Exercises chapter 9: Classes

Saul SL

June 2023

1 Setup

```python
import os
os.chdir('/home/saul/Documents/Apuntes/Python/Matthes-2019/Ch-9')
```

2 Exercise 9-1 Restaurant

Make a class called Restaurant. The `__init__()` method for Restaurant should store two attributes: a `restaurant_name` and a `cuisine_type`. Make a method called `describe_restaurant()` that prints these two pieces of information, and a method called `open_restaurant()` that prints a message indi- cating that the restaurant is open. Make an instance called restaurant from your class. Print the two attri- butes individually, and then call both methods.

```python
class Restaurant():
    """Displays basic information of a restaurant

    """
    def __init__(self, name, cuisine_type):
        # super(Restaurant, self).__init__()
        self.name = name
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        print(f"'{self.name.title()}' information:")
        print(f"Cuisine: {self.cuisine_type.title()}")

    def open_restaurant(self):
        print(f"The restaurant '{self.name.title()}' is open")



pacena = Restaurant('la pacena', 'bolivian')
pacena.describe_restaurant()
pacena.open_restaurant()
```

3 Exercise 9-2 Three Restaurants

Start with your class from Exercise 9-1. Create three different instances from the class, and call `describe_restaurant()` for each instance.

```python
tokio = Restaurant('tokio', 'japanese')
cocha = Restaurant('Cocha', 'bolivian')
gg = Restaurant('gg', 'chinese')

tokio.describe_restaurant()
```

4 Exercise 9-3 Users

Make a class called User. Create two attributes called `first_name` and `last_name`, and then create several other attributes that are typically stored in a user profile. Make a method called `describe_user()` that prints a summary of the user's information. Make another method called `greet_user()` that prints a personalized greeting to the user. Create several instances representing different users, and call both methods for each user.

```python
class User():
    """Displays information for a user

    """
    def __init__(self, first_name, last_name, age, location, language):
        # super(User, self).__init__()
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.location = location
        self.language = language

    def describe_user(self):
        user_name = f"{self.first_name.title()} {self.last_name.title()}"
        print(f"Name: {user_name}")
        print(f"Location: {self.location.title()}")
        print(f"Language: {self.language.title()}")
        print(f"Age: {self.age}")

    def greet_user(self):
        iname = self.first_name.title()
        print(f"Welcome {iname}")


saul = User('saul', 'sotomayor', age=40, location='la paz', language='spanish')
saul.describe_user()
saul.greet_user()

tim = User('tim', 'burton', age=55, location='LA', language='English')
tim.describe_user()
tim.greet_user()
```

5 Exercise 9-4 Number Served

Start with your program from Exercise 9-1 (page 162). Add an attribute called `number_served` with a default value of 0. Create an instance called restaurant from this class. Print the number of customers the restaurant has served, and then change this value and print it again. Add a method called `set_number_served()` that lets you set the number of customers that have been served. Call this method with a new number and print the value again. Add a method called `increment_number_served()` that lets you increment the number of customers who've been served. Call this method with any num- ber you like that could represent how many customers were served in, say, a day of business.

```python
class Restaurant():
    """Displays basic information of a restaurant

    """
    def __init__(self, name, cuisine_type, number_served=0):
        # super(Restaurant, self).__init__()
        self.name = name
        self.cuisine_type = cuisine_type
        self.number_served = number_served

    def describe_restaurant(self):
        print(f"'{self.name.title()}' information:")
        print(f"Cuisine: {self.cuisine_type.title()}")

    def open_restaurant(self):
        print(f"The restaurant '{self.name.title()}' is open")

    def set_number_served(self, number):
        self.number_served = number

    def increment_number_served(self, number):
        self.number_served += number


# A
italo = Restaurant('italo', 'italian')
italo.number_served
italo.number_served = 12

# B
italo.set_number_served(2)
italo.number_served

# C
italo.increment_number_served(2)
italo.number_served
```

6 Exercise 9-5 Login Attempts

Add an attribute called `login_attempts` to your User class from Exercise 9-3 (page 162). Write a method called `increment_login_attempts()` that increments the value of `login_attempts` by 1. Write another method called `reset_login_attempts()` that resets the value of `login_attempts` to 0. Make an instance of the User class and call `increment_login_attempts()` several times. Print the value of `login_attempts` to make sure it was incremented properly, and then call `reset_login_attempts()`. Print `login_attempts` again to make sure it was reset to 0.

```
class User():
    """Displays information for a user

    """
    def __init__(self, first_name,
                 last_name, age,
                 location, language,
                 login_attempts=0):
        # super(User, self).__init__()
        self.first_name = first_name
        self.last_name = last_name
        self.age = age
        self.location = location
        self.language = language
        self.login_attempts = login_attempts

    def describe_user(self):
        user_name = f"{self.first_name.title()} {self.last_name.title()}"
        print(f"Name: {user_name}")
        print(f"Location: {self.location.title()}")
        print(f"Language: {self.language.title()}")
        print(f"Age: {self.age}")

    def greet_user(self):
        iname = self.first_name.title()
        print(f"Welcome {iname}")
```

```python

    def increment_login(self):
        self.login_attempts += 1

    def reset_login_attempts(self):
        self.login_attempts = 0


saul = User('saul', 'soto', 40, 'la paz', 'spanish')
saul.increment_login()
saul.login_attempts
saul.reset_login_attempts()
saul.login_attempts
```

7 Exercise 9-6 Ice Cream Stand

An ice cream stand is a specific kind of restaurant. Write a class called IceCreamStand that inherits from the Restaurant class you wrote in Exercise 9-1 (page 162) or Exercise 9-4 (page 167). Either version of the class will work; just pick the one you like better. Add an attribute called flavors that stores a list of ice cream flavors. Write a method that displays these flavors. Create an instance of IceCreamStand, and call this method.

```python
class Restaurant():
    """Displays basic information of a restaurant

    """
    def __init__(self, name, cuisine_type):
        # super(Restaurant, self).__init__()
        self.name = name
        self.cuisine_type = cuisine_type

    def describe_restaurant(self):
        print(f"'{self.name.title()}' information:")
        print(f"Cuisine: {self.cuisine_type.title()}")

    def open_restaurant(self):
        print(f"The restaurant '{self.name.title()}' is open")


class IceCreamStand(Restaurant):
    """Class to model a Ice cream dely

    """
    def __init__(self, name, cuisine_type, flavors=None):
        super(IceCreamStand, self).__init__(name, cuisine_type)
        self.name = name
        self.cuisine = cuisine_type
        self.flavors = flavors

    def show_flavors(self):
        if self.flavors:
            print(f"These are the available flavors at {self.name.title()}")
            for flavor in self.flavors:
                print(f"- {flavor.title()}")
        else:
            print("We don't have any ice creams, for now")


frigo_flavors = ['canela', 'frutilla', 'chirimoya', 'uva']
frigo = IceCreamStand('frigo', 'ice cream', frigo_flavors)
frigo.show_flavors()
```

```python
panda = IceCreamStand('panda', 'ice cream', frigo_flavors)
panda.show_flavors()
```

8 Exercise 9-7 Admin

An administrator is a special kind of user. Write a class called Admin that inherits from the User class you wrote in Exercise 9-3 (page 162) or Exercise 9-5 (page 167). Add an attribute, privileges, that stores a list of strings like ”can add post”, ”can delete post”, ”can ban user”, and so on. Write a method called `show_privileges()` that lists the administrator’s set of privileges. Create an instance of Admin, and call your method.

```python
class User():
    """Displays information for a user
    """
    def __init__(self, first_name, last_name, location, language):
        # super(User, self).__init__()
        self.first_name = first_name
        self.last_name = last_name
        self.location = location
        self.language = language

    def describe_user(self):
        user_name = f"{self.first_name.title()} {self.last_name.title()}"
        print(f"Name: {user_name}")
        print(f"Location: {self.location.title()}")
        print(f"Language: {self.language.title()}")
        print(f"Age: {self.age}")

    def greet_user(self):
        iname = self.first_name.title()
        print(f"Welcome {iname}")


class Admin(User):
    """Defines privileges of a root user
    """
    default_powers = ['add post',
                      'flag post',
                      'edit post',
                      'delete post']

    def __init__(self, first_name, last_name, location, language, sudo_powers=default_powers):
        super(Admin, self).__init__(first_name, last_name, location, language)
        self.sudo_powers = sudo_powers

    def show_privileges(self):
        print("As an Admin you can:")
        for power in self.sudo_powers:
            print(f"- {power}")
        print("----------")


root = Admin('root', 'debian', 'la paz', 'spanish')
root.show_privileges()
```

9 Exercise 9-8 Privileges

Write a separate Privileges class. The class should have one attribute, privileges, that stores a list of strings as described in Exercise 9-7. Move the `show_privileges()` method to this class. Make a Privileges instance as an attribute in the Admin class. Create a new instance of Admin and use your method to show its privileges.

```python
class Privilege():
    """Describes privileges of a user
    """
    default_powers = ["add post", "flag post", "edit post", "delete post"]

    def __init__(self, privileges=default_powers):
        # super(Privilege, self).__init__()
        self.privileges = privileges

    def show_privileges(self):
        print("As an Admin you can:")
        for power in self.privileges:
            print(f"- {power}")
        print("----------")


class Admin(User):
    """Defines privileges of a root user
    """

    def __init__(self, first_name,
                 last_name, location,
                 language):
        super(Admin, self).__init__(first_name, last_name, location, language)
        self.sudo_powers = Privilege()


root2 = Admin('saul', 'soto', 'lp', 'spanish')
root2.sudo_powers.show_privileges()
```

10 Exercise 9-9 Battery Upgrade

Use the final version of `electric_car`.py from this section. Add a method to the Battery class called `upgrade_battery()`. This method should check the battery size and set the capacity to 100 if it isn't already. Make an electric car with a default battery size, call `get_range()` once, and then call `get_range()` a second time after upgrading the battery. You should see an increase in the car's range.

```python
class Car:
    """A simple attempt to represent a car."""

    def __init__(self, make, model, year):
        self.make = make
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        long_name = f"{self.year} {self.manufacturer} {self.model}"
        return long_name.title()

    def read_odometer(self):
        print(f"This car has {self.odometer_reading} miles on it.")

    def update_odometer(self, mileage):
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        self.odometer_reading += miles


class Battery:
    """A simple attempt to model a battery for an electric car."""

    def __init__(self, battery_size=75):
        """Initialize the battery's attributes."""
        self.battery_size = battery_size

    def describe_battery(self):
        """Print a statement describing the battery size."""
        print(f"This car has a {self.battery_size}-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 75:
            range = 260
        elif self.battery_size == 100:
            range = 315

        print(f"This car can go about {range} miles on a full charge.")

    def upgrade_battery(self):
        if self.battery_size < 100:
            self.battery_size = 100
        else:
            print(f"Battery can't be upgraded, size is {self.battery_size}")


class ElectricCar(Car):
    """Represent aspects of a car, specific to electric vehicles."""

    def __init__(self, make, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__(make, model, year)
        self.battery = Battery()


audi = ElectricCar('audi', 'evo', '2020')
audi.battery.get_range()
audi.battery.describe_battery()
audi.battery.upgrade_battery()

audi.battery.get_range()
audi.battery.describe_battery()
```

11 Exercise 9-10 Imported Restaurant

Using your latest Restaurant class, store it in a mod- ule. Make a separate file that imports Restaurant. Make a Restaurant instance, and call one of Restaurant's methods to show that the import statement is work- ing properly.

```python
from Restaurant import Restaurant, IceCreamStand

riel = Restaurant('la riel', 'bolivian')
riel.describe_restaurant()
```

12 Exercise 9-11 Imported Admin

Start with your work from Exercise 9-8 (page 173). Store the classes User, Privileges, and Admin in one module. Create a sepa- rate file, make an Admin instance, and call `show_privileges()` to show that everything is working correctly.

```python
import user as user
iuser = user.User('sam', 'smith', 'US', 'english')
iuser.describe_user()
iuser.greet_user()
```

13 Exercise 9-12 Multiple Modules

Store the User class in one module, and store the Privileges and Admin classes in a separate module. In a separate file, create an Admin instance and call `show_privileges()` to show that everything is still working correctly.

```python
from  user import User
from user_extension import Admin, Privilege

filio = User('ale', 'filio', 'mexico', 'spanish')
filio.describe_user()

root = Admin('root', 'of tree', 'land', 'leaf')
root.describe_user()
```

14 Exercise 9-13 Dice

Make a class Die with one attribute called sides, which has a default value of 6. Write a method called `roll_die()` that prints a random number between 1 and the number of sides the die has. Make a 6-sided die and roll it 10 times. Make a 10-sided die and a 20-sided die. Roll each die 10 times.

```python
from random import randint


class Dice():
    """Simulates a dice with n sides

    """
    def __init__(self, sides=6):
        # super(Die, self).__init__()
        self.sides = sides

    def roll_dice(self):
        return randint(1, self.sides)


dado6 = Dice()
dado10 = Dice(10)
dado20 = Dice(20)

# resultados = []
# for _ in range(1, 11):
#     resultados.append(dado10.roll_dice())

# for i in resultados:
#     print(i, end=" ")
# print("")
```

```python

for _ in range(1, 11):
    print(dado20.roll_dice(), end=" ")
print("")
```

15 Exercise 9-14 Lottery

Make a list or tuple containing a series of 10 numbers and five letters. Randomly select four numbers or letters from the list and print a message saying that any ticket matching these four numbers or letters wins a prize.

```python
from random import choice
import string


def draw_number(n):
    num_pool = list(range(1, 11))
    all_lett = list(string.ascii_lowercase)
    lett_pool = all_lett[0:5]
    ipool = num_pool + lett_pool
    inum = None
    for _ in range(1, n+1):
        i = choice(ipool)
        i = str(i)
        if inum is None:
            inum = i
        else:
            inum += i
    return inum


winner_num = draw_number(4)
print(f"Any ticket matching '{winner_num}' will win 1 million $")
```

16 Exercise 9-15 Lottery Analysis

You can use a loop to see how hard it might be to win the kind of lottery you just modeled. Make a list or tuple called `my_ticket`. Write a loop that keeps pulling numbers until your ticket wins. Print a message reporting how many times the loop had to run to give you a winning ticket.

```python
my_ticket = draw_number(4)
n_attempt = 1
while my_ticket != winner_num:
    my_ticket = draw_number(4)
    n_attempt += 1

print(f"I took {n_attempt} attempts to win the lottery")
```

17 Exercise 9-16 Python Module of the Week

One excellent resource for exploring the Python standard library is a site called Python Module of the Week. Go to `https://pymotw.com/3/` and look at the table of contents. Find a module that looks interesting to you and read about it, perhaps starting with the random module.

```python
# Module of the week
"""
```

```
The syntax used in Python’s re module is based on the syntax used for regular expressions in Perl, with a few Python-specific e

Although re includes module-level functions for working with regular expressions as text strings, it is more efficient to compi
"""
```