Query Parameters

When you declare other function parameters that are not part of the path parameters, they are automatically interpreted as "query" parameters.

Python 3.8+

```
from fastapi import FastAPI

app = FastAPI()

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


@app.get("/items/")
async def read_item(skip: int = 0, limit: int = 10):
    return fake_items_db[skip : skip + limit]
```

The query is the set of key-value pairs that go after the `?` in a URL, separated by `&` characters.

For example, in the URL:

```
http://127.0.0.1:8000/items/?skip=0&limit=10
```

...the query parameters are:

- `skip` : with a value of `0`
- `limit` : with a value of `10`

As they are part of the URL, they are "naturally" strings.

But when you declare them with Python types (in the example above, as `int`), they are converted to that type and validated against it.

All the same process that applied for path parameters also applies for query parameters:

- Editor support (obviously)
- Data "parsing"
- Data validation
- Automatic documentation

Defaults

As query parameters are not a fixed part of a path, they can be optional and can have default values.

In the example above they have default values of `skip=0` and `limit=10` .

So, going to the URL:

```
http://127.0.0.1:8000/items/
```

would be the same as going to:

```
http://127.0.0.1:8000/items/?skip=0&limit=10
```

But if you go to, for example:

```
http://127.0.0.1:8000/items/?skip=20
```

The parameter values in your function will be:

- `skip=20` : because you set it in the URL
- `limit=10` : because that was the default value

Optional parameters

The same way, you can declare optional query parameters, by setting their default to `None` :

Python 3.10+

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_item(item_id: str, q: str | None = None):
    if q:
        return {"item_id": item_id, "q": q}
    return {"item_id": item_id}
```

▸ 🤓 Other versions and variants

Python 3.8+

```python
from typing import Union

from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_item(item_id: str, q: Union[str, None] = None):
    if q:
        return {"item_id": item_id, "q": q}
    return {"item_id": item_id}
```

In this case, the function parameter `q` will be optional, and will be `None` by default.

Check

Also notice that **FastAPI** is smart enough to notice that the path parameter `item_id` is a path parameter and `q` is not, so, it's a query parameter.

Query parameter type conversion

You can also declare `bool` types, and they will be converted:

Python 3.10+

```python
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_item(item_id: str, q: str | None = None, short: bool = False):
    item = {"item_id": item_id}
    if q:
        item.update({"q": q})
    if not short:
        item.update(
            {"description": "This is an amazing item that has a long description"}
        )
    return item
```

▸ 🤓 Other versions and variants

Python 3.8+

```python
from typing import Union

from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_item(item_id: str, q: Union[str, None] = None, short: bool = False):
    item = {"item_id": item_id}
    if q:
        item.update({"q": q})
    if not short:
        item.update(
            {"description": "This is an amazing item that has a long description"}
        )
    return item
```

In this case, if you go to:

```
http://127.0.0.1:8000/items/foo?short=1
```

or

```
http://127.0.0.1:8000/items/foo?short=True
```

or

```
http://127.0.0.1:8000/items/foo?short=true
```

or

```
http://127.0.0.1:8000/items/foo?short=on
```

or

```
http://127.0.0.1:8000/items/foo?short=yes
```

or any other case variation (uppercase, first letter in uppercase, etc), your function will see the parameter `short` with a `bool` value of `True` . Otherwise as `False` .

Multiple path and query parameters

You can declare multiple path parameters and query parameters at the same time, **FastAPI** knows which is which.

And you don't have to declare them in any specific order.

They will be detected by name:

Python 3.10+

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/users/{user_id}/items/{item_id}")
async def read_user_item(
    user_id: int, item_id: str, q: str | None = None, short: bool = False
):
    item = {"item_id": item_id, "owner_id": user_id}
    if q:
        item.update({"q": q})
    if not short:
        item.update(
            {"description": "This is an amazing item that has a long description"}
        )
    return item
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from typing import Union

from fastapi import FastAPI

app = FastAPI()


@app.get("/users/{user_id}/items/{item_id}")
async def read_user_item(
    user_id: int, item_id: str, q: Union[str, None] = None, short: bool = False
):
    item = {"item_id": item_id, "owner_id": user_id}
    if q:
        item.update({"q": q})
    if not short:
        item.update(
            {"description": "This is an amazing item that has a long description"}
        )
    return item
```

Required query parameters

When you declare a default value for non-path parameters (for now, we have only seen query parameters), then it is not required.

If you don't want to add a specific value but just make it optional, set the default as `None` .

But when you want to make a query parameter required, you can just not declare any default value:

Python 3.8+

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_user_item(item_id: str, needy: str):
    item = {"item_id": item_id, "needy": needy}
    return item
```

Here the query parameter `needy` is a required query parameter of type `str` .

If you open in your browser a URL like:

```
http://127.0.0.1:8000/items/foo-item
```

...without adding the required parameter `needy`, you will see an error like:

```
{
  "detail": [
    {
      "type": "missing",
      "loc": [
        "query",
        "needy"
      ],
      "msg": "Field required",
      "input": null,
      "url": "https://errors.pydantic.dev/2.1/v/missing"
    }
  ]
}
```

As `needy` is a required parameter, you would need to set it in the URL:

```
http://127.0.0.1:8000/items/foo-item?needy=sooooneedy
```

...this would work:

```
{
    "item_id": "foo-item",
    "needy": "sooooneedy"
}
```

And of course, you can define some parameters as required, some as having a default value, and some entirely optional:

Python 3.10+

```
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_user_item(
    item_id: str, needy: str, skip: int = 0, limit: int | None = None
):
    item = {"item_id": item_id, "needy": needy, "skip": skip, "limit": limit}
    return item
```

▸ 🤓 Other versions and variants

Python 3.8+

```
from typing import Union

from fastapi import FastAPI

app = FastAPI()


@app.get("/items/{item_id}")
async def read_user_item(
    item_id: str, needy: str, skip: int = 0, limit: Union[int, None] = None
):
    item = {"item_id": item_id, "needy": needy, "skip": skip, "limit": limit}
    return item
```

In this case, there are 3 query parameters:

- `needy`, a required `str`.
- `skip`, an `int` with a default value of `0`.
- `limit`, an optional `int`.

Tip

You could also use `Enum`s the same way as with Path Parameters.