CS 61A HKN Review Session

Nathan Zhang, Caleb Wyllie

Disclaimer

CS61A staff may/may not have been involved in the creation of these slides, or material from which these slides have borrowed from. This is in no way representative of material which will be on the midterm. HKN is not affiliated with the course, and we will attempt to correct errors but are not responsible for the loss of points resulting from errors.

Questions?

We will not be stopping for questions today, due to the number of people here. However, we do have a few wonderful HKN volunteers who will be manning the pinned post on Piazza.

Today’s Agenda

1. What does Python DO? (+ environment diagrams)
2. Lambdas
3. Higher order functions
4. Recursion
5. Boolean operations

What Would Python Do?

Python Function Semantics

1. Evaluate Operator
2. Evaluate Operands (left to right)
3. Create new frame (note: parent of frame is the frame where the function was defined)
4. Bind operands to variable names for arguments
5. Execute function (R.V. is None if function reaches end of body without other return)

Python Environment Semantix

Variable Lookup: (for now)

1. If a variable is available in the current frame, use it.
2. Go up the frame-parent chain until you find it.
3. Throw an error.

Variable Definition: (for now)

1. Add variable to current frame

“Simple” Example

```
def f(a, b, c):
    def g(a, c):
        def h(b):
            return a - b + c
        return h
    return g

f(3, 2, 1)(5, 7)(9)
```

link

Another “Simple” Example

```
def f(x):
    def g():
        return x
    if x % 2 == 0:
        x -= 3
    else:
        x -= 1
    return g
print(f(32)())
```

link

Not really “Simple” Example

```
def f(x):
    k = lambda m : x
    while x != 0:
        def g(s):
            return s + x
        if x % 3 == 1:
            k = g
        x -= 1
    return g
print(f(3)(8))
```

link

The Lambdas

Lambdas are Syntactic Sugar for Fun

```
lambda <args> : <expr>
```

```
def _____(<args>):
    return <expr>
```

```
Note: These are
unnamed functions.
```

Higher Order Functions (+ envs)

The Joker, the Slider, and the Killer

```
J = lambda x: x

S = lambda x: lambda y : lambda z:
x(z)(y(z))

K = lambda x: lambda y : x
```

Joker: the identity function

Slider: slides y and z across x

Killer: kills the second input

Sliding the Killer into a Joker

```
J = lambda x: x

S = lambda x: lambda y : lambda z:
x(z)(y(z))

K = lambda x: lambda y : x
```

Create a combination of the Slider and Killer to recreate the Joker

Sliding the Killer into a Joker

```
J = lambda x: x

S = lambda x: lambda y : lambda z:
x(z)(y(z))

K = lambda x: lambda y : x
```

Create a combination of the Slider and Killer to recreate the Joker

J = S(K)(K)

Sliding the Killer

```
J = lambda x: x

S = lambda x: lambda y : lambda z:
x(z)(y(z))

K = lambda x: lambda y : x
```

What does S(K)(S) do?

Sliding the Killer

```
J = lambda x: x

S = lambda x: lambda y : lambda z:
x(z)(y(z))

K = lambda x: lambda y : x
```

What does S(K)(S) do?

J

More HoFs

```
J = lambda x: x

S = lambda x: lambda y : lambda z:
x(z)(y(z))

K = lambda x: lambda y : x
```

Given functions f(x) and g(x), what does S(K(f))(g) do?

More HoFs

```
J = lambda x: x

S = lambda x: lambda y : lambda z:
x(z)(y(z))

K = lambda x: lambda y : x
```

Given functions f(x) and g(x), what does S(K(f))(g) do?

Gives a function f(g(x))

Reduce

```
def reduce_range(f, low, high, start_value):
    total = start_value
    while low != high:
        total = f(total, low)
        low += 1
    return total
def add(x, y):
    return x + y
print(reduce(add, 0, 10, 0))
```

link

Recursion

What is?

- Defining a function in terms of itself
- Mutual Recursion = defining a set of functions in terms of themselves

How To?

1. Given an instance of the problem, how would you break it into smaller problems?
2. What are your base cases?

Fancy Factorial

```
def _factorial(x, y):
    if x == 0:
        return y
    return _factorial(x - 1, x * y)

def factorial(x):
    return _factorial(x, 1)

print(factorial(5))
```

link

Create your own recursion

Create a function that performs floor division

Create a function that computes gcd(x, y)

Create your own recursion

Create a function that performs floor division

```
def div(x, y):
    if x < y:
        return 0
    return 1 + div(x - y, y)
```

Create a function that computes gcd(x, y)

```
def gcd(x, y):
    if x > y:
        return gcd(y, x)
    if x == 0:
        return y
    return gcd(x, y - x)
```

Booleans

And and Or

<expr1> and <expr2>

<expr1> is evaluated. If <expr1> falsey, then give <expr1>. Otherwise, evaluate <expr2> and give <expr2>

<expr1> or <expr2>

<expr1> is evaluated. If <expr1> is truthy, then give <expr1>. Otherwise evaluate and give <expr2>.

And and Or

```
def a():
    print("A, True")
    return True

def b():
    print("B, False")
    return False
```

```
a() and a()

a() and b()

b() and a()

b() and b()
```

And and Or

```
def a():
    print("A, True")
    return True

def b():
    print("B, False")
    return False
```

```
a() and a()
A, True
A, True
a() and b()
A, True
B, False
b() and a()
B, False
b() and b()
B, False
```

And and Or

```
def a():
    print("A, True")
    return True

def b():
    print("B, False")
    return False
```

```
a() or a()

a() or b()

b() or a()

b() or b()
```

And and Or

```
def a():
    print("A, True")
    return True

def b():
    print("B, False")
    return False
```

```
a() or a()
A, True
a() or b()
A, True
b() or a()
B, False
A, True
b() or b()
B, False
B, False
```

Feedback?

We are more than happy to take feedback so we can improve the review experience for everyone!

Please email us at tutoring@hkn.eecs.berkeley.edu with any feedback you would like to give.

Good luck, and may the grace of Papa John forever be upon you.