CS 61A Structure and Interpretation of Computer Programs
Summer 2017 Midterm

- You have 2 hours and 50 minutes to complete this exam.
- This exam is closed book, closed notes, closed computer, closed calculator, except *two* 8.5" × 11" cheat sheets.
- Mark your answers **on the exam itself**. We will *not* grade answers written on scratch paper.
- For multiple choice questions, **fill in each option or choice completely.**
  - ☐ means mark **all options** that apply
  - ◯ means mark a **single choice**

Last name		
First name		
Student ID number		
CalCentral email (`_@berkeley.edu`)		
Teaching Assistant	◯ Alex Stennet ◯ Angela Kwon ◯ Ashley Chien ◯ Joyce Luong ◯ Karthik Bharathala ◯ Kavi Gupta	◯ Kelly Chen ◯ Michael Gibbes ◯ Michelle Hwang ◯ Mitas Ray ◯ Rocky Duan ◯ Samantha Wong
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)**		

0. (0 points) SOUL What makes you happy? (Alternatively, draw something or leave us feedback.)

1. (14 points) Is this name correct?

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. If an error occurs, write "Error", but include all output displayed before the error. If a function value is displayed, write "Function".

The first two parts have been provided as examples.

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

Assume that you have started `python3` and executed the following statements:

```
amoeba = dict() # make a new dictionary like {}
clone = ' (clone)'

def make_amoeboid(name):
    if name in amoeba:
        print('I am but a clone')
        amoeboid = make_amoeboid(name + clone)
        amoeba[name].append(amoeboid)
    else:
        print('My name is ' + name)
        amoeboid = name
        amoeba[name] = [amoeboid]
    return amoeboid

def parent(name):
    while name[:-len(clone)] in amoeba:
        name = name[:-len(clone)]
    return name

def find_amoeba(f, name):
    friends = []
    for amoeboid in amoeba[name]:
        if f(amoeboid):
            print('There you are, ' + amoeboid)
            friends.append(amoeboid)
    return friends

harry = make_amoeboid('harry')
harry = make_amoeboid('harry')
```

```
>>> pow(2, 3)
8
```

```
>>> print(harry, 'hairy') + 1
harry (clone) hairy
Error
```

```
>>> make_amoeboid('flora')
```

```
>>> print(parent(harry), print(harry))
```

```
>>> find_amoeba(lambda a: True, 'harry')
```

```
>>> find_amoeba(lambda a: parent(a) == harry, 'harry')
```

```
>>> find_amoeba(lambda a: make_amoeboid('gabby'), 'flora')
```

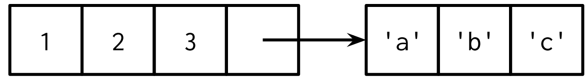
```
>>> find_amoeba(lambda a: parent(a) == 'flora',
...             make_amoeboid('flora'))
```

2. (8 points) Not very creative. . . ?

Fill in the environment diagram that results from executing each block of code below until the entire program is finished or an error occurs. Use box-and-pointer notation for lists. 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.

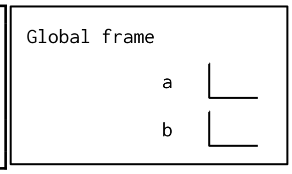
An example of the box-and-pointer representation of the list below is shown to the right.

```
[1, 2, 3, ['a', 'b', 'c']]
```



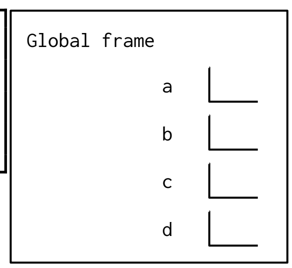
(a) (2 pt)

```
a = [1, 2, 3]
b = [1, 2, 3]
a.append(b.append(a))
b.append(a.append(b))
```



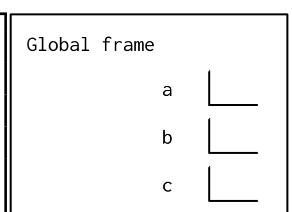
(b) (2 pt)

```
a = [1, 2, 3]
b = a
c = [list(a), [b]]
d = c[1:]
```



(c) (4 pt)

```
a = [1, 2, 3]
b = [1, 2, 3]
c = [a, [b], [[a]]]
while c:
    a.extend(c.pop())
    a, b = b, a
```



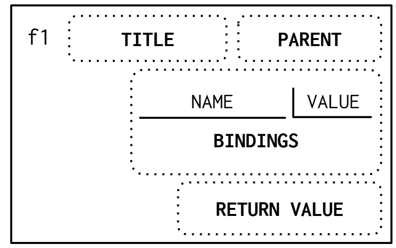
3. (10 points) Face Steak (You don't feel like it's made of real meat. . .)

(a) On the next page, 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.

(b) Then, for each FIELD below, fill in the corresponding bubble or *fig.* if referring to a drawn figure such as a list. Leave a row blank if the space in the environment diagram should be left blank.

To receive credit, you must list your bindings in the order in which they are ***first bound in the frame****.*



FRAME	FIELD	NAMES	VALUES
f1	Binding 1	default	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 2	args	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 3	wraps	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Return		○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
f2	Title	○ λ ○ r ○ w ○ wraps	
	Binding 1	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 2	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 3	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Return		○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
f3	Title	○ λ ○ r ○ w ○ wraps	
	Binding 1	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 2	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 3	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Return		○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
f4	Title	○ λ ○ r ○ w ○ wraps	
	Binding 1	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 2	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 3	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Return		○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
f5	Title	○ λ ○ r ○ w ○ wraps	
	Binding 1	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 2	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Binding 3	○ f ○ g ○ w ○ x ○ y	○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*
	Return		○ 1 ○ 2 ○ 3 ○ α ○ β ○ γ ○ δ ○ ϵ ○ *fig.*

Remember to draw figures in the designated box and fill out the choices to receive credit.

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.
- Include all *figures* or diagrams of objects (such as lists) in the **designated box**.

```
x, y = 1, 2

def r(f, g, y):
    return f(f(x, y), g(y, x))

def record(default):
    args = default[:]
    def wraps(f):
        def w(x):
            args.append(x)
            default.append(args)
            return f(x)
        return w
    return wraps

record([1, 2])(lambda y: r(min, max, y))(3)
```

Global frame

x	1
y	2
r	→ func r(f, g, y) [parent=Global]
record	→ func record(default) [parent=Global]

f1 record [parent=Global]

default	
args	
wraps	
Return Value	

f2 ____________ [parent=]

____________	
____________	
____________	
Return Value	

f3 ____________ [parent=]

____________	
____________	
____________	
Return Value	

f4 ____________ [parent=]

____________	
____________	
____________	
Return Value	

f5 ____________ [parent=]

____________	
____________	
____________	
Return Value	

All *figures* must go in above box

α func wraps(f) [parent=]

β func w(x) [parent=]

γ func lambda(y) [parent=]

δ built-in func min(...)

ϵ built-in func max(...)

4. (6 points) Instant Noodles (Comes with everything you need for a quick meal!)

For each of the functions below, choose the order of growth that best describes the execution time as a function of N, the size of the input number `n`, or "Infinite" if the function never terminates.

(a) (1.5 pt)

```
def foo(n):
    i = 1
    while i < n:
        i += 10
        n += 5
```

- ◯ $\Theta(1)$
- ◯ $\Theta(\log N)$
- ◯ $\Theta(N)$
- ◯ $\Theta(N \log N)$
- ◯ $\Theta(N^2)$
- ◯ $\Theta(N^3)$
- ◯ $\Theta(2^N)$
- ◯ $\Theta(3^N)$
- ◯ Infinite

(b) (1.5 pt)

```
def baz(n):
    i = 1
    while i < n:
        j = i
        while j < n:
            while j < n:
                j += 1
            j += 1
        i += 1
```

- ◯ $\Theta(1)$
- ◯ $\Theta(\log N)$
- ◯ $\Theta(N)$
- ◯ $\Theta(N \log N)$
- ◯ $\Theta(N^2)$
- ◯ $\Theta(N^3)$
- ◯ $\Theta(2^N)$
- ◯ $\Theta(3^N)$
- ◯ Infinite

(c) (1.5 pt)

```
def bar(n):
    i = 1
    while i < n:
        i += i
        i += i
```

- ◯ $\Theta(1)$
- ◯ $\Theta(\log N)$
- ◯ $\Theta(N)$
- ◯ $\Theta(N \log N)$
- ◯ $\Theta(N^2)$
- ◯ $\Theta(N^3)$
- ◯ $\Theta(2^N)$
- ◯ $\Theta(3^N)$
- ◯ Infinite

(d) (1.5 pt)

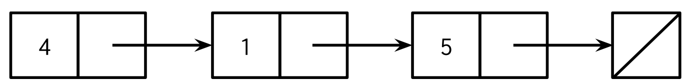
```
def garply(n):
    for i in range(n):
        for j in range(n):
            for k in range(i + j):
                return garply(n-1)
```

- ◯ $\Theta(1)$
- ◯ $\Theta(\log N)$
- ◯ $\Theta(N)$
- ◯ $\Theta(N \log N)$
- ◯ $\Theta(N^2)$
- ◯ $\Theta(N^3)$
- ◯ $\Theta(2^N)$
- ◯ $\Theta(3^N)$
- ◯ Infinite

5. (6 points) **Bisicle** (It's a two-pronged popsicle, so you can eat it twice.)

Implement `replicate_link`, which takes a non-empty linked list of integers `s` and returns a new linked list where each integer n appears n times. Negative numbers are ignored. The linked list data abstraction is below.

```
empty = ...  # The empty linked list

def link(first, rest=empty):
    """Construct a linked list from its first element and the rest."""

def first(s):
    """Return the first element of a linked list s."""

def rest(s):
    """Return the rest of the elements of a linked list s."""

def is_link(s):
    """Returns True if s is a linked list, and False otherwise."""

def is_empty(s):
    """Returns True if s is the empty linked list, and False otherwise."""

def print_link(s):
    """Print elements of a linked list s."""

def replicate_link(s):
    """Given a non-empty linked list of integers s, return a new linked list where each
    element of the linked list s appears element number of times. Negative numbers are ignored.

    >>> l = link(4, link(1, link(5)))
    >>> print_link(l)
    4 1 5
    >>> print_link(replicate_link(l))  # replicated linked list
    4 4 4 4 1 5 5 5 5 5
    >>> l = link(6, link(-1, link(-3, link(2, link(0, link(5, link(-10)))))))
    >>> print_link(l)                  # show input linked list
    6 -1 -3 2 0 5 -10
    >>> print_link(replicate_link(l))  # replicated linked list
    6 6 6 6 6 6 2 2 5 5 5 5 5
    """
```




```
    def replicate(________________________):

        if ___________________________________________________________________________:

            return ___________________________________________________________________________

        elif _________________________________________________________________________:

            return ___________________________________________________________________________

        return _______________________________________________________________________________

    return ______________________________________________________________________
```

6. (8 points) Hot Cat (Like a hot dog, but with little cat ears on the end.)

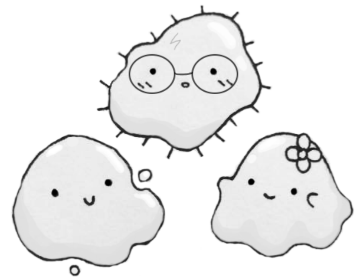
Implement `compress`, which takes a deep list of integers and returns a *new list* compressing all neighboring integers in the input list. Compression involves reducing a group of neighboring integers to a single number: the sum of the group. Values in a list are considered neighbors if their indices differ by 1.

Compressing `[1, 2, 3]` results in `[6]` since the input integers are all part of a group of neighboring integers.

```
def compress(lst):
    """Given a deep list of integers, return a new list compressing all neighboring integers.

    >>> compress([])
    []
    >>> compress([1, 2, 3])
    [6]
    >>> compress([0, 0, 0, 0])
    [0]
    >>> compress([1, 2, [3, 4]])
    [3, [7]]
    >>> compress([[11, 12], 3, 4, [1, 2], [5, 6], 7, 8, [9, 10]])
    [[23], 7, [3], [11], 15, [19]]
    >>> compress([1, 2, [3, [4, 5, 6], [7, 8], 9, 10], 11, 12])
    [3, [3, [15], [15], 19], 23]
    """

    total = 0

    result = ___________________________________

    ____________________________________________

    for element in lst:

        if type(element) == int:

            ____________________________________________________________________________________

            ____________________________________________________________________________________

        else:

            ____________________________________________________________________________________

            ____________________________________________________________________________________

            ____________________________________________________________________________________

            ____________________________________________________________________________________

            ____________________________________________________________________________________

    ____________________________________________________________________________

    ____________________________________________________________________________

    return result
```

7. (0 points) Designated Exam Fun Zone

Draw something. Leave a scent on the paper. It is up to you.



8. (10 points) Annoying Dog (A little white dog. It's fast asleep...)

(a) (2 pt) Implement a `list_counter` that returns a number in base 10 equal to the value of the `digits` in the given `base`. Numbers that are not digits in the given base are ignored. Each subsequent digit increases the value of the preceding digits by a factor of `base`.

The value of `list_counter(2, [1, 0, 1, 1])` is computed by reading the `digits` from left to right:

$$\left[\left(\left[\left([(\mathbf{1})\cdot 2] + \mathbf{0}\right)\cdot 2\right] + \mathbf{1}\right)\cdot 2\right] + \mathbf{1}$$

```
def list_counter(base, digits):
    """Return a number in base 10 equal to the value of the digits in the given base.
    Numbers that are not digits in the given base are ignored.

    >>> list_counter(2, [])
    0
    >>> list_counter(2, [1, 0, 1, 1])     # see example above
    11
    >>> list_counter(2, [1, 2, 3, 0, 1])  # 2 and 3 are not digits in base 2
    5
    >>> list_counter(4, [1, 2, 3, 0, 1])  # 1*(4**4) + 2*(4**3) + 3*(4**2) + 0*(4**1) + 1*1
    433
    """

    total = ____________________

    ________________________________________________________________________________

    ________________________________________________________________________________

    ________________________________________________________________________________

    ________________________________________________________________________________

    return total
```

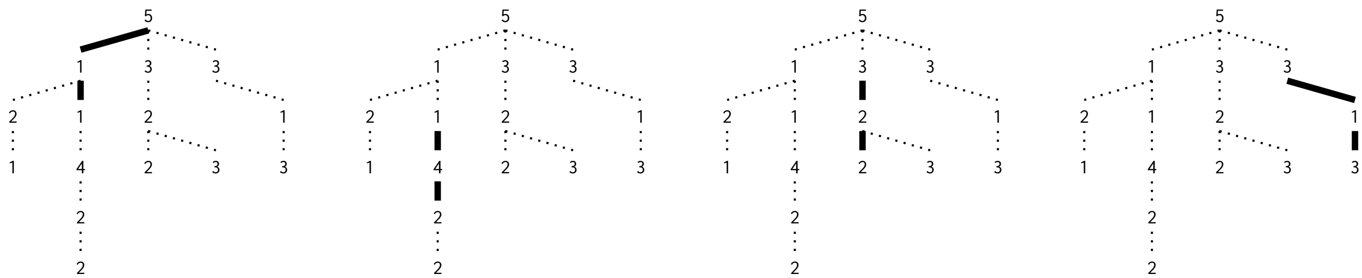
(b) **(8 pt)** Implement a `counter` that returns a function which accepts digits in a given base and returns the value in base 10 after encountering `'done'`. Numbers that are not digits in the given base are ignored.
Hint: What should `parse` return?

```
def counter(base):
    """Return a function which accepts digits in a given base and returns the value in base
    10 after encountering 'done'. Numbers that are not digits in the given base are ignored.

    >>> binary = counter(2)
    >>> binary('done')
    0
    >>> binary(1)(0)(1)(1)('done')         # see example from previous page
    11
    >>> binary(1)(2)(3)(0)(1)('done')      # 2 and 3 are not digits in base 2
    5
    >>> quaternary = counter(4)
    >>> quaternary(1)(2)(3)(0)(1)('done')  # 1*(4**4) + 2*(4**3) + 3*(4**2) + 0*(4**1) + 1*1
    433
    """

    def parse(__________________________):

        if _________________________________________________:

            return _________________________________________________________________________

        elif _______________________________________________:

            return _________________________________________________________________________

        else:

            return _________________________________________________________________________

    return _________________________________________________
```

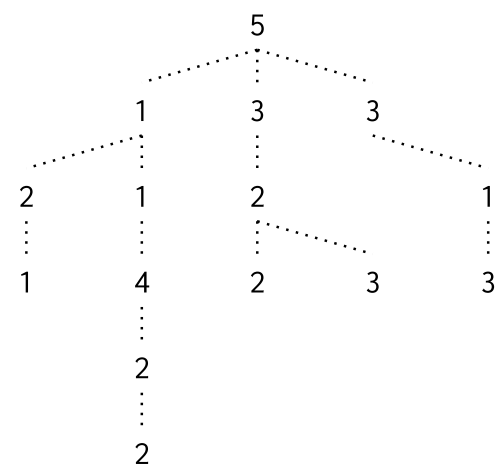
```
def tree(root, branches=[]):
    """Construct a tree with the given root value and a list of branches."""

def root(tree):
    """Return the root value of a tree."""

def branches(tree):
    """Return the list of branches of the given tree."""

def is_tree(tree):
    """Returns True if the given tree is a tree, and False otherwise."""

def is_leaf(tree):
    """Returns True if the given tree's list of branches is empty, and False otherwise."""

def print_tree(t, indent=0):
    """Print a representation of this tree in which each node is indented by two spaces times
    its depth from the root."""
```

9. (8 points) Temmie Flakes (It's just torn up pieces of construction paper.)

Implement `count_ways`, which takes a tree `t` and an integer `total` and returns the number of ways any top-to-bottom sequence of consecutive nodes can sum to `total`. Shown below with bolded edges are the four ways counted during `count_ways(t1, 7)`. The tree data abstraction is on the previous page.



```
def count_ways(t, total):
    """Return the number of ways that any sequence of consecutive nodes in a root-to-leaf path
    can sum to total.

    >>> t1 = tree(5, [tree(1, [tree(2, [tree(1)]),
    ...                        tree(1, [tree(4, [tree(2, [tree(2)])])])]),
    ...               tree(3, [tree(2, [tree(2),
    ...                                 tree(3)])]),
    ...               tree(3, [tree(1, [tree(3)])])])
    >>> count_ways(t1, 7)
    4
    >>> count_ways(t1, 4)
    6
    >>> t2 = tree(2, [tree(-10, [tree(12)]),
    ...               tree(1, [tree(1),
    ...                        tree(-1, [tree(2)])])])
    >>> count_ways(t2, 2)
    6
    >>> count_ways(t2, 4)
    3
    """

    def paths(__________________________________):

        ways = 0

        if ____________________________________:

            ____________________________________

        ways += sum(____________________________________________________________________________)

        if ____________________________________:

            ____________________________________________________________________________________

        return ways

    return _____________________________________________________________________
```



10. (0 points) `You feel a calming tranquility. You're filled with determination...`

In this extra credit problem, you may choose one of two options.

- Mark the choice to "Go alone" 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 95% of students choose the next option.
- Mark the choice to "Cooperate". If at least 95% 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 "Go alone" will receive zero (0) extra credit points.

Will you *go alone*? Or will you *cooperate*? It is up to you.

◯ Go alone ________

◯ Cooperate