National University of Singapore
School of Computing
CS1101S: Programming Methodology
Semester I, 2024/2025

S4
Tree recursion

Problems:

1. The following pattern of numbers is called Pascal's triangle.

```
                1
            1       1
        1       2       1
    1       3       3       1
1       4       6       4       1
```

   The numbers at the edge of the triangle are all 1, and each number inside the triangle is the sum of the two numbers above it.

   Write a function `pascal` that computes elements of Pascal's triangle by means of a recursive process, i.e. the function should take two arguments, `row` (counted from the top, starting with 0), and `position` (counted from the left, starting with 0), and return the element for the specified row and position. Note that only one element must be returned and NOT the entire row. E.g. calling the method with `row = 2` and `position = 1` should return `2`. Likewise calling the method with `row = 4` and `position = 2` should return `6`.

2. Similar to the tree for `fib(5)` in section 1.2.2 and the tree for `cc(11, 5)` in the solution for exercise 1.14, draw the tree illustrating the process generated by `pascal(4, 3)` to compute the value in Pascal's triangle in row 4 and position 3. Does your function `pascal` give rise to a recursive or an iterative process?

3. Many programming languages have the ability to work with *higher-order functions*, namely, functions that manipulate and generate other functions.

   Indeed, we will be using functions which may be applied to different types of arguments and may return different types of values. To keep track of this, it will be helpful to have some simple notation to describe types of values in Source.

   Two basic types of values are *Number* and *Boolean*. *Number* are the numbers in The Source such as 3, -4.2, 6.931479453e89. *Boolean* are the truth values `true`,`false`. The function `math_sqrt` may be applied to a *Number* and will return another *Number*. We indicate this with the notation:

   `math_sqrt` : *Number* → Number

   We call the type of this function *Number-Transformation*:

   Number-Transformation := *Number* → *Number*

   and thus we can also say

   `math_sqrt` : *Number-Transformation*

If `f` and `g` are functions of type *Number-Transformation*, then we may *compose* them:

```
function compose(f, g){
    return x => f(g(x));
}
```

and the result of `compose(f, g)` is another *Number-Transformation*.

Consider `compose(math_sqrt, math_log)`, which is a function of type *Number-Transformation*. What is the result of `compose(math_sqrt, math_log)(math_E)`?

What is the result of `compose(math_log, math_sqrt)(math_E * math_E)`?

The in-class sheet of Studio S4 will focus on higher-order functions and number-transformations. Please prepare the studio session by carefully reading the definitions above and section 1.3 of SICP JS.