

CS 61A Midterm 1 Notes

WWPD

- 0, 1 are inherently True or False

Ex) $3 == 4 \gg \text{False}$
 $5 == \text{True} \gg \text{False}$
 $1 == \text{True} \gg \text{True}$
 $0 == \text{False} \gg \text{True}$

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

Ex) $\text{print}(\text{print}(\text{print}(2)), \text{print}(3))$

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) f_1
- 2) Intrinsic Name Ex) square
- 3) Parent Ex) $[P=f_1]$

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

General Info

Lists

New List	Mutates	
list(list)	list += [T]	Returns None
list2 = list + [T]	list.extend([T])	
list[:]	list.append(T)	
	list.insert(i, T)	
	list.remove(T)	
	list.pop()	
		Removes first elem with value Default removes and returns last elem

- Slice assignment

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

will shift over if $\text{len}(\text{value}) > \text{slice}$

Str/Repr

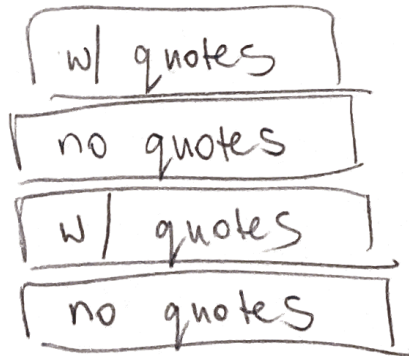
- If there's no str, uses repr

`str(...)` => `-- str --`

`print(...)` => `-- str --`

`repr(...)` => `-- repr --`

`object` => `-- repr --`



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

NWPD / 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 list possibilities
 - nonlocal <variable>
 - for loop for generators 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

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 applies 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

== / is
 != / is not

Scheme

eq?
 (not (eq? ...))

SQL

=
 != / <>

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 parentheses for printing 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