Exam

You have passed the quiz!

Your score:
65 of 75 Correct (86%)

Elapsed time:
51 minutes

75 of 75 questions answered

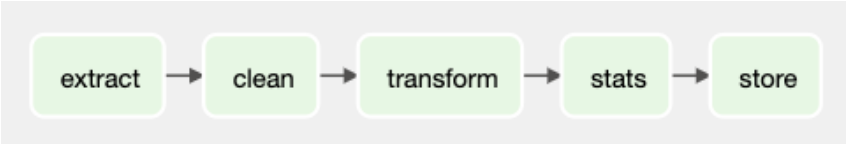
Hide Answers

Question 1: ✔ Correct answer
Once your DAG Run has been completed successfully, you would like to receive a Slack notification.

How can you do that?

- [x] **By using on_success_callback**
- [ ] ~~By implementing the SlackWebHookOperator as the last task to execute of your DAG~~
- [ ] ~~By using on_completion_callback~~

Question 2: ✔ Correct answer
Examine the following DAG:



If `wait_for_downstream` is set to `True` for the task `extract`, what happens?

- [ ] ~~The next DAG Run will run once all the tasks in the current DAG Run succeed~~
- [x] **The task extract runs in the next DAG Run once extract and clean succeed in the current DAG Run**
- [ ] ~~The task store will run in the next DAG Run once it succeeds in the current DAG Run~~

Question 3: ✖ Incorrect answer

You only have 2 DAGs

DAG A is in charge of extracting data daily.

DAG B is in charge of processing the extracted data daily.

DAG B depends on DAG A with the ExternalTaskSensor.

Your teammate says that as only DAG B is waiting for DAG A, you could merge those two DAGs in one for simplicity. Do you agree?

- [ ] **Yes**
- [x] ~~No~~

Question 4: ✔ Correct answer
Assume there are tasks in an Airflow instance that are reused across different DAGs. What are the best ways to deal with this? (select all that apply)

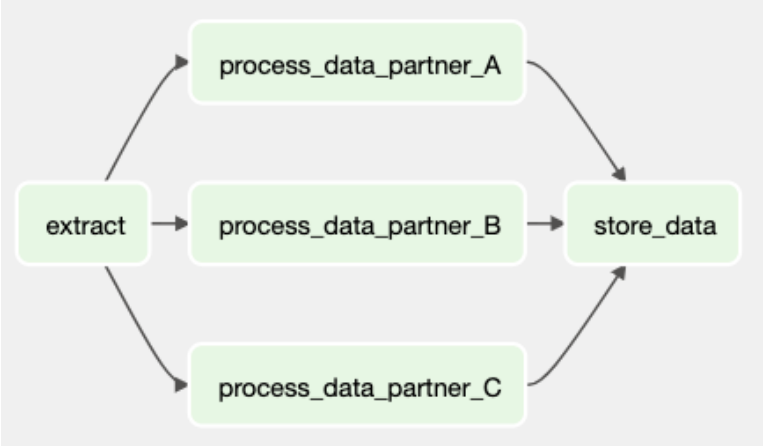
- [x] **Create one DAG with the common tasks and wait for its completion in the other DAGs (DAG dependency)**
- [x] **Create a python file with the common tasks and import them in the different DAGs**
- [ ] ~~Put the common tasks in each DAG~~

Question 5: ✔ Correct answer
You know that XComs are limited in size according to the database used. Therefore, you decided to configure an XCom Backend with S3.

Does this mean that the best practices for exchanging data between your tasks with XComs no longer apply? (e.g., size limitations)

- [ ] ~~Yes~~
- [x] **No**

Question 6: ✔ Correct answer
Examine the DAG below:



Once the task `extract` gets completed, we want to execute: `process_data_partner_B` first, then `process_data_partner_A` , and finally `process_data_partner_C` .

To do that, we defined the following properties for each task:

process_data_partner_B

- *pool*: "partner_B"
- *priority_weight*: 3

process_data_partner_A

- *pool*: "partner_A"
- *priority_weight*: 2

process_data_partner_C

- *pool*: "partner_C"
- *priority_weight*: 1

Will the execution order be respected?

- [ ] ~~Yes~~
- [x] **No**

Question 7: ✔ Correct answer
What kind of operator should be used if a DAG needs to wait for a condition to be met before executing the next task?

- [ ] ~~Action Operators~~
- [ ] ~~Transfer Operators~~
- [x] **Sensors**

Question 8: ✔ Correct answer
10 of your DAGs do similar things, with only a few parameters changing between them.

What could you do to simplify your code?

- [ ] ~~Nothing, we have to create manually each DAG even for slight differences between them~~
- [x] **Generate DAGs dynamically**

Question 9: ✔ Correct answer
Given the environment variable **`AIRFLOW_VAR_MY_API=mysecretapi`** , and the following DAG:

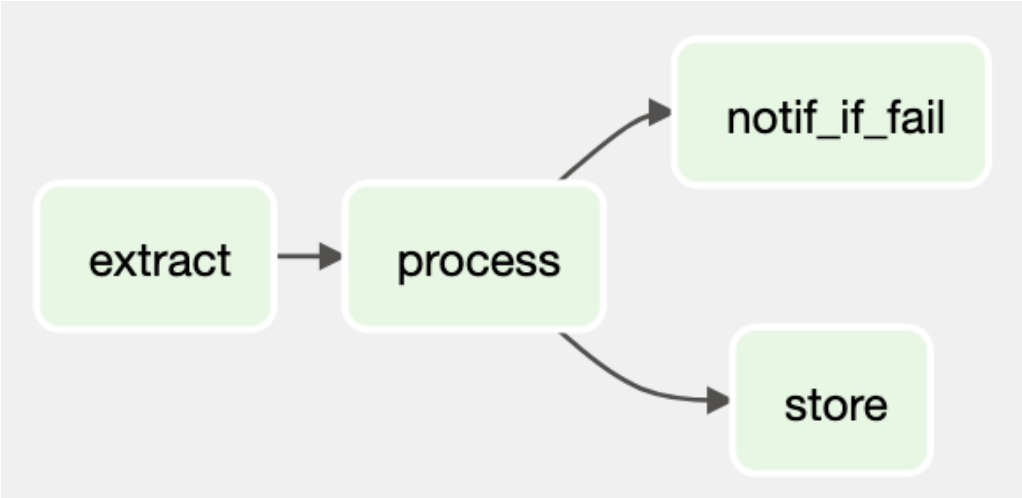
```
with DAG("car", start_date=datetime(2023, 1 ,1), schedule="@daily", catchup=False):

    process = PythonOperator(
        task_id="process",
        python_callable=_process,
        op_args=[Variable.get("my_api")]
    )
```

Does it request the database to fetch the variable **`my_api`** ?

- [ ] ~~Yes~~

- [x] **No**
- [ ] ~~You can't define Airflow variables with environment variables~~
- [ ] ~~You can't fetch an Airflow variable with Variable.get~~

Question 10: ✔ Correct answer
Examine the following DAG:



If the task **`process`** fails, the DAG should execute **`notif_if_fail`** . If it succeeds, it should execute **`store`** .

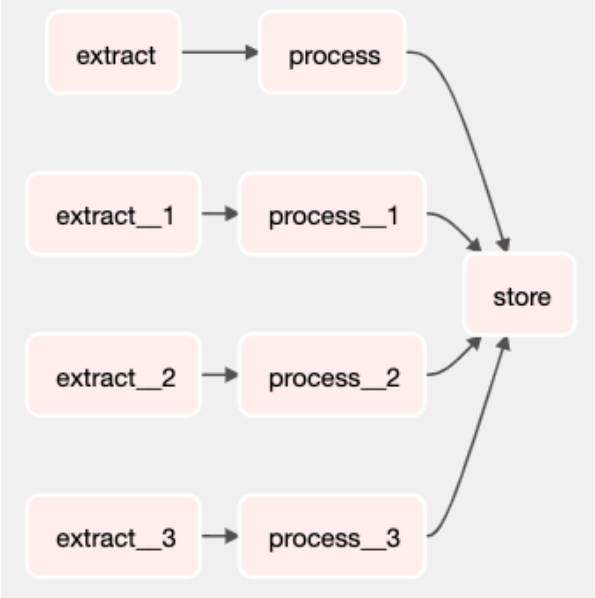
Which trigger rule should be applied to **`notif_if_fail`** ?

- [ ] ~~all_done~~
- [x] **all_failed**
- [ ] ~~none_failed~~

Question 11: ✔ Correct answer
What is the purpose of the ***`none_failed_or_skipped`*** trigger rule?

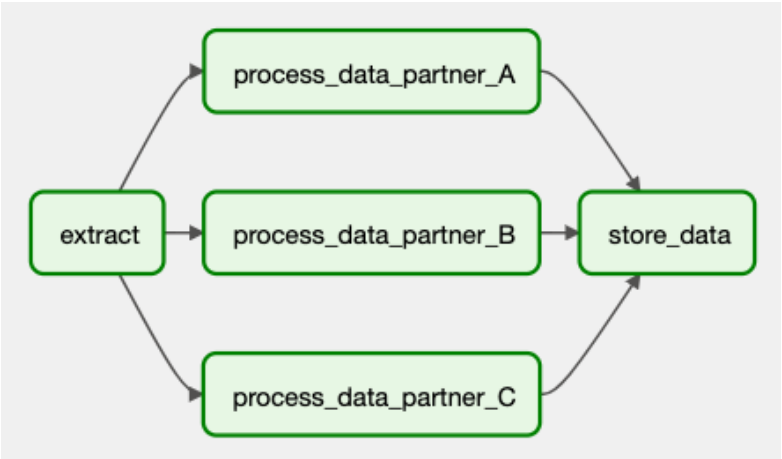
- [x] **To avoid getting skipped the task that depends on the tasks selected by the BranchPythonOperator**
- [ ] ~~To trigger your task once one upstream task gets skipped~~

Question 12: ✔ Correct answer
Can you limit the concurrency of the "process" tasks so that they are executed sequentially while other task are executed simultaneously?



- [x] **Yes, by creating a pool with 1 slot**
- [ ] ~~No, because pools are set at the DAG level~~
- [ ] ~~Yes, by changing parallelism to 1~~

Question 13: ✔ Correct answer
You would like to group process_data_partner tasks together.



What is the easiest and best way to do that?

- [ ] ~~Create a SubDag~~
- [x] **Create a TaskGroup**

Question 14: ✔ Correct answer
What does it mean when a DAG's `schedule_interval` parameter is set to a value of `None`?

- [ ] ~~It's impossible to set the schedule_interval to None~~
- [x] **The DAG must be triggered manually**
- [ ] ~~The DAG will be triggered once and only once~~

Question 15: ✔ Correct answer
Assume a DAG has:

- A start date set to the 1st of January 2021
- A schedule interval set to @daily
- An end date set to the 5th of January 2021

How many DAG Runs will occur when this DAG is triggered?

- [ ] ~~2~~
- [ ] ~~3~~
- [ ] ~~4~~
- [x] **5**

Question 16: ✔ Correct answer
By default, XComs are serialized with...

- [x] **Json**
- [ ] ~~Pickle~~

Question 17: ✔ Correct answer
Can you have a TaskGroup within a TaskGroup?

- [x] **Yes**
- [ ] ~~No~~

Question 18: ✔ Correct answer
Which implementation results in the least amount of requests to the metadata database?

A)

```
def _process(ti):
    for task_id in ['task_a', 'task_b', 'task_c']:
        ti.xcom_pull(key='filename', task_ids=task_id)
```

B)

```
def _process(ti):
    ti.xcom_pull(key='filename', task_ids=['task_a', 'task_b', 'task_c'])
```

- [ ] ~~A~~
- [x] **B**

Question 19: ✔ Correct answer
Examine the DAG below:

```
job_name = Variable.get("job_name")
job_prefix = Variable.get("job_prefix")

def _process():
    output = launch_spark_job(job_name)
    return f"{job_prefix}_{output}"

with DAG("jobs", schedule="@daily", catchup=False):

    process = PythonOperator(
        task_id="process",
        python_callable=_process
    )
```

Why fetching variables outside of tasks is not a best practice?

- [ ] ~~There is nothing wrong with fetching variables outside of tasks~~
- [ ] ~~Variables shouldn't be used to store values~~
- [ ] ~~We must fetch variables under the DAG object, not in tasks~~
- [x] **That makes a request to the database every time the DAG is parsed**

Question 20: ✔ Correct answer
What are the different ways of creating DAG dependencies? (Select all that apply)

- [x] **ExternalTaskSensor**
- [ ] ~~BranchPythonOperator~~
- [x] **TriggerDagRunOperator**
- [x] **SubDAGs**

Question 21: ✔ Correct answer
How would you know which arguments of an Operator are templated?

- [x] **By looking at the attribute template_fields**
- [ ] ~~By looking at the attribute template_ext~~
- [ ] ~~By testing each argument~~

Question 22: ✔ Correct answer
With the PythonOperator, what is the most efficient way to push multiple XComs at once?

A)

```
@task
def extract():
    return {"path": "/tmp/data", "filename": "data.csv"}
```

B)

```python
@task
def extract() -> Dict[str, str]:
    return {"path": "/tmp/data", "filename": "data.csv"}
```

C)

```python
def _extract(ti):
    ti.xcom_push(key='path', value='/tmp/data')
    ti.xcom_push(key='filename', value='data.csv')
```

- ☐ ~~A~~
- ☑ **B**
- ☐ ~~C~~

Question 23: ✔ Correct answer
Does the DAG below work (assuming that the rest of the code works)?

```python
@dag(start_date=datetime(2021, 1, 1), catchup=False)
def my_etl():

    @task(task_id='a')
    def pre_process():
        print("pre_process")

    @task_group(group_id='process')
    def process_group():
        @task(task_id='a')
        def process_a():
            print('process a')

        @task(task_id='b')
        def process_b():
            print('process b')

        @task(task_id='c')
        def process_c():
            print('process c')
```

- ☑ **Yes**
- ☐ ~~No~~

Question 24: ✔ Correct answer

Between 9 AM and 10 AM, a file is expected to land in a specific folder. Assume a DAG is using a Sensor to determine if the file exists every 10 minutes. Which mode should the sensor be using?

- ~~poke~~
- **reschedule**

Question 25: ✔ Correct answer
If an Airflow instance has a lot of similar DAGs, they can be generated dynamically.

One method of doing this is to execute a script **(not a DAG)** that will generate DAGs based on a template and different inputs. Therefore, DAGs are created dynamically with one file and one DAG, and NOT with one file and multiple DAGs.

What are the benefits of this method? (Select all that apply)

- **Scalable as DAGs are not generated each time the folder dags/ is parsed by the Scheduler**
- ~~There is no need to run the script to generate DAGs, it's done by Airflow~~
- **Full visibility of the DAG code (one DAG -> one file)**
- **Full control over the way DAGs are generated (script)**

Question 26: ✔ Correct answer
Give the code snippets below:

A)

```
process = PythonOperator(
    task_id="process",
    python_callable=_process,
    op_args=["{{ var.json.job_name }}"]
)
```

B)

```
process = PythonOperator(
    task_id="process",
    python_callable=_process,
    op_args=[Variable.get("job_name")]
)
```

What is the main difference between A and B in fetching a variable with the PythonOperator?

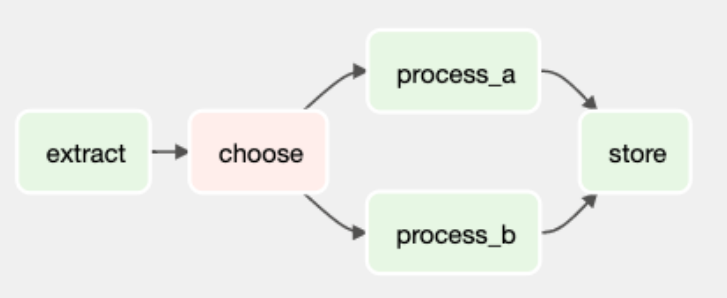
- ~~There is no difference, both ways will make a connection to the DB each time the DAG is parsed~~
- ~~Only A will make a connection to the DB each time the DAG is parsed~~
- **Only B will make a connection to the DB each time the DAG is parsed**
- ~~B doesn't work~~

Question 27: ✔ Correct answer
By default, the argument `parameters` of the PostgresOperator is not templated.

Is there anything you can do so that `parameters` becomes a templated argument?

- [ ] ~~No~~
- [x] **Yes**

Question 28: ✖ Incorrect answer
In the DAG below, the task named *choose* is a **`BranchPythonOperator`** that executes either task *process_a* or *task process_b*.



If the DAG is triggered, what will be the status of the task *store*?

- [x] ~~Success~~
- [ ] **Skipped**
- [ ] ~~Failed~~
- [ ] ~~None~~

Question 29: ✔ Correct answer
Examine the task below:

```
trigger_cleaning_xcoms = TriggerDagRunOperator(
    task_id='trigger_cleaning_xcoms',
    trigger_dag_id='cleaning_dag',
    execution_date='{{ ds }}',
    wait_for_completion=True,
    poke_interval=60,
    reset_dag_run=False,
    failed_states=['failed']
)
```

Can this task be run twice on the same execution date?

- [ ] ~~Yes~~
- [x] **No**

Question 30: ✔ Correct answer
Is there any difference between `schedule_interval="*/10 * * * *"` and `schedule_interval=timedelta(minutes=10)`?

- [ ] ~~No, they are both identical~~
- [x] **A Cron expression is stateless, whereas a timedelta is relative**

- [ ] ~~A schedule_interval parameter can't be defined with a timedelta object~~

Question 31: ✔ Correct answer
How does Airflow detect that a python file is a DAG?

- [x] **It looks for "dag" and "airflow" in python files**
- [ ] ~~It looks for python files only~~
- [ ] ~~It looks for DAGs in all files~~

Question 32: ✔ Correct answer
Is there a way to limit the number of tasks running at the same time for a specific DAG?

- [x] **Yes, with the argument concurrency**
- [ ] ~~Yes, by modifying dag_concurrency in airflow.cfg~~
- [ ] ~~No, we can't~~

Question 33: ✖ Incorrect answer
Examine the DAG below:

```
job_prefix = Variable.get("job_prefix")

def _process():
    job_name = Variable.get("job_name")
    output = launch_spark_job(job_name)
    return f"{job_prefix}_{job_name}"

with DAG("car", schedule="@daily", catchup=False):

    process = PythonOperator(
        task_id="process",
        python_callable=_process
    )
```

How many requests will be sent to the database each time the DAG is parsed? (without counting the request for the parsing process)

- [x] ~~2~~
- [ ] **1**
- [ ] ~~3~~
- [ ] ~~0~~

Question 34: ✔ Correct answer
Examine the following DAG Operator code:

```
fetch_data = PythonOperator(
    task_id='fetch_data',
    python_callable=_fetch
    dag=dag
)

process_data = PythonOperator(
    task_id='process_data',
    python_callable=_fetch
    dag=dag
)

store_data = PythonOperator(
    task_id='store_data',
    python_callable=_fetch
    dag=dag
)
```

What ways would remove the redundant **`dag=dag`** assignment from the tasks? (select all that apply)

- [x] **By declaring a DAG with the @dag decorator**
- [x] **By declaring a DAG with a context manager**
- [ ] ~~We can't, we have to explicitly assign the dag object to each task~~

Question 35: ✔ Correct answer
You have a DAG running (version 1) with three tasks:

- Task A is completed
- Task B and C are queued

In the meantime, you deploy a new version of the DAG (version 2) with a code change to tasks B and C.

Which version will the queued tasks be based on?

- [ ] ~~DAG version 1~~
- [x] **DAG version 2**

Question 36: ✔ Correct answer
To avoid having a lot of DAG Runs as soon as you schedule your DAGs, you set the **`catchup`** parameter to the value of **`False`** .

Can you still backfill the DAG?

- [x] **Yes, through the CLI for example.**
- [ ] ~~No. Once the catchup is disabled, you can't run non-triggered DAG Runs.~~

Question 37: ✔ Correct answer
An Airflow user wants to trigger DAG A from DAG B.

Which operator should be used to accomplish this?

- [x] **TriggerDagRunOperator**
- [ ] ~~ExternalTaskSensor~~

Question 38: ✔ Correct answer
You want to share data between your tasks, and your Airflow instance runs with a MySQL database.

Can you share 3 Gigabytes of data between 2 tasks using an XCom?

- [ ] ~~Yes~~
- [x] **No**

Question 39: ✔ Correct answer
One of your tasks sends requests to an API. If for some reasons the API is not available, then you want to retry your task.

Which parameter can you define to make the retry process more dynamic and avoid overloading the API?

- [ ] ~~retry_delay~~
- [ ] ~~retries~~
- [x] **retry_exponential_backoff**

Question 40: ✔ Correct answer
Examine the following DAG:

```
14 with DAG('car_dag', start_date=datetime(2021, 1, 1), schedule_interval='@daily',
15     default_args=default_args, catchup=False) as dag:
16
17     process = PythonOperator(
18         task_id="process",
19         python_callable=_process,
20         op_args=Variable.get("my_api")
21     )
```

When will the **3rd DAG run** be triggered by Scheduler?

- [ ] ~~2021/01/02 00:00~~
- [ ] ~~2021/01/03 00:00~~
- [x] **2021/01/04 00:00**
- [ ] ~~2021/01/05 00:00~~

Question 41: ✔ Correct answer
Is it possible to wait for the DAG triggered by the TriggerDagRunOperator to complete before executing the next task?

- [ ] ~~No~~

- [x] **Yes**

Question 42: ✔ Correct answer
What mechanism notifies an Airflow user if a task is taking longer than expected to complete?

- [x] **sla**
- [ ] ~~timeout~~
- [ ] ~~execution_timeout~~

Question 43: ✔ Correct answer
You made a DAG for which DAG Runs are dependent on each other. The output of the previous DAG Run is used as input for the next DAG Run.

How can you limit the number of DAG Runs for this specific DAG to 1?

- [ ] ~~In airflow.cfg, define max_active_runs_per_dag=1~~
- [x] **In the DAG object, define max_active_runs =1**
- [ ] ~~In airflow.cfg, define parallelism=1~~

Question 44: ✔ Correct answer
Examine the tasks below:

```
# The function
def _load_data_from_today():
    data = get_data_from_bucket(datetime.now())
    add_data_to_db(data)

# In the DAG
load_data_from_today = PythonOperator(
    task_id="load_data_from_today",
    python_callable=_load_data_from_today
)
```

Assuming that **`get_data_from_bucket`** and **`add_data_to_db`** work, is there any issue with this code?

- [ ] ~~Yes, the Python function _load_data_from_today() expects a parameter~~
- [ ] ~~No, everything works~~
- [ ] ~~Yes, data isn't given from the PythonOperator task~~
- [x] **Yes, datetime.now()**

Question 45: ✔ Correct answer
What happens if you return nothing from the python function called by the BranchPythonOperator?

- [ ] ~~All the next tasks are skipped~~
- [ ] ~~The next task gets randomly executed~~
- [x] **You get an error**

Question 46: ✔ Correct answer
Which operators allow a DAG to choose the next task to execute according to a condition? (select all that apply)

- [x] **BranchSQLOperator**
- [x] **BranchPythonOperator**
- [ ] ~~BashOperator~~

Question 47: ✔ Correct answer
In Airflow, it's possible to have a single Python file that generates DAGs dynamically based on some input parameters. This avoids having a lot of similar DAGs and duplicate code.

However, this method has some drawbacks. What are they? (Select all that apply)

- [x] **DAGs are generated every time the Scheduler parses the DAG folder**
- [x] **There is no way to see the code of the generated DAG in the UI**
- [x] **If you change the number of generated DAGs, you might have DAGs that don't exist anymore in the UI**
- [ ] ~~You need at least two Schedulers to generate your DAGs~~

Question 48: ✔ Correct answer
Would the following DAG dependency relationship work?

```
[t1, t2, t3] >> [t4, t5, t6]
```

- [ ] ~~Yes~~
- [x] **No**

Question 49: ✔ Correct answer
Can you choose to execute more than one task with the BranchPythonOperator?

- [x] **Yes**
- [ ] ~~No, you can return only one task_id~~

Question 50: ✖ Incorrect answer
Different teams are working on the same Airflow instance. Some DAGs belong to the Data Science team, others to the Data Engineering team.

What could you do to make DAGs easier to manage? (select all that apply)

- [x] **By defining tags**
- [x] **By defining owners**
- [x] **By defining permission roles**
- [ ] **By defining pools**

Question 51: ✔ Correct answer
Is it possible to define a task group outside of your DAG and then import it for use?

- [x] **Yes**
- [ ] ~~No, the TaskGroup must be defined within the DAG file~~

Question 52: ✔ Correct answer
Does the following DAG work?

```python
@dag(start_date=datetime(2021, 1, 1), catchup=False)
def my_etl():

    @task
    def extract() -> Dict[str, str]:
        return {"path": "/tmp/data", "filename": "data.csv"}

    process = BashOperator(
        task_id="process",
        bash_command="echo '{{ ti.xcom_pull(key='path', task_ids=['extract']) }}'"
    )

    extract() >> process
```

- [x] **Yes**
- [ ] ~~No~~

Question 53: ✔ Correct answer
You have a task that you want to run only if the previous instance of that task succeeded.

How can you do that?

- [ ] ~~It's not possible to create task dependencies between DAG Runs~~
- [x] **By setting depends_on_past to true on that task**
- [ ] ~~By setting concurrency=1 to the DAG object~~

Question 54: ✔ Correct answer
Examine the DAG below:

```python
def _process():
    job_settings = ?
    output = launch_spark_job(job_settings["job_name"])
    return f"{job_settings['job_prefix']}_{output}"

with DAG("car", schedule="@daily", catchup=False):

    process = PythonOperator(
        task_id="process",
        python_callable=_process
    )
```

What is the missing value for **`job_settings`** in **`_process()`** ?

- [ ] ~~job_settings = Variable.get("job_settings")~~
- [x] **job_settings = Variable.get("job_settings", deserialized_json=True)**
- [ ] ~~job_settings = "{{ var.json.job_settings }}"~~
- [ ] ~~You can't use JSON with variables~~

Question 55: ✔ Correct answer
Is there a view on the Airflow UI (version >= 2.1) where DAG dependencies can be observed?

- [x] **Yes**
- [ ] ~~No~~

Question 56: ✖ Incorrect answer
Examine the following DAG code:

```
@task
def extract() -> Dict[str, str]:
    return {"path": "/tmp/data", "filename": "data.csv"}

@task
def process(path):
    print(path)

@task
def store():
    print('store')

tasks = []
for i in range(4):
    tasks.append(process(extract()['path']))

tasks >> store()
```

How many `extract`, `process` and `store` tasks do you get from this DAG, respectively?

- [ ] **4,4,1**
- [x] ~~1,4,1~~
- [ ] ~~1,1,1~~
- [ ] ~~No tasks at all. It doesn't work.~~

Question 57: ✔ Correct answer
You've created a SubDAG in order to group some tasks in your DAG. Now, you want to limit the concurrency of the tasks in that SubDAG to 1 with a pool having 1 slot.

Where should you assign the pool?

- [ ] ~~To the SubDagOperator~~
- [x] **To the tasks in the SubDag**
- [ ] ~~We can't do that with a pool~~

Question 58: ✔ Correct answer

What is a pool?

- [x] **A pool contains a number of worker slots that get taken by running tasks to limit execution parallelism**
- [ ] ~~A pool is the best place to go in hot weather~~
- [ ] ~~A pool contains a number of worker slots occupied by DAG Runs, which limits DAG concurrency~~

Question 59: ✖ Incorrect answer
You have a DAG with a start_date set to 1st of February 2021. You run it until March, and then you change the start_date to the 1st of January 2021.

What will happen?

- [ ] **Nothing, the DAG Runs between the 1st of January and the 1st of February won't get triggered automatically.**
- [x] ~~The DAG Runs between the 1st of January and the 1st of February will get triggered automatically.~~
- [ ] ~~Your DAG will stop getting triggered.~~

Question 60: ✔ Correct answer
Examine the DAG below:

```
with DAG("car", schedule="@daily", catchup=False):

    create_table = PostgresOperator(
        task_id="create_table",
        postgres_conn_id="postgres",
        sql="""
                CREATE TABLE cars(
                    serial_number BIGINT PRIMARY KEY,
                    maker TEXT NOT NULL,
                    nb_doors INTEGER NOT NULL,
                    color INTEGER NOT NULL
                );
        """
    )
```

What happens if you run this DAG twice?

- [ ] ~~The task creates two tables "cars"~~
- [x] **The task throws an error that the able "cars" already exists**
- [ ] ~~The task doesn't work as a parameter is missing~~
- [ ] ~~You can't use SQL requests like that, they must be in files~~

Question 61: ✔ Correct answer
What is the difference between the **`execution_timeout`** from the **`BaseOperator`** and **`timeout`** from the **`BaseSensorOperator`** ?

- [ ] ~~There is no difference, timeout overwrites the value of execution_timeout when defined~~
- [x] **With timeout and soft_fail set to True, the Sensor will be marked as SKIPPED, whereas with execution_timeout it will be marked as FAILED**

Question 62: ✔ Correct answer

In Airflow, a SLA is relative to...

- [x] **The DAG execution date**
- [ ] ~~The task start time~~

Question 63: ✔ Correct answer
One of your tasks can take a lot time to finish, which may cause delay in your DAG Run completion.

What parameter can you use to **stop the DAG Run** if it is not completed in a given interval of time?

- [x] **dagrun_timeout**
- [ ] ~~timeout~~
- [ ] ~~execution_timeout~~

Question 64: ✔ Correct answer
Examine the following DAG:

```
default_args={
    'retries': 1
}

with DAG('car_dag', schedule_interval='*/8 * * * *',
    default_args=default_args, catchup=False) as dag:

    extract = BashOperator(
        task_id="extract",
        bash_command="curl -O output.json http://myapi.com"
        retries=3
    )
```

How many times does the task get retried before failing?

- [ ] ~~0~~
- [ ] ~~1~~
- [ ] ~~2~~
- [x] **3**
- [ ] ~~4~~

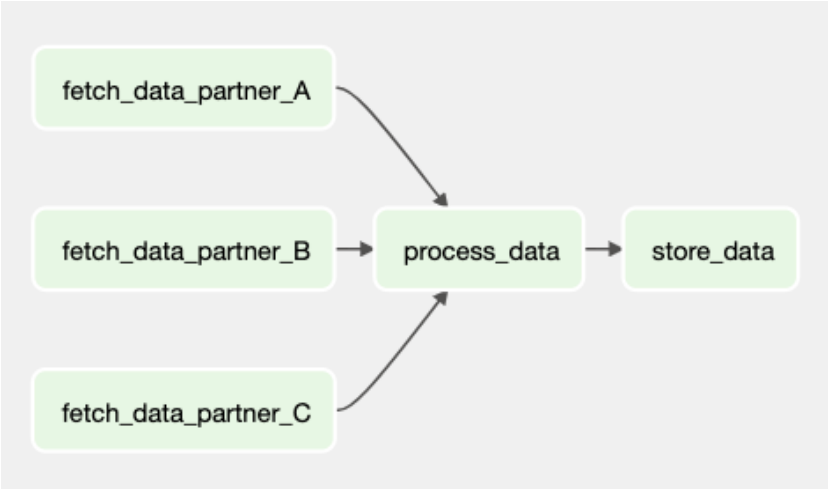
Question 65: ✔ Correct answer
Examine the following task:

```
extract = BashOperator(
    task_id="extract",
    bash_command="curl -O output.json http://myapi.com"
    depends_on_past=True
)
```

What does **`depends_on_past`** mean?

- [x] **If the task extract didn't succeed in the previous DAG Run, then it doesn't get triggered in the current DAG Run**
- [ ] ~~If the task extract succeeded in the previous DAG Run, then it doesn't get triggered in the current DAG Run~~
- [ ] ~~If the previous DAG Run didn't succeed, then it doesn't get triggered in the current DAG Run~~

Question 66: ✖ Incorrect answer
Examine the DAG below:



Assume the goal is to execute **`fetch_data_partner_C`** first, then **`fetch_data_partner_B`**, and finally **`fetch_data_partner_A`**. Is this possible?

- [x] ~~Yes, with pools.~~
- [ ] ~~No we can't. The scheduler will randomly choose which task to execute first.~~
- [ ] **Yes, by modifying the priority_weight of each task.**

Question 67: ✔ Correct answer
You created the following task to process data and send a Slack notification once it's done:

```
def _compute_data(input_path):
    process_data(input_path)
    send_slack_notification()
```

Is the task atomic?

- [ ] ~~Yes~~

- [x] **No**

Question 68: ✖ Incorrect answer
Given two DAGs with the same DAG ID. What do you expect to happen?

- [x] ~~Airflow will throw an error because it does not allow two DAGs to share the same dag_id.~~
- [ ] ~~Airflow will use the DAG with the earlier start_date parameter value and disregard the other DAG.~~
- [ ] ~~Airflow will automatically assign a new dag_id parameter value to one of the two DAGs.~~
- [ ] **Airflow will switch randomly between the two DAGs.**

Question 69: ✔ Correct answer
Which trigger rule allows a DAG to execute a task as soon as one of the parent tasks has succeeded?

- [x] **one_success**
- [ ] ~~all_success~~
- [ ] ~~all_done~~
- [ ] ~~all_finished~~

Question 70: ✔ Correct answer
Examine the following DAG:

```
@dag(start_date=datetime(2021, 1, 1))
def my_etl(dag_id):

    @task
    def extract():
        return 'my_data'

    @task
    def process(data):
        print(data)
        return 1

    @task
    def store(is_processed):
        if (is_processed):
            print(f"store {is_processed}")

    store(process(extract()))
```

What is the output of the task `store`?

- [ ] ~~my_data~~
- [x] **store 1**

- [ ] ~~no output~~

Question 71: ✔ Correct answer
What happens if there is no set **timeout** for a sensor?

- [ ] ~~It will run indefinitely~~
- [ ] ~~It will timeout after 1 day~~
- [x] **It will timeout after 7 days**
- [ ] ~~It will timeout after 10 days~~

Question 72: ✖ Incorrect answer
Is the following task idempotent?

```
fetch_data = BashOperator(
    task_id='fetch_data',
    bash_command='mkdir -p /tmp/data && curl -o /tmp/data/{{ds}}.json http://my_api:5000'
)
```

- [ ] **Yes**
- [x] ~~No~~

Question 73: ✔ Correct answer
You have a DAG with a start_date set to 1st of February 2021 and a schedule_interval set to @daily. You run it for a while, and now you change the schedule_interval so that the DAG is triggered every 10mins.

What will happen?

- [ ] ~~Nothing~~
- [x] **Previously TaskInstances won't align with the new schedule interval**

Question 74: ✖ Incorrect answer
You want to process your data incrementally. Therefore, you need to get the current execution date of your DAG Run.

Which of the following is the **best** way to get the execution date using the PythonOperator?

A)

```
def _process(ds):
    data = get_data(ds)
    process_data(data)
```

B)

```
def _process(**context):
    data = get_data(context['ds'])
    process_data(data)
```

C)

```
def _process():
    ctx = get_current_context()
    data = get_data(ctx['ds'])
    process_data(data)
```

- [ ] **A**
- [ ] ~~B~~
- [x] ~~C~~

Question 75: ✔ Correct answer
You want to wait for a task to complete in DAG A before executing a task in DAG B.

Which operator should you use?

- [ ] ~~TriggerDagRunOperator~~
- [x] **ExternalTaskSensor**