Lab 6 Solutions **lab06.zip (lab06.zip)**

Solution Files

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.

Required Questions

Nonlocal Codewriting

For the following question, write your code in `lab06.py`.

Q1: Make Adder Increasing

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

```
def make_adder_inc(n):
    """
    >>> 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
    """
    def adder(x):
        nonlocal n
        value = n + x
        n = n + 1
        return value
    return adder
```

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!

```
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
    """
    cur, next = 0, 1
    def fib():
        nonlocal cur, next
        result = cur
        cur, next = next, cur + next
        return result
    return fib
```

Use Ok to test your code:

```
python3 ok -q make_fib
```

Generators

Generators also allow us to represent infinite sequences, such as the sequence of natural numbers (1, 2, ...).

```
def naturals():
    """A generator function that yields the infinite sequence of natural
    numbers, starting at 1.

    >>> m = naturals()
    >>> type(m)
    <class 'generator'>
    >>> [next(m) for _ in range(10)]
    [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
    """
    i = 1
    while True:
        yield i
        i += 1
```

Q3: Scale

Implement the generator function `scale(it, multiplier)`, which yields elements of the given iterable `it`, scaled by `multiplier`. As an extra challenge, try writing this function using a `yield from` statement!

```
def scale(it, multiplier):
    """Yield elements of the iterable it scaled by a number multiplier.

    >>> m = scale([1, 5, 2], 5)
    >>> type(m)
    <class 'generator'>
    >>> list(m)
    [5, 25, 10]

    >>> m = scale(naturals(), 2)
    >>> [next(m) for _ in range(5)]
    [2, 4, 6, 8, 10]
    """
    for elem in it:
        yield elem * multiplier

# Alternate solution
def scale_alt(it, multiplier):
    yield from map(lambda x: x*multiplier, it)
```

Use Ok to test your code:

```
python3 ok -q scale
```

Q4: Hailstone

Write a generator that outputs the hailstone sequence from homework 1.

Here's a quick reminder of how the hailstone sequence is defined:

1. Pick a positive integer `n` as the start.
2. If `n` is even, divide it by 2.
3. If `n` is odd, multiply it by 3 and add 1.
4. Continue this process until `n` is 1.

For some extra practice, try writing a solution using recursion. Since `hailstone` returns a generator, you can `yield from` a call to `hailstone`!

```
def hailstone(n):
    """
    >>> for num in hailstone(10):
    ...     print(num)
    ...
    10
    5
    16
    8
    4
    2
    1
    """
    while n > 1:
        yield n
        if n % 2 == 0:
            n //= 2
        else:
            n = n * 3 + 1
    yield n

# Alternate Solution

def hailstone_alt(n):
    yield n
    if n > 1:
        if n % 2 == 0:
            yield from hailstone_alt(n // 2)
        else:
            yield from hailstone_alt(n * 3 + 1)

    # Video Walkthrough: https://youtu.be/fQlIJa2_yqw?t=1h18m52s
```

Use Ok to test your code:

```
python3 ok -q hailstone
```

Submit

Make sure to submit this assignment by running:

```
python3 ok --submit
```

CS 61A (/)

Weekly Schedule (/weekly.html)

Office Hours (/office-hours.html)

Staff (/staff.html)

Resources (/resources.html)

Studying Guide (/articles/studying.html)

Debugging Guide (/articles/debugging.html)

Composition Guide (/articles/composition.html)

Policies (/articles/about.html)

Assignments (/articles/about.html#assignments)

Exams (/articles/about.html#exams)

Grading (/articles/about.html#grading)