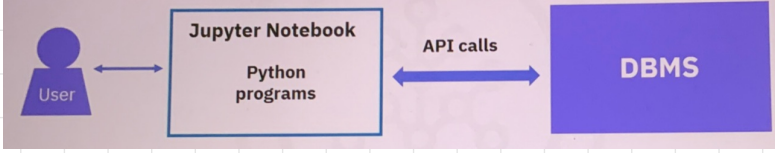
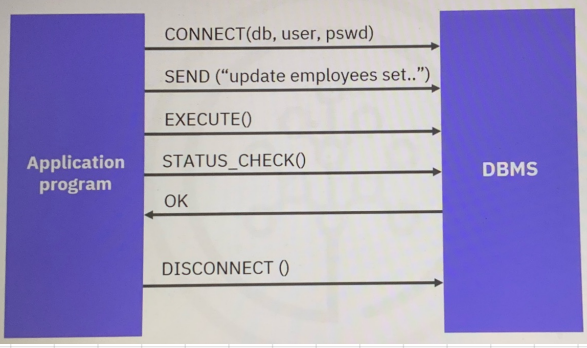
Accessing databases using Python

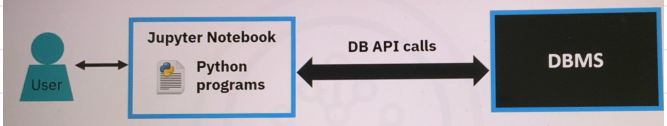


What is a SQL API?



Writing Code Using DB-API

What is a DB-API?



- Python's standard API for accessing relational databases
- Allows a single program that to work with multiple kinds of relational databases
- Learn DB-API functions once, use them with any database

Benefits of using DB-API

- Easy to implement and understand
- Encourages similarity between the Python modules used to access databases
- Achieves consistency
- Portable across databases
- Broad reach of database connectivity from Python

Concepts of the Python DB API

Connection Objects
- Database connections
- Manage transactions

Cursor Objects
- Database Queries
- Scroll through result set
- Retrieve results

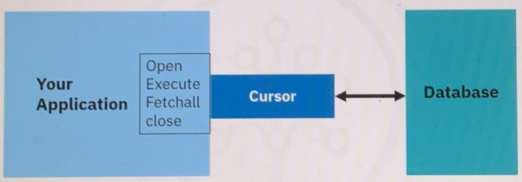
What are Connection methods?

- .cursor()
- .commit()
- .rollback()
- .close()

- .callproc()
- .execute()
- .executemany()
- .fetchone()
- .fetchmany()
- .fetchall()
- .nextset()
- .arraysize()
- .close()

นั่นคือการเปลี่ยนแปลงใด ๆ ที่ทำกับ

What is a database cursor?



Writing code using DB-API

```
from dbmodule import connect
#Create connection object
Connection = connect('databasename', 'username', 'pswd')
#Create a cursor object
Cursor=connection.cursor()
```

1 (create connection object)

2 (create a cursor object) — point to data that has connected

```
#Run Queries
Cursor.execute('select * from mytable')
Results=cursor.fetchall()
#Free resources
Cursor.close()
```

3 Queries

Results=cursor.fetchall() ← retrieve data that has executed

4 → Important: to prevent Resources unnessary used

Accessing Databases with SQL Magic

Magic Statements in Jupyter Notebooks

Magic commands are special commands that provide special functionalities.

- They are not valid Python code but affect the behavior of the notebook.
- They are designed to solve standard data analysis problems.

jupyter

Types of Cell Magics

Line Magics — Commands that are prefixed with a single % character and operate on a single line of input.

Cell Magics — Commands that are prefixed with two %% characters and operate on multiple lines of input.

Using Line Magic Statements

Line Magics	Uses
%pwd	prints the current working directory
%ls	lists all files in the current directory
%history	shows the command history
%reset	resets the namespace by removing all names defined by the user
%who	lists all variables in the namespace
%whos	provides more detailed information about all variables in the namespace
%matplotlib inline	makes matplotlib plots appear within the notebook
%timeit	times the execution of a single statement
%lsmagic	lists all available line magics

ทั้ง คู่ใน line [illegible]

Using Line Magic Statements

```
%pwd
%ls
```

→ print current directory

Both Line magics in the same cell

```
%timeit <statement>
```

Line Magic: Time for executing single statement

```
%%timeit
<statement_1>
<statement_2>
<statement_3>
```

Cell Magic: Time for executing the whole cell

```
%%writefile myfile.txt
<statement_1>
<statement_2>
<statement_3>
```

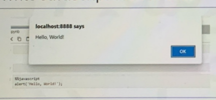
Writes all statements of the cell to myfile.txt

Using Cell Magic Statements

%%HTML — Write HTML code in cells and render it

```
%%HTML
<h>Hello world</h1>
```

%%javascript — Write JavaScript code in cells



%%bash cell — Write bash commands

```
%%bash
echo "Hello world!"
Hello, World!
```

Using SQL Magic

- Install ipython-sql by running the following statement:

```
!pip install --user ipython-sql
```

- Enable the SQL magic in Juypter notebook using this statement:

```
%load_ext sql
```

%SQL or %%SQL to run SQL

Using SQL Magic with SQLite Database

```
import sqlite3
conn = sqlite3.connection('HR.db')

%load_ext sql

%sql sqlite:///HR.db

%sql SELECT * FROM Employee
```

ใช้ SQL magic ได้เลย
ไม่ต้องไปเก็บในตัวแปรอยู่

Analyzing data with python

McDonald's nutrition fact dataset from kaggle (used for Analysis)

Load CSV to SQLite3 with Pandas

```
import pandas as pd
import sqlite3
data = pd.read_csv('./menu.csv')
conn = sqlite3.connect('McDonalds.db')
data.to_sql('MCDONALDS_NUTRITION', conn)
```

table name

Using pandas

```
df = pd.read_sql("SELECT * FROM MCDONALDS_NUTRITION", conn)
print(df)
```

Out[23]:

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fibe
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	120	13.0	20	5.0	25	0.0	...	31	10	4
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	70	8.0	12	3.0	15	0.0	...	30	10	4
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	200	23.0	35	8.0	42	0.0	...	29	10	4
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	250	28.0	43	10.0	52	0.0	...	30	10	4
4	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	400	210	23.0	35	8.0	42	0.0	...	30	10	4

View first few rows

```
df1.head()
```

← now pandas : a row view like screen

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Carbohydrates (% Daily Value)	Dietary Fiber	Dietary Fiber (% Daily Value)	Si
0	Breakfast	Egg McMuffin	4.8 oz (136 g)	300	120	13.0	20	5.0	25	0.0	...	31	10	4	17	3
1	Breakfast	Egg White Delight	4.8 oz (135 g)	250	70	8.0	12	3.0	15	0.0	...	30	10	4	17	3
2	Breakfast	Sausage McMuffin	3.9 oz (111 g)	370	200	23.0	35	8.0	42	0.0	...	29	10	4	17	2
3	Breakfast	Sausage McMuffin with Egg	5.7 oz (161 g)	450	250	28.0	43	10.0	52	0.0	...	30	10	4	17	2
		Sausage														

Learn about your data

```
df.describe(include = 'all')
```

Out[34]:

	Category	Item	Serving Size	Calories	Calories from Fat	Total Fat	Total Fat (% Daily Value)	Saturated Fat	Saturated Fat (% Daily Value)	Trans Fat	...	Carbohydrates	Ca (% Va
count	260	260	260	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	260.000000	...	260.000000	26
unique	9	260	107	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
top	Coffee & Tea	Nonfat Caramel Mocha (Large)	16 fl oz cup	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
freq	95	1	45	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	Na
mean	NaN	NaN	NaN	368.269231	127.096154	14.165385	21.815385	6.007692	29.965385	0.203846	...	47.346154	15
std	NaN	NaN	NaN	240.269886	127.875914	14.205998	21.885199	5.321873	26.639209	0.429133	...	28.252232	9.4
min	NaN	NaN	NaN	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.0
25%	NaN	NaN	NaN	210.000000	20.000000	2.375000	3.750000	1.000000	4.750000	0.000000	...	30.000000	10

show it all, then we can go more further

Which food item has maximum sodium content?

What we need to see? Identify req. first!

- Main source of sodium is table salt
- Average American eats 5 teaspoons/day
- Sodium mostly added during preparation
- Foods that don't taste salty may be high in sodium
- Sodium controls fluid balance in our bodies
- Too much sodium may raise blood pressure
- Target less than 2,000 milligrams/day

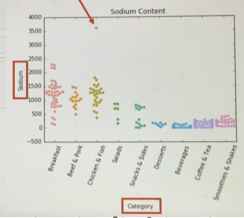
Which food item has maximum sodium content?

Notice this high sodium value

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

### Categorical scatterplots

plot = sns.swarmplot(x="Category", y='Sodium', data=df)
plt.setp(plot.get_xticklabels(), rotation=70)
plt.title('Sodium Content')
plt.show()
```



visualization

scatter plot

Which food item has maximum sodium content?

Code 1

check, which food has max sodium

```
In [17]: df['Sodium'].describe()

Out[17]: count     260.000000
         mean      495.750000
         std       577.026323
         min         0.000000
         25%       107.500000
         50%       190.000000
         75%       865.000000
         max      3600.000000
         Name: Sodium, dtype: float64
```

Code 2

Identify index value by idxmax()

```
In [24]: df['Sodium'].idxmax()

Out[24]: 82
```

Code 3

use index, to locate the item in df.

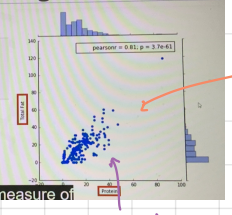
```
In [56]: df.at[82,'Item']

Out[56]: 'Chicken McNuggets (40 piece)'
```

Further data exploration using visualizations

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.jointplot(x="Protein", y='Total Fat', data=df)
plot.show()
```



called 'scatter plot'

Correlation is a measure of

intend to go straight line in positive direction

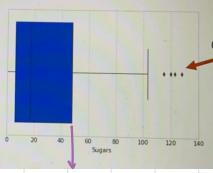
Box plot

Further data exploration using visualizations

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

plot = sns.set_style("whitegrid")
ax = sns.boxplot(x=df["Sugars"])
plot.show()
```

as input



avg sugar in food

food items that have sugar around 128 g, so they are outliers here

candy is among them

IBM Db2

Connecting to a database using ibm_db API (connect with Python)

Identify database connection credentials

```
dsn_driver = "{IBM DB2 ODBC DRIVER}"
dsn_database = "BLUDB"            # e.g. "BLUDB"
dsn_hostname = "YourDb2Hostname" # e.g.: "dashdb-txn-sbox-yp-dal09-04.services.dal.bluemix.net"
dsn_port = "50000"                # e.g. "50000"
dsn_protocol = "TCPIP"            # i.e. "TCPIP"
dsn_uid = "***************"        # e.g. "abc12345"
dsn_pwd = "***************"      # e.g. "7dBZ3wWt9XN6$o0J"
```

Create a database connection

```
#Create database connection
dsn = (
    "DRIVER={{IBM DB2 ODBC DRIVER}};"
    "DATABASE={0};"
    "HOSTNAME={1};"
    "PORT={2};"
    "PROTOCOL=TCPIP;"
    "UID={3};"
    "PWD={4};").format(dsn_database, dsn_hostname, dsn_port, dsn_uid, dsn_pwd)

try:
    conn = ibm_db.connect(dsn, "", "")
    print ("Connected!")

except:
    print ("Unable to connect to database")
```

```
Connected!
```

Creating tables

Serial No	Model	Manufacturer	Engine Size	Class
A1234	Lonestar	International Trucks	Cummins ISX15	Class 8
B5432	Volvo VN	Volvo Trucks	Volvo D11	Heavy Duty Tractor Class 8
C5674	Kenworth W900	Kenworth Truck Company	Caterpillar C9	Class 8

ibm_db.exec_immediate()

The parameters for the function are:

- Connection → to valid database
- Statement
- Options → Dic of type of cursor

Python code to create a table

```
stmt = ibm_db.exec_immediate(conn,
"CREATE TABLE Trucks(
serial_no varchar(20) PRIMARY KEY NOT NULL,
model VARCHAR(20) NOT NULL,
manufacturer VARCHAR(20) NOT NULL,
Engine_size VARCHAR(20) NOT NULL,
Truck_Class VARCHAR(20) NOT NULL) "
)
```

→ Created Trucks's table

Python code to insert data into the table

```
stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no,
model,manufacturer,Engine_size, Truck_Class)
VALUES('A1234','Lonestar','International
Trucks','Cummins ISX15','Class 8');")
```

→ first : connection resource

SQL statement : to add value in table

Insert more rows to the table

```
stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no,model,manufacturer,Engine_size, Truck_Class)
VALUES('B5432','Volvo VN','Volvo Trucks','Volvo D11','Heavy Duty Class 8');")

stmt = ibm_db.exec_immediate(conn,
"INSERT INTO Trucks(serial_no,model,manufacturer,Engine_size, Truck_Class)
VALUES('C5674','Kenworth W900','Kenworth Truck Co','Caterpillar C9','Class 8');")
```

Python code to query data

```
In [10]: stmt = ibm_db.exec_immediate(conn,"SELECT * FROM Trucks")

         ibm_db.fetch_both(stmt)

Out[10]: {0: 'A1234',
          1: 'Lonestar',
          'MANUFACTURER': 'International Trucks',
          3: 'Cummins ISX15',
          'SERIAL_NO': 'A1234',
          'ENGINE_SIZE': 'Cummins ISX15',
          'MODEL': 'Lonestar',
          'TRUCK_CLASS': 'Class 8',
          2: 'International Trucks',
```

→ Fetch Table

Fetch Data from DB2 Table (Trucks)

Using pandas

```
In [19]: import pandas
         import ibm_db_dbi
         pconn = ibm_db_dbi.Connection(conn)
         df = pandas.read_sql('SELECT * FROM Trucks', pconn)
         df
```

→ conn

Out[19]:

	SERIAL_NO	MODEL	MANUFACTURER	ENGINE_SIZE	TRUCK_CLASS
0	A1234	Lonestar	International Trucks	Cummins ISX15	Class 8
1	B5432	Volvo VN	Volvo Trucks	Volvo D11	Heavy Duty Class 8
2	C5674	Kenworth W900	Kenworth Truck Co	Caterpillar C9	Class 8