TA Review Session

Stephen Martinis, Keegan Mann, Albert Wu, Julia Oh

Overview

- Evaluation
- Higher Order Functions
- Newton's Method
- Environment Diagrams
- Questions

Evaluation

Process of taking an expression and executing the commands it details according to how Python is defined as a language.

How you do anything in a program.

Here is a review of how to evaluate things.

- Not exhaustive
- Doesn't deal with environment diagrams

Evaluation Rules: Names

To find a **name** (variable)

1. Look in **local** frame first
2. Look up the name in the **parent** frame *recursively.*
3. If there is **no parent** frame, raise an Error

Evaluation Rules: Assignment

To **assign** a value to a variable

```
x = add(2, 3)
```

1. Evaluate the **right** side of = sign.
2. Bind name on **left** side of = sign to value from step 1

Evaluation Rules: Functions

```
foo(7 * 2, lambda y: x(y), add(3, mul(8,4)))
```

1. Evaluate the **operator**
2. Evaluate the **operands**
3. **Apply** the operator to the operands

Evaluation Rules: Definitions

To **define** function:

```
def foo():
    return 3
```

1. **Create** a new function
2. Label its **parent** as the **current frame**
3. **Bind** the function to its name in the current frame

Evaluation (questions)

```
>>> def foo(x):
        def bar(x):
            return x + y
        y = 6
        return bar

>>> y = 5
>>> foo(3)(5)
```

Evaluation (questions)

```
>>> def doctor(who):
        if knocks == 4:
            return 11
        return 10

>>> def ood(angry):
        if angry > 0:
            return red
        return blue
```

```
>>> knocks = 4
>>> red = 2
>>> blue = 5
>>> who = 0
```

Evaluation (questions)

	Eval to	**Outputs**
`doctor(15)`		
`doctor(36) == 10`		
`print(print(red + blue))`		
`ood(doctor(42))`		
`knocks -= ood(who)`		
`print(doctor(who))`		
`ood(knocks)`		

Functions

Syntax

```
def <name>(<arg>, <arg>, ...):
    <body>
```

Ex:

```
def foo(bar, baz):
    return 3
```

Functions (questions)

Q1

Define a function called `denero` which takes two numbers and finds their difference.

Q2

Define a function `harvey` which takes no arguments and returns the number 61.

Higher-Order Functions

```
def double(f):
    def h(x):
        return f(f(x))
    return h
```

What are the **inputs** and **outputs**?

- numbers
- booleans
- strings
- functions

Higher-Order Functions (questions)

```
def test(x):
    def review(f):
        return f(x)
    return review

def half(x):
    print(x // 2)
```

	Eval to	Outputs
half(3)	None	1
test(5)		
test(5)(half)		
test(5)(test)		

Higher-Order Functions (questions)

```
def silly(name):
    def fun():
        print('first')
        return name(y)
    print('second')
    return fun

def bop(y):
    return y // 10 + 1

y = 34
reserve = silly(bop)
```

	Eval to	Outputs
reserve()		
silly(bop(y))		
silly(print)()		
bop(reserve())		
silly(reserve)()		

Higher-Order Functions (questions)

```
def one(x):
    def two():
    def two(y):
        def three(z):
            return x + y + z
            return x(y) + z
        return three(y)
        return three(z)
        return three
    y = 4
    return two(x)
    return two
```

Cross out lines on the **right** so that the **doctests** below pass.

```
>>> one(lambda x: x*x)
<function ...two at ...>
>>> one(lambda x: x*x)()
20
>>> one(lambda x: x*x)()(4)
TypeError ...
```

Lambda Expressions

To evaluate a **lambda**

```
lambda x: 2 * x
```

- Create a new **function**
- Label its **parent** as the **current frame**
- Do NOT evaluate the expression in the colon yet

Lambda Expressions

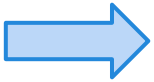
```
def <name>(<arg>, ...):
    return <expr>
```

```
<name> = lambda <arg>: <expr>
```

```
def hi():
    return 1
```

```
hi = lambda : 1
```

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



```
square = lambda x: x * x
```

```
def mul(a, b):
    return a * b
```

```
mul = lambda a, b: a * b
```

What are the **intrinsic names** of lambdas?

Lambda Expressions (questions)

Convert into lambda expressions.

Q1

```
def one(x):
    def two(y):
        return x + y
    return two
```

Q2

```
def branch(cond):
    if cond:
        return one(1)
    return one(0)
```

Lambda Expressions (questions)

```
mul = lambda a, b: a * b
curry = lambda f: lambda x: lambda y: f(x, y)
new_mul = curry(mul)(3)
```

	Eval to	Outputs
curry(mul)		
new_mul(4)		
curry(curry)(5)		
curry(curry)(1)(2)		
(lambda x, y: mul(y, x))(3, 2)		

Newton's Method (questions)

Wikipedia article

Write a function that returns the x-value at which a local maximum or minimum occurs.

You may use `approx_deriv()`, *and* `newtons_method()`

```
def find_max(fn, guess):
    return ______
```

Newton's Method (questions)

```
def newtons_method(fn, guess=1):
    ALLOWED_ERROR_MARGIN = 0.0000001
    while abs(fn(guess)) > ALLOWED_ERROR_MARGIN:
        guess -= fn(guess) / deriv(fn, guess)
    return guess
```

True or **False**?

- `newtons_method` will always terminate.
- `fn` is called 3 times in a single while loop.
- removing `abs` could break the function

Environment Diagrams

Show how Python executes code

Rules: http://www-inst.eecs.berkeley.edu/~cs61a/sp13/pdfs/environment-diagrams.pdf

Online Python Tutor:
http://inst.eecs.berkeley.edu/~cs61a-py/OnlinePythonTutor/v3/tutor.html

Environment Diagrams (rules)

Assignment

1. Evaluate **right**-hand expression
2. Write **name** in current frame
3. **Bind** expression to name

Function call

1. Draw **frame** (label p frame, intrinsic name)
2. Bind formal **parameters**
3. Evaluate function **body**

Environment Diagrams (rules)

Lookup

1. Check for name in **current frame**
2. If not found, check in **parent frame**
3. If **no parent**, error

Environment Diagrams (rules)

DO **NOT**:

- draw a new frame when **defining** functions
- draw frames for **built-in** functions
- point variables to **other variables**
- forget to label the frame's **parent**
- forget to label the frame's **intrinsic name**
- give **Global** a return value

Environment Diagrams (questions)

```
def mul(a, b):
    if b == 0:
        return a
    return a * mul(a, b - 1)
mul(3, 2)
```

Environment Diagrams (questions)

```
def dream1(f):
    kick = lambda x: mind()
    def dream2(secret):
        mind = f(secret)
        kick(2)
    return dream2

inception = lambda secret: lambda: secret
real = dream1(inception)(42)
```

Environment Diagrams (questions)

```
the = 4
def boom(goes):
    def dynamite():
        return boom(goes-1)
    if goes < the:
        return 9
    goes += 4
    return dynamite
the = boom(5)()
boom(10)
```

Conclusion

- Good luck!
- Don't panic, and you'll be fine.
- Questions?