OOP in Python

next steps

Agenda

1. ORM in Python with peewee
2. Useful tricks and helpers
    - Properties (private attributes)
    - Importing packages/functions
    - *args
    - **kwargs

Object-relational mapping (ORM)

- ORMs help you a lot in writing complex CRUD operations, which are a pain to write via manual SQL
  - Create a new object in a database
  - Update an object in a database
  - Delete an object from a databse

- We will use peewee in this course to try-out object-relationa mapping: http://docs.peewee-orm.com/en/latest/

ZOO Animals

- Create classes that describe ZOO animals (name, rating) and s**tore the animals into a simple SQL database.**

1.
Write the Class for Zoo Animal

2.
Write the SQL code for creating a table, inserts, deletes, saves

ORM Approach with peewee

1. Import peewee (install it through pip)
2. Set the database name
3. Create the class for the ZOO animals by inheriting from peewee.Model
4. Create the table for the class
5. Create a few instances of the ZOO animal
6. Save them in the database

1. Importing pewee & 2. Setting DB

```python
# we first import the ORM
import peewee as pw

# we set the name of the Sqlite database
db = pw.SqliteDatabase('animals2.db')
```

3. Create the class for the ZooAnimal

```
class ZooAnimal(pw.Model):
    """
    ORM model of the ZooAnimal
    """
    animal_name = pw.TextField()
    a_popular = pw.IntegerField() # 1 least, 5 most

    class Meta:
        '''
        we connect the database to the models via the nested class
        '''
        database = db
```

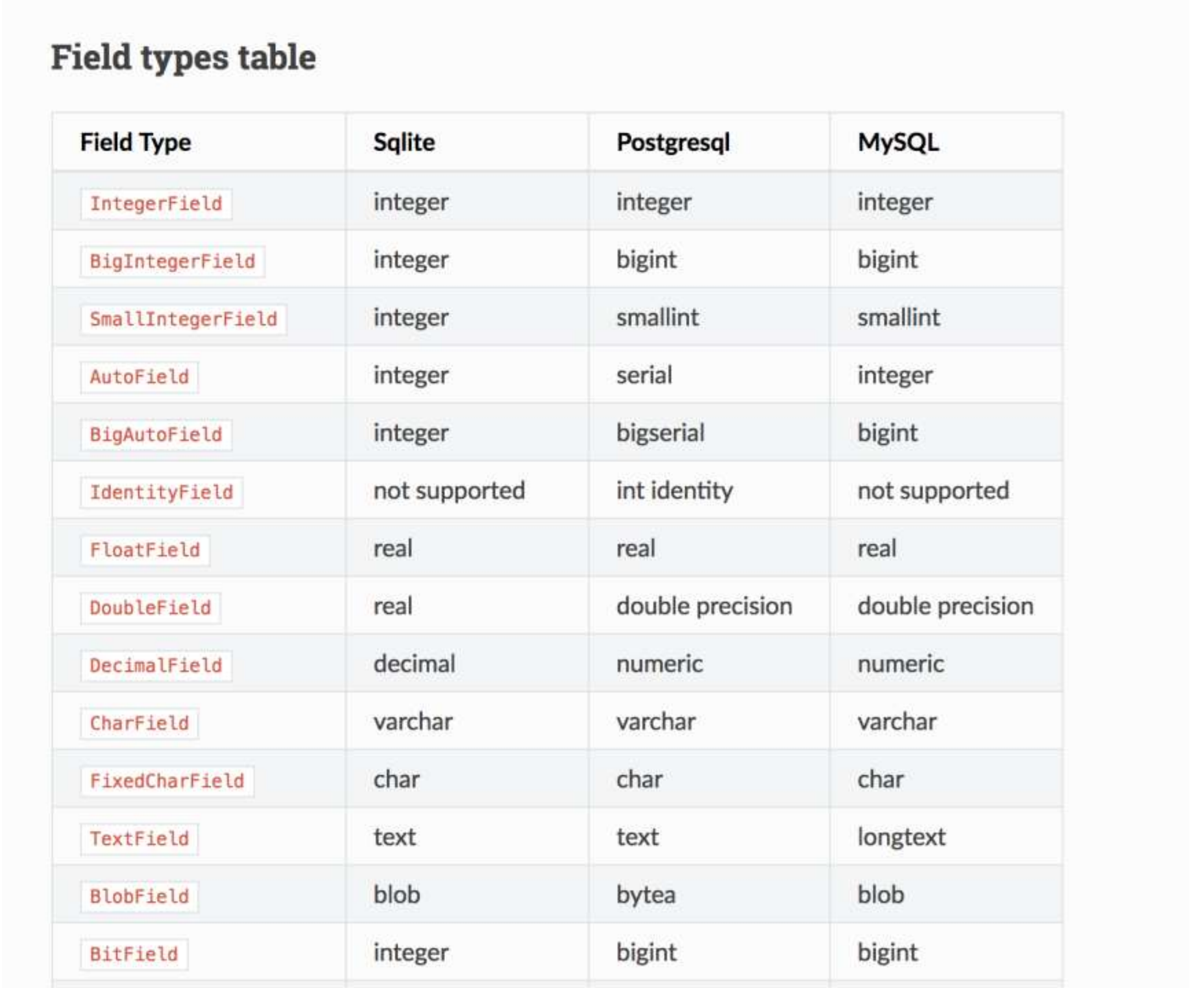
We inherit from peewee.Model class

```
class ZooAnimal(pw.Model):
    """
    ORM model of the ZooAnimal
    """
    animal_name = pw.TextField()
    a_popular = pw.IntegerField() # 1 least, 5 most

    class Meta:
        '''
        we connect the database to the models via the nested class
        '''
        database = db
```

The attributes need to be of a type!

http://docs.peewee-orm.com/en/latest/peewee/models.html#field-types-table

```
class ZooAnimal(pw.M  el):
    """
    ORM model of the   oAnimal
    """
    animal_name = pw.TextField()
    a_popular = pw.IntegerField() # 1 least, 5 most

    class Meta:
        '''
        we connect the database to the models via the nested class
        '''
        database = db
```



Field types table

Field Type	Sqlite	Postgresql	MySQL
`IntegerField`	integer	integer	integer
`BigIntegerField`	integer	bigint	bigint
`SmallIntegerField`	integer	smallint	smallint
`AutoField`	integer	serial	integer
`BigAutoField`	integer	bigserial	bigint
`IdentityField`	not supported	int identity	not supported
`FloatField`	real	real	real
`DoubleField`	real	double precision	double precision
`DecimalField`	decimal	numeric	numeric
`CharField`	varchar	varchar	varchar
`FixedCharField`	char	char	char
`TextField`	text	text	longtext
`BlobField`	blob	bytea	blob
`BitField`	integer	bigint	bigint

Model configuration is kept namespaced in a special class called Meta

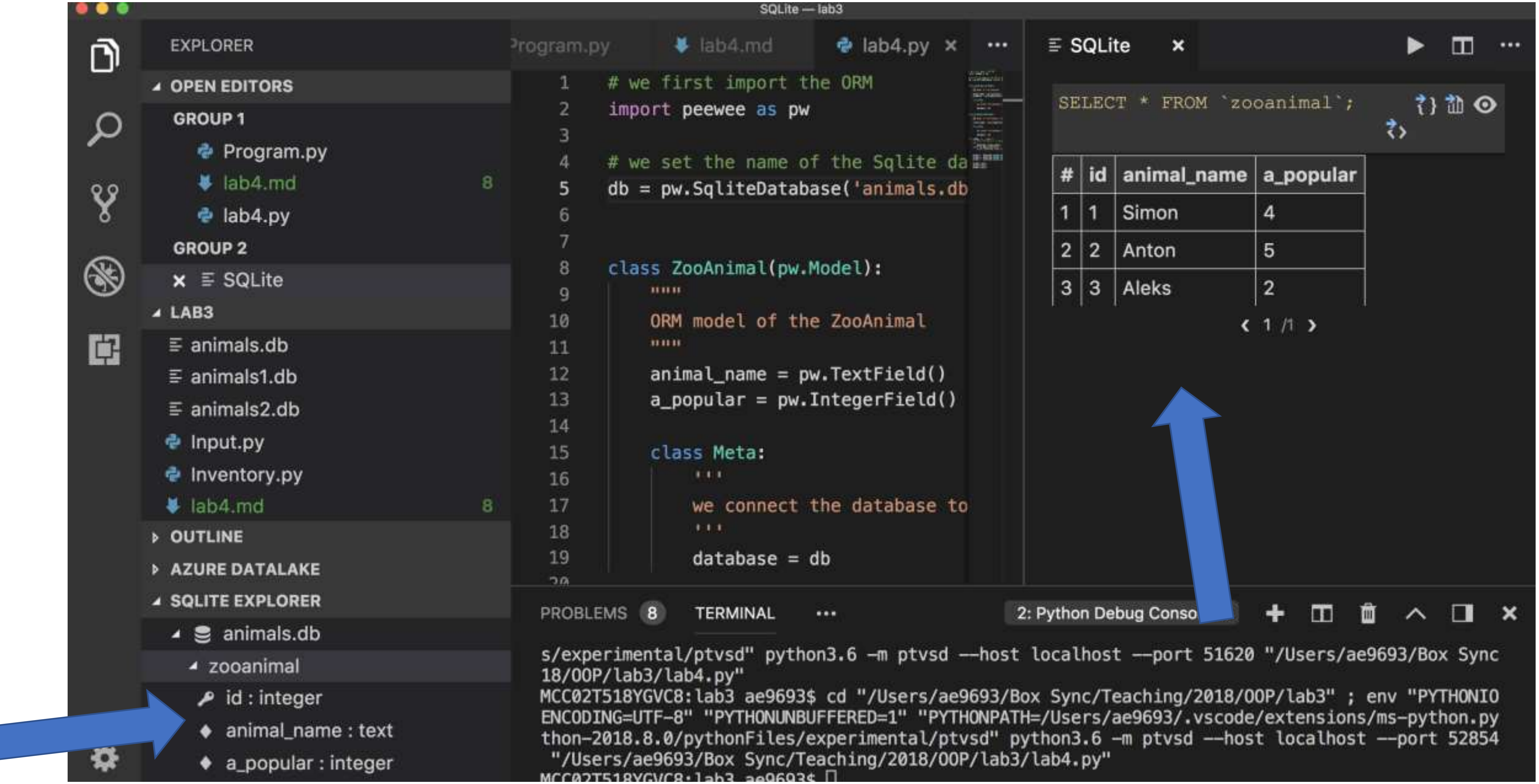
```
class ZooAr[illegible]al(pw.Model):
    """
    ORM mod[illegible] of the ZooAnimal
    """
    animal_[illegible]e = pw.TextField()
    a_popul[illegible] = pw.IntegerField() # 1 least, 5 most

    class Meta:
        '''
        we connect the database to the models via the nested class
        '''
        database = db
```

4. Creating the table for the Class

```python
# Create the table if it does not exist yet
try:
    ZooAnimal.create_table()
except pw.OperationalError:
    print("ZooAnimal table already exists!")
```

5. Create a few instances of the ZOO animal
6. Save them into a database

```
animal1 = ZooAnimal(animal_name="Simon", a_popular=4)
animal2 = ZooAnimal(animal_name="Anton", a_popular=5)
animal3 = ZooAnimal(animal_name="Aleks", a_popular=2)

animal1.save()
animal2.save()
animal3.save()
```



```python
# we first import the ORM
import peewee as pw

# we set the name of the Sqlite da
db = pw.SqliteDatabase('animals.db


class ZooAnimal(pw.Model):
    """
    ORM model of the ZooAnimal
    """
    animal_name = pw.TextField()
    a_popular = pw.IntegerField()

    class Meta:
        '''
        we connect the database to
        '''
        database = db
```

```sql
SELECT * FROM `zooanimal`;
```

#	id	animal_name	a_popular
1	1	Simon	4
2	2	Anton	5
3	3	Aleks	2

‹ 1 /1 ›

```
s/experimental/ptvsd" python3.6 -m ptvsd --host localhost --port 51620 "/Users/ae9693/Box Sync
18/OOP/lab3/lab4.py"
MCC02T518YGVC8:lab3 ae9693$ cd "/Users/ae9693/Box Sync/Teaching/2018/OOP/lab3" ; env "PYTHONIO
ENCODING=UTF-8" "PYTHONUNBUFFERED=1" "PYTHONPATH=/Users/ae9693/.vscode/extensions/ms-python.py
thon-2018.8.0/pythonFiles/experimental/ptvsd" python3.6 -m ptvsd --host localhost --port 52854
 "/Users/ae9693/Box Sync/Teaching/2018/OOP/lab3/lab4.py"
MCC02T518YGVC8:lab3 ae9693$ 
```

*Install the vs code sqlite extension!

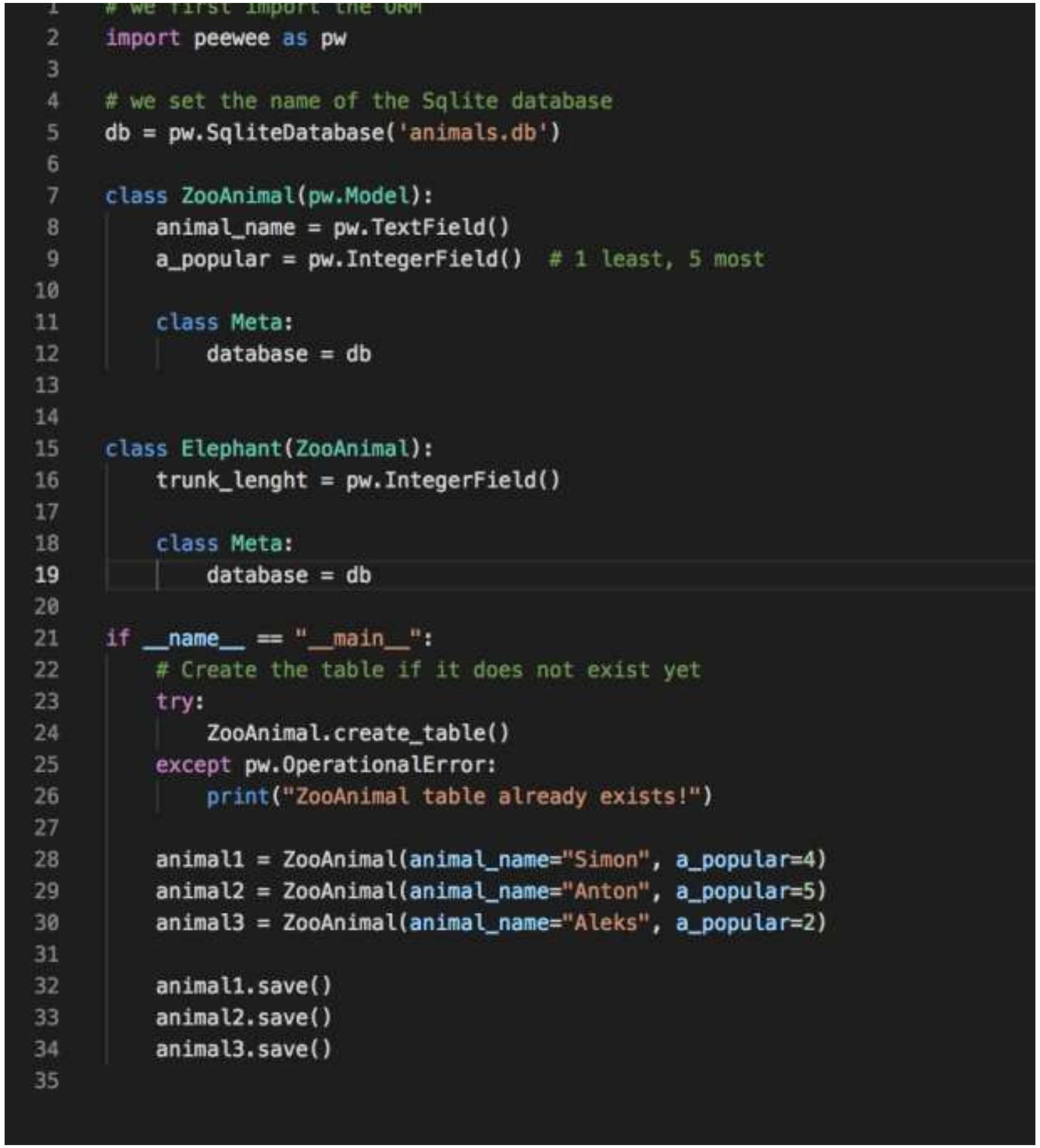
Other peewee things

- You can query for items in a database using .select()
- You can update items using .save()
- You can set primary and foreign keys

…

Exercise

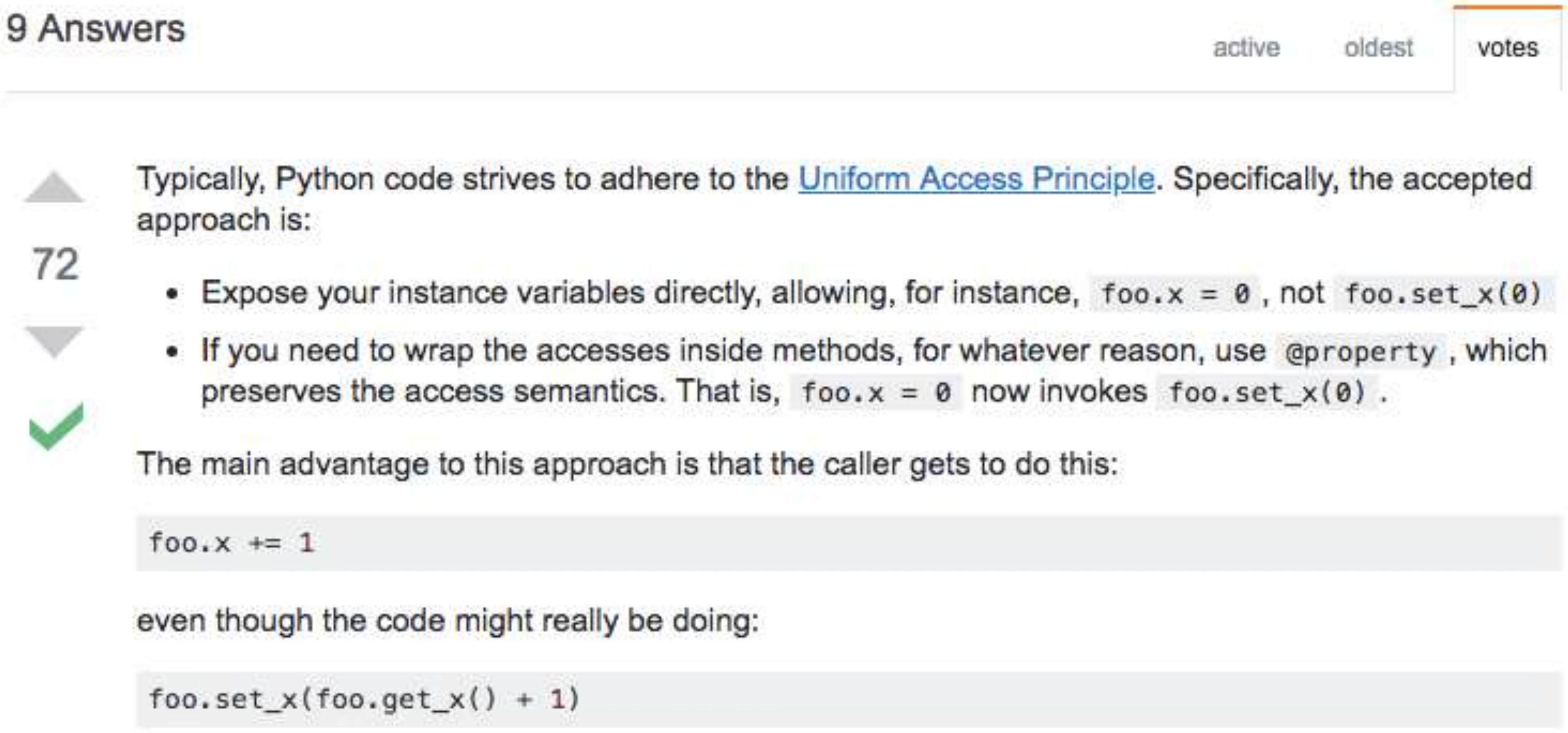
1. Update your Bus model (driver_name, number_of_seets) to peewee model.
2. Create a few objects of type Bus.
3. Create a table for buses in the code
4. Store your buses in the database
5. Explore the table



```python
# we first import the ORM
import peewee as pw

# we set the name of the Sqlite database
db = pw.SqliteDatabase('animals.db')

class ZooAnimal(pw.Model):
    animal_name = pw.TextField()
    a_popular = pw.IntegerField()  # 1 least, 5 most

    class Meta:
        database = db


class Elephant(ZooAnimal):
    trunk_lenght = pw.IntegerField()

    class Meta:
        database = db

if __name__ == "__main__":
    # Create the table if it does not exist yet
    try:
        ZooAnimal.create_table()
    except pw.OperationalError:
        print("ZooAnimal table already exists!")

    animal1 = ZooAnimal(animal_name="Simon", a_popular=4)
    animal2 = ZooAnimal(animal_name="Anton", a_popular=5)
    animal3 = ZooAnimal(animal_name="Aleks", a_popular=2)

    animal1.save()
    animal2.save()
    animal3.save()
```

2. Usefull things in OOP/ Python

Private Attributes



9 Answers

active oldest votes

72

Typically, Python code strives to adhere to the Uniform Access Principle. Specifically, the accepted approach is:

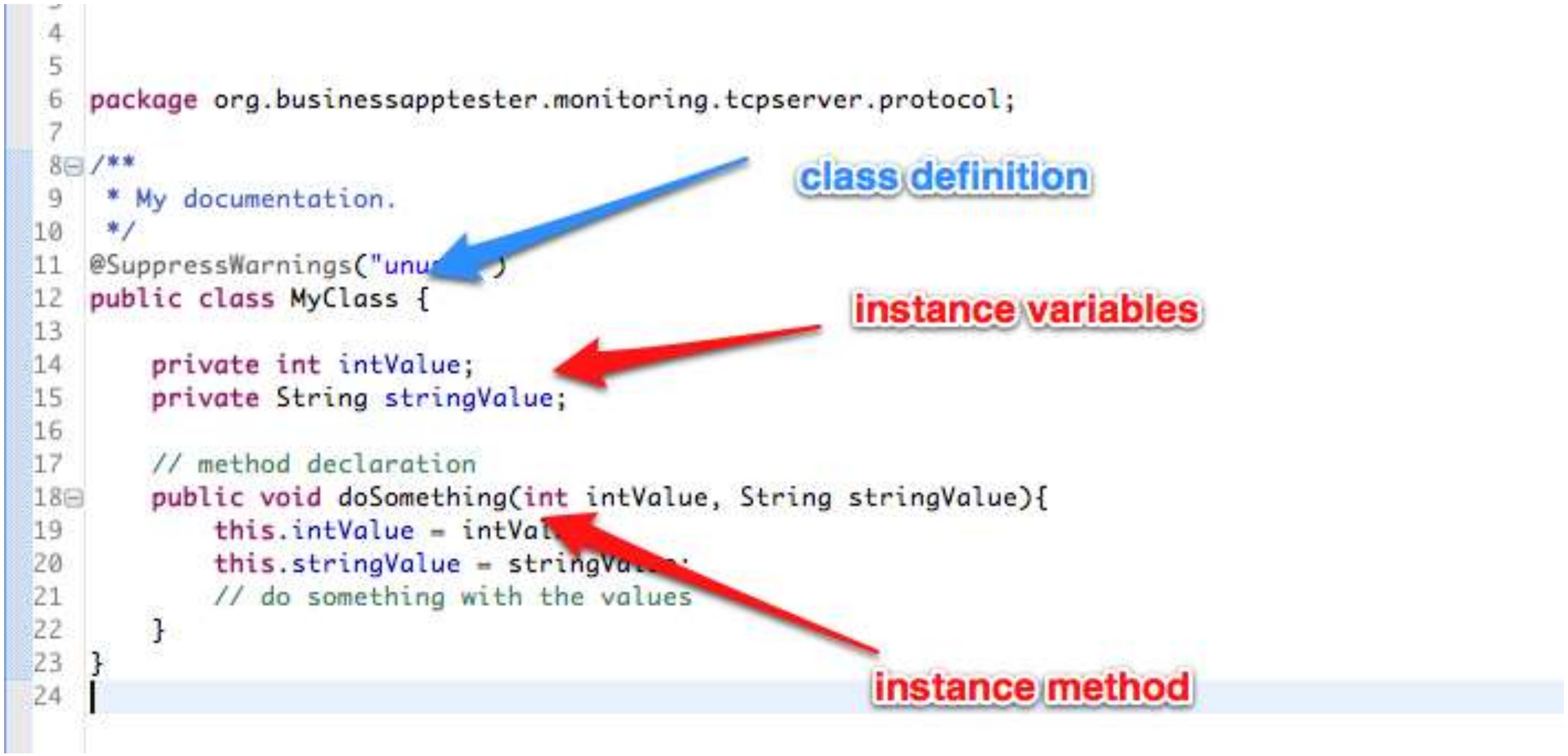
- Expose your instance variables directly, allowing, for instance, `foo.x = 0` , not `foo.set_x(0)`
- If you need to wrap the accesses inside methods, for whatever reason, use `@property` , which preserves the access semantics. That is, `foo.x = 0` now invokes `foo.set_x(0)` .

The main advantage to this approach is that the caller gets to do this:

```
foo.x += 1
```

even though the code might really be doing:

```
foo.set_x(foo.get_x() + 1)
```



```java


package org.businessapptester.monitoring.tcpserver.protocol;

/**
 * My documentation.
 */
@SuppressWarnings("unu    )
public class MyClass {

    private int intValue;
    private String stringValue;

    // method declaration
    public void doSomething(int intValue, String stringValue){
        this.intValue = intVal
        this.stringValue = stringVa
        // do something with the values
    }
}

```

Understanding: _, __, __str__

- _One underline in the beginning: private attribute, you should NOT access it directly but rather through a method / property
- __two underlines in the beginning: you should not override this method in a child class!
- __four__ underlines (2begining, 2 after): don’t call this method, python does it for you.

Properties

```
class Celsius:
def __init__(self, temperature = 0):
        self._temperature = temperature

def to_fahrenheit(self):
        return (self.temperature * 1.8) + 32

@property
def temperature(self):
        print("Getting value")
        return self._temperature

@temperature.setter
def temperature(self, value):
        if value < -273:
                raise ValueError(”T below -273 is not possible")
        print("Setting value")
        self._temperature = value
```

```
c = Celsius(10)
print(c.temperature)
```

Importing…

```
import os, sys
import config
from my.package.content import *
```

```
import os
import sys
# explicit is better than implicit
from my.package import config
from my.package.content import Octopus, Blowfish
```

*args and **kvargs

*args and **kwargs allow you to pass **a variable number** of arguments to a function.

What variable means here is that you do not know beforehand how many arguments can be passed to your function by the user.

http://book.pythontips.com/en/latest/args_and_kwargs.html

*argv

- *argv is used to pass a variable number of arguments without their keys (names)

```
def test_variable_arguments(f_arg, *argv):
print("first normal arg:", f_arg)
        for x in argv:
        print("another arg through *argv:", x)

test_variable_arguments('Aleks', 'Anton', 'Nancy', 'Annabella')
```

**kvargs

kwargs allows you to pass **keyworded variable length of arguments to a function.

```
def print_people(**kvargs):
        for key,value in kvargs.items():
                print("Key: {k}, Value: {v}".format(k=key, v=value))


people = {"Aleks": 28, "Anton": 28}
print_people(**people)
```

Summary

1. We learned how to implement ORM in Python using peewee
2. We discuss useful tricks and helpers
   1. Properties
   2. Importing packages/functions
   3. *args
   4. **kwargs

Check: https://github.com/google/styleguide