EXERCISE 1

```python
import pika

def correction(ch, method, properties, body):
    # doing nothing
    pass

def validation(ch, method, properties, body):
    pass

def confirmation(ch, method, properties, body):
    pass

# task 1
# 1- Using Pika's syntax, declare the necessary queues/exchanges and their connections as shown in Figure 1
if __name__ == '__main__':
    try:
        credentials = pika.PlainCredentials("guest", "guest")
        connection = pika.BlockingConnection(pika.ConnectionParameters("rabbitmq", 5672, "/", credentials))
        channel = connection.channel()

        # Declare the exchange
        channel.exchange_declare(exchange='Assignment', exchange_type='direct')

        # Declare the queues
        channel.queue_declare(queue="Correction queue")
        channel.queue_declare(queue="Validation queue")
        channel.queue_declare(queue="Result queue")

        # Bind the queues to the exchange with routing keys
        channel.queue_bind(exchange='Assignment', queue='Correction queue', routing_key='correction')
        channel.queue_bind(exchange='Assignment', queue='Validation queue', routing_key='validation')
        channel.queue_bind(exchange='Assignment', queue='Result queue', routing_key='confirmation')

        '''task2
        2-demonstrator subscribing to correction queue'''
        channel.basic_consume(queue='Correction queue', on_message_callback=correction, auto_ack=False)
        #TAs can listen to validation queue
        channel.basic_consume(queue='Validation queue', on_message_callback=validation, auto_ack=False)
        #Module coordinator can listen to result queue for confirmation
        channel.basic_consume(queue='Result queue', on_message_callback=confirmation, auto_ack=False)
        print(' [*] Waiting for messages. To exit press CTRL+C')
    except KeyboardInterrupt:
        print("Exiting")
```

The above code snippet is for the first exercise. This establishes a connection channel to the rabbitmq container.
Important snippets are marked . They are

- The snippets marked in yellow that come first are for declaring the exchange and queues. The exchange type is set as direct which means messages from the exchange are sent to the queues based on a matching routing key.
- Snippets marked red are for binding the queues to the exchange to ensure that they can communicate with each other.
- The third phase is where the python script starts listening to the queues for potential messages and puts in place callbacks to be invoked when messages arrive.

To run ex1

docker-compose up –build.

EXERCISE 2

In this exercise we add new python scripts to handle callbacks.

- Student

```
def publish():
    assignment= json.dumps({
        'id': 1,
        'title': "testing",
        'status': "submitted"  # Assuming Status is an Enum
    })
    channel.basic_publish(exchange="Assignment",routing_key='',body=assignment)

publish()
```

In student the important piece of code is publishing to the assignment exchange. The routing key is set to empty, so that all queues can consume the message.

- task1_2.py

```
def correction(ch, method, properties, body):
    demonstrator.correction(body)
    ch.basic_ack(delivery_tag=method.delivery_tag)

def validation(ch, method, properties, body):
    teaching_assistant.validate(body)
    ch.basic_ack(delivery_tag=method.delivery_tag)

def confirmation(ch, method, properties, body):
    module_coordinator.confirmation(body)
    ch.basic_ack(delivery_tag=method.delivery_tag)

# task 1
# 1- Using Pika's syntax, declare the necessary queues/exchanges and their connections as shown in Figure 1
if __name__ == '__main__':
    channel = None
    connection = None
    credentials = None

    try:
        credentials = pika.PlainCredentials("guest", "guest")
        connection = pika.BlockingConnection(pika.ConnectionParameters("rabbitmq", 5672, "/", credentials))
        channel = connection.channel()

        # Declare the exchange
        channel.exchange_declare(exchange='Assignment', exchange_type='fanout')

        # Declare the queues
        channel.queue_declare(queue="Correction queue")
        channel.queue_declare(queue="Validation queue")
        channel.queue_declare(queue="Result queue")

        # Bind the queues to the exchange with routing keys
        channel.queue_bind(exchange='Assignment', queue='Correction queue')
        channel.queue_bind(exchange='Assignment', queue='Validation queue')
        channel.queue_bind(exchange='Assignment', queue='Result queue')

        '''task2
        2-demonstrator subscribing to correction queue'''
        channel.basic_consume(queue='Correction queue', on_message_callback=correction, auto_ack=False)
        # TAs can listen to validation queue
        channel.basic_consume(queue='Validation queue', on_message_callback=validation, auto_ack=False)
        # Module coordinator can listen to result queue for confirmation
        channel.basic_consume(queue='Result queue', on_message_callback=confirmation, auto_ack=False)
```

- Assignment exchange is declared with exchange type as fanout. This is so that it can broadcast messages to all listening waiting queues.
- Green snippet marked indicates now that none of the queues have a routing key.
- Callbacks marked in red have message acknowledgment code which serves as receipt for rabbitmq that intended message was consumed.

- Teaching_assistant.py

```
def validate(assignment):
    try:
        #print(assignment)
        credentials = pika.PlainCredentials("guest","guest")
        connection = pika.BlockingConnection(
            pika.ConnectionParameters("rabbitmq",5672, "/",credentials))
        channel=connection.channel()

        message_body_string = assignment.decode('utf-8')
        assignment_data = json.loads(message_body_string)
        if assignment_data['status']!='corrected':
            #print("Skip processing by TA")
            return
        assignment_data['status']='validated'
        print("TA changing status for user "+
            str(assignment_data['id'])+" to "+assignment_data['status'])
        print(assignment_data)
        channel.basic_publish(exchange="Assignment",routing_key='',
                              body=json.dumps(assignment_data))
    except Exception as e:
        print(e)
```

Each consumer(ta or module coordinator or demonstrator) has to have a new check in place to ensure that it processes only the relevant message. With fanout exchange all the consumers receive all messages, including those not intended for them. This needs to be handled at the application level.

To run the application

- docker-compose up —build
- Once the message is up and running, do python student.py

EXERCISE 3

- Student

```python
import pika
import json


credentials = pika.PlainCredentials("guest","guest")
connection = pika.BlockingConnection(
    pika.ConnectionParameters("localhost",5672, "/",credentials))
channel=connection.channel()


def publish():
    assignment= json.dumps({
        'id': 1,
        'title': "CC",
        'status': "submitted"  # Assuming Status is an Enum
    })
    channel.basic_publish(exchange="Assignment",routing_key='CC',body=assignment)
    assignment= json.dumps({
        'id': 2,
        'title': "DM",
        'status': "submitted"  # Assuming Status is an Enum
    })
    channel.basic_publish(exchange="Assignment",routing_key='DM',body=assignment)

publish()
```

The student now sends multiple messages. Each with a different routing key. The title in the message body has also been set accordingly to understand the flow of messages from logs.

- Initializer

  A single correction queue is now replaced with dedicated queues for Cloud computing and data mining modules.
  The assignment exchange's communication mode changes to a direct type. This means that it no longer broadcasts messages, and directs messages based on routing key.

```python
# task 1
# 1- Using Pika's syntax, declare the necessary queues/exchanges and their connections as shown in Figure 1
if __name__ == '__main__':
    channel = None
    connection = None
    credentials = None

    try:
        credentials = pika.PlainCredentials("guest", "guest")
        connection = pika.BlockingConnection(pika.ConnectionParameters("rabbitmq", 5672, "/", credentials))
        channel = connection.channel()

        # Declare the exchange
        channel.exchange_declare(exchange='Assignment', exchange_type='direct')

        # Declare the queues
        channel.queue_declare(queue="Cloud Computing queue")
        channel.queue_declare(queue="Data Mining queue")
        channel.queue_declare(queue="Validation queue")
        channel.queue_declare(queue="Result queue")

        # Bind the queues to the exchange with routing keys
        channel.queue_bind(exchange='Assignment', queue='Cloud Computing queue', routing_key="CC")
        channel.queue_bind(exchange='Assignment', queue='Data Mining queue', routing_key="DM")
        channel.queue_bind(exchange='Assignment', queue='Validation queue', routing_key="validation")
        channel.queue_bind(exchange='Assignment', queue='Result queue',routing_key="confirmation")

        #callback for cloud computing queue
        channel.basic_consume(queue='Cloud Computing queue', on_message_callback=cloud_computing, auto_ack=False)
        #callback for cloud computing queue
        channel.basic_consume(queue='Data Mining queue', on_message_callback=data_mining, auto_ack=False)
        # TAs can listen to validation queue
        channel.basic_consume(queue='Validation queue', on_message_callback=validation, auto_ack=False)
        # Module coordinator can listen to result queue for confirmation
        channel.basic_consume(queue='Result queue', on_message_callback=confirmation, auto_ack=False)

        print(' [*] Waiting for messages. To exit press CTRL+C')
        channel.start_consuming()
        connection.close()

    except Exception as e:
        print (e)
```

- Teaching_assistant.py
  An example of a consumer can be explained using teaching_assistant.
  The consumer no longer has to filter out messages on its end. With the routing key in place, each consumer only receives messages that it needs.
  Post changing the status each consumer publishes back to the assignment exchange with the relevant routing key.

```python
def validate(assignment):
    try:
        credentials = pika.PlainCredentials("guest","guest")
        connection = pika.BlockingConnection(
            pika.ConnectionParameters("rabbitmq",5672, "/",credentials))
        channel=connection.channel()

        message_body_string = assignment.decode('utf-8')
        assignment_data = json.loads(message_body_string)
        assignment_data['status']='validated'
        print("TA changing status for user "+
            str(assignment_data['id'])+" to "+assignment_data['status'])
        print(assignment_data)
        channel.basic_publish(exchange="Assignment",routing_key='confirmation',
                              body=json.dumps(assignment_data))
    except Exception as e:
        print(e)
```

To run the application

- docker-compose up –build
- Python student.py

EXERCISE 4

The changes in this exercise are to the docker related files, in an attempt to dockerise all components.

- Dockerfile

```
FROM python:3

WORKDIR /usr/src/app

COPY requirements.txt ./
RUN pip install --no-cache-dir -r requirements.txt
ENV PYTHONUNBUFFERED=1
```

The dockerfile has been generalized to be used for all components.

- Docker-compose

For each component ,its respective python file is mounted on to the container. There are containers for student, cloud and data mining, teaching assistant and module coordinator.
The containers depend on the rabbit mq container. The consumers(services apart from student) have been configured to restart if they crash.

```yaml
student:
  image: python:3
  build:
    context: .
    dockerfile: Dockerfile
  environment:
    - PYTHONUNBUFFERED=1
  command: sh -c "sleep 10 && python -u /usr/src/app/student.py 2>&1"
  depends_on:
    - rabbitmq
  networks:
    - rabbitmq_go_net
  tty: true
  #restart: unless-stopped
  volumes:
    - ./student.py:/usr/src/app/student.py
demonstrator_cc:
  image: python:3
  build:
    context: .
    dockerfile: Dockerfile
  restart: unless-stopped
  command: sh -c "sleep 5 && python -u /usr/src/app/demonstrator_cc.py 2>&1"
  depends_on:
    - rabbitmq
  networks:
    - rabbitmq_go_net
  tty: true
  volumes:
    - ./demonstrator_cc.py:/usr/src/app/demonstrator_cc.py
demonstrator_dm:
  image: python:3
  build:
    context: .
    dockerfile: Dockerfile
  restart: unless-stopped
  command: sh -c "sleep 5 && python -u /usr/src/app/demonstrator_dm.py 2>&1"
  depends_on:
    - rabbitmq
  networks:
    - rabbitmq_go_net
  tty: true
```

Steps to run ex4

- docker-compose up rabbitmq ta demonstrator_cc demonstrator_dm module_coordinator
- python student.py