

# Discussion 02

## Environment Diagrams, Higher-Order Functions

Aditya Balasubramanian

`aditbala [at] berkeley [dot] edu`

# Announcements

# Environment Diagrams



# Call Expressions

- Syntax: `function_name(arg1, arg2, ...)`
- `sum(square(2), 2 + 2)`
  - Create new frame for call expression
  - Steps for evaluating:
    1. Evaluate operator (function)
      - See if it exists
        - `sum` adds two numbers
    2. Evaluate operands (args)
      - simplify args
        - `square(2)` -> `4`
        - `2 + 2` -> `4`
    3. Apply operator to the operands
      - `sum(4, 4)` -> `8`

# Creating New Frames

- Give frame with unique index ( `f1` , `f2` , `f3` )
- Label frame with name of function object
  - not always the variable name
- Label function's parent
- [INCLUDE EXAMPLE]

# Variable Lookup

- Start in current frame
- If variable does not exist, search parent frame
- If variable still does not exist, continue looking in parent frames
- If variable does not exist, program errors

**[ADD QUESTIONS]**

# Lambda

# Functions $\lambda$



# lambda Functions $\lambda$

- What are they?
  - A quicker and simpler way to define a function
  - Can also be used as the operator for a function
- Why use them?
  - Useful for scenarios in which you only want to use a function once and never again
- Syntax
  - written in 1 line
  - `lambda <args> : <body>`

# lambda Function Examples Pt. 1

```
def add_and_square(x, y, z):  
    return (x + y + z) ** 2
```

```
lambda_mul_by_three = lambda x, y, z : (x + y + z) ** 2
```

```
def error():  
    return 1 + 2 / 0
```

```
lambda_mul_by_three = lambda : 1 + 2 / 0
```

## lambda Examples Pt. 2

```
>>> lambda x : x // 3
Function
>>> (lambda x : x / 3)(5)
1
>>> func = lambda x: lambda y: lambda z: x * y * z
>>> func(3)(7)(5)
105
>>> lambda()
```

FIX TO SHOW INDIVIDUALLY

# Higher Order Functions

# Higher Order Functions (HOF)

- What are they?
  - Functions that either return functions as output or take in other functions as inputs
- Why use them?
  - When you want to use a function within another function
  - Treat them as an object
- Important Note
  - Let's see we have function `foo()` that takes in zero parameters
  - `foo` refers to the function object and is **NOT** calling the function
  - `foo()` shows that we are actually calling the function

# HOF Function as Input Example

```
>>> def exec_func(func, a):  
    return func(a)
```

```
>>> exec_func(lambda x : x * 4, 4)  
16
```

```
>>> exec_func(lambda x : pow(x, 2), 2)  
4
```

# HOF Function as Output Example

```
>>> def compose1(f, g):  
    def h(x):  
        return f(g(x))  
    return h  
  
>>> compose1(mul, )
```