WEB CONNECTIONS

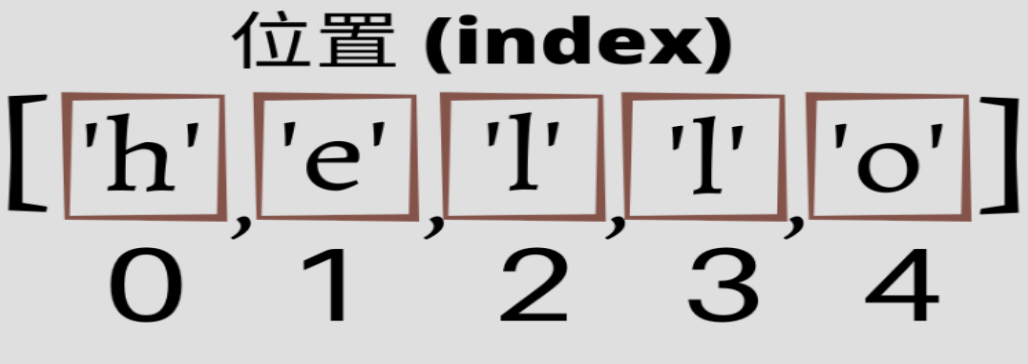
Contents

- List
- DataFrame and Pandas
- pymysql and streamlit
- Flask Frame
- Tiny Project

List

What is a list in Python?

- A list is a data structure in Python that **is a mutable, or changeable, ordered sequence of elements**. Each element or value that is inside of a list is called an item. Just as strings are defined as characters between quotes, lists are defined by having values between square brackets [ ] .
- For example, [‘h’, ‘e’, ‘l’, ‘l’, ‘o’]. List items are indexed, the first item has index [0], the second item has index [1] etc.



- List items can be of any data type

- When we say that lists are ordered, it means that the items have a defined order, and that order will not change. If you add new items to a list, the new items will be placed at the end of the list.
  - There are some list methods that will change the order, but in general: the order of the items will not change.
- The list is changeable, meaning that we can change, add, and remove items in a list after it has been created.
  - https://www.w3schools.com/python/python_lists_methods.asp
- To determine how many items a list has, use the *len()* function
- From Python's perspective, lists are defined as objects with the data type '`<class 'list'>`

- It is also possible to use the list() constructor when creating a new list.

```
# note the double round-brackets
thislist = (("apple", 'apple', 500,  54.66, True))
print(thislist)
print(type(thislist))
```

- There are four collection data types in the Python programming language:
  - List is a collection which is ordered and changeable. Allows duplicate members.
  - Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
  - Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members. Set *items* are unchangeable, but you can remove and/or add items whenever you like.
  - Dictionary is a collection which is ordered** and changeable. No duplicate members. As of Python version 3.7, dictionaries are *ordered*. In Python 3.6 and earlier, dictionaries are *unordered*.
- List operations

```
#Insert the value "orange" as the second element of the list:
thislist = ["apple", 'orange', 500,  54.66, True]
print(thislist)
thislist.insert(1,"KSU")
print(thislist)
thislist.pop(1)
print(thislist)
thislist.index(500)
```

```
['apple', 'orange', 500, 54.66, True]
['apple', 'KSU', 'orange', 500, 54.66, True]
['apple', 'orange', 500, 54.66, True]
Out[14]: 2
```

WEB+DB_ex0101.py

```python
thislist = ["apple", "banana", "cherry"]
print(thislist)
print(type(thislist))

thislist = ["apple", "banana", 500]
print(thislist)
print(type(thislist))

thislist = ["apple", 500,  54.66]
print(thislist)
print(type(thislist))

thislist = ["apple", 'apple', 500,  54.66]
print(thislist)
print(type(thislist))

thislist = ["apple", 'apple', 500,  54.66, True]
print(thislist)
print(type(thislist))

# note the double round-brackets
thislist = (("apple", 'apple', 500,  54.66, True))
print(thislist)
print(type(thislist))

#The append() method appends an element to the end of the list.
thislist = ["apple", 'orange', 500,  54.66, True]
thislist.append("KSU")
print(thislist)

#Insert the value "orange" as the second element of the list:
thislist = ["apple", 'orange', 500,  54.66, True]
print(thislist)
thislist.insert(1,"KSU")
print(thislist)
thislist.pop(1)
print(thislist)
thislist.index(500)
```

```
['apple', 'banana', 'cherry']
<class 'list'>
['apple', 'banana', 500]
<class 'list'>
['apple', 500, 54.66]
<class 'list'>
['apple', 'apple', 500, 54.66]
<class 'list'>
['apple', 'apple', 500, 54.66, True]
<class 'list'>
('apple', 'apple', 500, 54.66, True)
<class 'tuple'>
['apple', 'orange', 500, 54.66, True, 'KSU']
['apple', 'orange', 500, 54.66, True]
['apple', 'KSU', 'orange', 500, 54.66, True]
['apple', 'orange', 500, 54.66, True]
```

DataFrame and Pandas

DataFrame

- DataFrame is **a 2-dimensional labeled data structure with columns of potentially different types**. You can think of it like a spreadsheet or SQL table, or a dict of Series objects. It is generally the most commonly used pandas object.
- A Data frame is a two-dimensional data structure, i.e., **data is aligned in a tabular fashion in rows and columns**. We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.
- Pandas is a Python library **for data analysis**. Started by Wes McKinney in 2008 out of a need for a powerful and flexible quantitative analysis tool, pandas has grown into one of the most popular Python libraries.
- Pandas: It is an open-source, BSD-licensed library written in Python Language. Pandas provide high performance, fast, easy to use data structures and data analysis tools for manipulating numeric data and time series. Two structures are series and dataFrame in pandas.

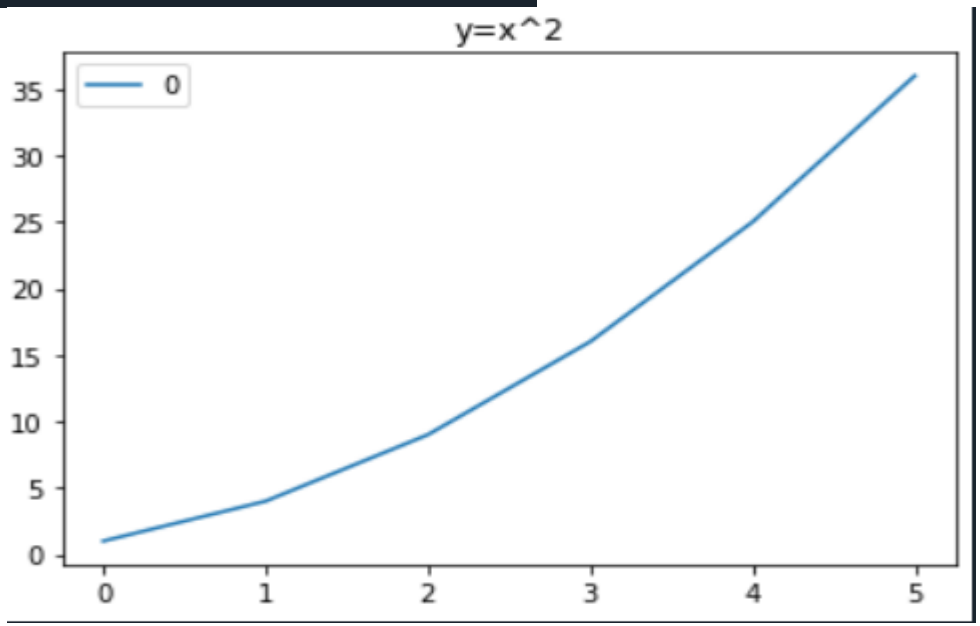
```python
import pandas as pd
# 準備傳入
data = {
    'name': ['David Wang', 'Merry Can', 'Fire Station', 'Spider Wang'],
    'email': ['min@gmail.com', 'hchang@gmail.com',
              'laioding@gmail.com', 'hsulight@gmail.com'],
    'grades': [60, 77, 92, 43]
}
# 建立 DataFrame 物件
student_df = pd.DataFrame(data)
print(student_df)
student_df01 = pd.DataFrame(data,
                  index=['one', 'second', 'third','fourth'])
print(student_df01)
```

```
          name              email  grades
0    David Wang      min@gmail.com      60
1     Merry Can   hchang@gmail.com      77
2  Fire Station laioding@gmail.com      92
3   Spider Wang hsulight@gmail.com      43
                name              email  grades
one        David Wang      min@gmail.com      60
second      Merry Can   hchang@gmail.com      77
third    Fire Station laioding@gmail.com      92
fourth    Spider Wang hsulight@gmail.com      43
```

```python
# 引入 pandas 套件，使用別名 pd 可以少打字
import pandas as pd
import numpy as np

df = pd.DataFrame([[np.nan, 2, np.nan, 0],
                   [3, 4, np.nan, 1],
                   [np.nan, np.nan, np.nan, 5],
                   [np.nan, 3, np.nan, 4]],
                  columns=list('ABCD'))

print(df)
print (df.fillna(0))
```

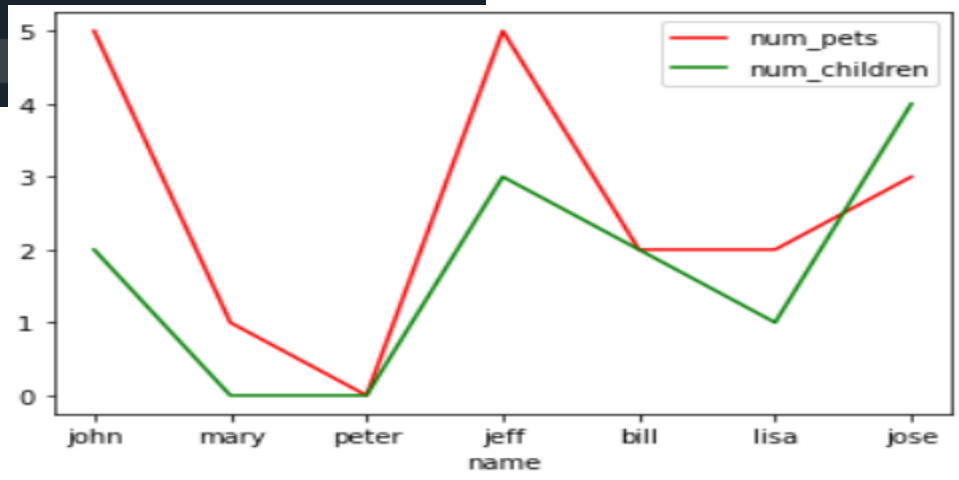
```
     A    B   C  D
0  NaN  2.0 NaN  0
1  3.0  4.0 NaN  1
2  NaN  NaN NaN  5
3  NaN  3.0 NaN  4
     A    B    C  D
0  0.0  2.0  0.0  0
1  3.0  4.0  0.0  1
2  0.0  0.0  0.0  5
3  0.0  3.0  0.0  4
```

```python
#繪圖(y=x^2)串列數據
import pandas as pd

dataframe = pd.DataFrame([1,4,9,16,25,36])
dataframe.plot(kind='line',title='y=x^2')
```



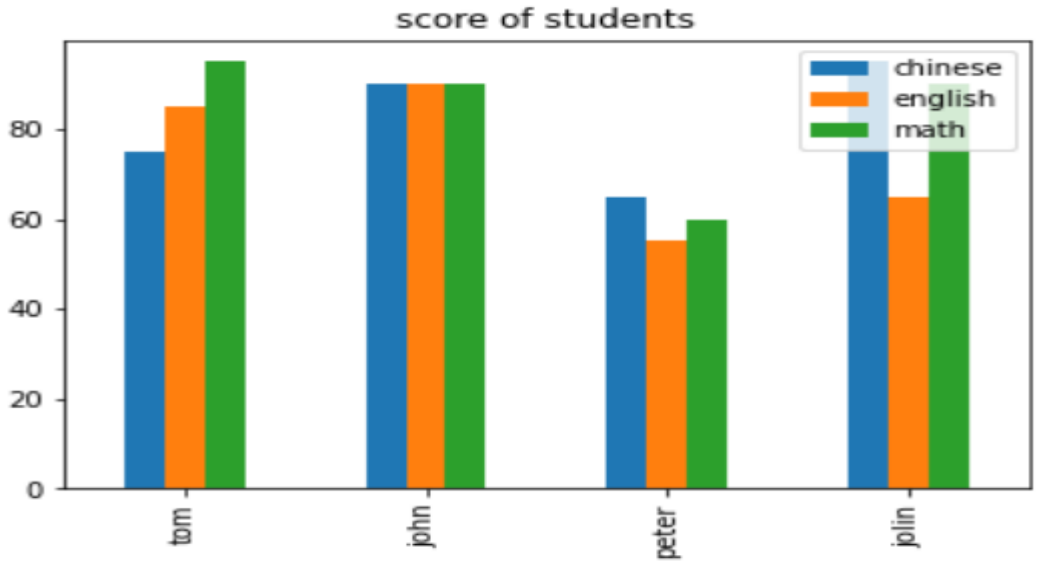
WEB+DB_ex0104.py

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

df = pd.DataFrame({
    'name':['john','mary','peter','jeff','bill','lisa','jose'],
    'age':[23,78,22,19,45,33,20],
    'gender':['M','F','M','M','M','F','M'],
    'state':['california','dc','california','dc',
             'california','texas','texas'],
    'num_children':[2,0,0,3,2,1,4],
    'num_pets':[5,1,0,5,2,2,3]
})
# gca 代表取得目figure前坐標軸 axis（gca = get current figure）
ax1 = plt.gca()

#顯示每人寵物數量（座標軸 = ax1）
df.plot(ax=ax1, kind='line',x='name',y='num_pets',color='red')
#顯示每人的小孩數量（座標軸 = ax1）
df.plot(ax=ax1, kind='line',x='name', y='num_children',color='green')
#兩個圖組，一起繪出
plt.show()
```



WEB+DB_ex0105.py

```python
#
import pandas as pd

#數據資料表，採用三個串列List(coulumn=course,record_name=name, record_data = score)
score = [[75,85,95],[90,90,90],[65,55,60],[95,65,90]]
name = ['tom','john','peter','jolin']
course = ['chinese','english','math']

#設定dataFrame設定資料結構
df = pd.DataFrame(score, index = name, columns = course)

#繪bar柱狀圖
df.plot(kind='bar',title='score of students')
```



WEB+DB_ex0106.py

pymysql and streamlit

https://docs.streamlit.io/library/api-reference

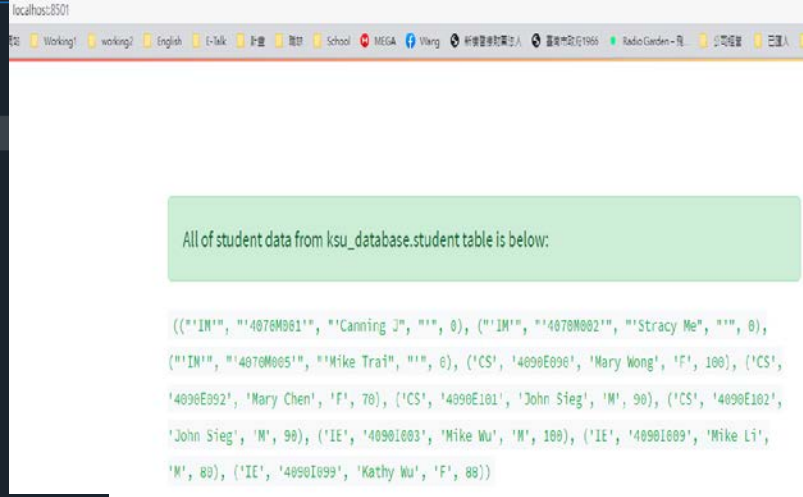
pymysql

- The package contains a pure-Python MySQL client library. Most public APIs are compatible with mysqlclient and MySQLdb.
- There are thee MySQL adapters for Python that are currently maintained:
  - mysqlclient – is connector for CPython. Requires the mysql-connector-c C library to work.
  - pyMySQL - Pure Python MySQL client. According to the maintainer of both mysqlclient and PyMySQL, you should use PyMySQL if:
    - You can't use libmysqlclient for some reason.
    - You want to use monkeypatched socket of gevent or eventlet.
    - You wan't to hack mysql protocol.
  - Mysqldb - MySQL connector developed by the MySQL group at Oracle, also written entirely in Python. It's performance appears to be the worst out of the three. Also, due to some licensing issues, you can't download it (but it's now available through conda).

streamlit

- Streamlit is an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science.
- Streamlit is an awesome new tool that allows engineers to quickly build highly interactive web applications around their data, machine learning models, and pretty much anything.
- The best thing about Streamlit is it doesn’t require any knowledge of web development.

```python
from pymysql import *
import streamlit as st

def GetDBdata():
    # create a connection to DB
    conn = connect(host='localhost',
                   port=3306,user='root',
                   password='',
                   database='ksu_database',
                   charset='utf8'
                   )
    # create cursor object
    cursor_data = conn.cursor()
    # send an SQL
    count = cursor_data.execute('SELECT * FROM student')

    # the rows selected from the DB table
    print("The number of rows is: %d " % count)

    # close the used objects in Memory
    cursor_data.close()
    conn.close()
    # require all of cursor's data
    result = cursor_data.fetchall()
    return(result)

###
st.success("All of student data from ksu_database.student table is below:\n")
st.write(GetDBdata())
```



Steps to run your code:

1. Open up Apache's MySQL
2. Load the schema "student_tableschema.sql"
3. "streamlit run [pythonFilename]" under the right folder
4. Go to your browser to see the results

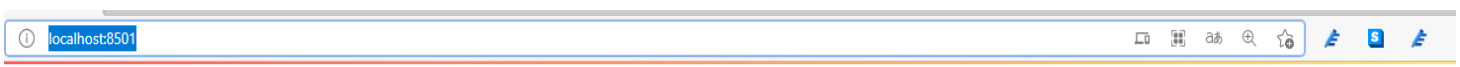
Major	Student_ID	Name	Sex	Grade
'IM'	'4070M001'	'Canning J	'	0
'IM'	'4070M002'	'Stracy Me	'	0
'IM'	'4070M005'	'Mike Trai	'	0
CS	4090E090	Mary Wong	F	100
CS	4090E092	Mary Chen	F	70
CS	4090E101	John Sieg	M	90
CS	4090E102	John Sieg	M	90
IE	4090I003	Mike Wu	M	100
IE	4090I009	Mike Li	M	80
IE	4090I099	Kathy Wu	F	88

```
(base) E:\教學\___python+資料分析+AI +DM\exercises>streamlit run  GetDBandShowDBdata01.py
2021-12-22 12:14:11.588 INFO    numexpr.utils: NumExpr defaulting to 2 threads.

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.109:8501

  A new version of Streamlit is available.

  See what's new at https://discuss.streamlit.io/c/announcements

  Enter the following command to upgrade:
  $ pip install streamlit --upgrade
The number of rows is: 10
```

GetDBandShowDBdata01.py

Output

```
(base) E:\教學\___python+資料分析+AI +DM\exercises>streamlit run  GetDBandShowDBdata01.py
2021-12-22 12:14:11.588 INFO    numexpr.utils: NumExpr defaulting to 2 threads.

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.109:8501


  A new version of Streamlit is available.

  See what's new at https://discuss.streamlit.io/c/announcements

  Enter the following command to upgrade:
  $ pip install streamlit --upgrade

The number of rows is: 10
```



All of student data from ksu_database.student table is below:

```
(("'IM'", "'4070M001'", "'Canning J", "'", 0), ("'IM'", "'4070M002'", "'Stracy Me", "'", 0),
("'IM'", "'4070M005'", "'Mike Trai", "'", 0), ('CS', '4090E090', 'Mary Wong', 'F', 100), ('CS',
'4090E092', 'Mary Chen', 'F', 70), ('CS', '4090E101', 'John Sieg', 'M', 90), ('CS', '4090E102',
'John Sieg', 'M', 90), ('IE', '4090I003', 'Mike Wu', 'M', 100), ('IE', '4090I009', 'Mike Li',
'M', 80), ('IE', '4090I099', 'Kathy Wu', 'F', 88))
```

GetDBandShowDBdata01.py

```python
from pymysql import *
import streamlit as st

# create a connection to DB
conn = connect(host='localhost',
               port=3306,user='root',
               password='',
               database='ksu_database',
               charset='utf8')

# create cursor object
cursor = conn.cursor()

# send an SQL
count = cursor.execute('SELECT * FROM student')
result_all = cursor.fetchall()

# the rows selected from the DB table
print("The number of rows is: %d " % count)

# for Web page
st.success("All of student data from ksu_database.student table is below:\n")

for row in result_all:
    print   (row[0], "\t", row[1], "\t",
             row[2], "\t", row[3], "\t", row[4] )
    st.write(row[0], " ", row[1], " ",
             row[2], " ", row[3], " ", row[4] )

# close the used objects in Memory
cursor.close()
conn.close()
```

GetDBandShowDBdata02.py

Output

```
(base) E:\教學\___python+資料分析+AI +DM\exercises>streamlit run  GetDBandShowDBdata02.py
2021-12-22 12:41:29.393 INFO    numexpr.utils: NumExpr defaulting to 2 threads.
The number of rows is: 10

  You can now view your Streamlit app in your browser.

  Local URL: http://localhost:8501
  Network URL: http://192.168.1.109:8501

The number of rows is: 10
'IM'      '4070M001'      'Canning J      '       0
'IM'      '4070M002'      'Stracy Me      '       0
'IM'      '4070M005'      'Mike Trai      '       0
CS        4090E090        Mary Wong       F       100
CS        4090E092        Mary Chen       F       70
CS        4090E101        John Sieg       M       90
CS        4090E102        John Sieg       M       90
IE        4090I003        Mike Wu         M       100
IE        4090I009        Mike Li         M       80
IE        4090I099        Kathy Wu        F       88
```

All of student data from ksu_database.student table is below:

'IM' '4070M001' 'Canning J ' 0

'IM' '4070M002' 'Stracy Me ' 0

'IM' '4070M005' 'Mike Trai ' 0

CS 4090E090 Mary Wong F 100

CS 4090E092 Mary Chen F 70

CS 4090E101 John Sieg M 90

CS 4090E102 John Sieg M 90

IE 4090I003 Mike Wu M 100

IE 4090I009 Mike Li M 80

IE 4090I099 Kathy Wu F 88

```python
import pandas as pd

# create a connection to DB
conn = connect(host='localhost',
               port=3306,user='root',
               password='',
               database='ksu_database',
               charset='utf8')

# create cursor object
cursor = conn.cursor()

# send an SQL
sql='SELECT * FROM student'
count = cursor.execute(sql)
result_all = cursor.fetchall()

# the rows selected from the DB table
print("The number of rows is: %d " % count)

# for Web page
st.title('Web output!')
st.text ('All of student data from ksu_database.student table is below:')
st.text ('* Show DB data using stream.write():')
html_string = "<h3 style=\"background-color:yellow\"> this is an html string</h3>"
st.markdown(html_string, unsafe_allow_html=True)

# Show DB data using stream.write()
for row in result_all:
    print   (row[0], "\t", row[1], "\t",
             row[2], "\t", row[3], "\t", row[4] )
    st.write(f"{row[0]}  {row[1]}  {row[2]}  {row[3]}  {row[4]}" )

# transfor DB data to dataframe without indexing
st.text ('* Show DB data using DataFrame:')
df=pd.DataFrame(result_all)
#df.set_index(row[0], inplace=True)
styler = df.style.hide_index()
st.write(styler.to_html(), unsafe_allow_html=True)

# Show DB data using stream.table():
st.text ('* Show DB data using stream.table():')
st.table(result_all)
st.success("Streamlit runs successfully!")

# close the used objects in Memory
cursor.close()
conn.close()
```

GetDBandShowDBdata06.py

Web output!

All of student data from ksu_database.student table is below:

* Show DB data using stream.write():

this is an html string

'IM' '4070M001' 'Canning J ' 0

'IM' '4070M002' 'Stracy Me ' 0

'IM' '4070M005' 'Mike Trai ' 0

CS 4090E090 Mary Wong F 100

CS 4090E092 Mary Chen F 70

CS 4090E101 John Sieg M 90

CS 4090E102 John Sieg M 90

IE 4090I003 Mike Wu M 100

IE 4090I009 Mike Li M 80

IE 4090I099 Kathy Wu F 88

* Show DB data using DataFrame:

0	1	2	3	4
'IM'	'4070M001'	'Canning J	'	0
'IM'	'4070M002'	'Stracy Me	'	0
'IM'	'4070M005'	'Mike Trai	'	0
CS	4090E090	Mary Wong	F	100
CS	4090E092	Mary Chen	F	70
CS	4090E101	John Sieg	M	90
CS	4090E102	John Sieg	M	90
IE	4090I003	Mike Wu	M	100
IE	4090I009	Mike Li	M	80
IE	4090I099	Kathy Wu	F	88

* Show DB data using stream.table():

	0	1	2	3	4
0	'IM'	'4070M001'	'Canning J	'	0
1	'IM'	'4070M002'	'Stracy Me	'	0
2	'IM'	'4070M005'	'Mike Trai	'	0
3	CS	4090E090	Mary Wong	F	100
4	CS	4090E092	Mary Chen	F	70
5	CS	4090E101	John Sieg	M	90
6	CS	4090E102	John Sieg	M	90
7	IE	4090I003	Mike Wu	M	100
8	IE	4090I009	Mike Li	M	80
9	IE	4090I099	Kathy Wu	F	88

Streamlit runs successfully!

GetDBandShowDBdata08.py

```
# sidebar creation
st.sidebar.header('User Input Parameters')
table_name = st.sidebar.text_input('table name', '')

try:
    # create a connection to DB
    conn = connect(host='localhost', port=3306,user='root',
                   password='', database='ksu_database',
                   charset='utf8')

    # create cursor object
    cursor = conn.cursor()

    # send an SQL
    sql='SELECT * FROM '+ table_name
    count = cursor.execute(sql)
    result_all = cursor.fetchall()

    # the rows selected from the DB table
    print("The number of rows is: %d ", count)

    # for Web page
    st.title('Web output! ')
    st.text ('All of student data from ksu_database.student table is below:')

    # transfor DB data to dataframe without indexing
    st.header ('* Show DB data using DataFrame:')
    df=pd.DataFrame(result_all)
    styler = df.style.hide_index()
    st.write(styler.to_html(), unsafe_allow_html=True)

    # close the used objects in Memory
    cursor.close()
    conn.close()

except:
    st.write('Waiting!')
```

Output

User Input Parameters

table name

student

Web output!

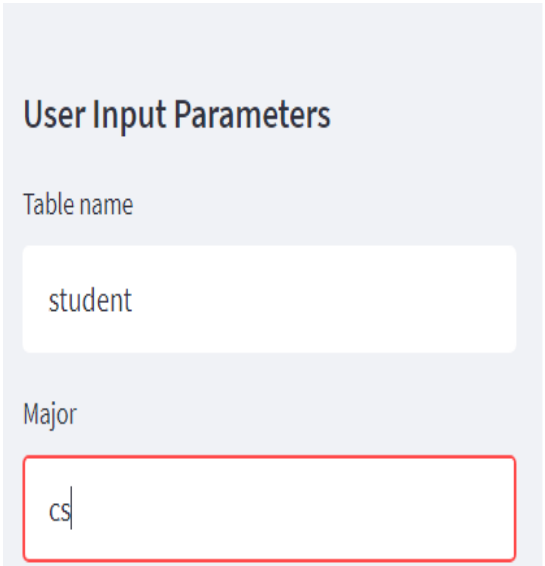
All of student data from ksu_database.student table is below:

* Show DB data using DataFrame:

0	1	2	3	4
IM	4070M001	Canning F	M	96
IM	4070M002	Canning F	M	96
IM	4070M003	Canning WU	M	96
CS	4090E090	Mary Wong	F	100
CS	4090E092	Mary Chen	F	70
CS	4090E101	John Sieg	M	90
CS	4090E102	John Sieg	M	90
IE	4090I003	Mike Wu	M	100
IE	4090I009	Mike Li	M	80
IE	4090I099	Kathy Wu	F	88

Output

GetDBandShowDBdata09.py

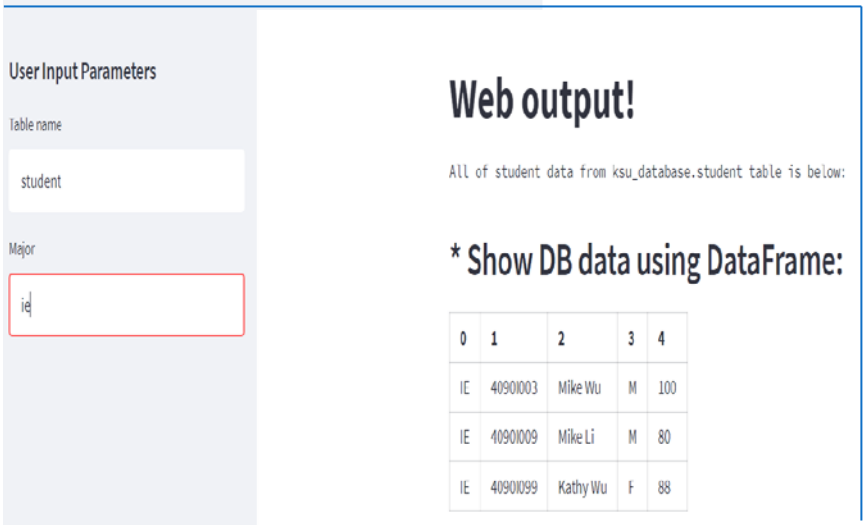


Web output!

All of student data from ksu_database.student table is below:

* Show DB data using DataFrame:

0	1	2	3	4
CS	4090E090	Mary Wong	F	100
CS	4090E092	Mary Chen	F	70
CS	4090E101	John Sieg	M	90
CS	4090E102	John Sieg	M	90



GetDBandShowDBdata10.py

User Input Parameters

Insert data into student table

Major

WW

Student ID

0910E4040

Name

Dave Wang

Sex

M

Grade

99

Web output!

* Show DB data using DataFrame:

0	1	2	3	4
WW	0910E4040	Dave Wang	M	99
IM	4070M001	Canning F	M	96
IM	4070M002	Canning F	M	96
IM	4070M003	Canning WU	M	96
CS	4090E090	Mary Wong	F	100
CS	4090E092	Mary Chen	F	70
CS	4090E101	John Sieg	M	90
CS	4090E102	John Sieg	M	90
IE	4090I003	Mike Wu	M	100
IE	4090I009	Mike Li	M	80
IE	4090I099	Kathy Wu	F	88

GetDBandShowDBdata11.py

User Input Parameters in column 1

Table name

student

Web output!

User Input Parameters in column 2

Major

CS

* Show DB data using DataFrame:

0	1	2	3	4
CS	4090E090	Mary Wong	F	100
CS	4090E092	Mary Chen	F	70
CS	4090E101	John Sieg	M	90
CS	4090E102	John Sieg	M	90

GetDBandShowDBdata11a.py

User Input Parameters in column 1

Table name

student

User Input Parameters in column 2

Major

IE

The favorate date

2022/01/20

* Show DB data using DataFrame:

0	1	2	3	4
IE	40901003	Mike Wu	M	100
IE	40901009	Mike Li	M	80
IE	40901099	Kathy Wu	F	88

January 2022

S	M	T	W	T	F	S
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

User Input Parameters in column 1

Table name

student

* Show DB data using DataFrame:

0	1	2	3	4
IE	40901003	Mike Wu	M	100
IE	40901009	Mike Li	M	80
IE	40901099	Kathy Wu	F	88

Column 1

Table name

student

Column 2

Major

CS

Major	StudentID	Name	Sex	Grade
CS	4090E090	Mary Wong	F	100
CS	4090E092	Mary Chen	F	70

Column 1

Table name

student

CS

Major	StudentID	Name	Sex	Grade
CS	4090E090	Mary Wong	F	100
CS	4090E092	Mary Chen	F	70
CS	4090E101	John Sieg	M	90
CS	4090E102	John Sieg	M	

Column 1

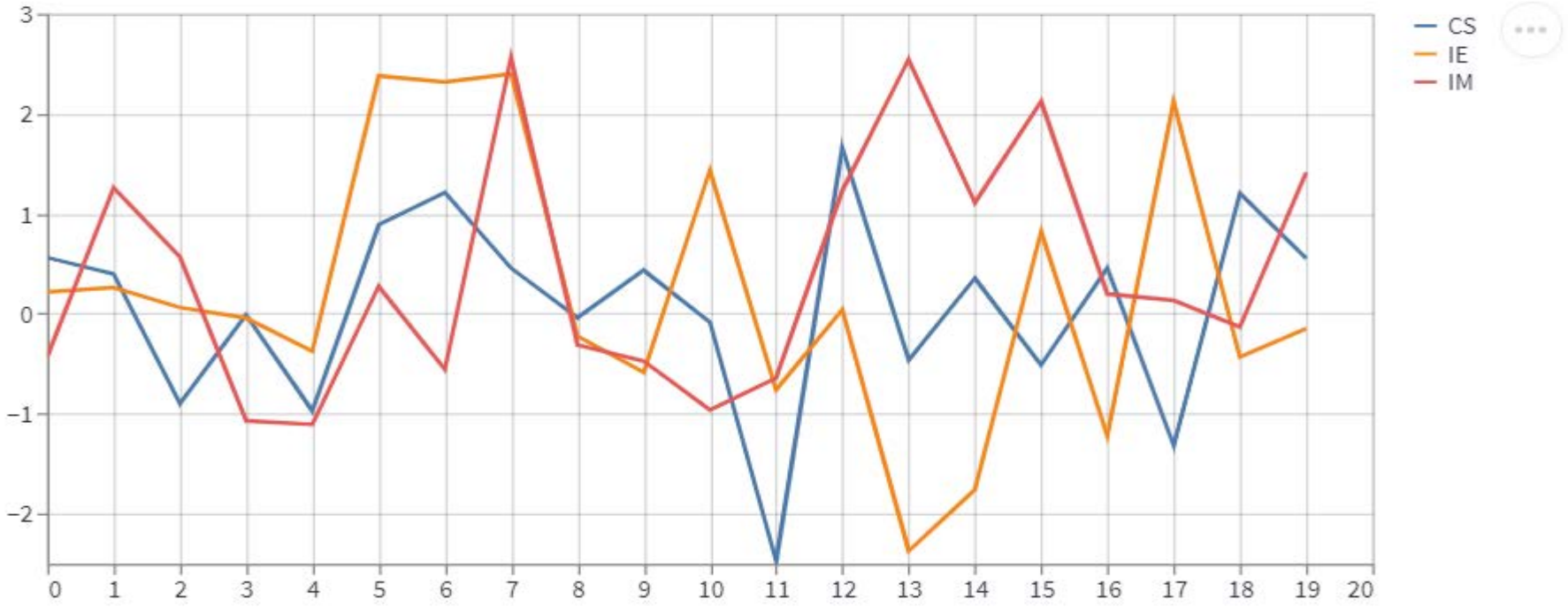
Table name

student

Column 2

Major

CS



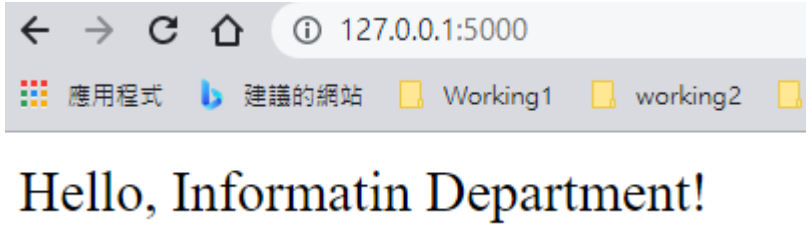
Flask Frame

Python Flask

- FYI: https://flask.palletsprojects.com/en/2.0.x/
- Flask is a lightweight web frame of Python. It provides the user with libraries, modules and tools to help build Web-Applications.
- Installation and run:
  - conda install flask or pip install flask
  - save this as app.py

```
# save this as app.py
import flask
app = flask.Flask(__name__)

# http://127.0.0.1:5000/  or  http://127.0.0.1:5000/hello
@app.route("/")
# @app.route("/hello")
def hello():
    return "Hello, Informatin Department!"
```

  - flask run (p.s. go to the right folder where app.py is located on your terminal session!)
  - run http://127.0.0.1:5000/ on your Chrome



Extra talk: import

- Use the second way to make your code concise
  - way 1:

```
import Flask
app = flask.Flask(__name__)
```

  - way 2:

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

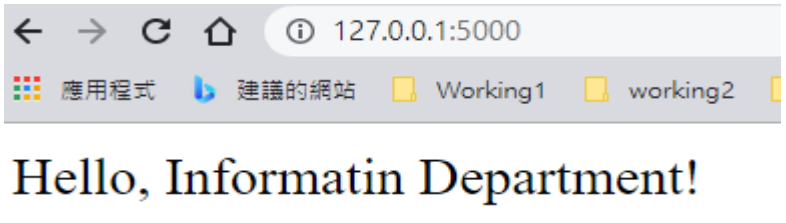
Extra talk: __name__

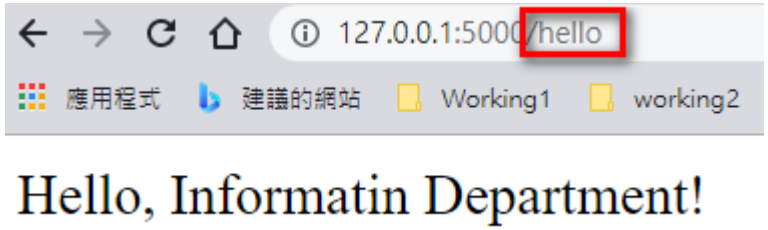
- Flask(__name__) : If you are using a single module, **name** is always the correct value. If you however are using a package, it's usually recommended to hardcode the name of your package there. Namely, tell Flask your application name here!
- Try more: use Ctrl+ "C" to cut off the connection.Then rerun it

WEBDBbar.py × WEBDBfoo.py ×

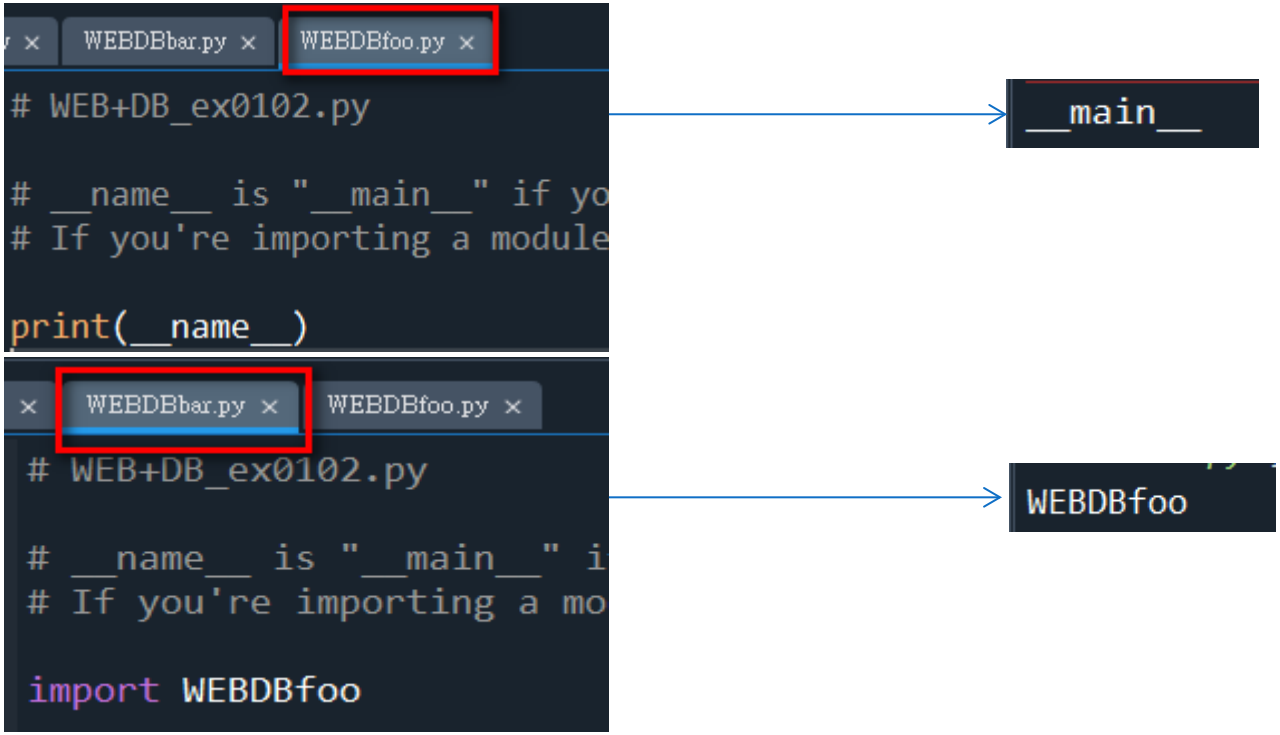
```
# save this as app.py
import flask
app = flask.Flask(__name__)

# http://127.0.0.1:5000/  or  http://127.0.0.1:5000/hello
@app.route("/")
@app.route("/hello")
def hello():
    return "Hello, Informatin Department!"
```

App routing is used to map the specific URL with the associated function that is intended to perform some task. I n other words, we can say that if we visit the particular URL mapped to some particular function, the output of that function is rendered on the browser's screen





- __name__ is "__main__" if you're executing the script directly.
- If you're importing a module, __name__ is the name of the module.



Extra talk: @app.route()

- App routing is used to map the specific URL with the associated function that is intended to perform some task. I n other words, we can say that if we visit the particular URL mapped to some particular function, the output of that function is rendered on the browser's screen
- Namely, tell Flask which `url` we want to use and what its correspond function to process the `url`
- The ”flask run” command is the preferred way to start the development server.

Debug on

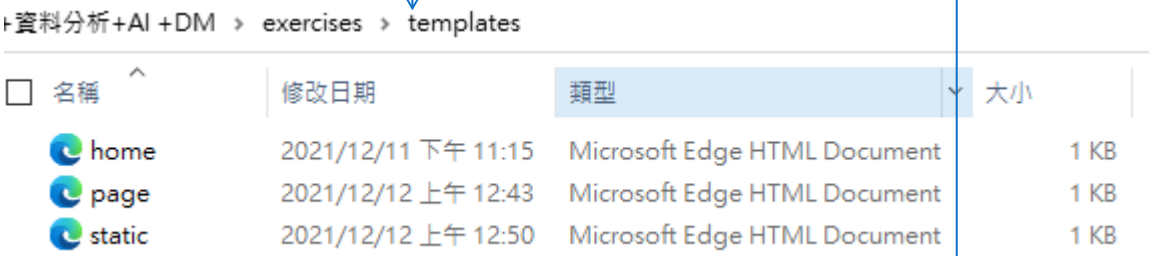
- Setup debug mode on by using FLASK_ENV=development on your terminal. It can dubug and reload your code

```
# save this as app.py
import flask
app = flask.Flask(__name__)

# http://127.0.0.1:5000/  or  http://127.0.0.1:5000/hello
@app.route("/")
@app.route("/hello")
def hello():
    return "Hello, Informatin Department!"
if __name__ == '__main__':
    app.run(debug=True)
```

```python
# save this as app.py
import flask
app = flask.Flask(__name__)

# http://127.0.0.1:5000/  or  http://127.0.0.1:5000/hello
@app.route("/")
@app.route("/hello")
def hello():
    return "Hello, Informatin Department!"

# http://127.0.0.1:5000/Android   String=>Android
@app.route('/<name>', methods=['GET'])
def queryDataMessageByName(name):
    print("type(name) : ", type(name))
    return 'String => {}'.format(name)

# http://127.0.0.1:5000/66   int=> 66
@app.route('/<int:id>', methods=['GET'])
def queryDataMessageById(id):
    print("type(id) : ", type(id))
    return 'int => {}'.format(id)

# http://127.0.0.1:5000/77.34   float=> 77.34
@app.route('/<float:num>', methods=['GET'])
def queryDataMessageByVersion(num):
    print("type(num) : ", type(num))
    return 'float => {}'.format(num)

@app.route('/text')
def text():
    return '<html><body><h1>Hello World</h1></body></html>'

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

app.py

The related files settings



- /templates
  - /home.html
  - /page.html
  - /static.html
- /static
  - /script.js
  - /style.css

```
<!DOCTYPE html>
<html lang="en">
<head>
        <meta charset="UTF-8">
        <title>Home</title>
</head>
<body>
        <h1>My Website Text</h1>
        <table border="1">
                <tr>
                        <td>Wang Wang Wang</td>
                        <td>Text Text Text</td>
                        <td>Text Text Text</td>
                </tr>
                <tr>
                        <td>Prof. Wang</td>
                        <td>Text Text Text</td>
                        <td>Text Text Text</td>
                </tr>
                <tr>
                        <td>Dr. Wang</td>
                        <td>Text Text Text</td>
                        <td>Text Text Text</td>
                </tr>
        </table>
</body>
</html>
```

Home.html

Change your app.py

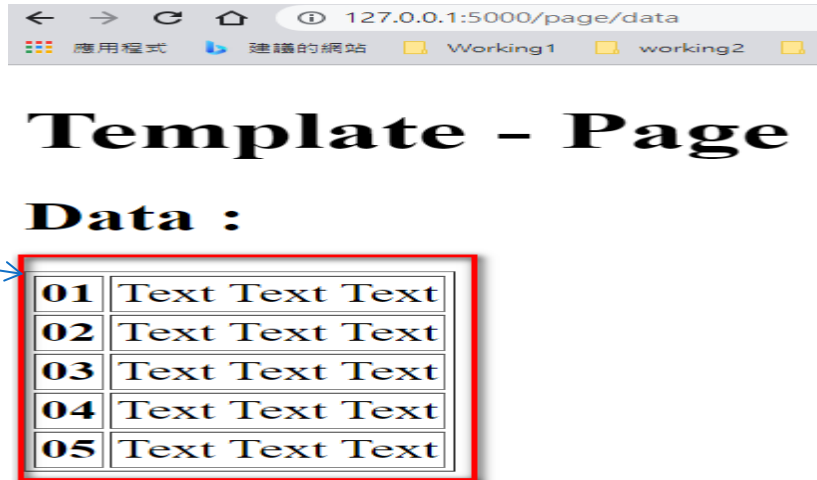
```python
# save this as app.py
from flask import Flask, render_template
app = Flask(__name__)

###
@app.route('/text')
def text():
    return '<html><body><h1>Hello World</h1></body></html>'

@app.route('/home')
def home():
    return render_template('home.html')

@app.route('/page/text')
def pageText():
    return render_template('page.html', text="Python Flask !")

@app.route('/page/app')
def pageAppInfo():
    appInfo = {  # dict
        'id': 5,
        'name': 'Python - Flask',
        'version': '1.0.1',
        'author': 'Enoxs',
        'remark': 'Python - Web Framework'
    }
    return render_template('page.html', appInfo=appInfo)

@app.route('/page/data')
def pageData():
    data = {  # dict
        '01': 'Text Text Text',
        '02': 'Text Text Text',
        '03': 'Text Text Text',
        '04': 'Text Text Text',
        '05': 'Text Text Text'
    }
    return render_template('page.html', data=data)

@app.route('/static')
def staticPage():
    return render_template('static.html')
```

```python
###
@app.route("/")
@app.route("/hello")
def hello():
    return "Hello, Informatin Department!"

# http://127.0.0.1:5000/Android   String=>Android
@app.route('/<name>', methods=['GET'])
def queryDataMessageByName(name):
    print("type(name) : ", type(name))
    return 'String => {}'.format(name)

# http://127.0.0.1:5000/66   int=> 66
@app.route('/<int:id>', methods=['GET'])
def queryDataMessageById(id):
    print("type(id) : ", type(id))
    return 'int => {}'.format(id)

# http://127.0.0.1:5000/77.34   float=> 77.34
@app.route('/<float:num>', methods=['GET'])
def queryDataMessageByVersion(num):
    print("type(num) : ", type(num))
    return 'float => {}'.format(num)

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

app.py

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width,
          initial-scale=1.0">
    <title>Template - Page</title>
</head>

<body>
    <h1>Template - Page </h1>
    <h2>{{text}}</h2>

    {% if appInfo != undefined %}

    <h2>AppInfo : </h2>
    <p>id : {{appInfo.id}}</p>
    <p>name : {{appInfo.name}}</p>
    <p>version : {{appInfo.version}}</p>
    <p>author : {{appInfo.author}}</p>
    <p>remark : {{appInfo.remark}}</p>
    {% endif %}

    {% if data != undefined %}
    <h2>Data : </h2>
    <table border="1">
        {% for key, value in data.items() %}
        <tr>
            <th> {{ key }} </th>
            <td> {{ value }} </td>
        </tr>
        {% endfor %}
    </table>
    {% endif %}
</body>
</html>
```

page.html



Jinja2

- Python Flask uses Jinja2's template. Jinja2 is a Python library that allows us to build expressive and extensible templates. It has special placeholders to serve dynamic data. It also can be used in Django.
  - {{ }}

    It doesn't mean anything in html itself, it means something in Django template language. For example:

    ```
    {{ choice.choice_text }}
    ```

    will substitute value of that variable during template rendering.

  - {% %}

    The other one {% and %} is used for template processing, for example to indicate to template processor that some task needs to be completed. Good example is:

    ```
    {% if error %}
        {{ error }}
    {% endif %}
    ```

    That means that the variable error will be displayed (rendered) only if it exists, or to be more precise if it has some value.

Check Variable in Jinja2

Check if variable *is defined* (*exists*):

```
{% if variable is defined %}
    variable is defined
{% else %}
    variable is not defined
{% endif %}
```

Check if variable *is empty*:

```
{% if variable|length %}
    variable is not empty
{% else %}
    variable is empty
{% endif %}
```

```
{% if appInfo != undefined %}
```

If `appInfo` has data, then the following `html` code work!

For loop in Jinja2

- Use for loop to show the values of keys and contents. An display the output on the browser together with html table code.

```
<table border="1">
    {% for key, value in data.items() %}
    <tr>
        <th> {{ key }} </th>
        <td> {{ value }} </td>
    </tr>
    {% endfor %}
</table>
```

render_template()

- render_template is a Flask function from the flask. ... render_template is **used to generate output from a template file based on the Jinja2 engine** that is found in the application's templates folder

`flask.render_template(template_name_or_list, **context)`

Renders a template from the template folder with the given context.

Parameters:
- **template_name_or_list** (*Union[str, List[str]]*) – the name of the template to be rendered, or an iterable with template names the first one existing will be rendered
- **context** (*Any*) – the variables that should be available in the context of the template.

Return type: str

```
@app.route('/page/text')
def pageText():
    return render_template('page.html', text="Python Flask !")
```

The second parameter in render_template() can be assignment statement!

```python
@app.route('/static')
def staticPage():
    return render_template('static.html')
```

static/script.js

```javascript
function sayHello(){
    alert("Hi Everyone!  I am in script.js!");
}
```

templates/static.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Static - Page</title>
</head>
<body>
    <h1>Template - Page</h1>
    <button id="btnHello" onClick="sayHello()">Say Hello</button>
    <!-- <script src="../static/script.js"></script> -->
    <script type = "text/javascript"
     src = "{{ url_for('static', filename = 'script.js') }}" ></script>
</body>
</html>
```

For static folder

File name

```
from flask import Flask, request, render_template, redirect, url_for
app = Flask(__name__)

@app.route('/form')
def formPage():
    return render_template('Form.html')

@app.route('/submit', methods=['POST', 'GET'])
def submit():
    if request.method == 'POST':
        user = request.form['user']
        print("post : user => ", user)
        return redirect(url_for('success', name=user, action="post"))
    else:
        user = request.args.get('user')
        print("get : user => ", user)
        return redirect(url_for('success', name=user, action="get"))

@app.route('/success/<action>/<name>')
def success(name, action):
    return '{} : Welcome {} ~ !!!'.format(action, name)

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

Modify your code again!

templates/form.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Form - Submit</title>
</head>
<body>
    <h2>POST</h2>
    <form action="/submit" method="post">
        <h2>Enter Name:</h2>
        <p><input type="text" name="user" /></p>
        <p><input type="submit" value="submit" /></p>
    </form>
    <h2>GET</h2>
    <form action="/submit" method="get">
        <h2>Enter Name:</h2>
        <p><input type="text" name="user" /></p>
        <p><input type="submit" value="submit" /></p>
    </form>
</body>
</html>
```

form.html

Get v.s. Post

- GET is used for viewing something, without changing it, while POST is used for changing something.
- For example, a search page should use GET to get data while a form that changes your password should use POST . Essentially GET is used to retrieve remote data, and POST is used to insert/update remote data.
- Additionally, Both GET and POST method is used to transfer data from client to server in HTTP protocol but Main difference between POST and GET method is that **GET carries request parameter appended in URL string** **while POST carries request parameter in message body** which makes it more secure way of transferring data from client to

jQuery - Ajax

- /templates
  - /data.html
- /static
  - /jquery-3.6.0.min.js
  - /data
    - /input.json
    - /message.json

Tiny Project

This course will give you ideas and examples how to use list, dataframe, pymysql, streamlit, and Flask by using Python. However, the integration of pymysql and Flask will not be included in the course contents.

The tiny project will give you the opportunity for the integration. Additionally, you can improve your personal learning capability by the project as you work in IT.

Subject: E-Resume

- Please construct your e-resume with skills in this Chapter.
- The computer languages you possibly use are python, html, cs, and Javascript. However, no php will be used.
- The tools and for you possible are xampp, streamlit, flask, and pymysql.
- Please follow the requirements above.
- Submission:
  - Please use app.py as your entry point file name in which it will connect to other files.
  - Please submit zip your flask structure files including app.py, a plain text file describing the names and IDs for your team members, your flask structure and corresponding table schema. Attention, you can put your description anything on the plain text you want to remind me.
  - Please connect to MySQL to get your data for the partial data for your e-resume.
  - The more people is in a team, the more works need to be done for getting same quality grade as less people team.
  - 4 people at most a team.
  - Please change the Flask framework I give you. And modify and submit it.
  - No late submission.
  - Additionally, the team with unique works and fitting the above requirements will have a higher grade.
  - 2022/01/07 Due on the project.

The End