learning_peewee_database

February 15, 2018

1 Meet Peewee, A ORM (Object Relational Mapping)

- Peewee's docs
- Connecting to PostgreSQL
- Connecting to MySQL
- Database Foundations
- Charles Leifer's tutorial

1.0.1 Notes:

- The Python class in the Peewee ORM that represents tables in SQL databases are called *Models*.
- Each column or attribute in a table is an attribute of the model class.
- Since ORMs usually sanitize the queries they run against your database, a common benefit of using an ORM is that you're protected against malicious queries.
- The db extension is not required, its just a convention used in peewee to signify a file is a database.
- Notice that our class name is singular and not plural. This is because classes in Peewee represent a single item in a database. This convention is also used heacily in Jango.

1.1 Importing Peewee

```
In [5]: from peewee import *
```

2 Modeling

- `model` - A code object that represents a database table
- `SqliteDatabase` - The class from Peewee that lets us connect to an SQLite database
- `Model` - The Peewee class that we extend to make a model
- `CharField` - A Peewee field that holds onto characters. It's a varchar in SQL terms
- `max_length` - The maximum number of characters in a CharField
- `IntegerField` - A Peewee field that holds an integer
- `default` - A default value for the field if one isn't provided
- `unique` - Whether the value in the field can be repeated in the table
- `.connect()` - A database method that connects to the database
- `.create_tables()` - A database method to create the tables for the specified models.

- `safe` - Whether or not to throw errors if the table(s) you're attempting to create already exist
- `Meta` - This class tells the model class which database it belongs to. (this is NOT an actual Meta class, its just called one. BTW this convention is also seen in Jango)
  - This class can be used to specify which fields should be indexed.
  - How things should be ordered by default.
  - etc.

Notes: * `{0.username}` - The zero represents what is returned from the function, and the username is one of two attributes (the other being points) that it has, therefore when you add in `.username` your only asking for username attribute.

```
In [7]: #students.py
        from peewee import *

        db = SqliteDatabase('students.db')


        class Student(Model):
            username = CharField(max_length=255, unique=True)
            points = IntegerField(default=0)

            class Meta:
                database = db

        if __name__ == '__main__':
            db.connect()
            db.create_tables([Student], safe=True)
        #     add_students()
        #     print('Our top student right now is: {0.username}.'.format(top_student()))
```

2.1 A note on Peewee's import style:

Peewee's convention is to import everything with *. Why is this usually considered a bad practice?

- Your local namespace gets flooded by a huge number of items.
- Things you've defined locally, or already imported, can be overwritten by import.
- Peewee's contents are no longer contained in the peewee namespace.

2.2 Using Sqlite3 in Terminal

- `.tables` - Shows tables in database
- `select * from student;` - Will show the entire contents of student database

```
   sql sh-3.2# sqlite3 students.db SQLite version 3.13.0 2016-05-18 10:57:30
Enter ".help" for usage hints. sqlite> .tables student sqlite> select * from
student;  sqlite> .exit
```

2.3 Code Challenge

- Import everything from the peewee library.
- Now we need to make a database connection. Make an SqliteDatabase() named "challenges.db". Assign it to the variable db.
- Alright, now for the biggest part. Make a model named Challenge that has two fields, name and language. Both fields should be of the type CharField with a max_length of 100.
- Now add a Meta class to Challenge and set the database attribute equal to db.

```
from peewee import *

db = SqliteDatabase("challenges.db")

class Challenge(Model):
    name = CharField(max_length=100)
    language = CharField(max_length=100)

    class Meta:
        database = db
```

3 Queries are your Friend

- `.create()` - creates a new instance all at once
- `.select()` - finds records in a table
- `.save()` - updates an existing row in the database
- `.get()` - finds a single record in a table
- `.delete_instance()` - deletes a single record from the table
- `.order_by()` - specify how to sort the records
- `__name__` - a special variable that refers to the current namespace.
- `if __name__ == '__main__'` - a common pattern for making code only run when the script is run and not when it's imported.
- `db.close()` - not a method we used, but often a good idea. Explicitly closes the connection to the database.
- `.update()` - also something we didn't use. Offers a way to update a record without `.get()` and `.save()`. Example: `Student.update(points=student['points']).where(Student.username == student['username']).execute()`

Peewee Query Methods
Notes:

- The try block below allows for information about pupils already in the database to be updated while handling the `IntegrityError` caused by the `unique=True` argument in the username `CharField`.
- `Student.select()` - Grabs all data on each student
- `.order_by(Student.points.desc())` - Orders students by the amount of points they have, starting with the highest.
- `.get()` - Retrieves only the first record.

```
In [8]: # students.py continued

        students = [
            {'username': 'charlieTucker',
            'points': 934},
            {'username': 'Mwallace',
            'points': 30234},
            {'username': 'FantasticSam',
            'points': 26323},
            {'username': 'JuliePeaches',
            'points': 64890}
        ]

        def add_students():
            for pupil in students:
                try:
                    Student.create(username=pupil['username'],
                                   points=pupil['points'])
                except IntegrityError:
                    pupil_record = Student.get(username=pupil['username'])
                    pupil_record.points = pupil['points']
                    pupil_record.save()

        def top_student():
            student = Student.select().order_by(Student.points.desc()).get()
            return student
```

```
In [9]: ! python students.py
```

```
Our top student right now is: JuliePeaches.
```

If we simply change the values in the students list: `python students = [ {'username': 'charlieTucker', 'points': 934}, {'username': 'Mwallace', 'points': 30234}, {'username': 'FantasticSam', 'points': 26323}, {'username': 'JuliePeaches', 'points': 648934333632290} ]` And then rerun the script, the top student changes:

`! python students.py Our top student right now is: JuliePeaches.`

3.1 What's CRUD

- Create
- Read
- Update
- Delete

3.2 Code Challenge

- Import the Challenge class from models.py.
- Now, create a variable named all_challenges. It should select all of the available challenges from the database.
- Next, create a new Challenge. The language should be “Ruby”, the name should be “Booleans”.
- Finally, make a variable named sorted_challenges that is all of the Challenge records, ordered by the steps attribute on the model. The order should be ascending, which is the default direction.

```
from models import Challenge

all_challenges = Challenge.select()
Challenge.create(name='Booleans',
                 language='Ruby')

sorted_challenges = all_challenges.order_by(Challenge.steps)
```

4 Making a Diary w/ a Sqlite3 Database

- `TextField()` - a field that holds a blob of text of any size
- `DateTimeField()` - a field for holding a date and a time

`/usr/bin/env` what?

If you’re not sure what to put after `/usr/bin/env`, test it out in your terminal program.

Type in `/usr/bin/env python` and you should get a Python shell like normal. If it says 2.7 or something other than the 3.4 you should be expecting, try `/usr/bin/env python3`. Whichever of these gets you the correct Python shell is the one you should put at the top of your file.

Notes:

- Notice that the `datetime.datetime.now` is missing the parenthesis that typically follows after `.now`. This is because had we added the parenthesis, `datetime.datetime.now` would not have been seen as a function call, and instead would have recorded the timestamp when the script was initally run, rather than when the record or diary entry was created.

4.0.1 Using Switches

- `OrderedDict` - a handy container from the collections module that works like a dict but maintains the order that keys are added
- `.__doc__` - a magic variable that holds the docstring of a function, method, or class

4.0.2 Working with Python’s Sys Library

- `sys` - a Python module that contains functionality for interacting with the system
- `sys.stdin` - a Python object that represents the standard input stream. In most cases, this will be the keyboard

4.0.3 View and Seach Queries

- `.where()` - method that lets us filter our .select() results
- `.contains()` - method that specifies the input should be inside the specified field

4.0.4 Working with OS Library

- `os` - Python module that lets us integrate with the underlying OS
- `os.name` - attribute that holds a name for the style of OS
- `os.system()` - method to allow Python code to call OS-level programs

```
In [10]: #!/Users/lawerencelee/anaconda/bin/python
         from collections import OrderedDict
         import datetime
         import sys
         import os

         from peewee import *


         db = SqliteDatabase('diary.db')


         class Entry(Model):
             content = TextField()
             timestamp = DateTimeField(default=datetime.datetime.now)

             class Meta:
                 database = db

         def initalize():
             db.connect()
             db.create_tables([Entry], safe=True)

         def __clear():
             """
             Clears the terminal screen.
             """
             os.system("cls" if os.name == "nt" else "clear")

         def menu_loop():
             '''Show The Menu.'''
             choice = None

             while choice != 'q':
                 __clear()
                 print('Enter q to QUIT')
                 for key, value in menu.items():
                     print('{}) {}'.format(key, value.__doc__))
```

```
        choice = input('\nAction: ').lower().strip()

        if choice in menu:
            menu[choice]()

def add_entry():
    """Add an Entry."""
    __clear()
    print('Enter your entry. Press ctrl+d when finished.')
    data = sys.stdin.read().strip()

    if data:
        if input('\nSave entry? [Y/n]: ').lower() != 'n':
            Entry.create(content=data)
            input("\nSaved Successfully, press ENTER to Continue. ")

def view_entries(search_query=None):
    """View Previous Entries."""
    entries = Entry.select().order_by(Entry.timestamp.desc())
    if search_query:
        entries = entries.where(Entry.content.contains(search_query))

    for entry in entries:
        __clear()
        timestamp = entry.timestamp.strftime('%A %B %d, %Y %I:%M%p')
        print(timestamp)
        print('='*len(timestamp))
        print(entry.content, '\n\n')
        print('='*len(timestamp))
        print('n) Next Entry')
        print('d) Delete Entry')
        print('q) Main Menu')

        next_action = input('\nAction: [N/d/q] ').lower().strip()
        if next_action == 'q':
            break
        elif next_action == 'd':
            delete_entries(entry)

def search_entries():
    '''Search Via Keyword'''
    __clear()
    view_entries(input('Search query: '))

def delete_entries(entry):
    """Delete Entries"""
    if input('\nAre you sure? [y/N] ').lower() == 'y':
        entry.delete_instance()
```

```
                __clear()
                input('Entry Deleted, press Enter to Continue. ')

        menu = OrderedDict([
            ('a', add_entry),
            ('v', view_entries),
            ('s', search_entries)
        ])


        if __name__ == '__main__':
            initalize()
            menu_loop()
```

```
Enter q to QUIT
a) Add an Entry.
v) View Previous Entries.
s) Search Via Keyword

Action:
Enter q to QUIT
a) Add an Entry.
v) View Previous Entries.
s) Search Via Keyword

Action: q
```

After running the script, `diary.db` was created.

```
In [11]: ! ls
```

```
diary.db                          notebook.tex
diary.py                          students.db
learning_peewee_database.ipynb students.py
```

Using the shebang (#!/Users/lawerencelee/anaconda/bin/python) allowed us to do the following:

```
In [12]: ! ./diary.py
```

```
/bin/sh: ./diary.py: /Users/lawerencelee/anaconda/bin/python: bad interpreter: No such file or
```

4.0.5 Code Challenge

- Create a variable named db that is an SqliteDatabase with a filename of challenges.db.
- Now add db as the database attribute in the Meta class for Challenge.

- Finally, create a function named initialize. Your initialize() function should connect to the database and then create the Challenge table. Make sure it creates the table safely.

```
In [13]: from peewee import *

         db = SqliteDatabase('challenges.db')

         class Challenge(Model):
             name = CharField(max_length=100)
             language = CharField(max_length=100)
             steps = IntegerField(default=1)

             class Meta:
                 database = db

         def initialize():
             db.connect()
             db.create_tables([Challenge], safe=True)
```

4.0.6 Code Challenge

- Import OrderedDict from the collections module.
- Now create an OrderedDict named menu that has the menu items exactly as listed in the comment. Both keys and values will be strings.

```
In [14]: from collections import OrderedDict

         menu = OrderedDict([
         ('n', 'new challenge'),
         ('s', 'new step'),
         ('d', 'delete a challenge'),
         ('e', 'edit a challenge')
         ])
```

4.0.7 Code Challenge

- Create a function named create_challenge() that takes name, language, and steps arguments. Steps should be optional, so give it a default value of 1. Create a Challenge from the arguments. create_challenge should not return anything.

```
from models import Challenge

def create_challenge(name, language, steps=1):
    Challenge.create(name=name, language=language, steps=steps)
```

5 Code Challenge

- Create a function named search_challenges that takes two arguments, name and language. Return all Challenges where the name field contains name argument and the language field is equal to the language argument. Use == for equality. You don't need boolean and or binary & for this, just put both conditions in your where().

```
from models import Challenge

def create_challenge(name, language, steps=1):
    Challenge.create(name=name,
                     language=language,
                     steps=steps)

def search_challenges(name, language):
    challenges = Challenge.select()
    return challenges.where(Challenge.name.contains(name) and Challenge.language == language)
```

5.0.1 Code Challenge

- Create a function named delete_challenge that takes a Challenge as an argument. Delete the specified Challenge. Your function shouldn't return anything.

```
from models import Challenge


def create_challenge(name, language, steps=1):
    Challenge.create(name=name,
                     language=language,
                     steps=steps)


def search_challenges(name, language):
    return Challenge.select().where(
        Challenge.name.contains(name),
        Challenge.language==language
    )

def delete_challenge(challenge):
    challenge.delete_instance()
```