Learning Outcome: Students will have a solid understanding of how to work with multiple threads/processes and how to write thread safe programs in Python.

Part 1: A Brute Force Attack against Symmetric Cryptography

In symmetric ciphers, a secret key, also known as a symmetric key, is used to encrypt and decrypt messages. A straightforward way to crack such a cipher is to adopt a brute force approach where an attacker searches the key used in encryption/decryption by testing every possible key until a match is found. In order to launch a brute force attack, attackers must have at least one pair of an original message and its encrypted version.

Open the **keysearch.py** module and examine it. The encryption/decryption functions along with the attack method are provided for you to use or modify. The `enc_dec` function is provided as well to illustrate how encryption and decryption are generally used with a common key source.

Copy the program to a new file named **fast_keysearch.py** and run it. You should see the following output:

```
enc_message = b'gAAAAABhGAoFb5oomlH2GPVg0hDYLWYZxN-PMUdfeN2ukc8PuzBzIAlh507
87SYQqMeQVep1fmwKK2rvYxW-70Hvt5d8KFwQdcLC-PhAjMYnt2diA8dsPfw='
dec_message = b'Hello, Welcome to CSEC 201!'
```

Once you get the overall idea of how the program works, comment out the call to the `enc_dec` function and uncomment the block of code at the bottom in the `if __name__ == '__main__'` block. Try to run it. You may need to fix a minor error in the program. You should be able to have the following output though the elapsed time may vary:

```
Slow search starts...
Key source found is AAAAAAAAAAAAAAAAAAAAAAAAAAAAAxwM
Slow search is done in 15.62 seconds
```

Note that the code you just uncommented does not use the encrypt. Instead, you will need to work on the pair of the hard-coded plaintext and ciphertext in the script. Your task is to experiment with multiple threads/processes to the performance of the brute force key search algorithm. Your final solution must show a significant time reduction as follows:

```
Fast search starts...
Key source found is AAAAAAAAAAAAAAAAAAAAAAAAAAAAAxwM
Fast search is done in 3.32 seconds
```

Requirements:

1. Use Python's `threading` or `multiprocessing` module. There is no restriction on the number of processes or threads you may use.
   Note: Some of you may know a better/newer way for launching parallel tasks in Python such as the `concurrent.futures` module. Though such features are more

convenient and easier to use, you are discouraged to use them in this lab as they may hide some important details you need to learn in this lab.
2. Your main script needs to receive the result, i.e. keysource, computed from another thread or process.
3. In order to get a full credit, your solution must be at least two times faster than the serial search.
4. You are not allowed to use the `encrypt` function (for some reason).

Part 2: The Dining Philosophers Problem

The Dining Philosophers is a classic synchronization problem in the field of Computer Science. You can read more about the problem here:

https://pages.mtu.edu/~shene/NSF-3/e-Book/MUTEX/TM-example-philos-1.html.

Your job is to implement the best solution possible to the dining philosophers problem in Python by modifying the provided code **dining_philosopher.py**. Your solution must meet the following criteria:

1. Your script should create an arbitrary number of **locks**, called forks. You will need to store these forks in a list.
2. Your script should spawn an arbitrary number of philosopher threads.
3. Each philosopher N will try to acquire forks N and N+1, with the last philosopher trying to grab the last fork and the first fork.
4. Philosophers should not be able to start eating until she obtains both forks.
5. Each philosopher should be able to repeat the think-and-eat cycle independently.
6. Your program must ensure that deadlocks never occur.

Analysis of the dining philosopher problem

Each of the examples below contains a partial output of a student's solution to the dining philosopher problem. If the output is incorrect, mark the **first** message which should not be displayed in a correct solution. The first example has been answered already as follows:

Output A:

1. philosopher 0 is thinking...
2. philosopher 0 picks up left fork.
3. philosopher 1 is thinking...
4. philosopher 1 picks up left fork.
5. philosopher 1 picks up right fork. **(wrong)**
6. philosopher 1 is eating...
7. philosopher 1 puts down left fork
8. philosopher 1 puts down right fork

9. philosopher 0 picks up right fork.
10. …

Output B:

1. philosopher 0 is thinking…
2. philosopher 1 is thinking...
3. philosopher 0 picks up left fork.
4. philosopher 0 picks up right fork.
5. philosopher 0 is eating
6. philosopher 0 puts down left fork.
7. philosopher 1 picks up left fork.
8. philosopher 0 puts down right fork.
9. philosopher 1 picks up right fork.
10. philosopher 1 is eating...
11. …

Output C:

1. philosopher 0 is thinking…
2. philosopher 1 is thinking...
3. philosopher 1 picks up left fork.
4. philosopher 0 picks up left fork.
5. philosopher 0 puts down left fork.
6. philosopher 1 picks up right fork.
7. philosopher 1 is eating...
8. …

Output D:

1. philosopher 0 is thinking...
2. philosopher 0 picks up left fork.
3. philosopher 0 picks up right fork.
4. philosopher 1 is thinking...
5. philosopher 0 puts down left fork.
6. philosopher 0 puts down right fork.
7. philosopher 1 picks up left fork.
8. philosopher 1 picks up right fork.
9. philosopher 1 is eating...
10. …

Output E:

1. philosopher 0 is thinking...
2. philosopher 1 is thinking...
3. philosopher 1 picks up left fork.

4. philosopher 1 picks up right fork.
5. philosopher 1 is eating...
6. philosopher 1 puts down left fork.
7. philosopher 0 picks up left fork.
8. philosopher 1 puts down right fork.
9. philosopher 0 picks up right fork
10. philosopher 1 is thinking…
11. philosopher 0 is eating...
12. …

Output F:

1. philosopher 0 is thinking...
2. philosopher 0 picks up left fork.
3. philosopher 1 is thinking...
4. philosopher 1 picks up left fork
5. philosopher 0 puts down left fork.
6. philosopher 0 picks up left fork.
7. philosopher 0 puts down left fork.
8. philosopher 0 is thinking... (wrong- thinking twice in a row, he should try to pick up left fork)
9. philosopher 1 picks up right fork
10. philosopher 1 is eating...
11. …

There is a separate text file named **dining_philosopher.txt** provided for you to use. Write your answers on the text file and upload it to GitHub before the deadline.

Deliverables:

a. You must upload working code **dining_philsopher.py** and **fast_keysearch.py**, and **dining_philosopher.txt** under the Lab1 folder under the week1 GitHub repository by 7 PM, 9/1/2021.

b. You must get sign-off from one of the instructors or TAs during their office hours before submitting it. During the meeting, the instructor or TA will grade your lab based on the learning outcomes stated in the sign-off sheet attached.

Student Name:

Sign offs – Each signature is worth 1/N of your lab grade where N is the number of signatures

- The student could use the threading/multiprocessing module properly to speed up a CPU intensive task.

- The student could write a Python program that allows processes to exchange data.

- The student could distinguish correct and incorrect outputs of the dining philosopher problem and explain why some of the example outputs are incorrect.

- The student was able to write a thread safe program that solves the Dining Philosophers problem satisfying all the requirements listed in Part 2.