CS250 homework 4

name: Austen Nelson

Due: 7/23/19

1. The Fibonacci numbers are a fun sequence of numbers that show up in math a lot. The sequence is $1, 1, 2, 3, 5, 8, 13, 21, \ldots$, and we compute each number by adding the previous two. Write an iterative (while loop) python function to compute the n^{th} Fibonacci number. so `fib(6)` should return `8`.

```
def fib_iter(a):
    num0 = 0
    num = 1

    for i in range(a - 1):
        old = num
        num = num + num0
        num0 = old

    return num
```

2. Now write a recursive program to compute the n^{th} Fibonacci number.

```
def fib_rec(a):
    if a <= 1:
        return a

    return fib_rec(a - 1) + fib_rec(a - 2)
```

3. Let $e = gcd(n, m\%n)$, Prove that $e|m$ and $e|n$

$n, m, e, x, y, r, q, z \in \mathbf{N}$	variable declarations
$m \mod n = r$	initializing r
$m = qn + r$	rewriting modulus
$e \mid n$	definition of gcd
$n = ze$ so $qn \equiv qze$	substitution
$e \mid qn$ or $qn = xe$	because $qn = qze$ and definition of divides
$e \mid r$ or $r = ye$	definition of gcd and definition of divides
$m = xe + ye$ or $m = e(x + y)$	substitution and algebra
$e \mid m$	definition of divides

4. Prove that for any $a, b, c \in \mathbf{N}$ if $a|b$ and $a|c$ then $a|bx + cy$ for any $x, y \in \mathbf{N}$.

$a, b, c, x, y, s, z \in \mathbf{N}$	variable declarations
$b = az$ and $c = as$	definition of divides
$bx + cy \equiv azx + asy \equiv a(zx + sy)$	substitution and factor
$a \mid a(zx + sy)$	definition of divides

5. The distance between two vertices in a graph is the length of the shortest path between them. We usually write the distance between u and v in graph G as $d(u, v)$. Show that for any three vertices $u, v, t \in G$ $d(u, v) \leq d(u, t) + d(t, v)$.

   Case 1: t is on the shortest path from u to v. $d(u, t) + d(t, v) = d(u, v)$ because if there exists a shorter path (u,t) or (t,v) then there exists a path shorter than our shortest path (u,v).
   Case 2: t is not on the shortest path from u to v. Path (u,t) must diverge from shortest path (u,v). $d(u, t) + d(t, v) > d(u, v)$ because path $u \to t \to v$ is not the shortest path (u,v).

6. In the video `https://www.youtube.com/watch?v=2SUvWfNJSsM` he shows that you can solve towers of hanoi by counting in binary. Make this explicit by writing a python function `move_disk(n)` where `n` is a binary number. `move_disk` should return the number of the disk to be moved. So, to move the 3^{rd} disk, you'd return 3. You can test your code in `hanoi.py` on d2l. Right now it just loops forever

```
def move_disk(i):
   h = 5
   while h > 0:
      if i % 2**h == 0:
         return h + 1
      h = h - 1
   return 1
```

7. give a bijection as a python function for

   - $\mathbf{E} \to \mathbf{N}$

```
def E_N(even)
   return even/2
```

   - $\mathbf{N} \to \mathbf{Z}$

```
import math

def N_Z(nat)
   return (-1 ** nat) * math.floor(nat / 2)
```

   - $\mathbf{N} \to \mathbf{Q}^+$

```
import math
from fractions import Fraction

def N_Qpos(nat)
   z = 0

   #Calkin-Wilf series
   for i in range(nat):
      z = Fraction(1, 2 * math.floor(z) - z + 1)

   return z
```

   - $\mathbf{Z} \to \mathbf{Q}$

```
def Z_Q(an_int)
   if an_int == 0:
      return 0

   if an_int > 0:
      return N_Qpos(an_int)

   return -1 * N_Qpos(-1 * an_int)
```

- **E → Q**

```
def E_Q(even)
    return Z_Q(N_Z(E_N(even)))
```

8. Let's try to find a use for this infinity nonsense.

   In computer science it can be useful to look at problems as a language.
   A language is just a set of finite strings.
   So we can make a language describing π as $L_\pi = \{"3", "3.1", "3.14", "3.141", \ldots\}$

   So why do we care about languages?
   Well, we can phrase all of our problems in CS as different languages.
   For example $L_{factor} = \{"6 = 2 \cdot 3", "12 = 2 \cdot 2 \cdot 3", \ldots\}$
   is the language of numbers and their factors.
   We can make the language of graphs and their shortest paths, the languages of lists and their sorting.
   Really we can make a language for any problem.

   A language is decidable if there is a program that can (eventually) produce any string in that language.

   We want to prove that there is at least 1 undecidable language.
   That is, there is a problem that can't be solved by a program.

   First show that there are countably many programs we can write.
   Hint: what happens when you compile a program?
   A program is just a string. A string can be represented as a binary number. Therefore, a program is just a number within **N**. This means we have an injection from programs to **N** and that there are countably infinite possible programs.
   second show that there are uncountably many languages.
   As stated in the example every real number can be represented as a language. This means we there exists a surjection from programs to the reals and that there is an uncountable number of languages.