DAG Tasks and Operators

Tasks define a unit of work within a DAG. You can implement tasks using `operators` which define what task actually does. You've already used the `DummyOperator` from exercises before, now let's checkout two common operators that helps define custom operations on our workers.

Dog Pictures DAG

This example uses a simple DAG that fetches random dog image urls from `dog.ceo` api and then download them.

You can see the response of the API from here

```
GET https://dog.ceo/api/breed/corgi/images/random/5
```

Response:


```
{
   "message":[
      "https://images.dog.ceo/breeds/corgi-cardigan/n02113186_1120.jpg",
      "https://images.dog.ceo/breeds/corgi-cardigan/n02113186_3457.jpg",
      "https://images.dog.ceo/breeds/corgi-cardigan/n02113186_4536.jpg",
      "https://images.dog.ceo/breeds/corgi-cardigan/n02113186_6949.jpg",

"https://images.dog.ceo/breeds/corgi/11952722_10153542299813449_4751098630760063887_o.jpg"
   ],
   "status":"success"
}
```


BashOperator

We use the `BashOperator` to run commands on the bash shell (available on linux). In our example, we use it to send a GET request and save the response to `/tmp/malamutes.json` where we will parse it using our `PythonOperator`.

```
    fetch_dog_images = BashOperator(
        task_id="fetch_dogs_list",
        bash_command="curl -o /tmp/malamutes.json -L 'https://dog.ceo/api/breed/corgi/images/random/5'"
    )
```

You can get more examples to use `curl` from here : https://www.geeksforgeeks.org/curl-command-in-linux-with-examples/

You can also check what a shell command does if you're not sure using https://explainshell.com/ and type a shell command example `curl -o`

PythonOperator

Unlike the `BashOperator` which accepts a string, `PythonOperator` accepts a callable which is usually a function.

In our example, the parameter that accepts the function is `python_callable`

```
    get_dog_pictures = PythonOperator(
        task_id="get_dog_pictures",
        python_callable=_get_dog_pictures
    )
```

And in the `_get_dog_pictures` we do the following (check comments):

```
def _get_dog_pictures():
    # Make sure that /tmp/images folder exists.
    pathlib.Path("/tmp/images").mkdir(parents=True, exist_ok=True)

    # Open the json file
    with open("/tmp/corgis.json") as f:
        corgis = json.load(f)

        # Check that the API response status is "success"
        if corgis["status"] == "success":
            image_urls = corgis["message"]
            # Iterate over the list of URLS
            for image_url in image_urls:
                try:
                    # Save them to an images folder
                    response = requests.get(image_url)
                    image_filename = image_url.split("/")[-1]
                    target_file = f"/tmp/images/{image_filename}"
                    with open(target_file, "wb") as f:
                        f.write(response.content)
                except:
                    raise AirflowFailException
        else:
            raise AirflowFailException
```

Note that raising `AirFlowFailException` will prevent automatic retry

Other operators

There are other operators you can use with Airflow and you can even make your own. You can find a list of other operators here: https://airflow.apache.org/docs/apache-airflow/stable/_api/airflow/operators/index.html