CS 61A Midterm 1 Notes

Jeffrey Shen
3034555577

WWPD

- 0, 1 are inherantly True or False

  Ex) 3==4 >>> False
  5==True >>> False
  1==True >>> True
  0==False >>> True

- Don't forget to print outside too after Evaluating Inside

  Ex) print(print(print(2)), print(3))

  2
  None
  3
  > None None

- Make environment diagrams if needed

Environment Diagrams

- When you define a lambda, remember where the parent is, especially when it is a return or parameter, it may be Global or Outside

- When calling a function
  1) Label Frame Ex) f1
  2) Intrinsic Name Ex) square
  3) Parent Ex) [p=f1]
  4) Make parameter the first variable right away

- Don't forget RV and where you are

- Don't forget to write func before λ and def

Writing Programs

Steps

1) Read the description
2) Verify the examples & pick a simple one
3) Read the template
4) Implement w/o template or use template
5) Annotate names w/ values from chosen example
6) Write code to compute result
=> 7) Did you really return the right thing?
8.) Check your solution w/ other examples

- Use 'not' instead of ! unless !=

Hard One

- Break down the prompt and take your time
- Try to figure out where the functions and lambdas are, and where/what to return
- Just get the placement first, don't worry about how to get it to work
- Test the base case w/ program and try to get that to work

- Some have return the HOF with new parameters

CS61A Midterm 2 Notes

Jeffrey Shen
3034555577

General Info

Lists

New List	Mutates
list(lst)	lst += [7]
lst2 = lst + [7]	lst.extend([7])
list[:]	lst.append(7)
	lst.insert(i, 7)
	lst.remove(7)
	lst.pop()

Returns None

Removes first elem with value

Default removes and returns last elem

- Slice assignment

```
s[0:0] = [2,3]
```

will shift over if len(value) > slice

Str/Repr

- If there's no str, uses repr

str(...)	=>	__str__	w/ quotes
print(...)	=>	__str__	no quotes
repr(...)	=>	__repr__	w/ quotes
object	=>	__repr__	no quotes

Methods

- any(iterable) returns True if any are True
- all(iterable) returns True if all elements are True

Dictionary

- keys()
- values()
- get(key, value) value is returned if not found

Tree
- is_leaf()

Dot expression <expression>.<name>

1) Evaluate left (object)
2) Search for <name> in ① instance vars ② class vars
   or if it points to method return bound method

WWPD / Environment Diagrams

- Make sure you know what frame vars and methods are called
- Look closely for errors
- Watch for quotes
- Need self, object, or class in front of var or it calls global var or errors
- When making new list, objects will still point to same place

Coding

- Use if statement in list comprehension if want to return same thing in list
- know what type returns need to be
- empty lst possibilites
  - nonlocal <variable>
  - for loop for gererators in recursive generator tree
- %2, //2 for binary digits

* Work through how you would solve w/o template
* Work through example seeing what each part returns
* Get base case w/o recursion

CS 61A Final Notes

Jeffrey Sun
3034555577

Scheme

- boolean (everything is #t except #f)
- macros do not eval input

Streams

- car gets car of stream
- 'cdr gets rest of stream
- cdr-stream evals the next elem of stream

- Scheme eval on each expression
- Scheme apply applys operator on operands

SQL

SELECT [cols] FROM [tables] WHERE [cond] ORDER BY [attr] LIMIT [num]

SELECT [cols] FROM [tables] GROUP BY [expression] HAVING [expression];

Aggregate

functions: min, max, count, sum

weight/legs
p1, p2

count(*) > 1
min(dist) = 5

- "Group By X" to partition rows into groups and apply the aggregation func on each group
- "Having" selects only a subset of groups
- Only use Group By and Having if aggregating at least one col

Python	Scheme	SQL
== / is	eq?	=
!= / is not	(not (eq? ...))	!= / <>

Macros (Scheme)

- Think about what return needs to be, then put in list form

Ex) (list 'map (list 'lambda (list formal) body) iterable)

define-macro (for formal iterable body)

Iterator / Generators

- "yield from" gets all values
- next() traverses iterator
- iter(iterable) creates iterator

Linked Lists

Can be
- destructive - modify original Linked List
- non-destructive - creates new list

- Vars can be "pointers" to Linked Lists and refer to same
- Make sure var = var.rest for traversing in loop
- Don't need last set of parenthases for pairing pairs

Tips

- Pay attention to extend, it could add multiple elems to a list or refer to list
- Don't forget self. when doing OOP
- yu(skate(yu)) when you eval inside, it knows outside func even if assignment changes it
- Add(arg1, arg2), sum(iterable)

* Reread question, check arguments closely