Proiect final

Semelbauer Iaroslav Stefan

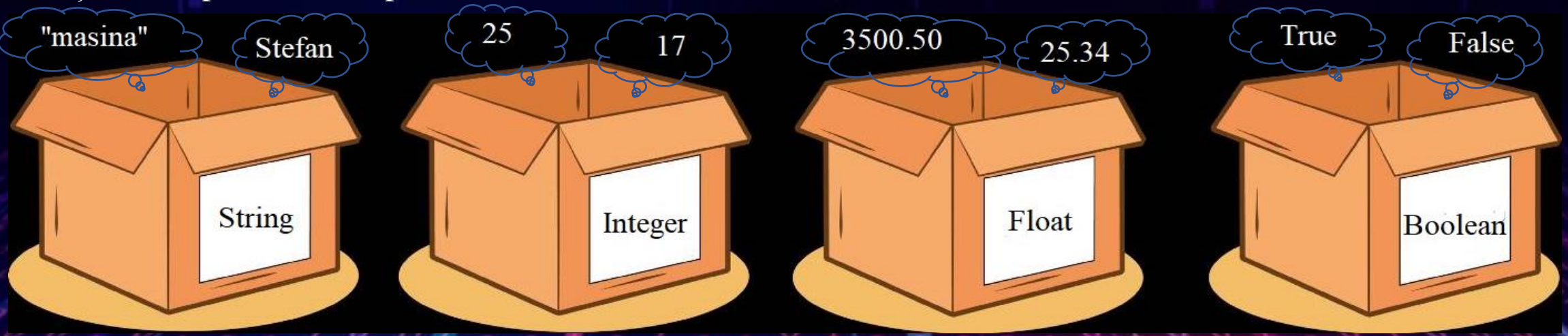
14-Octombrie-2023

Variabilele

- Variabilele putem să le privim ca pe niște cutii din memorie ce stochează diferite tipuri de valori (tipuri de date). Acestea sunt case sensitive, dețin un nume unic, încep cu literă mică, nu conțin spații în numele lor. Valoarea lor poate fi schimbată ulterior prin suprascriere.
- Tipuri de date primitive:

byte: poate lua valori în intervalul [-128, 127], are o memorie de 8-biți (1 byte); ***short***: ia valori în intervalul [-32.768, 32.767] (2 bytes); ***int***: poate lua valori în intervalul [-231, 231-1] (4 bytes); ***long***: ia valori în intervalul [-263, 263-1] (8 bytes); ***float***: are o precizie de 6 sau 7 zecimale (4 bytes); ***double***: are o precizie până la 15 sau 16 zecimale (8 bytes); ***boolean***: poate avea valorile true (1) sau false (0); ***char***: ia valori în intervalul [0, 65.535] (2 bytes).

- Pe lângă tipurile de date din imagine, există niște colecții, care sunt tot tipuri de date (list, dict, set, tuple).
- Funcția type() are o logică de cod predefinită ce ne returnează tipul de date al variabilei date. Funcția print() afișează ce punem între paranteze.



If else

- If-ul reprezintă dacă, iar condiția acestuia se pune în paranteză urmată de “:” ca apoi să fie indentarea.
- Aceasta delimitează de unde până unde ține blocul de cod al acelei condiții.
- Se folosește un if la început poate fi urmat de oricâte elif-uri sunt necesare, și un else (altfel) la final care se va executa automat dacă condițiile anterioare sunt false.

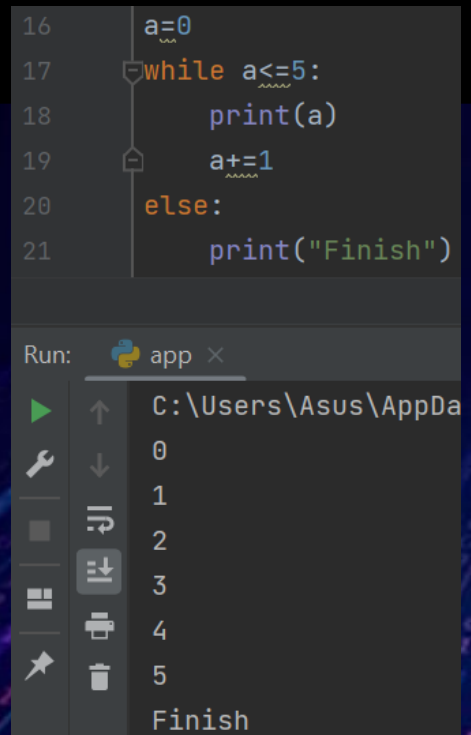
Listă, Dicționar, Set, Tuple

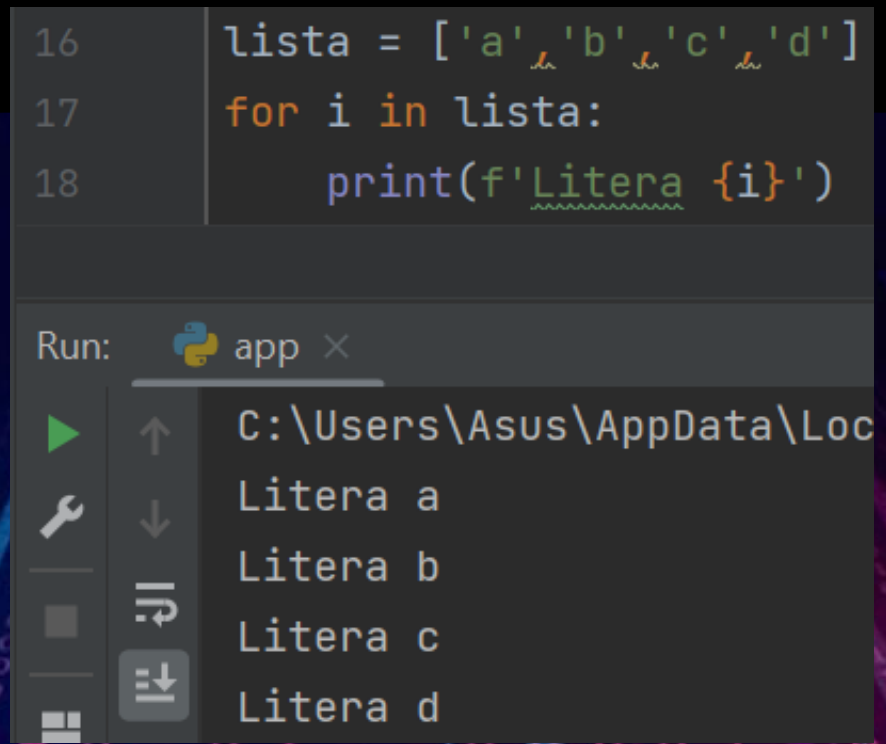
- Listele conțin mai multe valori doar într-o variabilă, având index care începe de la 0. Se pot modifica ștergând, adăugând valori sau modificând ordinea lor. Specific Python în liste se pot găsi diferite tipuri de date.
- Dicționarele au forma unor dicționare propriu-zise, în sensul că datele sunt scrise după un tipar specific: cheie-valoare. Cheile pot fi considerate porecle pentru index-uri, acestea fiind unice.
- Set-urile nu sunt ordonate sau indexate ca si listele, dicționarele și tuple-urile. De aceea conțin valori unice într-o variabilă, încât pot fi adăugate/șterse elemente, dar nu și schimbate ordinea acestora.
- Tuple-urile păstrează valori ce pot fi duplicate, care odată definite, așa rămân, neputând fi modificate în vreun fel.

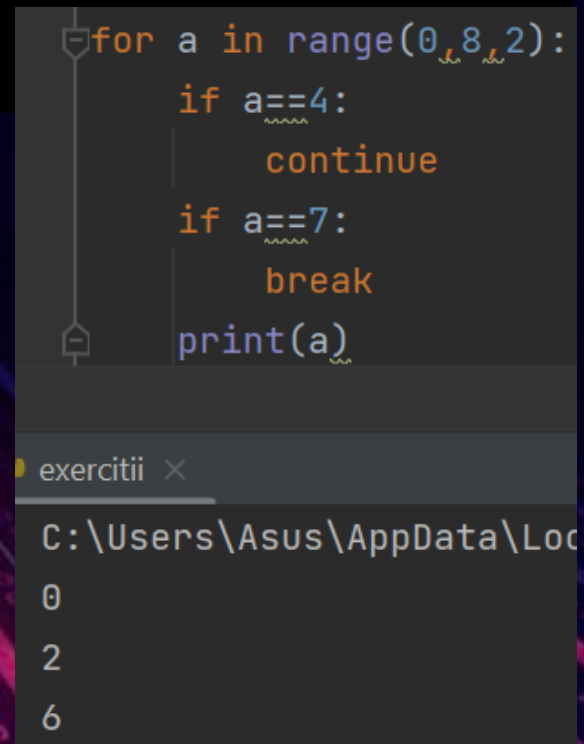
```python
lista = ['Stefan', 25, True, 'mere']
dictionar = {
    "nume": "Semelbauer",
    "prenume": "Stefan",
    "varsta": 21
}
set = {'luni', 'marti', 'miercuri'}
tuple = ("joi", "vineri", "sambata")
```

While, For, Break, Continue

- While reprezintă o buclă, cât timp condiția e adevărată se va repeta instrucțiunea dată.
- Instrucțiunea for este utilă în cazul în care dorim să executăm niște instrucțiuni de un număr fix de ori, spre deosebire de while, când codul se execută pe baza unei condiții. 0 = de unde; 8 = până unde; 2 = pasul.
- Continue îl putem considera ca fiind un skip ce sare peste condiția pusă sau peste blocul de cod de după acesta care face parte din while/for.
- Elementele dintr-o colecție vor fi „străbătute” până la break, ieșind din buclă, iar codul scris după break din cadrul unui while/for nu se va mai executa.







Funcții, Parametrii

- O funcție reprezintă niște linii de cod cu o utilitate anume, o scriem odată și putem să o folosim de câte ori ne dorim pe parcursul programului(shortcut).
- Parametrii reprezintă datele de intrare pentru funcție, aceștia sunt opționali, pot fi câți dorim noi, fiind despărțiți de virgulă. Îi putem privi ca pe niște variabile declarate, dar neinițializate, ce vor primi valori în momentul apelării funcției. Astfel, funcția poate rula pentru diferite persoane cu ajutorul parametrului care ajută la păstrarea utilizatorilor să fie distincți.

```python
# codul:
def Buna_seara():
    print("Buna seara!")
    print("Introduceti parola:")

# apelarea codului:
Buna_seara()
```

```
app ×
C:\Users\Asus\AppData\Local\Prog
Buna seara!
Introduceti parola:
```

```python
# codul:
def Buna_ziua(utilizator):
    print(f"Buna ziua domnule {utilizator}!")
    print(f"{utilizator} te rog sa introduci parola:")

# apelarea codului:
Buna_ziua("Stefan")
```

```
app ×
C:\Users\Asus\AppData\Local\Programs\Python\Python310\
Buna ziua domnule Stefan!
Stefan te rog sa introduci parola:
```

Clasă, Obiect, Constructor

- Clasa e un set de planuri(sau rețetă) după care pot fi contruite obiectele. Aceasta e alcătuită din atribute/fields(variabile) și metode(funcții). Self face ca funcția să poată accesa atributele clasei. Clasa poate fi utilizată de câte ori dorim.
- Fiecare obiect are memoria sa proprie unde se păstrează valorile tuturor datelor sale; Obiectul reprezintă o instanță a clasei. Toate obiectele au aceleași atribute și metode. Atributele pot fi modificate după inițializarea obiectului.
- Constructorul ajută la crearea obiectelor prin setarea unor date indispensabile acestora, adică oferă valori atributelor.

```python
# initializăm obiecte de tip Masina
masinutza1 = Masina()
masinutza2 = Masina()
print(masinutza1.model) # afiseză modelul masinutzei1
print(masinutza2.an) # afiseaza anul masinutzei2
masinutza1.model = "Octavia" # suprascriem valoarea initială pentru masinutza1
masinutza2.culoare = "albastră" # atribuire culoare masinutzei2
masinutza2.caldura() # porneste căldura masinutzei2
masinutza1.radio() # porneste radio-ul masinutzei1
```

```python
class Masina:
    # atribute:
    marca = "Skoda"
    model = None
    culoare = None
    an = None

    # contructorul
    def __init__(self, mdl, clr, an):
        self.culoare = clr
        self.model = mdl
        self.an = an

masinutza1 = Masina('Fabia','rosie',2020)
masinutza2 = Masina('Superb','albastră',2021)
print(masinutza2.an) # afiseză anul masinutzei2
print(masinutza1.model) # afiseză modelul masinutzei1
print(masinutza2.culoare) # afiseză culoarea masinutzei2
```

Cei 4 piloni OOP

- **Abstractizarea** e procesul prin care "ascundem" anumite funcționalități ale programului, adică nu e nevoie să cunoști acel cod în detaliu, ci doar să-i știi utilitatea ca să-l "apelezi" pentru a-l folosi. Clasele abstracte vor fi moștenite de clasele copil ce vor scrie logica metodelor.
- Prin intermediul **moștenirii**, în clasele derivate și nou formate poți reutiliza codul din supraclasele (clasele părinte) deja existente, fără a fi nevoie să-l rescrii. Astfel, economisești timp și, codul din program va fi mai scurt.

```python
from abc import ABC, abstractmethod
class Angajat(ABC): # supraclasa sau clasa părinte
    def ang_id(self,id,nume,varsta,salariu): # Abstractizare
        pass
class Angajat_pers1(Angajat):
    def ang_id(self,id):
        print("ang_id este 25")

ang1 = Angajat_pers1()
ang1.ang_id(id)
```

```
app
C:\Users\Asus\AppData\Local\Programs\Python\Python310\python.ex
ang_id este 25
```

```python
class Angajat(): # supraclasă sau clasă părinte
    def __init__(self,nume,varsta,salariu):
        self.nm = nume
        self.vrst = varsta
        self.slr = salariu

class Angajat_pers1(Angajat): # subclasă sau clasă copil
    def __init__(self,nume,varsta,salariu,id):
        self.nm = nume
        self.vrst = varsta
        self.slr = salariu
        self.id = id

ang1 = Angajat("Stefan",21,2500)
print(ang1.nm)
print(ang1.slr)
```

```
app
C:\Users\Asus\AppData\Local\Programs\Python\Python310\pytho
Ștefan
2500
```

Cei 4 piloni OOP

- Polimorfismul e procesul prin care putem să facem mai multe copii ale aceleiași metode ce poate să primească date de intrare diferite. Exemplu: o femeie poate fi, în același timp, soție, mamă, bunică, mătușă, angajată și fiică. Astfel, această persoană trebuie să aibă mai multe caracteristici, însă e nevoită să le implementeze pe fiecare în funcție de situație și condiții.

- Prin încapsulare creem o capsulă care delimitează interiorul de exterior, adică ținem datele și funcțiile separate de exterior.
- Variabilele de instanță sunt păstrate private (exemplu: username-uri și parole), iar metodele de dezvoltare sunt făcute publice.

```python
class Angajat(): # supraclasă sau clasă părinte
    def __init__(self,nume,varsta,id,salariu):
        self.nm = nume
        self.vrst = varsta
        self.id = id
        self.slr = salariu
    def earn(self):
        pass

class Angajat_pers1(Angajat):
    def earn(self): # Polimorfismul
        print("Angajatul 1 câștigă...")

class Angajat_pers2(Angajat):
    def earn(self): # Polimorfismul
        print("Angajatul 2 câștigă...")

a1 = Angajat_pers1
a2 = Angajat_pers2
a1.earn(Angajat)
a2.earn(Angajat)
```

```
app
C:\Users\Asus\AppData\Local\Programs\Python\
Angajatul 1 câștigă...
Angajatul 2 câștigă...
```

```python
class Individ:
    def __init__(self,nume,id):
        self.nm = nume
        self.id = id

    def afiseaza(self):
        print(self.nm)
        print(self.id)

ind1 = Individ("Stefan",2517)
ind1.afiseaza() # accesare folosind metodele clasei
print("=============")
print(ind1.nm) # accesare direct din exterior
print(ind1.id) # accesare direct din exterior
```

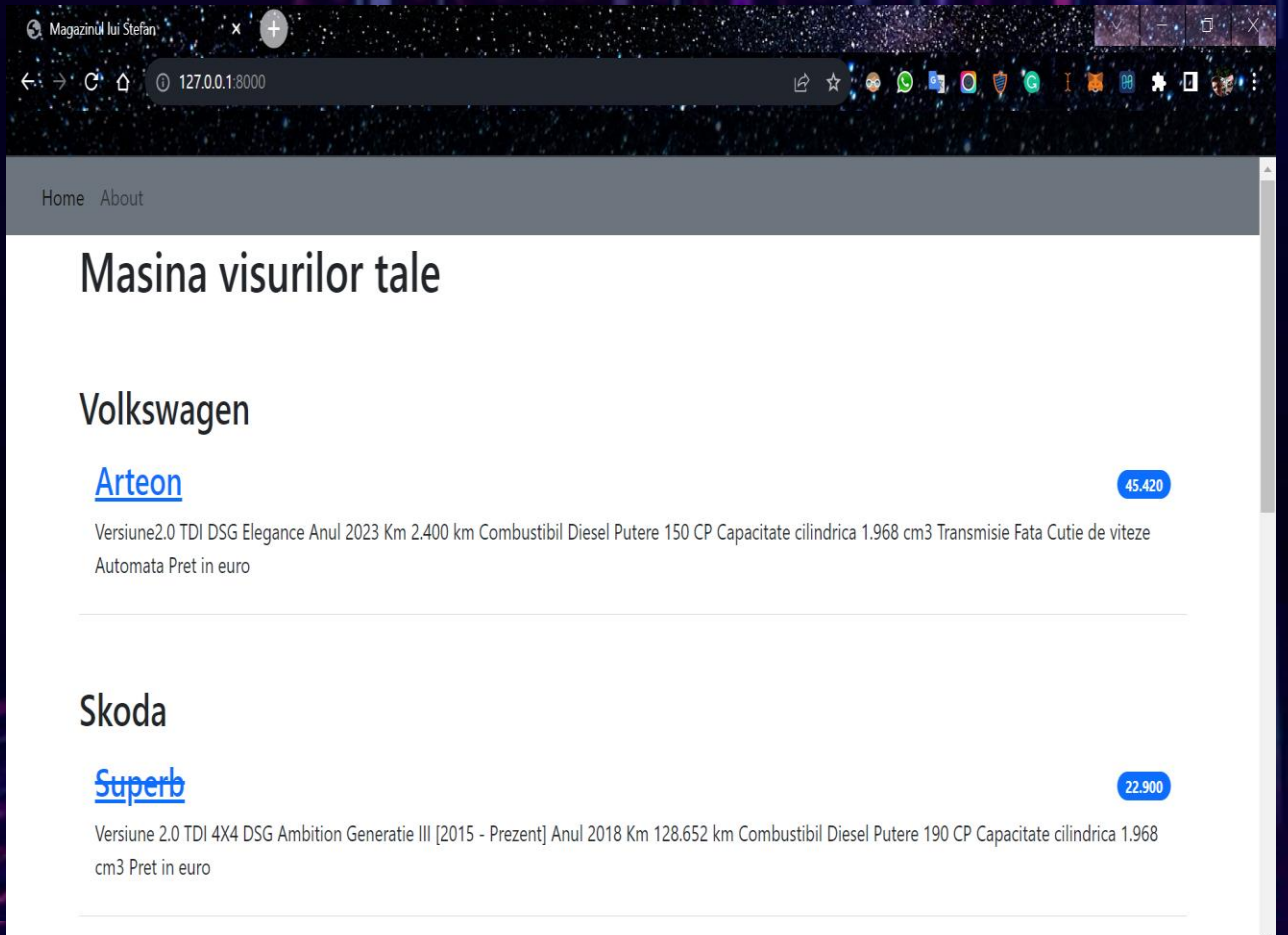
```
app
C:\Users\Asus\AppData\Local\Programs\Python\Python310\
Ștefan
2517
=============
Ștefan
2517
```

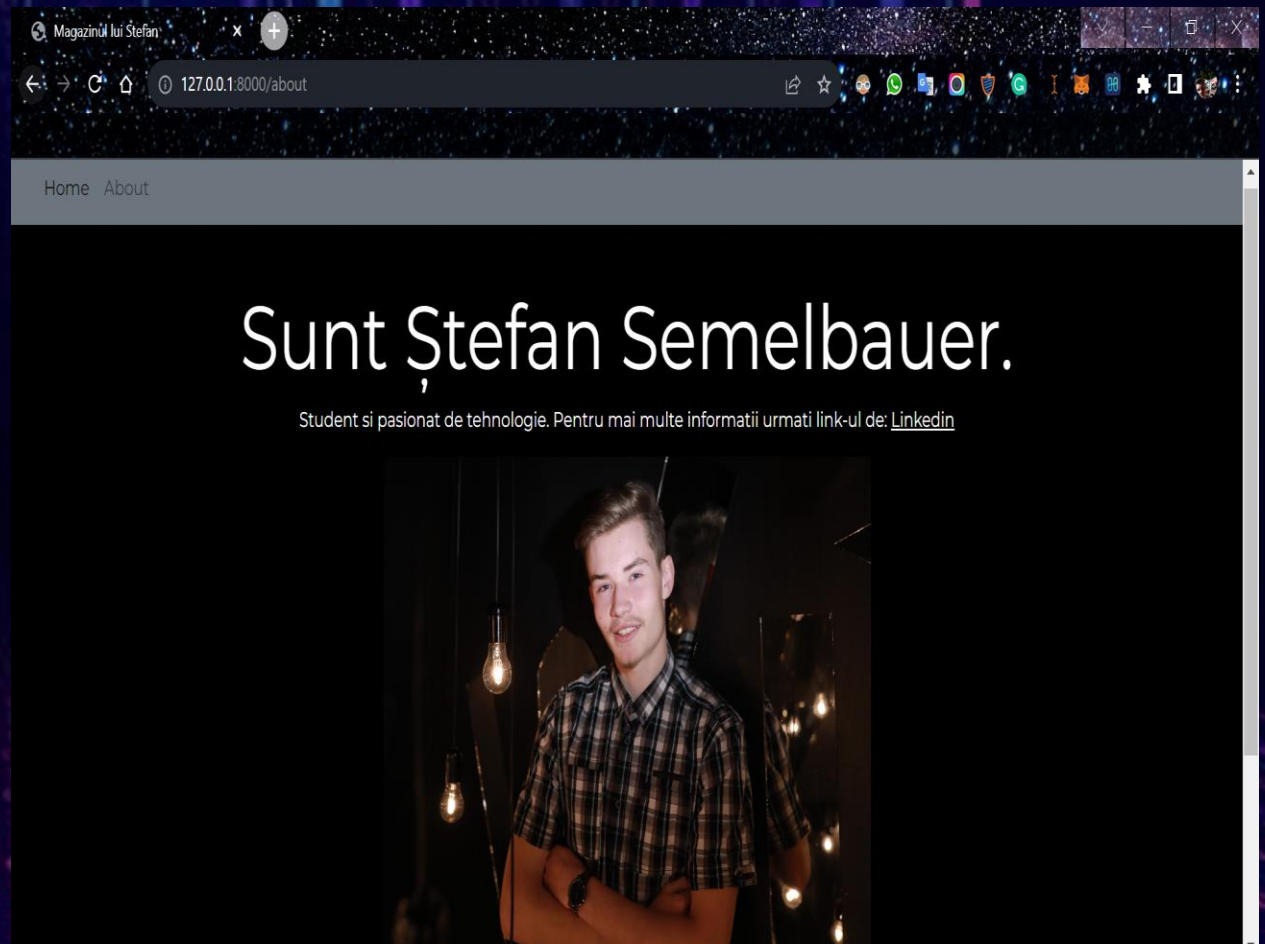
Instalarea și utilizarea aplicației

- Am dezvoltat o aplicație web cu python și HTML în mediul de dezvoltare Pycharm și pe care o găsiți în link ul: https://github.com/Stefan2517/App.car.marketplace.git

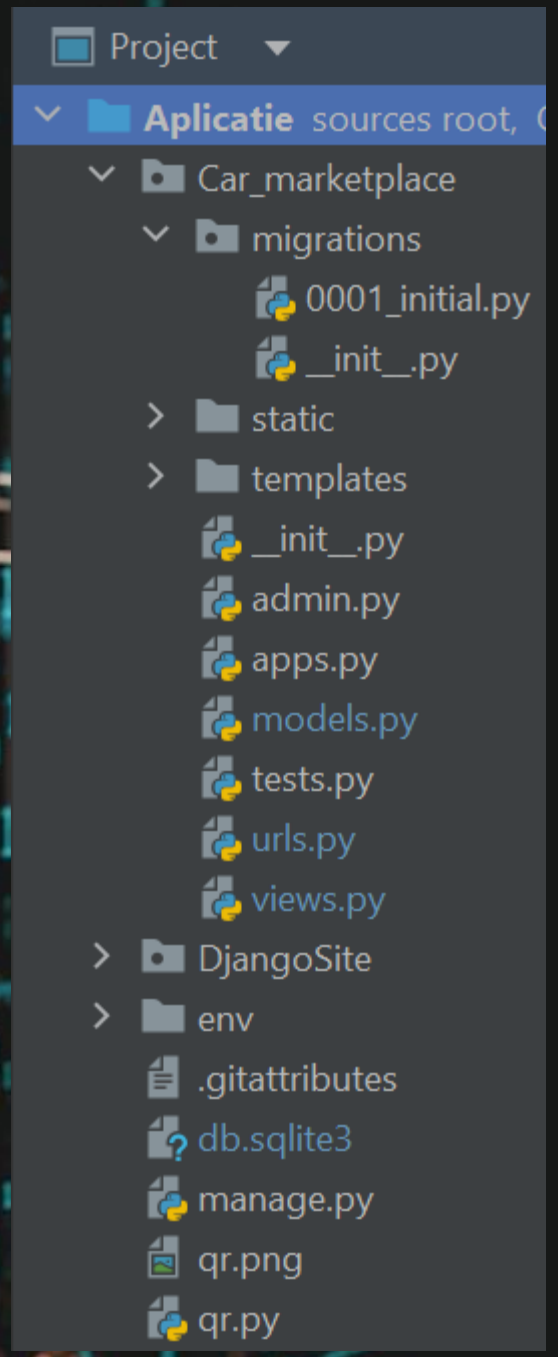
- Am utilizat framework-ul **Django**, alături de două librării: Qrcode și Pillow. Aceste două din urmă m-au ajutat în crearea codului Qr care să reprezinte adresa site-ului web cu scopul accesării acestuia într-un mod cât mai ușor.
- Pentru a rula proiectul trebuie descărcat de pe link-ul de GitHub. Apoi deschis în Pycharm. Ne vom ajuta de terminalul din Pycharm în care trebuie să scriem **python manage.py runserver**, apăsând **ENTER.** Ne va genera un link prin care putem accesa site-ul.

- Inițial se va deschide HomePage-ul aplicației în care va fi afișat o bară de meniu în partea de sus cu butoanele: **Home, About**. De asemenea vor fi regăsite diferite mașini în funcție de marcă, cu caracteristicile acestora: nume, starea disponibilității (în cazul în care e indisponibilă numele va fi ,,tăiat'' cu o linie), descriere, preț.
- Secțiunea de About am dorit să reprezinte câteva cuvinte despre mine, cât și link-ul meu de LinkedIn ca și contact.





- În realizarea aplicației, am avut la îndemână documentația de la Django.
- Inițial am creat un folder „Aplicație” , pe care l-am deschis în Pycharm. Apoi, am făcut un mediu virtual în care să dezvolt aplicația, utilizând: ***pip install virtualenv***. După care în terminal l-am activat cu: ***C:\Users\Asus\Desktop\Aplicatie\env\scripts\activate.bat*** . Am instalat django, qrcode, pillow cu: ***pip install*** (înaintea numelor lor). Am creat un proiect DjangoSite, folosind: ***django-admin startproject DjangoSite*** . De asemenea și aplicația Car_marketplace, cu: ***python manage.py startapp Car_marketplace*** .
- Pentru a verifica am rulat comanda: ***python manage.py runserver***. Cu ***python manage.py makemigrations*** creează un folder ***migrations*** în aplicație(Car_marketplace) cu 2 fișiere care conțin caracteristicile dorite: preț, descriere, stare, etc. Iar cu ***python manage.py migrate***, rulează prin cele 2 fișiere create anterior: ***0001_initial.py*** și ***_init_.py*** din ***migrations*** să fie transformat în database (fișierul ***db.sqlite3***).



- În ***Settings*** din proiectul DjangoSite, la installed_apps, adaug aplicația creată (Car_marketplace). Din Car_marketplace, în ***models***, scriu toate atributele obiectului. Completez ***views*** din Car_marketplace.
- Am creat folder-ul ***templates*** în folder-ul Aplicație, în care se vor găsi fișierele html. Am realizat fișierul ***urls.py*** din Car_marketplace. Apoi am completat ***urls.py*** din proiectul DjangoSite. Url-urile apelează views-urile care la rândul lor fișierele html.

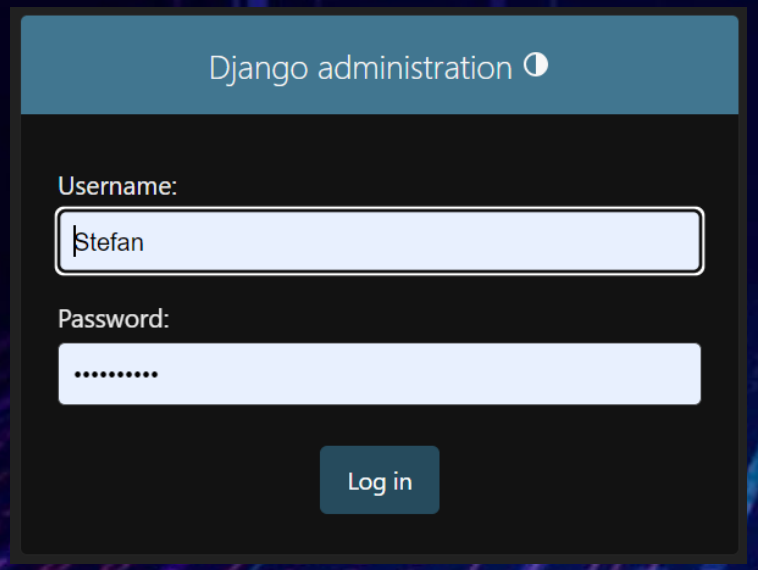
```
33    INSTALLED_APPS = [
34        'django.contrib.admin',
35        'django.contrib.auth',
36        'django.contrib.contenttypes',
37        'django.contrib.sessions',
38        'django.contrib.messages',
39        'django.contrib.staticfiles',
40        'Car_marketplace',
41    ]
```

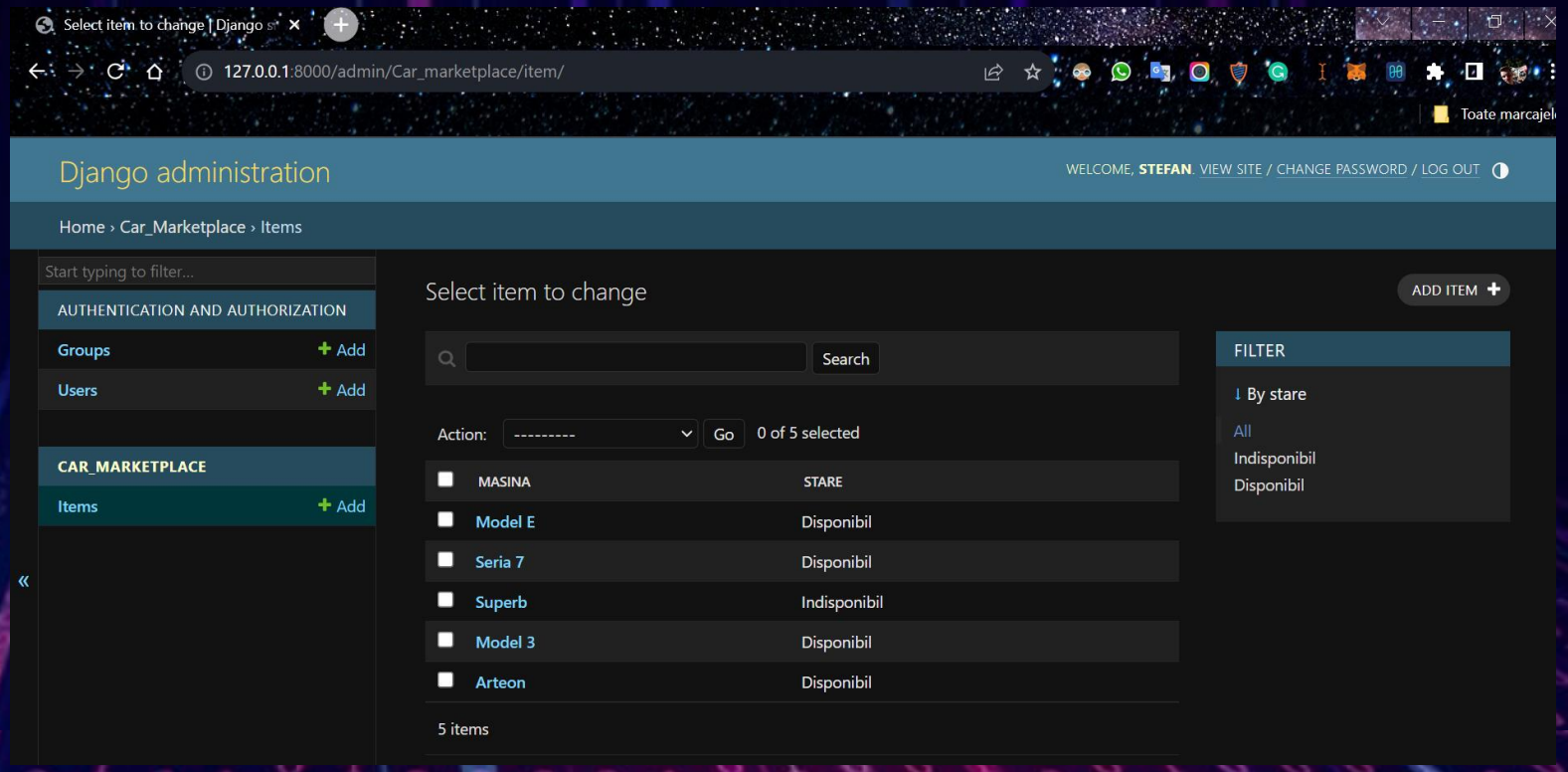
```
class PaginaPrincipala(generic.ListView):  # reprezinta pagina cu toate masinile adica cea principala
    queryset = Item.objects.order_by('-data_crearii') # minusul inverseaza ordinea
    template_name = 'index.html'

    def get_context_data(self, **kwargs):
        context = super().get_context_data(**kwargs)
        context['masini'] = CAR_BRAND
        return context


class MasinaDetalii(generic.DetailView):
    model = Item
    template_name = 'masina_detalii.html'

class About(TemplateView):

    template_name = 'about.html'
```

```
class Item(models.Model):
    masina = models.CharField(max_length=1000, unique=True)
    descriere = models.CharField(max_length=2000)
    pret = models.DecimalField(max_digits=10, decimal_places=3)
    marca_masina = models.CharField(max_length=200, choices=CAR_BRAND)
    autor = models.ForeignKey(User, on_delete=models.PROTECT) # foloses
    stare = models.IntegerField(choices=STARE, default=1)
    data_crearii = models.DateTimeField(auto_now_add=True)
    data_actualizarii = models.DateTimeField(auto_now=True)

    def __str__(self):
        return self.masina
```

➢ Am luat masina, descriere, stare din ***models*** în ***admin*** (ambele din Car_marketplace). Am cuplat câteva modele de element din ***models*** cu administratorul de elemente de meniu din ***admin***. Pentru crearea contului de admin trebuie să avem server-ul oprit, ne asigurăm că suntem pe mediul de dezvoltare virtuală (venv), și scriem: ***python manage.py createsuperuser***. Completăm cu user și parolă, apoi din acest meniu putem adăuga, șterge, cât și modifica articole(mașini).

```python
class MenuItemAdmin(admin.ModelAdmin):
    list_display = ('masina', 'stare')
    list_filter = ('stare',) # daca nu puneam virgula nu il lua ca si tuple
    search_fields = ('masina', 'descriere')

admin.site.register(Item, MenuItemAdmin)
```






Va multumesc pentru atentie!