Question 1

An incubator generates another image for every embryo video after a random number of minutes - sometimes the interval is six minutes, sometimes it is 30 minutes, etc.
Each embryo produces an unknown number of images, and then it is marked as “completed”, meaning there aren’t going to be any more images for this embryo.
However, typically shortly after an embryo “completes”, new embryos are put in the incubator, and it start to produce new images.
Our goal is to write a program or a script that will build videos with their latest images, as closest as possible to when the incubator generates the new image, while maintaining the correct order of images.
The current implementation of the script (see below) is using a method that retrieves a single image. While this works, it is inefficient, and another method, “get_all_images” proves to be much more efficient when retrieving many images (50+).

Your task is to suggest edits and changes to this script (either by describing the changes or changing the code) that will bring it closer to the goal – keeping up with the latest images as close as possible to when they are generated.

```
class Incubator:
    def get_all_embryo_ids(self):
        """
        :return: returns a list of embryo_ids
        """
        pass

    def get_image(self, requested_embryo_id, requested_image_index):
        """
        returns the image of the reqeusted index for an embryo
        if the requested index does not exist, the value of image will
        be None/Null
        in addition, also returns if the embryo video has been completed
        (boolean value)
        :param requested_embryo_id: embryo_id to get the image of
        :param requested_image_index: index of the requested image
        :return: tuple(image -> bytes, completed -> boolean)
        """
        pass

    def get_all_images(self, requested_embryo_id):
        """
        returns a list of all the images for a requested embryo
        when requesting a large amount of images (50+) -
            this method is much more efficient than using "get_image" method
            or the same amount of images
        :param requested_embryo_id: embryo_id to get the images of
        :return: tuple(images -> list[bytes, bytes, bytes..], completed ->
        boolean)
        """
        pass

class VideoBuilder:
    def append_images(self, embryo_video, images_list):
        """
```

```
        appends the images in images_list to an existing video file
        :param embryo_video: embryo video file to append the images to
        :param images_list: the images that need to be appended
        :return: void
        """
        pass


# start script
inc = Incubator()
vid_builder = VideoBuilder()

embryo_stats = dict()
while True:
   # start iteration
   for embryo_id in inc.get_all_embryo_ids():

        if embryo_id not in embryo_stats.keys():
            # never-seen-before embryo; start getting images from image 0

            embryo_stats[embryo_id] = dict()
            request_index = 0
            image,completed = inc.get_image(embryo_id, request_index)

            while image is not None:
                embryo_stats[embryo_id]['last_request_time'] = datetime.now()
                embryo_stats[embryo_id]['last_index'] = request_index
                embryo_stats[embryo_id]['completed'] = completed
                vid_builder.append_images(embryo_id, [image])
                request_index+=1
                image,completed = inc.get_image(embryo_id, request_index)

        elif embryo_stats[embryo_id]['completed']:
            # embryo is already completed - pass
            pass

        elif not embryo_stats[embryo_id]['completed']:
            # embryo is in progress, and may have new images since last fetch
            request_index = embryo_stats[embryo_id]['last_index'] + 1

            image,completed = inc.get_image(embryo_id, request_index)

            while image is not None:
                embryo_stats[embryo_id]['last_request_time'] = datetime.now()
                embryo_stats[embryo_id]['last_index'] = request_index
                embryo_stats[embryo_id]['completed'] = completed
                vid_builder.append_image(embryo_id, [image])

                request_index+=1
                image,completed = inc.get_image(embryo_id, request_index)
    # finish iteration

# end script
```

Question 2

This question tests your experience with writing Python code.

The answer should be no more than 100 lines of code.
The code does not need to be compiled or running, semantics, writing patterns and efficiency is what will be tested.
We have a REST API with the following types of requests and responses.
You may assume a function with the name "call_api" exists, is always returning JSONs (=dictionaries), and is accepting the following variables:
call_api(endpoint='some api string', input='{"some":"dictionary"}')

1. login
expected input: JSON in the format of {'username': 'X', 'password': 'Y'}
output: {"access_token": "1000-digits-long-jibberish"}

2. look_for_patient
expected input: JSON in the format of {"access_token": "X", "patient_id": "Y"}
output: {"treatments": [{"date": "YYYY-MM-DD", "treatment_id": integer } ,
{"date": "YYYY-MM-DD", "treatment_id": another_integer },
and so on..]}

3. get_treatment_embryos
expected input: JSON in the format of {"access_token": "X", "treatment_id": Y}
output: {"embryos": [{"embryo_id": integer, "embryo_video": "/path/to/videoA"},
{"embryo_id": another_int, "embryo_video": "/path/to/videoB"},
{"embryo_id": another, "embryo_video": "/path/to/videoC"},
{"embryo_id": and_another, "embryo_video": "/path/to/videoD"}
]}

Write a piece of code using this API, to get all of the embryo-videos' paths of a certain patient for a specific treatment date, for example patient with id '12345' and his treatment at '2021-01-01'.
To clarify the question, here is an example for the first call of the API, which is logging in and receiving the access token:

```
### start code ###
username = "lior"
password = "liors password"
credentials = {"username": username, "password": password}
response = call_api(endpoint='login', input=credentials)
access_token = response['access_token']
### your code here ###
```

Question 3

This question tests your experience with data querying languages, preferably SQL.

The queries / code does not need to be compiled or running, semantics, writing patterns and efficiency is what will be tested.

For the given database structure, write queries that will answer the questions below.

- Every table has an "id" field that is unique, and a "pk" field which is a primary key and its' values are incrementing integers.
- Fields that have primary-key – foreign key relationships are marked in the same colors.
- Relationships are one:many – every patient can have many treatments, and every treatment will have many embryos.

Patient	
Column name	**Data type**
pk	Integer
id	String
name	string
date_of_birth	Date

Treatment	
Column name	**Data type**
pk	Integer
id	String
patient_fk	Integer
treatment_date	Date
treatment_type	String

Embryo	
Column name	**Data type**
pk	Integer
id	String
treatment_fk	Integer
number_of_images	Integer
first_image_time	Timestamp
last_image_time	Timestamp

a. Write a query that calculates the average time interval between each image.

b. Write a query that calculates each patient's age at the time of her treatment

c. Write a query that shows a distribution of average video length (the difference between last_image_time and first_image_time in "embryo" table) for each value in "treatment_type" field.

d. Write a query that shows a distribution of average video length for each age of a patient.

e. Someone made a mistake, and for whatever reason the "id" field in "embryo" table was not configured to be unique. As a result, there are now many instances of the same "id" values with different "pk" values.
Write a command that will remove all the duplicated rows of each ID and will leave only one row (it doesn't matter which row will be kept, as long as its' only one).