

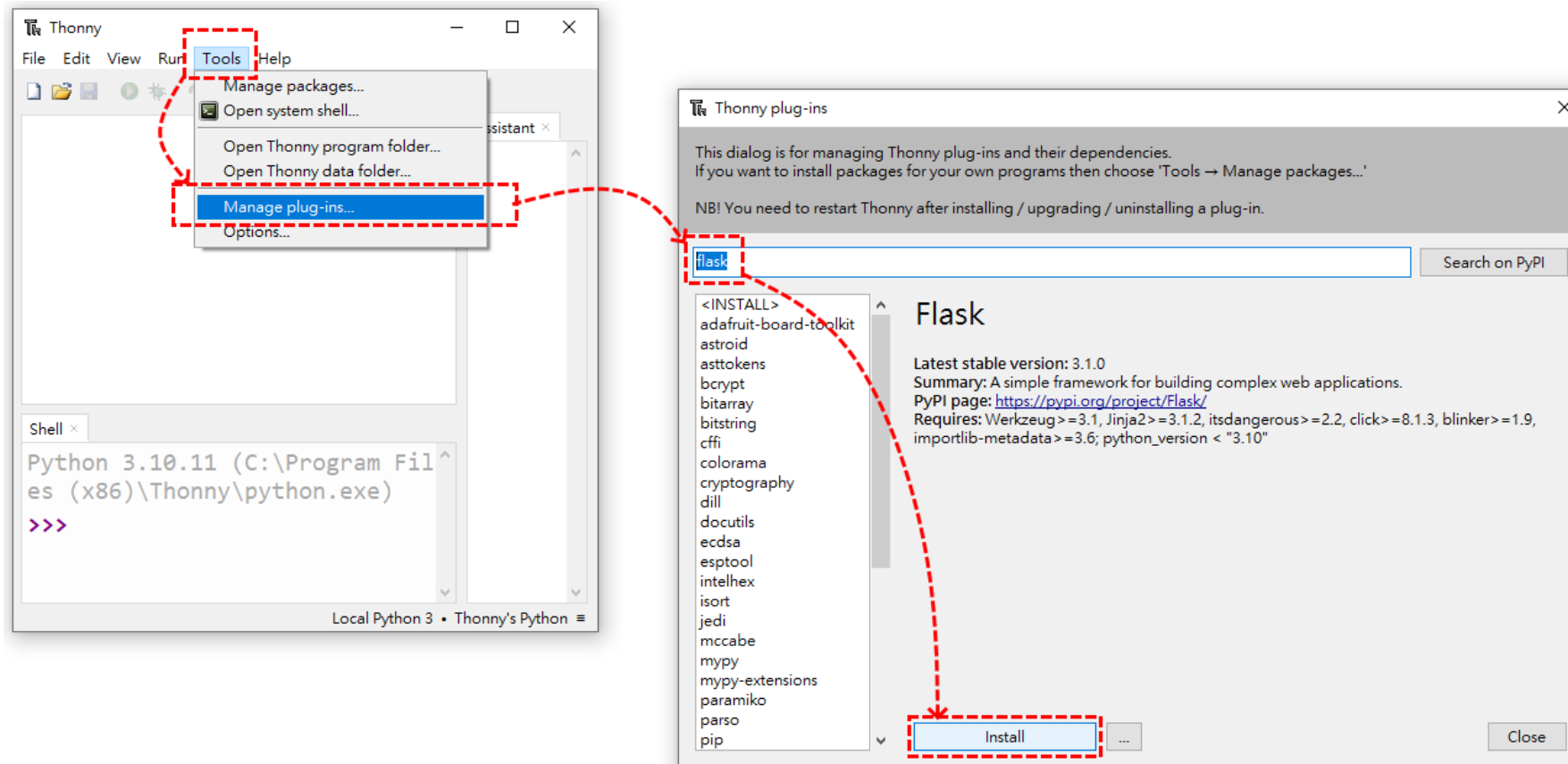
Flask Server App Example

1. Install Python IDE--Thonny

Not necessarily thonny, you can choose other familiar IDE for you.

IDE Download: <https://thonny.org/>

2. Install Flask



3. Write Code: Server1.py and Run

Library: <https://flask.palletsprojects.com/en/stable/tutorial/layout/>

The image shows a Thonny IDE window on the left and a web browser window on the right. The Thonny window has a menu bar (File, Edit, View, Run, Tools, Help) and a toolbar with a red 'Run' button. The code editor shows the following Python code in `Server1.py`:

```
1 from flask import Flask
2
3 app = Flask(__name__)
4
5
6 @app.route('/')
7 def hello():
8     return 'Hello, World!'
```

The Shell window shows the output of running the code:

```
>>> %Run Server1.py
# Running the app with options chosen by Thonny. See Help for
details.
* Serving Flask app 'Server1'
* Debug mode: off
WARNING: This is a development server. Do not use it in a prod
uction deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

The Assistant window provides feedback: "The code in `Server1.py` looks good. If it is not working as it should, then consider using some general debugging techniques. [Was it helpful or confusing?](#)"

The web browser window shows the URL `127.0.0.1:5000` and the response "Hello, World!". A red dashed arrow points from the URL in the browser to the Shell window output. A blue dashed box highlights the Python interpreter path in the Shell window: `Local Python 3 • C:\Users\...AppData\Local\Programs\Python\Python38-32\python.exe`. A blue text box at the bottom right says: "If can't run the server app, you can try different Python environments."

4. Write Code: Get Method

Library: flask

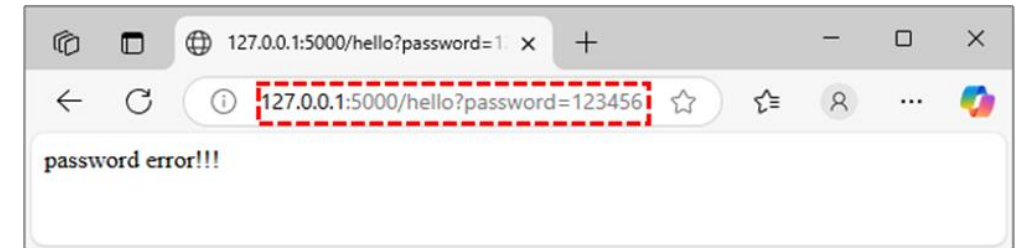
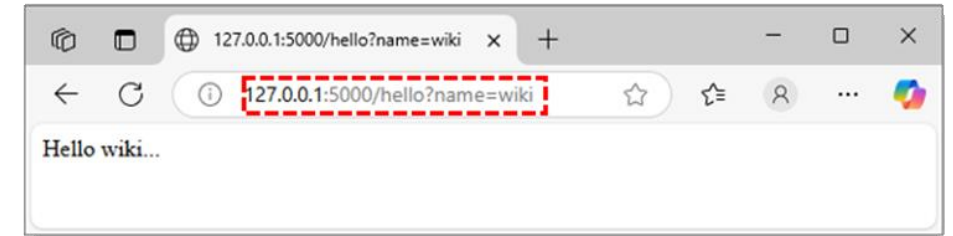
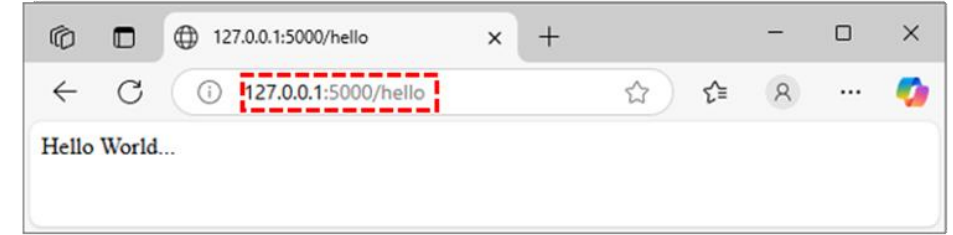
```
from flask import Flask, request, jsonify

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello World!'

# GET Method
@app.route('/hello', methods=['GET'])
def hello_get():
    name = request.args.get('name', 'World') #'World' is default
    password = request.args.get('password', 'None') #'None' is default
    if password=='None':
        return f'Hello {name}...'
    else:
        if name=='wiki' and password=='123456':
            return f'Hello {name}, welcome to system.'
        elif name=='chen' and password=='456789':
            return f'Hello {name}, welcome to system.'
        else:
            return f'password error!!!'
```

Web Output



This is a fast network transmission method. If you have passwords or personal information, please don't use this method. So **this example is a bad application.**

4. Write Code: Get Method

This is a good application.

The screenshot shows the Google Translate web interface. The browser tab is labeled 'Google 翻譯'. The address bar contains the URL `translate.google.com/?sl=zh-TW&tl=en&text=這是個不錯的運用&op=translate`, which is highlighted with a red dashed box. The page header shows the Google Translate logo and a settings icon. Below the header, the source language is set to '中文 (繁體)' and the target language is '英文'. The input text '這是個不錯的運用' is shown in a box, with its pinyin 'Zhè shìgè bùcuò de yùnyòng' below it. The output text 'This is a good application' is shown in a box. The interface also includes a microphone icon, a character count '8 / 5,000', and a '注' (note) button.

5. Write Code: Post Method

Library: form AI

```
from flask import Flask, request, jsonify
```

```
app = Flask(__name__)
```

```
@app.route('/')
def hello():
    return 'Hello World!'
```

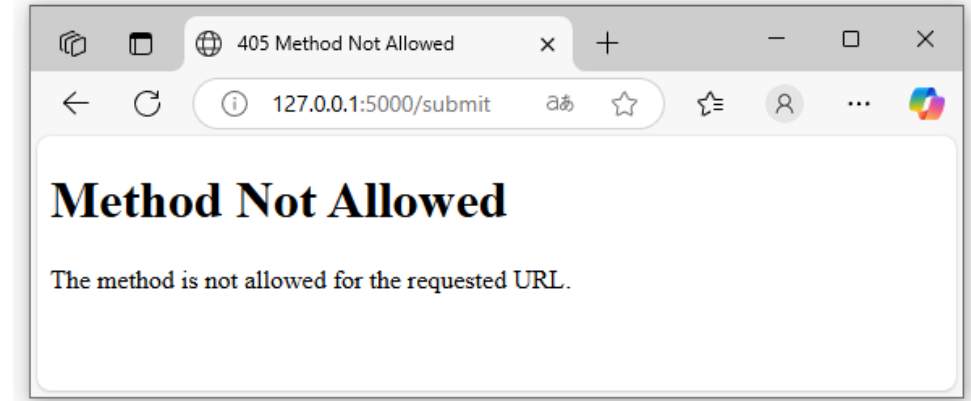
POST Method

```
@app.route('/submit', methods=['POST'])
```

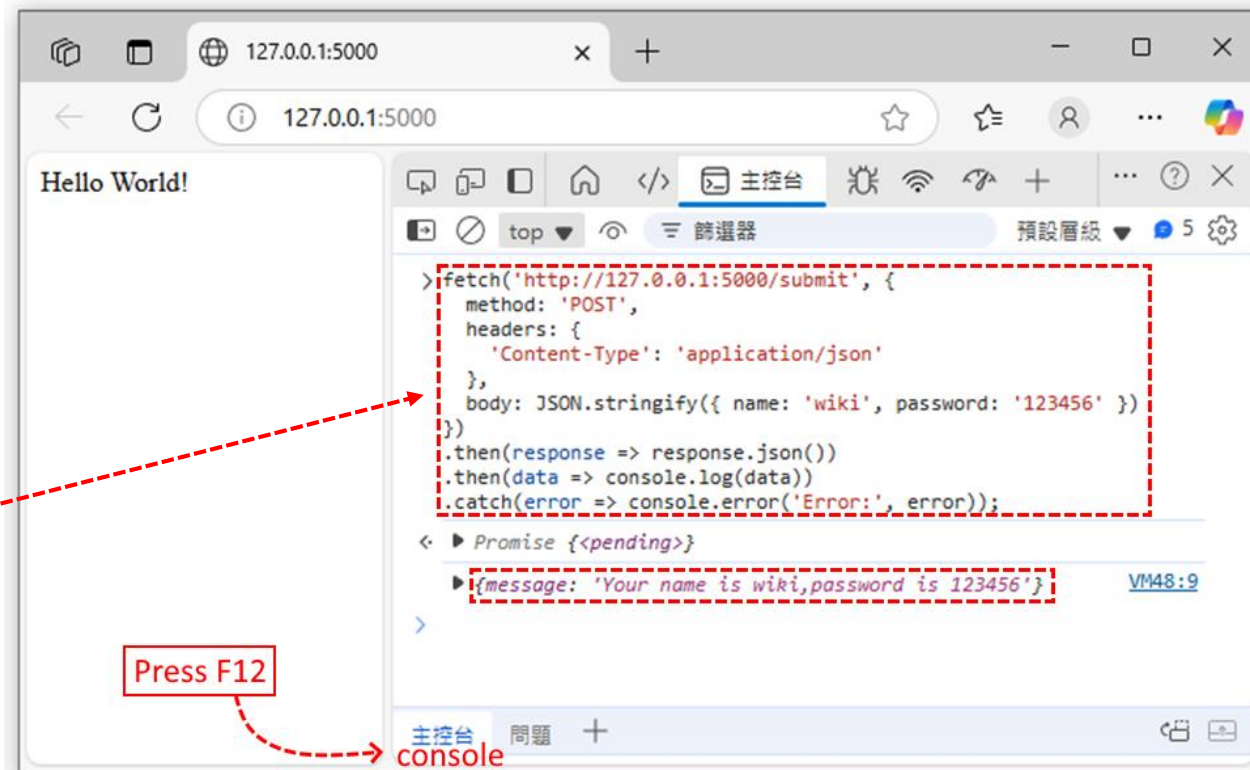
```
def submit_form():
    data = request.get_json()
    name = f"{data.get('name')}}"
    password = f"{data.get('password')}}"
    return jsonify(message=f'Your name is {name}, password is {password}')
```

```
fetch('http://127.0.0.1:5000/submit', { method: 'POST',
headers: { 'Content-Type': 'application/json' }, body:
JSON.stringify({ name: 'wiki', password:
'123456' })}).then(response => response.json()).then(data =>
console.log(data)).catch(error => console.error('Error:', error));
```

Web Output

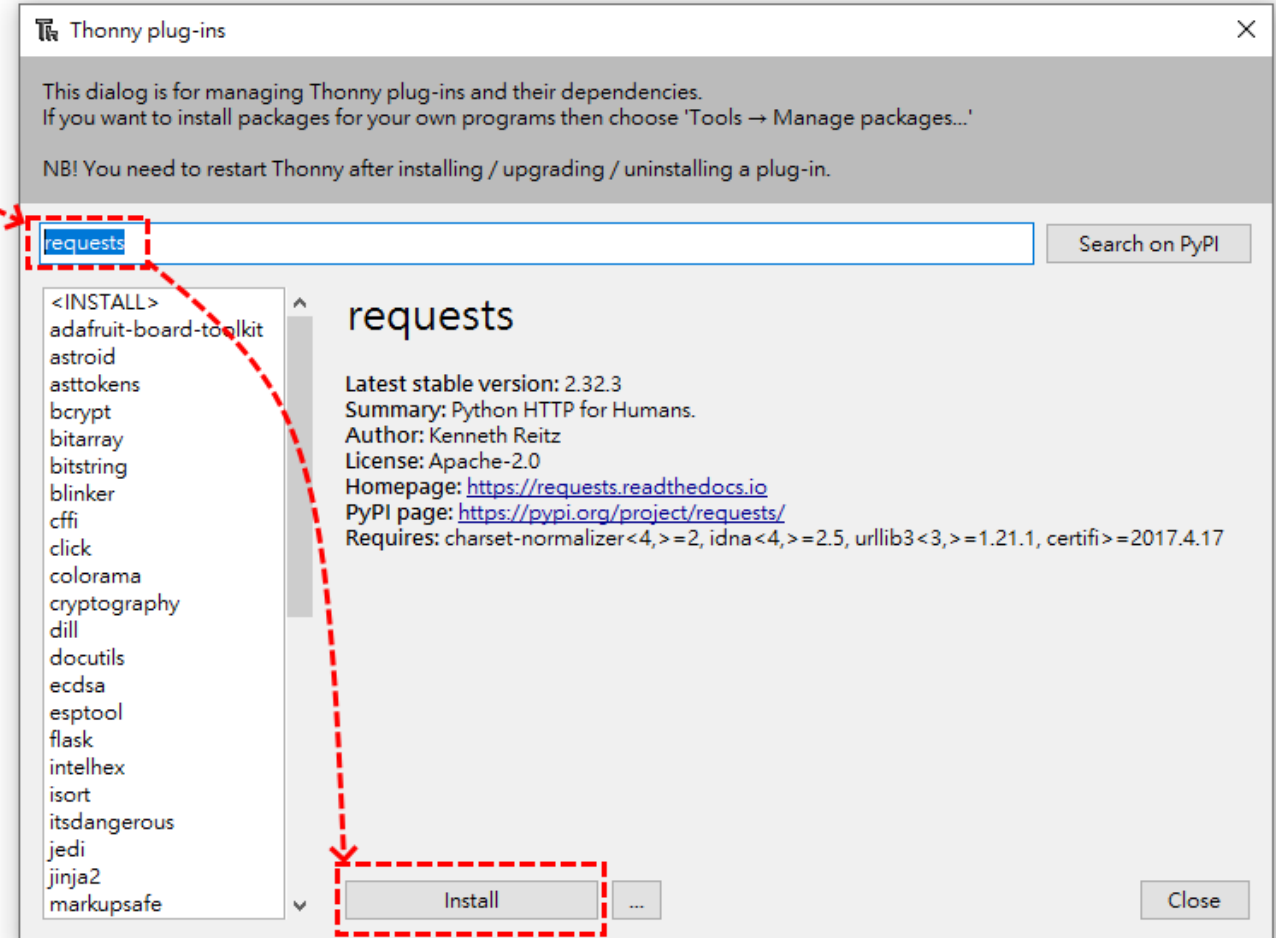
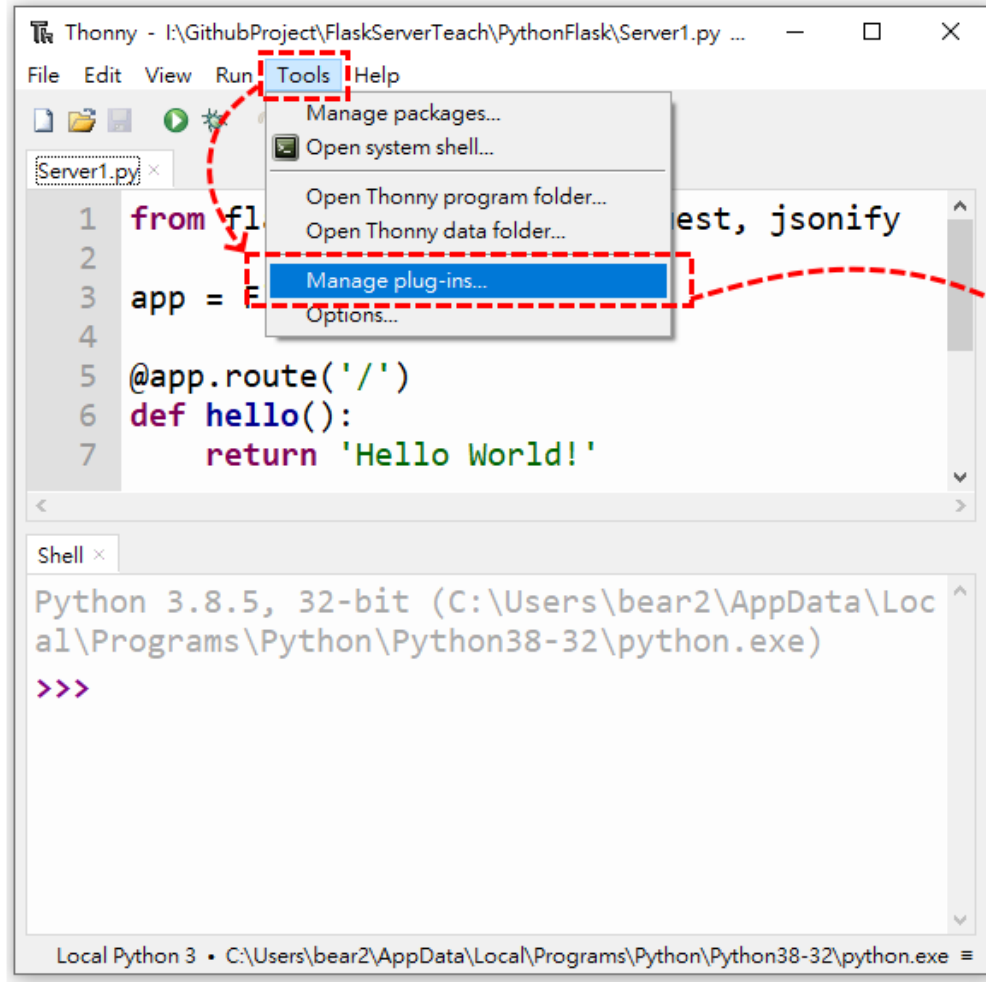


Web Console Output



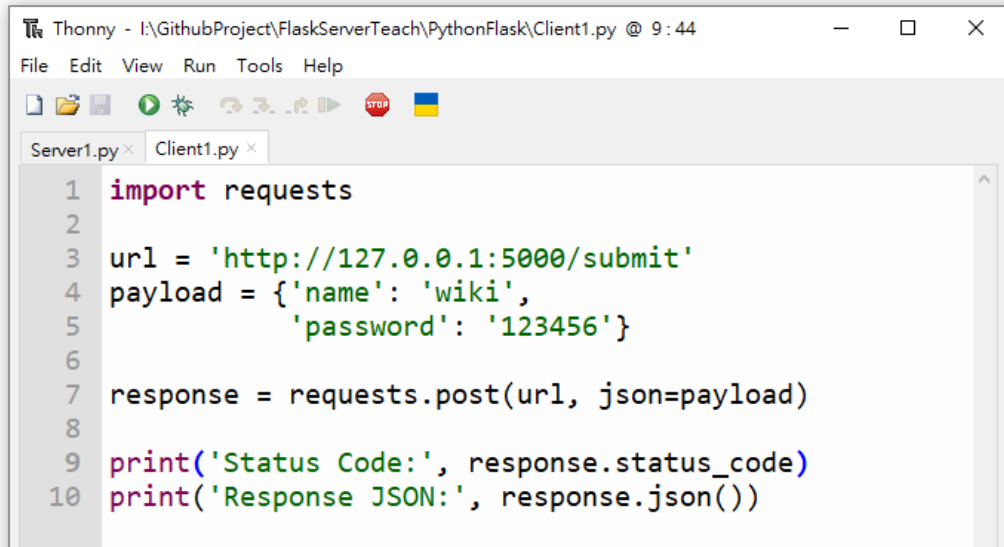
6. Write Code: Post Method Test

Step1: install requests



6. Write Code: Post Method Test

Step2: Write Code on Client1.py

A screenshot of the Thonny IDE window. The title bar shows 'Thonny - I:\GithubProject\FlaskServerTeach\PythonFlask\Client1.py @ 9:44'. The menu bar includes File, Edit, View, Run, Tools, and Help. The toolbar has icons for file operations, running, and debugging. The code editor shows the following Python code:

```
1 import requests
2
3 url = 'http://127.0.0.1:5000/submit'
4 payload = {'name': 'wiki',
5            'password': '123456'}
6
7 response = requests.post(url, json=payload)
8
9 print('Status Code:', response.status_code)
10 print('Response JSON:', response.json())
```

```
import requests
```

```
url = 'http://127.0.0.1:5000/submit'
payload = {'name': 'wiki',
           'password': '123456'}
```

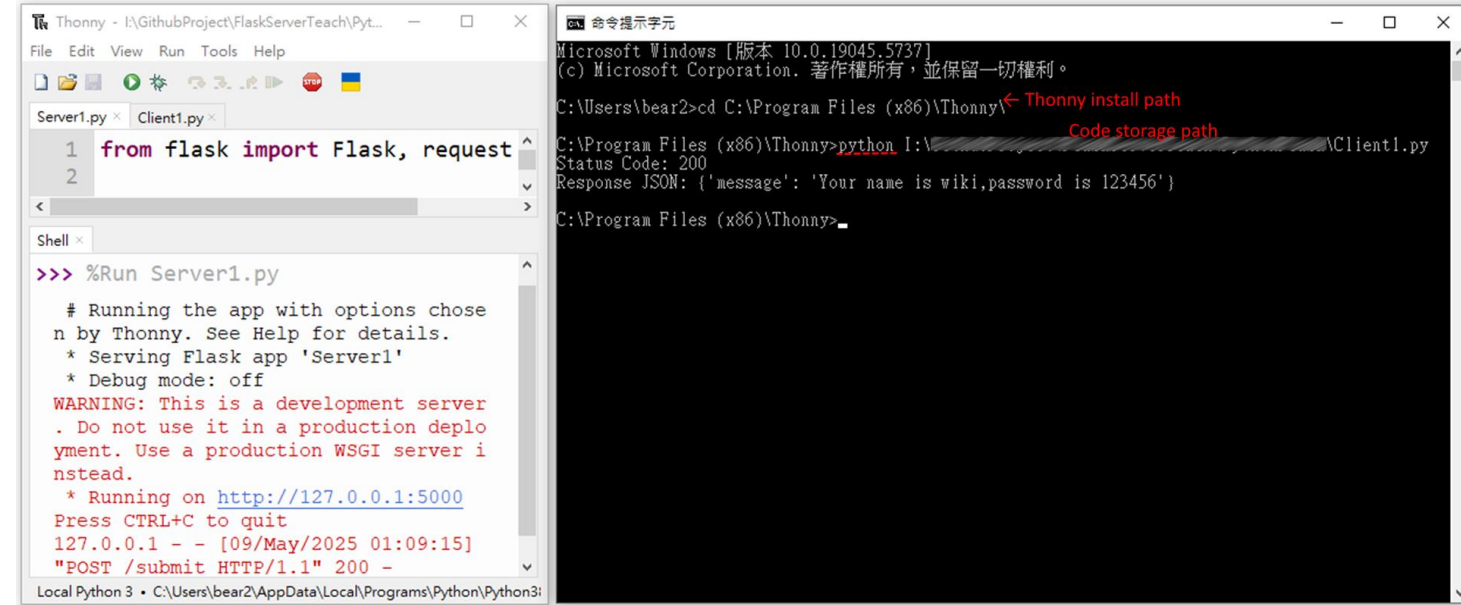
```
response = requests.post(url, json=payload)
```

```
print('Status Code:', response.status_code)
print('Response JSON:', response.json())
```

Step3: Run Server1.py and Client1.py

Server1.py Run on IDE Shell.

Client1.py Run on Terminal.

Two screenshots are shown side-by-side. The left screenshot shows the Thonny IDE with 'Server1.py' open. The code is:

```
1 from flask import Flask, request
2
```

The Shell window at the bottom shows the command to run the server:

```
>>> %Run Server1.py
# Running the app with options chosen by Thonny. See Help for details.
* Serving Flask app 'Server1'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [09/May/2025 01:09:15]
"POST /submit HTTP/1.1" 200 -
```

The right screenshot shows a Windows Command Prompt window. It contains the following commands and output:

```
Microsoft Windows [版本 10.0.19045.5737]
(c) Microsoft Corporation. 版权所有，并保留一切权利。
C:\Users\bear2>cd C:\Program Files (x86)\Thonny\
C:\Program Files (x86)\Thonny>python I:\GithubProject\PythonFlask\Client1.py
Status Code: 200
Response JSON: {'message': 'Your name is wiki,password is 123456'}
C:\Program Files (x86)\Thonny>
```

Client1.py is run on terminal, because Thonny IDE can't run two code at same time.

If Client1.py can't run, you may need to change the Python environment.

7. Write Code: Post Method Test

Step1: Write Code on Server1.py

```
from flask import Flask, request, jsonify
from datetime import datetime

app = Flask(__name__)

@app.route('/')
def hello():
    return 'Hello World!'

# POST Method
@app.route('/submit', methods=['POST'])
def submit_form():
    data = request.get_json()
    name = data.get('name')
    password = data.get('password')
    birthday = datetime.strptime((f'{data.get("birthday")}' ), "%Y-%m-%d")
    gender = data.get('gender')
    height = data.get('height')
    weight = data.get('weight')

    # bmi count
    bmi = round(weight / height **2,2)

    # age count
    today = datetime.today()
    age = today.year - birthday.year - ((today.month, today.day) < (birthday.month, birthday.day))

    return jsonify(message=f'Your AGE={age},BMI={bmi}')
```

Step2: Write Code on Client1.py

```
import requests

url = 'http://127.0.0.1:5000/submit'
payload = {'name': 'wiki',
           'password': '123456',
           'birthday': '1990-04-24',
           'height': 1.75,
           'weight': 60}

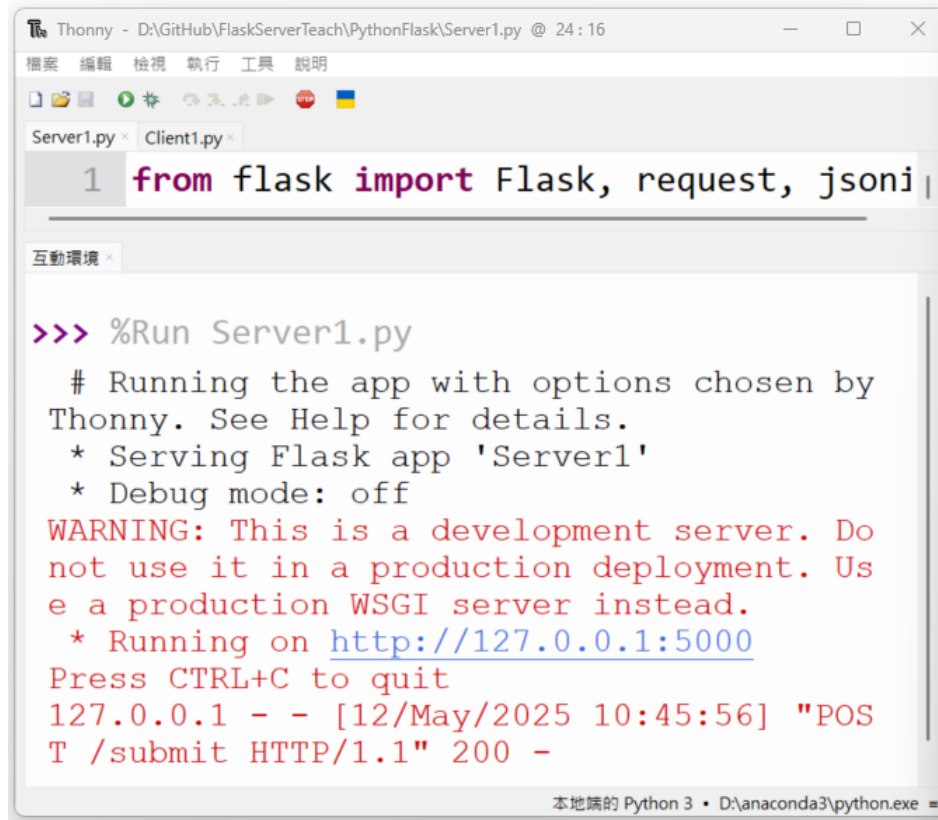
response = requests.post(url, json=payload)

print('Status Code:', response.status_code)
print('Response JSON:', response.json())
```

7. Write Code: Post Method Test

Step3 Run Server1.py and Client1.py

Server1.py Run on IDE Shell.

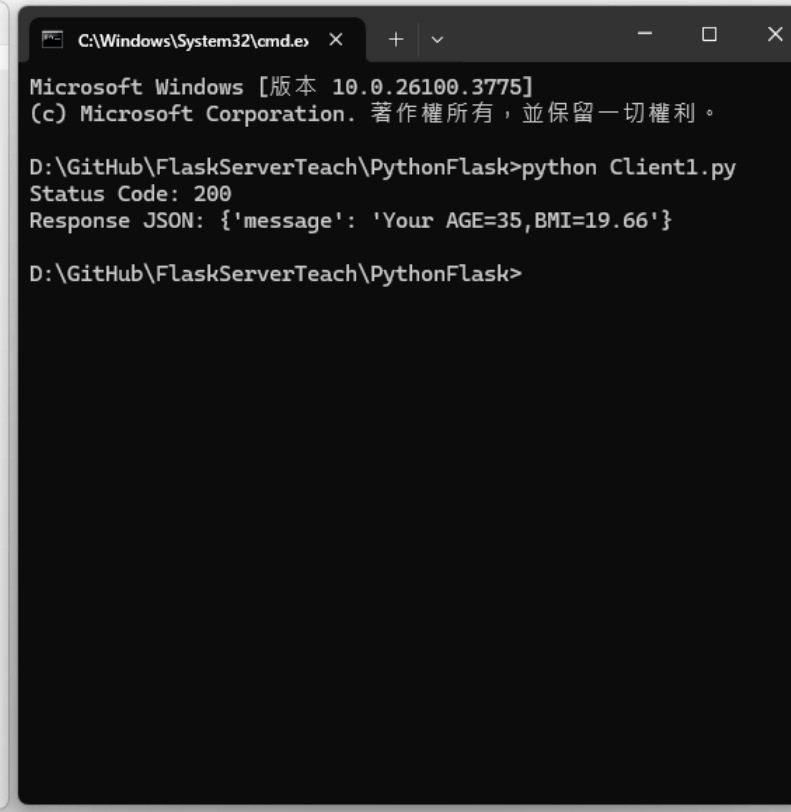


The screenshot shows the Thonny IDE interface. The top pane displays the code for Server1.py, which includes imports for Flask, request, and jsonify, and a route for POST /submit. The bottom pane shows the interactive shell output after running the code. The output includes a warning about using a development server and a successful POST request response.

```
1 from flask import Flask, request, jsonify

>>> %Run Server1.py
# Running the app with options chosen by
Thonny. See Help for details.
* Serving Flask app 'Server1'
* Debug mode: off
WARNING: This is a development server. Do
not use it in a production deployment. Us
e a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [12/May/2025 10:45:56] "POS
T /submit HTTP/1.1" 200 -
```

Client1.py Run on Terminal.



The screenshot shows a Windows command prompt window. It displays the command to run Client1.py and the resulting output, which shows a status code of 200 and a JSON response containing a message about age and BMI.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [版本 10.0.26100.3775]
(c) Microsoft Corporation. 著作權所有，並保留一切權利。

D:\GitHub\FaskServerTeach\PythonFlask>python Client1.py
Status Code: 200
Response JSON: {'message': 'Your AGE=35,BMI=19.66'}

D:\GitHub\FaskServerTeach\PythonFlask>
```

Client1.py is run on terminal, because Thonny IDE can't run two code at same time.

If Client1.py can't run, you may need to change the Python environment.

8. Write Code: Post Method Test

Step1: Write Code on Server1.py

```
from flask import Flask, request, jsonify
from datetime import datetime
import math
import pandas as pd
app = Flask(__name__)
# POST Method
@app.route('/submit', methods=['POST'])
def submit_form():
    data = request.get_json()
    name = data.get('name')
    password = data.get('password')
    birthday = datetime.strptime((f"{data.get('birthday')}" ), "%Y-%m-%d")
    gender = data.get('gender')
    height = data.get('height')
    weight = data.get('weight')
    bmi = round(weight / height **2,2)# bmi count
    today = datetime.today()# age count
    age = today.year - birthday.year - ((today.month, today.day) < (birthday.month, birthday.day))
    df = pd.read_csv('BMI_Normal.csv')
    matched_rows = df[df['age'] == age].iloc[0]
    print(matched_rows)
    bmi_judge = 'wrong data'
    if gender=='male':
        if bmi<matched_rows['male_min']:
            bmi_judge='Too thin'
        elif bmi>=matched_rows['male_min'] and bmi<=matched_rows['male_max']:
            bmi_judge='Normal'
        else:
            bmi_judge='Too fat'
    elif gender=='female':
        if bmi<matched_rows['female_min']:
            bmi_judge='Too thin'
        elif bmi>=matched_rows['female_min'] and bmi<=matched_rows['female_max']:
            bmi_judge='Normal'
        else:
            bmi_judge='Too fat'
    return jsonify(message=f'Your AGE={age},BMI={bmi},{bmi_judge}.')
```

Step2: Write Code on Client1.py

```
import requests
```

```
url = 'http://127.0.0.1:5000/submit'
payload = {'name': 'wiki',
           'password': '123456',
           'birthday': '1990-04-24',
           'gender': 'male',
           'height': 1.75,
           'weight': 60}
```

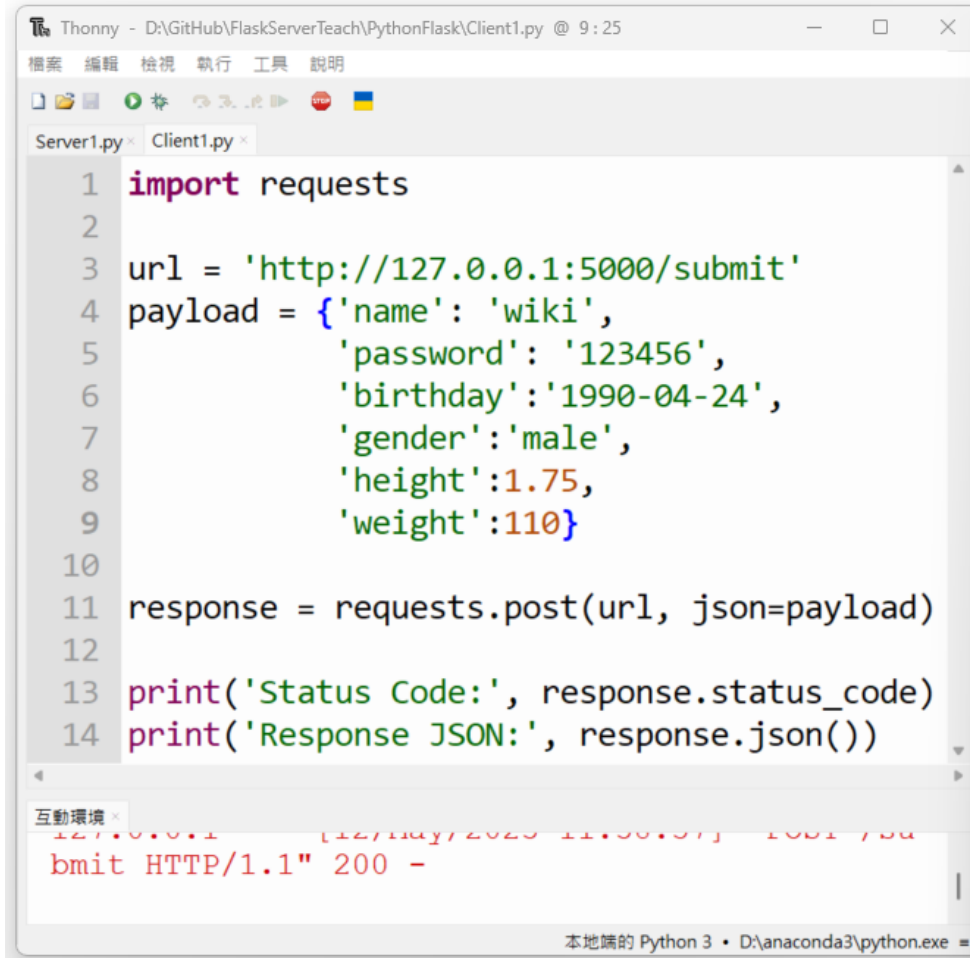
```
response = requests.post(url, json=payload)
```

```
print('Status Code:', response.status_code)
print('Response JSON:', response.json())
```

8. Write Code: Post Method Test

Step3 Run Server1.py and Client1.py

Server1.py Run on IDE Shell.



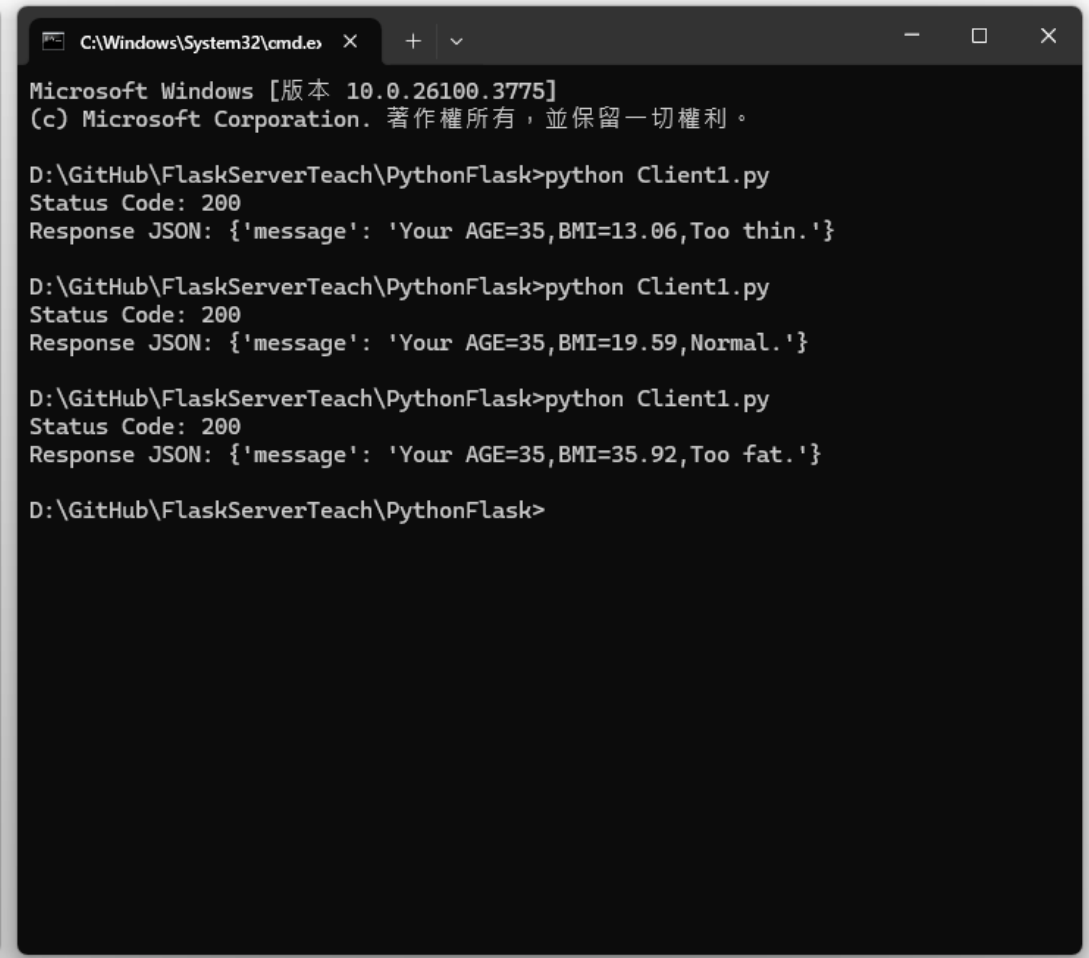
The screenshot shows the Thonny IDE interface. The top window displays the code for Server1.py, which uses the requests library to send a POST request to a local server. The code is as follows:

```
1 import requests
2
3 url = 'http://127.0.0.1:5000/submit'
4 payload = {'name': 'wiki',
5            'password': '123456',
6            'birthday': '1990-04-24',
7            'gender': 'male',
8            'height': 1.75,
9            'weight': 110}
10
11 response = requests.post(url, json=payload)
12
13 print('Status Code:', response.status_code)
14 print('Response JSON:', response.json())
```

The bottom window shows the interactive environment output, which includes the status code and the JSON response from the server:

```
127.0.0.1 [127.0.0.1:5000] - POST /submit HTTP/1.1" 200 -
```

Client1.py Run on Terminal.



The screenshot shows a Windows command prompt window. The user has run the Client1.py script three times, each time with different weight values. The output shows the status code (200) and the JSON response from the server:

```
D:\GitHub\FlaskServerTeach\PythonFlask>python Client1.py
Status Code: 200
Response JSON: {'message': 'Your AGE=35,BMI=13.06,Too thin.'}

D:\GitHub\FlaskServerTeach\PythonFlask>python Client1.py
Status Code: 200
Response JSON: {'message': 'Your AGE=35,BMI=19.59,Normal.'}

D:\GitHub\FlaskServerTeach\PythonFlask>python Client1.py
Status Code: 200
Response JSON: {'message': 'Your AGE=35,BMI=35.92,Too fat.'}

D:\GitHub\FlaskServerTeach\PythonFlask>
```

Client1.py is run on terminal, because Thonny IDE can't run two code at same time.
If Client1.py can't run, you may need to change the Python environment.

9. Write Code: Post Method Test

Step1: Write Code on Server1.py

```
from flask import Flask, request, jsonify
from datetime import datetime
import math
import pandas as pd
import os

app = Flask(__name__)

# POST Method
@app.route('/submit', methods=['POST'])
def submit_form():
    data = request.get_json()
    name = data.get('name')
    password = data.get('password')
    birthday = datetime.strptime((f"{data.get('birthday')}" ), "%Y-%m-%d")
    gender = data.get('gender')
    height = data.get('height')
    weight = data.get('weight')
    bmi = round(weight / height **2,2)# bmi count
    today = datetime.today()
    age = today.year - birthday.year - ((today.month, today.day) < (birthday.month,
birthday.day))
    df = pd.read_csv('BMI_Normal.csv')
    matched_rows = df[df['age'] == age].iloc[0]
    #print(matched_rows)
    bmi_judge = 'wrong data'
    if gender=='male':
        if bmi<matched_rows['male_min']:
            bmi_judge='Too thin'
        elif bmi>=matched_rows['male_min'] and bmi<=matched_rows['male_max']:
            bmi_judge='Normal'
        else:
            bmi_judge='Too fat'
```

```
    elif gender=='female':
        if bmi<matched_rows['female_min']:
            bmi_judge='Too thin'
        elif bmi>=matched_rows['female_min'] and bmi<=matched_rows['female_max']:
            bmi_judge='Normal'
        else:
            bmi_judge='Too fat'
    filename = f"UserData\{name}.csv"
    if os.path.exists(filename): # file exists
        load_df = pd.read_csv(filename)# file load
        stored_password = str(load_df.iloc[0]['password'])# password load
        if stored_password != password:# password incorrect
            return jsonify(message="password error!"), 403
        else:# password correct
            user_data = { #Save Data
                'height': height,
                'weight': weight,
                'bmi': bmi,
                'bmi_judge':bmi_judge,
                'build_time':today
            }
            load_df = pd.concat([load_df, pd.DataFrame([user_data])], ignore_index=True)
            load_df.to_csv(filename, index=False)
        else:# no files
            user_data = {#Save Data
                'password': password,
                'birthday': birthday,
                'gender': gender,
                'height': height,
                'weight': weight,
                'bmi': bmi,
                'bmi_judge':bmi_judge,
                'build_time':today
            }
            save_df = pd.DataFrame([user_data])
            save_filename = f"UserData\{name}.csv".replace("/", "_")
            save_df.to_csv(save_filename, index=False)
            return jsonify(message=f'Your AGE={age},BMI={bmi},{bmi_judge}.')
```

9. Write Code: Post Method Test

Step2: Run Server1.py and Client1.py

The screenshot illustrates the process of testing a POST method using a Python client. It consists of four main components:

- File Explorer:** Shows four CSV files: wiki3.csv, wiki2.csv, wiki1.csv, and wiki.csv. A red dashed arrow points from the wiki.csv file to the Excel spreadsheet.
- Excel Spreadsheet:** The 'wiki.csv' file is open in Excel. The data table is as follows:

| | A | B | C | D | E | F | G | H |
|---|----------|-----------|--------|--------|--------|-------|-----------|------------|
| 1 | password | birthday | gender | height | weight | bmi | bmi_judge | build_time |
| 2 | abc123 | 1990/4/24 | male | 1.75 | 110 | 35.92 | Too fat | 57:29.0 |
| 3 | | | | 1.8 | 90 | 27.78 | Too fat | 57:57.1 |
| 4 | | | | 1.82 | 88 | 26.57 | Normal | 58:20.6 |
| 5 | | | | 1.77 | 40 | 12.77 | Too thin | 58:59.3 |
| 6 | | | | | | | | |
| 7 | | | | | | | | |
| 8 | | | | | | | | |
- Thonny IDE:** The 'Client1.py' file is open, showing the following Python code:

```
1 import requests
2 url = 'http://127.0.0.1:5000/submit'
3 payload = {'name': 'wiki',
4            'password': 'abc123',
5            'birthday': '1990-04-24',
6            'gender': 'male',
7            'height': 1.77,
8            'weight': 40}
9 response = requests.post(url, json=payload)
10 print('Status Code:', response.status_code)
11 print('Response JSON:', response.json())
```
- Windows Command Prompt:** The script is executed from the directory 'D:\GitHub\FlaskServerTeach\PythonFlask'. The output shows four successful POST requests, each returning a status code of 200 and a JSON response with calculated BMI and build status.

```
D:\GitHub\FlaskServerTeach\PythonFlask>python Client1.py
Status Code: 200
Response JSON: {'message': 'Your AGE=35,BMI=35.92,Too fat.'}

D:\GitHub\FlaskServerTeach\PythonFlask>python Client1.py
Status Code: 200
Response JSON: {'message': 'Your AGE=35,BMI=27.78,Too fat.'}

D:\GitHub\FlaskServerTeach\PythonFlask>python Client1.py
Status Code: 200
Response JSON: {'message': 'Your AGE=35,BMI=26.57,Normal.'}

D:\GitHub\FlaskServerTeach\PythonFlask>python Client1.py
Status Code: 200
Response JSON: {'message': 'Your AGE=35,BMI=12.77,Too thin.'}
```

Client1.py is run on terminal, because Thonny IDE can't run two code at same time. If Client1.py can't run, you may need to change the Python environment.

10.Web Server Building for cPanel

Step1: Paid and apply for a cloud server. (My teaching uses bigcloud.com.tw)

The screenshot displays the Big Cloud user interface. At the top left is the Big Cloud logo. A navigation bar at the top right contains links: 首頁, 服務, 網域, 帳務, 技術支援, and 建立服務單. The main content area is divided into several sections:

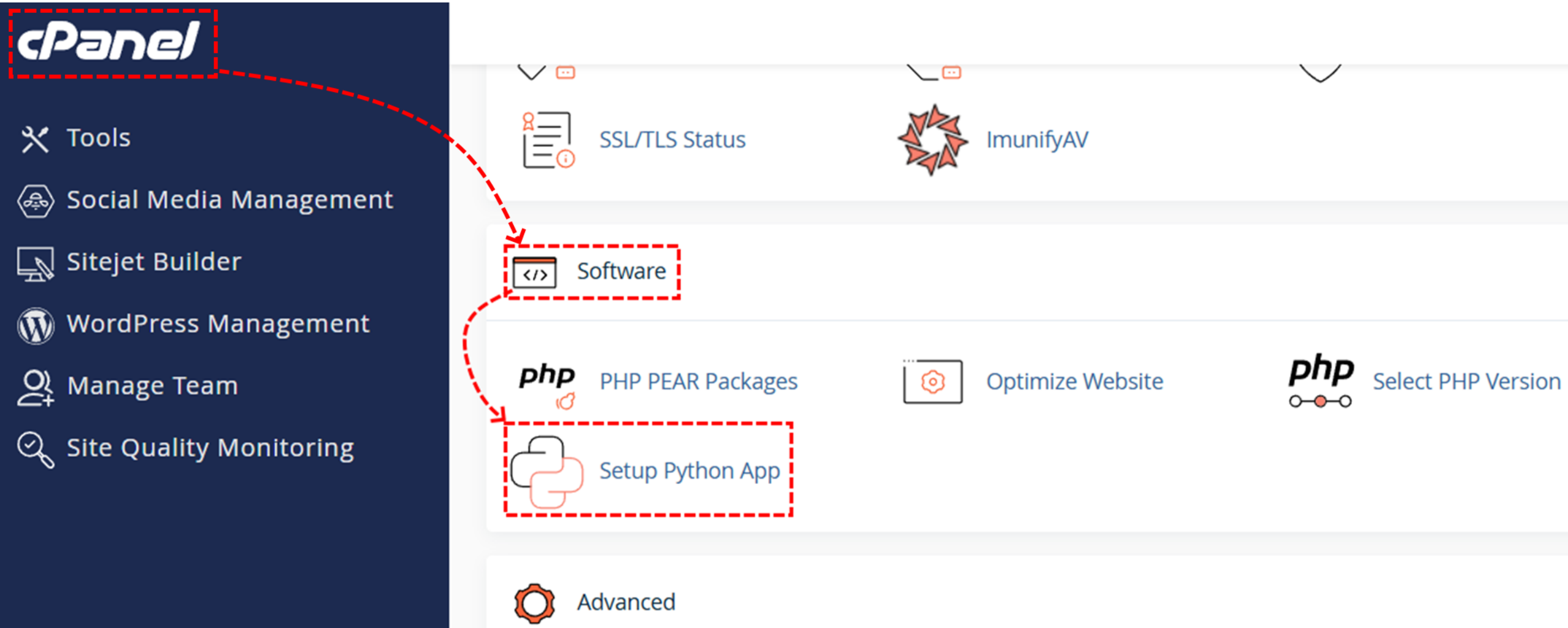
- Left Sidebar (Actions):** Includes a '動作' (Actions) header and a list of links: '登入 cPanel' (highlighted with a red dashed box), 'Login cPanel' (in red), '登入 Webmail', '變更密碼', and '要求撤銷'.
- Package/Domain:** Shows '台灣大無限 台灣大 T' and 'www.wiciar.com'. It features two buttons: '造訪網站' (Visit Website) and '管理網域' (Manage Domain).
- Usage Statistics:** Displays two gauges: 'Disk Usage' at 2 (2373 M / 100000 M) and 'Bandwidth Usage' at 0 (44.45 M / 10000000 M). A note at the bottom states 'Last Updated 2025/05/12 (16:05)'.
- Quick Shortcuts:** A grid of icons for various services: Email Accounts, Forwarders, Autoresponders, File Manager, Backup, Subdomains, Addon Domains, Cron Jobs, MySQL Databases, phpMyAdmin, and Awstats.

Not necessarily Bigcloud, you can use AWS, Azure, firebase or Alibaba Cloud...

But when you want to bid for the R.O.C gov. Project, Data center must for Chunghwa Telecom's Device.

10.Web Server Building for cPanel

Step2: Create python server on cPanel.



10.Web Server Building for cPanel

Step2: Create python server on cPanel.

WEB APPLICATIONS

The screenshot displays the cPanel 'WEB APPLICATIONS' interface. At the top right, a blue button with a plus icon and the text 'CREATE APPLICATION' is highlighted with a red dashed box. Below this is a table listing existing web applications. The table has four columns: 'App URI', 'App Root Directory', 'Status', and 'Actions'. One application is listed with the URI 'wiciar.com/bmi', root directory '/home/wiciarco/PythonBMI', and status 'started (v3.9.21)'. This row and its URI are also highlighted with a red dashed box. A red dashed arrow points from this box to the 'CREATE APPLICATION' form below. The form has a header with 'WEB APPLICATIONS' and a 'CREATE APPLICATION' button. The form fields are: 'Python version' (set to 3.9.21), 'Application root' (set to PythonBMI), 'Application URL' (set to wiciar.com and bmi), 'Application startup file' (set to server1.py), and 'Application Entry point' (set to app). A red dashed box encloses these form fields. Another red dashed arrow points from the bottom of this box back to the 'CREATE' button in the form's header. The 'CREATE' button is also highlighted with a red dashed box.

| App URI | App Root Directory | Status | Actions |
|----------------|--------------------------|-------------------|---------|
| ... | ... | ... | ... |
| wiciar.com/bmi | /home/wiciarco/PythonBMI | started (v3.9.21) | ... |

WEB APPLICATIONS **CREATE APPLICATION** CANCEL **CREATE**

Python version 3.9.21

Application root PythonBMI

Application URL wiciar.com bmi

Application startup file server1.py

Application Entry point app

10.Web Server Building for cPanel

Step3: upload and Create app root of files & folders.

The image shows the cPanel interface with the File Manager tool selected. The left sidebar contains various tools, with 'File Manager' highlighted. The main area displays the 'PythonBMI' directory. A red dashed line indicates the workflow: starting from the cPanel logo, moving to the 'Files' icon, then to the 'File Manager' icon, and finally to the '+ PythonBMI' button. A blue dashed line shows the path from the 'File Manager' icon to the 'Upload' button and then to the 'UserData' folder. A yellow dashed line shows the path from the 'File Manager' icon to the 'PythonBMI' button and then to the 'requirements.txt' file. A red dashed line also shows the path from the 'Upload' button to the 'BMI_Normal.csv' and 'server1.py' files.

cPanel

- Tools
- Social Media Management
- Sitejet Builder
- WordPress Management
- Manage Team
- Site Quality Monitoring

File Manager

Files

File Manager

Disk Usage

Backup Wizard

PythonBMI

Go

Collapse All

(/home/wiciarco)

Home Up One Level

Name

- __pycache__
- public
- tmp
- UserData
- BMI_Normal.csv
- passenger_wsgi.py
- requirements.txt
- server1.py
- stderr.log

+ PythonBMI

10.Web Server Building for cPanel

Step4:update python server environmental from requirements.txt

WEB APPLICATIONS [WICIAR.COM/BMI](#) DESTROY CANCEL SAVE

Setup wsgi callable object for your application

Configuration files

▶ Run Pip Install ▼

requirements.txt + Add

requirements.txt Edit Delete

Execute python script

You can also enter a command to execute (e.g `/path/to/manage.py migrate`). Note, only python scripts allowed.

Edit requirements.txt

Enter the path to the script file

| | |
|---|--------|
| 1 | flask |
| 2 | pandas |

CANCEL SAVE

```
graph TD
    subgraph TopBar [WEB APPLICATIONS WICIAR.COM/BMI]
        DESTROY[DESTROY]
        CANCEL1[CANCEL]
        SAVE1[SAVE]
    end

    subgraph MainContent
        direction TB
        A[▶ Run Pip Install ▼]
        B[requirements.txt + Add]
        C[requirements.txt Edit Delete]
        D[1 flask  
2 pandas]
        E[SAVE]
    end












    A -.-> B
    B -.-> C
    C -.-> D
    D -.-> E
```

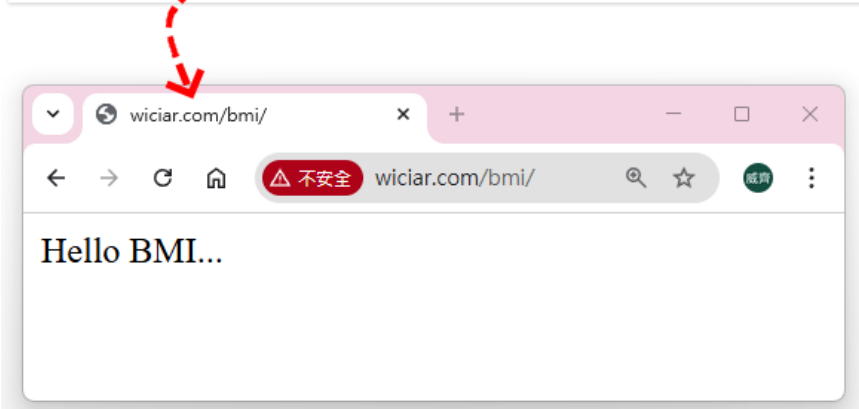
10.Web Server Building for cPanel

Step5:Run Server

WEB APPLICATIONS

[+ CREATE APPLICATION](#)

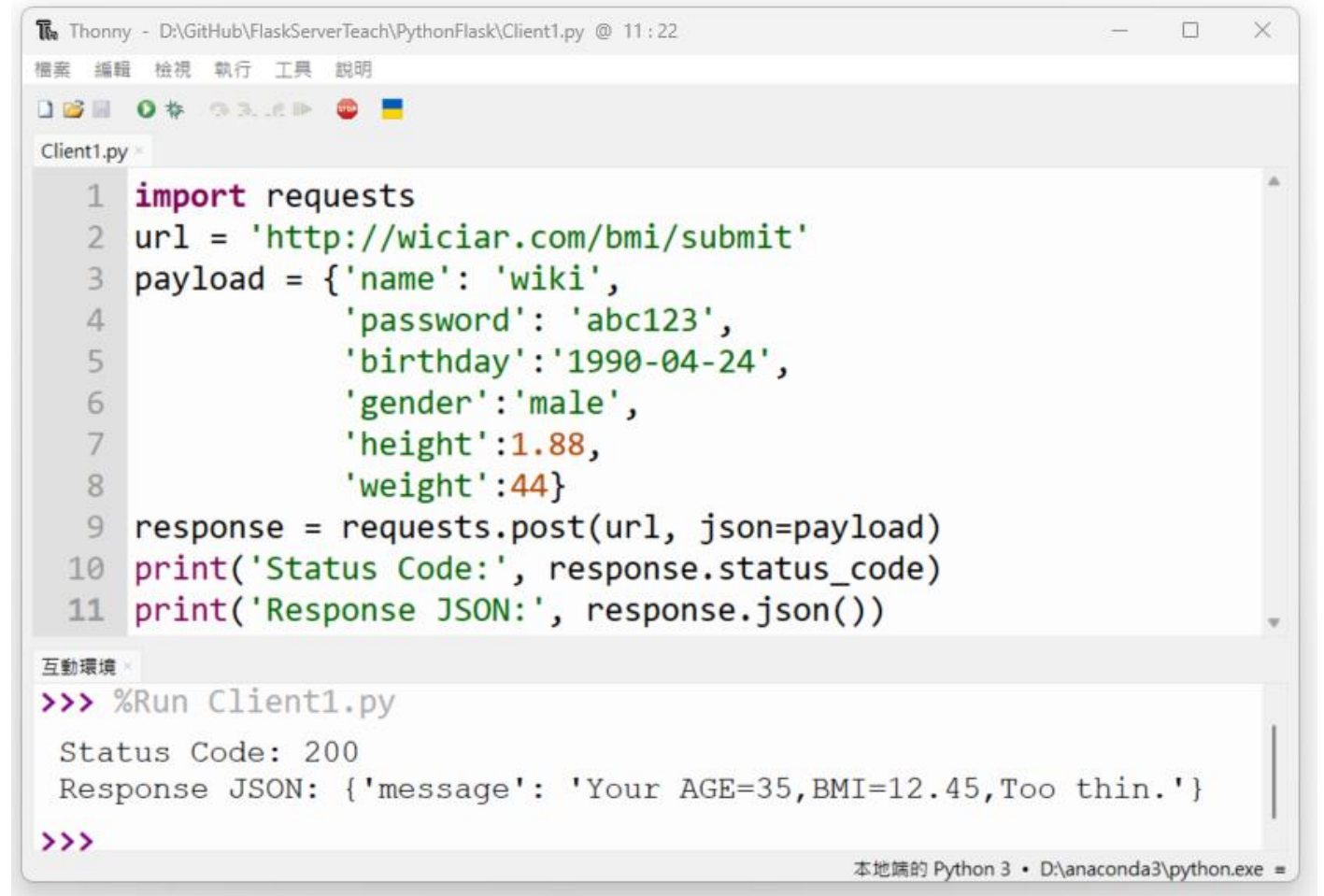
| App URI | App Root Directory | Status | Actions |
|--|---|---------------------|---|
| wiciar.com/iot | /home/wiciarco/PythonApp | ● started (v3.9.21) |     |
| wiciar.com/TimingArrangement | /home/wiciarco/PythonAppTimingArrangement | ● started (v3.9.21) |     |
| wiciar.com/bmi 2. | /home/wiciarco/PythonBMI | ● stopped (v3.9.21) | 1.    |



10.Web Server Building for cPanel

Step6: Client1.py Code & Test Run

```
import requests
url = 'http://wiciar.com/bmi/submit'
payload = {'name': 'wiki',
           'password': 'abc123',
           'birthday': '1990-04-24',
           'gender': 'male',
           'height': 1.88,
           'weight': 44}
response = requests.post(url, json=payload)
print('Status Code:', response.status_code)
print('Response JSON:', response.json())
```



The screenshot shows the Thonny IDE interface. The top pane displays the code for Client1.py, which imports the requests library, defines a URL and a JSON payload, and sends a POST request. The bottom pane shows the interactive environment where the script was run, displaying the status code 200 and the JSON response from the server.

```
Thonny - D:\GitHub\FlaskServerTeach\PythonFlask\Client1.py @ 11:22
檔案 編輯 檢視 執行 工具 說明

Client1.py
1 import requests
2 url = 'http://wiciar.com/bmi/submit'
3 payload = {'name': 'wiki',
4           'password': 'abc123',
5           'birthday': '1990-04-24',
6           'gender': 'male',
7           'height': 1.88,
8           'weight': 44}
9 response = requests.post(url, json=payload)
10 print('Status Code:', response.status_code)
11 print('Response JSON:', response.json())

互動環境
>>> %Run Client1.py
Status Code: 200
Response JSON: {'message': 'Your AGE=35,BMI=12.45,Too thin.'}
>>>

本地端的 Python 3 • D:\anaconda3\python.exe
```

10.Web Server Building for cPanel

Server1.py Code

```
from flask import Flask, request, jsonify
from datetime import datetime
import math
import pandas as pd
import os

app = Flask(__name__)

# GET Method
@app.route('/')
def hello():
    return 'Hello BMI...'

# POST Method
@app.route('/submit', methods=['POST'])
def submit_form():
    data = request.get_json()
    name = data.get('name')
    password = data.get('password')
    birthday = datetime.strptime((f"{data.get('birthday')}" ), "%Y-%m-%d")
    gender = data.get('gender')
    height = data.get('height')
    weight = data.get('weight')
    bmi = round(weight / height **2,2)# bmi count
    today = datetime.today()
    age = today.year - birthday.year - ((today.month, today.day) < (birthday.month, birthday.day))
    df = pd.read_csv('BMI_Normal.csv')
    matched_rows = df[df['age'] == age].iloc[0]
    #print(matched_rows)
    bmi_judge = 'wrong data'
    if gender=='male':
        if bmi<matched_rows['male_min']:
            bmi_judge='Too thin'
        elif bmi>=matched_rows['male_min'] and bmi<=matched_rows['male_max']:
            bmi_judge='Normal'
        else:
            bmi_judge='Too fat'
```

```
elif gender=='female':
    if bmi<matched_rows['female_min']:
        bmi_judge='Too thin'
    elif bmi>=matched_rows['female_min'] and bmi<=matched_rows['female_max']:
        bmi_judge='Normal'
    else:
        bmi_judge='Too fat'
filename = f"UserData/{name}.csv"
if os.path.exists(filename): # file exists
    load_df = pd.read_csv(filename)# file load
    stored_password = str(load_df.iloc[0]['password'])# password load
    if stored_password != password:# password incorrect
        return jsonify(message="password error!"), 403
    else:# password correct
        user_data = { #Save Data
            'height': height,
            'weight': weight,
            'bmi': bmi,
            'bmi_judge':bmi_judge,
            'build_time':today
        }
        load_df = pd.concat([load_df, pd.DataFrame([user_data])], ignore_index=True)
        load_df.to_csv(filename, index=False)
else:# no files
    user_data = {#Save Data
        'password': password,
        'birthday': birthday,
        'gender': gender,
        'height': height,
        'weight': weight,
        'bmi': bmi,
        'bmi_judge':bmi_judge,
        'build_time':today
    }
    save_df = pd.DataFrame([user_data])
    #save_filename = f"UserData\{name}.csv".replace("/", "_")
    save_df.to_csv(filename, index=False)
    return jsonify(message=f'Your AGE={age},BMI={bmi},{bmi_judge}')
```


11.Client UI Design for Python

```
import tkinter as tk
import requests
```

Step1: Write Code--Client_UI1.py

```
root = tk.Tk()
root.title('BMI Input')
root.geometry('720x720')
```

```
gender = tk.StringVar()
name = tk.StringVar()
password = tk.StringVar()
birthday = tk.StringVar()
height = tk.StringVar()
weight = tk.StringVar()
jsonText = tk.StringVar()
serverText = tk.StringVar()
```

```
radio_btn1 = tk.Radiobutton(root,text='Male',font=('Arial',30,'bold'),variable=gender,value='male')
radio_btn2 = tk.Radiobutton(root,text='Female',font=('Arial',30,'bold'),variable=gender,value='female')
radio_btn1.select()
```

```
labelName = tk.Label(root, text='Name:')
entryName = tk.Entry(root,font=('Arial',30,'bold'),textvariable=name)
```

```
labelPassword = tk.Label(root, text='Password:')
entryPassword = tk.Entry(root,show='*',font=('Arial',30,'bold'),textvariable=password)
```

```
labelHeight = tk.Label(root, text='Height(m):')
entryHeight = tk.Entry(root,font=('Arial',30,'bold'),textvariable=height)
```

```
labelWeight = tk.Label(root, text='Weight(kg):')
entryWeight = tk.Entry(root,font=('Arial',30,'bold'),textvariable=weight)
```

```
labelBirthday = tk.Label(root, text='Birthday(UTC):')
entryBirthday = tk.Entry(root,font=('Arial',30,'bold'),textvariable=birthday)
```

```
Jsonlabel = tk.Label(root,textvariable=jsonText,font=('Arial',20,'bold'),fg='#f00')
```

```
Serverlabel = tk.Label(root,textvariable=serverText,font=('Arial',20,'bold'),fg='#00f')
```

```
def submit():
    global gender,name,password,birthday,height,weight,jsonText
    temp
    =f'gender={gender.get()}\nname={name.get()}\npassword={password.get()}\nbi
rthday={birthday.get()}\nheight={height.get()}\nweight={weight.get()}'
    jsonText.set(temp)
    url = 'http://wiciar.com/bmi/submit'
    jsonSet = {'name': name.get(),
               'password': password.get(),
               'birthday':birthday.get(),
               'gender':gender.get(),
               'height':float(height.get()),
               'weight':float(weight.get())}
    #print(jsonSet)
    try:
        response = requests.post(url, json=jsonSet)
        serverText.set(response.json()['message'])
        print('Response JSON:', response.json())
    except:
        serverText.set('Server Link error!!!')
        print('Status Code:', response.status_code)
```

```
submit_btn1 =
tk.Button(root,command=submit,text='PassData',font=('Arial',30,'bold'),padx=1
0,pady=10,activeforeground='#f00')
```

```
radio_btn1.grid(column=0, row=0)
radio_btn2.grid(column=1, row=0)
labelName.grid(column=0, row=1)
entryName.grid(column=1, row=1)
labelPassword.grid(column=0, row=2)
entryPassword.grid(column=1, row=2)
labelHeight.grid(column=0, row=3)
entryHeight.grid(column=1, row=3)
labelWeight.grid(column=0, row=4)
entryWeight.grid(column=1, row=4)
labelBirthday.grid(column=0, row=5)
entryBirthday.grid(column=1, row=5)
submit_btn1.grid(column=1, row=6)
Jsonlabel.grid(column=1, row=7)
Serverlabel.grid(column=1, row=8)
root.mainloop()
```

11.Client UI Design for Python

Step2: Run Client_UI1.py

The image shows a Python GUI application for BMI calculation and its corresponding code. On the left is the application window titled "BMI Input". It features two radio buttons for "Male" and "Female", with "Female" selected. Below these are input fields for Name (wiki2), Password (*****), Height(m) (1.96), Weight(kg) (47.1), and Birthday(UTC) (1999-9-9). A "PassData" button is at the bottom. Below the button, the collected data is displayed in red text: gender=female, name=wiki2, password=abc123, birthday=1999-9-9, height=1.96, weight=47.1. At the bottom, a blue message states "Your AGE=25,BMI=12.26,Too thin." On the right is the Thonny IDE showing the Python code for Client_UI1.py. The code uses Tkinter for the GUI and includes logic for data collection and calculation. A red dashed arrow points from the "PassData" button in the UI to the submit_btn1 widget in the code. The bottom of the IDE shows the interactive environment with the command "%Run Client_UI1.py" and the resulting JSON response: {"message": "Your AGE=25,BMI=12.26,Too thin."}. A blue dashed arrow points from this response back to the application window.

BMI Input

☐ Male ☒ Female

Name: wiki2

Password: *****

Height(m): 1.96

Weight(kg): 47.1

Birthday(UTC): 1999-9-9

PassData

gender=female
name=wiki2
password=abc123
birthday=1999-9-9
height=1.96
weight=47.1

Your AGE=25,BMI=12.26,Too thin.

```
Thonny - D:\GitHub\FlaskServerTeach\PythonFlask\Client_UI1.py @ 72 : 36
檔案 編輯 檢視 執行 工具 說明
Client_UI1.py
60 submit_btn1 = tk.Button(root,command=submit)
61
62 radio_btn1.grid(column=0, row=0)
63 radio_btn2.grid(column=1, row=0)
64 labelName.grid(column=0, row=1)
65 entryName.grid(column=1, row=1)
66 labelPassword.grid(column=0, row=2)
67 entryPassword.grid(column=1, row=2)
68 labelHeight.grid(column=0, row=3)
69 entryHeight.grid(column=1, row=3)
70 labelWeight.grid(column=0, row=4)
71 entryWeight.grid(column=1, row=4)
72 labelBirthday.grid(column=0, row=5)
73 entryBirthday.grid(column=1, row=5)
74 submit_btn1.grid(column=1, row=6)
75 Jsonlabel.grid(column=1, row=7)
76 Serverlabel.grid(column=1, row=8)
77 # 顯示按鈕
78 root.mainloop() # 放在主迴圈中

互動環境
>>> %Run Client_UI1.py
Response JSON: {'message': 'Your AGE=25,BMI=12.26,Too thin.'}
```

本地化的 Python 3 • D:\anaconda3\python.exe

12.Client get User historical information

Step1: Write Code--Client2.py

```
import requests
import pandas as pd
import matplotlib.pyplot as plt

url = 'http://wiciar.com/bmi/data_get'
payload = {'name': 'wiki2',
           'password': 'abc123'}
print('Json Code:', payload)
response = requests.post(url, json=payload)
print('Status Code:', response.status_code)
#print('Response JSON:', response.json()['message'])
#Data visualization output
data_list = response.json()['message']
df = pd.DataFrame(data_list)
df['build_time'] = pd.to_datetime(df['build_time'])
df = df.dropna(subset=['bmi', 'build_time'])
df = df.sort_values(by='build_time')
plt.figure(figsize=(10, 5))
plt.plot(df['build_time'], df['bmi'], marker='o', linestyle='-', color='blue', label='BMI')
plt.title('BMI Trend Over Time')
plt.xlabel('Build Time')
plt.ylabel('BMI')
plt.grid(True)
plt.xticks(rotation=45)
plt.tight_layout()
plt.legend()
plt.show()
```

Server1.py Code Design

```
#omit...
# GET Method
@app.route('/')
def hello():
    return 'Hello BMI...'

# POST Method
@app.route('/data_get', methods=['POST'])
def UserDataGet():
    data = request.get_json()
    name = data.get('name')
    password = data.get('password')

    if not name or not password:
        return jsonify(message="get me 'name' & 'password'. cant be empty"), 400

    filename = f"UserData/{name}.csv"

    if os.path.exists(filename):
        df = pd.read_csv(filename)
        if (df["password"] == password).any():
            df = pd.concat([df, pd.DataFrame()], ignore_index=True)
        else:
            return jsonify(message="Password error!"), 403

        df = df.drop(columns=["password"], errors="ignore")
        print(df)
        return jsonify(message=df.to_dict(orient="records"))

    else:
        return jsonify(message="No User Data!"), 403

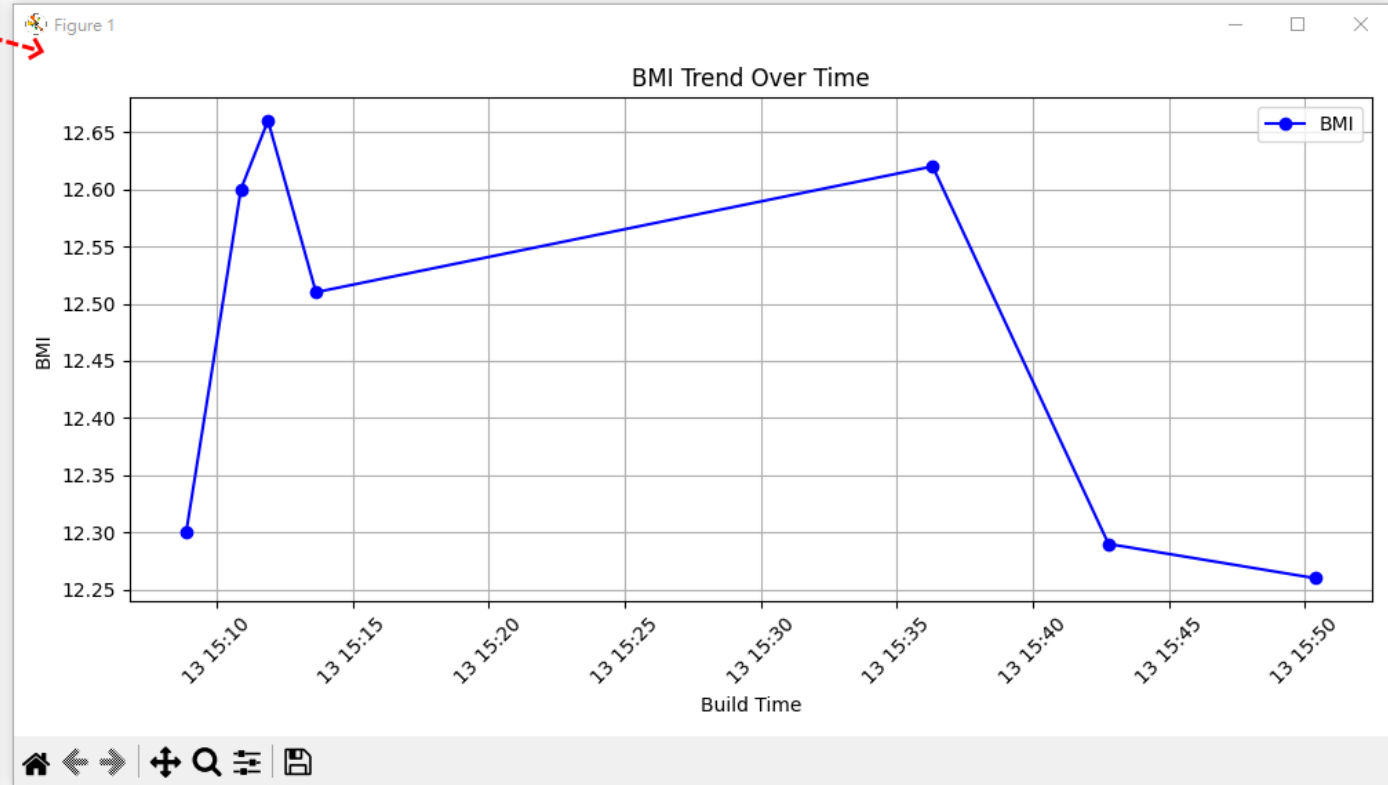
# POST Method
@app.route('/submit', methods=['POST'])
def submit_form():
    #omit...
```

#omit... this can Refer to Section 10

12.Client get User historical information

Step2: Run Code--Client2.py

```
Thonny - I:\GithubProject\FlaskServerTeach\PythonFlask\Client2.py @ 21:1
File Edit View Run Tools Help
Client2.py x
1 import requests
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 url = 'http://wiciar.com/bmi/data_get'
6 payload = {'name': 'wiki2',
7            'password': 'abc123'}
8 print('Json Code:', payload)
9 response = requests.post(url, json=payload)
10 print('Status Code:', response.status_code)
11 #print('Response JSON:', response.json()['message'])
12 data_list = response.json()['message']
13 df = pd.DataFrame(data_list)
14 df['build_time'] = pd.to_datetime(df['build_time'])
15 df = df.dropna(subset=['bmi', 'build_time'])
16 df = df.sort_values(by='build_time')
17 plt.figure(figsize=(10, 5))
18 plt.plot(df['build_time'], df['bmi'], marker='o', linestyle=
19 plt.title('BMI Trend Over Time')
20 plt.xlabel('Build Time')
21 plt.ylabel('BMI')
22 plt.grid(True)
23 plt.xticks(rotation=45)
24 plt.tight_layout()
25 plt.legend()
26 plt.show()
```



13.Client get User historical information

Step1: Client2.py Redesign

```
Thonny - I:\GithubProject\FlaskServerTeach\PythonFlask\Client2.py @ 5 : 1
File Edit View Run Tools Help

Client2.py x
1 import requests
2 import pandas as pd
3 import matplotlib.pyplot as plt
4
5 class UserHistoricalInformation:
6     def BMIshow():
7         url = 'http://wiciar.com/bmi/data_get'
8         payload = {'name': 'wiki2',
9                   'password': 'abc123'}
10        print('Json Code:', payload)
11        response = requests.post(url, json=payload)
12        print('Status Code:', response.status_code)
13        #print('Response JSON:', response.json()['message'])
14        data_list = response.json()['message']
15        df = pd.DataFrame(data_list)
16        df['build_time'] = pd.to_datetime(df['build_time'])
17        df = df.dropna(subset=['bmi', 'build_time'])
18        df = df.sort_values(by='build_time')
19        plt.figure(figsize=(10, 5))
20        plt.plot(df['build_time'], df['bmi'], marker='o', linestyle='-', color='blue', label='BMI')
21        plt.title('BMI Trend Over Time')
22        plt.xlabel('Build Time')
23        plt.ylabel('BMI')
24        plt.grid(True)
25        plt.xticks(rotation=45)
26        plt.tight_layout()
27        plt.legend()
28        plt.show()
29
```

```
import requests
import pandas as pd
import matplotlib.pyplot as plt

class UserHistoricalInformation:
    def BMIshow():
        url = 'http://wiciar.com/bmi/data_get'
        payload = {'name': 'wiki2',
                  'password': 'abc123'}
        print('Json Code:', payload)
        response = requests.post(url, json=payload)
        print('Status Code:', response.status_code)
        #print('Response JSON:', response.json()['message'])
        data_list = response.json()['message']
        df = pd.DataFrame(data_list)
        df['build_time'] = pd.to_datetime(df['build_time'])
        df = df.dropna(subset=['bmi', 'build_time'])
        df = df.sort_values(by='build_time')
        plt.figure(figsize=(10, 5))
        plt.plot(df['build_time'], df['bmi'], marker='o', linestyle='-', color='blue', label='BMI')
        plt.title('BMI Trend Over Time')
        plt.xlabel('Build Time')
        plt.ylabel('BMI')
        plt.grid(True)
        plt.xticks(rotation=45)
        plt.tight_layout()
        plt.legend()
        plt.show()
```

13.Client get User historical information

Step2: Client_UI1.py Redesign

```
#omit...
submit_btn1 =
tk.Button(root,command=submit,text='PassData',font=('Arial',30,'bold'),padx=10,pady=10,activeforeground='#f00')

from Client2 import UserHistoricalInformation as UHI
def submit_UHI():
    UHI.BMIshow()

submit_btn2 =
tk.Button(root,command=submit_UHI,text='UserBmiHistory',font=('Arial',30,'bold'),padx=10,pady=10,activeforeground='#f00')
submit_btn2.grid(column=1, row=6)

radio_btn1.grid(column=0, row=0)
radio_btn2.grid(column=1, row=0)
#omit...
```

#omit... this can Refer to Section 11

```
Client_UI1.py x
58     print('Status Code:', response.status_code)
59
60 submit_btn1 = tk.Button(root,command=submit,text='PassData')
61
62 from Client2 import UserHistoricalInformation as UHI
63 def submit_UHI():
64     UHI.BMIshow()
65
66 submit_btn2 = tk.Button(root,command=submit_UHI,text='UserBmiHistory')
67 submit_btn2.grid(column=1, row=6)
68
69 radio_btn1.grid(column=0, row=0)
70 radio_btn2.grid(column=1, row=0)
71 labelName.grid(column=0, row=1)
```

Step3: Run Client_UI1.py

