<div style="border: 2px solid red;">

Instructions

- You are not allowed to use concepts not covered in class or from sections beyond *Libraries and Applications*.

- Make sure your code is clean, well-organized, uses meaningful variable names, includes useful comments, and is efficient.

- Make sure you edit the sections (`#1` mandatory, `#2` if applicable, and `#3` optional) in the given `notes.txt` file as appropriate. In section `#1`, for each problem, state its goal in your own words and describe your approach to solve the problem along with any issues you encountered and if/how you managed to solve those issues.

</div>

Part I: Warmup Exercises

The exercises in this part of the assignment are intended to give you solid practice on concepts (using and defining functions) needed to solve the problems in Part II.

Exercise 1. (*Reverse*) Implement the function `_reverse()` in `reverse.py` that reverses the one-dimensional list a in place, ie, without creating a new list.

```
×  ~/workspace/rsa_cryptosystem

$ python3 reverse.py
b  o  l  t  o  n
<ctrl-d>
n  o  t  l  o  b
$ python3 reverse.py
m  a  d  a  m
<ctrl-d>
m  a  d  a  m
```

Exercise 2. (*Euclidean Distance*) Implement the function `_distance()` in `distance.py` that returns the Euclidean distance between the vectors x and y represented as one-dimensional lists of floats. The Euclidean distance is calculated as the square root of the sums of the squares of the differences between the corresponding entries. You may assume that x and y have the same length.

```
×  ~/workspace/rsa_cryptosystem

$ python3 distance.py 2
1 0 <enter>
0 1 <enter>
1.4142135623730951
$ python3 distance.py 5
-9 1 10 -1 1 <enter>
-5 9 6 7 4 <enter>
13.0
```

Exercise 3. (*Transpose*) Implement the function `_transpose()` in `transpose.py` that creates and returns a new matrix that is the transpose of the matrix represented by the argument a. Note that a need not have the same number rows and columns. Recall that the transpose of an m-by-n matrix A is an n-by-m matrix B such that $B_{ij} = A_{ji}$, where $0 \leq i < n$ and $0 \leq j < m$.

```
×  ~/workspace/rsa_cryptosystem
$ python3 transpose.py 2 2
1 2 <enter>
3 4 <enter>
1.0 3.0
2.0 4.0
$ python3 transpose.py 2 3
1 2 3 <enter>
4 5 6 <enter>
1.0 4.0
2.0 5.0
3.0 6.0
```

Exercise 4. (*Palindrome*) Implement the function `_isPalindrome()` in `palindrome.py` that returns `True` if the argument s is a palindrome (ie, reads the same forwards and backwards), and `False` otherwise.

```
×  ~/workspace/rsa_cryptosystem
$ python3 palindrome.py bolton
False
$ python3 palindrome.py amanaplanacanalpanama
True
```

Exercise 5. (*Sine Function*) Implement the function `_sin()` in `sin.py` that calculates the sine of the argument x (in radians), using the formula

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots .$$

```
×  ~/workspace/rsa_cryptosystem
$ python3 sin.py 60
0.8660254037844385
```

Part II: RSA Cryptosystem

Goal: The goal of this assignment is to implement the RSA public-key cryptosystem.

Background: RSA (Rivest-Shamir-Adleman) cryptosystem is widely used for secure communication in browsers, bank ATM machines, credit card machines, mobile phones, smart cards, and operating systems. It works by manipulating integers. To thwart eavesdroppers, the RSA cryptosystem must manipulate huge integers (hundreds of digits), which is naturally supported by the `int` data type in Python. Your task is to implement a library that supports core functions needed for developing the RSA cryptosystem, and implement programs for encrypting and decrypting messages using RSA.

The Math Behind: The RSA cryptosystem involves three integers n, e, and d that satisfy certain mathematical properties. The *public key* (n, e) is made public on the Internet, while the *private key* (n, d) is only known to Bob. If Alice wants to send Bob a message $x \in [0, n)$, she encrypts it using the function

$$E(x) = x^e \bmod n,$$

where $n = pq$ for two distinct large prime numbers p and q chosen at random, and e is a random prime number less than $m = (p-1)(q-1)$ such that e does not divide m.

For example, suppose $p = 47$ and $q = 79$. Then $n = 3713$ and $m = 3588$. Further suppose $e = 7$. If Alice wants to send the message $x = 2020$ to Bob, she encrypts it as

$$E(2020) = 2020^7 \bmod 3713 = 516.$$

When Bob receives the encrypted message y, he decrypts it using the function

$$D(y) = y^d \bmod n,$$

where $d \in [1, m)$ is the multiplicative inverse of $e \bmod m$, ie, d is an integer that satisfies the equation $ed \bmod m = 1$.

Continuing the example above, if $d = 2563$, then when Bob receives the encrypted message $y = 516$ from Alice, he decrypts it to recover the original message as

$$D(516) = 516^{2563} \bmod 3713 = 2020.$$

Problem 1. (*RSA Library*) Implement a library called `rsa.py` that provides functions needed for developing the RSA cryptosystem. The library must support the following API:

`keygen(lo, hi)`	generates and returns the public/private keys as a tuple (n, e, d), picking prime numbers p and q needed to generate the keys from the interval `[lo, hi)`
`encrypt(x, n, e)`	encrypts `x` (int) using the public key (`n, e`) and returns the encrypted value
`decrypt(y, n, d)`	decrypts `y` (int) using the private key (`n, d`) and returns the decrypted value
`bitLength(n)`	returns the least number of bits needed to represent `n`
`dec2bin(n, width)`	returns the binary representation of `n` expressed in decimal, having the given width and padded with leading zeros
`bin2dec(n)`	returns the decimal representation of `n` expressed in binary

```
× ~/workspace/rsa_cryptosystem
$ python3 rsa.py S
encrypt(S) = 1743
decrypt(1743) = S
bitLength(83) = 7
dec2bin(83) = 1010011
bin2dec(1010011) = 83
```

Problem 2. (*Keygen Program*) Write a program called `keygen.py` that receives *lo* (int) and *hi* (int) as command-line inputs, generates public/private keys (n, e, d), and writes the keys as standard output, separated by a space. The interval $[lo, hi)$ specifies the interval from which prime numbers p and q needed to generate the keys are picked.

```
× ~/workspace/rsa_cryptosystem
$ python3 keygen.py 50 100
3599 1759 2839
```

Problem 3. (*Encryption Program*) Write a program called `encrypt.py` that receives the public-key n (int) and e (int) as command-line inputs and a message to encrypt as standard input, encrypts each character in the message, and writes its fixed-width binary representation as standard output.

```
×  ~/workspace/rsa_cryptosystem
```
```
$ python3 encrypt.py 3599 1759
CS110
<ctrl-d>
0001100000000010011010100001010100011001010100011001110000110010111100100
```

Problem 4. (*Decryption Program*) Write a program called `decrypt.py` that receives the private-key n (int) and d (int) as command-line inputs and a message to decrypt (produced by `encrypt.py`) as standard input, decrypts each character (represented as a fixed-width binary sequence) in the message, and writes the decrypted character as standard output.

```
×  ~/workspace/rsa_cryptosystem
```
```
$ python3 decrypt.py 3599 2839
0001100000000010011010100001010100011001010100011001110000110010111100100
<ctrl-d>
CS110
$ python3 encrypt.py 3599 1759 | python3 decrypt.py 3599 2839
Python is the mother of all languages.
<ctrl-d>
Python is the mother of all languages.
```

Data: Be sure to test your programs thoroughly using files provided under the `data` folder. For example:

```
×  ~/workspace/rsa_cryptosystem
```
```
$ python3 keygen.py 50 100
5963 4447 367
$ python3 encrypt.py 5963 4447 < data/adams.txt | python3 decrypt.py 5963 367
The major difference between a thing that might go wrong and a thing that cannot
possibly go wrong is that when a thing that cannot possibly go wrong goes wrong
it usually turns out to be impossible to get at and repair.
```

Files to Submit:

1. `reverse.py`

2. `distance.py`

3. `transpose.py`

4. `palindrome.py`

5. `sin.py`

6. `rsa.py`

7. `keygen.py`

8. `encrypt.py`

9. `decrypt.py`

10. `notes.txt`

Acknowledgement: Part II of this assignment is an adaptation of the RSA Public-key Cryptosystem assignment developed at Princeton University by Robert Sedgewick and Kevin Wayne.