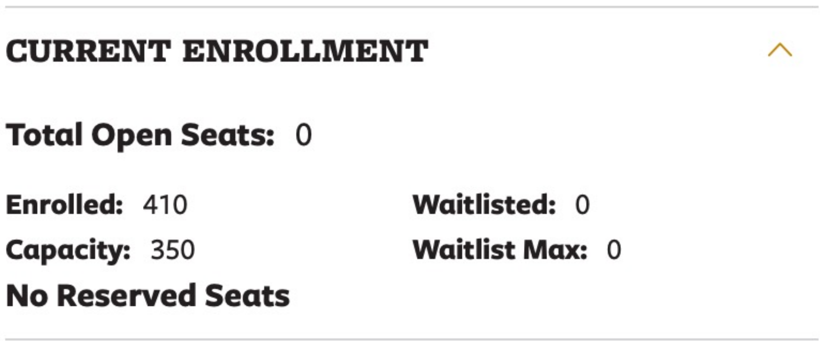
UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

HOFs & Environment Diagrams

Announcements

- The weeks-long saga is over!
- Early next week:
  - TWO new sections added
  - Will send out a form for transferring work from 61A
  - Will give extensions for Labs 1, 2, 3 and Homework 1, 2
- if I don't respond to your email about enrollment, I'm sorry.


CURRENT ENROLLMENT

Total Open Seats: 0

Enrolled: 410   **Waitlisted:** 0
Capacity: 350   **Waitlist Max:** 0
No Reserved Seats

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

HOFs and Lists

Functional List Operations

- Goal: Transform a list, and return a new result
- We'll use 3 functions that are hallmarks of functional programming
- Each of these takes in a function and a sequence

Function Name	Action	Input arguments	Input Fn. Returns	Output List
`map`	Transform every item	1 argument (each item)	"Anything", a new item	List of the same length, but possibly new values
`filter`	Return a list with fewer items	1 argument (each item)	A Boolean	List with possibly fewer items, but values are the same
`reduce`	"Combine" items together	2 arguments (current item, and the previous result)	Type should match the type each item	Usually a "single" item

Filter and Non-Boolean Functions

```
>>> list(filter(add_2, range(10)))
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> if 0:
...     print("0 is a true value")
... else:
...     print("0 is a false value")
...
0 is a false value
```

Why is 0 in the output of our filter?

Filter and Non-Boolean Functions

```
>>> [ x for x in range(10) if add_2(x) ]
```

Why is 0 in the output of our filter?

Filter calls our function, but always returns the original value!

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Lists & Higher Order Functions
Reduce

Learning Objectives

- Learn three new common Higher Order Functions:
  - map, filter, reduce
- These each apply a function to a sequence (list) of data

- Reduce: “Combines” items together, probably doesn’t return a list.
  - Input: A 2 item function and a sequence
  - A single value

REDUCE

`reduce(function, list_of_inputs)`

Successively **combine** items of our sequence

- function: add(), takes 2 inputs gives us 1 value.

Inputs (Domain):

- Function, with 2 inputs
- Sequence

Output (Range):

- An item, the type is the output of our function.

Note: We must import reduce from `functools`!

```
def reduce(function, sequence):
    result = function(sequence[0], sequence[1])
    for index in range(2, len(sequence)):
        result = function(result, sequence[index])
    return result
```

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Lists & Higher Order Functions
Acronym

Today’s Task: Acronym

```
Input: "The University of California at Berkeley"

Output: "UCB"

def acronym(sentence):
       """YOUR CODE HERE"""
```

P.S. Pedantry alert: This is really an *initialism* but that's rather annoying to say and type. ☺ (However, the code we write is the same, the difference is in how you pronounce the result.) The more you know!

Today's Task: Acronym

```
def acronym(sentence):
    """
    >>> acronym("The University of California at Berkeley")
    "UCB"
    """
    words = sentence.split()
    return reduce(add, map(first_letter, filter(long_word, words)))
```

P.S. Pedantry alert: This is really an *initialism* but that's rather annoying to say and type. ☺ (However, the code we write is the same, the difference is in how you pronounce the result.) The more you know!

Functional List Operations

- Goal: Transform a list, and return a new result
- We'll use 3 functions that are hallmarks of functional programming
- Each of these takes in a function and a sequence

Function Name	Action	Input arguments	Input Fn. Returns	Output List
`map`	Transform every item	1 argument (each item)	"Anything", a new item	List of the same length, but possibly new values
`filter`	Return a list with fewer items	1 argument (each item)	A Boolean	List with possibly fewer items, but values are the same
`reduce`	"Combine" items together	2 arguments (current item, and the previous result)	Type should match the type each item	Usually a "single" item

UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Environment Diagrams

Python Tutor Examples: compose

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

add_2 = make_adder(2)
add_3 = make_adder(3)
x = add_2(5)
y = add_3(x)
```

Environment Diagrams

- Organizational tools that help you understand code
- **Terminology:**
  - **Frame:** keeps track of variable-to-value bindings, each function call has a frame
  - **Global Frame:** global for short, the starting frame of all python programs, doesn't correspond to a specific function
  - **Parent Frame:** The frame of where a function is defined (default parent frame is global)
  - **Frame number:** What we use to keep track of frames, f1, f2, f3, etc
  - **Variable** vs **Value**: x = 1. x is the **variable**, 1 is the **value**

Environment Diagrams Reminders

1. Always draw the global frame first
2. When evaluating assignments (lines with single equal), always evaluate right side first
3. When you CALL a function MAKE A NEW FRAME!
4. When assigning a primitive expression (number, boolean, string) write the value in the box
5. When assigning anything else (lists, functions, etc.), draw an arrow to the value
6. When calling a function, name the frame with the intrinsic name – the name of the function that variable points to
7. The parent frame of a function is the frame in which it was defined in (default parent frame is global)
8. If the value for a variable doesn't exist in the current frame, search in the parent frame

Demo

Example 1:

- Primitives and Functions: Environment Diagram Python Tutor:

Example 2:

- make_adder Higher Order Function: Environment Diagram Python Tutor Link

Example 3:

- Compose Python Tutor Link

Example 1

```
a = "chipotle"
b = 5 > 3
c = 8

def foo(c):
    return c - 5

def bar():
    if b:
        a = "taco bell"

result1 = foo(10)
result2 = bar()
```

Example 2

```
def make_adder(n):
    def adder(k):
        return k + n
    return adder

n = 10
add_2 = make_adder(2)
x = add_2(5)
```

Python Tutor Examples

```
add_2 = make_adder(2)
add_3 = make_adder(3)

x = add_2(2)
def compose(f, g):
    def h(x):
        return f(g(x))
    return h

add_5 = compose(add_2, add_3)
z = add_5(x)
```

Environment Diagram Tips / Links

- NEVER draw an arrow from one variable to another.
- Useful Resources:
  - http://markmiyashita.com/cs61a/environment_diagrams/rules_of_environment_diagrams/
  - http://albertwu.org/cs61a/notes/environments.html