def is\_divisible(x, y): # funcytion from section 6.4

if x % y == 0:

return true

else :

return false

def is\_power (a, b):

if is\_divisible (a, b ) == True:

# This call the is divisible function to check, if a is divisible by b.

if a == b :

# Same numbers are the power of themselves unless I'm mistakened (3^1 == 3).

return True

elif b== 1 :

# codes moves onto the next condition if b does not equal to 1.

if a == 1 :

# This condittion checks that only a power of 1 is 1 itself

return True

else:

return False

elif: a / b % b == 0:

# This condition checks if a/b is a power of b

return True

else:

return False

print(" is\_power ( 10, 2) returns: ",

is\_power (10, 2))

print("is\_power (27, 3) returns: ",

is power ( 27, 3) )

print("is\_power(1, 1 ) returns: ",

is\_power(1, 1))

print("is\_power(10, 1) returns: ",

is\_power(10 , 1 ))

print("is\_power(3, 3) returns: ",

is\_power (3, 3))

# I'm sure theres a cleaner and more concice way tp write this but I am out of time

One way to think about recursive functions is to start with base cases — usually these are the things you know to be true just by looking. In this case you know two things:

• 1 is a power of everything because anything to the power of zero is 1  
• Only 1 is a power of 1

In code that base case might look like:

def is\_power(a, b):

if a == 1:

return True

if b == 1:

return a == 1

If a == 1 this returns true because of base one. If a is not 1 then this will look at b if b is one it will only return true is a is one.

Alone this returns True for is\_power(1, 1), but false for is\_power(10, 1).

So for everything else:

If a is not divisible by b then it's not power. If it is divisible by b then it's a power if a/b is a power of b. You can test the first with your existing function and the second by passing a/b back into this function (i.e. recursing):

def is\_divisible(x, y):

if x % y == 0:

return True

else:

return False

def is\_power(a, b):

if a == 1 or b == 1: # a more succinct way to test base case

return a == 1

return is\_divisible(a, b) and is\_power(a/b, b) # divisible and recursions report True?

Recursive functions take a while to get used to. It can help to add some print statements to the functions so you can see how it is being called.

Recursion is a method of programming or coding a problem, in which a function calls itself one or more times in its body. Usually, it is returning the return value of this function call. If a function definition satisfies the condition of recursion, we call this function a recursive function.  
  
Termination condition:  
A recursive function has to fulfil an important condition to be used in a program: it has to terminate. A recursive function terminates, if with every recursive call the solution of the problem is downsized and moves towards a base case. A base case is a case, where the problem can be solved without further recursion. A recursion can end up in an infinite loop, if the base case is not met in the calls.  
  
Example:  
  
4! = 4 \* 3!  
3! = 3 \* 2!  
2! = 2 \* 1  
  
Replacing the calculated values gives us the following expression  
  
4! = 4 \* 3 \* 2 \* 1  
  
Generally we can say: Recursion in computer science is a method where the solution to a problem is based on solving smaller instances of the same problem.