1 Mutation

1.1 For each row below, fill in the blanks in the output displayed by the interactive Python interpreter when the expression is evaluated. Expressions are evaluated in order, and expressions may affect later expressions.

```python
>>> cats = [1, 2]
>>> dogs = [cats, cats.append(23), list(cats)]
>>> cats
```

```python
>>> dogs[1] = list(dogs)
>>> dogs[1]
```

```python
>>> dogs[0].append(2)
>>> cats
```

```python
>>> dogs[2].extend([list(cats).pop(0), 3])
>>> dogs[3]
```

```python
>>> dogs
```
1.2 (Fall 2013) Draw the environment diagram for the following code.

```python
def miley(ray):
    def cy():
        def rus(billy):
            nonlocal cy
            cy = lambda: billy + ray
            return [1, billy]
        if len(rus(2)) == 1:
            return [3, 4]
        else:
            return [cy(), 5]
        return cy()[1]

billy = 6
miley(7)
```
2  Recursion

2.1 Write a procedure `merge(s1, s2)` which takes two sorted (smallest value first) lists and returns a single list with all of the elements of the two lists, in ascending order. Use recursion.

**Hint:** If you can figure out which list has the smallest element out of both, then we know that the resulting merged list will have that smallest element, followed by the merge of the two lists with the smallest item removed. Don’t forget to handle the case where one list is empty!

```python
def merge(s1, s2):
    
    # Merges two sorted lists
    >>> merge([1, 3], [2, 4])
    [1, 2, 3, 4]
    >>> merge([1, 2], [])
    [1, 2]
    
    if ________________________________________________________________________:
        return s2
    elif ________________________________________________________________________:
        return s1
    elif ________________________________________________________________________:
        return ________________________________________________________________________
    else:
        return ________________________________________________________________________
```

Consider the subset sum problem: you are given a list of integers and a number $k$. Is there a subset of the list that adds up to $k$? For example:

```python
>>> subset_sum([2, 4, 7, 3], 5)  # 2 + 3 = 5
True
>>> subset_sum([1, 9, 5, 7, 3], 2)
False
>>> subset_sum([1, 1, 5, -1], 3)
False
```

def subset_sum(seq, k):
    
    if ________________________________:
        return False

    elif ________________________________:
        return True

    else:
        return ________________________________

```
3 Trees

3.1 Assuming that every value in t is a number, define \texttt{average(t)}, which returns the average of all the values in t. You may not need to use all the provided lines.

\begin{verbatim}
def average(t):
    
    Returns the average value of all the nodes in t.
    >>> t0 = Tree(0, [Tree(1), Tree(2, [Tree(3)])])
    >>> average(t0)
    1.5
    >>> t1 = Tree(8, [t0, Tree(4)])
    >>> average(t1)
    3.0
    
    def sum_helper(t):

        total, count = _______________________________________________________________

        for ________________________________________________________________________:

        __________________________________________________________________________

        __________________________________________________________________________

        __________________________________________________________________________

        __________________________________________________________________________

        return total, count

    total, count = ___________________________________________________________________

    return total / count
\end{verbatim}
4 Streams

4.1 Write a function `merge` that takes 2 sorted streams s1 and s2, and returns a new sorted stream which contains all the elements from s1 and s2. Assume that both s1 and s2 have infinite length.

```lisp
(define (merge s1 s2)
  (if ____________________________________________________________
      ___________________________________________________________
      ___________________________________________________________
      ___________________________________________________________
      ___________________________________________________________
  ___________________________________________________________
  ___________________________________________________________
  ___________________________________________________________

  ___________________________________________________________
)
```

4.2 (Adapted from Fall 2014) Implement `cycle` which returns a stream repeating the digits 1, 3, 0, 2, and 4, forever. Write `cons-stream` only once in your solution!

**Hint:** $(3+2) \mod 5 = 0$.

```lisp
(define (cycle start)
  ___________________________________________________________
)
```
5 Generators

5.1 Implement `accumulate`, which takes in an iterable and a function \( f \) and yields each accumulated value from applying \( f \) to the running total and the next element.

```python
from operator import add, mul

def accumulate(iterable, f):
    """
    >>> list(accumulate([1, 2, 3, 4, 5], add))
    [1, 3, 6, 10, 15]
    >>> list(accumulate([1, 2, 3, 4, 5], mul))
    [1, 2, 6, 24, 120]
    """
    it = iter(iterable)
    """
    for
    """
```

Write a generator function that yields functions that are repeated applications of a one-argument function $f$. The first function yielded should apply $f$ 0 times (the identity function), the second function yielded should apply $f$ once, etc.

```python
def repeated(f):
    ""
    >>> double = lambda x: 2 * x
    >>> funcs = repeated(double)
    >>> identity = next(funcs)
    >>> double = next(funcs)
    >>> quad = next(funcs)
    >>> oct = next(funcs)
    >>> quad(1)
    4
    >>> oct(1)
    8
    >>> [g(1) for _, g in zip(range(5), repeated(lambda x: 2 * x))]
    [1, 2, 4, 8, 16]
    ""
```

```python
g = ________________________________________________________________

while True:
    ________________________________________________________________

    ________________________________________________________________
```

Ben Bitdiddle proposes the following alternate solution. Does it work?

```python
def ben_repeated(f):
    g = lambda x: x
    while True:
        yield g
        g = lambda x: f(g(x))
```
6 SQL

6.1 You’re trying to re-organize your music library! The table `tracks` below contains song titles and the corresponding album. Create another table `tracklist` with two columns: the album and a comma-separated list of all songs from that album in alphabetical order.

```sql
CREATE TABLE tracks AS
    SELECT "Human" AS title, "The Definition" AS album UNION
    SELECT "Simple and Sweet", "The Definition" UNION
    SELECT "Paper Planes", "Translations Through Speakers";

CREATE TABLE tracklist AS
    WITH songs(album, total) AS (  
        ____________________________________________________________________________
    ),  
    ____________________________________________________________________________ AS (  
        ____________________________________________________________________________
    ),  
    ____________________________________________________________________________ AS (  
        ____________________________________________________________________________
    )
    SELECT ________________________________
    WHERE ________________________________;
```

```sql
sqlite3> SELECT * FROM tracklist ORDER BY album;
The Definition|Human, Simple and Sweet
Translations Through Speakers|Paper Planes
```