Math 308: Bridge to Advanced Math
Programming Assignment 3
Solutions

1. The *tribonacci sequence* is a sequence of integers defined inductively by $a_0 = a_1 = 0$, $a_2 = 1$, and $a_{n+3} = a_n + a_{n+1} + a_{n+2}$ for integers $n \geq 0$. Write a function `tribonacci(m)` which takes as input an number $m \geq 1$ and returns the list $[a_0, a_1, a_2, \ldots, a_k]$ where a_k is the largest number in the sequence with $a_k < m$.

   Examples of successful input/output:

```
>>> tribonacci(100)
[0, 0, 1, 1, 2, 4, 7, 13, 24, 44, 81]
>>> tribonacci(1)
[0, 0]
>>> tribonacci(81)
[0, 0, 1, 1, 2, 4, 7, 13, 24, 44]
```

   Solution:

```
def tribonacci(m):
    l=[]
    if (m>0):
        l.append(0)
        l.append(0)
    if (m>1):
        l.append(1)
    last=1
    while last<m:
        last = l[len(l)-1]+l[len(l)-2]+l[len(l)-3]
        if last<m:
            l.append(last)
    return l
```

2. Viewing addition as a binary operation, the Catalan number C_k is the number of ways to write $k+1$ as a sum of $k+1$ ones. Here $k \geq 0$ is an integer. For example $C_0 = 1$ because 1 can only be expressed as 1, and $C_1 = 1$ because $2 = 1 + 1$ is the only way to write 2 as a sum of ones. But, $C_2 = 2$ because

$$3 = (1+1)+1 = 1+(1+1),$$

   and $C_3 = 5$ because

$$4 = 1+\big(1+(1+1)\big) = 1+\big((1+1)+1\big) = (1+1)+(1+1) = \big((1+1)+1\big)+1 = \big(1+(1+1)\big)+1.$$

   Suppose we have an expression for $k+1$ as a sum of ones. Then there is an outermost addition, and we can simplify the left and right sides. For example, the sum $\big(1+(1+1)\big)+1$ simplifies to $3+1$. Every sum representing $k+1$ simplifies to a sum of the form $a+b$ with $a \geq 1$, $b \geq 1$

and $a + b = k + 1$. Furthermore in such a sum a and b are each represented as a sum of ones. It follows that for $k \geq 1$ we have

$$C_k = \sum_{i=0}^{k-1} C_i C_{k-i-1}.$$

(Here each term $C_i C_{k-i-1}$ represents number of ways to write $k+1$ as a sum of ones which simplifies to $(i+1)+(k-i)$.)

Write a function `catalan_numbers(n)` which returns the list of the first n Catalan numbers: $[C_0, C_1, \ldots, C_{n-1}]$. *Remark:* All you really need to know about the Catalan numbers is that $C_0 = 1$ and the summation formula above. (The Catalan numbers show up in a lot of counting problems. Wikipedia has a nice article on the Catalan numbers.)

Examples of successful input/output:

```
>>> catalan_numbers(10)
[1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862]
```

Solution:

```
def catalan_numbers(n):
    if n<1:
        return []
    c=[1]
    for k in range(1,n):
        sum=0
        for i in range(k):
            sum = sum + c[i]*c[k-i-1]
        c.append(sum)
    return c
```

3. Write a recursive function `multiply(m,n)` which takes as input two integers m and n, and returns their product $m * n$ only using addition and subtraction. Hints:

$$m * 0 = 0, \quad m * n = m * (n-1) + m \quad \text{and} \quad m * n = m * (n+1) - m.$$

(*Remark: Your function should work for all integers.*)

Solution:

```
def multiply(m,n):
    if n==0:
        return 0
    if n>0:
        return multiply(m,n-1)+m
    if n<0:
        return multiply(m,n+1)-m
```

4. Suppose $f : \mathbb{R} \to \mathbb{R}$. Then, we define $f^{\circ k}$ to be f applied $k \in \mathbb{N}$ times:

$$f^{\circ k}(x) = \overbrace{f \circ f \circ \ldots \circ f}^{k}(x)$$

for $x \in \mathbb{R}$. Write a recursive function `iterate(f,k,x)` which takes as input a function $f : \mathbb{R} \to \mathbb{R}$, a natural number k, and a floating point number x and returns the value of $f^{\circ k}(x)$.

Examples of successful input/output:

```
>>> def g(x):
        return x**2

>>> iterate(g,3,1.1)
2.143588810000001
>>> def h(x):
        return 2*x*(1-x)

>>> iterate(h,100,0.25)
0.5
```

Solution:

```
def iterate(f,k,x):
    if k<=0:
        return x
    else:
        return iterate(f, k-1, f(x))
```

5. The middle third Cantor set is defined by a limiting process. We define $C_0 = [0, 1]$ and will define C_i inductively for integers $i \geq 0$. Each C_i is a finite union of closed intervals. For each integer $i \geq 0$, we define $C_{i+1} \subset C_i$ by removing the middle third of the intervals in C_i. So for example

$$C_1 = \left[0, \frac{1}{3}\right] \cup \left[\frac{2}{3}, 1\right] \quad \text{and} \quad C_2 = \left[0, \frac{1}{9}\right] \cup \left[\frac{2}{9}, \frac{1}{3}\right] \cup \left[\frac{2}{3}, \frac{7}{9}\right] \cup \left[\frac{8}{9}, 1\right].$$

The *middle third Cantor set* is defined by $C = \bigcap_{i=0}^{\infty} C_i$.

Write a function `cantors_set_contains(n,x)` which takes as input an integer $n \geq 0$ and returns the truth value of the statement $x \in C_n$.

You do not have to use recursion. But, if you want to use recursion, it might be helpful to use the function $f : C_1 \to C_0$ defined by

$$f(x) = \begin{cases} 3x & \text{if } x \in \left[0, \frac{1}{3}\right] \\ 3x - 2 & \text{if } x \in \left[\frac{2}{3}, 1\right]. \end{cases}$$

Then you can define C_i by iteration:

$$C_i = \left\{x \in [0, 1] : \{x, f(x), f^{\circ 2}(x), \ldots, f^{\circ (i-1)}(x)\} \subset C_1\right\}.$$

An equivalent way to define C_i is

$$C_i = \{x : f^{\circ i}(x) \text{ is well defined}\}.$$

(Can you see why these versions of C_i are correct?)

Hint: For testing, it may be useful to note that $\frac{1}{4} \in C$ while $\frac{1}{2 \cdot 3^k} \notin C$ for any integer $k \geq 0$.

Examples of successful input/output:

```
>>> cantors_set_contains(3, 5/6)
False
>>> cantors_set_contains(10, 1/4)
True
```

Solution:

```
def cantors_set_contains(n,x):
    if x < 0:
        return False
    if x <= 1:
        # Here we know that x lies in [0,1]
        if n==0:
            return True
        if x <= 1/3:
            # Here we know x lies in [0,1/3].
            return cantors_set_contains(n-1, 3*x)
        if x < 2/3:
            # Here we know x lies in (1/3, 2/3) and n>1.
            return False
        # Here we know x lies in [2/3, 1].
        return cantors_set_contains(n-1, 3*x-2)
    # Here we know x>1.
    return False
```