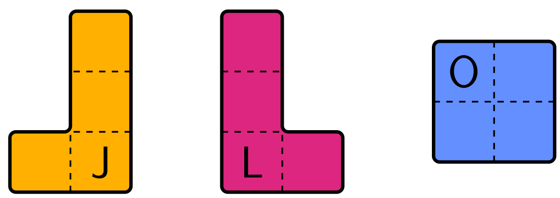
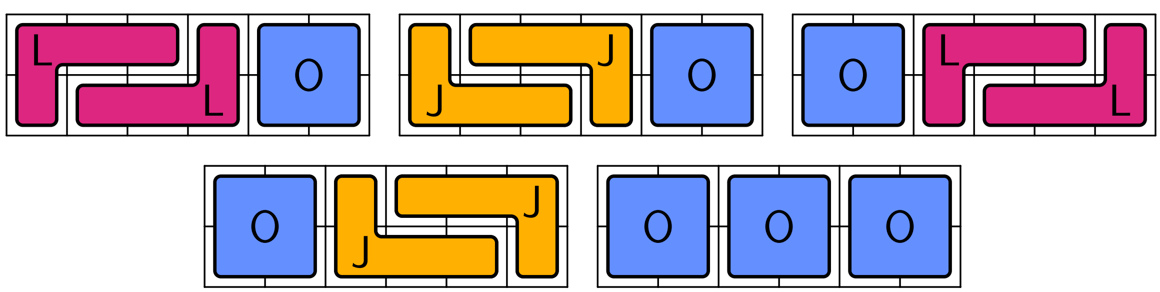
Due at 11:59PM, Friday, March 21st, on Canvas. Submissions must be ***typed*** (except for questions labeled with a “📷” icon) and in PDF format. Show your work for full credit!

Each question is about designing a backtracking algorithm. First, give a ***brief, verbal explanation*** of your solution to the problem (perhaps with the help of a ***mathematical recurrence***). Then, write down pseudocode for the backtracking algorithm. Each algorithm must be a ***pure function***, and should not contain any other function declarations.

1. The video game *Tetris* is played with seven different pieces that consist of four connected squares. For this problem, we consider three of those pieces: ***J, L, and O***:



Suppose you have a $2 \times n$ grid that you want to ***tile*** (every cell in the grid is covered, no pieces overlapping) with these pieces. Like in *Tetris*, you may rotate pieces, but you cannot flip any over (otherwise J becomes L, and vice versa). Below are the 5 ways of tiling the 2×6 grid:



Design a ***backtracking*** algorithm for computing the ***number of ways*** of tiling a $2 \times n$ grid with these pieces, for any integer $n \geq 1$ (or $n \geq 0$, if you'd like). Your algorithm should be a pure, integer-valued function of the form:

```
TETRIS(n):
    // your code here
```

2. In class, we looked at the text splitting problem: given a string S, determine whether S can be split into individual words. One of the algorithms we wrote down looks like the following:

```
SPLITTABLE(S[1,...,n], i):
    if i = n + 1
        return TRUE
    for j = i,...,n
        if S[i,...,j] is a word
            if SPLITTABLE(S, j + 1)
                return TRUE
    return FALSE
```

   By only ***modifying the highlighted parts*** and introducing ***a temporary variable*** in the above algorithm (and changing the name of the function), design a ***backtracking*** algorithm for ***counting*** the number of ways to split the string S into words. Your algorithm should be a pure, integer-valued function of the form:

```
COUNTSPLIT(S[1,...,n], i):
    // your code here
```

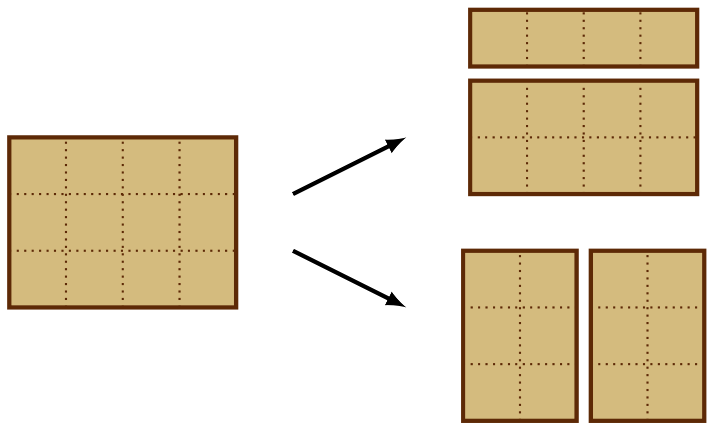
3. At Blossom Bake Shop, cupcakes come in sets of 6, 9, or 20. Suppose you need to buy a certain amount of cupcakes for a party. Is it possible to buy sets of cupcakes so that you have exactly the cupcakes you need? For example, if you needed $n = 38$ cupcakes, then you can buy a set of 20, and two sets of 9, but if you needed $n = 43$ cupcakes, you would have to buy some extra cupcakes.

   Design a ***backtracking*** algorithm for determining ***whether it is possible*** to buy exactly n cupcakes, for any integer $n \geq 0$. Your algorithm should be a pure, Boolean-valued function of the form:

```
CUPCAKE(n):
    // your code here
```

   (Note: negative values of n are not allowed, so that later, it'll be easier to convert your answer into a dynamic programming algorithm)

4. In this problem, we'll consider a two-dimensional variant of the woodcutting problem from class. Given a rectangular piece of wood, you are allowed to make an axis-aligned cut that segments the piece into two smaller rectangles. For example, if you have a 3×4 rectangle, some of the ways you can cut it look like the following:



Each of those smaller rectangles can be cut further into even smaller pieces, as well.

Suppose you are given a table of (integer-valued) selling prices $P[1, \ldots, M][1, \ldots, N]$ where $P[x][y]$ is the price of an $x \times y$ rectangle of wood (you may assume that $P[x][y] = P[y][x]$, that rotating the piece 90 degrees doesn't change its value). Describe a ***backtracking*** algorithm for computing the ***maximum amount of money*** you can make by cutting up a rectangle of size $m \times n$, where $m \leq M$ and $n \leq N$. Your algorithm should be a pure, integer-valued function of the form

```
RECT(P[1,...,M][1,...,N], m, n):
    // your code here
```

(Hint: careful, you might not be able to immediately cut out the first piece in the same way as we did for the 1D wood-cutting problem)