INSTRUCTIONS

- You have 2 hours to complete the exam individually.

- The exam is closed book, closed notes, closed computer, and closed calculator, except for one hand-written 8.5" × 11" crib sheet of your own creation.

- Mark your answers on the exam itself. We will *not* grade answers written on scratch paper.

Last (Family) Name	Euan
First (Given) Name	Ng
Student ID Number	
Berkeley Email	
Teaching Assistant	# Alex Stennet Alex Wang # Christina Zhang Derek Wan # Jennifer Tsui Jenny Wang # Cameron Malloy # # Erica Kong Chae Park # Griffin Prechter # Kevin Li # # Jemin Desai # Nancy Shaw # Chris Allsman # #
Exam Room and Seat	
Name of the person to your left	
Name of the person to your right	

All the work on this exam is my own. (please sign)	

POLICIES & CLARIFICATIONS

- You may use built-in Python functions that do not require import, such as min, max, pow, and abs.

- For fill-in-the blank coding problems, we will only grade work written in the provided blanks. You may only write one Python statement per blank line, and it must be indented to the level that the blank is indented.

- Unless otherwise specified, you are allowed to reference functions defined in previous parts of the same question.

1. (11 points)   No Capes!

For each of the expressions in the table below, write the output displayed by the interactive Python interpreter when the expression is evaluated. The output may have multiple lines. Each expression has at least one line of output.

- If an error occurs, write Error, but include all output displayed before the error.

- To display a function value, write Function.

- If an expression would take forever to evaluate, write Forever.

The interactive interpreter displays the value of a successfully evaluated expression, unless it is None.

Assume that you have started python3 (not ipython or other variants) and executed the code shown on the left first, then you evaluate each expression on the right in the order shown. Expressions evaluated by the interpreter have a cumulative effect.

```
1  mr, incredible = 13, 21
2
3          def el(ast, i, girl):
4          if ast > i or i / girl:
5          print('stretch')
6          return incredible
7
8  zen = 7
9
10         def fro(zone):
11         def where(is_my):
12         print('supersuit')
13         return is_my + 2
14         print(zone + zen)
15         return where
16
17 dash = 4
18
19    def edna(mo, de=4):
20    jack = 5    21    if de // 3 < 2:
22             return zen
23             return mo(jack) + dash
24
25 jack = lambda jack: edna(jack)
26
27         def vi(o, let):
28         if let % 3:
29         return o(3)
30         print('hidden')
31         return vi(print, let + 2)
```

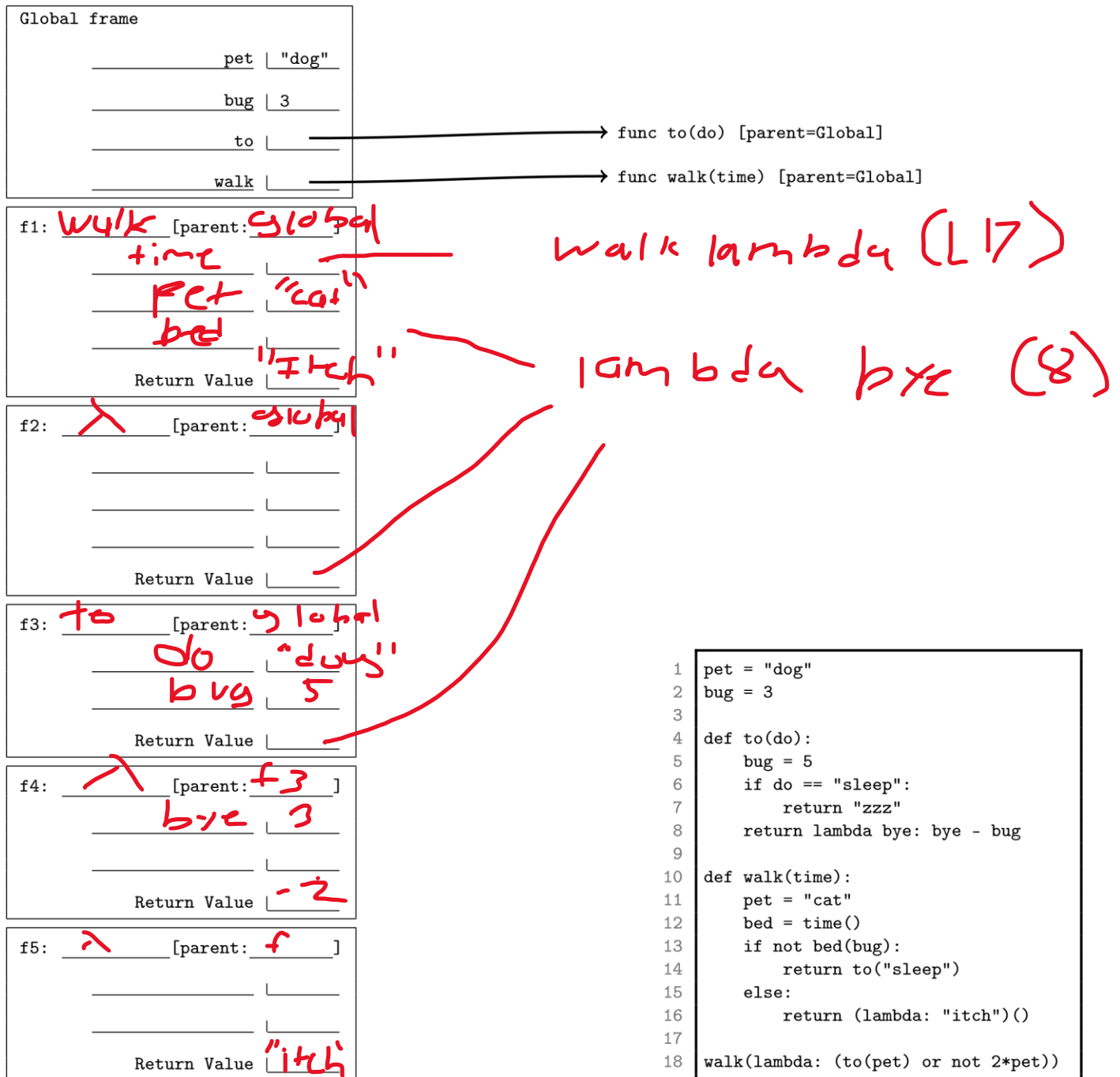| Expression	Interactive Output
print(4, 5) + 1	4 5 Error
incredible / dash	5.25
el(mr, 4, 0)	Stretch 21
sam = fro(zen)	14
edna(sam, 7)	Supersuit 11
vi(dash, incredible)	Hidden 3
vi(jack, dash)	7

2. (10 points)   Dog Days

Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.*

A complete answer will:

- Add all missing names and parent annotations to all local frames.
- Add all missing values created or referenced during execution.
- Show the return value for each local frame.

```
Global frame

                    pet | "dog"

                    bug | 3

                     to |          ──────────────→  func to(do) [parent=Global]

                    walk |         ──────────────→  func walk(time) [parent=Global]
```

f1: **walk** [parent: **global**]
 time
 pet **"cat"**
 bed **"Itch"**
 Return Value

walk lambda (L17)

f2: **λ** [parent: **global**]

 Return Value

lambda bye (8)

f3: **to** [parent: **global**]
 do **"dog"**
 bug **5**
 Return Value

f4: **λ** [parent: **f3**]
 bye **3**

 Return Value **-2**

f5: **λ** [parent: **f**]

 Return Value **"itch"**

```
1   pet = "dog"
2   bug = 3
3
4   def to(do):
5       bug = 5
6       if do == "sleep":
7           return "zzz"
8       return lambda bye: bye - bug
9
10  def walk(time):
11      pet = "cat"
12      bed = time()
13      if not bed(bug):
14          return to("sleep")
15      else:
16          return (lambda: "itch")()
17
18  walk(lambda: (to(pet) or not 2*pet))
```
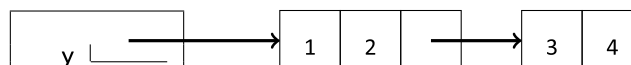
3. (7 points)   List-Man and the Box

Draw box-and-pointer diagrams for the state of the lists after executing each block of code below. You don't need to write index numbers or the word "list". Please erase or cross out any boxes or pointers that are not part of a final diagram.

An example is given below:

1 y = [1, 2, [3, 4]]

```
y ──────────→  | 1 | 2 | ──→ | 3 | 4 |
```

(a) (2 pt)

```
1   a = [1, 2, 3]
2   b = [4, 5, 6]
3   a.insert(a[0], b) 4 b.pop(a.pop(1))
```

a | 1 | 2 | 3
b | 4 | 5 | 6

(b) (2 pt)

```
1   c = [1, 2, 3]
2   d = [4, c, 6, 7]
3   e = [d[x] for x in c]
4   d.append(e)
```

c
d
e

1 2 3
4
6 7
6 7

(c) (3 pt)

```
1            f, h = [1], [2]
2            f.extend([h, 3])
3            g = f[1:] + f[1:]
4            i = 0
5            while i < len(f):
6            if i % 2 == 0:
7            h.append(g[f[i]])
8            i += 1
```

f
g
h
i

1 | | 3
1 | 3 | | 1 | 3
2 | 3 | 3

(6 points)

4.   Book Club

(a) (2 pt) A function is a higher order function if it has at least one of two particular properties. What are those two properties?

1. Return Functions

2. Take in Functions

(b) (2 pt) Give one reason why you shouldn't violate abstraction barriers. Limit your answer to 15 words or less.

Makes code harder to read

(c) (2 pt) What's a difference between tuples and lists besides syntax? Limit your answer to 10 words or less.

Lists can change tuples cant

5. (6 points)   Won't You Be My Neighbor?

(a) (4 pt) Write repeat_digits, which takes a positive integer n and returns another integer that is identical to n but with each digit repeated.

```
def repeat_digits(n):
    """Given a positive integer N, returns a number with each digit repeated.

    >>> repeat_digits(1234)
    11223344
    """
    last, rest = n%10 , n//10 If  n=0:

        return 0

    return repeat_digits(rest) * 100+last*10+last
```

(2 pt) Let d be the number of digits in n. What is the runtime of repeat_digits with respect to d?

$\sqrt{\_}$

$\Theta(1)$ **#** $\Theta(\log d)$ **#** $\Theta(\sqrt{d})$ **#** $\Theta(d)$ **#** $\Theta(d_2)$ **#** $\Theta(2_d)$

6.   Ocean's Eight

Write eight_path, which takes in a tree t and returns a list of labels following a path from the top of the tree (the root) to a leaf whose sum is divisible by 8. If there are multiple such paths, return the leftmost one. If there is no such path, return None.

The tree data abstraction is provided here for your reference. Do not violate the abstraction barrier! def tree(label, branches=[]): def is_leaf(t); return [label] + list(branches) return not branches(t)

def label(t):   def branches(t):
     return t[0]   return t[1:]

def eight_path(t):
    """Returns a path of the labels from the root to a leaf whose sum is a multiple of eight, or return None if no path exists.

    >>> t1 = tree(5, [tree(2), tree(1, [tree(3), tree(2)])
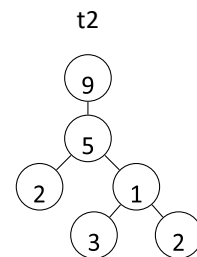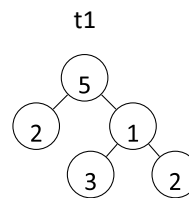   ])
    >>> eight_path(t1)
    [5, 1, 2]
    >>> t2 = tree(9, [t1])
    >>> eight_path(t2)
    [9, 5, 2]
    """ def helper(t, path_so_far):

_____ if

_____: return

     path_so_far for

_____:

       result = _____ if result:

         return _____

   return _____

7.   Never Tell Me The Odds

Han and Lando are playing a game of Sabacc. The rules are as follows:

(6 points)

- Two players take turns drawing from a deck of cards

- Each turn, players can choose to draw either 1 or 2 cards

- A player wins when there are no cards left at the start of their turn.

Han and Lando are both very good at the game, so they play *optimally*. That is, if there is a move they can take that will allow them to win (assuming their opponent also plays optimally), they will take it.

Write sabacc_winner, which takes a number of cards and two players, and returns the winner if both players play optimally and player 0 goes first.

You *must* use recursion to solve this problem. Writing the closed-form solution will receive no credit.

```
def sabacc_winner(cards, player0, player1):
    """Returns the winner of a game of Sabacc if players can take 1 or 2 cards per turn and both
    players play optimally. Assume that it is player0's turn.

    >>> sabacc_winner(0, 'Han', 'Lando')
    'Han'
    >>> sabacc_winner(1, 'Han', 'Lando')
    'Lando'
    >>> sabacc_winner(2, 'Han', 'Lando')
    'Han'
    >>> sabacc_winner(3, 'Han', 'Lando')
    'Han'
    >>> sabacc_winner(4, 'Han', 'Lando')
    'Lando' """


    If cards ==0:

        Return player0

    If cards ==1:

        Return player1

    Take_one = sabacc_winner(cards -1, player1, player0)

    Take_two = sabacc winner(cards -2, player, player0)

    If take_one == player0 or take_two player0:
```

return player0 return

player1

8. (8 points)   Mr. Stark, I Don't Feel So Good

A *messenger function* is a function that takes a single word and returns another messenger function, until a period is provided as input, in which case a sentence containing the words provided is returned. At least one word must be provided before the period.

As an example, here's a simple messenger function that returns a sentence with all of the words that have been provided.

```
>>> simple_messenger("Avengers")("assemble")(".")
'Avengers assemble.'
>>> simple_messenger("Get")("this")("man")("a")("shield")(".") 'Get this man a
shield.'
```

Write thanos_messenger, which is a messenger function that discards every other word that's provided. The first word should be included in the final sentence, the second word should be discarded, and so on.

```
def thanos_messenger(word):
    """A messenger function that discards every other word.

    >>> thanos_messenger("I")("don't")("feel")("so")("good")(".")
    'I feel good.'
    >>> thanos_messenger("Thanos")("always")("kills")("half")(".")
    'Thanos kills.'
    """
    assert word != '.', 'No words provided!' def
    make_new_messenger(message, skip_next):
        def new_messenger(word):

            if word == '.':

                return message + '.'

        If skip_next:

            Return make_new _messenger(messenger, false)

    Return make_new_messenger(message+ " " + word, true)


Return new_ messenger


Return make_new_messenger(word, True)
```

9. (0 points) Your Mission, Should You Choose To Accept It In this extra credit
problem, you may choose one of two options.

- Mark the choice to "Betray" and write a positive integer in the blank below. The one student who writes the *smallest, unique positive integer* will receive *two* (2) extra credit points but only if fewer than 90% of students choose the next option.
- Mark the choice to "Work Together". If at least 90% of students choose this option, all students who chose this option will receive *one* (1) extra credit point and those who marked the choice to "Betray" will receive zero (0) extra credit points.

Will you *work together*? Or will you *betray* your fellow students? It is up to you.

<div align="center">

Betray _____

#

Work Together

</div>

10. (0 points)   I've Been Waiting For You

If you finish early, please stay in your seat until time is up and we are ready to move on to the group section. After you've reviewed your work, if you have extra time, you can try this challenge problem while you wait (not for credit). Work on this problem quietly until the individual section is over. We will not accept questions on this problem.

(a) (0 pt) Implement lcs, which finds the length of the longest common subsequence between two words. Note that the ordering of the characters must be preserved.

```
def lcs(word1, word2):
    """ Finds the length of the longest common subsequence between two words.
    >>> lcs("water", "loo")
    0
    >>> lcs("mamma", "mia") # ma
    2
    >>> lcs("super", "trouper") # uper
    4
    """ if _____:

        return _____ elif

    _____: return

    _____ return

    _____
```

(b) (0 pt) What is the runtime (in the worst case) of lcs? By *worst case*, we mean we want you to consider inputs that would be particularly time consuming for lcs.

(c) (0 pt) Now, implement lcs_fast which does the same thing as lcs, but has a time complexity of $\Theta(m*n)$, where m is the number of characters in word1 and n is the number of characters in word2.

def lcs_fast(word1, word2):

(d) (0 pt) Provide a high-level proof as to why the runtime of your implementation of lcs_fast is $\Theta(m*n)$.

11. (0 points)   Terminal

 Done? Still need something to do?

(a) (0 pt) Can you name the movie coming out / already released this summer that each problem title is referencing?
1. ——————————————   5. ——————————————   9. ——————————————
2. ——————————————   6. ——————————————   10. ——————————————
3. ——————————————   7. ——————————————   11. ——————————————
4. ——————————————   8. ——————————————

(b)   (0 pt) Any feedback for us on how this exam went / how the course is going so far?