به نام خدای 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 معمولا 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 تمرین ها با استفاده از این API میپردازیم.

حل سوال 1 :

```python
"""
1 - Write a program to get a number and print all its even digits separated by *.
Input: 822145635
Output: 6*4*2*2*8
"""

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class NumberBody(BaseModel):
    number: int = 0


def q1(number: int):
    even_digits = [digit for digit in str(number) if int(digit) % 2 == 0]
    return '*'.join(even_digits)


@app.get("/q1/{number}")
def question1_path(number: int):
    result = q1(number)
    return {"result with (path parameter)": result}


@app.get("/q1/")
def question1_query(number: int):
    result = q1(number)
    return {"result with (query parameter)": result}


@app.post("/q1/")
def question1_body(number: NumberBody):
    result = q1(number.number)
    return {"result with (body)": result}
```

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 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، برنامه ای 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 مستقیما 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.

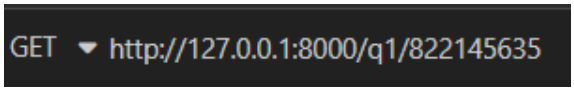
در خطوط ۲۸ تا ۳۱، با استفاده از خط ۲۸، این مسیر نیز از نوع GET است، اما در اینجا پارامتر عدد به‌صورت (Query Parameter) دریافت می‌شود، به این معنا که مقدار عدد بعد از علامت در URL وارد می‌شود. این پارامتر به‌صورت ?number=... در انتهای URL قرار می‌گیرد. در این حالت، FastAPI به‌طور خودکار این پارامتر کوئری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

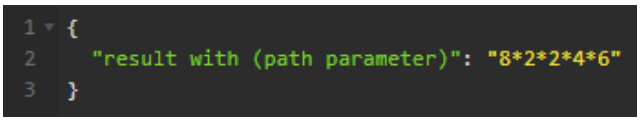
بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

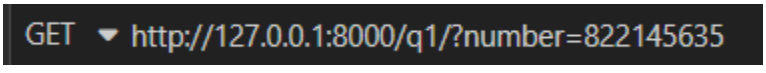


پاسخ سرور :

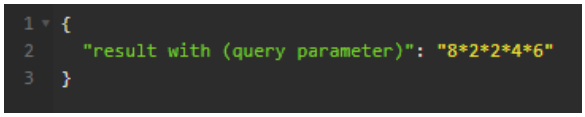


خروجی برای Query Parameters :

درخواست به سرور :

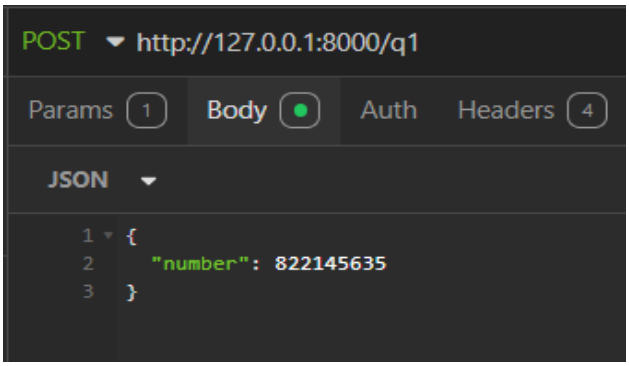


پاسخ سرور :



خروجی برای Request Body :

درخواست به سرور :



پاسخ سرور :

حل سوال 2 :

```python
"""
2 - Write a program to compute the following expression for 500 sentences:
(3! / 2+9) - (5! / 3+7) + (7! / 4+5) - (9! / 5+3) + (11! / 6+1) - (13! 7-1) …
"""

from fastapi import FastAPI
from pydantic import BaseModel
import math
from decimal import Decimal

app = FastAPI()


class NumberBody(BaseModel):
    n: int = 0


def q2(number: int):
    change_sign = True
    sign = Decimal(1)
```

```
    fact_start = 3
    first_num = 2
    second_num = 9
    result = Decimal(0)

    for _ in range(number):
        fact = Decimal(math.factorial(fact_start))
        denominator = Decimal(first_num + second_num)

        if denominator == 0:
            continue
        else:
            result += sign * (fact / denominator)

        fact_start += 2
        first_num += 1
        second_num -= 2

        if change_sign:
            sign = Decimal(-1)
            change_sign = False
        else:
            sign = Decimal(1)
            change_sign = True

    return result


@app.get("/q2/{n}")
def question2_path(n: int):
    result = q2(n)
    return {"result with (path parameter)": result}


@app.get("/q2/")
def question2_query(n: int):
    result = q2(n)
    return {"result with (query parameter)": result}


@app.post("/q2/")
def question2_body(n: NumberBody):
    result = q2(n.n)
    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 می‌شود. خط ۱۴ و ۱۵، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می‌شود. این مدل یک عدد (n) از نوع int تعریف می‌کند که مقدار پیش‌فرض 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 مستقیما 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 ۵۰، n: int در امضای تابع استفاده شده است. بعد از انجام محاسبات، در خط ۵۲ نتیجه برگردانده می‌شود.

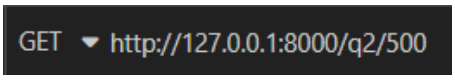
در خطوط ۵۵ تا ۵۸، با استفاده از خط ۵۵، این مسیر نیز از نوع GET است، اما در اینجا پارامتر عدد به‌صورت (Query Parameter) دریافت می‌شود، به این معنا که مقدار عدد بعد از علامت در URL وارد می‌شود. این پارامتر به‌صورت ?n=... در انتهای URL قرار می‌گیرد. در این حالت، FastAPI به‌طور خودکار این پارامتر کوئری را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

در خطوط ۶۱ تا ۶۴، با استفاده از خط ۶۱، 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 n را از JSON دریافت کرده و آن را به تابع q2 می‌فرستد. بعد از انجام محاسبات، در خط ۶۴ نتیجه برگردانده می‌شود.

بررسی خروجی ها :

خروجی برای Path Parameters :

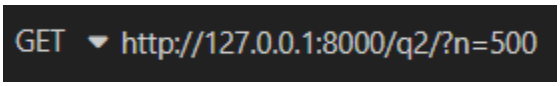
درخواست به سرور :



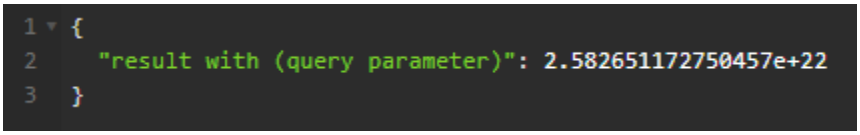
پاسخ سرور :

خروجی برای Query Parameters :

درخواست به سرور :

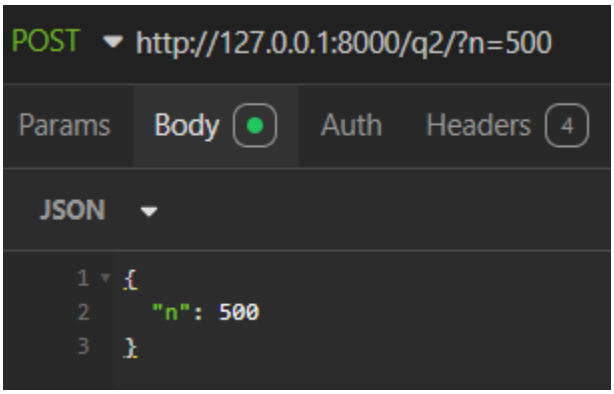


پاسخ سرور :

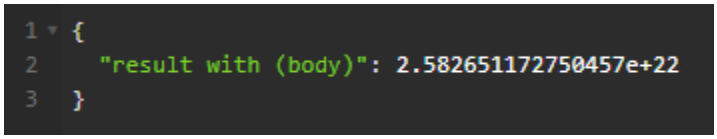


خروجی برای Request Body :

درخواست به سرور :



پاسخ سرور :



حل سوال 3 :

```python
"""
3- Write a program to print all 4 digits numbers (between 1000 and 9999) that the sum of the first
and second digits is equal with the product of the third and forth digits.
3466: 6 + 6 = 3 × 4
Output: 1110, 1101, ..., 2999 , ... , 3466 , ...
"""

from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class RangeBody(BaseModel):
    start: int = 1000
    end: int = 10000


def q3(start: int, end: int):
    results = []
    for num in range(start, end):
        number = str(num)
        digit1 = int(number[0])
        digit2 = int(number[1])
        digit3 = int(number[2])
        digit4 = int(number[3])
        if digit1 + digit2 == digit3 * digit4:
            results.append(num)
    return results


@app.get("/q3/{start}-{end}")
def question3_path(start: int, end: int):
    results = q3(start, end)
    return {"result with (path parameter)": results}


@app.get("/q3/")
def question3_query(start: int, end: int):
    results = q3(start, end)
    return {"result with (query parameter)": results}

```

```python

@app.post("/q3/")
def question3_body(range_body: RangeBody):
    results = q3(range_body.start, range_body.end)
    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 می‌شود. خط ۱۴ و ۱۵، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می‌شود. این مدل دو عدد start از نوع int و با مقدار اولیه ۱۰۰۰ و end از نوع int و با مقدار اولیه ۱۰۰۰۰ تعریف می‌کند.

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 مستقیما در URL دریافت می‌شود. کاربر باید اعداد 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 ۳۳، start: int و end: int در امضای تابع استفاده شده اند. بعد از انجام محاسبات، در خط ۳۵ نتیجه برگردانده می‌شود.

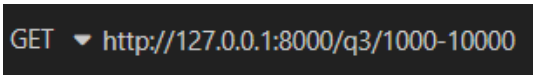
در خطوط ۳۸ تا ۴۱، با استفاده از خط ۳۸، این مسیر نیز از نوع GET است، اما در اینجا پارامترها به‌صورت (Query Parameter) دریافت می‌شوند، به این معنا که مقدار اعداد بعد از علامت در URL وارد می‌شوند. این پارامترها به‌صورت ?start=…&end=… در انتهای URL قرار می‌گیرد (نکته ای که اینجا وجود دارد این است که بین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

در خطوط ۴۴ تا ۴۷، با استفاده از خط ۴۴، این مسیر از نوع POST است و داده‌ها از طریق (Request Body) به سرور ارسال می‌شوند. در اینجا از مدل RangeBody 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 مقدارهای start و end را از JSON دریافت کرده و آن را به تابع q3 می‌فرستد. بعد از انجام محاسبات، در خط ۴۷ نتیجه برگردانده می‌شود.

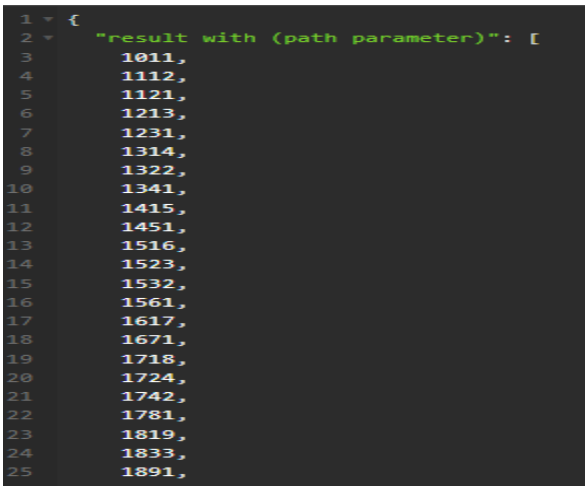
بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :



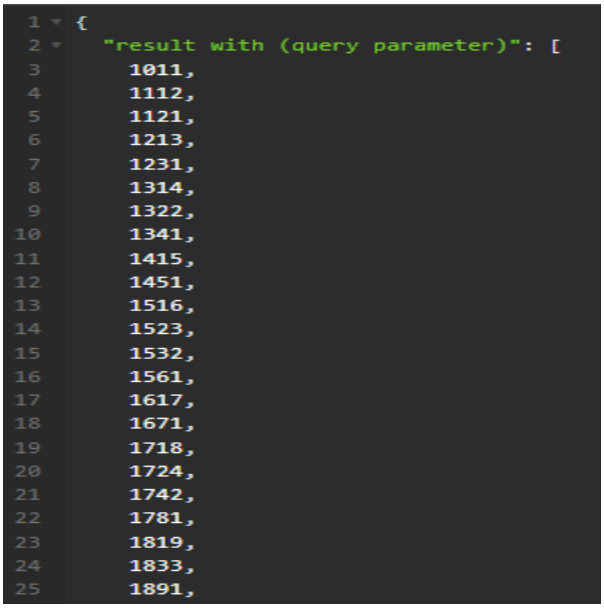
پاسخ سرور :



خروجی برای Query Parameters :

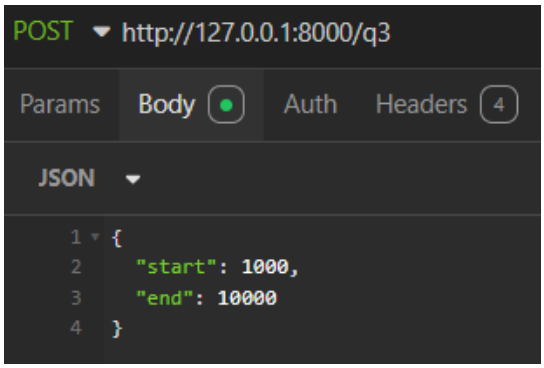
درخواست به سرور :

پاسخ سرور :

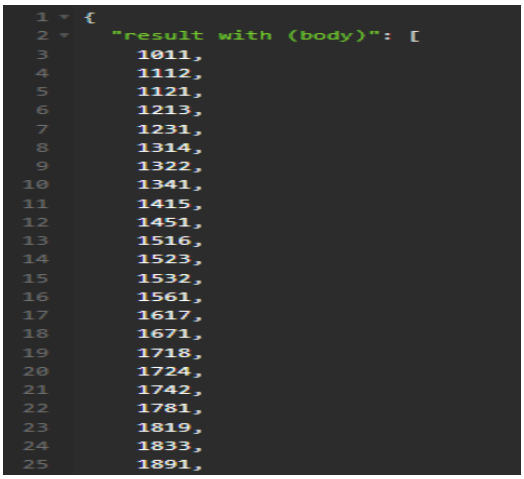


خروجی برای Request Body :

درخواست به سرور :



پاسخ سرور :



حل سوال 4 :

```python
"""
4- Write a program to print all 3 digits numbers (between a 100 and 999) that does not have odd digits.
Consider 0 as even digit.
Output: 200, 202,204, 206,208,220,222,…
"""
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class RangeBody(BaseModel):
    start: int = 100
    end: int = 999


def q4(start: int, end: int):
    results = []
    for num in range(start, end):
        number = str(num)
        digit1 = int(number[0])
        digit2 = int(number[1])
        digit3 = int(number[2])
        if digit1 % 2 == 0 and digit2 % 2 == 0 and digit3 % 2 == 0:
            results.append(num)
    return results


@app.get("/q4/{start}-{end}")
def question4_path(start: int, end: int):
    results = q4(start, end)
    return {"result with (path parameter)": results}


@app.get("/q4/")
def question4_query(start: int, end: int):
    results = q4(start, end)
    return {"result with (query parameter)": results}


@app.post("/q4/")
def question4_body(range_body: RangeBody):
    results = q4(range_body.start, range_body.end)
    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 می‌شود. خط ۱۲ و ۱۴، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می‌شود. این مدل دو عدد start از نوع int و با مقدار اولیه ۱۰۰ و end از نوع int و با مقدار اولیه ۹۹۹ تعریف می‌کند.

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 مستقیما در URL دریافت می‌شود. کاربر باید اعداد 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 ۳۰، start: int و end: int در امضای تابع استفاده شده اند. بعد از انجام محاسبات، در خط ۳۲ نتیجه برگردانده می‌شود.

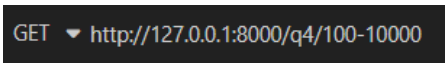
در خطوط ۳۵ تا ۳۸، با استفاده از خط ۳۵، این مسیر نیز از نوع GET است، اما در اینجا پارامترها به‌صورت (Query Parameter) دریافت می‌شوند، به این معنا که مقدار اعداد بعد از علامت در URL وارد می‌شوند. این پارامترها به‌صورت ?start=…&end=… در انتهای URL قرار می‌گیرد. در این حالت، FastAPI به‌طور خودکار این پارامترهای کوئری را به متغیرهای start با نوع int و end با نوع int تبدیل می‌کند. بعد از انجام محاسبات، در خط ۳۸ نتیجه برگردانده می‌شود.

در خطوط ۴۱ تا ۴۴، با استفاده از خط ۴۱، این مسیر از نوع POST است و داده‌ها از طریق (Request Body) به سرور ارسال می‌شوند. در اینجا از مدل RangeBody 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 مقدارهای start و end را از JSON دریافت کرده و آن را به تابع q4 می‌فرستد. بعد از انجام محاسبات، در خط ۴۴ نتیجه برگردانده می‌شود.

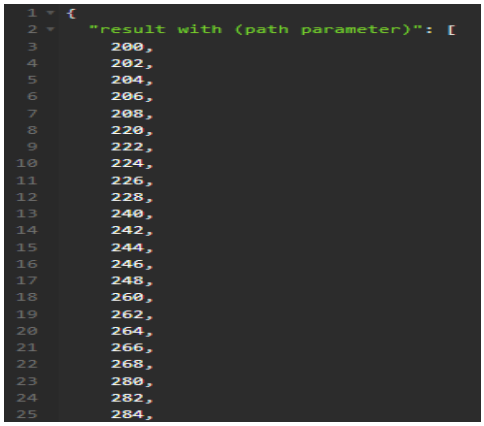
بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :



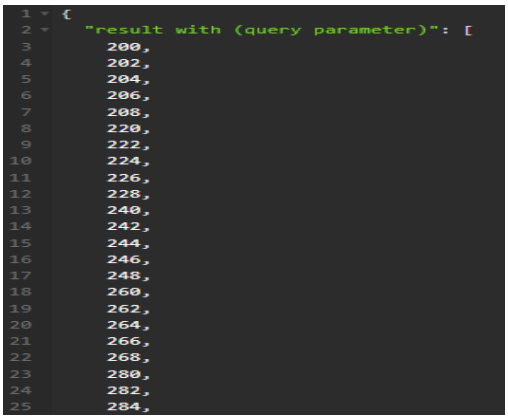
پاسخ سرور :



خروجی برای Query Parameters :

درخواست به سرور :

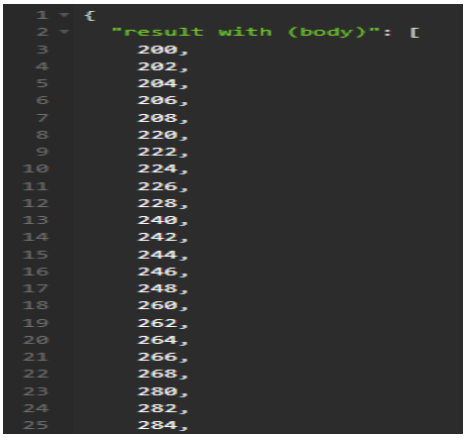
پاسخ سرور :



خروجی برای Request Body :

درخواست به سرور :

پاسخ سرور :



حل سوال 5 :

```python
"""
5 - Write a program to get n from user and print the following pattern:
Input: n=8
Output:
1
2 4
3 6 9
4 8 12 16
5 10 15 20 25
6 12 18 24 30 36
7 14 21 28 35 42 49
8 16 24 32 40 48 56 64
"""
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class PatternBody(BaseModel):
    n: int = 8


def q5(n: int):
    result = []
    for i in range(1, n + 1):
        row = []
        for j in range(1, i + 1):
            row.append(f"{i*j}")
        result.append(" ".join(row))
    return result


@app.get("/q5/{n}")
def question5_path(n: int):
    result = q5(n)
    return {"result with (path parameter)": result}


@app.get("/q5/")
def question5_query(n: int):
    result = q5(n)
    return {"result with (query parameter)": result}


@app.post("/q5/")
def question5_body(pattern_body: PatternBody):
    result = q5(pattern_body.n)
    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 می‌شود. خط ۲۰ و ۲۱، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می‌شود. این مدل عدد n از نوع int و با مقدار اولیه ۸ تعریف می‌کند.

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 مستقیما 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 ۳۵، n: int در امضای تابع استفاده شده اند. بعد از انجام محاسبات، در خط ۳۷ نتیجه برگردانده می‌شود.

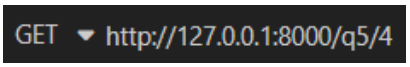
در خطوط ۴۰ تا ۴۳، با استفاده از خط ۴۰، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به‌صورت (Query Parameter) دریافت می‌شود، به این معنا که مقدار عدد بعد از علامت ? در URL وارد می‌شوند. این پارامتر به‌صورت n=... در انتهای URL قرار می‌گیرد. در این حالت، FastAPI به‌طور خودکار این پارامتر کوئری را به متغیر n با نوع int تبدیل می‌کند. بعد از انجام محاسبات، در خط ۴۳ نتیجه برگردانده می‌شود.

در خطوط ۴۶ تا ۴۹، با استفاده از خط ۴۶، این مسیر از نوع POST است و داده‌ها از طریق (Request Body) به سرور ارسال می‌شوند. در اینجا از مدل PatternBody 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 n را از JSON دریافت کرده و آن را به تابع q5 می‌فرستد. بعد از انجام محاسبات، در خط ۴۹ نتیجه برگردانده می‌شود.

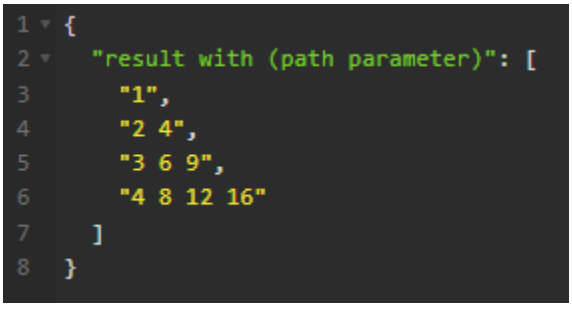
بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

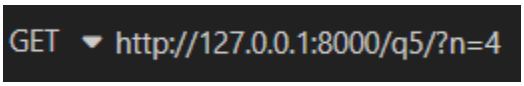


پاسخ سرور :

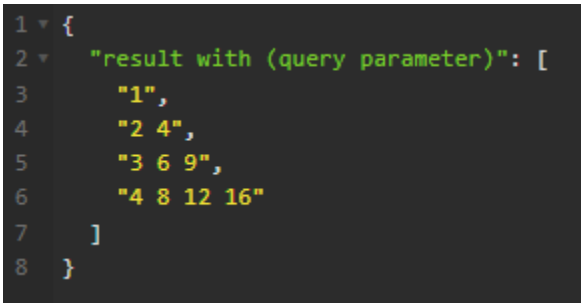


خروجی برای Query Parameters :

درخواست به سرور :



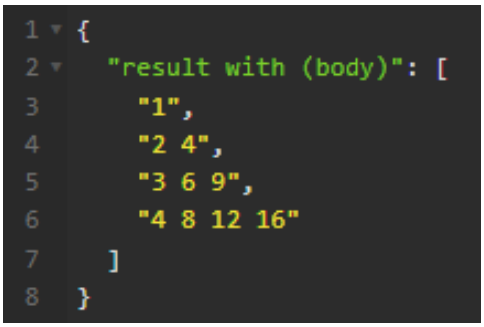
پاسخ سرور :



خروجی برای Request Body :

درخواست به سرور :

پاسخ سرور :



حل سوال 6 :

```python
"""
6 - Write a program to get n and then n numbers from user and compute maximum, minimum, average and
standard deviation. Test your program for 5 different inputs.
input:                          output:
Enter the n:5                   Maximum is: 84
Enter number 1:12               Minimum is: 9
Enter number 2:15               Average is: 26.2
Enter number 3:9                Standard Deviation is: 28.96
Enter number 4:11
Enter number 5:84
"""
from fastapi import FastAPI
from pydantic import BaseModel
import math

app = FastAPI()


class NumbersBody(BaseModel):
    numbers: list[float]



def q6(numbers: list[float]):
    maximum = max(numbers)
    minimum = min(numbers)
    average = sum(numbers) / len(numbers)
    variance = sum((x - average) ** 2 for x in numbers) / len(numbers)
    std = math.sqrt(variance)
    return maximum, minimum, average, std


@app.get("/q6/{numbers}")
def question6_path(numbers: str):
    number_list = list(map(float, numbers.split(',')))
    maximum, minimum, average, std = q6(number_list)
    return {
        "Type result": "path parameter",
        "Maximum": maximum,
        "Minimum": minimum,
        "Average": float(f"{average:.4f}"),
        "Standard Deviation": float(f"{std:.4f}")
    }


@app.get("/q6/")
def question6_query(numbers: str):
    number_list = list(map(float, numbers.split(',')))
    maximum, minimum, average, std = q6(number_list)
    return {
        "Type result": "query parameter",
        "Maximum": maximum,
        "Minimum": minimum,
        "Average": float(f"{average:.4f}"),
        "Standard Deviation": float(f"{std:.4f}")
    }


```

```python
@app.post("/q6/")
def question6_body(numbers_body: NumbersBody):
    numbers = numbers_body.numbers
    maximum, minimum, average, std = q6(numbers)
    return {
        "Type result": "body",
        "Maximum": maximum,
        "Minimum": minimum,
        "Average": float(f"{average:.4f}"),
        "Standard Deviation": float(f"{std:.4f}")
    }
```

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 می‌شود. خط ۱۹ و ۲۰، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می‌شود. این مدل عدد numbers از نوع لیست float تعریف می‌کند.

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 مستقیما در URL دریافت می‌شود. کاربر باید رشته اعداد مورد نظر خود را به‌طور مستقیم به‌عنوان بخشی از URL وارد کند (بین اعداد را با , جدا کند). بعد از انجام محاسبات، در خط ۳۶ نتیجه برگردانده می‌شود.

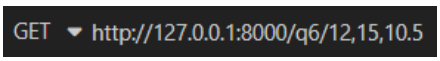
در خطوط ۴۵ تا ۵۵، با استفاده از خط ۴۵، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به‌صورت (Query Parameter) دریافت می‌شود، به این معنا که مقدار رشته بعد از علامت در URL وارد می‌شود. این پارامتر به‌صورت ?numbers=... در انتهای URL قرار می‌گیرد. بعد از انجام محاسبات، در خط ۴۹ نتیجه برگردانده می‌شود.

در خطوط ۵۸ تا ۶۸، با استفاده از خط ۵۸، این مسیر از نوع POST است و داده‌ها از طریق (Request Body) به سرور ارسال می‌شوند. در اینجا از مدل NumbersBody 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 numbers را از JSON دریافت کرده، اعداد را درون لیستی تفکیک و ذخیره می کند و لیست اعداد را به تابع q6 می‌فرستد. بعد از انجام محاسبات، در خط ۶۲ نتیجه برگردانده می‌شود.

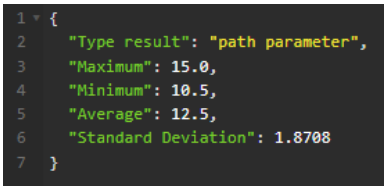
بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :



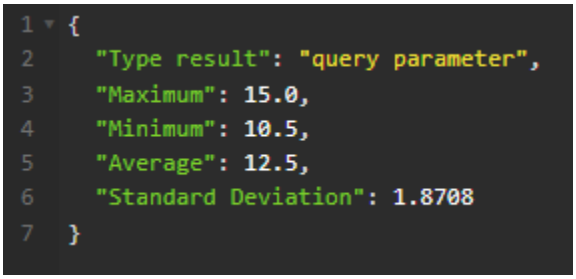
پاسخ سرور :



خروجی برای Query Parameters :

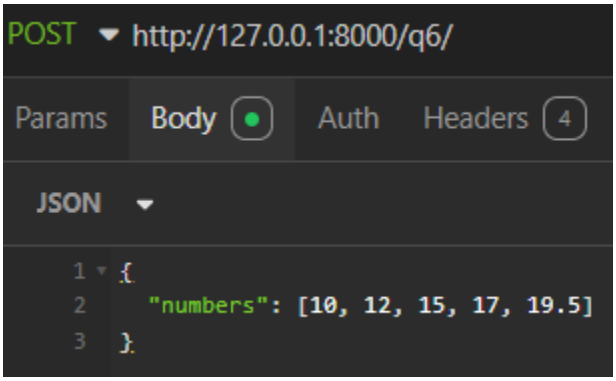
درخواست به سرور :

پاسخ سرور :

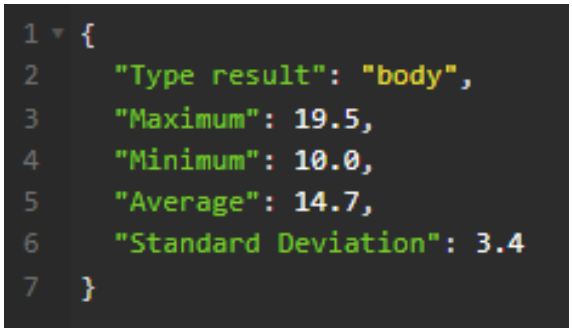


خروجی برای Request Body :

درخواست به سرور :



پاسخ سرور :



حل سوال 7 :

```python
"""
7 - Write 4 functions for question 6. Get 5 numbers from user in main.
Send numbers for Max1, Min1, Ave1 and STD1 functions. Max1 gives n numbers and return maximum.
Min1 gives n number and return minimum. Ave1 and STD1 give n number and print average and
standard deviation in their own body and do not return any value. Test your program for 5 different inputs.
"""
from fastapi import FastAPI
from pydantic import BaseModel
import math

app = FastAPI()


class NumbersBody(BaseModel):
    numbers: list[float]


def Max1(numbers: list[float]):
    maximum = numbers[0]
    for num in numbers:
        if num > maximum:
            maximum = num
    return maximum


def Min1(numbers: list[float]):
    minimum = numbers[0]
    for num in numbers:
        if num < minimum:
            minimum = num
    return minimum


def Ave1(numbers: list[float]):
    total = sum(numbers)
    average = total / len(numbers)
    return average


def STD1(numbers: list[float]):
    ave = Ave1(numbers)
    total = sum((num - ave) ** 2 for num in numbers)
    std = math.sqrt(total / len(numbers))
    return std


@app.get("/q7/{numbers}")
def question7_path(numbers: str):
    number_list = list(map(float, numbers.split(',')))
    return {
        "Type result": "path parameter",
        "Maximum": Max1(number_list),
        "Minimum": Min1(number_list),
        "Average": float(f"{Ave1(number_list):.4f}"),
        "Standard Deviation": float(f"{STD1(number_list):.4f}")
    }
```

```python
@app.get("/q7/")
def question7_query(numbers: str):
    numbers_list = list(map(float, numbers.split(',')))
    return {
        "Type result": "query parameter",
        "Maximum": Max1(numbers_list),
        "Minimum": Min1(numbers_list),
        "Average": float(f"{Ave1(numbers_list):.4f}"),
        "Standard Deviation": float(f"{STD1(numbers_list):.4f}")
    }


@app.post("/q7/")
def question7_body(numbers_body: NumbersBody):
    numbers = numbers_body.numbers
    return {
        "Type result": "body",
        "Maximum": Max1(numbers),
        "Minimum": Min1(numbers),
        "Average": float(f"{Ave1(numbers):.4f}"),
        "Standard Deviation": float(f"{STD1(numbers):.4f}")
    }
```

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 می‌شود. خط ۱۴ و ۱۵، یک کلاس با استفاده از pydantic.BaseModel است که برای دریافت ورودی از طریق درخواست POST استفاده می‌شود. این مدل عدد numbers از نوع لیست float تعریف می‌کند.

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 مستقیما در URL دریافت می‌شود. کاربر باید رشته اعداد مورد نظر خود را به‌طور مستقیم به‌عنوان بخشی از URL وارد کند (بین اعداد را با , جدا کند). بعد از انجام محاسبات، در خط ۵۶ نتیجه برگردانده می‌شود.

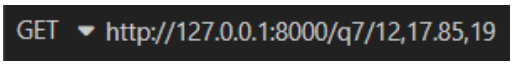
در خطوط ۵۹ تا ۶۸، با استفاده از خط ۵۹، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به‌صورت (Query Parameter) دریافت می‌شود، به این معنا که مقدار رشته بعد از علامت در URL وارد می‌شود. این پارامتر به‌صورت ?numbers=... در انتهای URL قرار می‌گیرد. بعد از انجام محاسبات، در خط ۶۸ نتیجه برگردانده می‌شود.

در خطوط ۷۱ تا ۸۰، با استفاده از خط ۷۱، این مسیر از نوع POST است و داده‌ها از طریق (Request Body) به سرور ارسال می‌شوند. در اینجا از مدل NumbersBody 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 numbers را از JSON دریافت کرده، اعداد را درون لیستی تفکیک و ذخیره می کند و لیست اعداد را به هرکدام از توابع می‌فرستد. بعد از انجام محاسبات، در خط ۸۰ نتیجه برگردانده می‌شود.

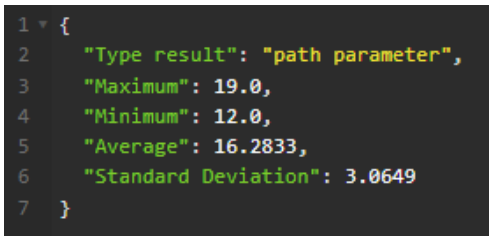
بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

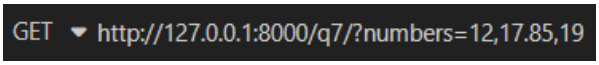


پاسخ سرور :

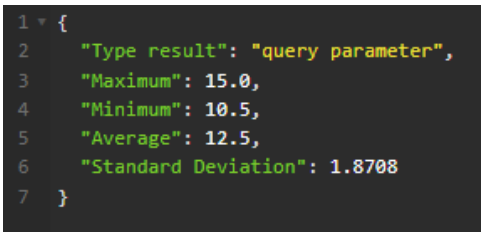


خروجی برای Query Parameters :

درخواست به سرور :



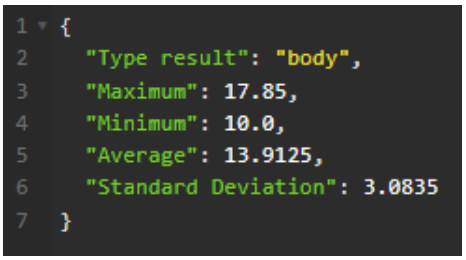
پاسخ سرور :



خروجی برای Request Body :

درخواست به سرور :

پاسخ سرور :



حل سوال 8 :

```python
"""
8 - Give a number with 5 digits in main. If the number is not a 5 digits number, ask repeatedly from user
to enter a valid number. Send the number for F1 function to find and return the maximum digit of the number.
In the next step, send the maximum digit and the input number for F2 function to delete the maximum digit from
number and return it. Finally, print the final output in main as shown in figure below.
"""
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


class NumberBody(BaseModel):
    number: int


def F1(number: int):
    digits = []
    while number > 0:
        digit = number % 10
        digits.append(digit)
        number //= 10
    max_digit = max(digits)
    return max_digit


def F2(number: int, digit: int):
    number = str(number).replace(f"{digit}", "").strip()
    return number


@app.get("/q8/{number}")
def question8_path(number: int):
    if len(str(number)) == 5:
        max_digit = F1(number)
        final_number = F2(number, max_digit)
        return {
            "Type result": "path parameter",
            "Maximum digit": max_digit,
            f"Final number without {max_digit}": final_number
        }
    else:
        return {"error": "The number is not 5 digits"}


@app.get("/q8/")
def question8_query(number: int):
    if len(str(number)) == 5:
        max_digit = F1(number)
        final_number = F2(number, max_digit)
        return {
            "Type result": "query parameter",
            "Maximum digit": max_digit,
            f"Final number without {max_digit}": final_number
        }
    else:
        return {"error": "The number is not 5 digits"}
```

```python
@app.post("/q8/")
def question8_body(data: NumberBody):
    number = data.number
    if len(str(number)) == 5:
        max_digit = F1(number)
        final_number = F2(number, max_digit)
        return {
            "Type result": "body",
            "Maximum digit": max_digit,
            f"Final number without {max_digit}": final_number
        }
    else:
        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 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 عدد number از نوع int تعریف می‌کند.

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 مستقیما در URL دریافت می‌شود. کاربر باید عدد مورد نظر خود را به‌طور مستقیم به‌عنوان 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 می‌فرستد و بعد از انجام محاسبات، در خط ۳۷ نتیجه برگردانده می‌شود. اما اگر ۵ رقمی نبود، پیغام خطایی را نمایش میدهد.

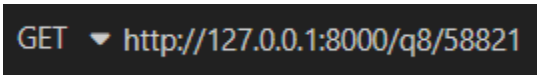
در خطوط ۴۶ تا ۵۷، با استفاده از خط ۴۶، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به‌صورت (Query Parameter) دریافت می‌شود، به این معنا که مقدار عدد بعد از علامت ? در URL وارد می‌شود. این پارامتر به‌صورت number=... در انتهای URL قرار می‌گیرد. در خط ۴۸ بررسی میکند که عدد ۵ رقمی است یا نه. اگر ۵ رقمی بود، عدد را به هرکدام از توابع می‌فرستد و بعد از انجام محاسبات، در خط ۵۱ نتیجه برگردانده می‌شود. 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 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 در خط ۶۳ بررسی میکند که عدد ۵ رقمی است یا نه. اگر ۵ رقمی بود، عدد را به هرکدام از توابع می‌فرستد و بعد از انجام محاسبات، در خط ۶۶ نتیجه برگردانده می‌شود. اما اگر ۵ رقمی نبود، پیغام خطایی را نمایش میدهد.

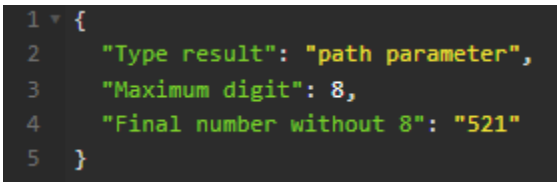
بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

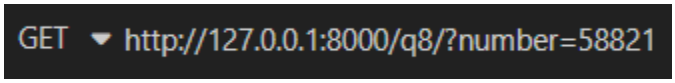


پاسخ سرور :

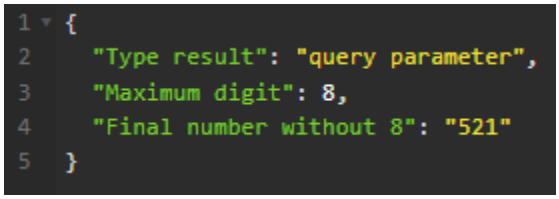


خروجی برای Query Parameters :

درخواست به سرور :

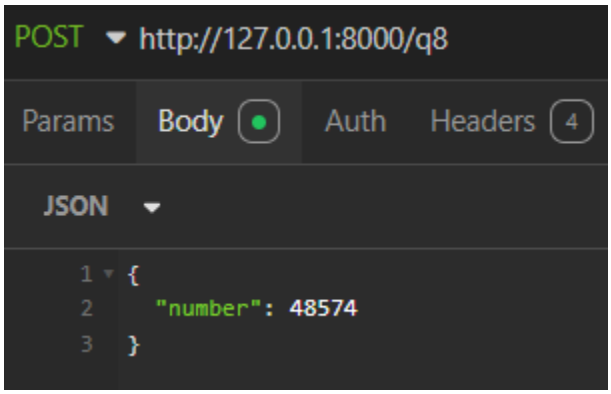


پاسخ سرور :

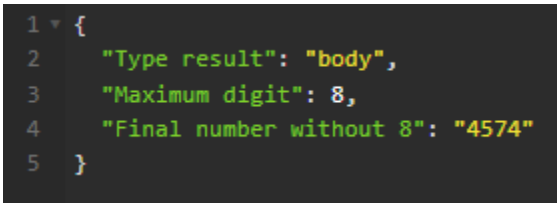


خروجی برای Request Body :

درخواست به سرور :



پاسخ سرور :



حل سوال 9 :

```python
"""
9-Global and Local variables: Run codes P1, P2, P3 and P4 and report outputs.
Explain reasons for their outputs in details.
"""
from fastapi import FastAPI

app = FastAPI()


def q9_p1():
    # P1
    def f():
        # local variable
        s1 = "I live in khorramabad"
        return s1

    # Main
    f()


def q9_p2():
    # P2
    try:
        def f():
            # local variable
            s2 = "I live in khorramabad"
            print("Inside Function:", s2)

        # Main
        f()
        return s2  # This will raise a NameError
    except NameError as e:
        # Catch the NameError and return it as a string
        return f"Error: {str(e)}"


def q9_p3():
    # P3
    def f():
        global s3
        s3 = 'I live in khorramabad.'
        print(s3)

    # Main: Global Scope
    s3 = 'I live in iran.'
    f()
    return s3


def q9_p4():
    # P4
    # This function uses global variable s
    def f():
        s4 = "I live in khorramabad."
        print(s4)

    # Main
    s4 = "I live in iran."
    f()
    return s4


```

```python
@app.get("/q9/")
def question9_path_query():
    return {
        "Type result": "path parameter and query parameter",
        "result p1 is": q9_p1(),
        "result p2 is": q9_p2(),
        "result p3 is": q9_p3(),
        "result p4 is": q9_p4()
    }


@app.post("/q9/")
def question9_body():
    return {
        "Type result": "body",
        "result p1 is": q9_p1(),
        "result p2 is": q9_p2(),
        "result p3 is": q9_p3(),
        "result p4 is": q9_p4()
    }
```

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 صرفا 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 درنظر 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 :

```json
{
  "Type result": "path parameter and query parameter",
  "result p1 is": null,
  "result p2 is": "Error: name 's2' is not defined",
  "result p3 is": "I live in iran.",
  "result p4 is": "I live in iran."
}
```

خروجی برای Request Body :

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

پاسخ سرور :

```json
{
  "Type result": "body",
  "result p1 is": null,
  "result p2 is": "Error: name 's2' is not defined",
  "result p3 is": "I live in iran.",
  "result p4 is": "I live in iran."
}
```

حل سوال 10 :

```python
"""
10-Study about Recursive Functions in python.
Explain in details that how does the following recursive function work?
"""
from fastapi import FastAPI
from pydantic import BaseModel

app = FastAPI()


def factorial(x: int):
    if x == 1:  # This is the base case
        return 1
    else:  # This is the recursive case
        return x * factorial(x - 1)


class FactorialBody(BaseModel):
    number: int


@app.get("/q10/{number}")
async def question10_path(number: int):
    if number < 1:
        return {"error": "The number should be greater than 0"}
    else:
        result = factorial(number)
        return {
            "Type result": "path parameter",
            "Factorial": result
        }


@app.get("/q10/")
async def question10_query(number: int):
    if number < 1:
        return {"error": "The number should be greater than 0"}
    else:
        result = factorial(number)
        return {
            "Type result": "query parameter",
            "Factorial": result
        }
```

```python

@app.post("/q10/")
async def question10_body(data: FactorialBody):
    number = data.number
    if number < 1:
        return {"error": "The number should be greater than 0"}
    else:
        result = factorial(number)
        return {
            "Type result": "body",
            "Factorial": result
        }
```

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 می‌شود. 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 می‌شود. این مدل عدد number از نوع int تعریف می‌کند.

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 مستقیما در URL دریافت می‌شود. کاربر باید عدد مورد نظر خود را به‌طور مستقیم به‌عنوان 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 می‌فرستد و بعد از انجام محاسبات، در خط ۲۸ نتیجه برگردانده می‌شود.

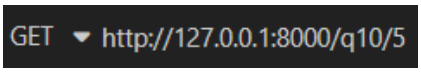
در خطوط ۳۴ تا ۴۳، با استفاده از خط ۳۴، این مسیر نیز از نوع GET است، اما در اینجا پارامتر به‌صورت (Query Parameter) دریافت می‌شود، به این معنا که مقدار عدد بعد از علامت ? در URL وارد می‌شود. این پارامتر به‌صورت number=... در انتهای URL قرار می‌گیرد. در خط ۳۶ بررسی میکند عدد کمتر از ۱ است یا نه. اگر کمتر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

در خطوط ۴۶ تا ۵۶، با استفاده از خط ۴۶، این مسیر از نوع POST است و داده‌ها از طریق (Request Body) به سرور ارسال می‌شوند. در اینجا از مدل FactorialBody 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 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 می‌فرستد و بعد از انجام محاسبات، در خط ۵۳ نتیجه برگردانده می‌شود.

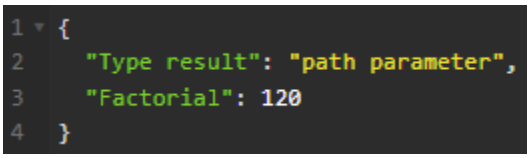
بررسی خروجی ها :

خروجی برای Path Parameters :

درخواست به سرور :

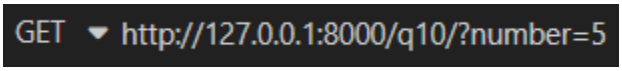


پاسخ سرور :

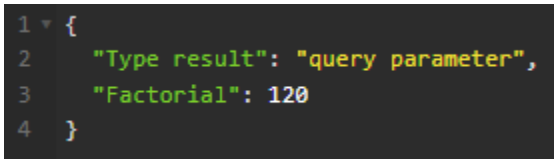


خروجی برای Query Parameters :

درخواست به سرور :



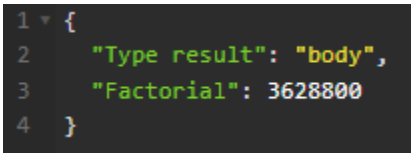
پاسخ سرور :



خروجی برای Request Body :

درخواست به سرور :

پاسخ سرور :



پایان