Classes as Dependencies

Before diving deeper into the **Dependency Injection** system, let's upgrade the previous example.

A `dict` from the previous example

In the previous example, we were returning a `dict` from our dependency ("dependable"):

Python 3.10+

```
from typing import Annotated

from fastapi import Depends, FastAPI

app = FastAPI()


async def common_parameters(q: str | None = None, skip: int = 0, limit: int = 100):
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: Annotated[dict, Depends(common_parameters)]):
    return commons


@app.get("/users/")
async def read_users(commons: Annotated[dict, Depends(common_parameters)]):
    return commons
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Depends, FastAPI

app = FastAPI()


async def common_parameters(
    q: Union[str, None] = None, skip: int = 0, limit: int = 100
):
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: Annotated[dict, Depends(common_parameters)]):
    return commons


@app.get("/users/")
async def read_users(commons: Annotated[dict, Depends(common_parameters)]):
    return commons
```

Python 3.8+

```
from typing import Union

from fastapi import Depends, FastAPI
from typing_extensions import Annotated

app = FastAPI()


async def common_parameters(
    q: Union[str, None] = None, skip: int = 0, limit: int = 100
):
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: Annotated[dict, Depends(common_parameters)]):
    return commons


@app.get("/users/")
async def read_users(commons: Annotated[dict, Depends(common_parameters)]):
    return commons
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI

app = FastAPI()


async def common_parameters(q: str | None = None, skip: int = 0, limit: int = 100):
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: dict = Depends(common_parameters)):
    return commons


@app.get("/users/")
async def read_users(commons: dict = Depends(common_parameters)):
    return commons
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI

app = FastAPI()


async def common_parameters(
    q: Union[str, None] = None, skip: int = 0, limit: int = 100
):
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: dict = Depends(common_parameters)):
    return commons


@app.get("/users/")
async def read_users(commons: dict = Depends(common_parameters)):
    return commons
```

But then we get a `dict` in the parameter `commons` of the *path operation function*.

And we know that editors can't provide a lot of support (like completion) for `dict`s, because they can't know their keys and value types.

We can do better...

What makes a dependency

Up to now you have seen dependencies declared as functions.

But that's not the only way to declare dependencies (although it would probably be the more common).

The key factor is that a dependency should be a "callable".

A "**callable**" in Python is anything that Python can "call" like a function.

So, if you have an object `something` (that might *not* be a function) and you can "call" it (execute it) like:

```
something()
```

or

```
something(some_argument, some_keyword_argument="foo")
```

then it is a "callable".

Classes as dependencies

You might notice that to create an instance of a Python class, you use that same syntax.

For example:

```
class Cat:
```

```
    def __init__(self, name: str):
        self.name = name


fluffy = Cat(name="Mr Fluffy")
```

In this case, `fluffy` is an instance of the class `Cat`.

And to create `fluffy`, you are "calling" `Cat`.

So, a Python class is also a **callable**.

Then, in **FastAPI**, you could use a Python class as a dependency.

What FastAPI actually checks is that it is a "callable" (function, class or anything else) and the parameters defined.

If you pass a "callable" as a dependency in **FastAPI**, it will analyze the parameters for that "callable", and process them in the same way as the parameters for a *path operation function*. Including sub-dependencies.

That also applies to callables with no parameters at all. The same as it would be for *path operation functions* with no parameters.

Then, we can change the dependency "dependable" `common_parameters` from above to the class `CommonQueryParams`:

Python 3.10+

```
from typing import Annotated

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+

```
from typing import Union

from fastapi import Depends, FastAPI
from typing_extensions import Annotated

app = FastAPI()
```

```
fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: CommonQueryParams = Depends(CommonQueryParams)):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: CommonQueryParams = Depends(CommonQueryParams)):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Pay attention to the `__init__` method used to create the instance of the class:

Python 3.10+

```python
from typing import Annotated

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

▸ Other versions and variants

Python 3.9+

```python
from typing import Annotated, Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+

```python
from typing import Union

from fastapi import Depends, FastAPI
from typing_extensions import Annotated

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: CommonQueryParams = Depends(CommonQueryParams)):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: CommonQueryParams = Depends(CommonQueryParams)):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

...it has the same parameters as our previous `common_parameters` :

Python 3.10+

```
from typing import Annotated

from fastapi import Depends, FastAPI

app = FastAPI()


async def common_parameters(q: str | None = None, skip: int = 0, limit: int = 100):
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: Annotated[dict, Depends(common_parameters)]):
    return commons


@app.get("/users/")
async def read_users(commons: Annotated[dict, Depends(common_parameters)]):
    return commons
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union
```

```
from fastapi import Depends, FastAPI

app = FastAPI()


async def common_parameters(
    q: Union[str, None] = None, skip: int = 0, limit: int = 100
):
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: Annotated[dict, Depends(common_parameters)]):
    return commons


@app.get("/users/")
async def read_users(commons: Annotated[dict, Depends(common_parameters)]):
    return commons
```

Python 3.8+

```
from typing import Union

from fastapi import Depends, FastAPI
from typing_extensions import Annotated

app = FastAPI()


async def common_parameters(
    q: Union[str, None] = None, skip: int = 0, limit: int = 100
):
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: Annotated[dict, Depends(common_parameters)]):
    return commons


@app.get("/users/")
async def read_users(commons: Annotated[dict, Depends(common_parameters)]):
    return commons
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI

app = FastAPI()


async def common_parameters(q: str | None = None, skip: int = 0, limit: int = 100):
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: dict = Depends(common_parameters)):
    return commons


@app.get("/users/")
async def read_users(commons: dict = Depends(common_parameters)):
    return commons
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI

app = FastAPI()


async def common_parameters(
    q: Union[str, None] = None, skip: int = 0, limit: int = 100
):
```

```
    return {"q": q, "skip": skip, "limit": limit}


@app.get("/items/")
async def read_items(commons: dict = Depends(common_parameters)):
    return commons


@app.get("/users/")
async def read_users(commons: dict = Depends(common_parameters)):
    return commons
```

Those parameters are what **FastAPI** will use to "solve" the dependency.

In both cases, it will have:

- An optional `q` query parameter that is a `str`.
- A `skip` query parameter that is an `int`, with a default of `0`.
- A `limit` query parameter that is an `int`, with a default of `100`.

In both cases the data will be converted, validated, documented on the OpenAPI schema, etc.

Use it

Now you can declare your dependency using this class.

Python 3.10+

```
from typing import Annotated

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

▸ Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+

```python
from typing import Union

from fastapi import Depends, FastAPI
from typing_extensions import Annotated

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```python
from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: CommonQueryParams = Depends(CommonQueryParams)):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: CommonQueryParams = Depends(CommonQueryParams)):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

FastAPI calls the `CommonQueryParams` class. This creates an "instance" of that class and the instance will be passed as the parameter `commons` to your function.

Type annotation vs `Depends`

Notice how we write `CommonQueryParams` twice in the above code:

Python 3.8+

```
commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]
```

Python 3.8+ non-Annotated

> **Tip**
>
> Prefer to use the `Annotated` version if possible.

```
commons: CommonQueryParams = Depends(CommonQueryParams)
```

The last `CommonQueryParams`, in:

```
... Depends(CommonQueryParams)
```

...is what **FastAPI** will actually use to know what is the dependency.

It is from this one that FastAPI will extract the declared parameters and that is what FastAPI will actually call.

In this case, the first `CommonQueryParams`, in:

Python 3.8+

```
commons: Annotated[CommonQueryParams, ...
```

Python 3.8+ non-Annotated

> **Tip**
>
> Prefer to use the `Annotated` version if possible.

```
commons: CommonQueryParams ...
```

...doesn't have any special meaning for **FastAPI**. FastAPI won't use it for data conversion, validation, etc. (as it is using the `Depends(CommonQueryParams)` for that).

You could actually write just:

Python 3.8+

```
commons: Annotated[Any, Depends(CommonQueryParams)]
```

Python 3.8+ non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
commons = Depends(CommonQueryParams)
```

...as in:

Python 3.10+

```
from typing import Annotated, Any

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[Any, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

▸ Other versions and variants

Python 3.9+

```
from typing import Annotated, Any, Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[Any, Depends(CommonQueryParams)]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+

```
from typing import Any, Union

from fastapi import Depends, FastAPI
from typing_extensions import Annotated

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[Any, Depends(CommonQueryParams)]):
```

```
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.10+ - non-Annotated

> **Tip**
>
> Prefer to use the `Annotated` version if possible.

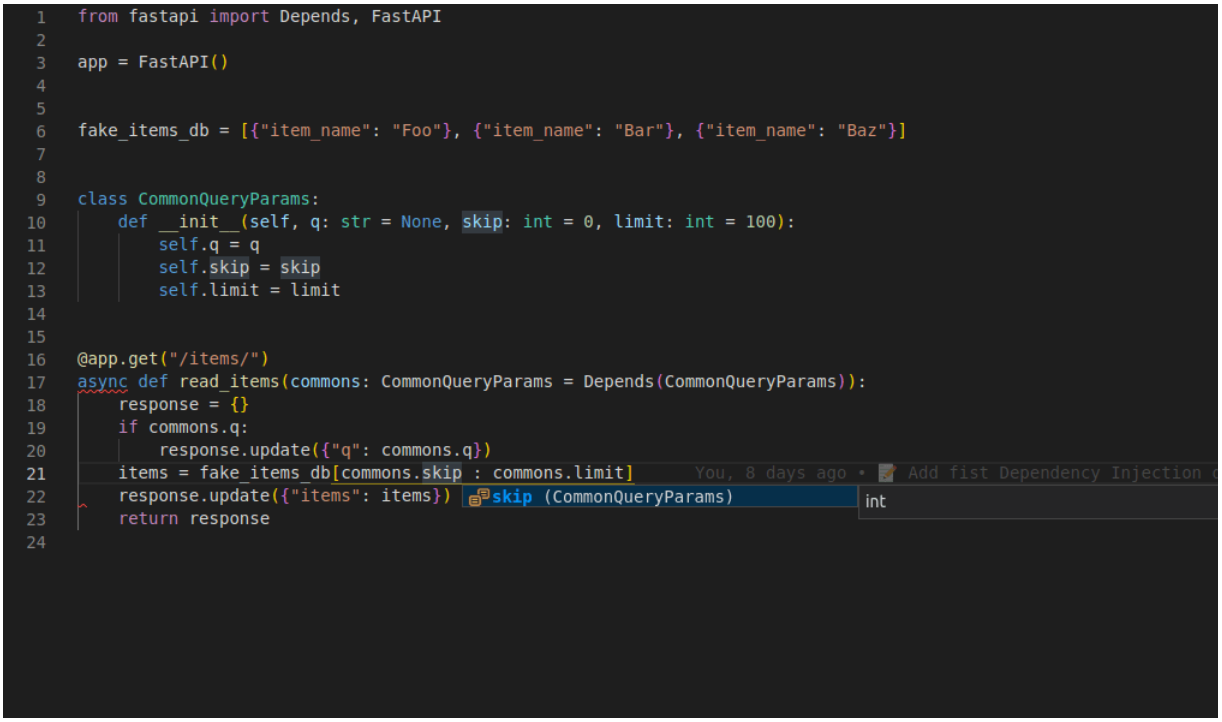
```
from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons=Depends(CommonQueryParams)):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+ - non-Annotated

> **Tip**
>
> Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons=Depends(CommonQueryParams)):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

But declaring the type is encouraged as that way your editor will know what will be passed as the parameter `commons`, and then it can help you with code completion, type checks, etc:



Shortcut

But you see that we are having some code repetition here, writing `CommonQueryParams` twice:

Python 3.8+

```
commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]
```

Python 3.8+ non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
commons: CommonQueryParams = Depends(CommonQueryParams)
```

FastAPI provides a shortcut for these cases, in where the dependency is *specifically* a class that **FastAPI** will "call" to create an instance of the class itself.

For those specific cases, you can do the following:

Instead of writing:

Python 3.8+

```
commons: Annotated[CommonQueryParams, Depends(CommonQueryParams)]
```

Python 3.8+ non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
commons: CommonQueryParams = Depends(CommonQueryParams)
```

...you write:

Python 3.8+

```
commons: Annotated[CommonQueryParams, Depends()]
```

Python 3.8 non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
commons: CommonQueryParams = Depends()
```

You declare the dependency as the type of the parameter, and you use `Depends()` without any parameter, instead of having to write the full class *again* inside of `Depends(CommonQueryParams)`.

The same example would then look like:

Python 3.10+

```
from typing import Annotated

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends()]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

▸ 🤓 Other versions and variants

Python 3.9+

```
from typing import Annotated, Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends()]):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+

```
from typing import Union

from fastapi import Depends, FastAPI
from typing_extensions import Annotated

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: Annotated[CommonQueryParams, Depends()]):
```

```
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.10+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: str | None = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: CommonQueryParams = Depends()):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

Python 3.8+ - non-Annotated

Tip

Prefer to use the `Annotated` version if possible.

```
from typing import Union

from fastapi import Depends, FastAPI

app = FastAPI()


fake_items_db = [{"item_name": "Foo"}, {"item_name": "Bar"}, {"item_name": "Baz"}]


class CommonQueryParams:
    def __init__(self, q: Union[str, None] = None, skip: int = 0, limit: int = 100):
        self.q = q
        self.skip = skip
        self.limit = limit


@app.get("/items/")
async def read_items(commons: CommonQueryParams = Depends()):
    response = {}
    if commons.q:
        response.update({"q": commons.q})
    items = fake_items_db[commons.skip : commons.skip + commons.limit]
    response.update({"items": items})
    return response
```

...and **FastAPI** will know what to do.

Tip

If that seems more confusing than helpful, disregard it, you don't *need* it.

It is just a shortcut. Because **FastAPI** cares about helping you minimize code repetition.