

به نام خدا که یکتا

گزارش تمرین دوم درس پایگاه داده پیشرفته

نام و نام خانوادگی :

مبین رمضانی خرم آبادی 40311415016

نام استاد :

آقای دکتر آرمین رشنو

مهر ۱۴۰۳

سوال تمرین :

۱- برای هر کدام از تمرینات سری اول، با استفاده از FastAPI یک API ایجاد کنید. برای هر تمرین یک یا چند ورودی انتخاب کنید و از ۳ روش query parameter و path parameter و body ورودی را برای API ارسال کرده و پاسخ API را نمایش دهید.

حل سوال :

ایجاد یک API به وسیله FastAPI، یک سری ساختار و اصول اولیه و کلی دارد که ابتدا به توضیح آن پرداخته و سپس هر تمرین را با این API حل میکنیم. FastAPI از فریمورک FastAPI برای ساخت API استفاده می‌شود. BaseModel از کتابخانه pydantic برای تعریف بدنه درخواست (request body) استفاده می‌شود. ابتدا کتابخانه های لازم را ایمپورت میکنیم :

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

سپس با استفاده از خط زیر، یک اپلیکیشن FastAPI با نام app ایجاد می‌شود. این شیء، ورودی اصلی برای تعریف مسیرها و درخواست‌ها است. که در ادامه کار با آن در هر تمرین توضیح داده خواهد شد :

```
app = FastAPI()
```

برای اجرای سرور هم از دستور زیر استفاده میکنیم (فرض کنیم نام فایل API ما main.py باشد) :

```
fastapi dev main.py
```

در FastAPI، سه روش مختلف برای دریافت ورودی‌ها از کلاینت وجود دارد: Path Parameters, Query Parameters و Request Body.

✓ **روش Path Parameters :** در این روش، پارامترهای مسیر (هر تعداد که باشند) به‌طور مستقیم در URL قرار می‌گیرند. این روش برای مواردی استفاده می‌شود که مقدار ورودی جزئی از مسیر URL است، مثل دریافت اطلاعات یک کاربر خاص یا یک محصول بر اساس شناسه.

✓ **روش Query Parameters :** در این روش، پارامترهای کوئری بعد از علامت در URL قرار می‌گیرند و با استفاده از key=value تعریف می‌شوند. این روش برای زمانی مناسب است که بخواهید اطلاعات اضافی را از طریق URL ارسال کنید، مثل فیلتر کردن یا مرتب‌سازی نتایج یک جستجو.

✓ **روش Request Body :** در این روش، داده‌ها در بدنه درخواست (Request Body) ارسال می‌شوند. بدنه درخواست معمولاً در درخواست‌های POST، PUT یا PATCH استفاده می‌شود. در اینجا از POST برای دریافت یک عدد در قالب بدنه درخواست

استفاده شده است. این روش برای ارسال داده‌های پیچیده‌تر یا اطلاعات حساس که در URL قابل نمایش نیستند مناسب است، مثل اطلاعات ورود کاربر یا داده‌های فرم.

اکنون به توضیح هر کدام از تمرین‌ها با استفاده از این API می‌پردازیم.

حل سوال 1 :

```
1 """
2 1 - Write a program to get a number and print all its even digits separated by *.
3 Input: 822145635
4 Output: 6*4*2*2*8
5 """
6
7 from fastapi import FastAPI
8 from pydantic import BaseModel
9
10 app = FastAPI()
11
12
13 class NumberBody(BaseModel):
14     number: int = 0
15
16
17 def q1(number: int):
18     even_digits = [digit for digit in str(number) if int(digit) % 2 == 0]
19     return '*'.join(even_digits)
20
21
22 @app.get("/q1/{number}")
23 def question1_path(number: int):
24     result = q1(number)
25     return {"result with (path parameter)": result}
26
27
28 @app.get("/q1/")
29 def question1_query(number: int):
30     result = q1(number)
31     return {"result with (query parameter)": result}
32
33
34 @app.post("/q1/")
35 def question1_body(number: NumberBody):
36     result = q1(number.number)
37     return {"result with (body)": result}
```

در خطوط ۷ و ۸، کتابخانه‌های لازم ایمپورت شده‌اند. در خط ۱۰، یک شی از کلاس FastAPI ایجاد شده و تحت عنوان متغیری با نام app ذخیره می‌شود. خط ۱۳ و ۱۴، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می‌شود. این مدل یک عدد (number) از نوع int تعریف می‌کند که مقدار پیش‌فرض آن ۰ است.

در خطوط ۱۷ تا ۱۹، برنامه‌ای که ما در تمرین سری اول نوشتیم را تحت عنوان تابعی با نام q1 ذخیره کرده ایم تا در API استفاده کنیم.

در خطوط ۲۲ تا ۲۵، با استفاده از خط ۲۲، مشخص شده که این مسیر از نوع GET است و { number } یک (path parameter) است. این پارامتر مستقیماً در URL دریافت می‌شود. کاربر باید عدد مورد نظر خود را به‌طور مستقیم به‌عنوان بخشی از URL وارد کند. FastAPI به‌طور خودکار مقدار ورودی مسیر را از رشته به نوع داده‌ای که مشخص کرده‌ایم (در اینجا int) تبدیل می‌کند. به همین دلیل در خط ۲۳، number: int در امضای تابع استفاده شده است. بعد از انجام محاسبات، در خط ۲۵ نتیجه برگردانده می‌شود.

در خطوط ۲۸ تا ۳۱، با استفاده از خط ۲۸، این مسیر نیز از نوع GET است، اما در اینجا پارامتر عدد به‌صورت (Query Parameter) دریافت می‌شود، به این معنا که مقدار عدد بعد از علامت در URL وارد می‌شود. این پارامتر به‌صورت ...number=? در انتهای URL قرار می‌گیرد. در این حالت، FastAPI به‌طور خودکار این پارامتر کوئری را به متغیر number با نوع int تبدیل می‌کند. بعد از انجام محاسبات، در خط ۳۱ نتیجه برگردانده می‌شود.

در خطوط ۳۴ تا ۳۷، با استفاده از خط ۳۴، این مسیر از نوع POST است و داده‌ها از طریق (Request Body) به سرور ارسال می‌شوند. در اینجا از مدل NumberBody که از pydantic.BaseModel ارث‌بری کرده است، برای اعتبارسنجی و دریافت ورودی استفاده می‌شود. بدنه درخواست (body) باید به‌صورت JSON ارسال شود. این مدل به‌طور خودکار مقدار number را از JSON دریافت کرده و آن را به تابع q1 می‌فرستد. بعد از انجام محاسبات، در خط ۳۷ نتیجه برگردانده می‌شود.

بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q1/822145635
```

پاسخ سرور :

```
1 {
2   "result with (path parameter)": "8*2*2*4*6"
3 }
```

خروجی برای Query Parameters :

درخواست به سرور :

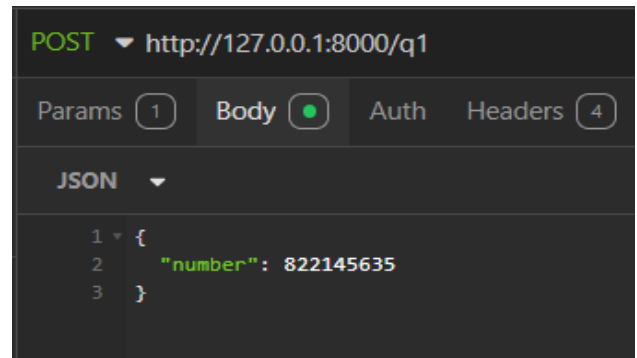
```
GET http://127.0.0.1:8000/q1/?number=822145635
```

پاسخ سرور :

```
1 {
2   "result with (query parameter)": "8*2*2*4*6"
3 }
```

خروجی برای Request Body :

درخواست به سرور :



پاسخ سرور :

```
1 {  
2   "result with (body)": "8*2*2*4*6"  
3 }
```

حل سوال 2 :

```
1 """  
2 2 - Write a program to compute the following expression for 500 sentences:  
3 (3! / 2+9) - (5! / 3+7) + (7! / 4+5) - (9! / 5+3) + (11! / 6+1) - (13! 7-1) ...  
4 """  
5  
6 from fastapi import FastAPI  
7 from pydantic import BaseModel  
8 import math  
9 from decimal import Decimal  
10  
11 app = FastAPI()  
12  
13  
14 class NumberBody(BaseModel):  
15     n: int = 0  
16  
17  
18 def q2(number: int):  
19     change_sign = True  
20     sign = Decimal(1)
```

```

21     fact_start = 3
22     first_num = 2
23     second_num = 9
24     result = Decimal(0)
25
26     for _ in range(number):
27         fact = Decimal(math.factorial(fact_start))
28         denominator = Decimal(first_num + second_num)
29
30         if denominator == 0:
31             continue
32         else:
33             result += sign * (fact / denominator)
34
35         fact_start += 2
36         first_num += 1
37         second_num -= 2
38
39         if change_sign:
40             sign = Decimal(-1)
41             change_sign = False
42         else:
43             sign = Decimal(1)
44             change_sign = True
45
46     return result
47
48
49     @app.get("/q2/{n}")
50     def question2_path(n: int):
51         result = q2(n)
52         return {"result with (path parameter)": result}
53
54
55     @app.get("/q2/")
56     def question2_query(n: int):
57         result = q2(n)
58         return {"result with (query parameter)": result}
59
60
61     @app.post("/q2/")
62     def question2_body(n: NumberBody):
63         result = q2(n.n)
64         return {"result with (body)": result}

```

در خطوط ۶ تا ۹، کتابخانه های لازم ایمپورت شده اند. در خط ۱۱، یک شی از کلاس FastAPI ایجاد شده و تحت عنوان متغیری با نام app ذخیره می شود. خط ۱۴ و ۱۵، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می شود. این مدل یک عدد (n) از نوع int تعریف می کند که مقدار پیش فرض آن ۰ است.

در خطوط ۱۸ تا ۴۶، برنامه ای که ما در تمرین سری اول نوشتیم را تحت عنوان تابعی با نام q2 ذخیره کرده ایم تا در API استفاده کنیم.

در خطوط ۴۹ تا ۵۲، با استفاده از خط ۴۹، مشخص شده که این مسیر از نوع GET است و {n} یک (path parameter) است. این پارامتر مستقیماً در URL دریافت می شود. کاربر باید عدد مورد نظر خود را به طور مستقیم به عنوان بخشی از URL وارد کند. FastAPI به طور خودکار مقدار ورودی مسیر را از رشته به نوع داده ای که مشخص کرده ایم (در اینجا int) تبدیل می کند. به همین دلیل در خط ۵۰، n: int در امضای تابع استفاده شده است. بعد از انجام محاسبات، در خط ۵۲ نتیجه برگردانده می شود.

در خطوط ۵۵ تا ۵۸، با استفاده از خط ۵۵، این مسیر نیز از نوع GET است، اما در اینجا پارامتر عدد به صورت (Query Parameter) دریافت می شود، به این معنا که مقدار عدد بعد از علامت در URL وارد می شود. این پارامتر به صورت n=...? در انتهای URL قرار می گیرد. در این حالت، FastAPI به طور خودکار این پارامتر کوئری را به متغیر n با نوع int تبدیل می کند. بعد از انجام محاسبات، در خط ۵۸ نتیجه برگردانده می شود.

در خطوط ۶۱ تا ۶۴، با استفاده از خط ۶۱، این مسیر از نوع POST است و داده ها از طریق (Request Body) به سرور ارسال می شوند. در اینجا از مدل NumberBody که از pydantic.BaseModel ارث بری کرده است، برای اعتبارسنجی و دریافت ورودی استفاده می شود. بدنه درخواست (body) باید به صورت JSON ارسال شود. این مدل به طور خودکار مقدار n را از JSON دریافت کرده و آن را به تابع q2 می فرستد. بعد از انجام محاسبات، در خط ۶۴ نتیجه برگردانده می شود.

بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q2/500
```

پاسخ سرور :

```
1 {
2   "result with (path parameter)": 2.582651172750457e+22
3 }
```

خروجی برای Query Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q2/?n=500
```

پاسخ سرور :

```
1 {  
2   "result with (query parameter)": 2.582651172750457e+22  
3 }
```

خروجی برای Request Body :

درخواست به سرور :

```
POST http://127.0.0.1:8000/q2/?n=500  
  
Params Body Auth Headers 4  
  
JSON  
  
1 {  
2   "n": 500  
3 }
```

پاسخ سرور :

```
1 {  
2   "result with (body)": 2.582651172750457e+22  
3 }
```



```

1  """
2  3- Write a program to print all 4 digits numbers (between 1000 and 9999) that the sum of the first
3  and second digits is equal with the product of the third and forth digits.
4  3466: 6 + 6 = 3 × 4
5  Output: 1110, 1101, ..., 2999 , ... , 3466 , ...
6  """
7
8  from fastapi import FastAPI
9  from pydantic import BaseModel
10
11  app = FastAPI()
12
13
14  class RangeBody(BaseModel):
15      start: int = 1000
16      end: int = 10000
17
18
19  def q3(start: int, end: int):
20      results = []
21      for num in range(start, end):
22          number = str(num)
23          digit1 = int(number[0])
24          digit2 = int(number[1])
25          digit3 = int(number[2])
26          digit4 = int(number[3])
27          if digit1 + digit2 == digit3 * digit4:
28              results.append(num)
29      return results
30
31
32  @app.get("/q3/{start}-{end}")
33  def question3_path(start: int, end: int):
34      results = q3(start, end)
35      return {"result with (path parameter)": results}
36
37
38  @app.get("/q3/")
39  def question3_query(start: int, end: int):
40      results = q3(start, end)
41      return {"result with (query parameter)": results}
42

```

```

43
44 @app.post("/q3/")
45 def question3_body(range_body: RangeBody):
46     results = q3(range_body.start, range_body.end)
47     return {"result with (body)": results}

```

در خطوط ۸ و ۹، کتابخانه های لازم ایمپورت شده اند. در خط ۱۱، یک شی از کلاس FastAPI ایجاد شده و تحت عنوان متغیری با نام app ذخیره می شود. خط ۱۴ و ۱۵، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می شود. این مدل دو عدد start از نوع int و با مقدار اولیه ۱۰۰۰ و end از نوع int و با مقدار اولیه ۱۰۰۰۰ تعریف می کند.

در خطوط ۱۹ تا ۲۹، برنامه ای که ما در تمرین سری اول نوشتیم را تحت عنوان تابعی با نام q3 ذخیره کرده ایم تا در API استفاده کنیم.

در خطوط ۳۲ تا ۳۵، با استفاده از خط ۳۲، مشخص شده که این مسیر از نوع GET است و {start} و {end} دوتا (path parameter) هستند. این پارامترها مستقیماً در URL دریافت می شود. کاربر باید اعداد مورد نظر خود را به طور مستقیم به عنوان بخشی از URL وارد کند. FastAPI به طور خودکار مقدار ورودی مسیر را از رشته به نوع داده ای که مشخص کرده ایم (در اینجا int) تبدیل می کند. به همین دلیل در خط ۳۳، start: int و end: int در امضای تابع استفاده شده اند. بعد از انجام محاسبات، در خط ۳۵ نتیجه برگردانده می شود.

در خطوط ۳۸ تا ۴۱، با استفاده از خط ۳۸، این مسیر نیز از نوع GET است، اما در اینجا پارامترها به صورت (Query Parameter) دریافت می شوند، به این معنا که مقدار اعداد بعد از علامت در URL وارد می شوند. این پارامترها به صورت start=...&end=... در انتهای URL قرار می گیرد (نکته ای که اینجا وجود دارد این است که بین دو عددی که قرار است وارد شود، باید حتماً علامت قرار داده شود و این مورد برای موارد بیشتر نیز صدق میکند). در این حالت، FastAPI به طور خودکار این پارامترهای کوئری را به متغیرهای start با نوع int و end با نوع int تبدیل می کند. بعد از انجام محاسبات، در خط ۴۱ نتیجه برگردانده می شود.

در خطوط ۴۴ تا ۴۷، با استفاده از خط ۴۴، این مسیر از نوع POST است و داده ها از طریق (Request Body) به سرور ارسال می شوند. در اینجا از مدل RangeBody که از pydantic.BaseModel ارث بری کرده است، برای اعتبارسنجی و دریافت ورودی استفاده می شود. بدنه درخواست (body) باید به صورت JSON ارسال شود. این مدل به طور خودکار مقدارهای start و end را از JSON دریافت کرده و آن را به تابع q3 می فرستد. بعد از انجام محاسبات، در خط ۴۷ نتیجه برگردانده می شود.

بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

GET http://127.0.0.1:8000/q3/1000-10000

پاسخ سرور :

```
1 {
2   "result with (path parameter)": [
3     1011,
4     1112,
5     1121,
6     1213,
7     1231,
8     1314,
9     1322,
10    1341,
11    1415,
12    1451,
13    1516,
14    1523,
15    1532,
16    1561,
17    1617,
18    1671,
19    1718,
20    1724,
21    1742,
22    1781,
23    1819,
24    1833,
25    1891,
```

خروجی برای Query Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q3/?start=1000&end=10000
```

پاسخ سرور :

```
1 {
2   "result with (query parameter)": [
3     1011,
4     1112,
5     1121,
6     1213,
7     1231,
8     1314,
9     1322,
10    1341,
11    1415,
12    1451,
13    1516,
14    1523,
15    1532,
16    1561,
17    1617,
18    1671,
19    1718,
20    1724,
21    1742,
22    1781,
23    1819,
24    1833,
25    1891,
```

خروجی برای Request Body :

درخواست به سرور :

```
POST http://127.0.0.1:8000/q3
Body
JSON
1 {
2   "start": 1000,
3   "end": 10000
4 }
```

پاسخ سرور :

```
1 {
2   "result with (body)": [
3     1011,
4     1112,
5     1121,
6     1213,
7     1231,
8     1314,
9     1322,
10    1341,
11    1415,
12    1451,
13    1516,
14    1523,
15    1532,
16    1561,
17    1617,
18    1671,
19    1718,
20    1724,
21    1742,
22    1781,
23    1819,
24    1833,
25    1891,
```

```

1  """
2  4=- Write a program to print all 3 digits numbers (between a 100 and 999) that does not have odd digits.
3  Consider 0 as even digit.
4  Output: 200, 202,204, 206,208,220,222,...
5  """
6  from fastapi import FastAPI
7  from pydantic import BaseModel
8
9  app = FastAPI()
10
11
12  class RangeBody(BaseModel):
13      start: int = 100
14      end: int = 999
15
16
17  def q4(start: int, end: int):
18      results = []
19      for num in range(start, end):
20          number = str(num)
21          digit1 = int(number[0])
22          digit2 = int(number[1])
23          digit3 = int(number[2])
24          if digit1 % 2 == 0 and digit2 % 2 == 0 and digit3 % 2 == 0:
25              results.append(num)
26      return results
27
28
29  @app.get("/q4/{start}-{end}")
30  def question4_path(start: int, end: int):
31      results = q4(start, end)
32      return {"result with (path parameter)": results}
33
34
35  @app.get("/q4/")
36  def question4_query(start: int, end: int):
37      results = q4(start, end)
38      return {"result with (query parameter)": results}
39
40
41  @app.post("/q4/")
42  def question4_body(range_body: RangeBody):
43      results = q4(range_body.start, range_body.end)
44      return {"result with (body)": results}

```

در خطوط ۶ و ۷، کتابخانه های لازم ایمپورت شده اند. در خط ۹، یک شی از کلاس FastAPI ایجاد شده و تحت عنوان متغیری با نام app ذخیره می شود. خط ۱۲ و ۱۴، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می شود. این مدل دو عدد start از نوع int و با مقدار اولیه ۱۰۰ و end از نوع int و با مقدار اولیه ۹۹۹ تعریف می کند.

در خطوط ۱۷ تا ۲۶، برنامه ای که ما در تمرین سری اول نوشتیم را تحت عنوان تابعی با نام q4 ذخیره کرده ایم تا در API استفاده کنیم.

در خطوط ۲۹ تا ۳۲، با استفاده از خط ۲۹، مشخص شده که این مسیر از نوع GET است و {start} و {end} دوتا (path parameter) هستند. این پارامترها مستقیماً در URL دریافت می شود. کاربر باید اعداد مورد نظر خود را به طور مستقیم به عنوان بخشی از URL وارد کند. FastAPI به طور خودکار مقدار ورودی مسیر را از رشته به نوع داده ای که مشخص کرده ایم (در اینجا int) تبدیل می کند. به همین دلیل در خط ۳۰، start: int و end: int در امضای تابع استفاده شده اند. بعد از انجام محاسبات، در خط ۳۲ نتیجه برگردانده می شود.

در خطوط ۳۵ تا ۳۸، با استفاده از خط ۳۵، این مسیر نیز از نوع GET است، اما در اینجا پارامترها به صورت (Query Parameter) دریافت می شوند، به این معنا که مقدار اعداد بعد از علامت در URL وارد می شوند. این پارامترها به صورت start=...&end=... در انتهای URL قرار می گیرد. در این حالت، FastAPI به طور خودکار این پارامترهای کوئری را به متغیرهای start با نوع int و end با نوع int تبدیل می کند. بعد از انجام محاسبات، در خط ۳۸ نتیجه برگردانده می شود.

در خطوط ۴۱ تا ۴۴، با استفاده از خط ۴۱، این مسیر از نوع POST است و داده ها از طریق (Request Body) به سرور ارسال می شوند. در اینجا از مدل RangeBody که از pydantic.BaseModel ارث بری کرده است، برای اعتبارسنجی و دریافت ورودی استفاده می شود. بدنه درخواست (body) باید به صورت JSON ارسال شود. این مدل به طور خودکار مقدارهای start و end را از JSON دریافت کرده و آن را به تابع q4 می فرستد. بعد از انجام محاسبات، در خط ۴۴ نتیجه برگردانده می شود.

بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q4/100-10000
```

پاسخ سرور :

```
1 - {
2 -   "result with (path parameter)": [
3 -     200,
4 -     202,
5 -     204,
6 -     206,
7 -     208,
8 -     220,
9 -     222,
10 -    224,
11 -    226,
12 -    228,
13 -    240,
14 -    242,
15 -    244,
16 -    246,
17 -    248,
18 -    260,
19 -    262,
20 -    264,
21 -    266,
22 -    268,
23 -    280,
24 -    282,
25 -    284,
```

خروجی برای Query Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q4/?start=100&end=1000
```

پاسخ سرور :

```
1 {
2   "result with (query parameter)": [
3     200,
4     202,
5     204,
6     206,
7     208,
8     220,
9     222,
10    224,
11    226,
12    228,
13    240,
14    242,
15    244,
16    246,
17    248,
18    260,
19    262,
20    264,
21    266,
22    268,
23    280,
24    282,
25    284,
```

خروجی برای Request Body :

درخواست به سرور :

```
POST http://127.0.0.1:8000/q4
Params Body Auth Headers 4
JSON
1 {
2   "start": 100,
3   "end": 1000
4 }
```

پاسخ سرور :

```
1 {
2   "result with (body)": [
3     200,
4     202,
5     204,
6     206,
7     208,
8     220,
9     222,
10    224,
11    226,
12    228,
13    240,
14    242,
15    244,
16    246,
17    248,
18    260,
19    262,
20    264,
21    266,
22    268,
23    280,
24    282,
25    284,
```

```

1  """
2  5 - Write a program to get n from user and print the following pattern:
3  Input: n=8
4  Output:
5  1
6  2 4
7  3 6 9
8  4 8 12 16
9  5 10 15 20 25
10 6 12 18 24 30 36
11 7 14 21 28 35 42 49
12 8 16 24 32 40 48 56 64
13 """
14 from fastapi import FastAPI
15 from pydantic import BaseModel
16
17 app = FastAPI()
18
19
20 class PatternBody(BaseModel):
21     n: int = 8
22
23
24 def q5(n: int):
25     result = []
26     for i in range(1, n + 1):
27         row = []
28         for j in range(1, i + 1):
29             row.append(f"{i*j}")
30         result.append(" ".join(row))
31     return result
32
33
34 @app.get("/q5/{n}")
35 def question5_path(n: int):
36     result = q5(n)
37     return {"result with (path parameter)": result}
38
39
40 @app.get("/q5/")
41 def question5_query(n: int):
42     result = q5(n)
43     return {"result with (query parameter)": result}
44
45
46 @app.post("/q5/")
47 def question5_body(pattern_body: PatternBody):
48     result = q5(pattern_body.n)
49     return {"result with (body)": result}

```


در خطوط ۱۴ و ۱۵، کتابخانه های لازم ایمپورت شده اند. در خط ۱۷، یک شی از کلاس FastAPI ایجاد شده و تحت عنوان متغیری با نام app ذخیره می شود. خط ۲۰ و ۲۱، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می شود. این مدل عدد n از نوع int و با مقدار اولیه ۸ تعریف می کند.

در خطوط ۲۴ تا ۳۱، برنامه ای که ما در تمرین سری اول نوشتیم را تحت عنوان تابعی با نام q5 ذخیره کرده ایم تا در API استفاده کنیم.

در خطوط ۳۴ تا ۳۷، با استفاده از خط ۳۴، مشخص شده که این مسیر از نوع GET است و {n} یک (path parameter) هست. این پارامتر مستقیماً در URL دریافت می شود. کاربر باید عدد مورد نظر خود را به طور مستقیم به عنوان بخشی از URL وارد کند. FastAPI به طور خودکار مقدار ورودی مسیر را از رشته به نوع داده ای که مشخص کرده ایم (در اینجا int) تبدیل می کند. به همین دلیل در خط ۳۵، n: int در امضای تابع استفاده شده اند. بعد از انجام محاسبات، در خط ۳۷ نتیجه برگردانده می شود.

در خطوط ۴۰ تا ۴۳، با استفاده از خط ۴۰، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به صورت (Query Parameter) دریافت می شود، به این معنا که مقدار عدد بعد از علامت در URL وارد می شوند. این پارامتر به صورت n=... در انتهای URL قرار می گیرد. در این حالت، FastAPI به طور خودکار این پارامتر کوئری را به متغیر n با نوع int تبدیل می کند. بعد از انجام محاسبات، در خط ۴۳ نتیجه برگردانده می شود.

در خطوط ۴۶ تا ۴۹، با استفاده از خط ۴۶، این مسیر از نوع POST است و داده ها از طریق (Request Body) به سرور ارسال می شوند. در اینجا از مدل PatternBody که از pydantic.BaseModel ارث بری کرده است، برای اعتبارسنجی و دریافت ورودی استفاده می شود. بدنه درخواست (body) باید به صورت JSON ارسال شود. این مدل به طور خودکار مقدار n را از JSON دریافت کرده و آن را به تابع q5 می فرستد. بعد از انجام محاسبات، در خط ۴۹ نتیجه برگردانده می شود.

بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q5/4
```

پاسخ سرور :

```
1 {
2   "result with (path parameter)": [
3     "1",
4     "2 4",
5     "3 6 9",
6     "4 8 12 16"
7   ]
8 }
```

خروجی برای Query Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q5/?n=4
```

پاسخ سرور :

```
1 {  
2   "result with (query parameter)": [  
3     "1",  
4     "2 4",  
5     "3 6 9",  
6     "4 8 12 16"  
7   ]  
8 }
```

خروجی برای Request Body :

درخواست به سرور :

```
POST http://127.0.0.1:8000/q5/  
  
Params Body Auth Headers 4  
  
JSON  
  
1 {  
2   "n": 4  
3 }
```

پاسخ سرور :

```
1 {  
2   "result with (body)": [  
3     "1",  
4     "2 4",  
5     "3 6 9",  
6     "4 8 12 16"  
7   ]  
8 }
```

حل سوال 6 :

```

1  """
2  6 - Write a program to get n and then n numbers from user and compute maximum, minimum, average and
3  standard deviation. Test your program for 5 different inputs.
4  input:                                output:
5  Enter the n:5                          Maximum is: 84
6  Enter number 1:12                      Minimum is: 9
7  Enter number 2:15                      Average is: 26.2
8  Enter number 3:9                       Standard Deviation is: 28.96
9  Enter number 4:11
10 Enter number 5:84
11 """
12 from fastapi import FastAPI
13 from pydantic import BaseModel
14 import math
15
16 app = FastAPI()
17
18
19 class NumbersBody(BaseModel):
20     numbers: list[float]
21
22
23 def q6(numbers: list[float]):
24     maximum = max(numbers)
25     minimum = min(numbers)
26     average = sum(numbers) / len(numbers)
27     variance = sum((x - average) ** 2 for x in numbers) / len(numbers)
28     std = math.sqrt(variance)
29     return maximum, minimum, average, std
30
31
32 @app.get("/q6/{numbers}")
33 def question6_path(numbers: str):
34     number_list = list(map(float, numbers.split(',')))
35     maximum, minimum, average, std = q6(number_list)
36     return {
37         "Type result": "path parameter",
38         "Maximum": maximum,
39         "Minimum": minimum,
40         "Average": float(f"{average:.4f}"),
41         "Standard Deviation": float(f"{std:.4f}")
42     }
43
44
45 @app.get("/q6/")
46 def question6_query(numbers: str):
47     number_list = list(map(float, numbers.split(',')))
48     maximum, minimum, average, std = q6(number_list)
49     return {
50         "Type result": "query parameter",
51         "Maximum": maximum,
52         "Minimum": minimum,
53         "Average": float(f"{average:.4f}"),
54         "Standard Deviation": float(f"{std:.4f}")
55     }
56
57

```

```

58 @app.post("/q6/")
59 def question6_body(numbers_body: NumbersBody):
60     numbers = numbers_body.numbers
61     maximum, minimum, average, std = q6(numbers)
62     return {
63         "Type result": "body",
64         "Maximum": maximum,
65         "Minimum": minimum,
66         "Average": float(f"average:.4f"),
67         "Standard Deviation": float(f"std:.4f")
68     }

```

در خطوط ۱۲ تا ۱۴، کتابخانه های لازم ایمپورت شده اند. در خط ۱۶، یک شی از کلاس FastAPI ایجاد شده و تحت عنوان متغیری با نام app ذخیره می شود. خط ۱۹ و ۲۰، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می شود. این مدل عدد numbers از نوع لیست float تعریف می کند.

در خطوط ۲۳ تا ۲۹، برنامه ای که ما در تمرین سری اول نوشتیم را تحت عنوان تابعی با نام q6 ذخیره کرده ایم تا در API استفاده کنیم. در خطوط ۳۲ تا ۴۲، با استفاده از خط ۳۲، مشخص شده که این مسیر از نوع GET است و {numbers} یک (path parameter) هست. این پارامتر مستقیماً در URL دریافت می شود. کاربر باید رشته اعداد مورد نظر خود را به طور مستقیم به عنوان بخشی از URL وارد کند (بین اعداد را با , جدا کند). بعد از انجام محاسبات، در خط ۳۶ نتیجه برگردانده می شود.

در خطوط ۴۵ تا ۵۵، با استفاده از خط ۴۵، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به صورت (Query Parameter) دریافت می شود، به این معنا که مقدار رشته بعد از علامت ? در URL وارد می شود. این پارامتر به صورت ?numbers=... در انتهای URL قرار می گیرد. بعد از انجام محاسبات، در خط ۴۹ نتیجه برگردانده می شود.

در خطوط ۵۸ تا ۶۸، با استفاده از خط ۵۸، این مسیر از نوع POST است و داده ها از طریق (Request Body) به سرور ارسال می شوند. در اینجا از مدل NumbersBody که از pydantic.BaseModel ارث بری کرده است، برای اعتبارسنجی و دریافت ورودی استفاده می شود. بدنه درخواست (body) باید به صورت JSON ارسال شود. این مدل به طور خودکار مقدار numbers را از JSON دریافت کرده، اعداد را درون لیستی تفکیک و ذخیره می کند و لیست اعداد را به تابع q6 می فرستد. بعد از انجام محاسبات، در خط ۶۲ نتیجه برگردانده می شود.

بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

GET http://127.0.0.1:8000/q6/12,15,10.5

پاسخ سرور :

```

1 {
2   "Type result": "path parameter",
3   "Maximum": 15.0,
4   "Minimum": 10.5,
5   "Average": 12.5,
6   "Standard Deviation": 1.8708
7 }

```

خروجی برای Query Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q6/?numbers=12,15,10.5
```

پاسخ سرور :

```
1 {  
2   "Type result": "query parameter",  
3   "Maximum": 15.0,  
4   "Minimum": 10.5,  
5   "Average": 12.5,  
6   "Standard Deviation": 1.8708  
7 }
```

خروجی برای Request Body :

درخواست به سرور :

```
POST http://127.0.0.1:8000/q6/  
  
Params Body Auth Headers (4)  
  
JSON  
  
1 {  
2   "numbers": [10, 12, 15, 17, 19.5]  
3 }
```

پاسخ سرور :

```
1 {  
2   "Type result": "body",  
3   "Maximum": 19.5,  
4   "Minimum": 10.0,  
5   "Average": 14.7,  
6   "Standard Deviation": 3.4  
7 }
```

```

1  """
2  7 - Write 4 functions for question 6. Get 5 numbers from user in main.
3  Send numbers for Max1, Min1, Ave1 and STD1 functions. Max1 gives n numbers and return maximum.
4  Min1 gives n number and return minimum. Ave1 and STD1 give n number and print average and
5  standard deviation in their own body and do not return any value. Test your program for 5 different inputs.
6  """
7  from fastapi import FastAPI
8  from pydantic import BaseModel
9  import math
10
11  app = FastAPI()
12
13
14  class NumbersBody(BaseModel):
15      numbers: list[float]
16
17
18  def Max1(numbers: list[float]):
19      maximum = numbers[0]
20      for num in numbers:
21          if num > maximum:
22              maximum = num
23      return maximum
24
25
26  def Min1(numbers: list[float]):
27      minimum = numbers[0]
28      for num in numbers:
29          if num < minimum:
30              minimum = num
31      return minimum
32
33
34  def Ave1(numbers: list[float]):
35      total = sum(numbers)
36      average = total / len(numbers)
37      return average
38
39
40  def STD1(numbers: list[float]):
41      ave = Ave1(numbers)
42      total = sum((num - ave) ** 2 for num in numbers)
43      std = math.sqrt(total / len(numbers))
44      return std
45
46
47  @app.get("/q7/{numbers}")
48  def question7_path(numbers: str):
49      number_list = list(map(float, numbers.split(',')))
50      return {
51          "Type result": "path parameter",
52          "Maximum": Max1(number_list),
53          "Minimum": Min1(number_list),
54          "Average": float(f"{Ave1(number_list):.4f}"),
55          "Standard Deviation": float(f"{STD1(number_list):.4f}")
56      }
57
58

```

```

59 @app.get("/q7/")
60 def question7_query(numbers: str):
61     numbers_list = list(map(float, numbers.split(',')))
62     return {
63         "Type result": "query parameter",
64         "Maximum": Max1(numbers_list),
65         "Minimum": Min1(numbers_list),
66         "Average": float(f"Ave1(numbers_list):.4f"),
67         "Standard Deviation": float(f"STD1(numbers_list):.4f")
68     }
69
70
71 @app.post("/q7/")
72 def question7_body(numbers_body: NumbersBody):
73     numbers = numbers_body.numbers
74     return {
75         "Type result": "body",
76         "Maximum": Max1(numbers),
77         "Minimum": Min1(numbers),
78         "Average": float(f"Ave1(numbers):.4f"),
79         "Standard Deviation": float(f"STD1(numbers):.4f")
80     }

```

در خطوط ۷ تا ۹، کتابخانه های لازم ایمپورت شده اند. در خط ۱۱، یک شی از کلاس FastAPI ایجاد شده و تحت عنوان متغیری با نام app ذخیره می شود. خط ۱۴ و ۱۵، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می شود. این مدل عدد numbers از نوع لیست float تعریف می کند.

در خطوط ۱۸ تا ۴۴، توابعی که در همین سوال ۷ در تمرین سری اول حل کردیم را نوشته ایم تا در API استفاده کنیم.

در خطوط ۴۷ تا ۵۶، با استفاده از خط ۴۷، مشخص شده که این مسیر از نوع GET است و {numbers} یک (path parameter) هست. این پارامتر مستقیماً در URL دریافت می شود. کاربر باید رشته اعداد مورد نظر خود را به طور مستقیم به عنوان بخشی از URL وارد کند (بین اعداد را با , جدا کند). بعد از انجام محاسبات، در خط ۵۶ نتیجه برگردانده می شود.

در خطوط ۵۹ تا ۶۸، با استفاده از خط ۵۹، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به صورت (Query Parameter) دریافت می شود، به این معنا که مقدار رشته بعد از علامت ? در URL وارد می شود. این پارامتر به صورت numbers=? در انتهای URL قرار می گیرد. بعد از انجام محاسبات، در خط ۶۸ نتیجه برگردانده می شود.

در خطوط ۷۱ تا ۸۰، با استفاده از خط ۷۱، این مسیر از نوع POST است و داده ها از طریق (Request Body) به سرور ارسال می شوند. در اینجا از مدل NumbersBody که از pydantic.BaseModel ارث بری کرده است، برای اعتبارسنجی و دریافت ورودی استفاده می شود. بدنه درخواست (body) باید به صورت JSON ارسال شود. این مدل به طور خودکار مقدار numbers را از JSON دریافت کرده، اعداد را درون لیستی تفکیک و ذخیره می کند و لیست اعداد را به هر کدام از توابع می فرستد. بعد از انجام محاسبات، در خط ۸۰ نتیجه برگردانده می شود.

بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

GET ▼ http://127.0.0.1:8000/q7/12,17.85,19

پاسخ سرور :

```
1 {  
2   "Type result": "path parameter",  
3   "Maximum": 19.0,  
4   "Minimum": 12.0,  
5   "Average": 16.2833,  
6   "Standard Deviation": 3.0649  
7 }
```

خروجی برای Query Parameters :

درخواست به سرور :

GET ▼ http://127.0.0.1:8000/q7/?numbers=12,17.85,19

پاسخ سرور :

```
1 {  
2   "Type result": "query parameter",  
3   "Maximum": 15.0,  
4   "Minimum": 10.5,  
5   "Average": 12.5,  
6   "Standard Deviation": 1.8708  
7 }
```

خروجی برای Request Body :

درخواست به سرور :

POST ▼ http://127.0.0.1:8000/q7/

Params Body Auth Headers (4)

JSON ▼

```
1 {  
2   "numbers": [12, 17.85, 10, 15.8]  
3 }
```

پاسخ سرور :

```
1 {  
2   "Type result": "body",  
3   "Maximum": 17.85,  
4   "Minimum": 10.0,  
5   "Average": 13.9125,  
6   "Standard Deviation": 3.0835  
7 }
```



```

1  """
2  8 - Give a number with 5 digits in main. If the number is not a 5 digits number, ask repeatedly from user
3  to enter a valid number. Send the number for F1 function to find and return the maximum digit of the number.
4  In the next step, send the maximum digit and the input number for F2 function to delete the maximum digit from
5  number and return it. Finally, print the final output in main as shown in figure below.
6  """
7  from fastapi import FastAPI
8  from pydantic import BaseModel
9
10 app = FastAPI()
11
12
13 class NumberBody(BaseModel):
14     number: int
15
16
17 def F1(number: int):
18     digits = []
19     while number > 0:
20         digit = number % 10
21         digits.append(digit)
22         number //= 10
23     max_digit = max(digits)
24     return max_digit
25
26
27 def F2(number: int, digit: int):
28     number = str(number).replace(f"{digit}", "").strip()
29     return number
30
31
32 @app.get("/q8/{number}")
33 def question8_path(number: int):
34     if len(str(number)) == 5:
35         max_digit = F1(number)
36         final_number = F2(number, max_digit)
37         return {
38             "Type result": "path parameter",
39             "Maximum digit": max_digit,
40             f"Final number without {max_digit}": final_number
41         }
42     else:
43         return {"error": "The number is not 5 digits"}
44
45
46 @app.get("/q8/")
47 def question8_query(number: int):
48     if len(str(number)) == 5:
49         max_digit = F1(number)
50         final_number = F2(number, max_digit)
51         return {
52             "Type result": "query parameter",
53             "Maximum digit": max_digit,
54             f"Final number without {max_digit}": final_number
55         }
56     else:
57         return {"error": "The number is not 5 digits"}
58
59

```

```

60 @app.post("/q8/")
61 def question8_body(data: NumberBody):
62     number = data.number
63     if len(str(number)) == 5:
64         max_digit = F1(number)
65         final_number = F2(number, max_digit)
66         return {
67             "Type result": "body",
68             "Maximum digit": max_digit,
69             f"Final number without {max_digit}": final_number
70         }
71     else:
72         return {"error": "The number is not 5 digits"}

```

در خطوط ۷ و ۸، کتابخانه های لازم ایمپورت شده اند. در خط ۱۰، یک شی از کلاس FastAPI ایجاد شده و تحت عنوان متغیری با نام app ذخیره می شود. خط ۱۳ و ۱۴، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می شود. این مدل عدد number از نوع int تعریف می کند.

در خطوط ۱۷ تا ۲۹، توابعی که در همین سوال ۸ در تمرین سری اول حل کردیم را نوشته ایم تا در API استفاده کنیم.

در خطوط ۳۲ تا ۴۳، با استفاده از خط ۳۲، مشخص شده که این مسیر از نوع GET است و {number} یک (path parameter) هست. این پارامتر مستقیماً در URL دریافت می شود. کاربر باید عدد مورد نظر خود را به طور مستقیم به عنوان بخشی از URL وارد کند. در خط ۳۴ بررسی میکند که عدد ۵ رقمی است یا نه. اگر ۵ رقمی بود، عدد را به هرکدام از توابع می فرستد و بعد از انجام محاسبات، در خط ۳۷ نتیجه برگردانده می شود. اما اگر ۵ رقمی نبود، پیغام خطایی را نمایش میدهد.

در خطوط ۴۶ تا ۵۷، با استفاده از خط ۴۶، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به صورت (Query Parameter) دریافت می شود، به این معنا که مقدار عدد بعد از علامت در URL وارد می شود. این پارامتر به صورت ...number=? در انتهای URL قرار می گیرد. در خط ۴۸ بررسی میکند که عدد ۵ رقمی است یا نه. اگر ۵ رقمی بود، عدد را به هرکدام از توابع می فرستد و بعد از انجام محاسبات، در خط ۵۱ نتیجه برگردانده می شود. اما اگر ۵ رقمی نبود، پیغام خطایی را نمایش میدهد.

در خطوط ۶۰ تا ۷۲، با استفاده از خط ۶۰، این مسیر از نوع POST است و داده ها از طریق (Request Body) به سرور ارسال می شوند. در اینجا از مدل NumberBody که از pydantic.BaseModel ارث بری کرده است، برای اعتبارسنجی و دریافت ورودی استفاده می شود. بدنه درخواست (body) باید به صورت JSON ارسال شود. این مدل به طور خودکار مقدار number را از JSON دریافت کرده و در خط ۶۳ بررسی میکند که عدد ۵ رقمی است یا نه. اگر ۵ رقمی بود، عدد را به هرکدام از توابع می فرستد و بعد از انجام محاسبات، در خط ۶۶ نتیجه برگردانده می شود. اما اگر ۵ رقمی نبود، پیغام خطایی را نمایش میدهد.

بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

GET ▼ http://127.0.0.1:8000/q8/58821

پاسخ سرور :

```
1 {  
2   "Type result": "path parameter",  
3   "Maximum digit": 8,  
4   "Final number without 8": "521"  
5 }
```

خروجی برای Query Parameters :

درخواست به سرور :

```
GET http://127.0.0.1:8000/q8/?number=58821
```

پاسخ سرور :

```
1 {  
2   "Type result": "query parameter",  
3   "Maximum digit": 8,  
4   "Final number without 8": "521"  
5 }
```

خروجی برای Request Body :

درخواست به سرور :

POST http://127.0.0.1:8000/q8

Params Body Auth Headers 4

JSON

```
1 {  
2   "number": 48574  
3 }
```

پاسخ سرور :

```
1 {  
2   "Type result": "body",  
3   "Maximum digit": 8,  
4   "Final number without 8": "4574"  
5 }
```

```

1  """
2  9-Global and Local variables: Run codes P1, P2, P3 and P4 and report outputs.
3  Explain reasons for their outputs in details.
4  """
5  from fastapi import FastAPI
6
7  app = FastAPI()
8
9
10 def q9_p1():
11     # P1
12     def f():
13         # local variable
14         s1 = "I live in khorramabad"
15     return s1
16
17     # Main
18     f()
19
20
21 def q9_p2():
22     # P2
23     try:
24         def f():
25             # local variable
26             s2 = "I live in khorramabad"
27             print("Inside Function:", s2)
28
29             # Main
30             f()
31             return s2 # This will raise a NameError
32     except NameError as e:
33         # Catch the NameError and return it as a string
34         return f"Error: {str(e)}"
35
36
37 def q9_p3():
38     # P3
39     def f():
40         global s3
41         s3 = 'I live in khorramabad.'
42         print(s3)
43
44     # Main: Global Scope
45     s3 = 'I live in iran.'
46     f()
47     return s3
48
49
50 def q9_p4():
51     # P4
52     # This function uses global variable s
53     def f():
54         s4 = "I live in khorramabad."
55         print(s4)
56
57     # Main
58     s4 = "I live in iran."
59     f()
60     return s4
61
62

```

```

63 @app.get("/q9/")
64 def question9_path_query():
65     return {
66         "Type result": "path parameter and query parameter",
67         "result p1 is": q9_p1(),
68         "result p2 is": q9_p2(),
69         "result p3 is": q9_p3(),
70         "result p4 is": q9_p4()
71     }
72
73
74 @app.post("/q9/")
75 def question9_body():
76     return {
77         "Type result": "body",
78         "result p1 is": q9_p1(),
79         "result p2 is": q9_p2(),
80         "result p3 is": q9_p3(),
81         "result p4 is": q9_p4()
82     }

```

در خط ۵، کتابخانه لازم ایمپورت شده است. در خطوط ۱۰ تا ۶۰، توابعی که در تمرین سری اول داده شده بود را نوشته ایم تا در API استفاده کنیم. در خطوط ۶۳ تا ۷۱، با استفاده از خط ۶۳، مشخص شده که این مسیر از نوع GET است و هم به روش Path Parameter و هم روش Query Parameter است. در خطوط ۷۴ تا ۸۲، با استفاده از خط ۷۴، مشخص شده که این مسیر از نوع POST است و به روش Request Body است. نکته مهمی که باید در اینجا به آن توجه شود، این است که چون ما صرفاً میخواهیم خروجی هر کدام از توابع را بدون هیچ ورودی بررسی کنیم، پس ورودی را برای هیچ کدام از روش ها قرار نمی دهیم. بر این اساس روش های Path و Query به یک شکل نوشته شده اند و روش Body هم بدون کلاس خاصی در نظر گرفته شده است.

بررسی خروجی ها :

خروجی برای Path Parameters و Query Parameters :

درخواست به سرور : GET http://127.0.0.1:8000/q9

پاسخ سرور :

```

1 {
2     "Type result": "path parameter and query parameter",
3     "result p1 is": null,
4     "result p2 is": "Error: name 's2' is not defined",
5     "result p3 is": "I live in iran.",
6     "result p4 is": "I live in iran."
7 }

```

خروجی برای Request Body :

درخواست به سرور : POST http://127.0.0.1:8000/q9

پاسخ سرور :

```

1 {
2     "Type result": "body",
3     "result p1 is": null,
4     "result p2 is": "Error: name 's2' is not defined",
5     "result p3 is": "I live in iran.",
6     "result p4 is": "I live in iran."
7 }

```

```

1  """
2  10-Study about Recursive Functions in python.
3  Explain in details that how does the following recursive function work?
4  """
5  from fastapi import FastAPI
6  from pydantic import BaseModel
7
8  app = FastAPI()
9
10
11 def factorial(x: int):
12     if x == 1: # This is the base case
13         return 1
14     else: # This is the recursive case
15         return x * factorial(x - 1)
16
17
18 class FactorialBody(BaseModel):
19     number: int
20
21
22 @app.get("/q10/{number}")
23 async def question10_path(number: int):
24     if number < 1:
25         return {"error": "The number should be greater than 0"}
26     else:
27         result = factorial(number)
28         return {
29             "Type result": "path parameter",
30             "Factorial": result
31         }
32
33
34 @app.get("/q10/")
35 async def question10_query(number: int):
36     if number < 1:
37         return {"error": "The number should be greater than 0"}
38     else:
39         result = factorial(number)
40         return {
41             "Type result": "query parameter",
42             "Factorial": result
43         }

```

```

44
45
46 @app.post("/q10/")
47 async def question10_body(data: FactorialBody):
48     number = data.number
49     if number < 1:
50         return {"error": "The number should be greater than 0"}
51     else:
52         result = factorial(number)
53         return {
54             "Type result": "body",
55             "Factorial": result
56         }

```

در خطوط ۵ و ۶، کتابخانه های لازم ایمپورت شده اند. در خط ۸، یک شی از کلاس FastAPI ایجاد شده و تحت عنوان متغیری با نام app ذخیره می شود. در خطوط ۱۱ تا ۱۵، تابعی که صورت سوال داده را نوشته ایم تا در API استفاده کنیم. خط ۱۸ و ۱۹، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می شود. این مدل عدد number از نوع int تعریف می کند.

در خطوط ۲۲ تا ۳۱، با استفاده از خط ۲۲، مشخص شده که این مسیر از نوع GET است و {number} یک (path parameter) هست. این پارامتر مستقیماً در URL دریافت می شود. کاربر باید عدد مورد نظر خود را به طور مستقیم به عنوان بخشی از URL وارد کند. در خط ۲۴ بررسی میکند عدد کمتر از ۱ است یا نه. اگر کمتر از ۱ بود، پیغام خطایی را نمایش میدهد. اما اگر بزرگتر یا مساوی ۱ بود، عدد را به تابع فاکتوریل می فرستد و بعد از انجام محاسبات، در خط ۲۸ نتیجه برگردانده می شود.

در خطوط ۳۴ تا ۴۳، با استفاده از خط ۳۴، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به صورت (Query Parameter) دریافت می شود، به این معنا که مقدار عدد بعد از علامت در URL وارد می شود. این پارامتر به صورت ...number=? در انتهای URL قرار می گیرد. در خط ۳۶ بررسی میکند عدد کمتر از ۱ است یا نه. اگر کمتر از ۱ بود، پیغام خطایی را نمایش میدهد. اما اگر بزرگتر یا مساوی ۱ بود، عدد را به تابع فاکتوریل می فرستد و بعد از انجام محاسبات، در خط ۴۰ نتیجه برگردانده می شود.

در خطوط ۴۶ تا ۵۶، با استفاده از خط ۴۶، این مسیر از نوع POST است و داده ها از طریق (Request Body) به سرور ارسال می شوند. در اینجا از مدل FactorialBody که از pydantic.BaseModel ارث بری کرده است، برای اعتبارسنجی و دریافت ورودی استفاده می شود. بدنه درخواست (body) باید به صورت JSON ارسال شود. این مدل به طور خودکار مقدار number را از JSON دریافت کرده و در خط ۴۹ بررسی میکند عدد کمتر از ۱ است یا نه. اگر کمتر از ۱ بود، پیغام خطایی را نمایش میدهد. اما اگر بزرگتر یا مساوی ۱ بود، عدد را به تابع فاکتوریل می فرستد و بعد از انجام محاسبات، در خط ۵۳ نتیجه برگردانده می شود.

بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

GET http://127.0.0.1:8000/q10/5

پاسخ سرور :

```
1 {  
2   "Type result": "path parameter",  
3   "Factorial": 120  
4 }
```

خروجی برای Query Parameters :

درخواست به سرور :

GET http://127.0.0.1:8000/q10/?number=5

پاسخ سرور :

```
1 {  
2   "Type result": "query parameter",  
3   "Factorial": 120  
4 }
```

خروجی برای Request Body :

درخواست به سرور :

POST http://127.0.0.1:8000/q10/?number=5

Params Body Auth Headers 4

JSON

```
1 {  
2   "number": 10  
3 }
```

پاسخ سرور :

```
1 {  
2   "Type result": "body",  
3   "Factorial": 3628800  
4 }
```

پایان