Dynamic Modelling Course - TEP4290: Warm-up 11

The point of this exercise is for you to get familiar with the terminology and logic of objects in python and for advanced students to get some first hands-on practice with classes. You can use the following resources:

- the python for everyone classes (video 1 and 2 for the mandatory part, 3 and 4 optionally) https://www.py4e.com/lessons/Objects
- notebook for advanced content: https://www.nbshare.io/notebook/986482930/Object-Oriented-Programming-In-Python/#Python-Classes

You will demonstrate some understanding of objects in teh mandatory part and to some light coding in the optional part of the assignment. The exercise is pass/fail (only the mandatory part is considered).

Good luck!

Quick summary

Fundamentally almost everything you use in Python has to do with objects - it is an object oriented language! This opens up exiting options like using built in functions for objects that you already are familiar with, but we also want to understand how an object is fundamentally built in Python.

Nomenclature

You should understand the following words and underlying concepts:

- class: a template for objects of a certain type
- method: function for a object of a certain type
- attribute: data (object) that belongs to an object
- instance: an object of a class

advanced:

- constructor: the function that takes care of making an instance of that class
- inheritance: the concept of taking the methods and attributes of a parent class into the child class

Anatomy of class definitions

In [1]:
```
#fundamental structure:
class ClassName: #definition. Class names _should_ start with capital letters
    class_attribute = 'attribute' # every instance will have this attribute

    def __init__(self, init_argument): #the constructor of the class, is called by
        self.init_argument = init_argument # self just points at this object, so th
    def method(self, argument): #defining a method for this class that does somethi
        print('Do some stuff with', argument)
```

A simple example

In [2]:
```
class Dog:
    '''Simple example class.'''
    def __init__(self, name, owner):
        self.name = name
        self.owner = owner

    def bark(self):
        print('Wuff! Wuff!')

    def print_owner(self):
        print('The owner of', self.name,'is', self.owner)

bo = Dog('Bo', 'Obama')
bo.bark()
bo.print_owner()

fluffy = Dog('Fluffy', 'unknown')
fluffy.print_owner()
```

```
Wuff! Wuff!
The owner of Bo is Obama
The owner of Fluffy is unknown
```

Mandatory Tasks

Complete the tasks outlined below and either answer the questions or complete the code to achieve the same output as you find in the original file.

Methods of objects you know

Here we want you to look up the methods that objects that you already used have!

In [3]:
```
name = 'otto'
print(name)

dir(name)
```

```
otto
```

Out[3]:
```
['__add__',
 '__class__',
 '__contains__',
 '__delattr__',
 '__dir__',
 '__doc__',
 '__eq__',
 '__format__',
 '__ge__',
 '__getattribute__',
 '__getitem__',
 '__getnewargs__',
 '__getstate__',
 '__gt__',
 '__hash__',
 '__init__',
 '__init_subclass__',
 '__iter__',
 '__le__',
 '__len__',
 '__lt__',
 '__mod__',
 '__mul__',
 '__ne__',
 '__new__',
 '__reduce__',
 '__reduce_ex__',
 '__repr__',
 '__rmod__',
 '__rmul__',
 '__setattr__',
 '__sizeof__',
 '__str__',
 '__subclasshook__',
 'capitalize',
 'casefold',
 'center',
 'count',
 'encode',
 'endswith',
 'expandtabs',
 'find',
 'format',
 'format_map',
 'index',
 'isalnum',
 'isalpha',
 'isascii',
 'isdecimal',
 'isdigit',
 'isidentifier',
 'islower',
 'isnumeric',
 'isprintable',
 'isspace',
 'istitle',
```

```
 'isupper',
 'join',
 'ljust',
 'lower',
 'lstrip',
 'maketrans',
 'partition',
 'removeprefix',
 'removesuffix',
 'replace',
 'rfind',
 'rindex',
 'rjust',
 'rpartition',
 'rsplit',
 'rstrip',
 'split',
 'splitlines',
 'startswith',
 'strip',
 'swapcase',
 'title',
 'translate',
 'upper',
 'zfill']
```

Use the first public method capitalize:

In [4]:
```
name.capitalize()
```

Out[4]:
```
'Otto'
```

What is what?

In this task you will categorize what parts of the example code do.

Provided example code

You don't need to do anything here

In [5]:
```
class Answer:
    punctuation_mark = '.'

    def __init__(self, answer):
        self.content = answer

    def print_content(self):
        print(self.content + self.punctuation_mark)

sentence1 = Answer('Photosynthesis')
sentence1.print_content()
record = sentence1.content
```

```
Photosynthesis.
```

In one line, what role(s) do the following expressions take? What can you call them when thinking about what you learned?

- line 1, "Answer":
  - it is the name of the class
- line 2, " punctuation_mark = '.' " :
  - it is an attribute that belongs to class 'Answer'
- line 2, " '.' ":
  - it is a string, which is the value of attribute 'punctuation_mark'
- line 4-5: "def **init**(self, answer): self.content = answer ":
  - it is the constructor of the class
  - the value of answer is assigned to the second attribute 'content'
- line 10: "sentence1":
  - it is a variable referring to an instance of class 'Answer'
- line 11: "sentence1.print_content()":
  - it is a call of method 'print_content' from class 'Answer'

Optional tasks

You really don't have to do this, but if you didn't fell challenged so far, knock yourself out.

Make your own instances

We will again play with the Dog class here, and also introduce a Owner class.

You will create some instances of both classes and use their methods.

In [6]:
```
class Dog:
    '''Simple example dog class. The dog has an owner and can bark.'''
    def __init__(self, name):
        self.name = name
        self.owner = Owner('unknown')

    def bark(self):
        print('Wuff! Wuff!')

    def print_owner(self):
        print('The owner of', self.name,'is', self.owner.name)


class Owner:
    '''
    Simple example owner class that stores the dogs of an owner.

    Attributes:
    - name
    - dogs: list of all dogs that the owner has (or had!)
```

```
    Methods:
    -assign_dog(dog): adds the dog to the list of dogs and changes the owner of the
    - print_dogs
    '''
    def __init__(self, name):
        self.name = name
        self.dogs = [] #in the beginning the owner has no dogs
    def assign_dog(self, dog):
        self.dogs.append(dog) #adding the dog to the owners list of dogs
        dog.owner = self #sets the name of the owner for the dog

    def print_dogs(self):
        print(self.name,'owns the following dogs:')
        for dog in self.dogs:
            print(dog.name)
        print('\n')
```

You will now initialize 2 dogs, Fluffy and Huffy and let both bark and print their owners.

In [7]:
```
fluffy = Dog('Fluffy')
huffy = Dog('Huffy')

fluffy.bark()
huffy.bark()
fluffy.print_owner()
huffy.print_owner()
```

```
Wuff! Wuff!
Wuff! Wuff!
The owner of Fluffy is unknown
The owner of Huffy is unknown
```

Fluffy and Huffy of course have owners: Matt is the owner of Fluffy and Phil the owner of Huffy.

Initialize Matt and Phil, assign their dogs and print their list of dogs, as well as the ownership of both dogs!

In [8]:
```
matt = Owner('Matt')
matt.assign_dog(fluffy)
matt.print_dogs()

phil = Owner('Phil')
phil.assign_dog(huffy)
phil.print_dogs()

fluffy.print_owner()
huffy.print_owner()
```

```
Matt owns the following dogs:
Fluffy


Phil owns the following dogs:
Huffy


The owner of Fluffy is Matt
The owner of Huffy is Phil
```

Phil dies, and Matt takes care of Huffy. Update Matts dog and show what changed.

In [9]:
```
matt.assign_dog(huffy)
matt.print_dogs()
```

```
Matt owns the following dogs:
Fluffy
Huffy
```

Inheritance: make a cat class and CatOwner

Now you finally will do some own coding! Make a cat class that works just like the dog class, add the meow method and make two cats, Grumpy and Nyan.

In [10]:
```
class Cat:

    def __init__(self, name):
        self.name = name
        self.owner = Owner('unknown')

    def bark(self):
        print('Wuff! Wuff!')

    def meow(self):
        print('Meow! Meow!')

    def print_owner(self):
        print('The owner of', self.name,'is', self.owner.name)


grumpy = Cat('Grumpy')
nyan = Cat('Nyan')
```

You made a cat from a dog, so your cats are special: they can meow *and* bark!

In [11]:
```
grumpy.bark()
grumpy.meow()
```

```
Wuff! Wuff!
Meow! Meow!
```

Tryto add the cats to Phil and Matt and see what happens!

In [12]:
```
phil.assign_dog(grumpy)
matt.assign_dog(nyan)

phil.print_dogs()
matt.print_dogs()
```

```
Phil owns the following dogs:
Huffy
Grumpy


Matt owns the following dogs:
Fluffy
Huffy
Nyan
```

Done! Done?

If you made it this far: kudos!

It might be fun for you to develop some more small objects that interact with each other - you can do that here, but it is up to you what you do! Just think about something you want to represent in your code (a school with students and classes, blackjack or a function that calculates areas of (regular) shapes), set up a structure for your code and start - maybe even with some support from code you found online.