1. What would Python display? If an error occurs, write “Error”. If a function is displayed, write “Function”. If nothing is returned, write “Nothing”.

```python
>>> a = [1, 2]
>>> a.append([3, 4])
>>> a

>>> b = list(a)
>>> a[0] = 5
>>> a[2][0] = 6
>>> b

>>> a.extend([7])
>>> a += [8]
>>> a += 9

>>> a

Challenge:
>>> b[2][1] = a[2:]
>>> a[2][1][0][0]
```
2. Draw the environment diagram that results from running the following code.

```python
a = [1, 2, [3]]
def mystery(s, t):
    s.pop(1)
    return t.append(s)
b = a
b += [b[0]]
a = mystery(b, a[1:])
```
3. Given some list \texttt{lst}, possibly a deep list, mutate \texttt{lst} to have the accumulated sum of all elements so far in the list. If there is a nested list, mutate it to similarly reflect the accumulated sum of all elements so far in the nested list. Return the total sum of \texttt{lst}.

\textit{Hint:} The \texttt{isinstance} function returns True for \texttt{isinstance(l, list)} if \texttt{l} is a list and False otherwise.

\begin{verbatim}
def accumulate(lst):
    """
    >>> l = [1, 5, 13, 4]
    >>> accumulate(l)
    23
    >>> l
    [1, 6, 19, 23]
    >>> deep_l = [3, 7, [2, 5, 6], 9]
    >>> accumulate(deep_l)
    32
    >>> deep_l
    [3, 10, [2, 7, 13], 32]
    """

    for \______________:\n        \______________
        if \______________:
            inside = \______________
        else:
            \______________
    \______________
\end{verbatim}
1. **Nonlocal Kale**
   Draw the environment diagram for the following code.

   ```python
   eggplant = 8
   def vegetable(kale):
       def eggplant(spinach):
           nonlocal eggplant
           nonlocal kale
           kale = 9
           eggplant = spinach
           print(eggplant, kale)
       eggplant(kale)
       return eggplant
   
   spinach = vegetable('kale')
   ```
2. Pingpong again...

The ping-pong sequence counts up starting from 1 and is always either counting up or counting down. At element \( k \), the direction switches if \( k \) is a multiple of 7 or contains the digit 7.

The first 30 elements of the ping-pong sequence are listed below, with direction swaps marked using brackets at the 7th, 14th, 17th, 21st, 27th, and 28th elements:

\[
1 \ 2 \ 3 \ 4 \ 5 \ 6 \ [7] \ 6 \ 5 \ 4 \ 3 \ 2 \ 1 \ [0] \ 1 \ 2 \ [3] \ 2 \ 1 \ 0 \ [-1] \ 0 \ 1 \ 2 \ 3 \ 4 \ [5] \ [4] \ 5 \ 6
\]

Implement a function `make_pingpong_tracker` that returns the next value in the pingpong sequence each time it is called. You may use assignment statements.

```python
def has_seven(k):
    # Use this function for your answer below
    if k % 10 == 7:
        return True
    elif k < 10:
        return False
    else:
        return has_seven(k // 10)

def make_pingpong_tracker():
    """ Returns a function that returns the next value in the pingpong sequence each time it is called."
    return

def pingpong_tracker():
    index, current, add = 1, 0, True
    _________________
    if add:
        _________________________
    else:
        _________________________
    if ________________________:
        add = not add
        _________________________
    return pingpong_tracker
```

```python
index, current, add = 1, 0, True
```

```python
>>> output = []
>>> x = make_pingpong_tracker()
>>> for _ in range(9):
...     output += [x()]
>>> output
[1, 2, 3, 4, 5, 6, 7, 6, 5]
```

The first 30 elements of the pingpong sequence are listed below, with direction swaps marked using brackets at the 7th, 14th, 17th, 21st, 27th, and 28th elements: