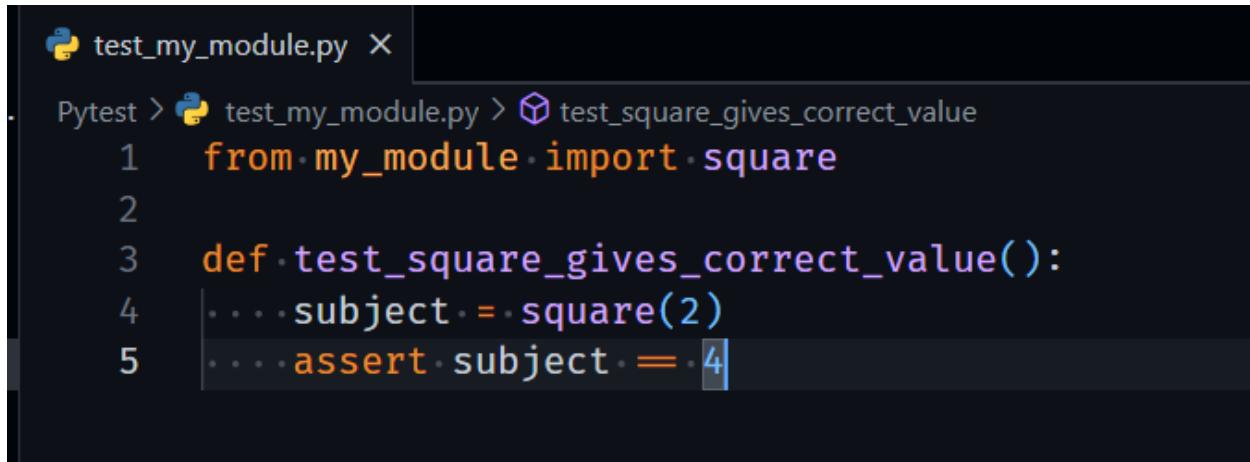
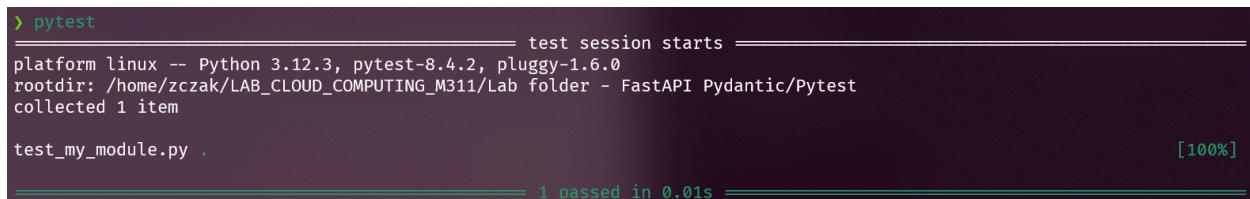


Lab FastAPI Pydantic - Cloud Computing - M311

1- Pytest



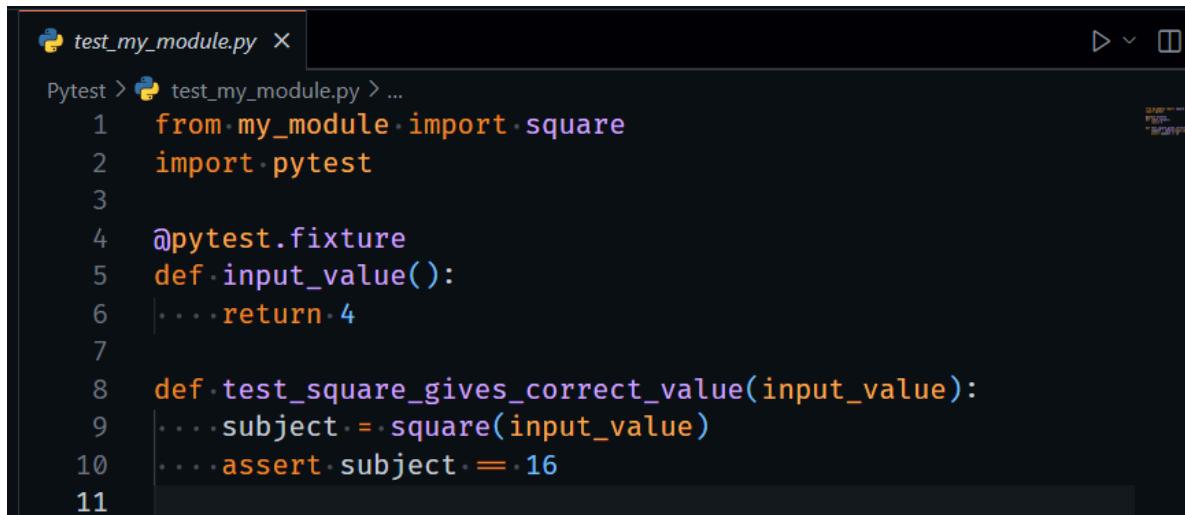
```
test_my_module.py X
Pytest > test_my_module.py > test_square_gives_correct_value
1     from my_module import square
2
3     def test_square_gives_correct_value():
4         subject = square(2)
5         assert subject == 4
```



```
> pytest
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: /home/zczak/LAB_CLOUD_COMPUTING_M311/Lab folder - FastAPI Pydantic/Pytest
collected 1 item

test_my_module.py . [100%]

===== 1 passed in 0.01s =====
```

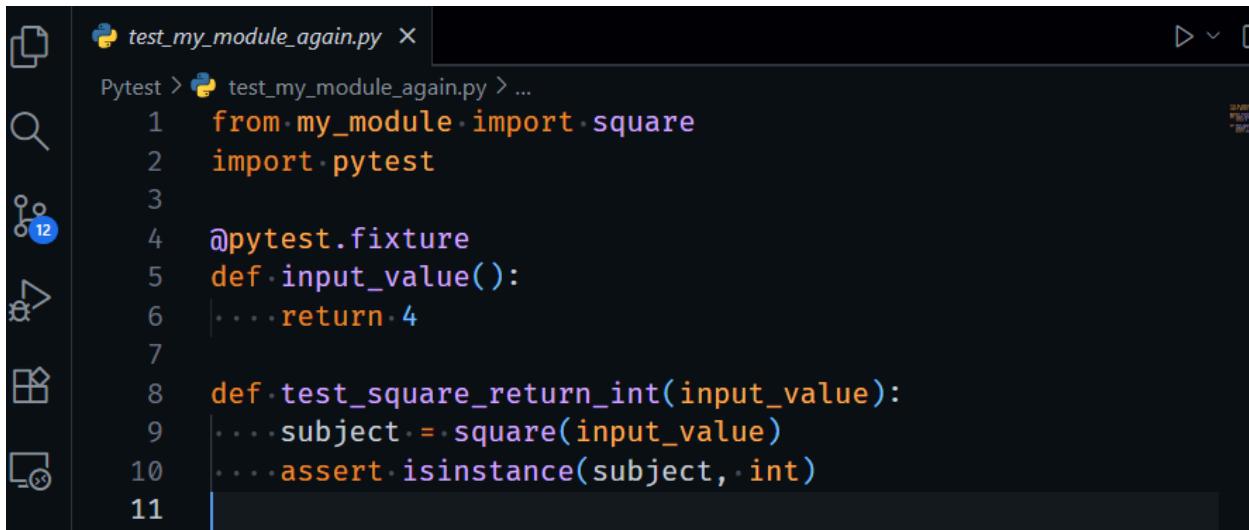


```
test_my_module.py X
Pytest > test_my_module.py > ...
1     from my_module import square
2     import pytest
3
4     @pytest.fixture
5     def input_value():
6         return 4
7
8     def test_square_gives_correct_value(input_value):
9         subject = square(input_value)
10        assert subject == 16
```

```
> pytest
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: /home/zczak/LAB_CLOUD_COMPUTING_M311/Lab folder - FastAPI Pydantic
collected 1 item

Pytest/test_my_module.py .

===== 1 passed in 0.01s =====
```



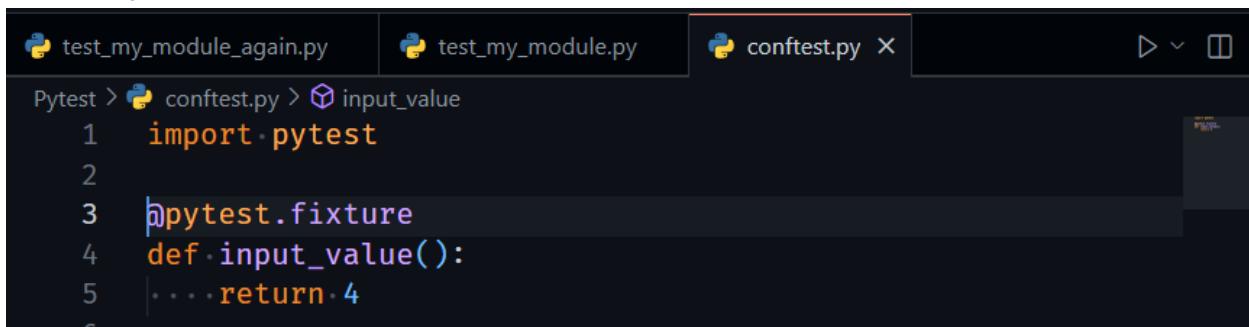
```
1  from my_module import square
2  import pytest
3
4  @pytest.fixture
5  def input_value():
6      return 4
7
8  def test_square_return_int(input_value):
9      subject = square(input_value)
10     assert isinstance(subject, int)
11
```

```
> pytest
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: /home/zczak/LAB_CLOUD_COMPUTING_M311/Lab folder - FastAPI Pydantic
collected 2 items

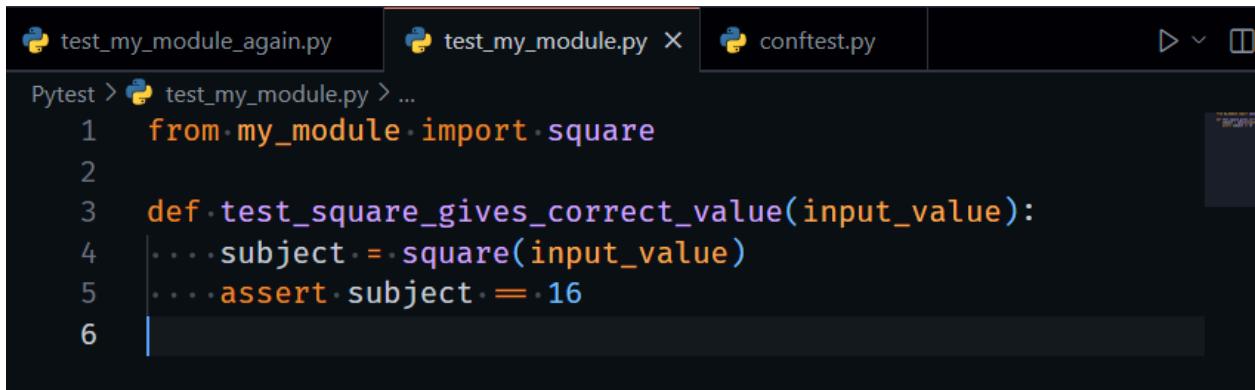
Pytest/test_my_module.py .
Pytest/test_my_module_again.py .

===== 2 passed in 0.01s =====
```

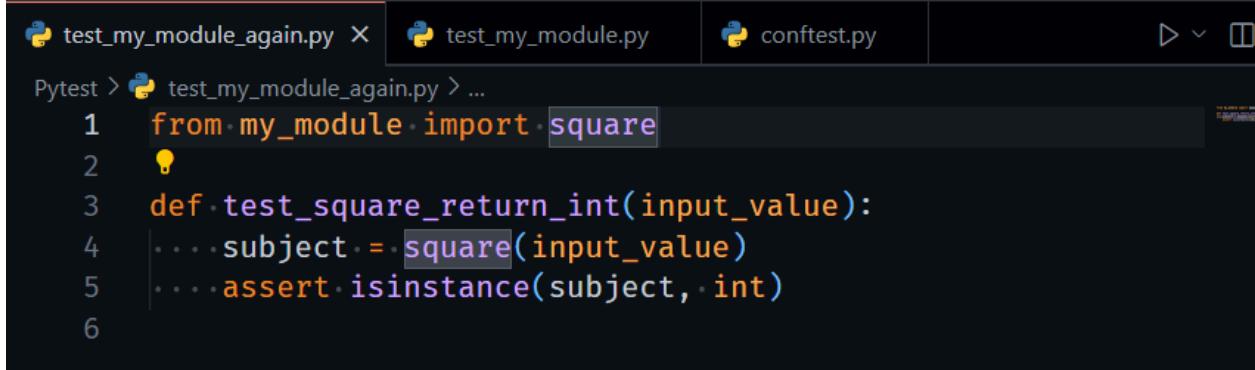
### conftest.py



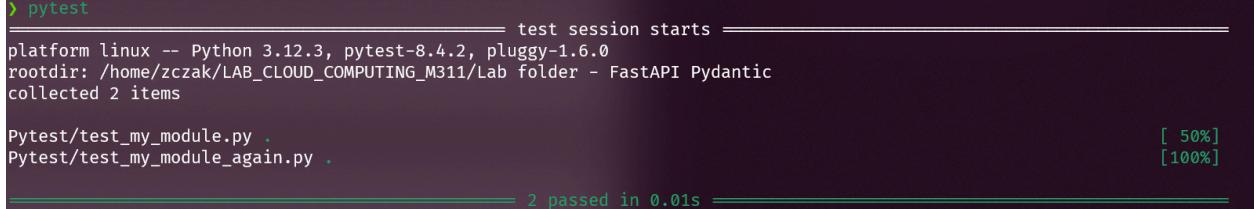
```
1  import pytest
2
3  @pytest.fixture
4  def input_value():
5      return 4
6
```



```
Pytest > 🐍 test_my_module.py > ...
1   from my_module import square
2
3   def test_square_gives_correct_value(input_value):
4       subject = square(input_value)
5       assert subject == 16
6
```



```
Pytest > 🐍 test_my_module_again.py > ...
1   from my_module import square
2
3   def test_square_return_int(input_value):
4       subject = square(input_value)
5       assert isinstance(subject, int)
6
```

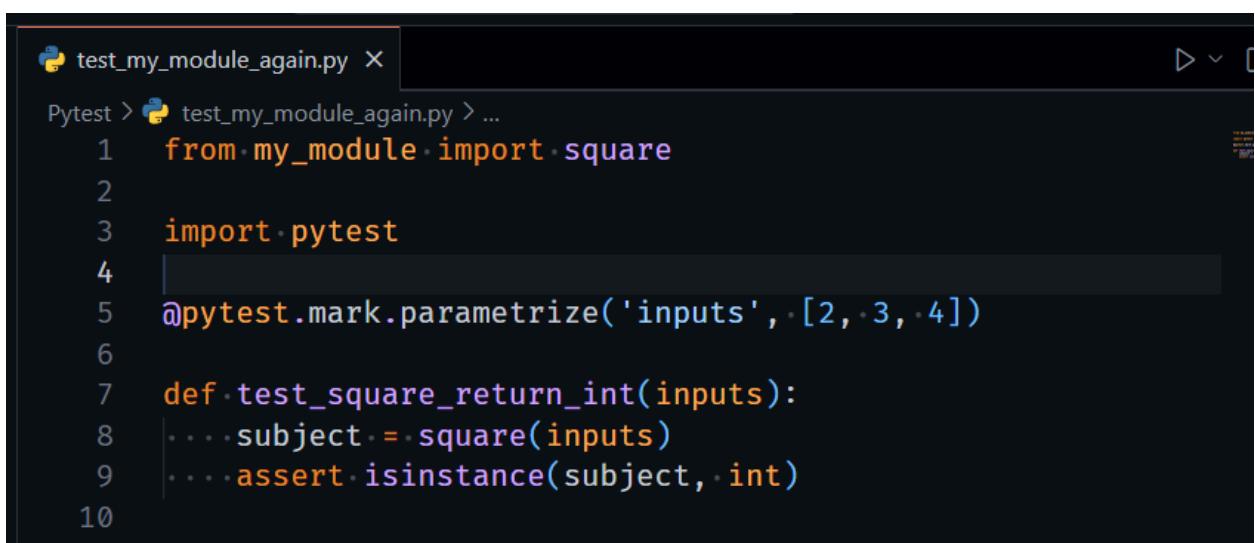


```
> pytest
===== test session starts =====
platform linux -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: /home/zczak/LAB_CLOUD_COMPUTING_M311/Lab folder - FastAPI Pydantic
collected 2 items

Pytest/test_my_module.py .
Pytest/test_my_module_again.py .

===== 2 passed in 0.01s =====
```

## Parametrized tests



```
Pytest > 🐍 test_my_module_again.py > ...
1   from my_module import square
2
3   import pytest
4
5   @pytest.mark.parametrize('inputs', [2, 3, 4])
6
7   def test_square_return_int(inputs):
8       subject = square(inputs)
9       assert isinstance(subject, int)
10
```

```
> pytest
platform linux -- Python 3.12.3, pytest-8.4.2, pluggy-1.6.0
rootdir: /home/zczak/LAB_CLOUD_COMPUTING_M311/Lab folder - FastAPI Pydantic
collected 4 items

Pytest/test_my_module.py .
Pytest/test_my_module_again.py ...

===== 4 passed in 0.01s =====
```

## 2- Pydantic

The screenshot shows a code editor with a file named `model.py` open. The code defines a `User` class using Pydantic's `BaseModel`. It includes attributes `id`, `name`, `signup_ts`, and `friends`. An external data dictionary is defined with values `'id': '123'`, `'signup_ts': '2019-06-01 12:22'`, and `'friends': [1, 2, '5']`. The `user` variable is initialized with `User(**external_data)`, and the `print(user.id)` statement is executed.

```
model.py > ...
2  from typing import List, Optional
3  from pydantic import BaseModel
4
5  class User(BaseModel):
6      id: int
7      name: str = 'John Doe'
8      signup_ts: Optional[datetime] = None
9      friends: List[int] = []
10
11     external_data = {
12         'id': '123',
13         'signup_ts': '2019-06-01 12:22',
14         'friends': [1, 2, '5'],
15     }
16
17     user = User(**external_data)
18     print(user.id)
```

In the terminal below, the command `python3 model.py` is run, and the output `123` is shown.

```
> python3 model.py
123
```

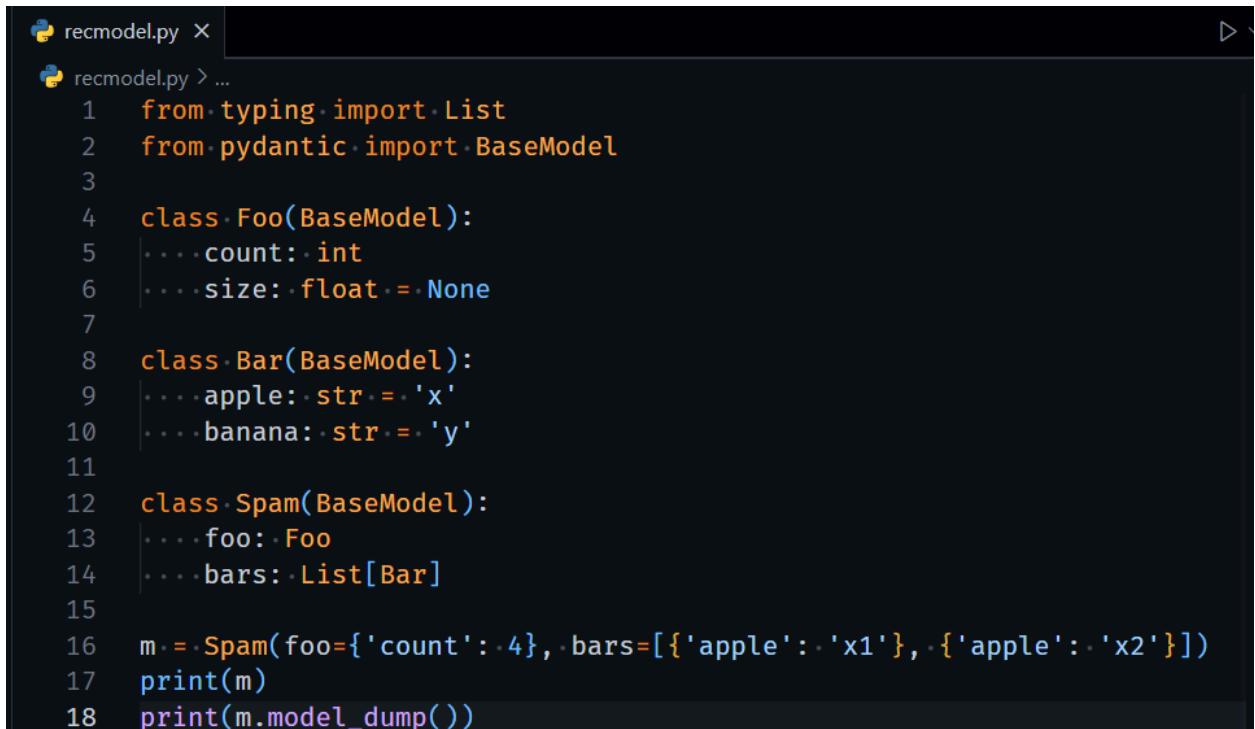
If we change '5' to 'a' which cannot be converted to int:

```
external_data = {
    'id': '123',
    'signup_ts': '2019-06-01 12:22',
    'friends': [1, 2, 'a'],
}
```

Throws an error

```
> python3 model.py
Traceback (most recent call last):
  File "/home/zczak/LAB_CLOUD_COMPUTING_M311/Lab folder - FastAPI Pydantic/model.py", line 17, in <module>
    user = User(**external_data)
           ^^^^^^^^^^^^^^^^^^
  File "/home/zczak/LAB_CLOUD_COMPUTING_M311/Lab folder - FastAPI Pydantic/venv/lib/python3.12/site-packages/pydantic/m
in.py", line 250, in __init__
    validated_self = self.__pydantic_validator__.validate_python(data, self_instance=self)
                      ^^^^^^
pydantic_core._pydantic_core.ValidationError: 1 validation error for User
friends.2
  Input should be a valid integer, unable to parse string as an integer [type=int_parsing, input_value='a', input_type=s
tr]
  For further information visit https://errors.pydantic.dev/2.12/v/int_parsing
```

## Recursive models:

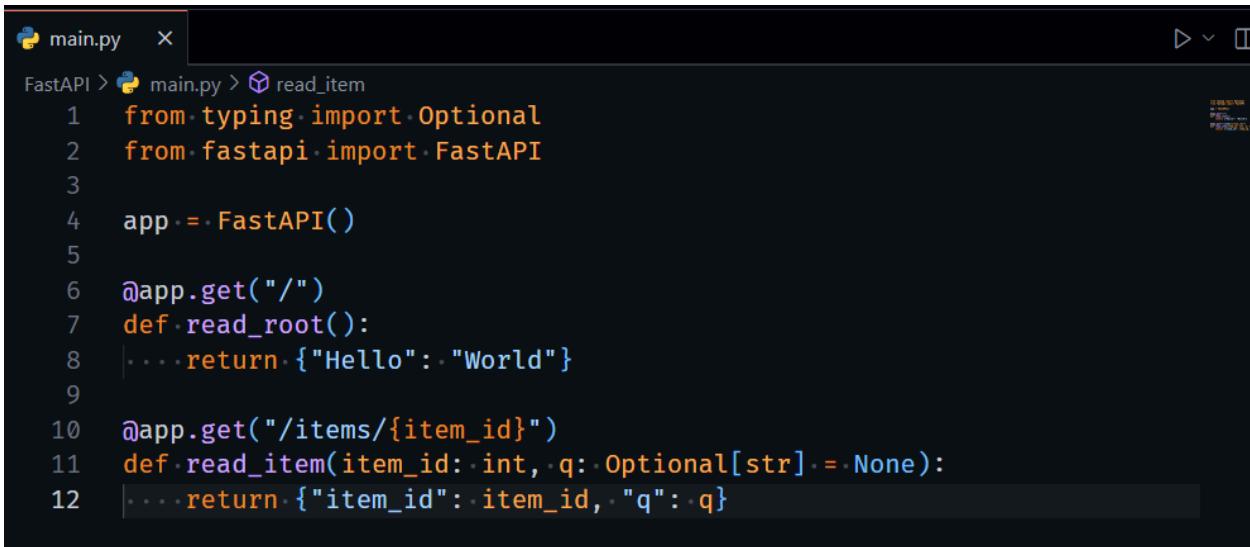


```
recremodel.py > ...
1  from typing import List
2  from pydantic import BaseModel
3
4  class Foo(BaseModel):
5      count: int
6      size: float = None
7
8  class Bar(BaseModel):
9      apple: str = 'x'
10     banana: str = 'y'
11
12 class Spam(BaseModel):
13     foo: Foo
14     bars: List[Bar]
15
16 m = Spam(foo={'count': 4}, bars=[{'apple': 'x1'}, {'apple': 'x2'}])
17 print(m)
18 print(m.model_dump())
```

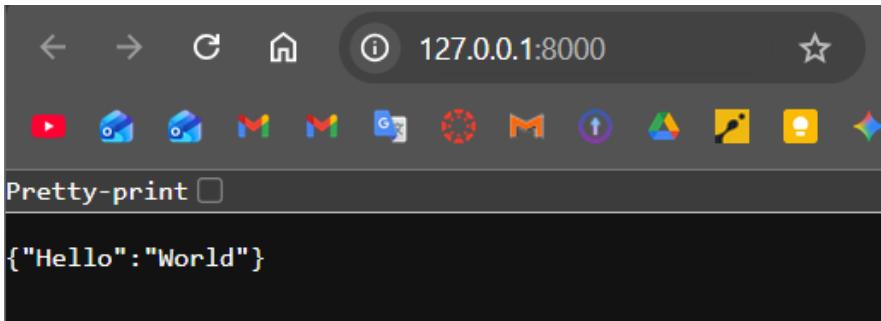
```
> python3 recremodel.py
foo=Foo(count=4, size=None) bars=[Bar(apple='x1', banana='y'), Bar(apple='x2', banana='y')]
{'foo': {'count': 4, 'size': None}, 'bars': [{'apple': 'x1', 'banana': 'y'}, {'apple': 'x2', 'banana': 'y'}]}
```

Validators are deprecated in Pydantic

### 3- FastAPI



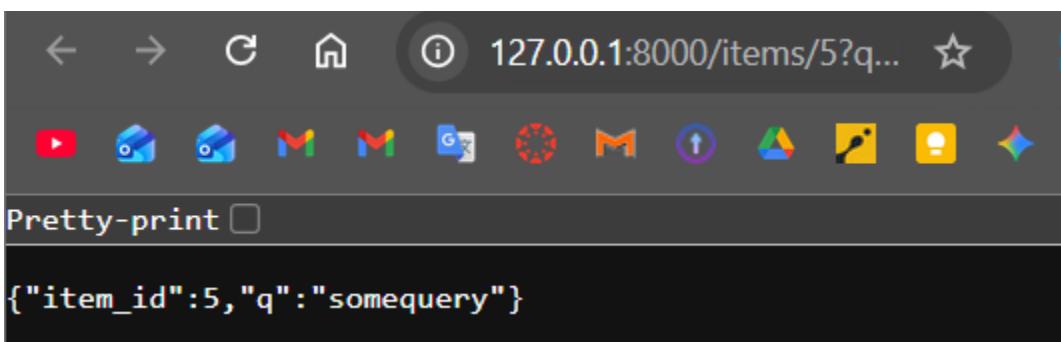
```
main.py
FastAPI > main.py > read_item
1  from typing import Optional
2  from fastapi import FastAPI
3
4  app = FastAPI()
5
6  @app.get("/")
7  def read_root():
8      return {"Hello": "World"}
9
10 @app.get("/items/{item_id}")
11 def read_item(item_id: int, q: Optional[str] = None):
12     return {"item_id": item_id, "q": q}
```



127.0.0.1:8000

Pretty-print

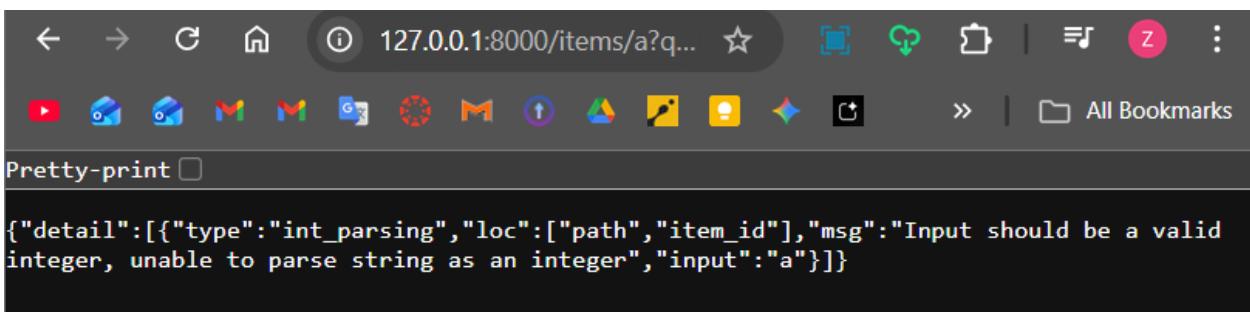
```
{"Hello": "World"}
```



127.0.0.1:8000/items/5?q...

Pretty-print

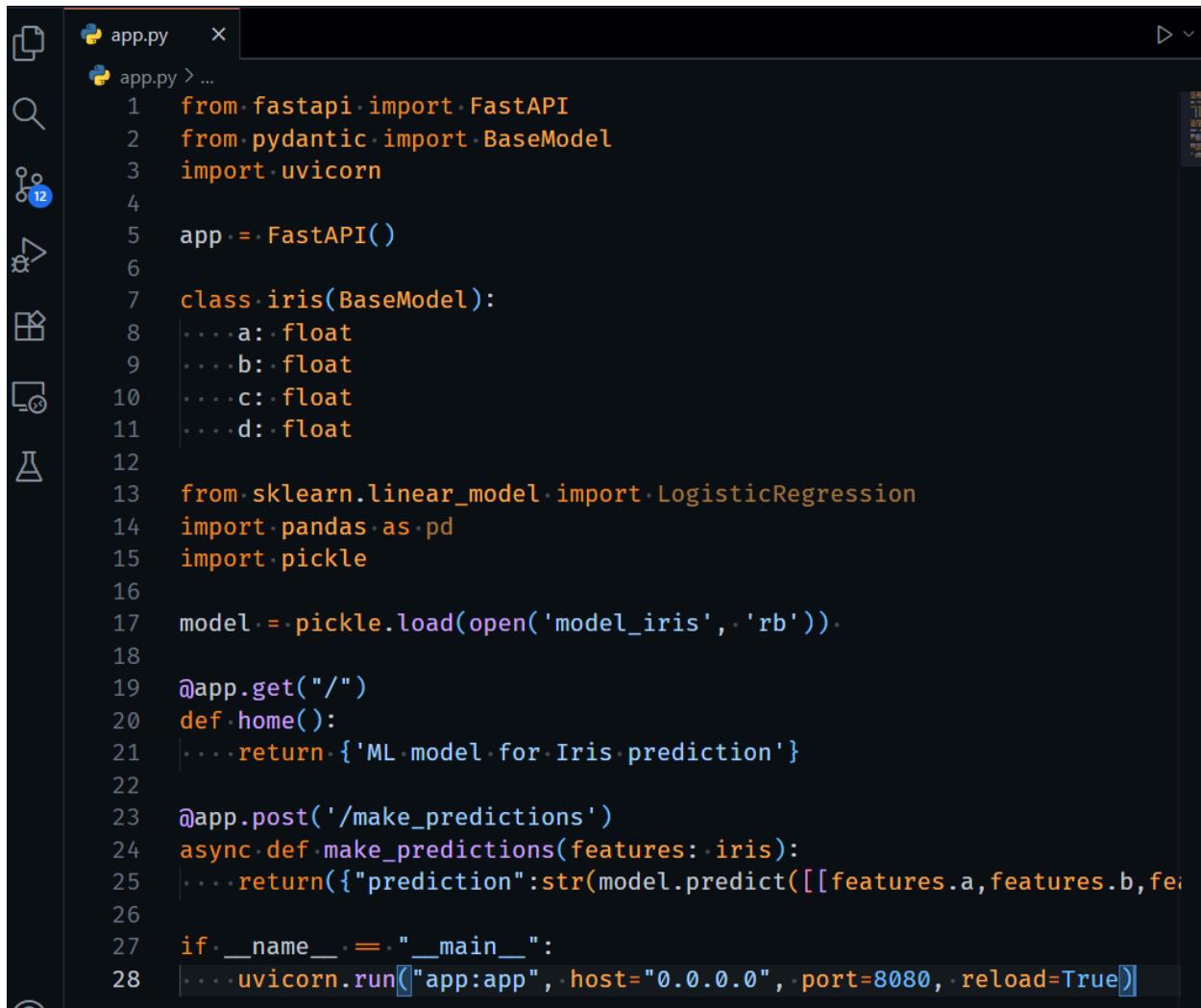
```
{"item_id": 5, "q": "somequery"}
```



127.0.0.1:8000/items/a?q...

Pretty-print

```
{"detail": [{"type": "int_parsing", "loc": ["path", "item_id"], "msg": "Input should be a valid integer, unable to parse string as an integer", "input": "a"}]}
```



The screenshot shows a code editor window with a dark theme. The file being edited is `app.py`. The code implements a FastAPI application that loads a logistic regression model from a pickle file. It defines a `iris` Pydantic model for input features and a `/make_predictions` endpoint that returns a prediction based on the input features. The code also includes a `home` endpoint that returns a message about the ML model.

```
from fastapi import FastAPI
from pydantic import BaseModel
import uvicorn

app = FastAPI()

class iris(BaseModel):
    a: float
    b: float
    c: float
    d: float

from sklearn.linear_model import LogisticRegression
import pandas as pd
import pickle

model = pickle.load(open('model_iris', 'rb'))

@app.get("/")
def home():
    return {"ML model for Iris prediction"}

@app.post('/make_predictions')
async def make_predictions(features: iris):
    return {"prediction": str(model.predict([[features.a, features.b, features.c, features.d]]))}

if __name__ == "__main__":
    uvicorn.run("app:app", host="0.0.0.0", port=8080, reload=True)
```



The screenshot shows the FastAPI documentation interface. The top bar displays the title "FastAPI 0.1.0 (OpenAPI 3.1)" and a link to `/openapi.json`. Below the title, there is a "default" section. Under "default", there is a "GET /" entry. In the main area, there is a "POST /make\_predictions Make Predictions" entry. The "Parameters" section indicates "No parameters". The "Request body" section is marked as "required" and has a dropdown menu set to "application/json". Below the dropdown, there is an "Edit Value" field containing a JSON schema for the request body:

```
{ "a": 1, "b": 2, "c": 3, "d": 4 }
```

Server response

Code	Details
200	<p>Response body</p> <pre>{ "prediction": "2" }</pre> <p>Download</p> <p>Response headers</p> <pre>content-length: 18 content-type: application/json date: Sun, 02 Nov 2025 16:48:04 GMT server: uvicorn</pre>

To Do:

Furniture dataset

### Predict furniture price

Category:

Sellable Online:  Yes  No  
Other color:  Yes  No

Depth:

Length:

Width:

[Reset](#) [Predict](#)

### Prediction Result

**Predicted Price: \$366.69**

Category: 0  
Sellable Online: 1  
Other Colors: 1  
Depth: 30.0  
Height: 50.0  
Width: 100.0

[Back to Form](#)

Iris dataset

## Iris Flower Classification

Sepal Length (cm)

Sepal Width (cm)

Petal Length (cm)

Petal Width (cm)

**Classify**

## Classification Result

**Predicted Species: virginica**

Sepal Length: 5.0

Sepal Width: 5.0

Petal Length: 5.0

Petal Width: 5.0

[Back to Form](#)