Recursive Functions

Here is a recursive function to find *n*! (factorial).

```
def fact(n):
   if n == 0:
      return 1
   else:
      return n * fact(n - 1)
```

To find `fact(4)`, we simply write out each step in the recursive computation:

fact(4)
4 * fact(3)
4 * 3 * fact(2)
4 * 3 * 2 * fact(1)
4 * 3 * 2 * 1 * fact(0)
4 * 3 * 2 * 1 * 1

Exercise 1

```
def m(a, b):
   if a < b:
      return a
   else:
      return m(a - b, b)
```

Find `m(3, 5)`.
Find `m(7, 5)`.
Find `m(14, 5)`.
What does `m` do?

Exercise 2

```
def gcd(a, b):
   if b == 0:
      return a
   else:
      return gcd(b, a % b)
```

Use this definition to find `gcd(15, 9)`.
Now find `gcd(13, 8)`.

Exercise 3

The function `power` should compute the value of b^n, where n is any non-negative integer. Fill in the blanks below, so that power behaves correctly.

```
   def power(b, n):

      if n == __________:

         return __________

      else:
         return __________ * power(__________, __________)
```

Hint: How does knowing the value of 3^9 help you find the value of 3^{10}?

Exercise 4

```
def double(n):
   if n == 0:
      return 1
   else:
      return double(n - 1) + double(n - 1)
```

Find `double(3)`.
What does this function do?
Can you modify the definition of `double` so that it computes the same result with a single recursive call?

Exercise 5

```
def f(x):
   if x == 1:
      return 1
   else:
      if x % 2 == 0:
         return f(x / 2)
      else:
         return f(3 * x + 1)
```

Find `f(3)`.
Find `f(7)`.
Can you find a positive integer `x` so that `f(x)` that results in an infinite loop?