

INTELLIGENT SYSTEMS

(Part Two – Web Services with Database access using Python)

Version: 0.1 (20181028T2218)

Author:

Ms Juan Antonio Castro Silva (juan.castro@usco.edu.co)

Ph.D Diego Hernán Peluffo Ordoñez

Objective:

In this second part of the course (tutorial), we are going to create two databases, one NoSQL (MongoDB) and a Relational Database (PostgreSQL). To access the databases we will use the Python programming language. Also, we will build Web Services using the Flask Microframework to list the contents of the databases.

Software Requirements:

Bellow, there is the list of the required software to build and run Intelligent Web Application.

- Python 3.5 +
- Spyder IDE (Python)
- Flask Microframework 1.0.2
- MongoDB (NoSQL)
- PostgreSQL (Object-Relational Database)

Download the iris dataset

Download the iris dataset and save the file as iris.csv.

<https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data>

Note: In the github repository (<https://github.com/juancasi/yachay>) you will find all the source files and the iris dataset file.

1 Create the database

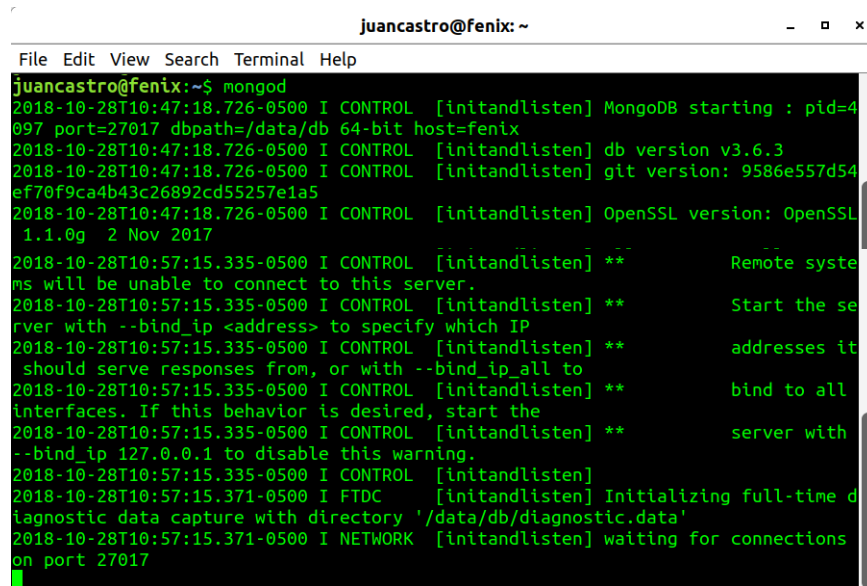
The process of database creation is:

1. Create a login role.
2. Create a database.
3. Create the table(s).
4. Import the data.

1.1 NoSQL database

Run the MongoDB server

To create a database in MongoDB, use the mongod command in a terminal to start (run) the server.

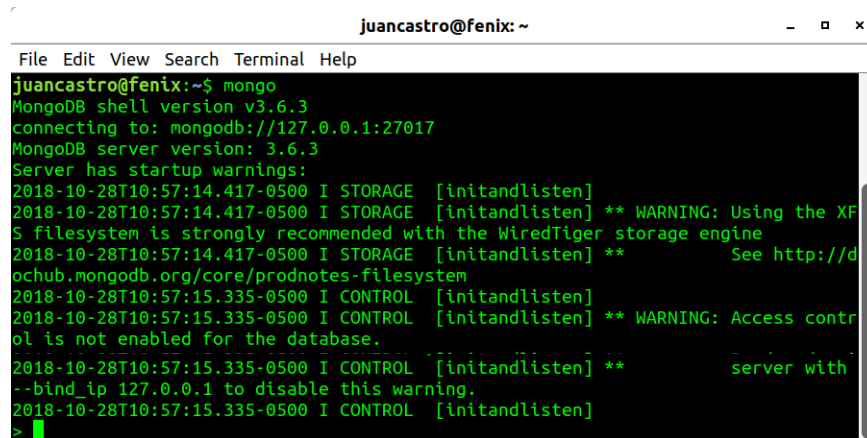


```
juancastro@fenix: ~  
File Edit View Search Terminal Help  
juancastro@fenix:~$ mongod  
2018-10-28T10:47:18.726-0500 I CONTROL [initandlisten] MongoDB starting : pid=4097 port=27017 dbpath=/data/db 64-bit host=fenix  
2018-10-28T10:47:18.726-0500 I CONTROL [initandlisten] db version v3.6.3  
2018-10-28T10:47:18.726-0500 I CONTROL [initandlisten] git version: 9586e557d54ef70f9ca4b43c26892cd55257e1a5  
2018-10-28T10:47:18.726-0500 I CONTROL [initandlisten] OpenSSL version: OpenSSL 1.1.0g 2 Nov 2017  
2018-10-28T10:57:15.335-0500 I CONTROL [initandlisten] ** Remote system will be unable to connect to this server.  
2018-10-28T10:57:15.335-0500 I CONTROL [initandlisten] ** Start the server with --bind_ip <address> to specify which IP addresses it should serve responses from, or with --bind_ip_all to bind to all interfaces. If this behavior is desired, start the server with --bind_ip 127.0.0.1 to disable this warning.  
2018-10-28T10:57:15.335-0500 I CONTROL [initandlisten]  
2018-10-28T10:57:15.371-0500 I FTDC [initandlisten] Initializing full-time diagnostic data capture with directory '/data/db/diagnostic.data'  
2018-10-28T10:57:15.371-0500 I NETWORK [initandlisten] waiting for connections on port 27017
```

This terminal window must be kept open. If you close this console, you kill the MongoDB server process (stops the server).

Run the MongoDB client

To run the MongoDB client, open a new terminal window and execute the mongo command.

A terminal window titled 'juancastro@fenix: ~' with a menu bar (File, Edit, View, Search, Terminal, Help). The terminal shows the command 'mongo' being executed. The output includes: 'MongoDB shell version v3.6.3', 'connecting to: mongodb://127.0.0.1:27017', 'MongoDB server version: 3.6.3', and several startup warnings from the server, including warnings about the XFS filesystem and access control. The prompt '>' is visible at the bottom.

```
juancastro@fenix: ~  
File Edit View Search Terminal Help  
juancastro@fenix:~$ mongo  
MongoDB shell version v3.6.3  
connecting to: mongodb://127.0.0.1:27017  
MongoDB server version: 3.6.3  
Server has startup warnings:  
2018-10-28T10:57:14.417-0500 I STORAGE [initandlisten]  
2018-10-28T10:57:14.417-0500 I STORAGE [initandlisten] ** WARNING: Using the XFS  
filesystem is strongly recommended with the WiredTiger storage engine  
2018-10-28T10:57:14.417-0500 I STORAGE [initandlisten] ** See http://d  
ochub.mongodb.org/core/prodnotes-filesystem  
2018-10-28T10:57:15.335-0500 I CONTROL [initandlisten]  
2018-10-28T10:57:15.335-0500 I CONTROL [initandlisten] ** WARNING: Access contr  
ol is not enabled for the database.  
2018-10-28T10:57:15.335-0500 I CONTROL [initandlisten] ** server with  
--bind_ip 127.0.0.1 to disable this warning.  
2018-10-28T10:57:15.335-0500 I CONTROL [initandlisten]  
>
```

1.1.1 Create the database

To create a MongoDB database employ the command `use database name`. If the database does not exist, it is created, else it is set as the default database (the database in use).

```
use iris
```

You will see.

A terminal snippet showing the command 'use iris' being entered, followed by the output 'switched to db iris' and a new prompt '>'.

```
> use iris  
switched to db iris  
>
```

1.1.2 Create a collection

In MongoDB, a collection is a set of documents, which is similar to a table in a relational database. To create a collection use the command `db.createCollection("collection_name")`. The keyword `db` is the database in use (iris).

```
db.createCollection("iris")
```

The server must return a `{"ok":1}` message.

A terminal snippet showing the command 'db.createCollection("iris")' being entered, followed by the output '{ "ok" : 1 }' and a new prompt '>'.

```
> db.createCollection("iris")  
{ "ok" : 1 }  
>
```

1.1.3 Import data from the iris.csv file

To import the data from the iris csv file use the command mongoimport.

```
mongoimport -d iris -c iris --type csv --file /home/user/datasets/iris.csv --fields "sepal_length,sepal_width,petal_length,petal_width,category"
```

The terminal window will be something similar to this.

```
juancastro@fenix:~$ mongoimport -d iris -c iris --type csv --file /home/juancastro/minerias/datasets/iris.csv --fields "sepal_length,sepal_width,petal_length,petal_width,category"
2018-10-28T17:10:07.730-0500    connected to: localhost
2018-10-28T17:10:07.792-0500    imported 150 documents
juancastro@fenix:~$
```

1.1.4 See the records of the iris table

To see all the documents of the iris collection execute the db.collection_name.find() command:

```
db.iris.find()
```

```
juancastro@fenix: ~
File Edit View Search Terminal Help
> db.iris.find()
{ "_id" : ObjectId("5bd633bf4ab681e5455775c2"), "sepal_length" : 4.6, "sepal_width" : 3.1, "petal_length" : 1.5, "petal_width" : 0.2, "category" : "Iris-setosa" }
{ "_id" : ObjectId("5bd633bf4ab681e5455775c3"), "sepal_length" : 4.4, "sepal_width" : 2.9, "petal_length" : 1.4, "petal_width" : 0.2, "category" : "Iris-setosa" }
{ "_id" : ObjectId("5bd633bf4ab681e5455775c4"), "sepal_length" : 4.9, "sepal_width" : 3.1, "petal_length" : 1.5, "petal_width" : 0.1, "category" : "Iris-setosa" }
{ "_id" : ObjectId("5bd633bf4ab681e5455775c5"), "sepal_length" : 5.4, "sepal_width" : 3.7, "petal_length" : 1.5, "petal_width" : 0.2, "category" : "Iris-setosa" }
{ "_id" : ObjectId("5bd633bf4ab681e5455775c6"), "sepal_length" : 4.8, "sepal_width" : 3.4, "petal_length" : 1.6, "petal_width" : 0.2, "category" : "Iris-setosa" }
{ "_id" : ObjectId("5bd633bf4ab681e5455775c7"), "sepal_length" : 4.8, "sepal_width" : 3, "petal_length" : 1.4, "petal_width" : 0.1, "category" : "Iris-setosa" }
```

1.1.5 Create a user

To create a role in MongoDB use the command db.createUser().

```
db.createUser({user: "iris", pwd: "iris", roles: ["readWrite", "dbAdmin"]})
```

The response message from the mongo server is [Successfully added user].

```
> db.createUser({user: "iris", pwd: "iris", roles: ["readWrite", "dbAdmin"]})
Successfully added user: { "user" : "iris", "roles" : [ "readWrite", "dbAdmin" ] }
>
```

1.2 PostgreSQL Database

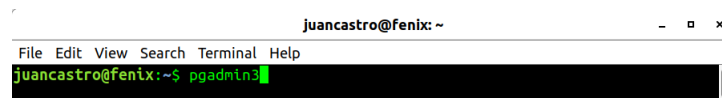
To create the database in PostgreSQL use pgadmin3 (or superior).

pgAdmin 3

pgAdmin 3 is the most popular and feature rich Open Source administration and development platform for PostgreSQL, the most advanced Open Source database in the world. The application may be used on Linux, FreeBSD, Solaris, macOS and Windows platforms to manage PostgreSQL 9.2 and above running on any platform, as well as commercial and derived versions of PostgreSQL such as EDB Postgres Advanced Server.

[<https://www.pgadmin.org/download/>]

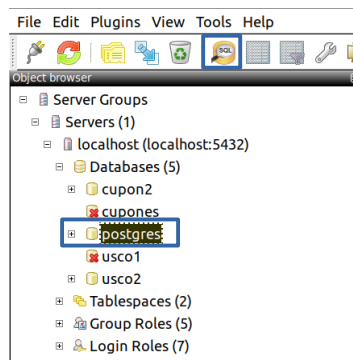
To launch or open the PostgreSQL administration and development platform, write pgadmin3 in a terminal (console-shell) and press the Enter key.

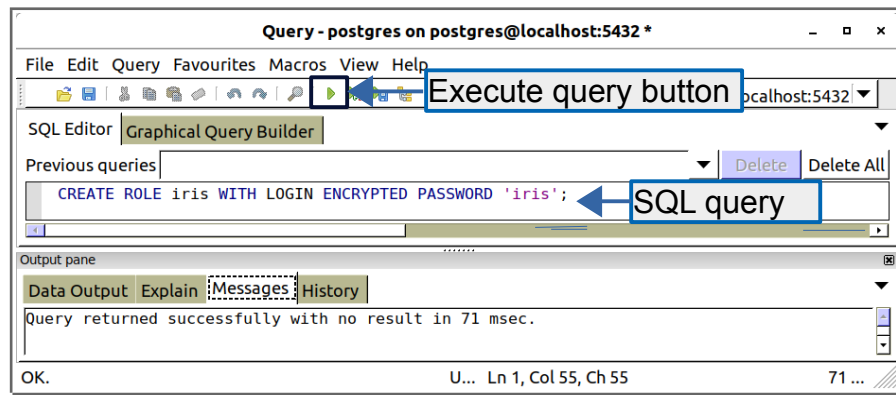


1.2.1 Executing SQL queries



To execute SQL queries in pgAdmin, select the postgres database and click de SQL button to open the SQL Editor.





In the SQL Editor write the query and click the Execute query button.

1.2.1 Create a login role

To create a login role execute the CREATE ROLE query:

```
CREATE ROLE iris WITH LOGIN ENCRYPTED PASSWORD 'iris';
```

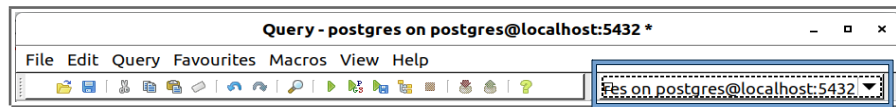
1.2.2 Create a database

To create a database execute the CREATE DATABASE query:

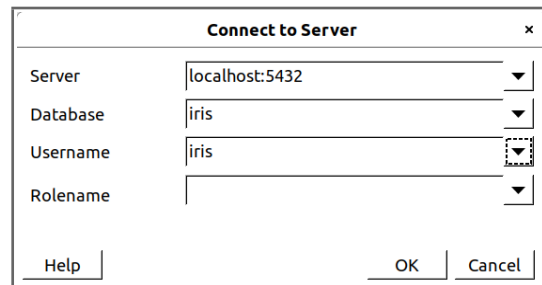
```
CREATE DATABASE iris
WITH OWNER = iris
ENCODING = 'UTF8'
TABLESPACE = pg_default
LC_COLLATE = 'en_US.UTF-8'
LC_CTYPE = 'en_US.UTF-8'
CONNECTION LIMIT = -1;
```

1.2.3 Create a table

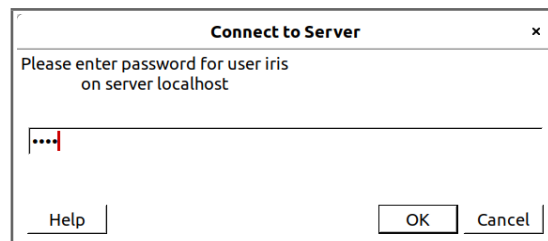
To create a table, change the database connection to iris, click the menu button and select the <new connection> option.



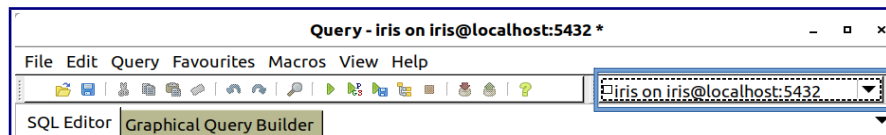
In the connection window select the iris database and the iris username, then click the OK button.



Enter the password for user iris and click the OK button.



After that you will see the iris user connected to the iris database:



To create a table execute the CREATE TABLE query:

```
CREATE TABLE public.iris
(
    id bigserial NOT NULL,
    sepal_length double precision,
    sepal_width double precision,
    petal_length double precision,
    petal_width double precision,
    category character varying(100),
    CONSTRAINT iris_pkey PRIMARY KEY (id)
)
WITH (
    OIDS=FALSE
);
ALTER TABLE public.iris
    OWNER TO iris;
```

1.2.4 Copying the iris dataset in the iris database table

To copy the data from the iris dataset csv file, use the COPY command in the SQL Editor.

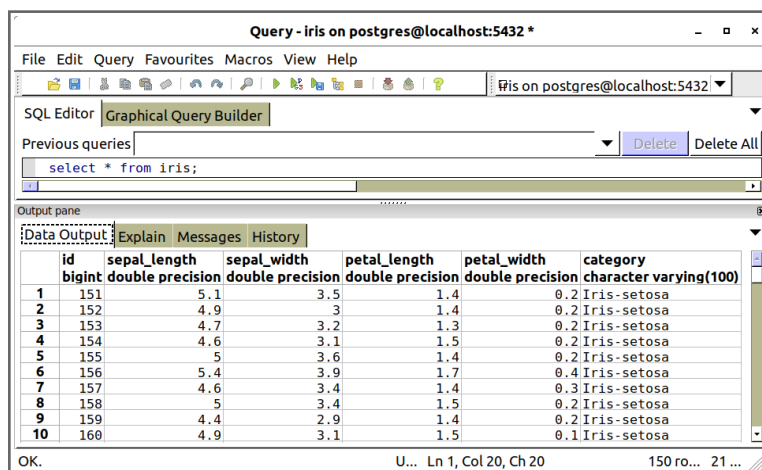
```
COPY iris(sepal_length,sepal_width,petal_length,petal_width,category)
FROM '/home/user/datasets/iris.csv' DELIMITER ',';
```

Change the path of your iris.csv file.

1.2.5 See the records of the iris table

To see all the records (rows) of the iris table execute the SELECT command:

```
SELECT * from iris;
```



The screenshot shows a PostgreSQL SQL Editor window titled "Query - iris on postgres@localhost:5432 *". The SQL Editor tab is active, showing the query "select * from iris;". The Output pane is open, displaying the results of the query in a table format. The table has 7 columns: id, sepal_length, sepal_width, petal_length, petal_width, and category. The data is as follows:

	id	sepal_length	sepal_width	petal_length	petal_width	category
	bigint	double precision	double precision	double precision	double precision	character varying(100)
1	151	5.1	3.5	1.4	0.2	Iris-setosa
2	152	4.9	3	1.4	0.2	Iris-setosa
3	153	4.7	3.2	1.3	0.2	Iris-setosa
4	154	4.6	3.1	1.5	0.2	Iris-setosa
5	155	5	3.6	1.4	0.2	Iris-setosa
6	156	5.4	3.9	1.7	0.4	Iris-setosa
7	157	4.6	3.4	1.4	0.3	Iris-setosa
8	158	5	3.4	1.5	0.2	Iris-setosa
9	159	4.4	2.9	1.4	0.2	Iris-setosa
10	160	4.9	3.1	1.5	0.1	Iris-setosa

1 Database access using Python

2.1 Required Software – Libraries

2.1.1 MongoDB

PyMongo is a Python distribution containing tools for working with [MongoDB](#), and is the recommended way to work with MongoDB from Python [1], [2].

It is recommended using [pip](#) to install pymongo on all platforms:

```
pip install pymongo
```

2.1.2 PostgreSQL

Psycopg

Psycopg is the most popular PostgreSQL database adapter for the Python programming language. Its main features are the complete implementation of the Python DB API 2.0 specification and the thread safety (several threads can share the same connection). It was designed for heavily multi-threaded applications that create and destroy lots of cursors and make a large number of concurrent “INSERT”s or “UPDATE”s [3].

You can install psycopg like any other Python package, using `pip` to download it from [PyPI](#):

```
pip install psycopg2
```

SQLAlchemy

To connect to the PostgreSQL database use SQLAlchemy.

SQLAlchemy is the Python SQL toolkit and Object Relational Mapper that gives application developers the full power and flexibility of SQL.

It provides a full suite of well known enterprise-level persistence patterns, designed for efficient and high-performing database access, adapted into a simple and Pythonic domain language [4].

When `pip` is available, the distribution can be downloaded from Pypi and installed in one step:

```
pip install SQLAlchemy
```

2.1.3 Flask Microframework

The flask microframework can be installed from a terminal with this command:

```
pip install Flask
```

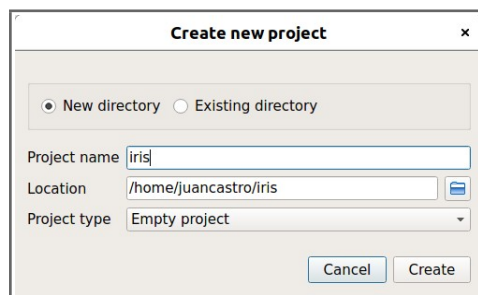
2.2 Flask test project

To implement the Web Services in python we will create a project named iris using the Spyder IDE.

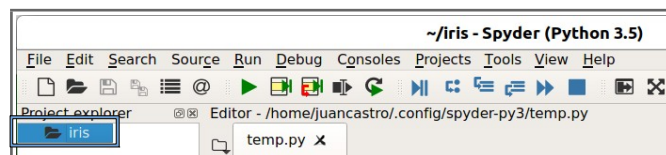
To create a project in the Spyder IDE click the [Projects] menu and select the [new project] option.



In the Create new project window write the Project name (iris) and click the [Create] button.



Click with the right button of the mouse on the iris project, select the [New] menu and click in the [Package...] option.

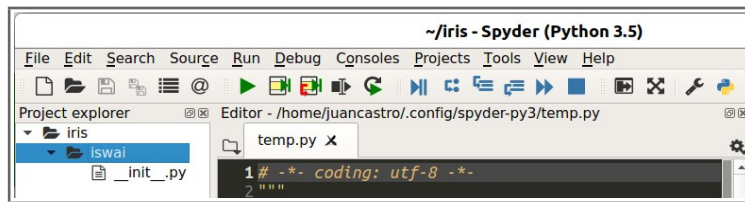


In the New package window write the Package name (iswai) and click the [OK] button.



ISWAI means Information System With Artificial Intelligence, write the package names in lower case.

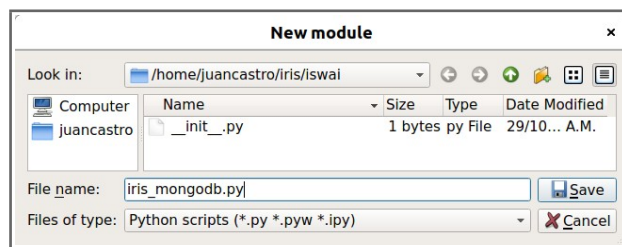
Right click the iswai package, select the [New] menu and click the [Module..] option.



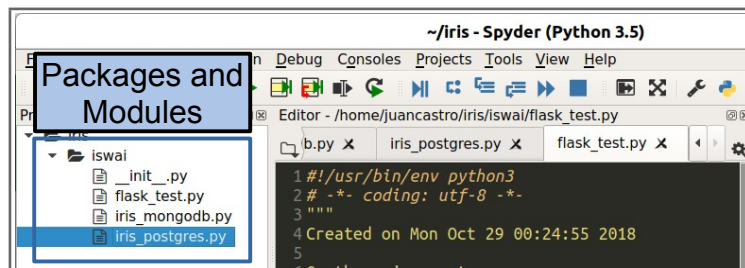
Create 3 modules with these filenames:

1. iris_mongodb.py
2. iris_postgres.py
3. flask_test.py

In the New module window write the File name (iris_mongodb.py) and click the [Save] button.



The package iswai have 3 modules:



The file contents are as follows.

2.2.1 IrisMongoDB

The IrisMongoDB class implement a method getConnection() that returns a connection to the MongoDB database (iris). To connect to the MongoDB database we will use PyMongo. The method getDataframe() returns a pandas dataframe including all the documents of the iris collection.

File: iris_mongodb.py

```
from pymongo import MongoClient
import pandas as pd

class IrisMongoDB:
    def getConnection(self):
        client = MongoClient()
        db = client['iris']
        collection = db['iris']
        return collection

    def getDataframe(self):
        collection = self.getConnection()
        cursor = collection.find({}, {'_id':0})
        dataframe = pd.DataFrame(list(cursor))
        print(dataframe.head(5))
        return dataframe
```

2.2.2 IrisPostgres Class

The IrisPostgres class implement a method getConnection() that returns a connection to the PostgreSQL database (iris). To connect to the PostgreSQL database we will use SQLAlchemy. The method getDataframe() returns a pandas dataframe including all the records of the iris table.

```
from sqlalchemy import create_engine
from sqlalchemy import text
from pandas import DataFrame

class IrisPostgres:
    def getConnection(self):
        # create connection
        engine = create_engine('postgresql://iris:iris@localhost:5432/iris')
        return engine

    def getDataframe(self):
        # execute query
        sql = text('select sepal_length, sepal_width, petal_length, petal_width, category from iris')
        engine = self.getConnection()
        result = engine.execute(sql)
        # convert sqlalchemy.engine.result to pandas dataframe
        dataframe = DataFrame(result.fetchall())
        dataframe.columns = result.keys()
        print(dataframe.head(5))
        return dataframe
```

2.2.3 Web Services with Flask

To implement Web Services with Python use the Flask Microframework. The Flask application includes 3 Web Services:

1. A hello world example (/) that return a JSON object.
2. A Web Service (/mongodb) the returns all documents from the iris collection.
3. A Web Service (/postgres) that returns all records from the iris table.

File: flask_test.py

```
from flask import Flask, jsonify
from iswai.iris_mongodb import IrisMongoDB
from iswai.iris_postgres import IrisPostgres

app = Flask(__name__)

@app.route('/')
def hello_world():
    message = {'id':123,'name':'Flask test'}
    return jsonify(message)

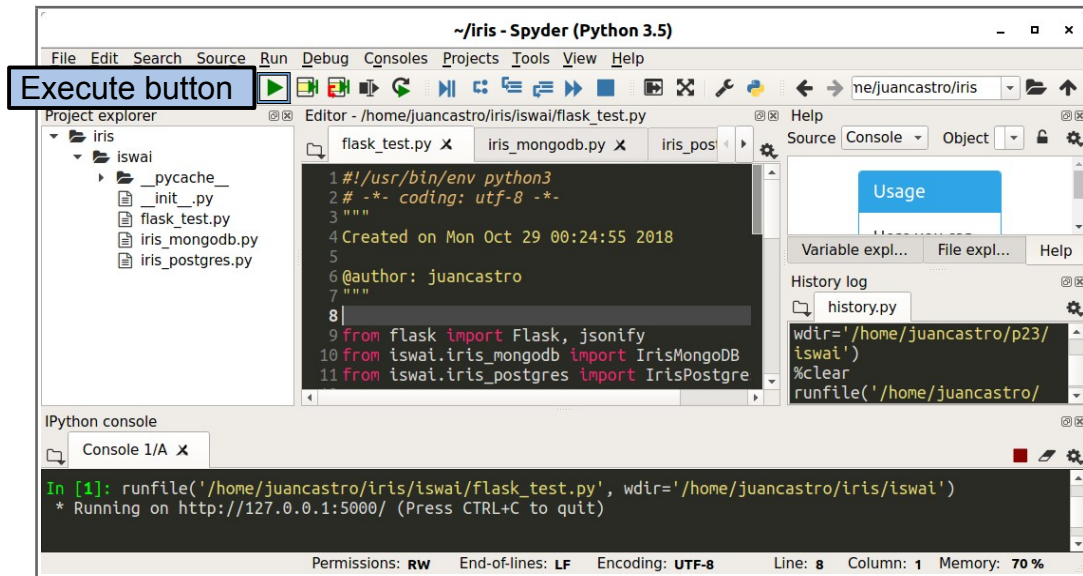
@app.route('/mongodb')
def mongodb():
    iris_mongodb = IrisMongoDB()
    dataframe = iris_mongodb.getDataframe()
    data_json = dataframe.to_json(orient='records')
    return jsonify(data_json)

@app.route('/postgres')
def postgres():
    iris_postgres = IrisPostgres()
    dataframe = iris_postgres.getDataframe()
    data_json = dataframe.to_json(orient='records')
    return jsonify(data_json)

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

2.3 Run the Flask server

To run the Flask server, open the flask_test.py file and click the execute button in the Spyder IDE.



To test the Flask server open the web service url in a browser:

<http://localhost:5000/>

or

<http://127.0.0.1.5000/>



This result corresponds to the JSON output from the `hello_world()` method:

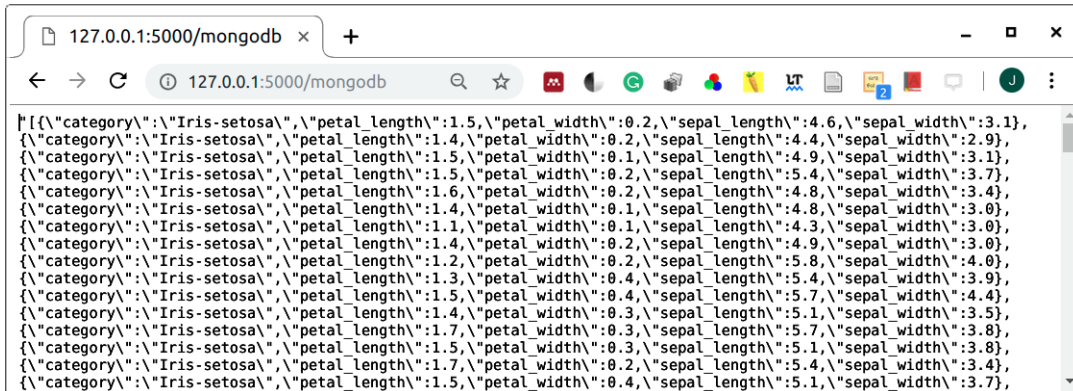
```
@app.route('/')
def hello_world():
    message = {'id':123,'name':'Flask test'}
    return jsonify(message)
```

MongoDB database access test

To test the Python access to the iris MongoDB database, in a browser open the url:

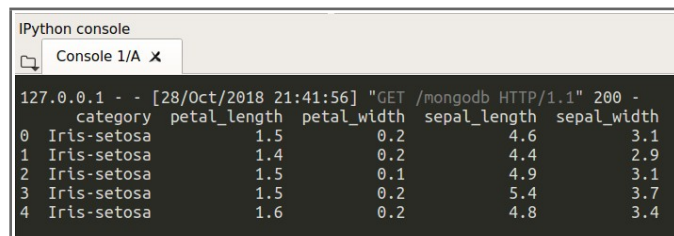
<http://127.0.0.1:5000/mongodb>

The Web Service /mongodb will return a JSON array object with all the documents in the iris collection.



The screenshot shows a web browser window with the address bar displaying `127.0.0.1:5000/mongodb`. The main content area displays a JSON array of 20 documents, all belonging to the 'Iris-setosa' category. Each document contains five fields: 'petal_length', 'petal_width', 'sepal_length', and 'sepal_width'. The values for these fields vary slightly across the documents, representing different individual flowers in the dataset.

In the Spyder IDE console, you must see the printing of the first 5 rows of the iris pandas dataframe [`print(dataframe.head(5))`].



The screenshot shows the IPython console in the Spyder IDE. The console output displays the first 5 rows of the iris pandas dataframe. The output is as follows:

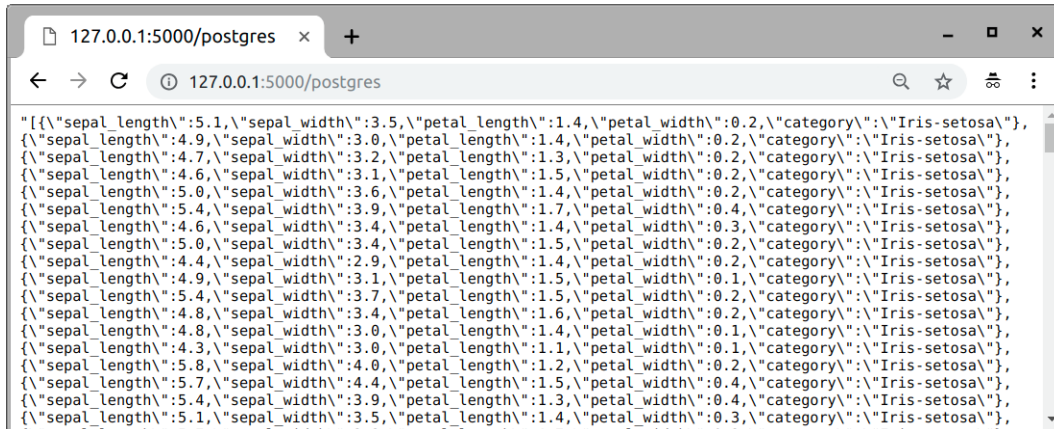
```
127.0.0.1 - - [28/Oct/2018 21:41:56] "GET /mongodb HTTP/1.1" 200 -
category petal_length petal_width sepal_length sepal_width
0 Iris-setosa 1.5 0.2 4.6 3.1
1 Iris-setosa 1.4 0.2 4.4 2.9
2 Iris-setosa 1.5 0.1 4.9 3.1
3 Iris-setosa 1.5 0.2 5.4 3.7
4 Iris-setosa 1.6 0.2 4.8 3.4
```


PostgreSQL database access test

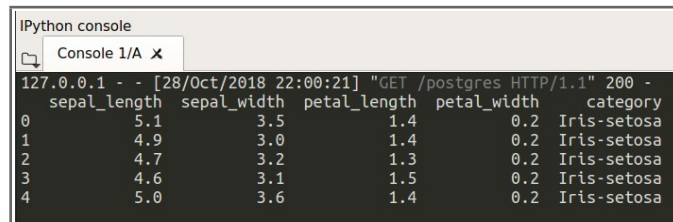
To test the Python access to the iris PostgreSQL database, in a browser open the url:

<http://127.0.0.1:5000/postgres>

The Web Service /postgres will return a JSON array object with all the records in the iris table.



In the Spyder IDE console, you must see the printing of the first 5 rows of the iris pandas dataframe `[print(dataframe.head(5))]`.



2 REFERENCES

- [1] “PyMongo 3.7.2 Documentation.” [Online]. Available: <https://api.mongodb.com/python/current/>. [Accessed: 28-Oct-2018].
- [2] “MongoDB Tutorial.” [Online]. Available: <http://api.mongodb.com/python/current/tutorial.html>. [Accessed: 28-Oct-2018].
- [3] “psycopg2 - Python-PostgreSQL Database Adapter.” [Online]. Available: <https://pypi.org/project/psycopg2/>. [Accessed: 28-Oct-2018].
- [4] “The Python SQL Toolkit and Object Relational Mapper.” [Online]. Available: <https://www.sqlalchemy.org/>. [Accessed: 28-Oct-2018].