IT1007 Introduction to Programming with Python and C
Lab Exercise 02

Submission instructions:

1. There are two parts in this lab. For Part A, you have to submit within the same day of your lab session. For your Part B, you have **six days** to work on it. (E.g. if your Tlab is on Monday, then your deadline is the coming Sunday midnight.) There are two separate skeleton files for each part of the lab. Please submit them separately. **Please do NOT merge the two parts into one single file**.
2. Submit your solution to the **CORRECT** group folder in the IVLE workbin.
3. Add your student number to your file in your submission, e.g. **`Lab_02_Part_A_A1234567X.py`**.
4. You have to name your functions exactly as the questions stated.
5. You should not need any "`print()`" in the final submission inside the function you wrote

Failure to follow each of the instruction will result in 10% deduction of your marks.

Part A (Deadline: Same day of your Tlab)

Question 1

We have 4 very good friends Albert(A), Billy(B), Carl(C) and David(D), and we want to see which pair of them are the ultimate best friends to each other. We have a machine that can test a pair of person how "matching" they are and give a score.

In order to find the best pair, we have to try every pair, namely:

AB, AC, AD, BC, BD and CD

So we have to do test **6** times. How many times do we have to do if we have *n* friends instead of 4? And what if we want to find the best matching team with *k* persons instead of a pair of 2 when the machine can measure *k* persons at the same time?

In fact, this is called "*n* choose *k*", or the ***binomial coefficient***, in combinatorics, or simply ${}_nC_k$ or $\binom{n}{k}$.

The formula is $\binom{n}{k} = \frac{n!}{k!(n-k)!}$

a. Write a function **nChooseK(n,k)** to compute this number by using the factorial function provided in the skeleton file.
b. Write a recursive version **nChooseK_recursive(n,k)** to compute the binomial coefficient by using recursion without using any factorial functions. The binomial coefficient can be expressed in another form:

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

And $\binom{n}{n} = \binom{n}{0} = 1$.

Sample output:

```
>>> print(nChooseK(6,2))
15
>>> print(nChooseK(10,5))
252
>>> print(nChooseK_recursive(10,5))
252
>>>
```

Question 2 (Fibonacci Sequences)

Fibonacci numbers are defined as

$$f(n) = \begin{cases} 1 & \text{if } n < 2 \\ f(n-1) + f(n-2) & \text{otherwise} \end{cases}$$

Please see the Tlab slides for more details and explanation.

Your task

a. Implement a recursive function **fib_recursive(n)**, which takes in a positive integer, *n*, as input and return the n^{th} Fibonacci number. What is the biggest number you can compute and why? Submit the answer within the code as comments.
b. Using **fib_recursive(n)** defined in part (a), implement function **fib_sum(n)**, which returns the sum of the first *n* Fibonacci numbers.

Sample output:

```
>>> fib_recursive(0)
1
>>> fib_recursive(1)
1
>>> fib_recursive(2)
2
>>> fib_recursive(5)
8
>>> fib_recursive(10)
89
>>> fib_recursive(20)
10946
```

Part B (Deadline: Day of Tlab + 6 days)

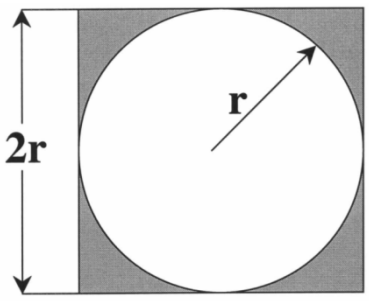
Question 3 (Computing the approximation of the value of π)

The constant π is used extensively in Mathematics, Physics and other related fields such as Engineering.

Scenario

Gambler Jack is no mathematician. His friend makes a bet that Jack does not know the number π, not the first three most significant digits. Jack is going to lose the bet but his girlfriend is an accountant and she is going to help. She told Jack the followings.

See? You see that dartboard on the wall with the frame?



The circle has a radius ***r*** and the frame is a square with length 2***r***. What we will do is, we will throw a lot of darts randomly onto the board. And the probability ***p*** of the dart hitting the circle will be:

$$p = \frac{\text{Area of Circle}}{\text{Area of Square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

So, if you throw a lot of darts within the frame and some of them hit the circle randomly, the probability is the same as ***p*** and we have

$$\frac{\text{No. of dart in circle}}{\text{Total no. of darts}} = p = \frac{\text{Area of Circle}}{\text{Area of Square}} = \frac{\pi r^2}{(2r)^2} = \frac{\pi}{4}$$

So, we can calculate π by the above formula! For example, if I throw 10 darts and 4 lands inside the circle, then $\pi \approx 1.6$

Your task

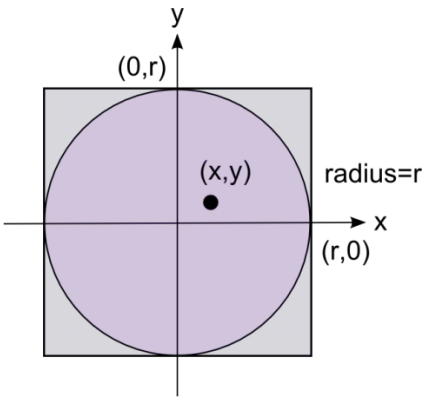
Your task is to implement a function, **monte_carlo_pi(n)** which returns the approximation of π by throwing the darts n times.

Sample output:

```
>>> monte_carlo_pi(10)
3.6
>>> monte_carlo_pi(100)
3.32
>>> monte_carlo_pi(1000)
3.084
>>> monte_carlo_pi(10000)
3.1348
>>> monte_carlo_pi(100000)
3.1412
>>> monte_carlo_pi(1000000)
3.140808
>>> monte_carlo_pi(10000000)
3.1413096
>>>
```

Hint:

The algorithm to approximate the value of π using the method described earlier is as follows.



1. Let the radius of the circle be r and the length of the square be 2r.
2. Generate two random numbers x, y, such that $-r \leq x, y \leq r$. The position (x, y) is our dart position. *Hint: Use the function in the package `random` that starts with the letter 'u'.*
3. Calculate the distance ***d*** from the dart position to the center of the circle.
4. If $d \leq r$. This implies that the point, (x, y) is within the circle. Otherwise, (x, y) falls outside of the circle, but in the square
5. Repeat the procedure n times, and keep track of the number of points that fall inside the circle, k.
6. After you have completed n trials, compute $p = \frac{k}{n}$.
7. For large n, the value of π can be approximated by $4p$.