HKN Review Sessions

CS 61A Midterm 1 Fall 2012

Lisa Yan and Allen Xiao

Disclaimer, etc.

- Drop-in Tutoring: weekdays 11AM-5PM, 345 Soda and 290 Cory
- Exam archive
- Course survey results
- hkn.eecs.berkeley.edu

Disclaimer:

This review session is based off of what we tutors believe to be relevant material. Always refer to the lectures/class material for the final say if there are discrepancies. Thanks!

Agenda

- Boolean operation
- Recursive List Structures
- Environment Diagrams
- Higher-order Functions
- Data Abstraction

- Your Questions!

Python - Boolean Evaluation

```
def quote(x, y):
   if (not x and y) or (x and not y):
      print('Screw the Rules')
   else:
      print('I have money')
Kaiba = lambda a: quote(a, True)
Kaiba(False)
Kaiba(print(True))
Mokuba = lambda : False
Kaiba(Mokuba)
```

Recursive List Structures

```
# rlist, empty_rlist
# first(rlist), rest(rlist)
def print_list(ls):
    while(____________________________):
        print(_________)
        _____________
```

Fill in the blanks so that print_list prints out each element of the list on a separate line.

```
>>> friends = ('Yugi', ('Joey', ('Tristan', ('Téa', None))))
>>> print_list(friends)
Yugi
Joey
Tristan
Téa
```

Recursive List Structures

```
# rlist, empty_rlist
# first, rest
def print_list(ls):
    while(ls is not empty_rlist): # is not is also !=
        print(first(ls))
        ls = rest(ls)
```

Fill in the blanks so that print_list prints out each element of the list on a separate line.

```
>>> friends = ('Yugi', ('Joey', ('Tristan', ('Téa', None))))
>>> print_list(friends)
Yugi
Joey
Tristan
Téa
```

Environment Diagrams

```
def first(s):
    return s[0]
def rest(s):
    return s[1]
empty_rlist = None
def print_list(s):
    while(s != empty_rlist):
        print(first(s))
        s = rest(s)

>>> egyptian_god_cards = ('Slifer', ('Obelisk', ('Winged Dragon of Ra', 
None)))
>>> print_list(egyptian_god_cards)
Slifer
Obelisk
Winged Dragon of Ra
```

Data Abstraction

Duke Devlin is developing the Python logic for his upcoming dice game, Dungeon Dice Monsters. However, he has some difficult math involving complex numbers, which his version of Python does not support. Duke decides to implement his own version, and so far he has these functions:

```
def complex(a, b): # constructs c = a+bi
    return (a, b)
def real_part(c): # selects a
    return c[0]
def imag_part(c): # selects b
    return c[1]
```

Help expand Duke's implementation by writing these operations:

`conjugate(c):`

$$\text{conjugate}(a + bi) = (a + bi)^* = a - bi$$

`multiply(c1, c2):`

$$(a_1 + ib_1)(a_2 + ib_2) = (a_1 a_2 - b_1 b_2) + i(a_1 b_2 + a_2 b_1)$$

`norm_squared(c):`

$$\text{norm\_squared}(a + bi) = ||a + bi||^2 = (a + bi)(a - bi)$$

Data Abstraction

```
def conjugate(c):
    return complex(real_part(c), - imag_part(c))
def multiply(c1, c2):
    a1 = real_part(c1)
    a2 = real_part(c2)
    b1 = imag_part(c1)
    b2 = imag_part(c2)
    return complex(a1*a2 - b1*b2, a1*b2 + a2*b1)
def norm_squared(c):
    return multiply(c, conjugate(c))
```

Environment Diagrams GX

Draw the environment diagram for the following code. What does the code print?

```python
def ch_maker(ch):
    def chs(n):
        while n > 0:
            print(ch)
            n -= 1
    return lambda : chs(9)
def its_time_to(slogan):
    def foo():
        slogan()
        print('duel!')
    return foo

its_time_to(ch_maker('d'))()
```

Higher Order Functions

Let f be a function that returns the stock price of Industrial Illusions at a given day n. LPF as defined below returns the average of day n and the day right before it, to help predict the trends of duelists around the world.

```
def lpf(f):
    return lambda n: (1/2) * (f(n) + f(n-1))
```

Kaiba Corp would like to do a little better than this. How would you define the general case, where we want an lpf of the last N values? In other words, Kaiba Corp wants to average the stock price of day n, day n-1, etc. all the way to day n-(N-1) (in total, N days).

```
def general_lpf(f,N):
    def moving_average_total_N(n):
        k = 0
        sum_so_far = 0
        while _____:
            ______________________________
        return ______________
    return lambda n: moving_average_total_N(n)
```

Higher Order Functions

Let `f` be a function that returns the stock price of Industrial Illusions at a given day `n`. LPF as defined below returns the average of day n and the day right before it, to help predict the trends of duelists around the world.

```
def lpf(f):
    return lambda n: (1/2) * (f(n) + f(n-1))
```

Kaiba Corp would like to do a little better than this. How would you define the general case, where we want an `lpf` of the last `N` values? In other words, Kaiba Corp wants to average the stock price of day `n`, day `n-1`, etc. all the way to day `n-(N-1)` (in total, `N` days).

```
def general_lpf(f,N):
    def moving_average_total_N(n):
        k = 0
        sum_so_far = 0
        while k < N:
            sum_so_far = sum_so_far + f(n-k)
            k = k + 1
        return sum_so_far / N
    return lambda n: moving_average_total_N(n)
```

Higher Order Functions

How could we define the original (two point) `lpf` in terms of our new `general_lpf`?

Higher Order Functions

How could we define the original (two point) `lpf` in terms of our new `general_lpf`?

```
def lpf(f):
    lambda f: general_lpf(f, 2)
```

Questions?

Lambda functions
Iterative improvement and Newton's method
Data Abstraction
Dispatcher function (data as functions)
decorators

Unused slides

Higher Order Functions

$$\sin(f(x)) \approx \sum_{k=1}^{n} (-1)^n \frac{f(x)^{(2k+1)}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + \frac{x^{(2n+1)}}{(2n+1)!}$$

Implement the taylor series expansion for sin(f(x)):

```
def sine_generic(f, n):
    k, ans = 1, 0
    while _____:
        term = _______________________________________
        ans += ____
```

```
Note: assume you have fact(n) = n!, and pow defined as follows:
pow(x, n) = x^n
pow(2, 3) = 2^3 = 8
```

Higher Order Functions

$$\sin(f(x)) \approx \sum_{k=1}^{n}(-1)^n \frac{f(x)^{(2k+1)}}{(2k+1)!} = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots + \frac{x^{(2n+1)}}{(2n+1)!}$$

Implement the taylor series expansion for sin(f(x)):

```
def sine_generic(f, n):
    k, ans = 1, 0
    while k < n:
        term = pow(-1, k) * pow(f(x), 2k+1)/fact(2k+1)
        ans += term
```

```
Note: assume you have fact(n) = n!, and pow defined as follows:
pow(x, n) = x^n
pow(2, 3) = 2^3 = 8
```