Computational Structures in Data Science

HOFs & Environment Diagrams

Berkeley
UNIVERSITY OF CALIFORNIA

Announcements / Info

- Reminder: Please only request extensions if >= 2 days or joining late
- Gradescope / Grading:
  - If you run into issues, please resubmit
  - When you post on Ed, please include a link to the submission.
  - Remember to run okpy on your computer!
    - `python3 ok --all`
    - `python3 ok --all -interactive`
    - `python3 ok -local`
    - `python3 ok -help`
- Maps project out soon!
  - **Recommended: Find a Partner!**

Computational Structures in Data Science

Joining Late

- Please request extensions for missing work.
- You can request extensions on lecture self-checks, but only through lecture 6 (i.e. Monday's class)
- Don't recommend choosing the attendance option.

Computational Structures in Data Science

HOFs and Sequences

Berkeley
UNIVERSITY OF CALIFORNIA

Functional Sequence (List) Operations

- Goal: Transform a *sequence*, 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 as arguments

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

Today’s Task: Acronym

```
def acronym(sentence):
    """
    >>> acronym("The University of California at Berkeley")
    "UCB"
    """
    words = sentence.split()
    return "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

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

Output: "UCB"
def acronym(sentence):
    """ (Some doctests)
    """
    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!

Acronym With HOFs

What is we want to control the filtering method?

```
def keep_words(word):
    specials = ['Los']
    return word in specials or long_word(word)

def acronym_hof(sentence, filter_fn):
    words = sentence.split()
    return reduce(add, map(first_letter, 
filter(filter_fn, words)))
```

Functional Sequence (List) Operations

- Goal: Transform a *sequence*, 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 as arguments

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

Computational Structures in Data Science

Functions That Return Functions

Berkeley
UNIVERSITY OF CALIFORNIA

Learning Objectives

- Learn how to use and create higher order functions:
- Functions can be used as data
- Functions can accept a function as an argument
- Functions can return a new function

Review: What is a Higher Order Function?

- A function that takes in another function as an argument

OR

- A function that returns a function as a result.

Higher Order Functions

- **A function that returns (makes) a function**

```
def leq_maker(c):
    def leq(val):
        return val <= c
    return leq
```

```
>>> leq_maker(3)
<function leq_maker.<locals>.leq at 0x1019d8c80>


>>> leq_maker(3)(4)
False

>>> [x for x in range(7) if leq_maker(3)(x)]
[0, 1, 2, 3]
```

Computational Structures in Data Science

Environment Diagrams

Why focus on environments?

- Environments are a simplification of why Python *actually* does
- Focus on building intuition for what will happen when you run code
- Sometimes tedious, but the practice helps you solve hard questions
  - In 88C (or 61A), even our hard questions are pretty short
  - Outside of class, things can get complex quickly.
- Every programming language is a bit different, but these rules are quite common
- I understand if you don't like them now
  - But you will be tested on them. ☺

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 Rules

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

Python Tutor 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()
```

Python Tutor 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 Example #3

```
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)
```

Demo

Example 1:

- make_adder Higher Order Function: Environment Diagram Python Tutor Link

Example 2:

- Primitives and Functions: Environment Diagram Python Tutor:

Example 3:

- Compose Python Tutor Link

A Real Example – Optional

```
from datetime import datetime, timedelta, time

def assignment_date_counter(first_date, frequency=7):
    asn_date = first_date - timedelta(days=frequency)
    def next_date(skip_weeks=0):
         nonlocal asn_date # We don't cover this in 88C!
         asn_date = asn_date + timedelta(frequency * (1 + skip_weeks))
        return asn_date
    return next_date

next_homework = assignment_date_counter(datetime(2026, 1, 27))
next_homework()
next_homework()
next_homework()
```

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

Why focus on environments?

- Environments are a simplification of why Python actually does
- Focus on building intuition for what will happen when you run code
- Sometimes tedious, but the practice helps you solve hard questions
  - In 88C (or 61A), even our hard questions are pretty short
  - Outside of class, things can get complex quickly.
- Every programming language is a bit different, but these rules are quite common