Request Body

When you need to send data from a client (let's say, a browser) to your API, you send it as a **request body**.

A **request** body is data sent by the client to your API. A **response** body is the data your API sends to the client.

Your API almost always has to send a **response** body. But clients don't necessarily need to send **request** bodies all the time.

To declare a **request** body, you use Pydantic models with all their power and benefits.

!!! info To send data, you should use one of: `POST` (the more common), `PUT` , `DELETE` or `PATCH` .

```
Sending a body with a `GET` request has an undefined behavior in the specifications,
nevertheless, it is supported by FastAPI, only for very complex/extreme use cases.

As it is discouraged, the interactive docs with Swagger UI won't show the
documentation for the body when using `GET`, and proxies in the middle might not
support it.
```

Import Pydantic's `BaseModel`

First, you need to import `BaseModel` from `pydantic` :

=== "Python 3.6 and above"

```
```Python hl_lines="4"
{!> ../../../docs_src/body/tutorial001.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="2"
{!> ../../../docs_src/body/tutorial001_py310.py!}
```
```

Create your data model

Then you declare your data model as a class that inherits from `BaseModel` .

Use standard Python types for all the attributes:

=== "Python 3.6 and above"

```
```Python hl_lines="7-11"
{!> ../../../docs_src/body/tutorial001.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="5-9"
{!> ../../../docs_src/body/tutorial001_py310.py!}
```
```

```
```

The same as when declaring query parameters, when a model attribute has a default value, it is not required.
Otherwise, it is required. Use `None` to make it just optional.

For example, this model above declares a JSON "`object`" (or Python `dict`) like:

```
{
    "name": "Foo",
    "description": "An optional description",
    "price": 45.2,
    "tax": 3.5
}
```

...as `description` and `tax` are optional (with a default value of `None`), this JSON "`object`" would also be
valid:

```
{
    "name": "Foo",
    "price": 45.2
}
```

Declare it as a parameter

To add it to your *path operation*, declare it the same way you declared path and query parameters:

=== "Python 3.6 and above"

```
```Python hl_lines="18"
{!> ../../../docs_src/body/tutorial001.py!}
```
```

=== "Python 3.10 and above"

```
```Python hl_lines="16"
{!> ../../../docs_src/body/tutorial001_py310.py!}
```
```

...and declare its type as the model you created, `Item`.

Results

With just that Python type declaration, **FastAPI** will:

- Read the body of the request as JSON.
- Convert the corresponding types (if needed).
- Validate the data.
  - If the data is invalid, it will return a nice and clear error, indicating exactly where and what was the
    incorrect data.
- Give you the received data in the parameter `item`.

- As you declared it in the function to be of type `Item`, you will also have all the editor support (completion, etc) for all of the attributes and their types.
- Generate [JSON Schema](#) definitions for your model, you can also use them anywhere else you like if it makes sense for your project.
- Those schemas will be part of the generated OpenAPI schema, and used by the automatic documentation UIs.

Automatic docs

The JSON Schemas of your models will be part of your OpenAPI generated schema, and will be shown in the interactive API docs:

And will be also used in the API docs inside each *path operation* that needs them:

Editor support

In your editor, inside your function you will get type hints and completion everywhere (this wouldn't happen if you received a `dict` instead of a Pydantic model):

You also get error checks for incorrect type operations:

This is not by chance, the whole framework was built around that design.

And it was thoroughly tested at the design phase, before any implementation, to ensure it would work with all the editors.

There were even some changes to Pydantic itself to support this.

The previous screenshots were taken with [Visual Studio Code](#).

But you would get the same editor support with [PyCharm](#) and most of the other Python editors:

!!! tip If you use [PyCharm](#) as your editor, you can use the [Pydantic PyCharm Plugin](#).

```
It improves editor support for Pydantic models, with:

* auto-completion
* type checks
* refactoring
* searching
* inspections
```

Use the model

Inside of the function, you can access all the attributes of the model object directly:

=== "Python 3.6 and above"

```Python hl_lines="21"
{!> ../../../docs_src/body/tutorial002.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="19"
{!> ../../../docs_src/body/tutorial002_py310.py!}
```

Request body + path parameters

You can declare path parameters and request body at the same time.

FastAPI will recognize that the function parameters that match path parameters should be **taken from the path**, and that function parameters that are declared to be Pydantic models should be **taken from the request body**.

=== "Python 3.6 and above"

```Python hl_lines="17-18"
{!> ../../../docs_src/body/tutorial003.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="15-16"
{!> ../../../docs_src/body/tutorial003_py310.py!}
```

Request body + path + query parameters

You can also declare **body**, **path** and **query** parameters, all at the same time.

FastAPI will recognize each of them and take the data from the correct place.

=== "Python 3.6 and above"

```Python hl_lines="18"
{!> ../../../docs_src/body/tutorial004.py!}
```

=== "Python 3.10 and above"

```Python hl_lines="16"
{!> ../../../docs_src/body/tutorial004_py310.py!}
```

The function parameters will be recognized as follows:

- If the parameter is also declared in the **path**, it will be used as a path parameter.
- If the parameter is of a **singular type** (like `int`, `float`, `str`, `bool`, etc) it will be interpreted as a **query** parameter.
- If the parameter is declared to be of the type of a **Pydantic model**, it will be interpreted as a request **body**.

!!! 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 you better support and detect errors.
```

Without Pydantic

If you don't want to use Pydantic models, you can also use **Body** parameters. See the docs for [Body - Multiple Parameters: Singular values in body](){.internal-link target=_blank}.