Problem 1.

Without using loops, the `**` operator, or any function in the `math` module, write a function with the signature

```
power(x, n)
```

which returns x raised to the power of n. You can assume n is a non-negative integer.

Problem 2.

Without using loops, write the function with the signature

```
interleave(L1, L2)
```

which takes `L1` and `L2`, two lists of the same length, and returns a list which consists of `L1` and `L2` interleaved, i.e., `[L1[0], L2[0], L1[1], L2[1], ..., L1[n-1], L2[n-1]]` (here, `n == len(L1) == len(L2)`).

Problem 3.

Without using loops or slicing, write a function that reverses a list in place. That is, the effect of calling

```
reverse_rec(L)
```

should be that `L` is reversed.

Here is how you might do this *with* loops:

```
def reverse_loop(L):
  for i in range(len(L)//2):
    L[i], L[-1-i] = L[-1-i], L[i]
```

Problem 4.

Fun fact: this question is taken from the final exam that I wrote in my first year.

Without using loops and without ever using `print` with a list (as opposed to individual elements of a list), write a function that, given a list `L` of size n (assume n is odd), prints the elements of `L` in the following order:

`L[n//2] L[n//2-1] L[n//2+1] L[n//2-2] L[n//2+2] L[n//2-3] L[n//2+3] ... L[n-1]`

Hint: here is a recursive function that prints the following sequence for a list `L` of size n:

`L[0] L[n-1] L[1] L[n-2] L[2] L[n-3] ... L[n//2]`

```
def zigzag1(L):
  if len(L) == 0:
    print('')
  elif len(L) == 1:
    print(L[0], end = "")
  else:
    print(L[0], L[-1], end = "")
    zigzag1(L[1:-1])
```

Problem 5.

Without using any loops or global variables, write a function the with signature (exactly, without default parameters)

`is_balanced(s)`

which returns `True` iff the string `s` is has "balanced" parentheses, i.e., all parentheses `()` in string `s` match exactly. For example, `"(()(()))"` is balanced but the following:

`"(well (I think), recursion works like that (as far as I know)"`

is not, since it's missing a closing parenthesis. To simplify matters, you may start by thinking about strings that contain only parentheses, but your final function should work with all strings. Your function should only care about balancing parentheses `()`, not brackets `[]` or braces `{}`.

Problem 6.

This is the last CSC180 lab this semester – say goodbye and thank you to your lab TAs!