

# Welcome to Data C88C!

---

## **Lecture 06: Functional Abstraction**

Tuesday, July 1st, 2025

Week 2

Summer 2025

Instructor: Eric Kim ([ekim555@berkeley.edu](mailto:ekim555@berkeley.edu))

# Announcements

---

- Lab03, HW03 released today!
- Due dates
  - Lab01, Lab02, HW01 due: Tues July 1st, 11:59 PM PST
  - Lab03, HW03 due: Sun July 6th, 11:59 PM PST

# Important University Deadlines

---

- **Add/Drop deadline: Thursday July 3rd**

- Advice: if you are feeling extremely behind AND you don't think that you can catch up, consider taking this course another semester
  - Or: take an alternate course like Data 8 / CS 10 first
- Temperature Check: by end of Week 02, feel comfortable with writing and understanding Python code
  - ex: know how to define and call functions, how to write while loops
  - If you're still struggling with Python syntax by the end of Week 02, you are behind and will likely struggle in this course without significant correction to your study habits.

- **Change grade option deadline: Friday August 1st**

- eg letter grade -> Pass/No Pass
-

# Xlab

---

“The Experimental Social Science Laboratory (Xlab) invites you to participate in research! All experiments conducted at Xlab are computerized social science studies that typically last between fifteen minutes to an hour. We have many online studies you can partake in. Participants earn \*\$20/hour\* on average. For more information, visit [xlab.berkeley.edu](http://xlab.berkeley.edu) or watch our short video! To sign up, visit [berkeley.sona-systems.com](http://berkeley.sona-systems.com)

We look forward to seeing you at the Xlab! Do science. Get paid.”

---

# Lecture Overview

---

- More Environment Diagrams practice
- Lambdas, HOFs
- Conditional expressions (bool)

# Zero-Argument Functions

# Dice Functions

A dice function returns an integer that is the outcome of rolling a six-sided die once. Ex: `six_sided()`

Implement `repeat(n, dice)`, which returns the # of times in n rolls that an outcome repeats.

5 3 3 4 2 1 6 5 3 4 2 2 2 4 4 3 4 3 5 5

`repeat(20, six_sided) -> 5`

```
>>> six_sided()
3
>>> six_sided()
5
```

```
def repeats(n, dice):
    count = 0
    previous = 0
    -----
    while n:
        outcome = dice()
        if previous == outcome:
            count += 1
            previous = outcome
        -----
        n -= 1
    return count
```

f1: repeats [parent=Global]	
n	20
dice	func ...
count	0
previous	0
outcome	3
Return value	

# Higher-Order Loops

(Demo PythonTutor: [\[link\]](#))



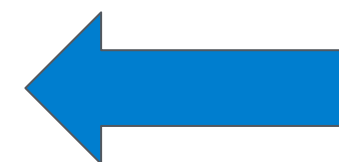
# Conditional Expressions Practice

# True and False Values

---

The built-in `bool(x)` returns `True` for true `x` and `False` for false `x`.

```
>>> bool(0)
False
>>> bool(-1)
True
>>> bool(0.0)
False
>>> bool(' ')
True
>>> bool('')
False
>>> bool(False)
False
>>> bool(print('fool'))
fool
False
```



`'None'` is a False-y value!

**Rule of thumb:** each datatype (int, float, string) has one False-y value (eg `0`, `0.0`, `' '`) and all other values are True-y values.

# True and False with Conditional Statements (if, while)

---

```
def is_even(x):  
    return x % 2 == 0  
val = is_even(2) # True
```

Somewhat redundant (not  
incorrect though)

```
if val == True:  
    print("In True block")  
else:  
    print("In False block")
```

(And similar for while statement  
condition check: `while x:`)

More idiomatic

```
if val:  
    print("In True block")  
else:  
    print("In False block")
```

Can think of above as (sort of) calling  
`bool()` for you as a convenience

```
if bool(val):  
    print("In True block")  
else:  
    print("In False block")
```

# Fall 2022 Midterm 1 Question 1a

## 1. (4.0 points) What Would Python Display?


Assume the following code has been executed.



```
bear = -1
oski = lambda print: print(bear)
bear = -2
```

```
s = "Knock"
```

For each expression below, write the output **displayed by the interactive Python interpreter** when the expression is evaluated. The output may have multiple lines. If an error occurs, write “Error”, but include all output displayed before the error. If evaluation would run forever, write “Forever”. To display a function value, write “Function”. The interactive interpreter displays the value of a successfully evaluated expression, unless it is `None`.

(a) (1.0 pt) `(3 and 4) - 5`

- ☐ -2
-  ☒ -1
- ☐ 1
- ☐ 2
- ☐ Error

`(3 and 4) - 5`  
  
Evals to: 4  
  
Evals to: -1

**Rule:** Python evaluates `and` expressions in Left-to-Right order, and returns either the first False-y value, or returns the right-most True-y value

Exam PDF: [\[link\]](#)

Tip: all previous midterms/finals (and their solutions!) can be found here:

dataC88C: [\[link\]](#) cs61a: [\[link\]](#)

## Conditional Expressions: Short Circuiting

---

```
>>> (3 and 0 and 4)
0
>>> ('' and 2 and -1)
''
```

**Rule:** Python evaluates ``and`` expressions in left-to-right order, and returns either the **first** False-y value, or the **last** True-y value

```
>>> (3 or 4)
3
>>> (3 or '' or 4)
3
>>> (3 or 0 or 4)
3
>>> ('' or 2 or 0)
2
>>> ('' or 0)
0
```

**Rule:** Python evaluates ``or`` expressions in left-to-right order, and returns either the **first** True-y value, or the **last** False-y value.

Python may or may not evaluate all terms in the and/or expressions. This early-termination is known as "**short circuiting**" in programming languages.

# Lambda Expressions Practice



# Lambda and Def

---

Any program containing lambda expressions can be rewritten using def statements.

```

                                twice
                                square
>>> (lambda f: lambda x: f(f(x))) (lambda y: y * y) (3)
81

>>> def twice(f):
...     def g(x):
...         return f(f(x))
...     return g
...
>>> def square(y):
...     return y * y
...
>>> twice(square)(3)
81
```

## Fall 2022 Midterm 1 Question 4(a)

---

**(2.0 pt)** Choose **all** correct implementations of `funsquare`, a function that takes a one-argument function `f`. It returns a one-argument function `f2` such that `f2(x)` has the same behavior as `f(f(x))` for all `x`.

```
>>> triple = lambda x: 3 * x
>>> funsquare(triple)(5)  # Equivalent to triple(triple(5))
45
```

A: `def funsquare(f):`  
    `return f(f)`

B: `def funsquare(f):`  
    `return lambda: f(f)`

C: `def funsquare(f, x):`  
    `def g(x):`  
        `return f(f(x))`  
    `return g`

☒ D: `def funsquare(f):`  
    `return lambda x: f(f(x))`

E: `def funsquare(f, x):`  
    `return f(f(x))`

☒ F: `def funsquare(f):`  
    `def g(x):`  
        `return f(f(x))`  
    `return g`



# Spring 2020 Midterm 1 Question 1

---

```
>>> snap = lambda chat: lambda: snap(chat)
```

```
>>> snap, chat = print, snap(2020)
```

*A: What is displayed here?*

```
>>> chat()
```

*B: What is displayed here?*

(Answer)

# A

(Nothing is printed)

# B

2020

PythonTutor [[link](#)]

# Call Expressions

# Assigning Names to Values

---

There are three ways of assigning a name to a value:

- Assignment statements (e.g., `y = x`) assign names in the current frame
- Def statements assign names in the current frame
- Call expressions assign names in a new local frame

```
h = lambda f: lambda x: f(f(x))  
h(abs)(-3)
```

```
f = abs  
x = -3  
f(f(x))
```

```
h = lambda f: f(f(x))  
x = -3  
h(abs)
```