# Query Parameters and String Validations

**FastAPI** allows you to declare additional information and validation for your parameters.

Let's take this application as example:

Python 3.6 and above

```Python
from typing import Union

from fastapi import FastAPI

app = FastAPI()


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

Python 3.10 and above

```Python
from fastapi import FastAPI

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = None):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

The query parameter `q` is of type `Union[str, None]` (or `str | None` in Python 3.10), that means that it's of type `str` but could also be `None`, and indeed, the default value is `None`, so FastAPI will know it's not required.

> **Note**
>
> FastAPI will know that the value of `q` is not required because of the default value `= None`.
>
> The `Union` in `Union[str, None]` will allow your editor to give you better support and d
> errors.

## Additional validation

We are going to enforce that even though `q` is optional, whenever it is provided, **its length doesn't exceed 50 characters**.

### Import `Query`

To achieve that, first import `Query` from `fastapi`:

**Python 3.6 and above**

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[str, None] = Query(default=None,
max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Python 3.10 and above**

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=None, max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

## Use `Query` as the default value

And now use it as the default value of your parameter, setting the parameter `max_length` to 50:

**Python 3.6 and above**

```
from typing import Union

from fastapi import FastAPI, Query
```

```
app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[str, None] = Query(default=None,
max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Python 3.10 and above**

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=None, max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

As we have to replace the default value `None` in the function with `Query()`, we can now set the default value with the parameter `Query(default=None)`, it serves the same purpose of defining that default value.

So:

```
q: Union[str, None] = Query(default=None)
```

...makes the parameter optional, the same as:

```
q: Union[str, None] = None
```

And in Python 3.10 and above:

```
q: str | None = Query(default=None)
```

...makes the parameter optional, the same as:

```
q: str | None = None
```

But it declares it explicitly as being a query parameter.

**Info**

Have in mind that the most important part to make a parameter optional is the part:

```
= None
```

or the:

```
= Query(default=None)
```

as it will use that `None` as the default value, and that way make the parameter **not required**.

The `Union[str, None]` part allows your editor to provide better support, but it is not what tells FastAPI that this parameter is not required.

Then, we can pass more parameters to `Query`. In this case, the `max_length` parameter that applies to strings:

```
q: Union[str, None] = Query(default=None, max_length=50)
```

This will validate the data, show a clear error when the data is not valid, and document the parameter in the OpenAPI schema *path operation*.

## Add more validations

You can also add a parameter `min_length`:

**Python 3.6 and above**

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(default=None, min_length=3, max_length=50)
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Python 3.10 and above**

```Python
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=None, min_length=3,
max_length=50)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

## Add regular expressions

You can define a regular expression that the parameter should match:

**Python 3.6 and above**

```Python
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(
        default=None, min_length=3, max_length=50, regex="^fixedquery$"
    )
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Python 3.10 and above**

```Python
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: str
    | None = Query(default=None, min_length=3, max_length=50,
regex="^fixedquery$")
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

This specific regular expression checks that the received parameter value:

- `^`: starts with the following characters, doesn't have characters before.
- `fixedquery`: has the exact value `fixedquery`.
- `$`: ends there, doesn't have any more characters after `fixedquery`.

If you feel lost with all these **"regular expression"** ideas, don't worry. They are a hard topic for many people. You can still do a lot of stuff without needing regular expressions yet.

But whenever you need them and go and learn them, know that you can already use them directly in **FastAPI**.

## Default values

The same way that you can pass `None` as the value for the `default` parameter, you can pass other values.

Let's say that you want to declare the `q` query parameter to have a `min_length` of `3`, and to have a default value of `"fixedquery"`:

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str = Query(default="fixedquery", min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

> **Note**
>
> Having a default value also makes the parameter optional.

## Make it required

When we don't need to declare more validations or metadata, we can make the `q` query parameter required just by not declaring a default value, like:

```
q: str
```

instead of:

```
q: Union[str, None] = None
```

But we are now declaring it with `Query`, for example like:

```
q: Union[str, None] = Query(default=None, min_length=3)
```

So, when you need to declare a value as required while using `Query`, you can simply not declare a default value:

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str = Query(min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

## Required with Ellipsis ( `...` )

There's an alternative way to explicitly declare that a value is required. You can set the `default` parameter to the literal value `...`:

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str = Query(default=..., min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Info**

If you hadn't seen that `...` before: it is a special single value, it is part of Python and is called "Ellipsis" [↪].

It is used by Pydantic and FastAPI to explicitly declare that a value is required.

This will let **FastAPI** know that this parameter is required.

## Required with `None`

You can declare that a parameter can accept `None`, but that it's still required. This would force clients to send a value, even if the value is `None`.

To do that, you can declare that `None` is a valid type but still use `default=...`:

**Python 3.6 and above**

```Python
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[str, None] = Query(default=..., min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Python 3.10 and above**

```Python
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=..., min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Tip**

Pydantic, which is what powers all the data validation and serialization in FastAPI, has a special behavior when you use `Optional` or `Union[Something, None]` without a default value, you can read more about it in the Pydantic docs about Required Optional fields [↪].

## Use Pydantic's `Required` instead of Ellipsis (`...`)

If you feel uncomfortable using `...`, you can also import and use `Required` from Pyd

```python
from fastapi import FastAPI, Query
from pydantic import Required

app = FastAPI()


@app.get("/items/")
async def read_items(q: str = Query(default=Required, min_length=3)):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

> **Tip**
>
> Remember that in most of the cases, when something is required, you can simply omit the `default` parameter, so you normally don't have to use `...` nor `Required`.

## Query parameter list / multiple values

When you define a query parameter explicitly with `Query` you can also declare it to receive a list of values, or said in other way, to receive multiple values.

For example, to declare a query parameter `q` that can appear multiple times in the URL, you can write:

**Python 3.6 and above**

```python
from typing import List, Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[List[str], None] = Query(default=None)):
    query_items = {"q": q}
    return query_items
```

**Python 3.9 and above**

```python
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()
```

```
@app.get("/items/")
async def read_items(q: Union[list[str], None] = Query(default=None)):
    query_items = {"q": q}
    return query_items
```

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: list[str] | None = Query(default=None)):
    query_items = {"q": q}
    return query_items
```

Then, with a URL like:

```
http://localhost:8000/items/?q=foo&q=bar
```

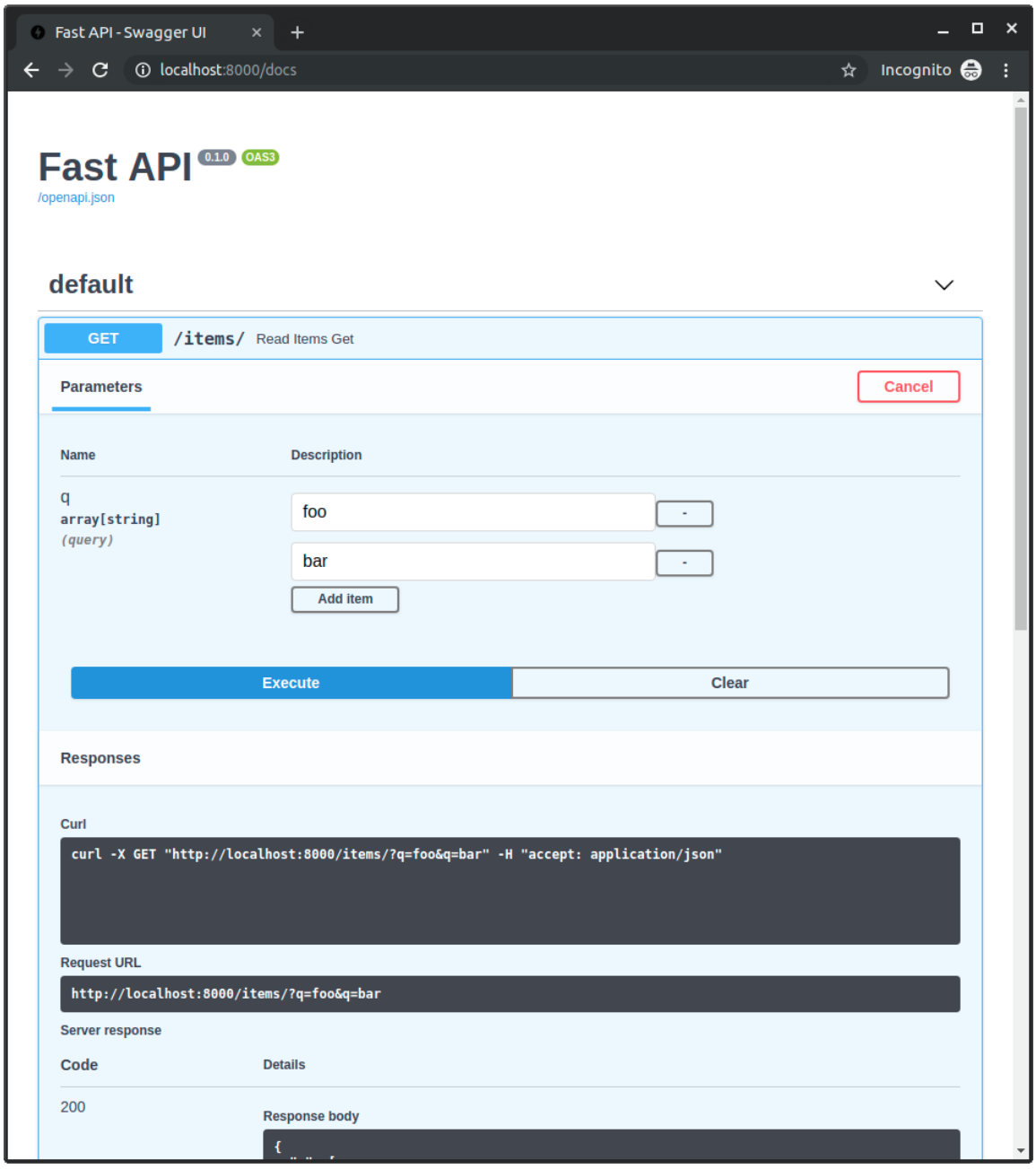
you would receive the multiple `q` *query parameters*' values ( `foo` and `bar` ) in a Python `list` inside your *path operation function*, in the *function parameter* `q` .

So, the response to that URL would be:

```
{
  "q": [
    "foo",
    "bar"
  ]
}
```

> **Tip**
>
> To declare a query parameter with a type of `list` , like in the example above, you need to explicitly use `Query` , otherwise it would be interpreted as a request body.

The interactive API docs will update accordingly, to allow multiple values:




## Query parameter list / multiple values with defaults

And you can also define a default `list` of values if none are provided:

**Python 3.6 and above**

```
from typing import List

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
```

```
async def read_items(q: List[str] = Query(default=["foo", "bar"])):
    query_items = {"q": q}
    return query_items
```

**Python 3.9 and above**

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: list[str] = Query(default=["foo", "bar"])):
    query_items = {"q": q}
    return query_items
```

If you go to:

```
http://localhost:8000/items/
```

the default of `q` will be: `["foo", "bar"]` and your response will be:

```
{
  "q": [
    "foo",
    "bar"
  ]
}
```

### Using `list`

You can also use `list` directly instead of `List[str]` (or `list[str]` in Python 3.9+):

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: list = Query(default=[])):
    query_items = {"q": q}
    return query_items
```

> **Note**
>
> Have in mind that in this case, FastAPI won't check the contents of the list.
>
> For example, `List[int]` would check (and document) that the contents of the list are integers. But `list` alone wouldn't.

## Declare more metadata

You can add more information about the parameter.

That information will be included in the generated OpenAPI and used by the documentation user interfaces and external tools.

> **Note**
>
> Have in mind that different tools might have different levels of OpenAPI support.
>
> Some of them might not show all the extra information declared yet, although in most of the cases, the missing feature is already planned for development.

You can add a `title`:

**Python 3.6 and above**

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(default=None, title="Query string",
min_length=3)
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Python 3.10 and above**

```
from fastapi import FastAPI, Query

app = FastAPI()
```

```
@app.get("/items/")
async def read_items(
    q: str | None = Query(default=None, title="Query string", min_length=3)
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

And a `description`:

**Python 3.6 and above**

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(
        default=None,
        title="Query string",
        description="Query string for the items to search in the database that have a good match",
        min_length=3,
    )
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Python 3.10 and above**

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: str
    | None = Query(
        default=None,
        title="Query string",
        description="Query string for the items to search in the database that have a good match",
        min_length=3,
    )
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
```

```
    if q:
        results.update({"q": q})
    return results
```

## Alias parameters

Imagine that you want the parameter to be `item-query`.

Like in:

```
http://127.0.0.1:8000/items/?item-query=foobaritems
```

But `item-query` is not a valid Python variable name.

The closest would be `item_query`.

But you still need it to be exactly `item-query`...

Then you can declare an `alias`, and that alias is what will be used to find the parameter value:

**Python 3.6 and above**

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: Union[str, None] = Query(default=None, alias="item-
query")):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Python 3.10 and above**

```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(q: str | None = Query(default=None, alias="item-query")):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

## Deprecating parameters

Now let's say you don't like this parameter anymore.

You have to leave it there a while because there are clients using it, but you want the docs to clearly show it as deprecated.

Then pass the parameter `deprecated=True` to `Query`:

**Python 3.6 and above**

```
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: Union[str, None] = Query(
        default=None,
        alias="item-query",
        title="Query string",
        description="Query string for the items to search in the database that
have a good match",
        min_length=3,
        max_length=50,
        regex="^fixedquery$",
        deprecated=True,
    )
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

**Python 3.10 and above**

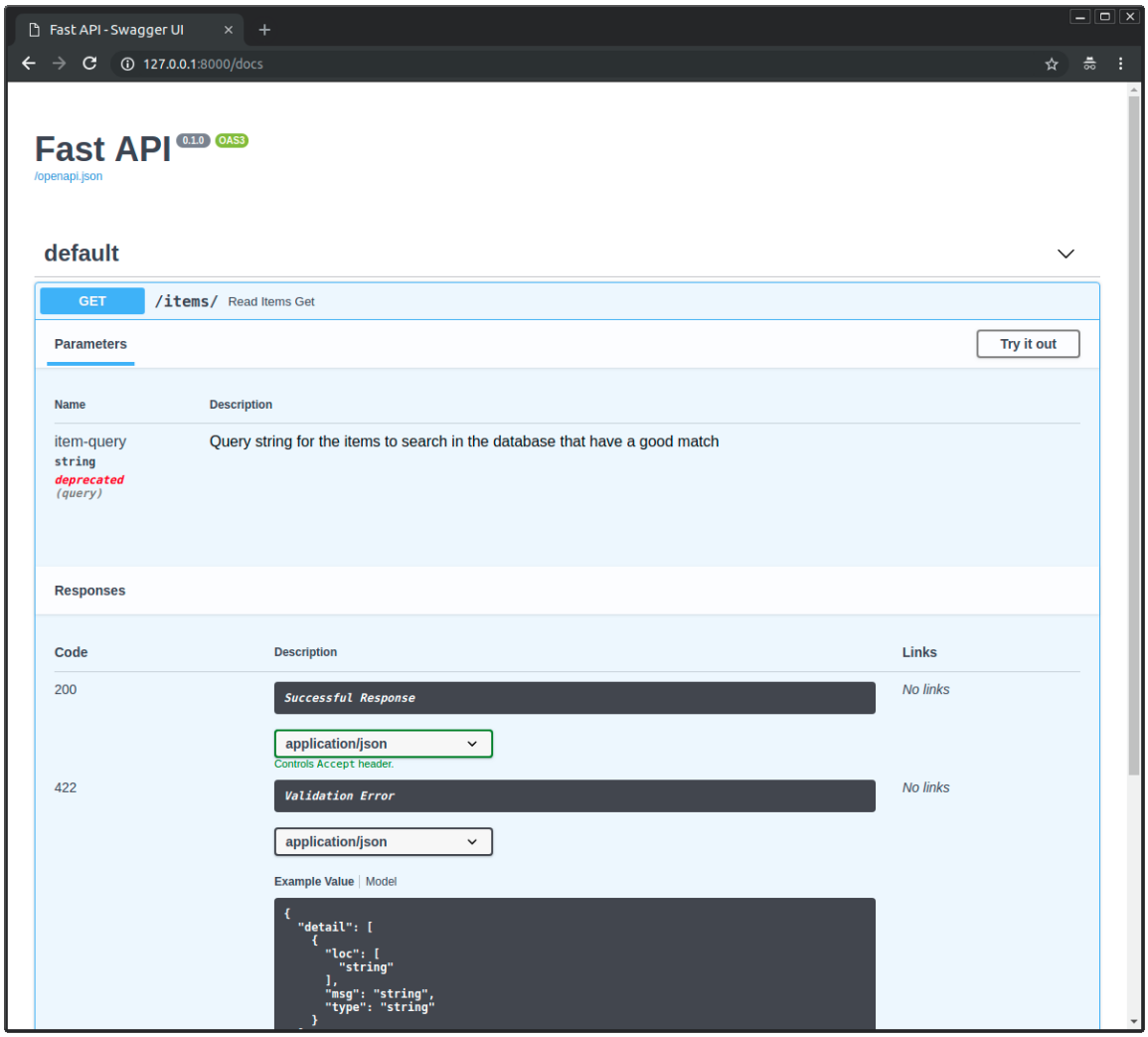
```
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    q: str
    | None = Query(
        default=None,
        alias="item-query",
        title="Query string",
        description="Query string for the items to search in the databas
have a good match",
        min_length=3,
```

```
        max_length=50,
        regex="^fixedquery$",
        deprecated=True,
    )
):
    results = {"items": [{"item_id": "Foo"}, {"item_id": "Bar"}]}
    if q:
        results.update({"q": q})
    return results
```

The docs will show it like this:




## Exclude from OpenAPI

To exclude a query parameter from the generated OpenAPI schema (and thus, from the automatic documentation systems), set the parameter `include_in_schema` of `Query` to `False`:

### Python 3.6 and above

```python
from typing import Union

from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    hidden_query: Union[str, None] = Query(default=None, include_in_schema=False)
):
    if hidden_query:
        return {"hidden_query": hidden_query}
    else:
        return {"hidden_query": "Not found"}
```

Python 3.10 and above

```python
from fastapi import FastAPI, Query

app = FastAPI()


@app.get("/items/")
async def read_items(
    hidden_query: str | None = Query(default=None, include_in_schema=False)
):
    if hidden_query:
        return {"hidden_query": hidden_query}
    else:
        return {"hidden_query": "Not found"}
```

## Recap

You can declare additional validations and metadata for your parameters.

Generic validations and metadata:

- `alias`
- `title`
- `description`
- `deprecated`

Validations specific for strings:

- `min_length`
- `max_length`
- `regex`

In these examples you saw how to declare validations for `str` values.

See the next chapters to see how to declare validations for other types, like numbers.