Lab 6: Nonlocal, Mutability

lab06.zip (lab06.zip)

Due by 11:59pm on Tuesday, October 6.

Starter Files

Download lab06.zip (lab06.zip). Inside the archive, you will find starter files for the questions in this lab, along with a copy of the Ok (ok) autograder.

Topics

Consult this section if you need a refresher on the material for this lab. It's okay to skip directly to the questions and refer back here should you get stuck.

Nonlocal

Mutability

Required Questions

Nonlocal Codewriting

Q1: Make Adder Increasing

Write a function which takes in an integer `a` and returns a one-argument function. This function should take in some value `b` and return `a + b` the first time it is called, similar to `make_adder`. The second time it is called, however, it should return `a + b + 1`, then `a + b + 2` the third time, and so on.

```
def make_adder_inc(a):
    """
    >>> adder1 = make_adder_inc(5)
    >>> adder2 = make_adder_inc(6)
    >>> adder1(2)
    7
    >>> adder1(2) # 5 + 2 + 1
    8
    >>> adder1(10) # 5 + 10 + 2
    17
    >>> [adder1(x) for x in [1, 2, 3]]
    [9, 11, 13]
    >>> adder2(5)
    11
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q make_adder_inc
```

Q2: Next Fibonacci

Write a function `make_fib` that returns a function that returns the next Fibonacci number each time it is called. (The Fibonacci sequence begins with 0 and then 1, after which each element is the sum of the preceding two.) Use a `nonlocal` statement! In addition, do not use python lists to solve this problem.

```
def make_fib():
    """Returns a function that returns the next Fibonacci number
    every time it is called.

    >>> fib = make_fib()
    >>> fib()
    0
    >>> fib()
    1
    >>> fib()
    1
    >>> fib()
    2
    >>> fib()
    3
    >>> fib2 = make_fib()
    >>> fib() + sum([fib2() for _ in range(5)])
    12
    >>> from construct_check import check
    >>> # Do not use lists in your implementation
    >>> check(this_file, 'make_fib', ['List'])
    True
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q make_fib
```

Mutability

Q3: List-Mutation

Test your understaninding of list mutation with the following questions. What would Python display? Type it in the intepreter if you're stuck!

```
python3 ok -q list-mutation -u
```

Note: if nothing would be output by Python, type `Nothing`.

```
>>> lst = [5, 6, 7, 8]
>>> lst.append(6)
------

>>> lst
------

>>> lst.insert(0, 9)
>>> lst
------

>>> x = lst.pop(2)
>>> lst
------

>>> lst.remove(x)
>>> lst
------

>>> a, b = lst, lst[:]
>>> a is lst
------

>>> b == lst
------

>>> b is lst
------
```

Q4: Insert Items

Write a function which takes in a list `lst`, an argument `entry`, and another argument `elem`. This function will check through each item present in `lst` to see if it is equivalent with `entry`. Upon finding an equivalent entry, the function should modify the list by placing `elem` into the list right after the found entry. At the end of the function, the modified list should be returned. See the doctests for examples on how this function is utilized. Use list mutation to modify the original list, no new lists should be created or returned.

Be careful in situations where the values passed into `entry` and `elem` are equivalent, so as not to create an infinitely long list while iterating through it. If you find that your code is taking more than a few seconds to run, it is most likely that the function is in a loop of inserting new values.

```
def insert_items(lst, entry, elem):
    """
    >>> test_lst = [1, 5, 8, 5, 2, 3]
    >>> new_lst = insert_items(test_lst, 5, 7)
    >>> new_lst
    [1, 5, 7, 8, 5, 7, 2, 3]
    >>> large_lst = [1, 4, 8]
    >>> large_lst2 = insert_items(large_lst, 4, 4)
    >>> large_lst2
    [1, 4, 4, 8]
    >>> large_lst3 = insert_items(large_lst2, 4, 6)
    >>> large_lst3
    [1, 4, 6, 4, 6, 8]
    >>> large_lst3 is large_lst
    True
    """
    "*** YOUR CODE HERE ***"
```

Use Ok to test your code:

```
python3 ok -q insert_items
```

Submit

Make sure to submit this assignment by running:

```
python3 ok --submit
```

CS 61A (/~cs61a/fa20/)

Weekly Schedule (/~cs61a/fa20/weekly.html)

Office Hours (/~cs61a/fa20/office-hours.html)

Staff (/~cs61a/fa20/staff.html)

Resources (/~cs61a/fa20/resources.html)

Studying Guide (/~cs61a/fa20/articles/studying.html)

Debugging Guide (/~cs61a/fa20/articles/debugging.html)

Composition Guide (/~cs61a/fa20/articles/composition.html)

Policies (/~cs61a/fa20/articles/about.html)

Assignments (/~cs61a/fa20/articles/about.html#assignments)

Exams (/~cs61a/fa20/articles/about.html#exams)

Grading (/~cs61a/fa20/articles/about.html#grading)