# Discussion 02

**Environment Diagrams, Higher-Order Functions** 

Aditya Balasubramanian aditbala [at] berkeley [dot] edu

#### Clarifications from last discussion

- Priority of booleans
  - 1. not
  - 2. and
  - 3. or
- Return the last thing you evaluate
- 4 and not 5 or 8 or 9
- Lab mini-lecture



# Announcements

- Hog is due Friday 9/9 @ 11:59pm.
  - Early submission bonus point for finishing by Thursday 9/8 @
     11:59pm.
- Homework 2 is due Thursday 9/8 @ 11:59pm.
- Midterm 1 is Monday 9/12 7pm-9pm at various locations around campus.
  - Read the midterm logistics post!
  - You will be provided with a printed copy of the Midterm 1 Study Guide.
  - Your assigned seat and room will be sent to you on Sunday 9/11.
  - Arrive at 7pm if you can; the 110-minute, 30-point exam will begin at 7:10pm.
  - o Complete this form for any exam alteration request (left-handed desk, DSP accommodation, conflict, etc.) by 11:59pm Friday 9/9.

### Midterm Advice



# Environment Diagrams

#### **Enviroment Diagrams**

- What are they?
  - A way to model how our program runs line by line
  - Keep track of variables, function calls and what they return, etc.
- Why use them?
  - Can help us understand where there is a bug in program (debugging)
  - Useful for other questions (WWPD, coding)
  - Exam points!

#### **Important Concepts**

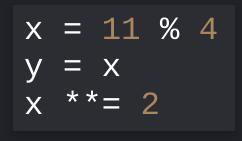
- Expressions
  - Evaluate to values
  - 0 1 + 1 -> 2
- Statements
  - Bind names to values
  - Names
    - def statements, assignment statements, variable names
  - Values
    - numbers, strings, functions, or other objects
  - $\circ$  x = 2
  - doesn't return anything

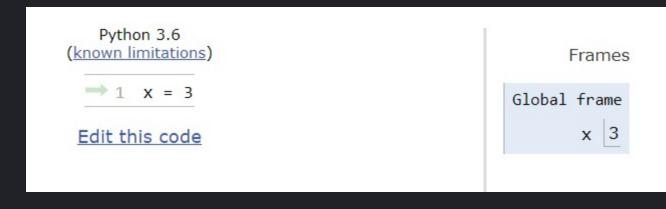
#### Frames

- What are they?
  - o Frames list the bindings of variables and their corresponding value
- What are they used for?
  - Used to look up the value of a variable
- Global Frame always exists

# **Assignment Statements**

- Assignment statements
   (denoted by =) creates new
   binding in frame
- Evaluate right side
   completely before binding
   to left side
- Example





#### Def statements

- def statements are used to bind function objects to a variable
- Binding name is function name
- Parent of the function is frame where function is defined
- Keep track of name, parameters, parent frame
- Example

```
x = 3
def square(x):
    return x ** 3
square(x)
```

#### Def statements

- Only bind, NO execution until function is called
  - o def foo(): -> define function called foo with no parameters
  - o foo() -> execute foo

#### 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
      - square(2) -> 4 squares number
      - sum adds two numbers
  - 2. Evaluate operands (args)
    - simplify args
      - **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
  - o not always the variable name
- Label function's parent (frame in which it is defined in)
- Every function has return value
  - Return value can be None

#### 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

### Question 1 (5 minutes)

Let's put it all together! Draw an environment diagram for the following code. You may not have to use all of the blanks provided to you.

```
def double(x):
    return x * 2

hmmm = double
wow = double(3)
hmmm(wow)
```

# Question 2 (walkthrough)

Draw the environment diagram that results from executing the code below.

(note: evaluate, then assign)

```
def f(x):
    return x
def g(x, y):
    if x(y):
         return not y
    return y
x = 3
x = g(f, x)
f = g(f, Qi)ya Balasubramanian
```

# lambda Functions \(\lambda\)

#### lambda Functions λ

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

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

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

```
>>> lambda x : x // 3
>>> (lambda x : x // 3)(5)
>>> wow = lambda x: lambda y: lambda z: x * y * z
>>> wow(3)(7)(5)
>>> (lambda x: x(3,4))(lambda a,b: a ** b)
```

```
>>> lambda x : x // 3
Function
>>> (lambda x : x // 3)(5)

>>> wow = lambda x: lambda y: lambda z: x * y * z
>>> wow(3)(7)(5)

>>> (lambda x: x(3,4))(lambda a,b: a ** b)
```

```
>>> lambda x : x // 3
Function
>>> (lambda x : x // 3)(5)
1
>>> wow = lambda x: lambda y: lambda z: x * y * z
>>> wow(3)(7)(5)
>>> (lambda x: x(3,4))(lambda a,b: a ** b)
```

```
>>> lambda x : x // 3
Function
>>> (lambda x : x // 3)(5)
1
>>> wow = lambda x: lambda y: lambda z: x * y * z
>>> wow(3)(7)(5)
105
>>> (lambda x: x(3,4))(lambda a,b: a ** b)
```

```
>>> lambda x : x // 3
Function
>>> (lambda x : x // 3)(5)
1
>>> wow = lambda x: lambda y: lambda z: x * y * z
>>> wow(3)(7)(5)
105
>>> (lambda x: x(3,4))(lambda a,b: a ** b)
81
```

### Question 3 (10 minutes)

Draw the environment diagram for the following code and predict what Python will output.

```
a = lambda x: x * 2 + 1
def b(b, x):
    return b(x + a(x))
x = 3
x = b(a, x)
```

# Higher Order Functions 3





#### **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
  - o 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)
```

#### **HOF Function as Output Example**

PythonTutor

```
>>> def first(x):
        def square(y):
            def mod(z):
                return (x ** y) % z
            return mod
        return square
>>> a = first(2)
>>> b = a(4)
>>> b(3)
```

#### HOF Function as Output Example

```
>>> def first(x):
        def square(y):
            def mod(z):
                return (x ** y) % z
            return h
        return g
>>> a = first(2)
>>> b = a(4)
>>> b(3)
```

### Question 4 (10 minutes)

Draw the environment diagram for the following code and predict what Python will output.

```
n = 9
def make_adder(n):
    return lambda k: k + n
add_ten = make_adder(n+1)
result = add_ten(n)
```

### Question 5 (10 min)

Write a function that takes in a number n and returns a function that can take in a single parameter cond. When we pass in some condition function cond into this returned function, it will print out numbers from 1 to n where calling cond on that number returns True.

```
def make_keeper(n):
    """Returns a function which takes one parameter cond and prints
    out all integers 1..i..n where calling cond(i) returns True.
    >>> def is_even(x):
            # Even numbers have remainder 0 when divided by 2.
            return x \% 2 == 0
    >>> make_keeper(5)(is_even)
    11 11 11
```



# Thank you!

Anon Feedback -> https://tinyurl.com/adit-anon