**AMALIY MASHG‘ULOT UCHUN O‘QUV MATERIALLARI**

**1-Mavzu:** “Python dasturlash tili” faniga kirish va asosiy tushunchalari.

**13-mashg‘ulot**. Pythonda Vorislik tushunchasi.

**O‘quv savollari:**

1. Vorislik.
2. Maxsus metodlar.
3. Klass xususiyatlari.
4. Dekoratorlar.

**1. Python tilidagi “Merosxo‘rlik” tushunchasi, atributlari va xossalari**

OOP da merosxo‘rlik konsepsiyasi mavjud class asosida yangi class yaratish imkonini beradi. Inkapsulyatsiya bilan bir qatorda merosxorlik ob’yektga yo‘naltirilgan dasturlashning asoslaridan hisoblanadi.

Merosxorlik konsepsiyasining asosiy tushunchalari: ***subklass*** va ***superclass*** lardir. ***Subklass*** barcha umumiy atributlar va metodlarni yuqori classdan meros qilib oladi. Shuningdek superclass, asosiy (**base class**) yoki ota (**parent class**) deb atalsa, subclass - voris class (**derived class**) yoki bola class (**child class**) deb ataladi.

Merosxorlik classini yaratish sintaksisi quyidagicha ko‘rinishda bo‘ladi:

**class** subclass (superclass):

    subclass\_metodlari

Masalan, shaxsni ifodalovchi Person sinfi mavjud bo‘lsin:

**class** Person:

**def** \_\_init\_\_(**self**, nom):

**self**.\_\_nom = nom

@property

**def** nom(**self**):

**return** **self**.\_\_nom

**def** display\_info(**self**):

print(f"Nomi: {**self**.\_\_nom} ")

Aytaylik, bizga qandaydir korxonada ishlaydigan ishchilar classi kerak. Biz noldan yangi sinf yaratishimiz mumkin, masalan, *Employee* sinfi:

**class** Employee:

**def** \_\_init\_\_(**self**, nom):

**self**.\_\_nom = nom

@property

**def** nom(**self**):

**return** **self**.\_\_nom

**def** display\_info(**self**):

print(f"Nomi: {**self**.\_\_nom} ")

**def** work(**self**):

print(f"{**self**.nom} works")

*Biroq, Employee* classi Person classi bilan bir xil atribut va metodlarga ega bo‘lishi mumkin, chunki xodimlar ham inson classiga mansub ob’yeklar hisoblanadi. Shunday qilib, yuqoridagi ***Employee*** sinfida faqat ***work*** metodi qo‘shiladi va qolgan kod Person sinfining funksionalligidan nusxalab olinadi. Agar mersoxorlikda foydalanilmasa, bir xil funksiyanallik ikkala sinfda ham takrorlanaveradi. Bunday holda merosxorlik konsepsiyasidan foydalanish ancha kodni tejalishiga olib keladi.

Shunday qilib, ***Person*** sinfidan ***Employee*** sinfini meros qilib olaylik:

**class** Person:

**def** \_\_init\_\_(**self**, nom):

**self**.\_\_nom = nom

@property

**def** nom(**self**):

return **self**.\_\_nom

**def** display\_info(**self**):

print(f"Nomi: {**self**.\_\_nom} ")

**class** Employee(Person):

**def** work(**self**):

print(f"{**self**.nom} works")

tomson = Employee("Tomson")

print(tomson.nom) *# Tomson*

tomson.display\_info() *# Nomi: Tomson*

tomson.work() *# Tomson works*

*Employee* sinfi ***Person*** sinfining funksiyalarini to‘liq o‘ziga nusxalab oladi, faqat **work()** metodi *Employee* sinfiga qo‘shimcha tarzda qo‘shiladi. *Shunga ko‘ra, Employee* sinfi ob’yektini yaratishda ***Person*** dan meros qilib olingan konstruktordan foydalanish mumkin :

tomson = Employee ("Tomson")

Shuningdek, meros qilib olingan atributlar/ xususiyatlar va metodlarga murojaat qilish mumkin:

print(tomson.nom) *# Tomson*

tomson.display\_info() *# Nomi: Tomson*

Shuni yodda tutish kerakki, Employee sinfi uchun Person sinfining yopiq atributi bo‘lgan **\_\_nom** atributiga kirishga ruxsat yo‘q. Masalan, ***work()*** metodi tarkibida **self.\_\_nom** xususiy atributiga murojaat qila olmaymiz :

def work(self):

print(f"{self.\_\_nom} works") *# Xatolik!*

**Ko‘p merosxorlik**

Python tilining ajralib turadigan xususiyatlaridan biri - bu bir ko‘p merosxorlik konsepsiyasini qo‘llab-quvvatlashligidir. Ya’ni bitta sinf bir nechta sinflardan meros olishi mumkin:

**class** Employee:

**def** work(**self**):

print("Employee jobs")

**class** Students:

**def** study(**self**):

print("Students studying")

**class** WorkingStudents(Employee, Students):

pass

tomson = WorkingStudent()

tomson.job() *# Employee jobs*

tomson.study() *# Students studying*

U firma xodimini ifodalovchi Employee sinfini va talabalarini ifodalovchi  Studentssinfini belgilaydi*.*Ishlayotgan talabani ifodalovchiWorkingStudents sinfi  tarkibida hech qanday metod va atributlar aniqlanmaganligi pass operatori bilan to‘ldirib qo‘yilgan holos.

WorkingStudents sinfi ikkita sinfdan Employee va  Studentsfunksionallikni meros qilib oladi. Shunday ekan ushbu sinf ob’yektida ikkala sinfning metodlarini chaqirish imkoniyati mavjud.

Shu bilan birga, meros qilib olingan sinflar funktsional jihatdan murakkabroq bo‘lishi mumkin, masalan:

**class** Employee:

**def** \_\_init\_\_(**self**, nom):

**self**.\_\_nom = nom

@property

**def** nom(**self**):

**return** **self**.\_\_nom

**def** work(**self**):

print(f"{**self**.nom} works")

**class** Students:

**def** \_\_init\_\_(**self**, nom):

**self**.\_\_nom = nom

@property

**def** nom(**self**):

**return** **self**.\_\_nom

**def** study(**self**):

print(f"{**self**.nom} studying")

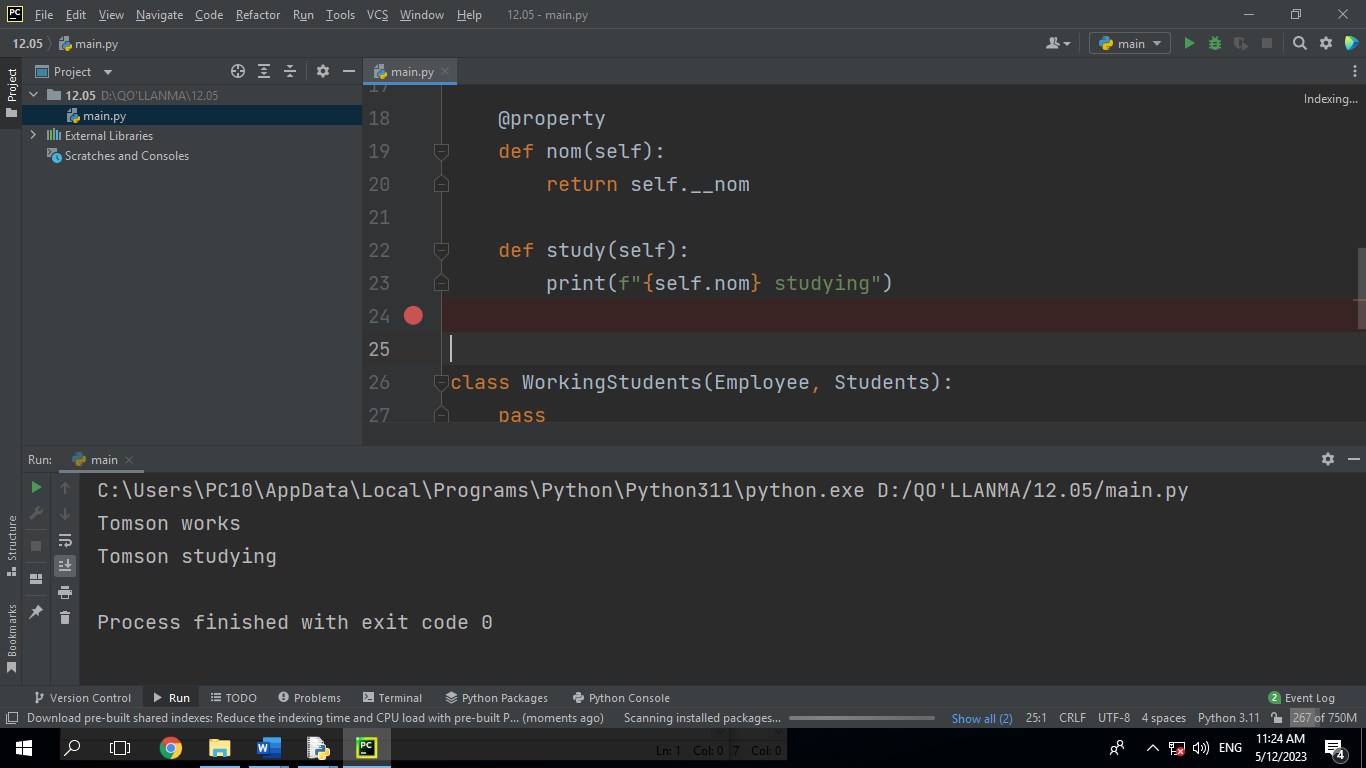
**class** WorkingStudents(Employee, Students):

**pass**

tomson = WorkingStudents("Tomson")

tomson.work()

tomson.study()



**2. Maxsus metodlar**

Pythonda obyektlar bilan ishlashni yanada qulay qilish uchun bir nechta maxsus metodlar bor. Bu metodlarning nomi ikki pastki chiziq bilan yozilgani uchun, double underscore yoki qisqa qilib dunder metodlar deb ataladi. Dunder metolar yordamida obyektlarga qo'shimcha qulayliklar va vazifalar qo'shishimiz mumkin. Klass yoki obyektga oid dunder metodlar ro'yxatini ko'rish uchun dir() funksiyasidan foydalanamiz:

>>> dir(Avto)

['\_Avto\_\_num\_avto',

'\_\_class\_\_',

'\_\_delattr\_\_',

'\_\_dict\_\_',

'\_\_dir\_\_',

'\_\_doc\_\_',

'\_\_eq\_\_',

'\_\_format\_\_',

'\_\_ge\_\_',

'\_\_getattribute\_\_',

'\_\_gt\_\_',

'\_\_hash\_\_',

'\_\_init\_\_',

'\_\_init\_subclass\_\_',

'\_\_le\_\_',

'\_\_lt\_\_',

'\_\_module\_\_',

'\_\_ne\_\_',

'\_\_new\_\_',

'\_\_reduce\_\_',

'\_\_reduce\_ex\_\_',

'\_\_repr\_\_',

'\_\_setattr\_\_',

'\_\_sizeof\_\_',

'\_\_str\_\_',

'\_\_subclasshook\_\_',

'\_\_weakref\_\_',

'make',

'model',

'narh',

'rang',

'yil']

Dunder metodlardan biz \_\_init\_\_ metodi bilan tanishdik. Bu metod klassdan obyekt yaratishda chaqiriladi va obyektning xususiyatlarini belgilaydi. Ushbu darsimizda esa maxsus metodlarning ba'zilari bilan tanishamiz.

**OBYEKT HAQIDA MA'LUMOT**

Obyektga print() yoki str() orqali murojat qilinganda obyekt haqida tushunarli ma'lumot qaytarish uchun \_\_repr\_\_va \_\_str\_\_ metodlaridan foydalanamiz. Tushunarli bo'lishi uchun avvalgi darsimizdagi Avto klassiga qaytamiz:

class Avto:

\_\_num\_avto = 0

"""Avtomobil klassi"""

def \_\_init\_\_(self,make,model,rang,yil,narh):

"""Avtomobilning xususiyatlari"""

self.make = make

self.model = model

self.rang = rang

self.yil = yil

self.narh = narh

Avto.\_\_num\_avto += 1

Yuqoridagi klassdan yangi obyekt yaratamiz va obyekt haqida ma'lumot olish uchun print() funksiyasini chaqiramiz:

avto1 = Avto("GM","Malibu","Qora",2020,40000)

print(avto1) # obyekt haqida ma'lumot

Natija: <\_\_main\_\_.Avto object at 0x00000238A6DAE0C8>

Qandaydur tushunarsiz ma'lumot. Ekrandagi natijadan biz faqat avto1 obyektimiz Avto klassiga tegishli ekanini ko'ramiz. Qanday qilib shuning o'rniga obyekt haqida tushunarliroq ma'lumot olishimiz mumkin?

Gap shundaki biz har gal obyketga print() (yoki str() yoki repr()) orqali murojat qilganimizda, Python obyket ichida \_\_str\_\_ yoki \_\_repr\_\_ metodlariga murojat qiladi. Agar biz bu metodlarni yozmagan bo'lsak, yuqoridagi kabi umumiy ma'lumot qayataradi.

Biz ushbu metodlarni yangidan yozib, biz istagan ma'lumotni qayataradian qilishimiz mumkin. Odatda, yuqoridagi ikki metoddan birini yozish kifoya. Odatda, \_\_repr\_\_ umumiyorq, \_\_str\_\_ esa batafsilroq ma'lumot olish uchun ishlatiladi.

Ikkalasidan birini tanlaganda, \_\_repr\_\_metodiga yon bosiladi, sababi bu metod print(), str() va repr() funksiyalarining hammasi bilan ishlaydi. Keling biz ham yuoqirdagi klassimizga\_\_repr\_\_metodini qo'shamiz:

class Avto:

\_\_num\_avto = 0

"""Avtomobil klassi"""

def \_\_init\_\_(self,make,model,rang,yil,narh):

"""Avtomobilning xususiyatlari"""

self.make = make

self.model = model

self.rang = rang

self.yil = yil

self.narh = narh

Avto.\_\_num\_avto += 1

def \_\_repr\_\_(self):

"""Obyekt haqida ma'lumot"""

return f"Avto: {self.rang} {self.make} {self.model}"

Qaytadan print() funksiyasini chaqiramiz:

avto1 = Avto("GM","Malibu","Qora",2020,40000)

print(avto1)

Natija: Avto: Qora GM Malibu

Mana endi natijamiz ancha tushunarli ko'rinishda chiqdi.

**SUPER KLASS VA VORIS KLASS**

Obyektga yo'naltirilgan dasturlashning qulayliklaridan biri bu klasslardan boshqa klass yaratish imkoniyati. Bizga kerak bo'lgan yangi klass, avval yaratilgan boshqa klass bilan o'xshashlik joylari bo'lsa, biz bu klassdan voris klass yaratishimiz mumkin. Bunda asl klass - ota, yoki super klass deb ataladi.

Super klassdan yaratilgan voris klass otaning barcha yoki tanlangan xususiyatlari va metodlarini meros olish bilan birga, o'ziga xos xususiyat va metodlariga ega bo'ladi.

Keling boshlanishiga Shaxs klassini yaratamiz, bu klassimiz keyinchalik boshqa klasslar uchun super klass vazifasini bajaradi:

class Shaxs:

"""Shaxslar haqida ma'lumot"""

def \_\_init\_\_(self,ism,familiya,passport,tyil):

self.ism = ism

self.familiya = familiya

self.passport = passport

self.tyil = tyil

def get\_info(self):

"""Shaxs haqida ma'lumot"""

info = f"{self.ism} {self.familiya}. "

info += f"Passport:{self.passport}, {self.tyil}-yilda tug`ilgan"

return info

def get\_age(self,yil):

"""Shaxsning yoshini qaytaruvchi metod"""

return yil - self.tyil

Klassimizni tekshirib ko'ramiz:

inson = Shaxs("Hasan","Alimov","FB001122",1995)

print(f"{inson.get\_info()}. {inson.get\_age(2021)} yoshda.")

Natija: Hasan Alimov. Passport:FB001122, 1995-yilda tug`ilgan. 26 yoshda.

**VORIS KLASS YARATISH**

Endi avvalgi darsimizda yaratgan Talaba klassimizni qaytadan yaratamiz. Bu safar, avvalgidan farqli ravishda, Talaba ni yaratishda, Shaxs dan super klass sifatida foydalanamiz:

class Talaba(Shaxs):

"""Talaba klassi"""

def \_\_init\_\_(self, ism, familiya, passport, tyil):

"""Talabaning xususiyatlari"""

super().\_\_init\_\_(ism, familiya, passport, tyil)

Kodimizni tahlil qilaylik:

1-qatorda klass nomidan so'ng, qavs ichida super klass nomini berdik

5-qatorda (def \_\_init\_\_ ichida) klassimiz super klassning xususiyatlarini meros olishini ko'rsatdik

Yangi yaratgan Talaba klassimiz Shaxsning barcha xususiyatlari va metodlariga ega bo'ladi.

talaba = Talaba("Valijon","Aliyev","FA112299",2000)

print(talaba.get\_info())

Natija: Valijon Aliyev. Passport:FA112299, 2000-yilda tug`ilgan

Talaba klassi uchun alohida get\_info() metodini yozmagan bo'lsakda, bu metod Talabaga Shaxsdan meros o'tdi.

Huddi shu kabi get\_age() metodiga ham murojat qilishimiz mumkin:

>>>print(talaba.get\_age(2021))

Dastur davomida super klass voris klasslardan avval yozilgan (chaqirilgan) bo'lishi kerak.

**VORIS KLASSGA XOS XUSUSIYATLAR VA METODLAR**

Hozirgi ko'rinishda Talaba va Shaxs klasslari o'rtasida hech qanday farq yo'q. Keling Talaba klassimizga o'ziga xos xususiyatlar va metodlar yarataylik. Avvalosiga, talabaning bosqichi va ID raqamini xususiyat sifatida qo'shamiz. Bunda ID raqami obyekt yaratilishida parameter sifatida uzatiladi, bosqich esa standart qiymatga ega.

class Talaba(Shaxs):

"""Talaba klassi"""

def \_\_init\_\_(self, ism, familiya, passport, tyil,idraqam):

"""Talabaning xususiyatlari"""

super().\_\_init\_\_(ism, familiya, passport, tyil)

self.idraqam = idraqam

self.bosqich = 1

Endi yangi, Talaba obyektini yaratishda qo'shimcha idraqam parametrini ham kiritish talab qilinadi:

Copy

talaba = Talaba("Valijon","Aliyev","FA112299",2000,"0000012")

So'ngra, bu qiymatlarni qaytaruvchi alohida metodlar yozamiz:

class Talaba(Shaxs):

"""Talaba klassi"""

def \_\_init\_\_(self, ism, familiya, passport, tyil,idraqam):

"""Talabaning xususiyatlari"""

super().\_\_init\_\_(ism, familiya, passport, tyil)

self.idraqam = idraqam

self.bosqich = 1

def get\_id(self):

"""Talabaning ID raqami"""

return self.idraqam

def get\_bosqich(self):

"""Talabaning o'qish bosqichi"""

return self.bosqich

Metodlarni tekshirib ko'ramiz:

>>>print(f"{talaba.get\_info()}. ID raqami:{talaba.get\_id()}")

Valijon Aliyev. Passport:FA112299, 2000-yilda tug`ilgan. ID raqami:0000012

>>>print(f"{talaba.get\_bosqich()}-bosqich talabasi")

1-bosqich talabasi

Shu zayilda yangi klassimizga istalgancha yangi xususiyatlar va metodlar qo'shishimiz mumkin. Bunda, agar yangi xususiyat yoki metod super klassga ham aloqador bo'lsa uni birdan super klassga qo'shish tavsiya qilinadi.

Voris klass boshqa klass uchun super klass bo'lishi mumkin.

**POLIMORFIZM — SUPER KLASS METODLARINI QAYTA YOZISH**

Voris klassga super klassdan meros qolgan istalgan metodni qayta talqin qilish mumkin. Avvalgi misolimizdagi get\_info() super metodini ko'raylik, bu metod talabaning ismi, familiyasi, passport raqami va tug'ilgan yilini qaytaradi:

>>> print(talaba.get\_info())

Valijon Aliyev. Passport:FA112299, 2000-yilda tug`ilgan

Endiget\_info() metodi talabaga mos ma'lumotlar qaytarishi uchun, Talaba klassi ichida huddi shu nomli metodni qayta yozamiz:

class Talaba(Shaxs):

"""Talaba klassi"""

def \_\_init\_\_(self,ism,familiya,passport,tyil,idraqam):

"""Talabaning xususiyatlari"""

super().\_\_init\_\_(ism, familiya, passport, tyil)

self.idraqam = idraqam

self.bosqich = 1

def get\_id(self):

"""Talabaning ID raqami"""

return self.idraqam

def get\_bosqich(self):

"""Talabaning o'qish bosqichi"""

return self.bosqich

def get\_info(self):

"""Talaba haqida ma'lumot"""

info = f"{self.ism} {self.familiya}. "

info += f"{self.get\_bosqich()}-bosqich. ID raqami: {self.idraqam}"

return info

Metodni tekshirib ko'ramiz:

>>> print(talaba.get\_info())

Valijon Aliyev. 1-bosqich. ID raqami: 0000012

**OBYEKT ICHIDA OBYEKT**

Ba'zida klassimiz xususiyatlar va ular bilan ishlaydigan metodlarga to'lib ketishi, bu esa o'z navbatida obyektga murojat qilishni qiyinlashitirishi mumkin. Shunday holatlarda ba'zi xususiyatlarni alohida klass ko'rinishida yozish va keyinchalik bu klassdan yaratilgan obyektni boshqa obyektning xususiyati sifatida foydalanish mumkin.

Misol uchun, yuqoridagi Shaxs klassimizga yana bir manzil degan xususiyat qo'shaylik. Odatda manzil bir nechta qismlardan iborat bo'ladi (xonadon, ko'cha, mahalla, tuman/shahar, viloyat, indeks va hokazo) va ularning har biri uchun Shaxs ichida alohida xususiyat yaratmasdan, alohida manzil degan klassga yuklash maqsadga muvofiq bo'ladi.

class Manzil:

"""Manzil saqlash uchun klass"""

def \_\_init\_\_(self,uy,kocha,tuman,viloyat):

"""Manzil xususiyatlari"""

self.uy = uy

self.kocha = kocha

self.tuman = tuman

self.viloyat = viloyat

def get\_manzil(self):

"""Manzilni ko'rish"""

manzil = f"{self.viloyat} viloyati, {self.tuman} tumani, "

manzil += f"{self.kocha} ko'chasi, {self.uy}-uy"

return manzil

Talaba klassimizga ham qo'shimcha manzil xususiyatini qo'shamiz:

class Talaba(Shaxs):

"""Talaba klassi"""

def \_\_init\_\_(self,ism,familiya,passport,tyil,idraqam,manzil):

"""Talabaning xususiyatlari"""

super().\_\_init\_\_(ism, familiya, passport, tyil)

self.idraqam = idraqam

self.bosqich = 1

self.manzil = manzil

def get\_id(self):

"""Talabaning ID raqami"""

return self.idraqam

def get\_bosqich(self):

"""Talabaning o'qish bosqichi"""

return self.bosqich

def get\_info(self):

"""Talaba haqida ma'lumot"""

info = f"{self.ism} {self.familiya}. "

info += f"{self.get\_bosqich()}-bosqich. ID raqami: {self.idraqam}"

return info

Keling endi talaba obyektini qayta yaratamiz. Bu safar talabaning manzili ham alohida obyekt sifatida talaba ga uzatiladi:

talaba\_manzil = Manzil(12,'Olmazor',"Bog'bon","Samarqand")

talaba = Talaba("Valijon","Aliyev","FA112299",2000,"0000012",talaba\_manzil)

Obyekt ichidagi obyektning xususiyatlari va metodlariga ham avvalgidek nuqta orqali murojat qilishimiz mumkin:

>>> print(talaba.manzil.get\_manzil())

Samarqand viloyati, Bog'bon tumani, Olmazor ko'chasi, 12-uy

>>> print(talaba.manzil.tuman)

**Amaliyot uchun masalalar**

1. Talaba klassiga yana bir, fanlar degan xususiyat qo'shing. Bu xususiyat parametr sifatida uzatilmasin va obyekt yaratilganida bo'sh ro'yxatdan iborat bo'lsin (self.fanlar=[])
2. Fan degan yangi klass yarating va bu klassdan turli fanlar uchun alohida obyektlar yarating.
3. Talaba klassiga fanga\_yozil() degan yangi metod yozing. Bu metod parametr sifatida Fan klassiga tegishli obyektlarni qabul qilib olsin va talabaning fanlar ro'yxatiga qo'shib qo'ysin.
4. Talabaning fanlari ro'yxatidan biror fanni o'chirib tashlash uchun remove\_fan() metodini yozing. Agar bu metodga ro'yxatda yo'q fan uzatilsa "Siz bu fanga yozilmagansiz" xabarini qaytarsin.
5. Yuqoridagi Shaxs klassidan boshqa turli voris klasslar yaratib ko'ring (masalan Professor, Foydalanuvchi, Sotuvchi, Mijoz va hokazo)
6. Har bir klassga o'ziga hoz xususiyatlar va metodlar yuklang.
7. Barcha yangi klasslar uchun get\_info() metodini qayta yozib chiqing.
8. Voris klasslardan yana boshqa voris klass yaratib ko'ring. Misol uchun Foydalanuvchi klassidan Admin klassini yarating.
9. Admin klassiga foydalanuvchida yo'q yangi metodlar yozing, masalan, ban\_user() metodi konsolga "Foydalanuvchi bloklandi" degan matn chiqarsin.

**Nazorat savollari:**

1. Merosxorlik tushunchasiga tarif bering;
2. Pythonda merosxorlik qanday amalga oshiriladi?
3. Inkopsulyatsiya nima maqsadda ishlatiladi?
4. Merosxor classning qanday afzalliklari mavjud?
5. Subclass va Superclass tushunchasiga tarif bering.