CS61A Exam Prep 4

We will start at Berkeley Time!

Summer 2021 Midterm Q2a

Write a function `order_order` which takes a non-empty list of 2-argument functions `operators` and returns a new 2-argument function. When called, this returned function should print the result of applying the first function in operators to the two parameters passed in. It should then return another function that applies the second function in `operators` to the parameters, and so on. When the returned function has called the last function in the `operators` list, it should cycle back to the beginning of the list and use the first function again on the next call.

See the doctest for an example.

```
def order_order(operators):
    """
    >>> from operator import add, mul, sub
    >>> ops = [add, mul, sub]
    >>> order = order_order(ops)
    >>> order = order(1, 2) # applies add and returns mul
    3
    >>> order = order(1, 2) # applies mul and returns sub
    2
    >>> order = order(1, 2) # applies sub and cycles back to return add
    -1
    >>> order = order(1, 2) # cycles back to applying add
    3
    >>> order = order(1, 2)
    2
    >>> order = order(1, 2)
    -1
    """
    def apply(x, y):
        print(_____)
        #          (a)
        return order_order(_____)
        #                  (b)
    return _____
    #          (c)
```

operators[1:] + [operators[0]]

[mul, sub] add

operators[0](x)(y)

add(x, y)

operators[0](x,y)

operators[1:]+[operators[0]] add(3,2)

apply

Summer 2021 Midterm Q2b

A skip list is defined as a sublist of a list such that each element in the sublist is non adjacent in the original list. For original list `[5, 6, 8, 2]`, the lists `[5, 8]`, `[5, 2]`, `[6, 2]`, `[5]`, `[6]`, `[8]`, `[2]`, `[]` are all skip lists of the original list. The empty list is always a skip list of any list.

Given a list `int_lst` of unique integers, return a list of all unique skip lists of `int_lst` where **each skip list contains integers in strictly increasing order**. The order in which the skip lists are returned does not matter.

```
def list_skipper(int_lst):
    """
    >>> list_skipper([5,8,2])
    [[5, 8], [5], [6], [8], [2], []]
    >>> list_skipper([1,2,3,4,5])
    [[1, 3, 5], [1, 3], [1, 4], [1, 5], [1], [2, 4], [2, 5], [2], [3, 5], [3], [4], [5], []]
    >>> list_skipper([])
    [[]]
    """
    if len(int_lst) == 0:
        return _____
                  # (a)
    with_first = _____
                  # (b)
    without_first = _____
                  # (c)
    with_first = [ _____ for x in with_first if x == [] or _____ ]
                  # (d)                                    (e)
    return with_first + without_first
```

Fall 2018 Midterm 2 Q4a

Definition. A *plus expression* for a non-negative integer `n` is made by inserting `+` symbols in between digits of `n`, such that there are **never more than two consecutive digits** in the resulting expression. For example, one plus expression for 2018 is `20+1+8`, and its value is 29. Assume that a two-digit number starting with 0 evaluates to its one's digit. For example, another plus expression for 2018 is `2+01+8`, and its value is 11.

(a) (3 pt) Implement `plus`, which takes a non-negative integer `n`. It returns the largest value of any plus expression for `n`.

```
def plus(n):
    """Return the largest sum that results from inserting +'s into n.

    >>> plus(123456)      # 12 + 34 + 56 = 102
    102
    >>> plus(1604)        # 1 + 60 + 4 = 65
    65
    >>> plus(160450)      # 1 + 60 + 4 + 50 = 115
    115
    """
    if n:
        return max(n%10 + plus(n//10), n%100 + plus(n//100))

    return 0
```

n%10 n//10
6 + 12345
56 + 1234

Fall 2018 Midterm 2 Q4b

(b) (5 pt) Implement `plusses`, which takes non-negative integers `n` and `cap`. It returns the number of plus expressions for `n` that have a value less than `cap`.

```
def plusses(n, cap):
    """Return the number of plus expressions for n with values below cap.

    >>> plusses(123, 16)  # 1+2+3=6 and 12+3=15, but 1+23=24 isn't below cap.
    2
    >>> plusses(2018, 38) # 2+0+1+8, 20+1+8, 2+0+18, and 2+01+8, but not 20+18.
    4
    >>> plusses(1, 2)
    1
    """
    if n<10 and n<cap:
        return 1
    elif cap<0:
        return 0
    else:
        return plusses(n//10, cap-n%10) + plusses(n//100, cap-n%100)
```

n%10
n//10
1+23