FLASK

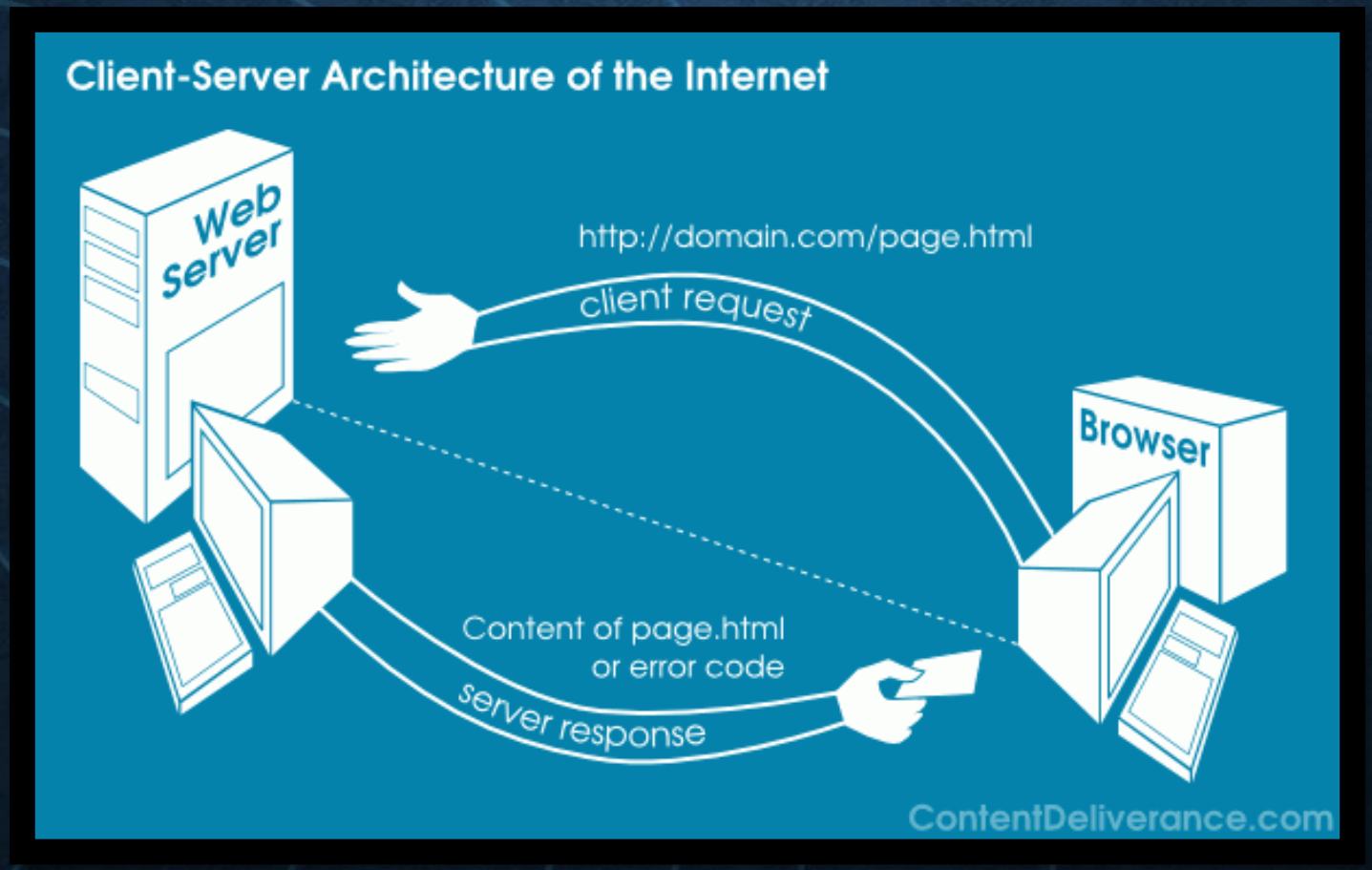
Lightweight WSGI web application framework

INTRODUCTION

INTRODUCTION

- Flask 是一個使用 Python 撰寫的輕量級 網頁應用程式框架，也稱為 micro-framework（微框架）
- 和 Django 不同之處在於，Flask 給予開發者非常大的彈性，可以選用不同的用的 extension 來增加其功能，但也意味著在設計上，需要更多巧思!
- 廣泛的被應用於小型的專案開發，與大型的網頁建設。目前採用 Flask 架設網站的企業:
    1) Netflix (https://www.netflix.com/tw/)
    2) Uber (https://www.uber.com/tw/zh-tw/)
    3) MIT (https://web.mit.edu/)

REQUEST-RESPONSE BASED CLIENT-SERVER PARADIGM

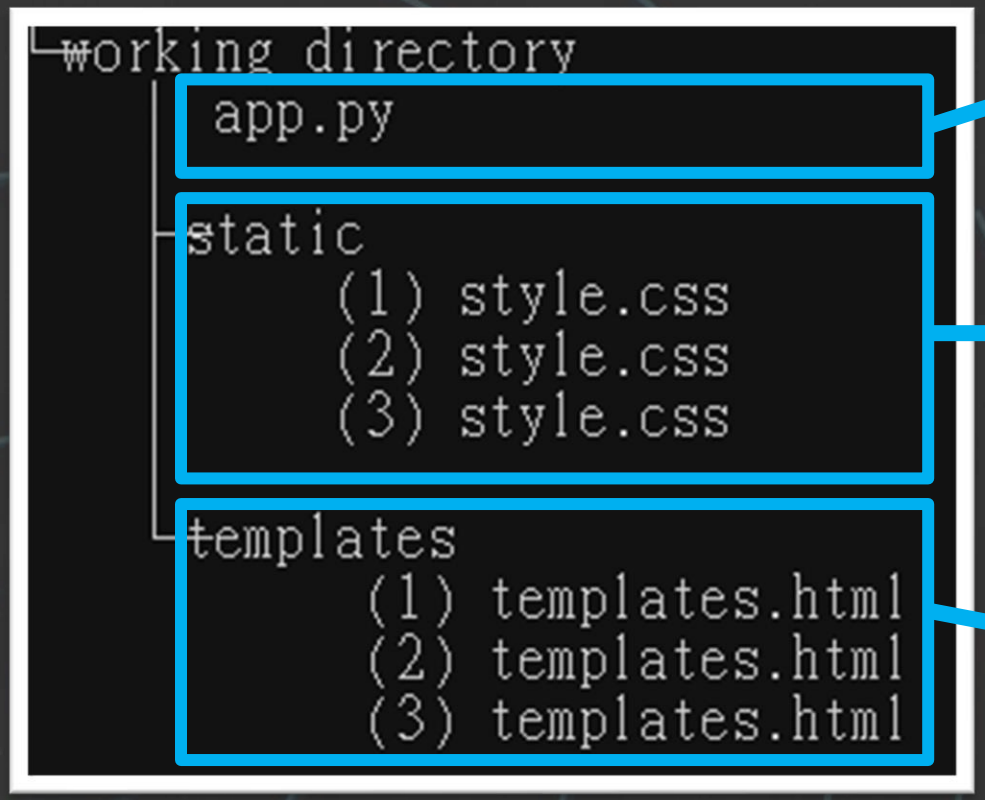


HTTP METHODS

- HTTP是一個客戶端（用戶）和伺服器端（網站）之間請求和應答的標準

Method	Usage
GET	ideal for retrieving data
POST	ideal for submitting data, resulting in modification of state
PUT	for replacing server's target resource with request resource
DELETE	for deleting a server resource

STRUCTURE OF YOUR WORKING DIRECTORY



主程式檔：

控制 Request 的處理與 Response 的發送

靜態檔：

為 HTML files 進行補充，如背景顏色、字體大小、又或是影像等，可為 CSS files, JS files, font files ……

模板檔：

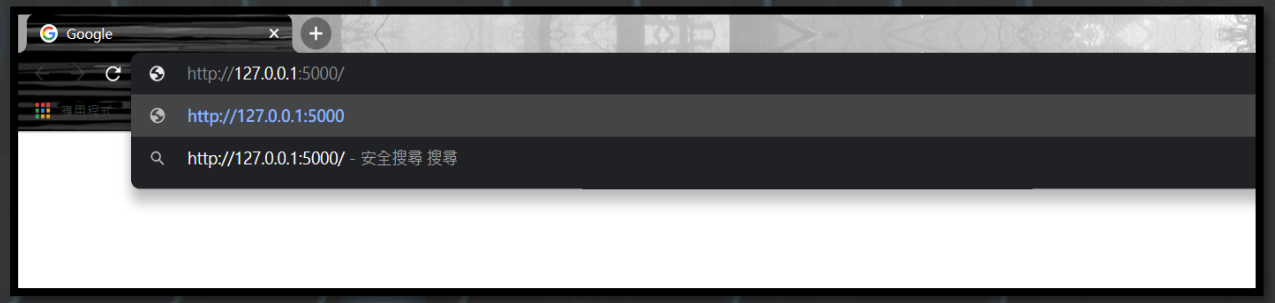
儲存不同URL底下，主程式檔須回傳的網頁訊息(即Response)，通常為HTML files

HOW TO STARTS THE DEVELOPMENT SERVER

- Step 1 : 在CMD (命令提示自元) 中執行主程式檔 EX : python XXXX.py
- Step 2 : 確認 Server 運作正常

```
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with windowsapi reloader
* Debugger is active!
* Debugger PIN: 373-652-876
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

- Step 3 : 複製連結至瀏覽器中



GET DOWN TO BUSINESS

FLASK - INSTALLATION

```
pip install flask
```

```
import flask
flask.__version__ # 最新版本 1.1.2
```

FLASK - BASICS

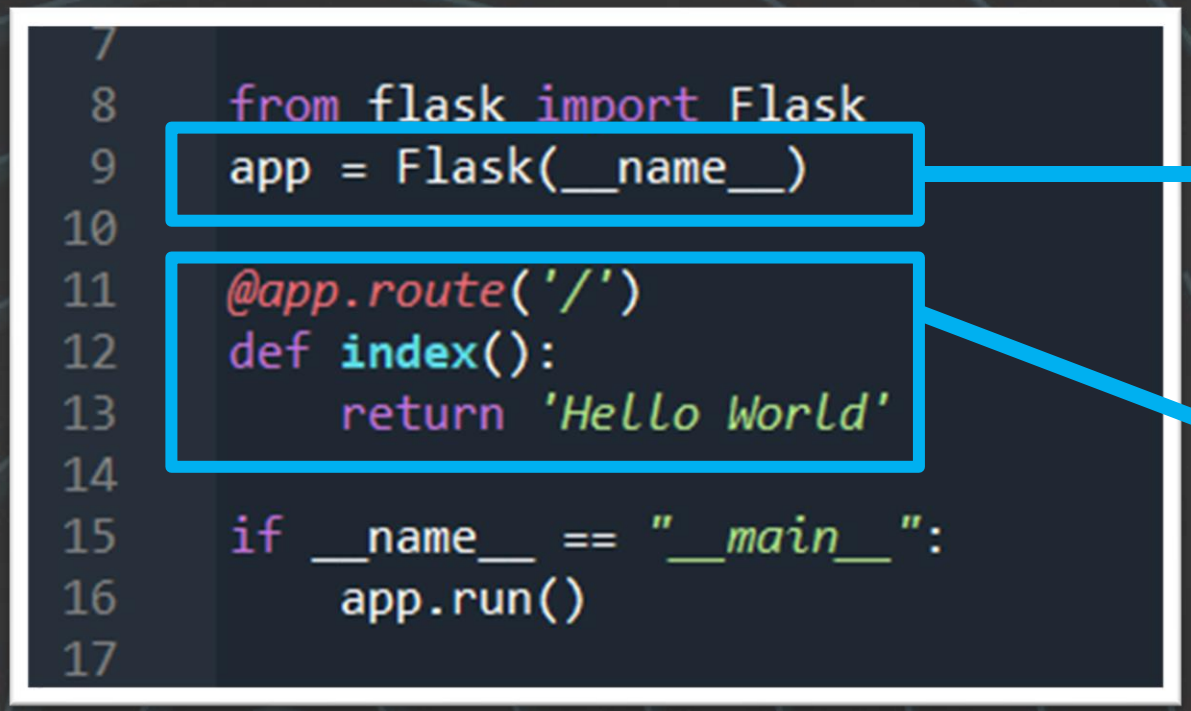
```python
from flask import Flask
app = Flask(__name__)

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

if __name__ == "__main__":
    app.run()
```

◎ 對應範例檔：hello_world.py

FLASK - BASICS



```
7
8   from flask import Flask
9   app = Flask(__name__)
10
11  @app.route('/')
12  def index():
13      return 'Hello World'
14
15  if __name__ == "__main__":
16      app.run()
17
```

◎ Instantiating Flask Class :

實際上為WSGI (Web Server Gateway Interface) ，Server 端會將其所接收的所有 Request，交與其進行後續的處理。以 __name__作為 argument。

◎ Creating Routes :

利用 Flask 提供的 Route Decorator (@app.route)，將URL ('/') 與 view function (index) 連結；其中 view function 為專門處理 Request 的函數。

每當 Server 接收到來自 root ('/') 的 Request 時， view function (index) 將被啟動，完成任務。

◎ 對應範例檔：hello_world.py

FLASK – BASICS (CREATING ROUTES)

```
 9    @app.route('/')
10    def index():
11        return 'Home Page'
12
13    @app.route('/career/')
14    def career():
15        return 'Career Page'
16
17    @app.route('/feedback/')
18    def feedback():
19        return 'Feedback Page'
```

```
23    @app.route('/contact/')
24    @app.route('/feedback/')
25    def feedback():
26        return 'Feedback Page'
```

◎ 對應範例檔：hello_world.py

FLASK – BASICS (CREATING ROUTES)

```
29    @app.route('/user/<id>/')
30    def user_profile(id):
31        return "Profile page of user #{}".format(id)
```

- Dynamic URLs :
  - ➔ URL 中包含某些 view function 會使用到的變數
  - ➔ 變數的部分以 <variable_name> 做表示，將帶入 view function 中

◎ 對應範例檔 : hello_world.py

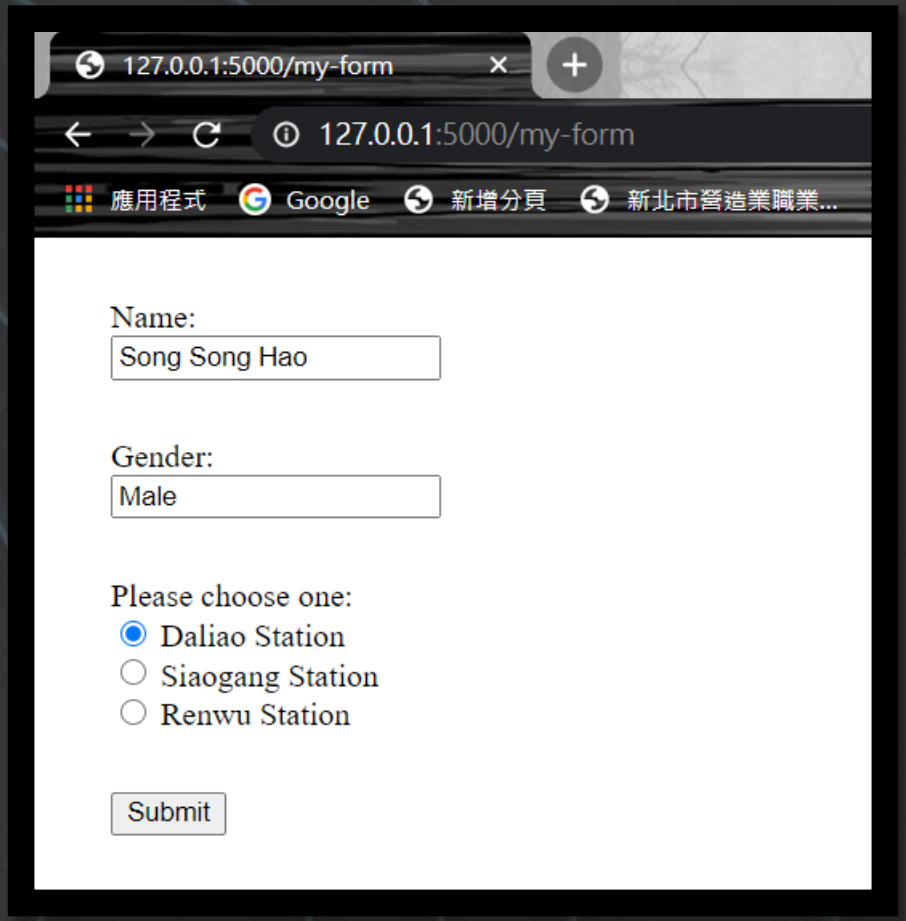
FLASK - REQUEST

- 我們可藉由 **Flask** 提供的 request 物件，獲取從 **client** 端發送的請求中，所包含的訊息
- **Request** 常用的屬性：

Attributes	Return
request.form	key/value pairs of data sent through POST
request.method	HTTP method used by the request
request.args	key/value pairs of data from the URL query string (through GET)
request.values	generic key/value pair extraction (for both GET, POST)
request.json	to obtain parsed JSON content
request.files	to obtain the sent files

◎ 對應範例檔：request_urlfor.py

FLASK – REQUEST.FORM



request.method == POST

◎ 對應範例檔：request_urlfor.py

FLASK – REQUEST.FORM

```
ImmutableMultiDict([('name', 'Song Song Hao'), ('gender', 'Male'), ('Station', 'Daliao Station')])
```

Python 實際接收到的資料為 類似 **Dictionary** 的物件

物件的操作上，與 **Dictionary** 相同！

```python
# handling form data
@app.route('/form-handler', methods=['POST'])
def handle_data():
    print(""*20)
    print(request.form)
    print(""*20)
    print(request.method)

    name = request.form['name']
    gender = request.form['gender']
    station = request.form['Station']

    return render_template(station+'.html')
```

◎ 對應範例檔：request_urlfor.py

FLASK - TEMPLATES

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

    <p>Name: {{ name }}</p>

</body>
</html>
```

◎ Templates :

包含靜態的 HTML 程式碼，以及動態可做調整的部分 ，將在 Server 送出 Response 前填上。

◎ Template rendering :

填補可調整部分，並產生可供呈現的 HTML 網頁，需要額外的 Package 來達成 !

Jinja template engine

FLASK - TEMPLATES

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Title</title>
</head>
<body>

    <p>Name: {{ name }}</p>

</body>
</html>
```

將檔名取為 template.html

◎ 對應範例檔：template.py

FLASK - TEMPLATES

```
8   from flask import Flask, render_template
9   app = Flask(__name__)
10
11  @app.route('/')
12  def template():
13      return render_template('template.html', name='Song Song Hao')
14
15  if __name__ == "__main__":
16      app.run(debug=True)
```

使用 **Flask** 提供的 render_template(html檔名 , 需填入的變數名稱)
讓 Jinja 產生完整的 HTML檔

◎ 對應範例檔 : template.py

FLASK - TEMPLATES

◎ 對應範例檔：template.py

FLASK - JINJA TEMPLATE LANGUAGE

- Evaluating Expression, Variables and Function call, {{ }} operator :

```
from jinja2 import Template


Template("{{ 10 // 3 }}").render()
# >>> 3
Template("{{ var[2] }}").render(var=("c", "c++", "python"))
# >>> 'python'

double = lambda x : x*2
Template("{{ 2*double(2) }}").render(double=double)
# >>> 8
```

◎ 對應範例檔 : Jinja.py

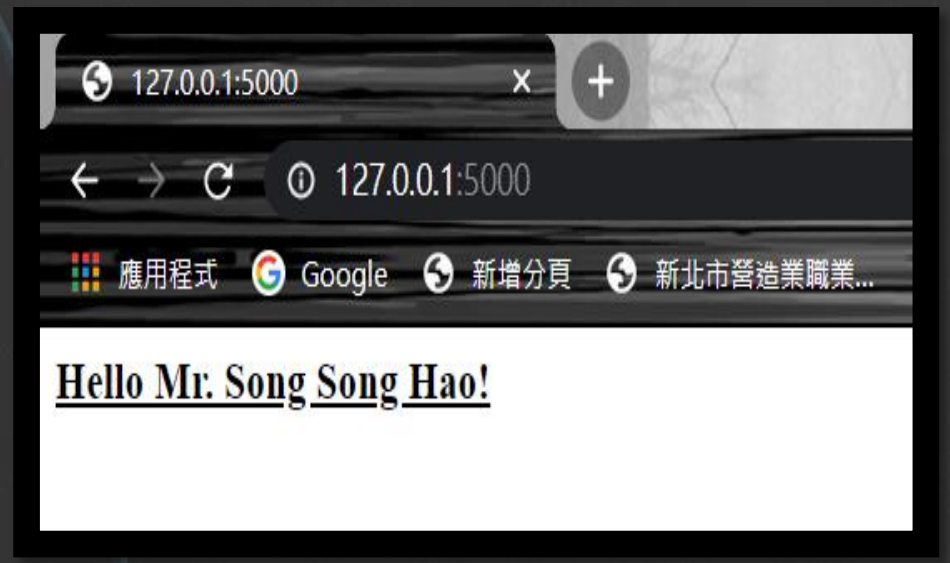
- Setting Variables, {{% %}} operator :

```
{% set employee = { 'name': 'tom', 'age': 25, 'designation': 'Manager' } %}
```

FLASK - JINJA TEMPLATE LANGUAGE

- If statement, {{% %}} operator :

```
<html>
    <body>
        {% if gender == "Male" %}
            <h3><u>Hello Mr. {{name}}!</u></h3>
        {% else %}
            <h3><u>Hello Mrs. {{name}}!</u></h3>
        {% endif %}
    </body>
</html>
```



render_template(if statement.html ,
name = “Song Song Hao”,
gender = “Male”)

◎ 對應範例檔 : if_statement.py

FLASK - JINJA TEMPLATE LANGUAGE

- For statement, {{% %}} operator :

```
<ul>
{% for number in num_list %}
        {% if number % 2 == 0 %}
                <li>{{ number }} is even !</li>
        {% endif %}
{% endfor %}
</ul>
```

render_template(for statement.html ,
num_list=[1,4,7,8,5,2])

◎ 對應範例檔 : for_statement.py

FLASK - CREATING URLS EFFICIENTLY

- Inefficient way : URLs = " /XXXX/YYYY/ZZZZ/ "
- More efficient way : url_for() function !
  - ➔ url_for(endpoint, *arg), 其中 endpoint 通常為 view function
  - ➔ 回傳 view function 所對應的 URL

```
14  @app.route("/")
15  def home():
16      return redirect(url_for("form"))
17
18  # serving form web page
19  @app.route("/my-form")
20  def form():
21      return render_template('form.html')
```

◎ 對應範例檔 : request_urlfor.py

如果需要儲存資料供每一個網頁使用？

SESSION

SQLALCHEMY

FLASK – SESSION

- 因為網頁與網頁之間資訊不會共享的特性，故出現了 Session 來補足這一問題。
- Session是一個被加密的小型文字檔案，作用是暫存文字，關閉伺服器就會消除資料。

FLASK – SESSION（用法類似DICTIONARY）

```python
from flask import Flask, session

app = Flask(__name__)
app.secret_key = "pisnai"


@app.route('/visits-counter/')
def visits():
    if 'visits' in session:
        session['visits'] = session.get('visits') + 1  # reading and updating session data
    else:
        session['visits'] = 1 # setting session data
    return "Total visits: {}".format(session.get('visits'))

@app.route('/delete-visits/')
def delete_visits():
    session.pop('visits', None) # delete visits
    return 'Visits deleted'

if __name__ == "__main__":
    app.run(debug=True)
```

若要使用**session**，必須要加。
secret_key可以自己設置

新增 一個**key : 'visits'** 到**session**中

將 **'visits'** 這個**key**從**session**中刪去

◎ 對應範例檔：session.py

FLASK - SESSION

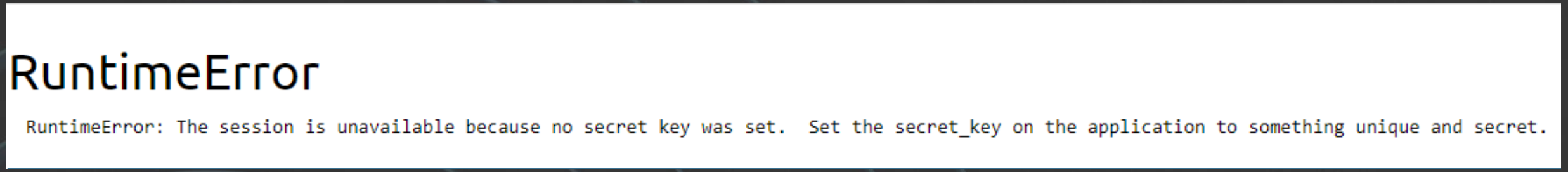
如果想要設定暫存的時間，將下面的程式碼加入。

```
24      from datetime import timedelta, datetime
25      session.permanet = True
26      session.permanent_session_lifetime = timedelta(hours = 1)
```

Attributes	說明
session.permanet	是否要定時刪去session
session.permanent_session_lifetime	session要儲存多少時間
timedelta	時間，可以放入minutes, hours, etc.

```
In [3]: print(timedelta(hours = 1))
1:00:00
```

FLASK - SESSION



如果出現上面的**ERROR**，請參考上兩頁的第**4**行，加上**secret_key**

FLASK_SQLALCHEMY

- Flask 本身不支援直接對資料庫操作，為了簡化 Flask 開發人員操作資料庫，產生了Flask 的擴充套件——Flask-SQLAlchemy 。

FLASK_SQLALCHEMY - INSTALLATION

```
pip install Flask-SQLAlchemy
```

FLASK - SQLALCHEMY

```python
from flask import Flask, url_for, redirect, render_template, flash, request, session
from datetime import timedelta, datetime
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.secret_key = "pisnai"
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///data.db"
app.config["SQLALCHEM

db = SQLAlchemy(app)

class data(db.Model):
    id = db.Column(db
    name = db.Column(
    classroom = db.Column( classroom , db.Integer, nullable = False)
    date_time = db.Column(db.DateTime, default = datetime.utcnow)

    def __init__(self, name, classroom):
        self.name = name
        self.classroom = classroom
db.create_all()
```

設定路徑（不可有中文）
防止Warning跳出

id	name	classroom	date_time

設定每筆資料需要的欄位

◎ 對應範例檔：sqlalchemy.py

FLASK - SQLALCHEMY

```
22    temp = data(name = "pisnai", classroom = "62113")
23    temp2 = data(name = "halosung", classroom = "62115")
24
25    db.session.add(temp)
26    db.session.add(temp2)
27    db.session.commit()
28    # or
29    db.session.add_all([temp, temp2])
30    db.session.commit()
31
32    for i in data.query.all():
33        print(i.id, i.name, i.classroom, i.date_time)
```

按照格式輸入資料

將temp和temp2加入資料集中

保存資料集

Output:

```
1 pisnai 62113 2020-11-11 14:06:15.945592
2 halosung 62115 2020-11-11 14:06:15.945592
```