Using custom functions and variables in templates

As discussed previously, there are several variables available during templating. A Jinja environment and Airflow runtime are different. You can view a Jinja environment as a very stripped-down Python environment. That, among other things, means modules cannot be imported. For example, this command won't work in a Jinja template:

```
from datetime import datetime
BashOperator(
    task_id="print_now",
    bash_command="echo It is currently {{ datetime.now() }}",  # raises jinja2.exceptions.UndefinedError: 'datetime' is undefined
)
```

However, it is possible to inject functions into your Jinja environment. In Airflow, several standard Python modules are injected by default for templating, under the name macros. For example, the previous code example can be updated to use `macros.datetime`:

```
BashOperator(
    task_id="print_now",
    bash_command="echo It is currently {{ macros.datetime.now() }}",  # It is currently 2021-08-30 13:51:55.820299
)
```

Besides pre-injected functions, you can also use self-defined variables and functions in your templates. Airflow provides a convenient way to inject these into the Jinja environment. In the following example, a function is added to the DAG to print the number of days since May 1st, 2015:

```
def days_to_now(starting_date):
    return (datetime.now() - starting_date).days
```

To use this inside a Jinja template, you can pass a dict to `user_defined_macros` in the DAG. For example:

```
def days_to_now(starting_date):
    return (datetime.now() - starting_date).days
with DAG(
    dag_id="demo_template",
    start_date=datetime(2021, 1, 1),
    schedule=None,
    user_defined_macros={
        "starting_date": datetime(2015, 5, 1),  # Macro can be a variable
        "days_to_now": days_to_now,  # Macro can also be a function
    },
) as dag:
    print_days = BashOperator(
        task_id="print_days",
        bash_command="echo Days since {{ starting_date }} is {{ days_to_now(starting_date) }}",  # Call user defined macros
    )
    # Days since 2015-05-01 00:00:00 is 2313
```

It's also possible to inject functions as Jinja filters using `user_defined_filters`. You can use filters as pipe-operations. The following example completes the same work as the previous example, only this time filters are used:

```
with DAG(
    dag_id="bash_script_template",
    start_date=datetime(2021, 1, 1),
    schedule=None,
    user_defined_filters={"days_to_now": days_to_now},  # Set user_defined_filters to use function as pipe-operation
    user_defined_macros={"starting_date": datetime(2015, 5, 1)},
) as dag:
    print_days = BashOperator(
        task_id="print_days",
        bash_command="echo Days since {{ starting_date }} is {{ starting_date | days_to_now }}",  # Pipe value to function
    )
    # Days since 2015-05-01 00:00:00 is 2313
```

Functions injected with `user_defined_filters` and `user_defined_macros` are both usable in the Jinja environment. While they achieve the same result, Astronomer recommends using filters when you need to import multiple custom functions because the filter formatting improves the readability of your code. You can see this when comparing the two techniques side-to-side:

```
"{{ name | striptags | title }}"  # chained filters are read naturally from left to right
"{{ title(striptags(name)) }}"  # multiple functions are more difficult to interpret because reading right to left
```