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 hl_lines="9"
{!> ../../../docs_src/query_params_str_validations/tutorial001.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="7"
{!> ../../../docs_src/query_params_str_validations/tutorial001_py310.py!}
```

The query parameter `q` is of type `Optional[str]` (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 `Optional` in `Optional[str]` is not used by FastAPI, but will allow your editor to give
```

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"

```Python hl_lines="3"
{!> ../../../docs_src/query_params_str_validations/tutorial002.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="1"
{!> ../../../docs_src/query_params_str_validations/tutorial002_py310.py!}
```

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"

```Python hl_lines="9"
{!> ../../../docs_src/query_params_str_validations/tutorial002.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="7"
{!> ../../../docs_src/query_params_str_validations/tutorial002_py310.py!}
```

As we have to replace the default value `None` with `Query(None)`, the first parameter to `Query` serves the same purpose of defining that default value.

So:

```
q: Optional[str] = Query(None)
```

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

```
q: Optional[str] = None
```

And in Python 3.10 and above:

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

```Python
= None
```

or the:

```Python
= Query(None)
```

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

The `Optional` part allows your editor to provide better support, but it is not what tells F

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

```
q: str = Query(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"

```Python hl_lines="9"
{!> ../../../docs_src/query_params_str_validations/tutorial003.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="7"
{!> ../../../docs_src/query_params_str_validations/tutorial003_py310.py!}
```

Add regular expressions

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

=== "Python 3.6 and above"

```Python hl_lines="10"
{!> ../../../docs_src/query_params_str_validations/tutorial004.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="8"
{!> ../../../docs_src/query_params_str_validations/tutorial004_py310.py!}
```

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 first argument to be used as the default value, 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"`:

Python hl_lines="7" {!../../../docs_src/query_params_str_validations/tutorial005.py!}

!!! 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: Optional[str] = None
```

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

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

So, when you need to declare a value as required while using `Query`, you can use `...` as the first argument:

Python hl_lines="7" {!../../../docs_src/query_params_str_validations/tutorial006.py!}

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

This will let **FastAPI** know that this parameter is 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 hl_lines="9"
{!> ../../../docs_src/query_params_str_validations/tutorial011.py!}
```
```

=== "Python 3.9 and above"

```Python hl_lines="9"
{!> ../../../docs_src/query_params_str_validations/tutorial011_py39.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="7"
{!> ../../../docs_src/query_params_str_validations/tutorial011_py310.py!}
```

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:

```JSON
{
  "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"

```Python hl_lines="9"
{!> ../../../docs_src/query_params_str_validations/tutorial012.py!}
```

=== "Python 3.9 and above"

```Python hl_lines="7"
{!> ../../../docs_src/query_params_str_validations/tutorial012_py39.py!}
```

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+):

```
Python hl_lines="7" {!../../../docs_src/query_params_str_validations/tutorial013.py!}
```

!!! 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 intege
```

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
```

You can add a `title`:

=== "Python 3.6 and above"

```
```Python hl_lines="10"
{!> ../../../docs_src/query_params_str_validations/tutorial007.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="7"
{!> ../../../docs_src/query_params_str_validations/tutorial007_py310.py!}
```
```

And a `description`:

=== "Python 3.6 and above"

```
```Python hl_lines="13"
{!> ../../../docs_src/query_params_str_validations/tutorial008.py!}
```
```

=== "Python 3.10 and above"

```Python hl_lines="12"
{!> ../../../docs_src/query_params_str_validations/tutorial008_py310.py!}
```

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"

```Python hl_lines="9"
{!> ../../../docs_src/query_params_str_validations/tutorial009.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="7"
{!> ../../../docs_src/query_params_str_validations/tutorial009_py310.py!}
```

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"

```Python hl_lines="18"
{!> ../../../docs_src/query_params_str_validations/tutorial010.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="17"
{!> ../../../docs_src/query_params_str_validations/tutorial010_py310.py!}
```

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 hl_lines="10"
{!> ../../../docs_src/query_params_str_validations/tutorial014.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="7"
{!> ../../../docs_src/query_params_str_validations/tutorial014_py310.py!}
```

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.