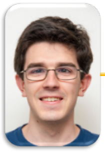


UC Berkeley EECS
Lecturer
Michael Ball

Computational Structures in Data Science

Lecture 10: Midterm Review

Announcements

- Midterm Wednesday!
  - 7-9pm
  - Look for room info on Piazza.
  - Accommodations have been emailed.
    - If you have not gotten an email post a private note
- Homework, do a practice midterm
  - Upload to Gradescope.
  - We will post a rubric online to grade yourself.
- Cheat Sheet Info:
  - 1 page, double-sided
  - Must be hand written!

Cheat Sheet Tips

- Writing by hand helps with memory
- Review the sheet we give you
- Environment Diagram rules!
- Confidence boosts / reminders to slow down
- https://docs.google.com/presentation/d/1i1Ojc8MJpNh195O-sf6ZDAf0urRYygdv0OZ7EYUWPYI/edit#slide=id.p

You've come so far!

- Data type: values, literals, operations,
  - e.g., int, float, string
- Expressions, Call expression
- Variables
- Assignment Statement
- Sequences: tuple, list
  - indexing
- Call Expressions
- Function Definition Statement
- Conditional Statement
- Iteration:
  - data-driven (list comprehension)
  - control-driven (for statement)
  - while statement
- Higher Order Functions
  - Functions as Values
  - Functions with functions as argument
  - Assignment of function values
- Higher order function patterns
  - Map, Filter, Reduce
- Recursion

On Computer Science Exams

In computer science exams, we try to assess the student’s understanding of concepts and his or her ability to practically apply these.

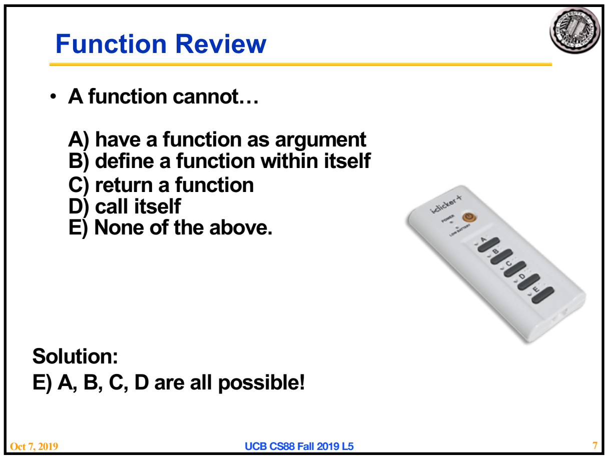
- In CS, we do not:
  - require extensive memorization (e.g. we allow cheat sheet)
  - require a lot of reading
  - require essay writing skills

In CS, we do:

- require the ability to translate a given textual problem into programming code
- require you to be able to read other people’s code
- value solutions that are almost right over no solution
- accept solutions we did not think about if they work
- prioritize math (logic) and science (experiment) over opinion or authority

How to prepare for a CS exam

- Explain the content of the computational concepts toolbox to somebody else
  - Describe the concept
  - What is an example of using it?
  - When does it not work? Corner cases?
  - Why does it exist?
- Practice programming:
  - Play around with the examples from lecture, lab, homework
  - Think about your own similar examples
- In the exam:
  - Make sure you understand the question: What is the given input? What is the required output?
  - Think of easy cases first (e.g. n=1?).
  - What is the iteration/recursion doing (e.g. i=i+1)?
  - What are corner cases that need explicit handling (e.g. division by zero, negative numbers, empty list)?



Function Review

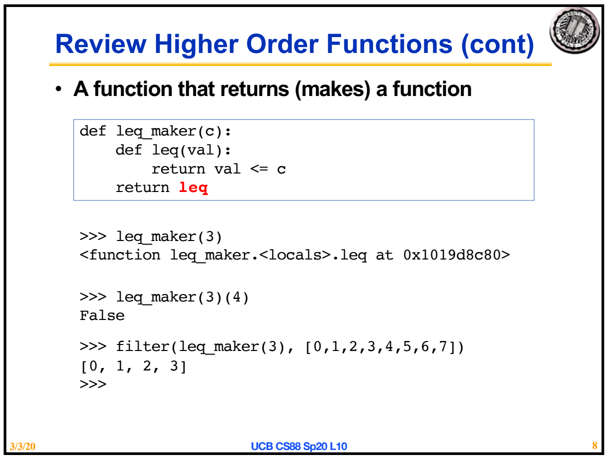
- A function cannot…

A) have a function as argument
B) define a function within itself
C) return a function
D) call itself
E) None of the above.

Solution:
E) A, B, C, D are all possible!

Oct 7, 2019 UCB CS88 Fall 2019 L5 7

7



Review Higher Order Functions (cont)

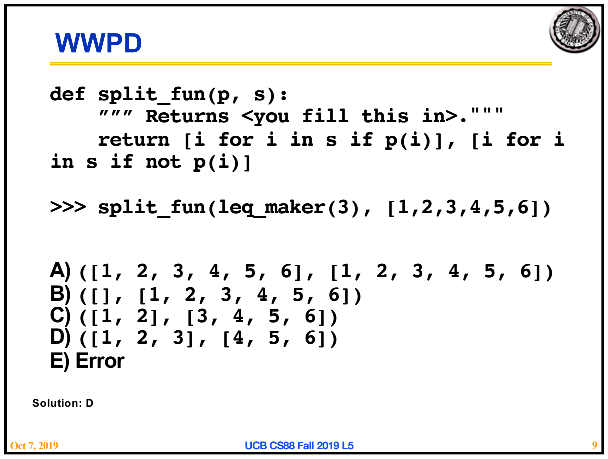
- A function that returns (makes) a function

```
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

>>> filter(leq_maker(3), [0,1,2,3,4,5,6,7])
[0, 1, 2, 3]
>>>
```

3/3/20 UCB CS88 Sp20 L10 8

8



WWPD

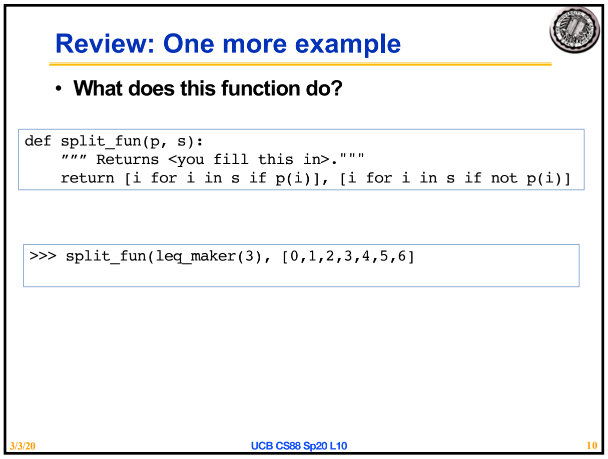
```
def split_fun(p, s):
    """ Returns <you fill this in>."""
    return [i for i in s if p(i)], [i for i
in s if not p(i)]

>>> split_fun(leq_maker(3), [1,2,3,4,5,6])
```

A) `([1, 2, 3, 4, 5, 6], [1, 2, 3, 4, 5, 6])`
B) `([], [1, 2, 3, 4, 5, 6])`
C) `([1, 2], [3, 4, 5, 6])`
D) `([1, 2, 3], [4, 5, 6])`
E) Error

Solution: D

Oct 7, 2019 UCB CS88 Fall 2019 L5 9

9



Review: One more example

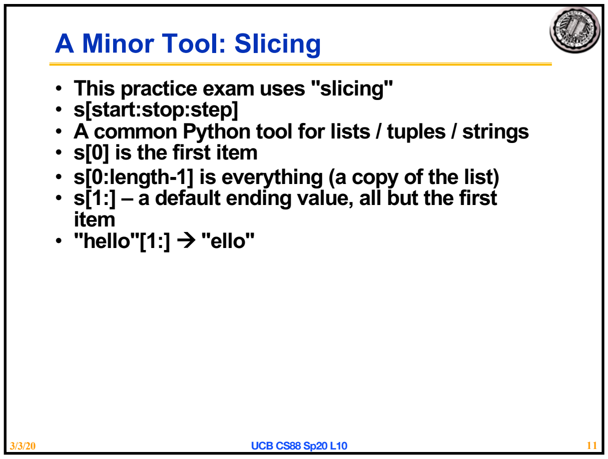
- What does this function do?

```
def split_fun(p, s):
    """ Returns <you fill this in>."""
    return [i for i in s if p(i)], [i for i in s if not p(i)]
```

```
>>> split_fun(leq_maker(3), [0,1,2,3,4,5,6]
```

3/3/20 UCB CS88 Sp20 L10 10

10

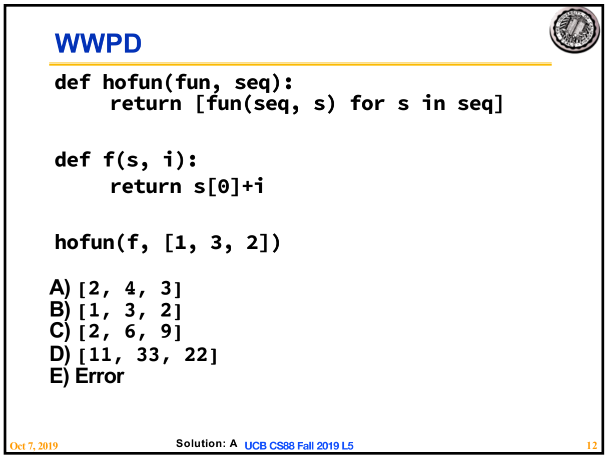


A Minor Tool: Slicing

- This practice exam uses "slicing"
- s[start:stop:step]
- A common Python tool for lists / tuples / strings
- s[0] is the first item
- s[0:length-1] is everything (a copy of the list)
- s[1:] – a default ending value, all but the first item
- "hello"[1:] → "ello"

3/3/20 UCB CS88 Sp20 L10 11

11



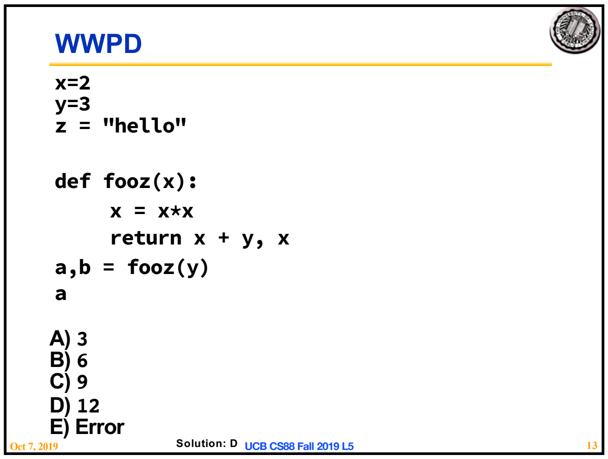
WWPD

```
def hofun(fun, seq):
    return [fun(seq, s) for s in seq]


def f(s, i):
    return s[0]+i


hofun(f, [1, 3, 2])
```

A) `[2, 4, 3]`
B) `[1, 3, 2]`
C) `[2, 6, 9]`
D) `[11, 33, 22]`
E) Error

Oct 7, 2019 Solution: A UCB CS88 Fall 2019 L5 12

12



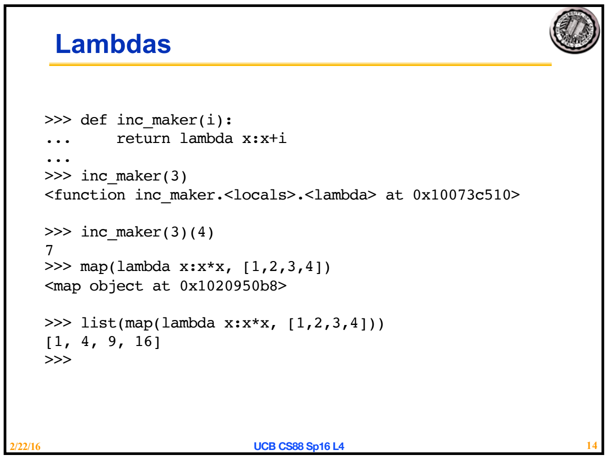
WWPD

```
x=2
y=3
z = "hello"

def fooz(x):
     x = x*x
     return x + y, x
a,b = fooz(y)
a
```

A) 3
B) 6
C) 9
D) 12
E) Error

Oct 7, 2019 Solution: D UCB CS88 Fall 2019 L5 13

13

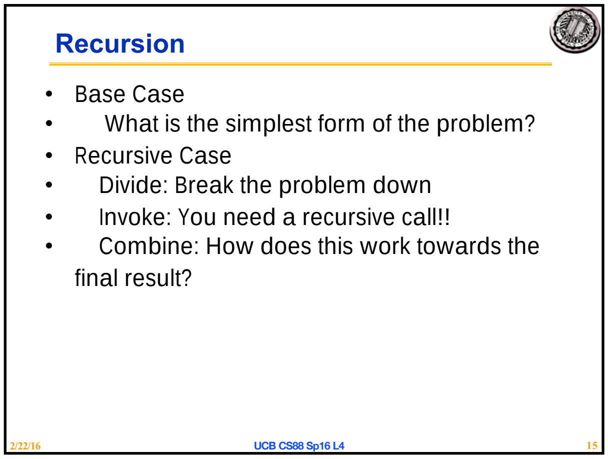


Lambdas

```
>>> def inc_maker(i):
...     return lambda x:x+i
...
>>> inc_maker(3)
<function inc_maker.<locals>.<lambda> at 0x10073c510>

>>> inc_maker(3)(4)
7
>>> map(lambda x:x*x, [1,2,3,4])
<map object at 0x1020950b8>

>>> list(map(lambda x:x*x, [1,2,3,4]))
[1, 4, 9, 16]
>>>
```

2/22/16 UCB CS88 Sp16 L4 14

14

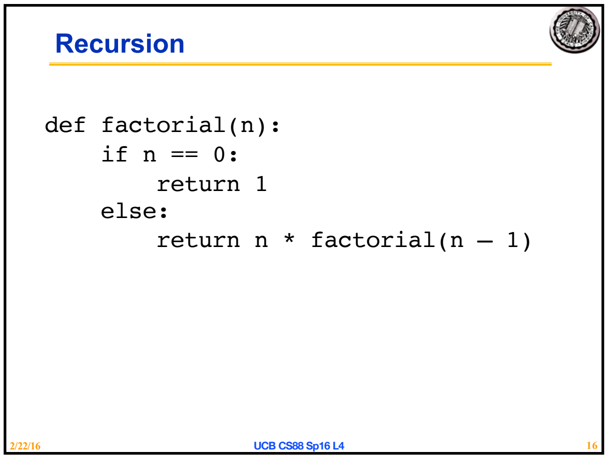


Recursion

- Base Case
- What is the simplest form of the problem?
- Recursive Case
- Divide: Break the problem down
- Invoke: You need a recursive call!!
- Combine: How does this work towards the final result?

2/22/16 UCB CS88 Sp16 L4 15

15



Recursion

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n – 1)
```

2/22/16 UCB CS88 Sp16 L4 16

16