Fall 2015, Midterm 2, #2a

2. (14 points) Exercises

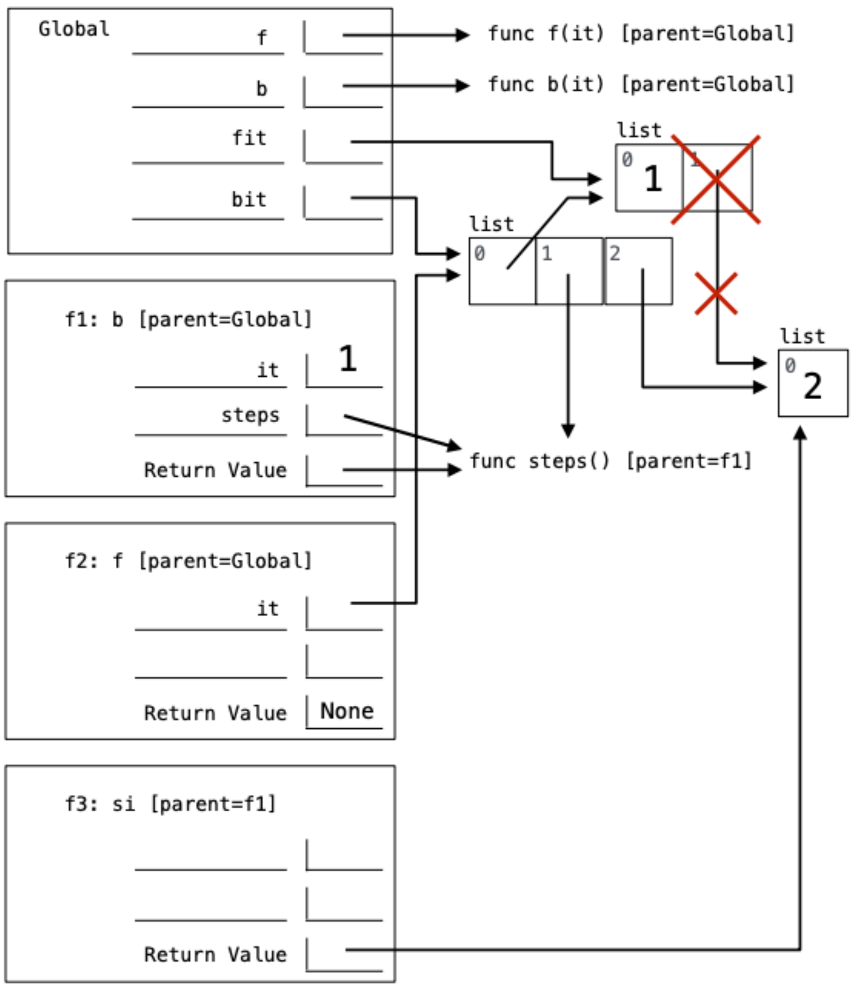
(a) (6 pt) Fill in the environment diagram that results from executing the code below until the entire program is finished, an error occurs, or all frames are filled. *You may not need to use all of the spaces or frames.*

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.

You are not required to write index numbers in list boxes.

```
def f(it):
    it.append(it[1]())

def b(it):
    def steps():
        nonlocal it
        it = fit[0]
        return fit.pop()
    return steps

fit = [1, [2]]
bit = [fit, b(fit[1])]
f(bit)
```



Fall 2015, Midterm 2, #3b

(b) (4 pt) Implement **adder**, which takes two lists x and y of digits representing positive numbers. It mutates x to represent the result of adding x and y. *Notes:* The built-in **reversed** function takes a sequence and returns its elements in reverse order. Assume that x[0] and y[0] are both positive.

```
def adder(x, y):
    """Adds y into x for lists of digits x and y representing positive numbers.

    >>> a = [3, 4, 5]
    >>> adder(a, [5, 5])            #  345 +    55 =   400
    [4, 0, 0]
    >>> adder(a, [8, 3, 4])         #  400 +   834 =  1234
    [1, 2, 3, 4]
    >>> adder(a, [3, 3, 3, 3, 3])  # 1234 + 33333 = 34567
    [3, 4, 5, 6, 7]
    """
    carry, i = 0, len(x)-1
    for d in reversed([0] + y):


        if i == -1:
            x.insert(0, 0)
            i = 0
        d = carry + x[i] + d

        carry, x[i], i = d // 10, d % 10, i-1

    if x[0] == 0:
        x.remove(0)
    return x
```

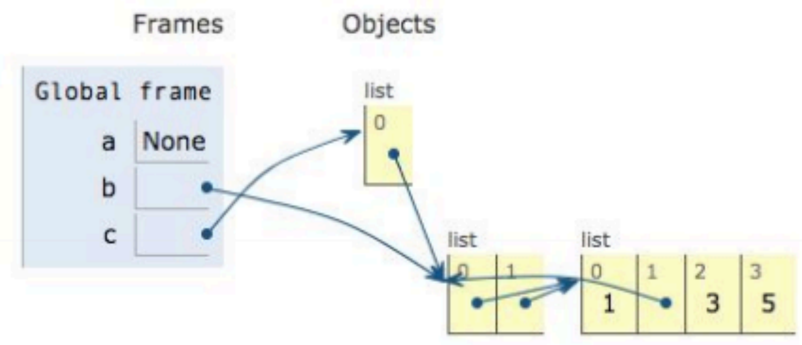
Spring 2018, Exam-Prep 4, #1

Lots of Lists

Draw the environment diagram that results from executing the following code.
PythonTutor

```
a = [1, 2, 3, 4, 5]
a.pop(3)
b = a[:]
a[1] = b
b[0] = a[:]
b.pop()
b.remove(2)
c = [].append(b[1])
a.insert(b.pop(1), a[-3:4:3])
b.extend(b)
if b == b[:] and b[1][1][0] is b[0][1][1]:
      a, b, c = [c] + a[-4:4:2]
```

Solution



Fall 2015, Midterm 2, #3e

(e) **(8 pt)** Implement `int_set`, which is a higher-order function that takes a list of non-negative integers called `contents`. It returns a function that takes a non-negative integer `n` and returns whether `n` appears in `contents`. Your partner left you this clue: Every integer can be expressed uniquely as a sum of powers of 2. E.g., 5 equals 1 + 4 equals `pow(2, 0) + pow(2, 2)`. The `bits` helper function encodes a list of `nums` using sequences of 0's and 1's that tell you whether each power of 2 is used, starting with `pow(2, 0)`.
Note: You may *not* use built-in tests of list membership, such as an `in` expression or a list's `index` method.

```
def bits(nums):
    """A set of nums represented as a function that takes 'entry', 0, or 1.

    >>> t = bits([4, 5]) # Contains 4 and 5, but not 2
    >>> t(0)(0)(1)('entry') # 4 = 0 * pow(2, 0) + 0 * pow(2, 1) + 1 * pow(2, 2)
    True
    >>> t(0)(1)('entry')    # 2 = 0 * pow(2, 0) + 1 * pow(2, 1)
    False
    >>> t(1)(0)(1)('entry') # 5 = 1 * pow(2, 0) + 0 * pow(2, 1) + 1 * pow(2, 2)
    True
    """
    def branch(last):

        if last == 'entry':

            return 0 in nums

        return bits([k // 2 for k in nums if k % 2 == last])

    return branch
```

Summer 2015, Midterm 2, #5a

5. (13 points) The weakest link

(a) (2 pt) For the following questions, assume that the following generator function is defined:

```
def naturals():
    i = 1
    while True:
        yield i
        i += 1
```

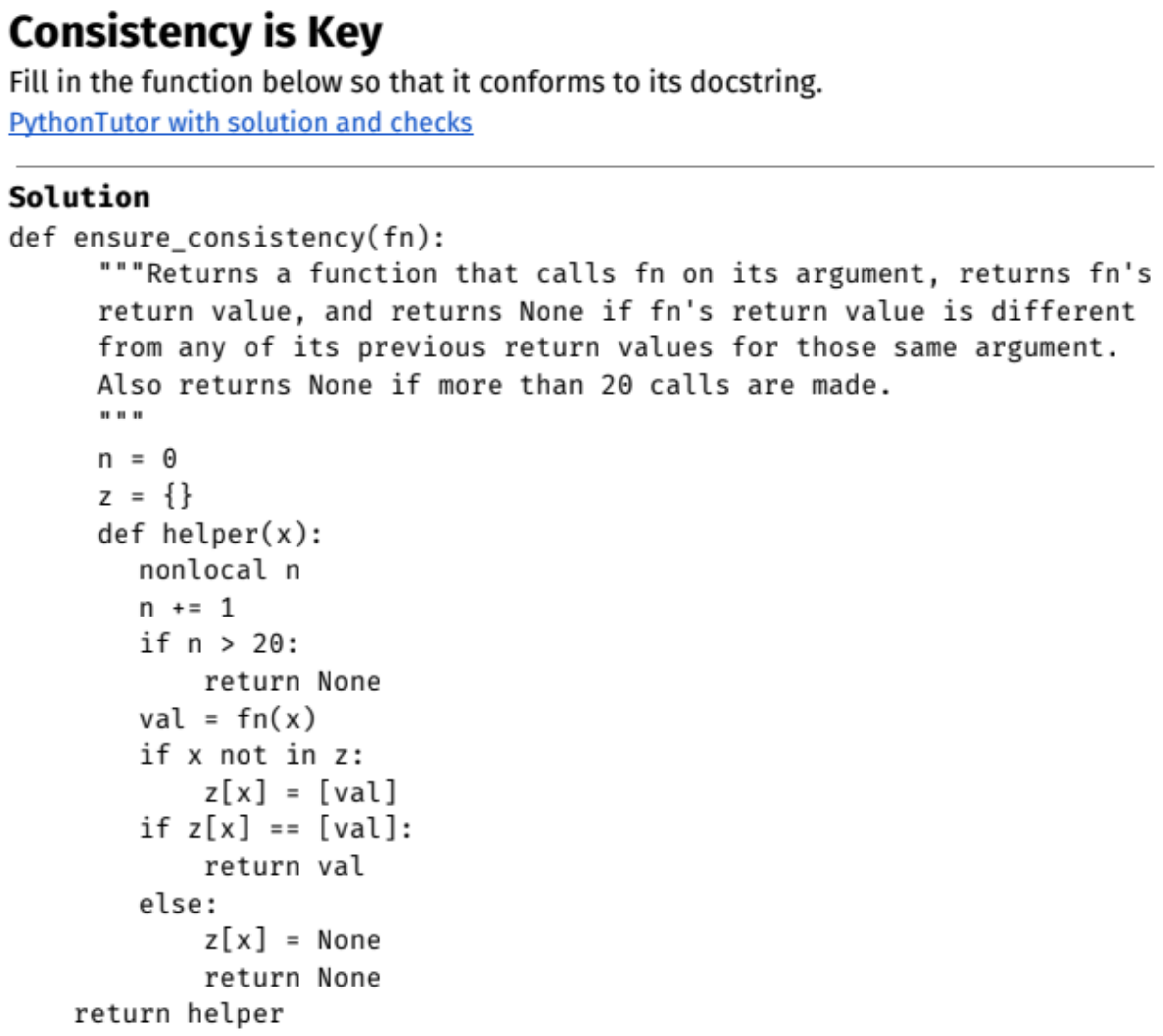
Implement a generator function called `filter(iterable, fn)` that only yields elements of `iterable` for which `fn` returns `True`.

See the doctests for expected behavior. **You may not use the built-in `filter` function or list comprehensions.**

Your solution should not require more than 3 lines, and you do not need to use all 3 lines.

```
def filter(iterable, fn):
    """
    >>> is_even = lambda x: x % 2 == 0
    >>> list(filter(range(5), is_even)
    [0, 2, 4]

    >>> all_odd = (2 * y - 1 for y in range(5))      # Generator object
    >>> list(filter(all_odd, is_even))
    []

    >>> s = filter(naturals(), is_even)
    >>> next(s)
    2
    >>> next(s)
    4
    """

    for elem in iterable:

        if fn(elem):

            yield elem
```

Spring 2018, Exam Prep 4, #4



Consistency is Key

Fill in the function below so that it conforms to its docstring.

PythonTutor with solution and checks

Solution

```
def ensure_consistency(fn):
     """Returns a function that calls fn on its argument, returns fn's
     return value, and returns None if fn's return value is different
     from any of its previous return values for those same argument.
     Also returns None if more than 20 calls are made.
     """
     n = 0
     z = {}
     def helper(x):
        nonlocal n
        n += 1
        if n > 20:
            return None
        val = fn(x)
        if x not in z:
            z[x] = [val]
        if z[x] == [val]:
            return val
        else:
            z[x] = None
            return None
    return helper
```