INSTRUCTIONS

- You have 3 hours to complete the exam.
- The exam is closed book, closed notes, and closed electronics, except three 8.5” × 11” cheat sheets, and The Environment Diagram Rules.
- Mark your answers ON THE EXAM ITSELF. Answers outside of the space allotted to problems will not be graded. If you are not sure of your answer you may wish to provide a brief explanation.

<table>
<thead>
<tr>
<th>Full name</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SID</td>
<td></td>
</tr>
<tr>
<td>Login</td>
<td></td>
</tr>
<tr>
<td>TA &amp; section time</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>
<tr>
<td>All the work on this exam is my own. (please sign)</td>
<td></td>
</tr>
</tbody>
</table>

0. (2 points) Your thoughts? What’s been fun? What are you grateful for?
1. (18 points) What will Python output?

Include all lines that the interpreter would display. If it would display a function, then write Function. If it would cause an error, write Error. Assume that you have started Python 3 and executed the following. These are entered into Python exactly as written.

class Cat:
    name = "meow"
    def __init__(self, fish):
        self.fish = fish

    def __iter__(self):
        while len(self.fish) > 0:
            yield lambda: self.fish[0].getname()
            self.fish = self.fish[1:]

class SuperCat(Cat):
    def __init__(self):
        self.lives = 9

    def __next__(self):
        if len(self.fish) == 0:
            raise StopIteration
        return self.fish

class Fish:
    def __init__(self, name):
        self.name = name

    def __len__(self):
        fish = ['fish1', 'fish2', 'fish3']
        return len(self.fish) + 2

    def getname(self):
        print(self.name)
<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>print('Cats are cool!')</td>
<td>Cats are cool!</td>
</tr>
<tr>
<td>dory = Fish('Dory')</td>
<td></td>
</tr>
<tr>
<td>marlene = Fish('Marlene')</td>
<td></td>
</tr>
<tr>
<td>dory.name</td>
<td></td>
</tr>
<tr>
<td>dari = Cat([dory, marlene, Fish('Nemo')])</td>
<td></td>
</tr>
<tr>
<td>print(dari.fish[2].getname())</td>
<td></td>
</tr>
<tr>
<td>fishes = iter(dari)</td>
<td></td>
</tr>
<tr>
<td>next(fishes)</td>
<td></td>
</tr>
<tr>
<td>pusheen = SuperCat()</td>
<td></td>
</tr>
<tr>
<td>pusheen.fish</td>
<td></td>
</tr>
<tr>
<td>Cat.<strong>init</strong>(pusheen, dari.fish)</td>
<td></td>
</tr>
<tr>
<td>next(pusheen).getname()</td>
<td></td>
</tr>
<tr>
<td>for a in pusheen:</td>
<td></td>
</tr>
<tr>
<td>a()</td>
<td></td>
</tr>
<tr>
<td>dari.getname = Fish.getname</td>
<td></td>
</tr>
<tr>
<td>dari.getname(dory)</td>
<td></td>
</tr>
<tr>
<td>print(len(pusheen.fish))</td>
<td></td>
</tr>
<tr>
<td>print(Fish.<strong>len</strong>(pusheen))</td>
<td></td>
</tr>
<tr>
<td>Expression</td>
<td>Interactive Output</td>
</tr>
<tr>
<td>------------</td>
<td>-------------------</td>
</tr>
</tbody>
</table>
| lst1 = [i for i in range(1, 3)]
lst2 = list(range(4, 6))
lst1 + lst2 | |
| x = [[lst1], [lst2]][0]
x[0] is lst1 | |
| y = lambda: lambda lst1: lambda: lst1
y()(x)() is lst1 | |
| lst3 = lst1.append(lst2)
print(lst3) | |
| lst1 | |
| lst1[2] is lst2 | |
| lst1[:1] | |
| lst1[2:] is lst1[2:] | |
| lst2[1] | |
| lst1[____] + lst2[____] + \
lst1[____] + lst2[____] | [4, 2, 5] |
2. (8 points) Alakazam!

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. You may want to keep track of the stack on the left, but this is not required.

A complete answer will:

- Add all missing names, labels, and parent annotations to all local frames.
- Add all missing values created during execution.
- Show the return value for each local frame.
- The first function created by `lambda` should be labeled $\lambda_1$, the next one should be $\lambda_2$, and so on.

```python
def magic(hat):
    if hat[0] == 7:
        return hat + [7]
    return abra(hat, lambda x, y: hat.append(7))

def abra(kad, abra):
    kad[0] = 0
    abra(1, 2)
    return magic(kad[4:])

switch = [3, 1, 3, 3, 7]
magic(switch)
```

```
Stack

  global

Return Value

Global frame

Return Value

Return Value

Return Value

func magic(hat) [p=global]

Return Value

Return Value

Return Value

Return Value

Return Value

Return Value

Return Value
```
3. (4 points) Counting Song

Let's write a song with generators! The song goes like this: 1, 1 2 1, 1 2 3 2 1 ... Write `counting_song`, a function which yields each line of the song represented as a list (see the doctests).

There are multiple ways to solve this.

```python
def counting_song():
    """
    >>> cs = counting_song()
    >>> next(cs)
    [1]
    >>> next(cs)
    [1, 2, 1]
    >>> next(cs)
    [1, 2, 3, 2, 1]
    >>> next(cs)
    [1, 2, 3, 4, 3, 2, 1]
    """

    song = [1]
    while True:
        yield __________________________________________________________
        __________________________________________________________

4. (5 points) Final tale of Waldo

Write `waldo-tail`, a tail-recursive version of the `wheres-waldo` function you saw on Midterm 2. As a reminder, `waldo-tail` is a Scheme procedure that takes in a Scheme list and outputs the index of `waldo` if the symbol `waldo` exists in the list. Otherwise, it outputs the symbol `nowhere`.

```
STk> (waldo-tail '(moe larry waldo curly))
2
STk> (waldo-tail '(1 2))
nowhere

(define (waldo-tail lst)
  (define (helper ____________________________________________)
    (cond ((____________________________________________) 'nowhere); Base Case
          (_____________________________________________); Base Case
          (else ____________________________________________)))
    (helper ____________________________________________))
```
5. (6 points) Guess-Tree

Many popular guessing games have the following format: We start with a single question, such as "Is it an animal?" If the answer is "yes", we ask a new question assuming what we're guessing is an animal ("Is it a dog?"). If the answer is "no", we ask a new question assuming what we're guessing is not an animal ("Is it a place?"). We keep repeating this process until we are ready to make a guess.

One possible representation of this game is using nested dictionaries. The dictionary has one key - the original question. Its value is another dictionary with at MOST two keys: a 'yes' key and a 'no' key (either of those keys may be missing). Each of those leads to another dictionary, and so on until we hit "Success!" or run into an empty dictionary.

Another possible representation is as a BinaryTree. Your task is to write make_guess_tree, which takes in a nested dictionary representation and converts it into a BinaryTree representation:

class BinaryTree:
    def __init__(self, entry, left=None, right=None):
        self.entry = entry
        self.left = left
        self.right = right

def make_guess_tree(guess):
    """
    results = {'Is it an animal?': \n        {'yes': {'Is it a dog?': \n            {'yes': 'Success!'}}, \n        'no': {'Is it a place?': \n            {'no': {'Is it a person?': \n                {}}, \n            'yes': {'Is it LA?': \n                {}}}}}

    >>> t = make_guess_tree(results)
    >>> t.pretty_print()
        Is it an animal?
          /  
        Is it a dog?     Is it a place?
          /  
        Success!     Is it LA?     Is it a person?
    """

    if type(guess) != type({}):
        question = list(guess.keys())[0] #gets out question

        new_tree = _________________

        if _________________
            _________________

        if _________________
            _________________

        if _________________
            _________________

        if _________________
            _________________

        if _________________
            _________________

        if _________________
            _________________
6. (3 points) What Will Scheme Output? (Streams)

For each of the following Scheme expressions, write the Scheme value to which it evaluates. If evaluation causes an error, write ERROR. If evaluation never completes, write FOREVER. Assume that you have started STk and executed the following statements in order.

```
(define ones (cons-stream 1 ones))
(define a
  (cons-stream 1
    (cons-stream 3 b))
(define b
  (cons-stream 2 a))
(define happy
  (cons-stream 2
    (stream-map * happy (stream-cdr happy))))
```

<table>
<thead>
<tr>
<th>Expression</th>
<th>Interactive Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>(ss ones)</td>
<td>(1 1 1 1 1 1 1 1 1 ...)</td>
</tr>
<tr>
<td>(ss (interleave a b))</td>
<td></td>
</tr>
<tr>
<td>(stream-car happy)</td>
<td></td>
</tr>
<tr>
<td>(stream-car (stream-cdr happy))</td>
<td></td>
</tr>
</tbody>
</table>

7. (5 points) Skipping Streams

Define a stream of streams where every \(i\)th element in the larger stream is a stream of the multiples of \(i\). Assume that the stream naturals and the function add-streams that we defined in the Streams discussion are given to you, in addition to the predefined functions in STk (such as stream-map, stream-filter, interleave, etc.) Do not add in any additional lines. Do not use any additional define expressions (but you may use lambda expressions).

```
STk> (ss naturals)
(1 2 3 4 5 6 7 8 9 10 ...)
STk> (ss (add-streams naturals naturals))
(2 4 6 8 10 12 14 16 18 20 ...)
STk> (ss nth-multiples 4)
((1 2 3 4 ...) (2 4 6 8 ...) (3 6 9 12 ...) (4 8 12 16 ...) ...)
STk> (ss (stream-car (stream-cdr nth-multiples)))
(2 4 6 8 10 12 14 16 18 20 ...)
```

```
(define nth-multiples

  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------
  -----------------------------------------------)
```
8. (6 points) Everyday I’m Shufflin

Define `shuffle`, which takes in a linked list and uses mutation to create a linked list with every pair of elements in the original list swapped (see the doctests). Your function **must return a linked list**, but that list must be created using only mutation of the `rest` attribute.

**You may NOT call the Link constructor. You may NOT assign to the first attribute. The point of this problem is to mutate rest attributes. USE RECURSION.**

```python
def shuffle(lst):
    """
    >>> shuffle(Link(1, Link(2, Link(3, Link(4)))))
    Link(2, Link(1, Link(4, Link(3))))
    >>> shuffle(Link('s', Link('c', Link(1, Link(6, Link('a'))))))
    Link('c', Link('s', Link(6, Link(1, Link('a')))))
    """
    if
        return
    new_head = lst.rest
    lst.rest =
    return
```

9. (5 points) Searching the Depths

Write `member` for deep lists in Logic. You may assume there is at most one occurrence of the element we are searching for. **Hint:** Consider how we could define separate facts to check for the element in either the car of the list or the cdr of the list.

```python
logic> (query (member 5 ((((5)))))) )
Success!
logic> (query (member 4 (3 2 (1 4))))
Success!
logic> (query (member ?what ()))
Failed.
```

```
10. (6 points) Subset Sum

Define `subset_sum`, a function which takes in a list of numbers `lst` and a number `n`, and returns a subset of the elements in `lst` that adds up to `n` if such a subset exists, otherwise it returns `None`. If there are multiple subsets, return any one of them. **Hint:** We can choose either to keep the first element of the `lst` in our subset, or we can choose to not have the first element in our subset.

```python
def subset_sum(lst, n):
    """
    >>> sorted(subset_sum([4, 8, 3], 11))
    [3, 8]
    >>> print(subset_sum([4, 8, 3], 10))
    None
    >>> sorted(subset_sum([5, 8, 10, 7, 23], 40))
    [7, 10, 23]
    """
    if ______________________________________________________________
        return []
    elif ____________________________________________________________
        return None
    no_first_element = ______________________________________________
    if no_first_element is not None:
        return ______________________________________________________
    using_first_element = ___________________________________________
    if using_first_element is not None:
        return ______________________________________________________
```

11. (3 points) Concurrency

These two threads are running concurrently starting with `x = 1, y = 1`.

```python
>>> x = x + y
>>> y = x ** 2
>>> y = x - 1
>>> x = y + 1
```

1) Line a1 read x 6) Line b1 read x
2) Line a1 read y 7) Line b1 write y
3) Line a1 write x 8) Line b2 read y
4) Line a2 read x 9) Line b2 write x
5) Line a2 write y

Determine whether `x = 5, y = 5` are possible values after the code is executed. If not possible, circle "impossible". If possible, list the order in which the 9 steps are taken by their step numbers.

Impossible

----- ---- ---- ---- ---- ---- ---- ---- ----
12. (9 points) Interpreters: Implementing Special Forms

In the Scheme project, you implemented several special forms, such as if, and, begin, and or. Now we’re going to look at a new special form: when. A when expression takes in a condition and any number of other subexpressions, like this:

```
(when <condition>
  <exp>
  <exp>
  ...
  <exp>)
```

If condition is true, all of the following subexpressions are evaluated in order and the value of the when is the value of the last subexpression. If it is false, none of them are evaluated and the value of the when is okay. For example, \( \text{(when } (= \text{ days-left } 0) \text{ (print 'im-free) \text{ 'jk-final)} } \) would first evaluate \( (+ \text{ 2 3}) \) and then \( (* \text{ 1 2}) \).

(a) (2 pt) Equivalent Scheme Expression

Rewrite the when expression below into another Scheme expression which uses only the special forms you already implemented in your project. That is, create a Scheme expression which does the same thing as this when expression without using when. You should use if in your answer.

```
(when (= days-left 0)
  (print 'im-free)
  'jk-final)
```

You may or may not need to use all of the lines provided.

```
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
---------------------------------------------------------------
```
(b) (2 pt) Box and Pointer Remember that do\_when\_form, like the other do\_something\_form functions in the Scheme project, takes in vals and the env to evaluate in. We will be drawing the box-and-pointer diagram for vals above. As an example, the box-and-pointer diagram for \( (+ 1 1) \) would be

![Box and Pointer Diagram]

In the example from the description, vals would be \( '((= 1 1) (+ 2 3) (* 1 2)) \). Draw the box-and-pointer diagram for this list in the space provided below.

(c) (3 pt) Implementing When Now implement do\_when\_form. Assume that the other parts of scheme.py have already been modified to accommodate this new special form. You may not need to use all of the lines provided. You do not need to worry about tail recursion. Remember that do\_when\_form must return two things - a Scheme expression or value and an environment or None.

```python
def do\_when\_form(vals, env):
    # Implementation goes here
```

(d) (2 pt) Implementing Another Special Form Now let's implement another special form until, which takes in a condition and a series of expressions, and evaluates the expressions in order only if the condition is NOT true. Implement do\_until\_form using do\_when\_form. (Remember that Scheme has a built-in not function which your interpreter can evaluate!)

```python
def do\_until\_form(vals, env):
    new\_expr = # Implementation goes here
    return do\_when\_form(new\_expr, env)
```