Laboratory 4: Functions

(1) Write a function of x that calculates the expression $\sqrt{\dfrac{x^2+1}{x^2+2}}$.

(2) Write a function of n that calculates $\displaystyle\sum_{\nu=0}^{n}\frac{1+\nu}{1+\nu^2}$. If n is negative, the function returns 0, if n is zero, then it returns 1.

(3) Write a function of n and m that prints out the n by m grid on the right.

```
+-+-+-+
| | | |
+-+-+-+
| | | |
+-+-+-+
| | | |
+-+-+-+
| | | |
+-+-+-+
| | | |
+-+-+-+
| | | |
+-+-+-+
```

```
*********
**** ****
***   ***
**     **
*       *
**     **
***   ***
**** ****
*********
```

(4) Write a function that prints out m asterisks followed by $2n$ spaces, followed by m asterisks. Then use this function repeatedly to print out the pattern on the left.

(5) Write a function of n that calculates the number of divisors of n exclusive 1 and n. (Just try out all numbers between 1 and n. Later, we will talk about more sophisticated methods.)

(6) Create a function of a sum, the annual interest rate, and a number of years that calculates the value of the sum after the stated number of years receiving annual interest payments. Create another function that accumulates interests every month with 1/12 of the rate. Write a program that for a sum of 10000 and interest rate between 2% and 5% shows the accumulated amount after 20 years.

(7) It is possible for a function that calls itself. For example, we have

$$n! = n \times (n-1)!$$

for positive n. In order to use this formula, we also need a base case, namely $0! = 1$. We can implement this in Python by:

```
def rfac(n):
    if n<= 0:
        return 1
```

```
    else:
        return n*rfac(n-1)
```

We can use the same pattern in order to calculate the Fibonacci numbers 0, 1, 1, 2, 3, 5, 8, 13, 21, … defined recursively as

$$\text{fib}_0 = 0, \text{fib}_1 = 1, \text{fib}_n = \text{fib}_{n-1} + \text{fib}_{n-2}$$

Your task is to implement the Fibonacci numbers using a recursive (i.e. self-calling) function. What do you observe when you calculate larger Fibonacci numbers?