# CS61A Exam Prep 3

We will start Berkeley Time!

Summer 2019 Final Q3a

**3. (14 points) One More Time**

**Definition.** An $(n)$-*repeater* for a single-argument function `f` takes a single argument `x`, calls `f(x)` $n$ times, then returns an $(n+1)$-repeater for `f`.

**(a) (6 pt)** Implement `repeater`, which takes a single-argument function `f` and a positive integer $n$. It returns an $(n)$-repeater for `f`. Also implement the helper function `repeat`.

```
def repeater(f, n):
    """Return an (n)-repeater for f.
    >>> r = repeater(print, 2)
    >>> s = r('CS')
    CS
    CS
    >>> t = s('CS')
    CS
    CS
    CS
    """
    def g(x):
        _____repeat(f, x, n)_____
        return _____repeater(f, n+1)_____
    return _____g_____

def repeat(f, x, n):
    """Call f(x) n times.
    >>> repeat(print, 'Hello', 3)
    Hello
    Hello
    Hello
    """
    if _____n > 0_____:
        _____f(x)_____
        _____repeat(f, x, n-1)_____
```

f(x) assume this works!

repeat(f, x, n-1)

n = 3

repeat(f, x, 1)

repeat(f, x, 0)

Summer 2021 Diagnostic Q3

**3. (1.0 points) Oh, Camel!**

**Definition**: A *camel sequence* is an integer in which each digit is either strictly less than or strictly greater than both of its adjacent digits. Write a function `is_camel_sequence` that takes in a nonnegative integer `n` and returns whether `n` is a *camel sequence*.

*Note: Any single digit integer is a valid camel sequence.*

**Restrictions**: You may not use `int`, `str`, `[` or `]` in your solution.

n//10

```
def is_camel_sequence(n):
    """
    >>> is_camel_sequence(15263) # 1 < 5, 5 > 2, 2 < 6, 6 > 3
    True
    >>> is_camel_sequence(98989)
    True
    >>> is_camel_sequence(123) # 1 < 2, but 2 is not greater than 3.
    False
    >>> is_camel_sequence(4114) # 1 is not strictly less than 1
    False
    >>> is_camel_sequence(1)
    True
    >>> is_camel_sequence(12)
    True
    >>> is_camel_sequence(11)
    False
    """
    def helper(n, incr):
        if _____:          #       (a)
            return True
        elif incr:
            return _____ and helper(_____, _____)
            #              (b)                (c)
        else:
            return _____ and helper(_____, _____)
            #              (d)                (e)
    return _____ or _____
    #          (f)        (g)
```

n%.10

(n//10)%.10

n==0 → True

n < 10 → True

previous 2 digits direction (bool)

n//10 == 0

True (n//10)%.10 < n%.10

(n//10)%. > n%.10

helper(n, True) or helper(n, False)

helper or False

helper(n, ~)

Spring 2021 Practice Midterm 1 Q2a

n//10

**2. (8.0 points) Significant Factors**

**(a) (3.0 points)**

Implement `significant`, which takes positive integers `n` and `k`. It returns the `k` *most significant* digits of `n` as an integer. These are the first `k` digits of `n`, starting from the left. If `n` has fewer than `k` digits, it returns `n`. You may not use `round`, `int`, `str`, or any functions from the `math` module.

**You may use** `pow`, which raises its first argument to the power of its second: `pow(9, 2)` is 81 and `pow(9, 0.5)` is 3.0.

```
def significant(n, k):
    """Return the K most significant digits of N.

    # Case 1
    >>> significant(12345, 3)
    123

    # Case 2
    >>> significant(12345, 7)
    12345
    """
    if _____________________:
#       (a)
        return n

    return significant(_____________, _____________)
#                      (b)           (c)
```

pow(10, k) > n

k or less digits

1000 > 12345 n X

n//10 1234

k 3

Spring 2021 Practice Midterm 1 Q2b

**(b) (5.0 points)**

Implement `factorize`, which takes two integers `n` and `k`, both larger than 1. It returns the number of ways that `n` can be expressed as a product of non-decreasing integers greater than or equal to `k`.

```
def factorize(n, k=2):
    """Return the number of ways to factorize positive integer n.

    # Case 1
    >>> factorize(7) # 7
    1

    # Case 2
    >>> factorize(12) # 2*2*3, 2*6, 3*4, 12
    4

    # Case 3
    >>> factorize(36) # 2*2*3*3, 2*2*9, 2*3*6, 2*18, 3*3*4, 3*12, 4*9, 6*6, 36
    9
    """
    if n == k:
        return 1
    elif __________:
#        (a)
        return 0
    elif __________:
#        (b)
        return factorize(n, k + 1)
    return __________
#          (c)
```

Fall 2014
Midterm 1
Q3

**3. (8 points) Express Yourself**

**(a) (3 pt)** A k-bonacci sequence starts with K-1 zeros and then a one. Each subsequent element is the sum of the previous K elements. The 2-bonacci sequence is the standard Fibonacci sequence. The 3-bonacci and 4-bonacci sequences each start with the following ten elements:

```
              n: 0, 1, 2, 3, 4, 5, 6,  7,  8,  9, ...

kbonacci(n, 2): 0, 1, 1, 2, 3, 5, 8, 13, 21, 35, ...

kbonacci(n, 3): 0, 0, 1, 1, 2, 4, 7, 13, 24, 44, ...

kbonacci(n, 4): 0, 0, 0, 1, 1, 2, 4,  8, 15, 29, ...
```

Fill in the blanks of the implementation of `kbonacci` below, a function that takes non-negative integer `n` and positive integer `k` and returns element $n$ of a $k$-bonacci sequence.

```
def kbonacci(n, k):
    """Return element N of a K-bonacci sequence.

    >>> kbonacci(3, 4)
    1
    >>> kbonacci(9, 4)
    29
    >>> kbonacci(4, 2)
    3
    >>> kbonacci(8, 2)
    21
    """

    if n < k - 1:

        return 0

    elif n == k - 1:

        return 1

    else:

        total = 0


        i = ____________________________________________________________

        while i < n:


            total = total + ____________________________________________

            i = i + 1

        return total
```

go.cs61a.org/melanie-feedback

recording and notes will be posted on the website later today!