Python Object Oriented Programming(Obyekt Yönümlü Proqramlaşdırma)

May 23, 2022

0.1 Python Object Oriented Programming(Obyekt Yönümlü Proqramlaşdırma)

əvvəlki keçdiyimiz mövzulardan fərqli olaraq bu bəhsdə yeni mövzumuz obyektyönümlü proqramlaşdırmaya giriş edirik.Biz funksiyalar modullar bəhsini öyrənmişdik.Bilirikki ən önəmli konsepsiyalardan biri də funksiyalar idi.Və funksiyalar pythonda birinci sinif vətəndaş olaraq adlandırılırdı.Hər nümunələrdə nəqədər işimizi asanlaşdırdığının şahidi olurduq.Bəs OOP nədir!OOP əsasən yazdığımız funksiyaları toplu şəkildə bir obyektə sığdıraraq,obyektin daxilində saxlamasını,kənar müdaxilələrdən qorumasını inkapsulyasiya (encapsulation) edən varlıqlardır.Və qeyd edimki burda obyekt ifadəsi,dil daxilində ifadə etdiyimiz var olan maddə(vəya maddələr) anlayışını daşıyır.Misal olaraq funksiyada avtomobilin xüsusiyyətlərini tərtib edirik,bizim qeyd etdiyimiz avtomobil obyekt sayılır,xüsusiyyətləri isə həmin obyekti ifadə edən əlamətlərdir.(markası,avtomobilin gücü,rəngi,mühərrikin tipi kimi predmetlər avtomobili var edən xüsusiyyətlərdir)

Obyekt yönümlü proqramlaşdırmanın əsasını iki aspekt təşkil edirki 1-ci sinif(class) 2-ci isə obyekt(object) dir.Obyekt əsas varlığı ifadə edir.Və obyektlər üçün hazırladığımız funksiyalara sahib ola bilərki,bu funksiyalar obyektin metodları adlanır.verilənlərin tiplərində istifadə etdiyimiz metodlar(siyahılarda append metodu) buna misal ola bilər.

Python dilində siniflərin hazırlanması,class açar sözü ilə başlayır,funksiyalarda def ifadəsindən istifadə etdiyimiz kimi,siniflərdə də class ifadəsindən istifadə edəcəyik

[2]:
```
print("""

    class NameClass:
        <statement-1>
        .
        .
        .
        <statement-N>
""")
```

```
class NameClass:
    <statement-1>
    .
    .
    .
```

```
        <statement-N>
```

ümumi halda yuxarıdakı formada ifadə edilir.Asta-asta məsələnin tam məğzini anlamağa çalışaq

[3]:
```
print("""
     class NameClass:
         .............

     və ya

     class NameClass():
         .............

    kimi yaza bilərik

""")
```

```
     class NameClass:
         …

     və ya

     class NameClass():
         …

    kimi yaza bilərik
```

ilk sinfi hazırlayaq

[4]:
```
class Vehicle():
    """docstring"""
```

Yuxarıdakı nümunədə Vehicle adlı sinif hazırladıq

Funksiyalarda olduğu kimi digər obyektə mənimsədə bilərik

[5]:
```
avtomobil = Vehicle()
```

[6]:
```
print(avtomobil)
```

```
<__main__.Vehicle object at 0x000001DA4C17E978>
```

və sinfi avtomobil ifadəsinə mənimsətdik,ardından print() funksiyası ilə çap etdik.və **main** əsas obyekt olması ilə yanaşı sinfin ünvanını da bizə göstərdi

[7]:
```
class Vehicle():
    """docstring"""
    _car = 'Bentley'
    _enginetype = 'Burmayster(B&W)'
    _color = 'Black'
    _cylinder = '3.5lt'
```

avtomobilin markası, rəngi, silindirin həcmi və mühərrikin tipi kimi məlumatları əlavə etdik

[8]:
```
avtomobil = Vehicle()
```

[9]:
```
print(avtomobil)
```

```
<__main__.Vehicle object at 0x000001DA4C1FB0F0>
```

Bəs sinif daxilində qeyd etdiyimiz ifadələri necə əldə edə bilərik!

[10]:
```
avtomobil._car
```

[10]:
```
'Bentley'
```

[11]:
```
avtomobil._color
```

[11]:
```
'Black'
```

[12]:
```
avtomobil._enginetype
```

[12]:
```
'Burmayster(B&W)'
```

Hazırladığımız obyekt daxilindəki dəyişənlərə kənardan müdaxilələr də edə bilərik

[13]:
```
avtomobil._color='Red'
```

[14]:
```
print(avtomobil._color)
```

```
Red
```

[15]:
```
avtomobil._enginetype='MAN'
```

[16]:
```
print(avtomobil._enginetype)
```

```
MAN
```

Yuxarıda öncədən rəng avtomobil markası kimi dəyişənləri qeyd etdik və sonrada dəyişə bilirik,amma biz funksiyalarda olduğu kimi obyektə birbaşa arqumentlərimizi necə əlavə edə biləcəyik!

[17]:
```
def funk(a,b):
    return ('Cavab:',a+b)
```

[18]:
```
result = funk(3,4)
```

[19]:
```
print(result)
```

```
('Cavab:', 7)
```

[20]:
```
class Vehicle():
    """docstring"""
    _car = 'Bentley'
    _enginetype = 'Burmayster(B&W)'
    _color = 'Black'
    _cylinder = '3.5lt'
```

[21]:
```
avtomobil = Vehicle()
```

[22]:
```
avtomobil('Mercedes','Catarpillar','Green',3.5)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-22-b3956bdfade2> in <module>()
----> 1 avtomobil('Mercedes','Catarpillar','Green',3.5)

TypeError: 'Vehicle' object is not callable
```

Və xəta aldıq

[23]:
```
#dir funksiyası ilə avtomobil sinfinin hansı metodlardan ibarət olduğunu öyrənək
dir(avtomobil)
```

[23]:
```
['__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__',
'_car',
'_color',
'_cylinder',
'_enginetype']
```

Sondan 4-lükdə bizim qeyd etdiyimiz dəyişənlər yer alır.Yuxarıda qeyd olunanlar isə siniflərin təməl metodlarıdır.Biz həm təməl metodları həm də obyekt üçün digər metodlar hazırlayacağıq.Və problemə köklənib ilk tərtibatçı metodumuza müraciət edək

init metodu

metod siniflər üçün tərtibatçı metod olub,obyekt qeyd olunduqdan sonra öz funksionallığını təmin etməyə başlayır.Bu metod həmçinin konstruktor olaraq da adlandırılır.Biz bu metodu qeyd etməsək belə,python bunu avtomatik tərtib edir.konstruktor sinfin inşaedicisidir.

[24]:
```
class Vehicle():

    """
    _car = 'Bentley'
    _enginetype = 'Burmayster(B&W)'
    _color = 'Black'
    _cylinder = '3.5lt'
    """
    # __init__ metodunu istifadə etmək məcburiyyətindəyik
    def __init__(self,car,engine,color,cylinder):
        self.car = car
        self.engine = engine
        self.color = color
        self.cylinder = cylinder
```

və sinfi tərtib edib " **init** " metodundan istifadə etdik.init metodu daxilində self ifadəsi həmin obyektin parametrlərinin global dəyərlər olduğuna dair işarədir.Yəni həmin parametrləri kənardan əldə etmək imkanımız var.

və ardından sinfimizi obyektə mənimsədib çağıraq

[25]:
```
avtomobil = Vehicle()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-25-d6aea6fdc9f4> in <module>()
----> 1 avtomobil = Vehicle()

TypeError: __init__() missing 4 required positional arguments: 'car', 'engine',
 ↪'color', and 'cylinder'
```

xəta aldıq.Çünki 4 ədəd arqument aldığı halda biz heç birini əlavə etmədik.yuxarıdakı xəta, aşağıdakı xəta ilə eyni mənanı kəsb edir

[553]:
```
def func(a,b):
    return a**b
```

[554]:
```
func()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-554-bd1982955a12> in <module>()
----> 1 func()

TypeError: func() missing 2 required positional arguments: 'a' and 'b'
```

[555]:
```
avtomobil = Vehicle('MAN','Catarpillar','Black',3.5)
```

[556]:
```
print(avtomobil.car)
```

```
MAN
```

[557]:
```
print(avtomobil.engine)
```

```
Catarpillar
```

gər diqqət etdinizsə ilk başda hazırladığımız sinif,əlavə etdiyimiz rəng tipi marka və sairə,avtomobil ifadəsinə mənimsətdikdən sonra print(avtomobil._car) avtomobildən sonra noqtə qoyub tab düyməsini basdıqda(python shell ə xas xüsusiyyətdir) heç bir metod qarşımıza çıxmırdı.Amma indi avtomobil ifadəsindən sonra nöqtə qoyduqda color,engine və sairə kimi ifadələr qarşımıza çıxdı.Biz var olan **init** metodunu , Vehicle sinfinə tətbiq etdik.

[558]:
```
avtomobil = Vehicle('Mercedes','MAK','Black',5.5)
```

[559]:
```
print(avtomobil.cylinder)
```

```
5.5
```

[560]:
```
print(avtomobil.engine)
```

MAK

Yuxarıdakı sinfimizə dəyişiklik edib marka silindir rəng tiplərini dəyişdirə bildik.Biz dəyişikliyimizi metod hazırlayaraq edə bilərikmi!Bəli,ən faydalı üsullardan biridir.

Yeni bir sinif hazırlayaq

[561]:
```
class Person():
    """docstring"""
    def __init__(self,ad,soyad,poçt,telno):
        self.name = ad
        self.surname = soyad
        self.email = poçt
        self.mobile = telno

    def _Set(self,name,surname,email,mobile):
        self.name=name
        self.surname = surname
        self.email = email
        self.mobile = mobile
```

[562]:
```
person = Person('Nicat','Məmmədov','example@gmail.com','+99455100')
```

[563]:
```
print(person.name)
```

```
Nicat
```

[564]:
```
print(person.mobile)
```

```
+99455100
```

[565]:
```
person._Set('Vüsalə','Xanlarova','example@gmail.com','+99455400')
```

[566]:
```
print(person.name)
```

```
Vüsalə
```

[567]:
```
print(person.surname)
```

```
Xanlarova
```

[568]:
```
print(person.email)
```

```
example@gmail.com
```

[569]:
```
print(person.mobile)
```

```
+99455400
```

[ ]:

0.2 Inheritance (varislik mexanizmi) mövzusu

Varislik mexanizmi nədir gəlin bu ifadəni ətraflı nəzəri olaraq öyrənək.Fərz edəkki şirkətə sahibiy və şirkətimiz müdriyyət müavin olmaqla digər işçilərdən ibarətdir.Varislik mexanizmində bir obyekt atributları ilə digər obyekt arasında əlaqənin olması vəziyyətində istifadə olunur.Ehtiyyac olmadıqda isə class sinfimiz enkapsulyasiya mexanizmində olaraq qalır.Mexanizm təkrar kod yazmaq yerinə varislik mexanizmindən istifadə edilərək hər dəfə istifadəçidən işçilər üçün ad soyad almaq, həmçinin müdriyyət üçün ayrı bir metod və ya funksiya hazırlamaq yerinə,ad soyad poçt ünvanı kimi atributları aparıcı sinfə əlavə edib daha sonrakı siniflərdə çağrılaraq metod və atributları istifadə edə bilərik.Həmçinin biz bu mövzu ilə əlaqəli olan Polimorfizm(polymorphism metodikası) anlayışına da toxunacağıq.Polimorfizm, aparıcı sinfə aid metod (və ya metodların) varislik mexanizmi ilə tərtib olunmuş sinifdə özünün davranışlarını başqacür aparmasıdır.

[570]:
```
class Company():
    """docstring"""

    def __init__(self,fullname,email,mobile,salary,employment):
        self.fullname = fullname
        self.email = email
        self.mobile = mobile
        self.salary = salary
        self.employment = employment
    def Info(self):
        print("""
        Person:{}
        Email :{}
        Mobile:{}
        Salary(per month):{}
        Employment:{}
        """.format(self.fullname,self.email,self.mobile,self.salary,self.
 ↪employment))
```

[571]:
```
company = Company('Nurlan  liyev','example@gmail.com','+99455100',2000,['Senior␣
 ↪PHP Developer'])
```

[572]:
```
company.Info()
```

```
        Person:Nurlan  liyev
        Email :example@gmail.com
        Mobile:+99455100
        Salary(per month):2000
        Employment:['Senior PHP Developer']
```

Yuxarıdakı kodlarımızda Company sinfi hazırladıq.Və təyin etdiyimiz atributlar; istifadəçidən alaraq,İnfo metodu vasitəsilə ekrana çap edirik.Biz sektorda işçilər ilə müdriyyəti eyni qrupa aid edə bilmərik.Ümumi işçilər arasında fərqləndirməni dəyərləndirməliyik.Bunun üçün yeni metod əlavə etmək əvəzinə yeni sinif tərtib edib Company sinfini varislik mexanizmi ilə əldə edə bilərik.Bu bizə təkrar kod yazmamağa kömək edəcək.Sinif daxilində fullname,email,mobile və s elementlər həmin sinfin atributları (həmçinin global elementlər),İnfo isə metoddur.Qeyd edimki funksiyaya bənzəri olsada,self arqumenti,metodu funksiyadan tamamilə ayırır.init ifadəsi ilə başlayan metodumuz konstruktor,Info isə aparıcı sinfin metodudur.

[573]:
```
class Company():
    """docstring"""

    def __init__(self,fullname,email,mobile,salary,employment):
        self.fullname = fullname
        self.email = email
        self.mobile = mobile
        self.salary = salary
        self.employment = employment
    def Info(self):
        print("""
        Person:{}
        Email :{}
        Mobile:{}
        Salary(per month):{}
        Employment:{}
        """.format(self.fullname,self.email,self.mobile,self.salary,self.
 ↪employment))

class Manager(Company):
    """docstring"""

    pass
```

[574]:
```
company = Company('Nurlan liyev','example@gmail.com','+99455100',2000,['Senior␣
 ↪PHP Developer'])
```

[575]:
```
company.Info()
```

```
        Person:Nurlan liyev
        Email :example@gmail.com
        Mobile:+99455100
        Salary(per month):2000
        Employment:['Senior PHP Developer']
```

[576]:
```
manager = Manager('Vüsalə Məmmədova','example@hotmail.
↪com','+99455200',5000,['Managment'])
```

[577]:
```
manager.Info()
```

```
Person:Vüsalə Məmmədova
Email :example@hotmail.com
Mobile:+99455200
Salary(per month):5000
Employment:['Managment']
```

Kodlarımız daxilində Company sinfində olan elementləri varis alıb manager sinfinə tətbiq etdik.Şirkətimizdə müdir və işçilər ayrı frazalarda göstərildi.

Kodlarımıza dəyişiklik edək

[578]:
```
class Company():
    """docstring"""
    #Ana sinfimiz.Ortaq elementlər və metodları digər siniflərdə istifadə␣
↪olunacaq
    def __init__(self,fullname,email,mobile,salary,employment):
        self.fullname = fullname
        self.email = email
        self.mobile = mobile
        self.salary = salary
        self.employment = employment

    def Info(self):
        print("""
        Person:{}
        Email: {}
        Mobile:{}
        Salary:{}
        Employment:{}
        """.format(self.fullname,self.email,self.mobile,self.salary,self.
↪employment))
class Manager(Company):
    pass
```

[579]:
```
company = Company('Nizami Mahmudov','example@gmail.
↪com','+99455300',400,['Back-end Developer'])
company.Info()
```

```
Person:Nizami Mahmudov
```

```
Email: example@gmail.com
Mobile:+99455300
Salary:400
Employment:['Back-end Developer']
```

[580]:
```
manager = Manager('Arzu  liyeva','example@gmail.
 ↪com','+99455100',5000,['Managment','Leader'])
manager.Info()
```

```
Person:Arzu  liyeva
Email: example@gmail.com
Mobile:+99455100
Salary:5000
Employment:['Managment', 'Leader']
```

Yuxarıdakı kodlarımızda company sinfinə daxil olan elementləri varis alıb manager sinfində istifadə etdik.Yuxarıdakı nümunə kodlarda gördüyünüz kimi miras aldığımız bütün atributlar və metodlar manager sinfinə də daxil oldu.self ifadəsi ilə bütün atributları xaricdən istifadə etməyə imkan verdik.

[585]:
```
dir(manager)
```

[585]:
```
['Info',
 '__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__',
```

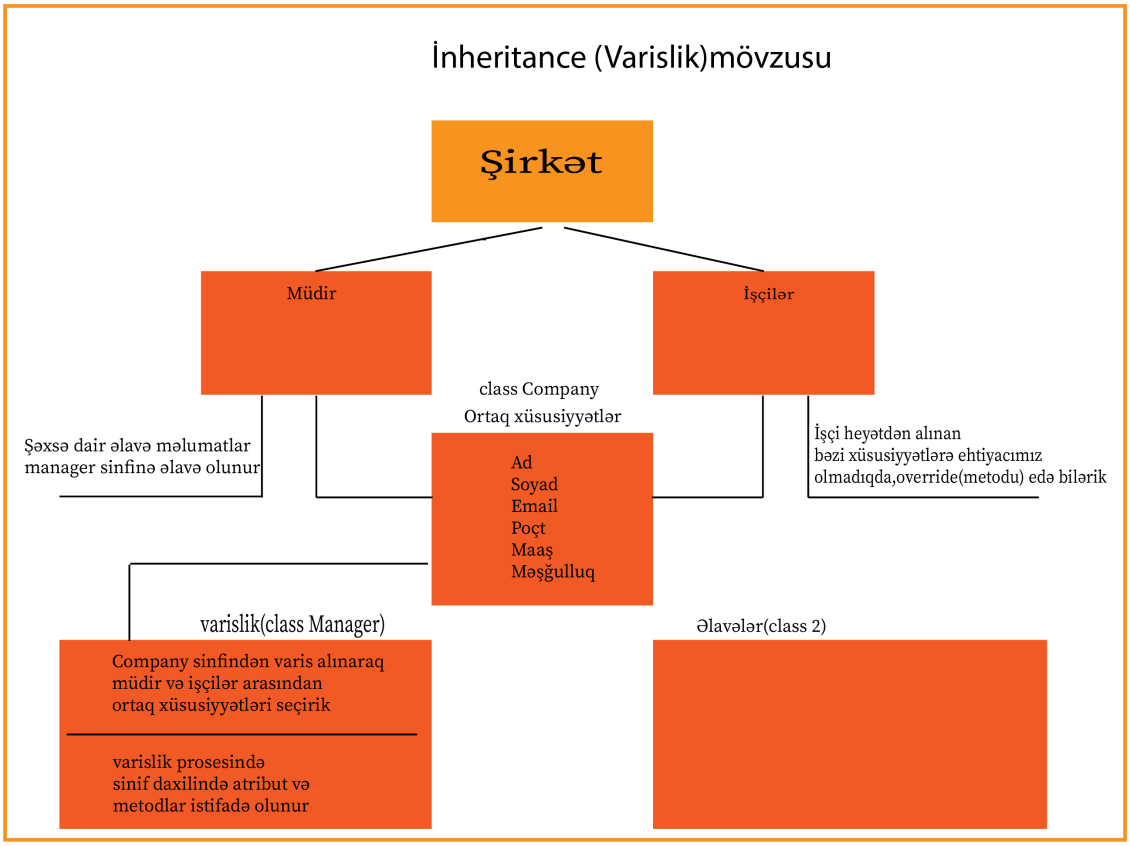
```
'__str__',
'__subclasshook__',
'__weakref__',
'email',
'employment',
'fullname',
'leader',
'mobile',
'salary']
```

dir funksiyası ilə Manager obyektinin içində 5 ədəd atributun Company sinfindən gəldiyini görürük,digərləri isə var olan metodlardır.

manager sinfi üçün metod və atributların təyin etdik.Aşağıdakı rəsmdə bacardığım qədər miras alma mövzusun izah etməyə çalışdım.Kodlarımızı yazdıqca düşünürəm mövzu sizə daha da aydın olacaq

[582]:
```
from IPython.display import Image
Image("image/inhert.png")
```

[582]:



Rəsmdə olan elementləri izah edək.Qeyd etdiyimiz qaydada Şirkətimiz var və şirkətimizdə əsas iki qism mövcuddur.Müdriyyət və işçilər.Bu iki qrupun ortaq xüsusiyyətləri ad soyad maaş və s kimi

xüsusiyyətlərdir.Biz aparıcı şəxs üçün yenidən ad,soyad və s kimi elementləri yazmaq yerinə ümumi company sinfi daxilindən bu atributları növbəti manager sinfinə çəkirik.Və bununla biz company sinfindən varis mexanizmi ilə atribut və metodları alırıq.

gər biz varislik mexanizmini tərtib edib,hazırladığımız manager sinfinə əlavə metodlarımızı yazmaq istəsək,company sinfinin tərtibatçısı metodu daxilindəki bütün atributları da almalıyıq.Daha sonra isə əlavələrimiz mümkündür.Bundan başqa varis etdiyimiz elementləri super metodunda çağırmalıyıq.Aşağıdakı kodlarda super metodu əvəzinə birbaşa varis aldığımız sinfin adını daxil etmişəm.əlavə olaraq leader ifadəsi əlavə edib rəhbər sayını qeyd etdim

[583]:
```
class Company():
    """docstring"""
    #Ana sinfimiz.Ortaq elementlər və metodları digər siniflərdə istifadə␣
 ↪olunacaq
    def __init__(self,fullname,email,mobile,salary,employment):
        self.fullname = fullname
        self.email = email
        self.mobile = mobile
        self.salary = salary
        self.employment = employment

    def Info(self):
        print("""
        Person:{}
        Email: {}
        Mobile:{}
        Salary:{}
        Employment:{}
        """.format(self.fullname,self.email,self.mobile,self.salary,self.
 ↪employment))

class Manager(Company):
    #company sifini miras alırıq
    def __init__(self,fullname,email,mobile,salary,employment,leader):
        Company.__init__(self,fullname,email,mobile,salary,employment)
        self.leader = leader

    def Info(self):
        print("""
        Person:{}
        Email: {}
        Mobile:{}
        Salary:{}
        Employment:{}
        Leader:{}
        """.format(self.fullname,self.email,self.mobile,self.salary,self.
 ↪employment,self.leader))
```

def **init**(self,fullname,email,mobile,salary,employment,leader): -> sətirdəki ifadələr varis aldığımız sinfin nümayəndələri ilə eynidir.hər bir atributu qeyd etmək məcburiyyətindəyik.Çünki başdan manager sinfinə Company sinfini varis alacağımızı bildirmişik.Daha sonra super metodunu istifadə edərək əmr edirikki ,Company.___init___(self,fullname,email,mobile,salary,employment) company sinfindən bu atributları al.konstruktorda leader atributu əlavə etdiyimiz üçün növbəti sətirdə bunu özəlləşdirməliyik –> self.leader = leader.Daha sonra aparıcı sinfin İnfo metodu deyil,manager sinfi üçün məlumatları çap edən yeni info Metodu tərtib etdik.Bu üsul polimorfizm metodudur,eyni adla müxtəlif metodlar

[ ]:

[584]:
```
#rəhbər sinfi
company = Company('Nizami Mahmudov','example@gmail.
 ↪com','+99455300',3000,['Back-end Developer'])
company.Info()
print('*'*50)
manager = Manager('Arzu  liyeva','exmple@hotmail.
 ↪com','+99455500',5000,['Managment'],1)
manager.Info()
```

```
        Person:Nizami Mahmudov
        Email: example@gmail.com
        Mobile:+99455300
        Salary:3000
        Employment:['Back-end Developer']

**************************************************

        Person:Arzu  liyeva
        Email: exmple@hotmail.com
        Mobile:+99455500
        Salary:5000
        Employment:['Managment']
        Leader:1
```

Polymorphism

[5]:
```
class Company():
    """docstring"""
    #Ana sinfimiz.Ortaq elementlər və metodları digər siniflərdə istifadə␣
 ↪olunacaq
    def __init__(self,fullname,email,mobile,salary,employment):
        self.fullname = fullname
        self.email = email
        self.mobile = mobile
        self.salary = salary
```

```
        self.employment = employment

    def Info(self):
        print("""
        Person:{}
        Email: {}
        Mobile:{}
        Salary:{}
        Employment:{}
        """.format(self.fullname,self.email,self.mobile,self.salary,self.
 ↪employment))
    def polimorfizm():
        return ('First class')

class Manager(Company):
    #company sifini miras alırıq
    def __init__(self,fullname,email,mobile,salary,employment,leader):
        Company.__init__(self,fullname,email,mobile,salary,employment)
        self.leader = leader
    def polimorfizm():
        return ('Second class')
    def Info(self):
        print("""
        Person:{}
        Email: {}
        Mobile:{}
        Salary:{}
        Employment:{}
        Leader:{}
        """.format(self.fullname,self.email,self.mobile,self.salary,self.
 ↪employment,self.leader))

print(Company.polimorfizm())
print(Manager.polimorfizm())
```

```
First class
Second class
```

[ ]:

0.2.1 Special Methods (Xüsusi Metodlar)

Sinifləri hazırlayarkən bir neçə metodları biz tətbiq etdik.Məsələn info metodu ilə sinfimizdə yer alan atribut elementlərini ekrana çap etdirdik.Bunların əvəzinə obyektin python tərəfindən hazırlanan xüsusi metodları var.Həmçinin **init** metodunu misal çəkə bilərik.Öncə yeni bir sinif hazırlayaq

[6]:
```
class Book():
    def __init__(self,title,author,date,pages):
        self.title = title
        self.author = author
        self.date = date
        self.pages = pages
```

[8]:
```
book = Book('Python proqramlaşdırma dili','Rəşad Qarayev','10.10.2017',358)
```

Book adlı sinfimizi hazırladır.dir funksiyası ilə hansı atribut və metodların olduğuna baxaq

[9]:
```
dir(book)
```

[9]:
```
['__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__',
 'author',
 'date',
 'pages',
 'title']
```

Aşağıda dördlükdə atributlarımız və yuxarı hissədə isə metodlar yer alır.Yaxşı biz özümüz metodları təyin ediriksə bu xüsusi metodlar nəyə lazımdır.! Bu metodları istifadə etməməyimiz hər hansısa səhv yol deyil,amma naşı bir yoldur.Hətda kökənində qarşılaşacağınız problemlər belə var.Gəlin nümunələrlə nəzər yetirək

[10]:
```
class Book():
    def __init__(self,title,author,date,pages):
        self.title = title
        self.author = author
        self.date = date
        self.pages = pages

book = Book('Python proqramlaşdırma dili','Rəşad Qarayev','10.10.2017',358)
```

[17]:
```
print(book)
```

```
<__main__.Book object at 0x000001C0F6059EB8>
```

Yuxarıda sinif qeyd etdik və parametrləri atributlara göndərdik.Daha sonra print() funksiyası ilə çap etdirdikdə yuxarıdakı məlumat çap olundu.Python həmin ərəfədə **str** metodunu çağırır.

[18]:
```
print(book.__str__())
```

```
<__main__.Book object at 0x000001C0F6059EB8>
```

Və gördüyünüz kimi hər iki nəticə eynidir.Və biz həmin məlumatın yerinə özümüz **str** metodu yazaraq ekrana məlumatlarımızı çap etdirə bilərik

[65]:
```
class Book():
    def __init__(self,title,author,date,pages):
        self.title = title
        self.author = author
        self.date = date
        self.pages = pages
    def __str__(self):
        return ("Book:{} ;Author:{}; Date:{}; Pages:{} ".format(self.title,self.
 ↪author,self.date,self.pages))

book = Book('Python proqramlaşdırma dili','Rəşad Qarayev','10.10.2017',358)
```

[66]:
```
print(book)
```

```
Book:Python proqramlaşdırma dili ;Author:Rəşad Qarayev; Date:10.10.2017;
Pages:358
```

[67]:
```
print(book.__str__())
```

```
Book:Python proqramlaşdırma dili ;Author:Rəşad Qarayev; Date:10.10.2017;
Pages:358
```

Və gördüyünüz kimi İnfo metodu hazırlamaq yerinə pythonun bizə təklif etdiyi xüsusi metodları istifadə edərək nəticəni əldə etdik

[68]:
```
print(len(book))
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-68-54c057cebb26> in <module>()
----> 1 print(len(book))

TypeError: object of type 'Book' has no len()
```

Çünki len metodunu tətbiq etmədiyimiz üçün xəta aldıq.

[113]:
```
class Book():
    """

    Kitab sinfinin hazırlanması
    Sinif daxilində Kitab adı,Müəllif,tarix və səhifə sayı atributları yer alır


    """
    def __init__(self,title,author,date,pages):
        self.title = title
        self.author = author
        self.date = date
        self.pages = pages
    def __str__(self):
        return ("Book:{} ;Author:{}; Date:{}; Pages:{} ".format(self.title,self.
 ↪author,self.date,self.pages))
    def __dir__(self):
        return ['title', 'author', 'date','pages']
    def __len__(self):
        return self.pages

    def __del__(self):
        return ("Kitab silindi")

book = Book('Python proqramlaşdırma dili','Rəşad Qarayev','10.10.2017',358)
```

[114]:
```
print(book.__doc__)
print(book.__len__())
print(len(book))
```

```
    Kitab sinfinin hazırlanması
    Sinif daxilində Kitab adı,Müəllif,tarix və səhifə sayı atributları yer alır
```

```
358
358
```

len funksiyasını ilk öncə ' **len** 'metodunu obyekt daxilində axtarır,varsa metoda əlavə etdiyimiz dəyərlər ekranda göstərilir

[ ]:

[115]:
```
print(book)
```

```
Book:Python proqramlaşdırma dili ;Author:Rəşad Qarayev; Date:10.10.2017;
Pages:358
```

[116]:
```
del(book)
```

[117]:
```
book
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-117-92f3aa8bac21> in <module>()
----> 1 book

NameError: name 'book' is not defined
```

del funksiyası ilə book obyektini tamamilə sildik.

Üç ədəd metod əlavə etdik,növbəti metod **dir** metodunu əlavə edək

[120]:
```
class Book():
    """

    Kitab sinfinin hazırlanması
    Sinif daxilində Kitab adı,Müəllif,tarix və səhifə sayı atributları yer alır


    """
    def __init__(self,title,author,date,pages):
        self.title = title
        self.author = author
        self.date = date
        self.pages = pages
    def __str__(self):
        return ("Book:{} ;Author:{}; Date:{}; Pages:{} ".format(self.title,self.
 ↪author,self.date,self.pages))
    def __dir__(self):
        return ['title', 'author', 'date','pages']
    def __len__(self):
        return self.pages
```

```
    def __del__(self):
        return ("Kitab silindi")

book = Book('Python proqramlaşdırma dili','Rəşad Qarayev','10.10.2017',358)
```

[121]:
```
print(dir(book))
```

```
['author', 'date', 'pages', 'title']
```

Daha öncə dir funksiyasından istifadə etdikdə pythonun obyekt üçün təklif etdiyi metodlar və hazırladığımız atributlar çap olunurdu.Amma biz dir metodunu özəlləşdirdiyimiz üçün sadəcə atributları əlavə etdik.

Və sonda qeyd edimki OOP mövzusunun hal-hazırda bəzi əsas hissələrini izah etdim.OOP metodologiyası geniş mövzudur.PyQt dərslərində biz davamlı olaraq class siniflərdən istifadə edəcəyik.Bu üsulla siz,həm qrafik istifadəçi interfeysi bəhsini öyrənəcək həmdə OOP mövzusunu təkrarlamış olacaqsınız.

[ ]:
```

```