INSTRUCTIONS

- You have 3 hours to complete the exam individually.
- The exam is closed book, closed notes, closed computer, and closed calculator, except for two 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.

<table>
<thead>
<tr>
<th>Last (Family) Name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>First (Given) Name</td>
<td></td>
</tr>
<tr>
<td>Student ID Number</td>
<td></td>
</tr>
<tr>
<td>Berkeley Email</td>
<td></td>
</tr>
<tr>
<td>Teaching Assistant</td>
<td></td>
</tr>
<tr>
<td>Exam Room</td>
<td></td>
</tr>
<tr>
<td>Name of the person to your left</td>
<td></td>
</tr>
<tr>
<td>Name of the person to your right</td>
<td></td>
</tr>
</tbody>
</table>

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

POLICIES & CLARIFICATIONS

- If you need to use the restroom, bring your phone and exam to the front of the room.
- You may use built-in Python functions that do not require import, such as min, max, pow, len, and abs.
- You may not use example functions defined on your study guides unless a problem clearly states you can.
- 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. (10 points) WWPD

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**.
- If nothing is displayed, write **NOTHING**

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

Assume that you have started `python3` 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.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>print(4, 5) + 1</code></td>
<td>4 5 <strong>ERROR</strong></td>
</tr>
<tr>
<td><code>(p() and p()) or p()</code></td>
<td></td>
</tr>
<tr>
<td><code>print(e(5))</code></td>
<td></td>
</tr>
<tr>
<td><code>f(x, y)</code></td>
<td></td>
</tr>
<tr>
<td><code>g(h, 4, 10)</code></td>
<td></td>
</tr>
<tr>
<td><code>g(lambda x: x % 2, y, x)</code></td>
<td></td>
</tr>
</tbody>
</table>
2. (10 points) Environment Diagram

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.

```python
def view(re):
    ice = 4
    def k(means):
        return re(means)
    return k

def review(pr):
    return re(pr * 2)

ice = 3
re = view(lambda pr: pr * ice)
review(1)
```
3. (8 points) Boxing Day

Draw box-and-pointer diagrams for the state of the lists after executing each block of code below. Each box should contain a value or a pointer to an object, as in an environment diagram.

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:

(a) (3 pt)

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

```
y ______
1 2
func max(...) 
3 4
```

```
1 lst = [2, 4, lambda: lst]
2 lst2 = lst
3 lst = lst[2:]
4 lst3 = lst[0]()
```

```
lst ______
lst2 ______
lst3 ______
lst ______
```

(b) (3 pt)

```
1 lst = [[5], 2, 4, 10]
2 lst2 = lst[1:3] + lst[:3]
3 for n in lst2[:2]:
4     lst.append(lst2[n])
```

```
lst ______
lst2 ______
```

(c) (2 pt)

```
1 lst = ['goodbye', 0, None, 8, 'hello', 1]
2 while lst.pop():
3     x = lst.pop()
4     if x:
5         lst.pop()
6     else:
7         lst.append('three')
```

```
lst ______
```
4. (2 points) Conceptual Questions

There are three problems here, which cover topics from this week’s lecture. Each problem is worth 1 point, but you can only earn a maximum of 2 points on this problem, so you only need to know two answers.

Please make sure to fill in the bubble completely when answering. Each question has only one right answer.

(a) (1 pt) Mutable Functions

Declaring a variable nonlocal allows you to modify a binding for that variable in:

- Any parent frame
- The global frame
- Any parent frame other than global
- The current frame

(b) (1 pt) Growth

Out of these options, what’s the best way to measure the performance of a program?

- Run the program many times and calculate the average time it takes to run
- Count the number of operations executed by the program for different inputs
- Count how many while and for loops the code contains

(c) (1 pt) Iterators and Generators

Select the statement that is not true. A generator function is a function that...

- is an iterator
- returns an iterator
- returns a generator
- contains the "yield" keyword

5. (4 points) Negative Feedback

Write a function clear_negatives that takes in a list containing integers, and returns a list which is identical to the original, but in each case where a negative integer appears, delete a number of elements from the list equivalent to the magnitude of the negative integer.

For example, if -4 occurs in the list, remove a total of four elements: -4 and the next three elements. See the doctests for examples.

Your solution must use recursion, and should not modify the input list.

def clear_negatives(lst):
    ""
    >>> clear_negatives([1, 0, 6])
    [1, 0, 6]
    >>> clear_negatives([3, -2, 1, 6, 2])
    [3, 6, 2]
    >>> lst = [-3, -4, 5, 5]
    >>> clear_negatives(lst)
    [5]
    >>> lst
    [-3, -4, 5, 5]
    ""

    if _________________________________________________________________:
        return []

    elif ________________________________________________________________:
        return ____________________________

    else:
        return ____________________________
Let’s play a guessing game! In order to do this, we’ll use higher order functions.

Write a function, `make_guess`, which takes in a number that we want someone to try to guess, and returns a guessing function.

A **guessing function** is a one-argument function which takes in a number. If the number passed in is the number we wanted to guess, it will return the number of incorrect guesses made prior to the correct guess. Otherwise, it returns another guessing function, which keeps track of the fact that we’ve made an incorrect guess.

Solutions which use lists, object mutation, nonlocal, or global will receive no credit.

```python
def make_guess(n):
    "":
    >>> guesser = make_guess(10)
    >>> guess1 = guesser(6)
    >>> guess2 = guess1(7)
    >>> guess3 = guess2(8)
    >>> guess3(10)
    3
    >>> guess2(10)
    2
    >>> a = make_guess(5)(1)(2)(3)(4)(5)
    >>> a
    4
    "":
    def update_guess(num_incorrect):
        def new_guess(x):
            if _________________________________________________________________:
                "":
            else:
                "":
            return ____________________________________________
```
7. (12 points) Scrabbled Eggs

In this problem, we will build (part of) the ultimate Scrabble AI! Scrabble is a game where, given a collection of letters that each have a point value assigned, you try to make the word with the greatest score.

For example, if "a" and "s" are both worth 1 point, and "b" and "c" are both worth 2 points, the word "cab" would get 5 points, and "sass" would get 4 points.

(a) (8 pt) Implement the function `is_subseq` which takes two strings, \( w_1 \) and \( w_2 \), and returns `True` if \( w_1 \) is a subsequence of \( w_2 \).

Recall from Lab 5 that a string is a subsequence of another string if all the letters in the first string appear in the second string in the same order, although not necessarily directly next to each other. Put another way, you can transform the first string to the second string just by adding letters at any position in the first string.

For example, "put" and "cot" are both subsequences of "computer", but "car" and "rot" are not.

```python
def is_subseq(w1, w2):
    """ Returns True if \( w_1 \) is a subsequence of \( w_2 \) and False otherwise. """

    >>> is_subseq("word", "word")
    True
    >>> is_subseq("compute", "computer")
    True
    >>> is_subseq("put", "computer")
    True
    >>> is_subseq("computer", "put")
    False
    >>> is_subseq("sin", "science")
    True
    >>> is_subseq("nice", "science")
    False
    is_subseq("boot", "bottle")
    False
    """

    if __________________________________________________________________________:
        return ______________________________________________________________________
    elif ________________________________________________________________________:
        return ______________________________________________________________________
    else:
        with_elem = __________________________ and _________________________________
        without_elem = ______________________________________________________________
        return ______________________________________________________________________
```
(b) (4 pt) Now, implement `scrabbler` which takes in a string `chars`, a list of strings `words`, and a dictionary `values` which maps letters to point values. It should return a dictionary where each key is a word in `words` which can be formed from the letters in `chars` and each value is the point value of that word.

For the purposes of this problem, we will only consider words we can form by using letters in `chars` in the same order they appear in the string. Assume `values` contains all the letters across valid words as keys. You may additionally assume `is_subseq` from the previous part has been implemented correctly.

```python
def scrabbler(chars, words, values):
    """ Given a list of words and point values for letters, returns a
dictionary mapping each word that can be formed from letters in chars
to their point value. You may not need all lines
    
    >>> words = ['easy', 'as', 'pie']
    >>> values = {'e': 2, 'a': 2, 's': 1, 'p': 3, 'i': 2, 'y': 4}
    >>> scrabbler('heuaisosby', words, values)
    {'easy': 9, 'as': 3}
    >>> scrabbler('piayse', words, values)
    {'pie': 7, 'as': 3}
    """

    result = ________________________________________________________________
    for _____________________________________________________________________:
        if _________________________________________________________________:
            ___________________________________________________________________
            ___________________________________________________________________
            ___________________________________________________________________
            ___________________________________________________________________
    return result
```
8. (8 points) Leaf It To Me

Write a function `max_path` which takes in a tree `t` with positive labels and a number `k`, and returns the path with length at most `k` for which the sum of the labels in the path is greatest. If there are multiple paths with the greatest sum, return the leftmost one.

The path does not have to start at the root of the tree - the path can contain any top-to-bottom sequence of nodes.

```
def tree(label, branches=[]):
    return [label] + list(branches)

def is_leaf(t):
    return not branches(t)

def label(t):
    return t[0]

def branches(t):
    return t[1:]
```

The tree data abstraction is provided here for your reference. Do not violate the abstraction barrier!

```
def max_path(t, k):
    """ Return a list of the labels on any path in tree t of length at most k with the greatest sum
    >>> t1 = tree(6, [tree(3, [tree(8)]), tree(1, [tree(9), tree(3)])])
    >>> max_path(t1, 3)
    [6, 3, 8]
    >>> max_path(t1, 2)
    [3, 8]
    >>> t2 = tree(5, [t1, tree(7)])
    >>> max_path(t2, 1)
    [9]
    >>> max_path(t2, 2)
    [5, 7]
    >>> max_path(t2, 3)
    [6, 3, 8]
    """
    def helper(t, k, on_path):
        if ________________:
            return []
        elif ________________:
            return [label(t)]
        a = ________________
        if ________________:
            return ________________(______________, key = ________________)
        else:
            b = ________________
            return ________________(______________, key = ________________)
        return helper(t, k, False)
```
9. (0 points) The Prisoner’s Dilemma
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.

  ○ Betray _______
  ○ Work Together

10. (0 points) The End

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

(b) (0 pt) Use this space to draw a picture, write a note, or otherwise express yourself.