Sub-dependencies

You can create dependencies that have **sub-dependencies**.

They can be as **deep** as you need them to be.

FastAPI will take care of solving them.

First dependency "dependable"

You could create a first dependency ("dependable") like:

=== "Python 3.6 and above"

```
```Python hl_lines="8-9"
{!> ../../../docs_src/dependencies/tutorial005.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="6-7"
{!> ../../../docs_src/dependencies/tutorial005_py310.py!}
```
```

It declares an optional query parameter `q` as a `str`, and then it just returns it.

This is quite simple (not very useful), but will help us focus on how the sub-dependencies work.

Second dependency, "dependable" and "dependant"

Then you can create another dependency function (a "dependable") that at the same time declares a dependency of its own (so it is a "dependant" too):

=== "Python 3.6 and above"

```
```Python hl_lines="13"
{!> ../../../docs_src/dependencies/tutorial005.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="11"
{!> ../../../docs_src/dependencies/tutorial005_py310.py!}
```
```

Let's focus on the parameters declared:

- Even though this function is a dependency ("dependable") itself, it also declares another dependency (it "depends" on something else).
    - It depends on the `query_extractor`, and assigns the value returned by it to the parameter `q`.
- It also declares an optional `last_query` cookie, as a `str`.

- If the user didn't provide any query `q`, we use the last query used, which we saved to a cookie before.

Use the dependency

Then we can use the dependency with:

=== "Python 3.6 and above"

```
```Python hl_lines="21"
{!> ../../../docs_src/dependencies/tutorial005.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="19"
{!> ../../../docs_src/dependencies/tutorial005_py310.py!}
```
```

!!! info Notice that we are only declaring one dependency in the *path operation function*, the `query_or_cookie_extractor`.

```
But **FastAPI** will know that it has to solve `query_extractor` first, to pass the
results of that to `query_or_cookie_extractor` while calling it.
```

```
graph TB

query_extractor(["query_extractor"])
query_or_cookie_extractor(["query_or_cookie_extractor"])

read_query["/items/"]

query_extractor --> query_or_cookie_extractor --> read_query
```

Using the same dependency multiple times

If one of your dependencies is declared multiple times for the same *path operation*, for example, multiple dependencies have a common sub-dependency, **FastAPI** will know to call that sub-dependency only once per request.

And it will save the returned value in a "cache" and pass it to all the "dependants" that need it in that specific request, instead of calling the dependency multiple times for the same request.

In an advanced scenario where you know you need the dependency to be called at every step (possibly multiple times) in the same request instead of using the "cached" value, you can set the parameter `use_cache=False` when using `Depends`:

```
async def needy_dependency(fresh_value: str = Depends(get_value, use_cache=False)):
    return {"fresh_value": fresh_value}
```

Recap

Apart from all the fancy words used here, the **Dependency Injection** system is quite simple.

Just functions that look the same as the *path operation functions*.

But still, it is very powerful, and allows you to declare arbitrarily deeply nested dependency "graphs" (trees).

!!! tip All this might not seem as useful with these simple examples.

```
But you will see how useful it is in the chapters about **security**.

And you will also see the amounts of code it will save you.
```