## Computer data base

You have been asked to create a data base for Computer store.

A **computer** will always have:

* **brand - string of any asci character without space and comma**
* **id – integer number**
* **ram - integer point number**
* **cores- integer number**

This store accepts only two types of computers: Laptop and PC. Both types has the same parameters as a basic computer but also:

* Laptop has another property – sub\_brand - **string of any asci character without space and comma**
* PC **has another property – serial\_number – integer number**

**Store does not get just a basic computer. They must be one of the two types mentioned above.**

You will start receiving input data in format:

**\* Laptop({name}, {id}, { ram }, { cores }, {** sub\_brand **})**

-*if there is no fifth parameter you should print* ***'\_\_init\_\_() missing 1 required positional argument: '*** sub\_brand ***''***

*-if the fifth parameter is not a string you should print* ***'*** sub\_brand ***must be string'***

***In both cases you do not add the laptop in DB***

**\*** PC **({name}, {id}, { ram }, { cores }, { serial\_number})**

*-if there is no fifth parameter you should print '****\_\_init\_\_() missing 1 required positional argument: '* serial\_number *''***

*-if the fifth parameter is not a string you should print* ***'* serial\_number *must be int'***

***In both cases you do not add the PC in DB***

Is it possible a basic computer to be delivered to the store. If you receive a command:

**\*Computer({name}, {id}, { ram }, { cores })**

You must print **‘Can't instantiate abstract class Computer with abstract methods \_\_init\_\_’**  and you must not add it in the DB.

## Output

When you receive a command ‘**end’ you must print the information about the army in the following format:**

* Sum of all **serial numbers** of computers: { **serial\_number** }
* Sum of all cores: {core}
* Laptop: {count}; PC: {count}

Where { serial\_number } is the sum of all Laptops in the DB;

{ core } is the sum of all cores of all laptops and PCs;

{count} is the number of all Laptops/PCs in DB

## Mini-Lab “Pet store site”

We have task to create pet store site. The site should have – users, food, animals, toys and cages. Every animal has its own cage. A user can buy or just look around in our site, NOTE that just register users can do this things. An unregistered user has no permissions at all in our site.

## Part 1 “Create ‘architecture’ and user”

Let’s create a pet store site. It isn’t ease to handle this task, but every task is possible if we managed to create small sub-tasks and implement them step by step. First of all we have to make new pycharm project. In this project as sub-folder create folder called DB. Next to this folder create two more – “Views” and “Models”

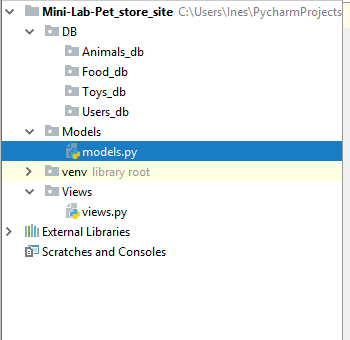
Take a minute and think what a user would have as a properties. You should be able to present the user his data for animals, food, toys and cages he has bought.

HINT:

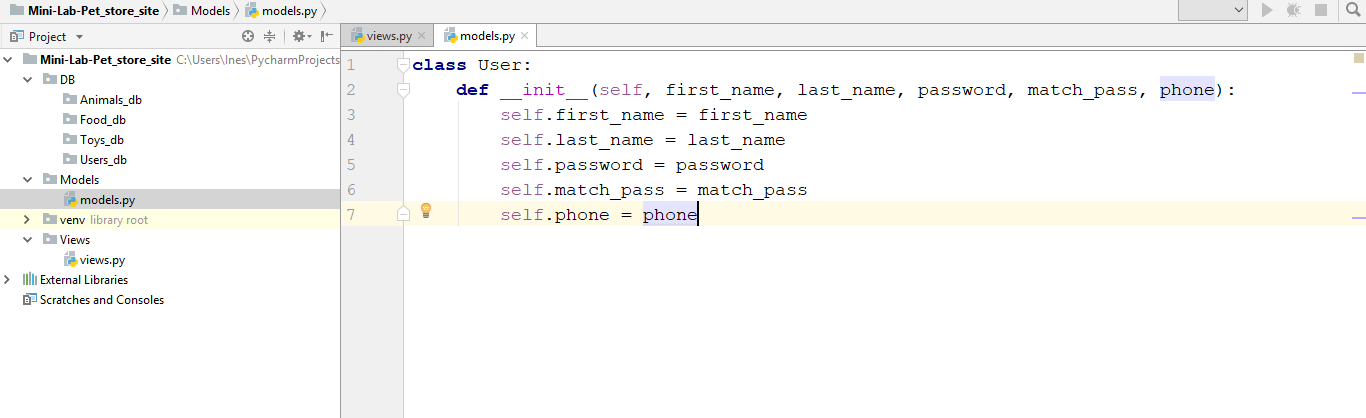
Usually basic user register with this properties:

First\_name, last\_name, password, repeat\_pass, phone\_number. Store the users you will receive in a safe place, let’s call it a folder “Users\_DB”. There we will have a file with all of ours users, called User\_db in DB folder.

Go to Models folder and create python file called models.py. There create your class for User representation.



Your class should look something like this so far:



## Part 2 “Validate user (encapsulation and validation)”

Sadly it could not be that way – it has no encapsulation and validation, so someone can take advantage and could messed up our DB. So let’s fix that

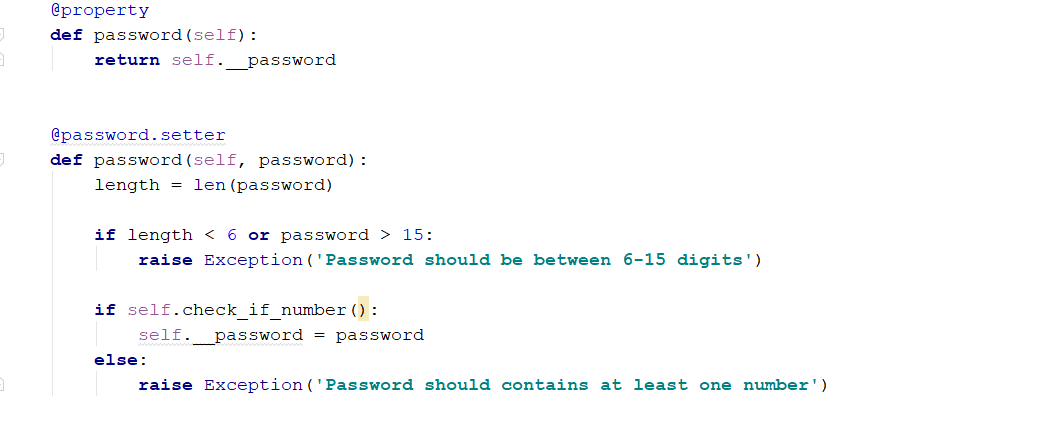
First\_name should be at least 3 characters and max 20, otherway print(in our case exception).

Last\_name -> same as fist\_name

Password should be at least 6 char long and should contain at least one number. Max 15 chars

Match\_pass must be the same as password.

Phone number should be a valid Bulgarian phone number starting with +359 and 9 numbers after that



Create validations of every property, so that they can pass only valid data.

## Part 3 “Other classes”

Now it is your turn – create class Toy which has kind (dog’s or cat’s) and price, class Food which has property kilograms kind (dog’s, cat’s), price and date of validity, class Cage which has volume, kind (dog/cat), price and create classes Dog and Cat. Dog has name, age, playfulness (int), Cat has name, age and laziness (int). All this classes go to models.py file. Think of using abstraction for Food, Toy and Cage. They have properties which are the same – price and kind. Create class which holds this properties and their validation and inherit other classes from it. EVERY CLASS MUST HAVE PROPERTY CALLED id (int).

## Part 4 “Other classes validation and encapsulation”

Think of possible validations of the classes mentioned above. For example – price and age can not be negative numbers…. And so on….

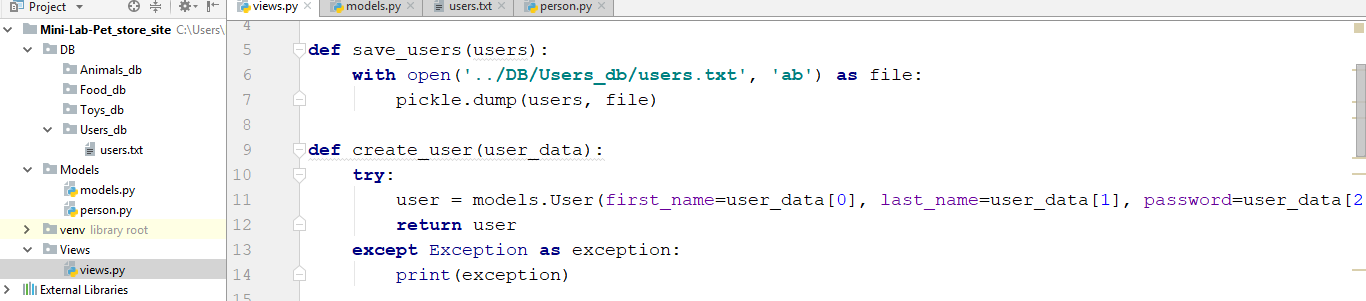
## Part 5 “Users data registration”

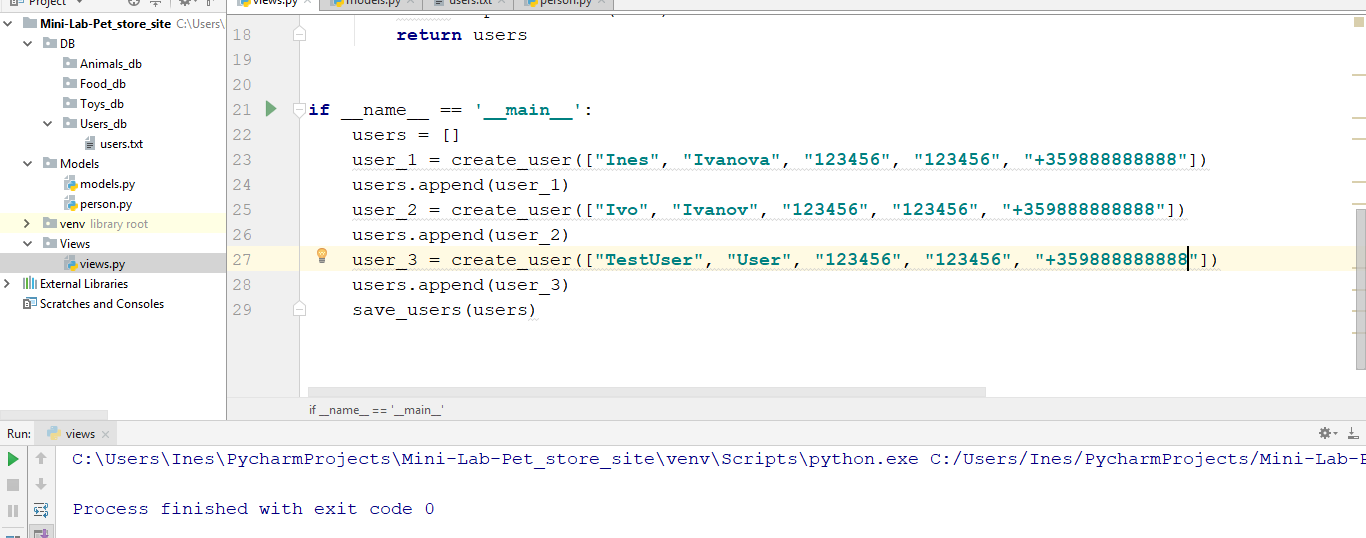
Register at least 3 valid users in your application. The registration logic should be:

-in views.py make function register user and import your user class from Models/model.py

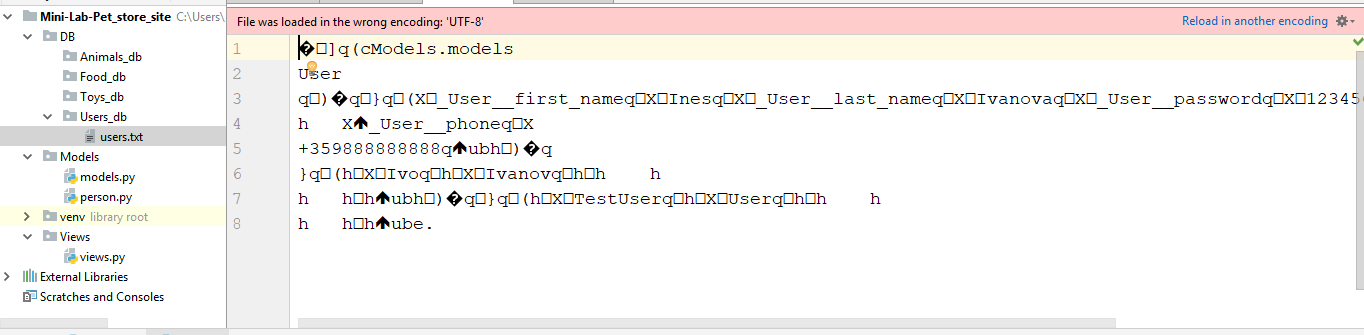
- if a user object is created successfully store it in list and put that list DB/User\_db/users.txt file in binary mode using pickles module

- if user is invalid print the exception without exiting with code 1, but 0.





If you have successfully registered your users, the txt file should look like something like this



## Part 5 “Animals, toys, cages and foods data registration”

Register at least 3 objects from any class and store them in the DB directory into files

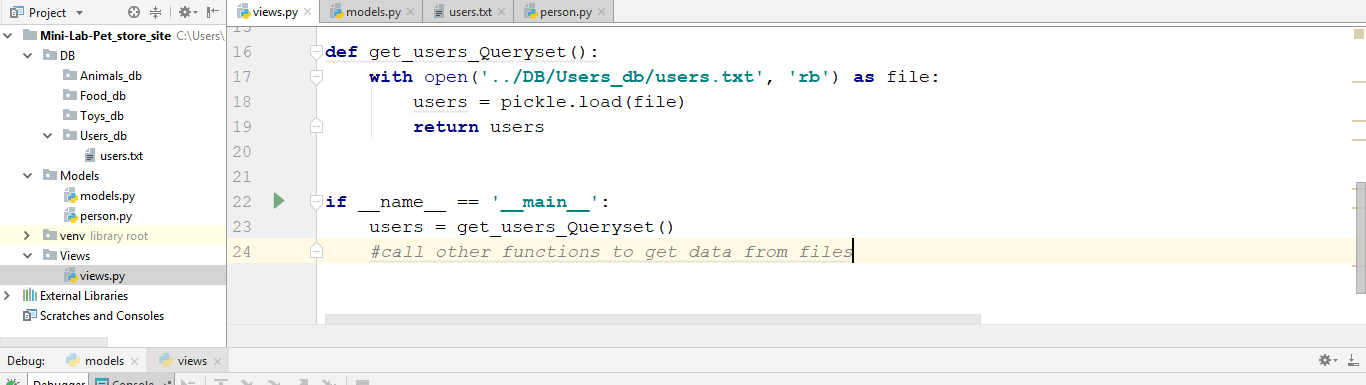
Sub-directories in binary mode.

After that in views create couple of functions: get\_users\_Queryset, get\_animals\_Queryset and so on… the result from them store in a couple of lists.

When you creating objects keep an eye of the id property – for every object of the same class the ids must be different. Check for module/library which can generate random ids.

Give your application a starting point – **if** \_\_name\_\_ == **'\_\_main\_\_'**:

and in this scope call the get functions



## Part 6 “Time to go shopping”

Now we are almost there – we have our users, our animals, cage, toys and food. Now let’s create a class, which is responsible for a purchasing logic. This class has user (who bought the item), the item itself and quantity.

Imagine you will receive a query from console in format

{first\_name} {last\_name} {password} {item\_id}

Add at least 3 purchases in your DB. Keep attention to the user – before you can create Purchase object you should check if in DB there is such user with given names and pass and if in DB there is an item with the given id, If it is so – create a purchase, If not print – “No such user or item is unavailable”. Do not forget to remove this product from DB.

## Part 7 “Report time”

In this part you should think of possible reports and implement them. For example – How many cats are there in DB? Or to receive a first\_name, last\_name and pass and you should print all purchases of that user. Sorting things depends on you. You have the freedom to think of every possible query and implement it.

## Part 8 “Extend/Refactor the project”

Think of the things you can improve in the project (for example user should also have id, and maybe email – this should be the unique thing in the user profile, maybe a user\_name?). You can add more products to the store, or even services – let’s say we want to send email to the user when he buys something? Keep the logic and files separated – this way you can extend and refactor your project much easier.