Lab 3 (10 points) **Due date:** June 18th, 2023, 11:59 PM ET

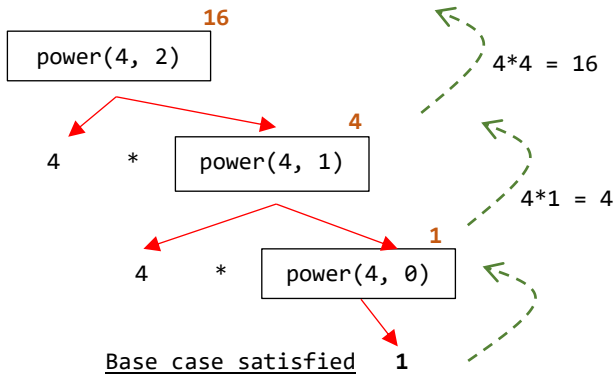
Before you start…

Keep in mind these tips from your instructor:

- Watch the lectures in Module 4 before you start this assignment, including the hands-on videos
- While all functions in this assignment can be easily completed using loops, the use of any form of loop is not allowed and using them will result in a 0 score for the function. This is an assignment meant to help you practice recursion, thus, all functions must be implemented recursively
- Outline a strategy before you start typing code and share it with the course staff for additional support
- Recursion trees and environment diagrams are great tools to help with design and debugging of recursive functions

```
def power(x, n):
    if n == 0:
        return 1
    else:
        return x * power(x, n-1)
```

4^2: The value of power(4, 2) is calculated as:



When designing a recursive function, you don't need to trace out the entire sequence of recursive calls in your code. If the function returns a value, all you need to do is to confirm that the following three properties are satisfied:

1. Each base case (stopping condition) returns the correct value for that case
2. There is no infinite recursion (every chain of recursive calls eventually reaches the base case)
3. For cases that involve recursion, if all recursive calls return the correct value, then the final value returned by the function is the correct value

- Ask questions using our Lab 3 channel in Microsoft Teams. Reach out to the course staff id you are stuck and need help

REMINDER: As you work on your coding assignments, it is important to remember that passing the examples provided does not guarantee full credit. While these examples can serve as a helpful starting point, it is ultimately your responsibility to thoroughly test your code and ensure that it is functioning correctly and meets all the requirements and specifications outlined in the assignment instructions. Failure to thoroughly test your code can result in incomplete or incorrect outputs, which will lead to deduction in points for each failed case.

Create additional test cases and share them with you classmates on Teams, we are in this together!

90% > Passing all test cases 10% > Clarity and design of your code

IMPORTANT: You are not allowed to use loops in any form, global variables, the **in** operator, or list methods in any portion of this assignment (len() is ok to use). You will not receive credit if you use them. The use of helper functions is NOT allowed. Other restrictions apply

is_power_of(base, num) (1.5 points)

Determines whether *num* is a power of *base* (there is some non-negative integer *k* such that pow(*base*, *k*) equals *num*.

- You are not allowed to use pow, the ** operator, or import any function (such as math.log)

Preconditions

```
base: int -> Positive integer
num: int -> Non-negative integer
```

Examples:

```
>>> is_power_of(7, 7)      # 7**1 = 7
True
>>> is_power_of(7, 0)      # 0 is not a power of any base
False
>>> is_power_of(7, 2401)   # 7**4 = 2401
True
>>> is_power_of(7, 1)      # 7**0 = 1
True
>>> is_power_of(7, 2458)   # 2458 is not a power of 7
False
```

cut(a_list) (2 points)

Takes a list of integers and returns a list identical to *a_list*, but when a negative number appears, the function deletes the negative number and the next (x – 1) elements, where x is the absolute value of the negative number deleted. The function should not mutate *a_list*.

- You are not allowed to use the *in* operator
- You are not allowed to use the index() method
- You are not allowed to use the copy() method
- You are not allowed to make full copies of numList

Preconditions
`a_list: list -> sequence of items`

Examples:

```
>>> cut([7, 4, -2, 1, 9])   # Found -2: Delete -2 and 1
[7, 4, 9]
>>> cut([-4, -7, -2, 1, 9]) # Found -4: Delete -4, -7, -2 and 1
[9]
```

right_max(num_list) (2.5 points)

Given a list of numbers, return a new list in which each element is replaced by the maximum of itself and all the elements following it. The function should not mutate *num_list*.

- You are not allowed to use the *in* operator
- You are not allowed to use the index() method
- You are not allowed to use the copy() method
- You are not allowed to make full copies of numList
- You are not allowed to use any built-in sorting methods

Preconditions
`num_list: list -> sequence of items`

Examples:

```
>>> right_max([3, 7, 2, 8, 6, 4, 5])
[8, 8, 8, 8, 6, 5, 5]
>>> right_max([1, 2, 3, 4, 5, 6])
[6, 6, 6, 6, 6, 6]
>>> right_max([1, 25, 3, 48, 5, 6, 12, 14, 89, 3, 2])
[89, 89, 89, 89, 89, 89, 89, 89, 89, 3, 2]
```

consecutive_digits (num) (1.5 points)

Returns True if *num* contains two consecutive digits that are the same, False otherwise. As described in Homework 1, floor division (//) and modulo (%) operators are useful here.

- You are not allowed to convert *n* to other data type such str() or list()
- You are not allowed to use lists
- You are not allowed to use pow, the ** operator, or import any function (such as math.log)

Preconditions
`n: int -> positive integer`

```
>>> consecutive_digits(2222466666678)
True
>>> consecutive_digits(12345684562)
False
>>> consecutive_digits(122)
True
```

only_evens (num) (2.5 points)

Takes a positive number and returns an integer formed by removing the odd digits from *num*. If *num* is zero or does not have even digits, return 0. As described in Homework 1, floor division (//) and modulo (%) operators are useful here.

- You are not allowed to convert *n* to other data type such str() or list()
- You are not allowed to use lists
- You are not allowed to use pow, the ** operator, or import any function (such as math.log)

Preconditions

`n: int -> positive integer`

```
>>> only_evens(4386112)
4862
>>> only_evens(0)
0
>>> only_evens(357997555531)
0
>>> only_evens(13847896213354889741236)
84862488426
```