Working with templates

INTRODUCTION TO AIRFLOW IN PYTHON

Mike Metzger
Data Engineer

What are templates?

Templates:

- Allow substituting information during a DAG run
- Provide added flexibility when defining tasks
- Are created using the `Jinja` templating language

Non-Templated BashOperator example

Create a task to echo a list of files:

```python
t1 = BashOperator(
        task_id='first_task',
        bash_command='echo "Reading file1.txt"',
        dag=dag)

t2 = BashOperator(
        task_id='second_task',
        bash_command='echo "Reading file2.txt"',
        dag=dag)
```

Templated BashOperator example

```
templated_command="""
  echo "Reading {{ params.filename }}"
"""

t1 = BashOperator(task_id='template_task',
       bash_command=templated_command,
       params={'filename': 'file1.txt'}
       dag=example_dag)
```

Output:

```
Reading file1.txt
```

Templated BashOperator example (continued)

```
templated_command="""
  echo "Reading {{ params.filename }}"
"""

t1 = BashOperator(task_id='template_task',
       bash_command=templated_command,
       params={'filename': 'file1.txt'}
       dag=example_dag)

t2 = BashOperator(task_id='template_task',
       bash_command=templated_command,
       params={'filename': 'file2.txt'}
       dag=example_dag)
```

Let's practice!

INTRODUCTION TO AIRFLOW IN PYTHON

More templates

INTRODUCTION TO AIRFLOW IN PYTHON

Mike Metzger
Data Engineer

Quick task reminder

- Take a list of filenames
- Print "Reading <filename>" to the log / output
- Templated version:

```
templated_command="""
echo "Reading {{ params.filename }}"
"""
t1 = BashOperator(task_id='template_task',
       bash_command=templated_command,
       params={'filename': 'file1.txt'}
       dag=example_dag)
```

More advanced template

```
templated_command="""
{% for filename in params.filenames %}
  echo "Reading {{ filename }}"
{% endfor %}
"""

t1 = BashOperator(task_id='template_task',
       bash_command=templated_command,
       params={'filenames': ['file1.txt', 'file2.txt']}
       dag=example_dag)
```

```
Reading file1.txt
Reading file2.txt
```

Variables

- Airflow built-in runtime variables
- Provides assorted information about DAG runs, tasks, and even the system configuration.
- Examples include:

```
Execution Date: {{ ds }}                                   # YYYY-MM-DD
Execution Date, no dashes: {{ ds_nodash }}                 # YYYYMMDD

Previous Execution date: {{ prev_ds }}                     # YYYY-MM-DD
Prev Execution date, no dashes: {{ prev_ds_nodash }}       # YYYYMMDD

DAG object: {{ dag }}

Airflow config object: {{ conf }}
```

[1] https://airflow.apache.org/docs/stable/macros-ref.html

Macros

In addition to others, there is also a `{{ macros }}` variable.

This is a reference to the Airflow macros package which provides various useful objects / methods for Airflow templates.

- `{{ macros.datetime }}` : The `datetime.datetime` object
- `{{ macros.timedelta }}` : The `timedelta` object
- `{{ macros.uuid }}` : Python's `uuid` object
- `{{ macros.ds_add('2020-04-15', 5) }}` : Modify days from a date, this example returns 2020-04-20

Let's practice!

INTRODUCTION TO AIRFLOW IN PYTHON

Branching

INTRODUCTION TO AIRFLOW IN PYTHON

Mike Metzger
Data Engineer

Branching

Branching in Airflow:

- Provides conditional logic
- Using `BranchPythonOperator`
- `from airflow.operators.python_operator import BranchPythonOperator`
- Takes a `python_callable` to return the next task id (or list of ids) to follow

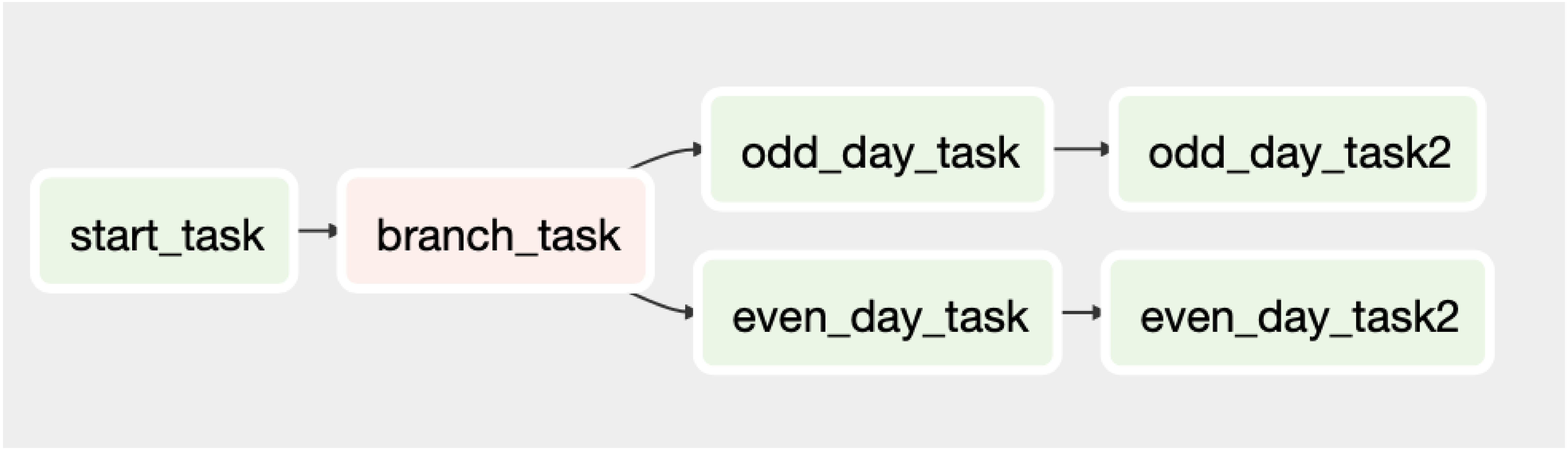
Branching example

```python
def branch_test(**kwargs):
  if int(kwargs['ds_nodash']) % 2 == 0:
    return 'even_day_task'
  else:
    return 'odd_day_task'
```

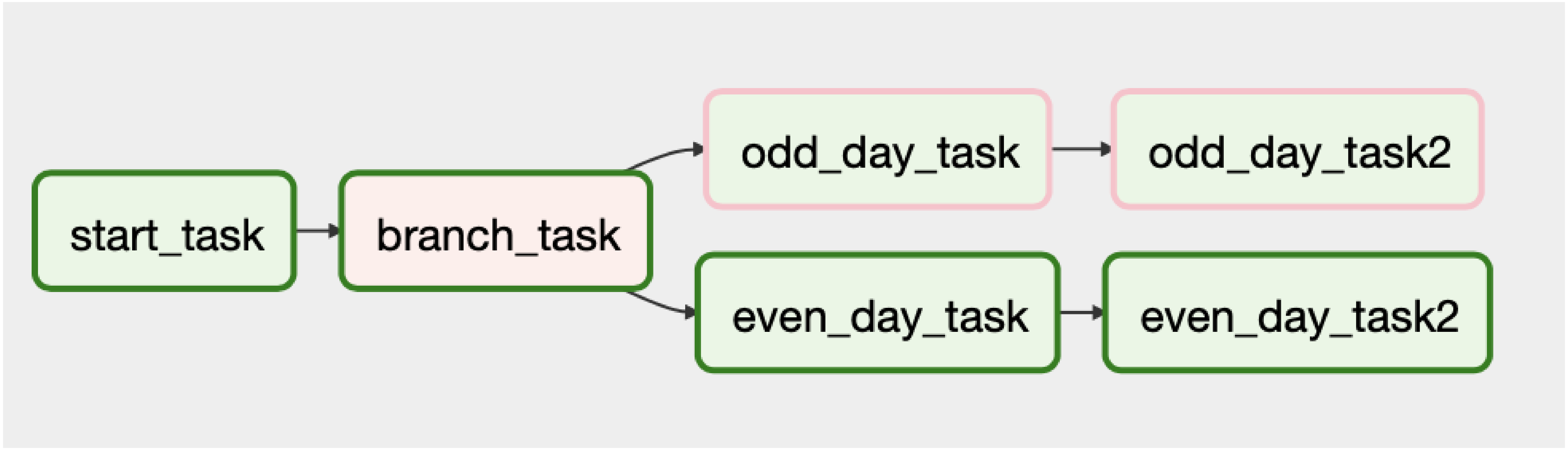
Branching example

```python
def branch_test(**kwargs):
  if int(kwargs['ds_nodash']) % 2 == 0:
    return 'even_day_task'
  else:
    return 'odd_day_task'


branch_task = BranchPythonOperator(task_id='branch_task',dag=dag,
       provide_context=True,
       python_callable=branch_test)


start_task >> branch_task >> even_day_task >> even_day_task2
branch_task >> odd_day_task >> odd_day_task2
```

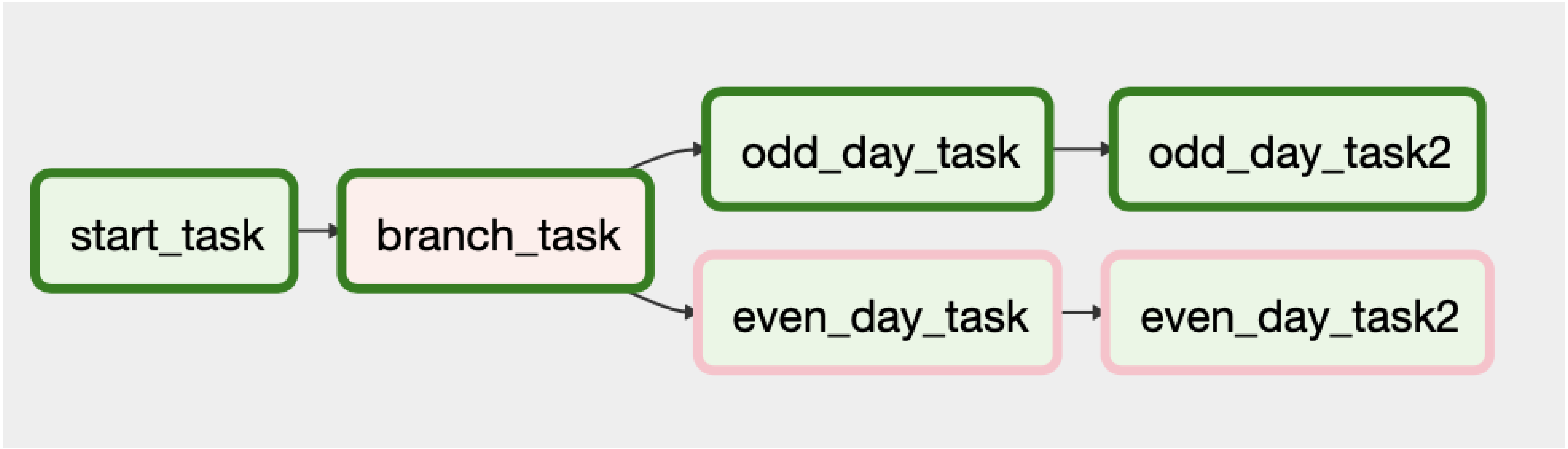
Branching graph view



Branching even days



Branching odd days



Let's practice!

INTRODUCTION TO AIRFLOW IN PYTHON

Creating a production pipeline

INTRODUCTION TO AIRFLOW IN PYTHON

Mike Metzger
Data Engineer

Running DAGs & Tasks

To run a specific task from command-line:

```
airflow run <dag_id> <task_id> <date>
```

To run a full DAG:

```
airflow trigger_dag -e <date> <dag_id>
```

Operators reminder

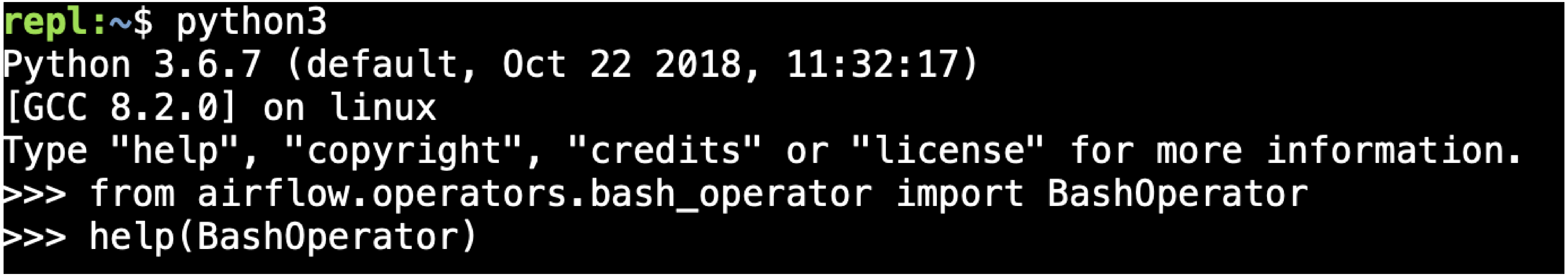
- BashOperator - expects a `bash_command`
- PythonOperator - expects a `python_callable`
- BranchPythonOperator - requires a `python_callable` and `provide_context=True` . The callable must accept `**kwargs` .
- FileSensor - requires `filepath` argument and might need `mode` or `poke_interval` attributes

Template reminders

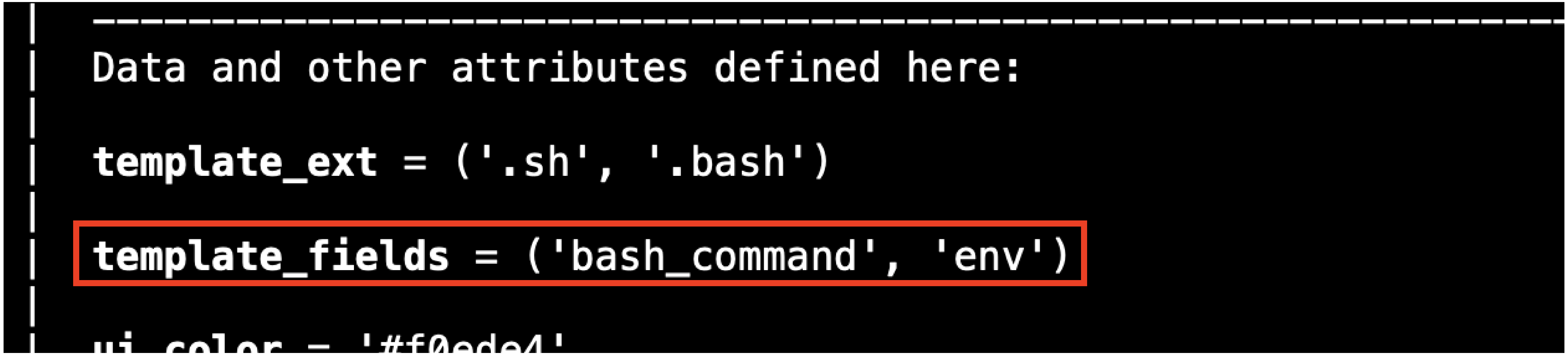
- Many objects in Airflow can use templates
- Certain fields may use templated strings, while others do not
- One way to check is to use built-in documentation:

1. Open python3 interpreter
2. Import necessary libraries (ie, `from airflow.operators.bash_operator import BashOperator`)
3. At prompt, run `help(<Airflow object>)` , ie, `help(BashOperator)`
4. Look for a line that referencing *template_fields*. This will specify any of the arguments that can use templates.

Template documentation example



```
repl:~$ python3
Python 3.6.7 (default, Oct 22 2018, 11:32:17)
[GCC 8.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from airflow.operators.bash_operator import BashOperator
>>> help(BashOperator)
```



```
 |   ----------------------------------------------------------------------
 |   Data and other attributes defined here:
 |
 |   template_ext = ('.sh', '.bash')
 |
 |   template_fields = ('bash_command', 'env')
 |
 |   ui_color = '#f0ede4'
```

Let's practice!

INTRODUCTION TO AIRFLOW IN PYTHON

Congratulations!

INTRODUCTION TO AIRFLOW IN PYTHON

Mike Metzger
Data Engineer

What we've learned

- Workflows / DAGs
- Operators (BashOperator, PythonOperator, EmailOperator)
- Tasks
- Dependencies / Bitshift operators
- Sensors
- Scheduling
- SLAs / Alerting
- Templates
- Branching
- Airflow command line / UI
- Airflow executors
- Debugging / Troubleshooting

Next steps

- Setup your own environment for practice
- Look into other operators / sensors
- Experiment with dependencies
- Look into parts of Airflow we didn't cover
    - XCom
    - Connections
    - Refer to docs for more
- Keep building workflows!

Thank you!

INTRODUCTION TO AIRFLOW IN PYTHON