Virtual Environments, Database interaction with Python

Fernando Pineda 140.636 Created: Oct. 09, 2018 version 0.2

1. Python virtual environments

When using modules or Python, there inevitably arises the problem of compatibility between the Python version you are running and the modules you are importing. On the JHPCE cluster there are several different version of Python, e.g. (2.6.1, 2.6.6, 2.7.6, 2.7.0, 3.4.0, 3.4.3, 3.6.3, etc.). You can use the module command to use different versions, e.g.

```
module load python/3.4.3
```

You may also want to have more up-to-date versions of modules than those that are installed on the cluster. Moreover, you may have a range of Python apps and projects that all need different versions of Python or different versions of specific modules. Or perhaps you have an Python project that you want to send to a collaborator, but your collaborator may be running a different version of Python and different versions of important modules, so you can't be sure that your project will work on their system.

It would be nice if we could create custom environments for individual projects and also be able to send around these virtual environments along with the project, so your application would run on another machine. This is the problem that is solved by virtual environments. Virtual environments create isolated Python environments for Python projects.

Here we will focus on using Python3 and doing things in a Python3 virtual environment. The documentation for creating virtual environments in Python3 can be found at https://docs.python.org/3/library/venv.html .

If you are running Python 3.x you already have the `virtualenv` application. If you are on the JHPCE cluster and using Python2 you should also already be installed, but if not, you can install it with `pip`:

```
pip install virtualenv
```

creating a virtual environment

In Python3 a virtual environment is created with the venv command. The command, if run with -h, will show the available options:

```
python3 -m venv -h
```

Let's create a demo virtual environment in your home directory and initialize it:

```
module load python/3.4.3         # Python version for this virtual environment
python3 -m venv ~/venvdemo       # creates file system for virtual environment
```

Inspect the contents of the directory that was created:

```
ls ~/venvdemo
bin  include  lib  lib64  pyvenv.cfg
```

1. The `bin` directory has executables for your enviroment, e.g. a link to the version of python that was used to create the environment, the pip and pip3 installers, and some utility scripts for working with virtual environments.
2. The `lib` directory has various core Python packages. It also has the `site-packages` subdirectory where dependencies will be installed by `pip`.
3. The `include` directory contains C header files for compiling the Python packages.
4. The `lib64` link points to the versions of system libraries needed by the particular version of python and it's modules.
5. Finally, the `pyvenv.cfg` file contains configuration information for the virtual environment.

To start using your virtual environment you have to "activate" it:

```
source ~/venvdemo/bin/activate
```

This sets up shell environment to use your environment

Henceforth, if you want to use your virtual environment (say if you log off and log back in or if you execute from a shell script) you will need to setup your shell by activating your virtual environment `source ~/venvdemo/bin/activate`.

You can always deactivate your virtual environment and return to the default Python environment by issuing a `deactivate` command at the shell prompt.

If you run the command with a -h option, you will see the available options:

```

```

Using and maintaining the virtual environment

Now whenever your virtual environment is activated, you will be using the Python that is in your virtual environment as well as the modules and packages that your installed when your environment as activated. Whenever you install a python3 module using `pip3` you will be installing the module into your virtual environment.

Virtual environments are nifty tools for managing dependencies in Python projects. You may want to create a virtual environment for each of major project.

We have only touched the surface. You will want to learn the fine points such as using the `--no-site-packages` option to prevent inclusion of any globally installed packages that on the system, or using `pip3 freeze` to create a list of requirements that can be used to recreate the package library (e.g. if you are distributing your project).

Finally, the `virtualenvwrapper` tool simplifies the maintenance of virtual environments. To learn more, refer to the web sites below.

References

1. http://docs.python-guide.org/en/latest/dev/virtualenvs/
2. https://realpython.com/blog/python/python-virtual-environments-a-primer/

2. Example: Interacting with relational databases from Python

sqlite3

1. create the walton database in sqlite

We will use the `walton` database as an example. Begin by copying the necessary files into a directory in your home directory and then create the database:

```
mkdir ~/walton
cd ~/walton
cp /users/sph140636/shared/code_examples/15_databases/walton_demo/*  . # copy files
```

There are three files that matter: `employee.txt`, `department.txt` and `walton_sqlite.sql`. The first two contain the data for the employee and department tables. The last one contains the SQL statements to create the tables from the text files. Next let's create the database:

```
sqlite3 walton.db < walton_sqlite.sql
```

This creates the database and saves it in the file `walton.db`. You should now be able query the walton database from the sqlite3 application:

```
sqlite3 walton.db
```

The `.schema` command will provide the schema. The `.quit` command will quit sqlite3. You can enter SQL queries, e.g. `select * from employee`.

2. Querying an sqlite3 database from Python

A module for connecting with an sqlite3 database comes with Python. Useful documentation for the database is here:

https://docs.python.org/2/library/sqlite3.html.

A very useful tutorial is here:

https://sebastianraschka.com/Articles/2014_sqlite_in_python_tutorial.html

Here we will simply select and print rows.

Briefly, to use it you just need to import the module and create a connection. Here is an example script (`sqlite3_demo.py` in the examples directory)

```
#!/usr/bin/env Python3
import sqlite3
conn = sqlite3.connect('walton.db')
cursor = conn.execute("SELECT EMPNO, ENAME, JOB FROM EMPLOYEE")
for row in cursor:
  print("{0} : {1}\t{2} ".format(row[0],row[1],row[2]))
```

MySQL

Interacting with MySQL in Python is very similar to doing it in Perl. We will use the `MySQLdb` module. `MySQLdb` is a Python wrapper around the `_mysql` Python module, which in turn implements the MySQL C API. The latter is an implementation of the Python DB API interface (version 2).

We begin by installing the MySQL DBI adapter into our virtual environment:

```
pip3 install pymysql
```

We test that the module has loaded properly with a Python one-liner:

```
python3 -c 'import pymysql'
```

Everything is fine if this returns no errors.

Next let's try connecting to a mysql server and perform a simple query. We will connect to the public MySQL server at useast.ensembl.org. (see e.g. http://useast.ensembl.org/info/data/mysql.htmlwith)

Use the following script:

```
#!/usr/bin/env python3

import pymysql

# connect to the database
db = pymysql.connect(host="ensembldb.ensembl.org", # host
                     user="anonymous",             # username
                     passwd="",                    # password
                     db="")                        # database

# create a Cursor object to execute queries
cursor = db.cursor()

# Execute SQL statements with cur.execute() and then fetch each
# row of the result with cur.fetchall()

cursor.execute("SHOW DATABASES")

# print all the first field of each row
for row in cursor.fetchall():
    print(row[0])
```

```
db.close()
```

This will produce a list of the databases at the ensemble mirror.

References

1. http://mysql-python.sourceforge.net/MySQLdb.html
2. https://www.python.org/dev/peps/pep-0249/
3. http://zetcode.com/db/mysqlpython/

ORM with peewee

To begin we install the `peewee` modules into our virtual environment.

```
pip install peewee
```

The `peewee` module is a simple and light-weight Python module for ORM. `peewee` includes the `pwiz` module for reverse-engineering database schema and code generation, for creating Python ORM objects. Try this:

```
python3 -m pwiz --engine=sqlite walton.db
```

You will see Python code stream by. It has one class definition for each table in the walton database. This code can be pasted into a Python script. Consider for example the following script

```
python3 -m pwiz --engine=sqlite walton.db > orm_example.py
```

This creates a script template `orm_example.py`. We can edit it and add code to the end.

```
#!/usr/bin/env python3

# -- this code is automatically generated by the following comamnd:
# python -m pwiz --engine=sqlite walton.db

from peewee import *

database = SqliteDatabase('walton.db', **{})

class UnknownField(object):
    def __init__(self, *_, **__): pass

class BaseModel(Model):
    class Meta:
        database = database

class Department(BaseModel):
    dname = CharField(db_column='DNAME')
    id = PrimaryKeyField(db_column='ID', null=True)
```

```
    loc = CharField(db_column='LOC')

    class Meta:
        db_table = 'DEPARTMENT'

class Employee(BaseModel):
    comm = IntegerField(db_column='COMM', null=True)
    deptid = ForeignKeyField(db_column='DEPTID', rel_model=Department, to_field='id')
    empno = PrimaryKeyField(db_column='EMPNO', null=True)
    ename = CharField(db_column='ENAME')
    hiredate = CharField(db_column='HIREDATE')
    job = CharField(db_column='JOB')
    mgr = ForeignKeyField(db_column='MGR', rel_model='self', to_field='empno')
    sal = IntegerField(db_column='SAL')

    class Meta:
        db_table = 'EMPLOYEE'

# -- end of automatically generated code

# now connect to the database and perform a few queries

database.connect()

# dump the department table
print("The department table")
for dept in Department.select() :
    print(dept.dname)

# select from the employee table
print("\n\n")´
print("employees and their hire dates")
for employee in Employee.select() :
    print("person =",employee.ename, ", hiredate =",employee.hiredate)
```

References

1. https://sqlite.org/docs.html
2. http://docs.peewee-orm.com/en/latest/