Python Programlama

Ders 7

Ali Mertcan KOSE Msc.
amertcankose@ticaret.edu.tr

İstanbul Ticaret Üniversitesi

Soyutlama

Soyutlama, bir nesnenin görevlerini nasıl yerine getirdiğiyle ilgili ayrıntıları gizlemek anlamına gelir.

Bir nesneyle ne yapabileceğimizi biliyoruz, ancak nesnenin bunu nasıl yapacağını bilmiyoruz. Örneğin, listenin dizin işlevinin bir listede arama yapacağını biliyoruz, ancak bu algoritmanın nasıl uygulandığını bilmiyoruz.

Nesnelerin, yakalayan bir veri soyutlaması olduğunu söylüyoruz

1. Dahili bir temsil (bir nesne hakkında bildiklerimiz)
   - Veri özniyelikleri aracılığıyla
2. Nesneyle etkileşim için bir arayüz (nesneyi nasıl kullanabiliriz)
   - yöntemler aracılığıyla (prosedürler/işlevler)
   - Davranışları tanımlar ancak uygulamayı gizler

Nesne Yönelimli Programlamanın Avantajları

NNP ile verileri, verileri işlemek için kullanılan işlevlerle birlikte depolayabiliriz.

Böl ve yönet geliştirme

- Her sınıfın davranışını ayrı ayrı uygulayın ve test edin
- Artan modülerlik karmaşıklığı azaltır

Sınıflar kodu yeniden kullanmayı kolaylaştırır

- Birçok Python modülü yeni sınıflar tanımlar.
- Her sınıfın ayrı bir ortamı vardır (işlev adlarında çakışma olmaz)
- Kalıtım, alt sınıfların bir üst sınıfın davranışının seçili bir alt kümesini yeniden tanımlamasına veya genişletmesine olanak tanır.

Nesneler

Python'daki her şey bir nesnedir Python, birçok veri türünü destekler. Bazı örnekler;

```
1234      3.14159      "Hello"      [1, 5, 7, 11, 13]
{"CA": "California", "MA": "Massachusetts"}
```

Figure 1: Nesneler

Her biri bir nesnedir ve her nesne:

- Bir tür
- Bir iç veri gösterimi (ilkel veya bileşik)
- Nesneyle etkileşim için bir dizi prosedür

Nesne, bir türün örneğidir.

Sınıflar

Sınıf, bir nesne kümesinin tasarımını (veri ve davranışları) tanımlayan bir türdür. Örneğin, str sınıfı tüm dizelerin davranışlarını tanımlar.

Sınıf, nesnenin davranışlarını tanımlamak için yöntemler ve bir nesnenin durumunu tanımlamak için değişkenler uygular.

Tür tanımlandıktan sonra, işlevlerine erişmek için aynı türden birçok nesne oluşturabiliriz (örnekleyebiliriz).

Öznitelik Verileri ve Yöntemleri

Öznitelikler, sınıf tarafından tanımlanan veriler ve yöntemlerdir (işlevler).

Veri öznitelikleri

- Bir nesne hakkında bilgi.
- Örneğin öğrencilerin isimleri, kimlik numaraları, dersleri vardır.

Yöntemler

- Yalnızca sınıfın nesneleriyle çalışan işlevler.
- Sınıfla arayüz, nesneyle nasıl etkileşim kurduğumuz.
- Nesnelerle neler yapabiliriz, örneğin öğrenciler not ortalamasını hesaplayabilirler.

Kendi Türünüzü (Sınıflarınızı) tanımlayın

Yeni bir tür tanımlamak için class anahtar sözcüğünü kullanın.

```
#class Friend( object ):
#define the attributes here
```

İşlevlerde olduğu gibi, hangi ifadelerin sınıfın bir parçası olduğunu belirten girinti kodu. Nesne kelimesi, sınıfın bir Python nesnesi olduğu ve tüm özniteliklerini (miras) devraldığı anlamına gelir.

Yöntem *__init__* örnekler oluşturma

```
#class Friend( object ):
#define the attributes here
```

__init__ yöntemi, nesneyi oluşturmaktan ve özniteliklerini başlatmaktan sorumludur.

class Friend(object):

def *__init__* (self, name, phone, year):

self.fname = name

self.fphone = phone

self.year = year

Self Referans

Self, sınıfın mevcut örneğini ifade eder ve mevcut nesnenin verilerini güncellememize/erişmemize izin verir.

__init__ yöntemi çağrıldığında, ilk parametre her zaman başlatılan nesneye bir başvuru olan self başvurudur.

Bir nesne üzerinde bir yöntem çağırdığımızda, örneğin s1.index('abc'), self, yöntemin çağrıldığı nesneyi ifade eder, bu durumda s1.

Nesneye özgü verileri kullanan herhangi bir yöntem, ilk parametre olarak self referansına sahiptir.

Nesneleri Örnekleme

Bir nesnenin nitelikleri, bir sınıfın (türün) belirli bir örneğiyle ilişkili değişkenler olan örnek değişkenleri olarak da adlandırılır.

Bir nesne oluşturduğumuzda, örnek değişkenlerinin başlangıç değerlerini ayarlarız.

f1 = Friend('Jane Doe','5551234',2015)

f2 = Friend('John Smith','5559876',2017)

Yukarıdaki ifadeler 2 Friend nesnesi oluşturur.

__init__ yöntemi örtük olarak (otomatik olarak) çağrılır ve parametre olarak iletilen değerleri kullanarak her Friend nesnesinin ad, telefon ve yıl değerlerini ayarlar.

Self'in parametre olarak geçirilmediğine dikkat edin. Python, öz başvuruyu nesnenin yöntemlerine otomatik olarak iletir.

Örnek Verileri

```
1234      3.14159     "Hello"     [1, 5, 7, 11, 13]
{"CA": "California", "MA": "Massachusetts"}
```

Figure 2: Örnek Veri

Yöntem Nedir?

Nesneye özgü verileri kullanan, bir sınıfta tanımlanan bir işlev.

Diğer nesne türlerinde gördüğümüz gibi, nokta operatörünü (.) kullanarak bir nesnenin yöntemlerine erişebiliriz.

Bir nesnede bir yöntem çağrıldığında, çağrılan yöntem otomatik olarak iletilir (self) ve yöntem bu nesnenin verilerine erişebilir

Python'da Yöntemleri Tanımlama

Sınıflarımız birkaç yöntem tanımlayacaktır. Bu yöntemler, belirli örnekler veya nesnelerle ilgili verileri kullanır.

Bir yöntemin uygulanması, bir işlevin uygulanmasına çok benzer, temel bir farkla, yöntem gövdesindeki self nesnesinin örnek değişkenlerine erişirsiniz.

Her yöntem, özel self parametre değişkenini içermeli ve ilk olarak listelenmelidir. Bu değişken, yöntemi içinde verisi kullanılacak olan nesneyi ifade eder.

Yöntemler bir veya daha fazla parametre alabilir, ancak ilk Self olmalıdır.

Özel Yöntemler dize gösterimi (__str__)

Bir nesneyi görüntülemek veya bir nesneyi bir dize ile birleştirmek istediğimizde, nesnenin dize temsilini almamız gerekir. Örneğin, bir nesneyi yazdırdığımızda, python nesneyi otomatik olarak bir string'e dönüştürür.

Bir nesneyi dizeye dönüştürmek için str() yöntemi çağrılır.

Varsayılan olarak, özel nesnelerimiz için str() yöntemi nesnenin adresini ve türünü döndürür, ancak nesneye özgü verileri döndürmez.

print deyimini kullanarak bir nesne yazdırırsak, nesnenin adresi görüntülenecektir.

Nesnemizin dize gösterimini döndürmek için kullanılacak yöntem. Yöntemimiz varsayılan yöntemin yerini alır (geçersiz kılar).

Özel Yöntemler dize gösterimi (__*repr*__)

(__*str*__) ve (__*repr*__) yöntemlerinin her ikisi de bir nesnenin String temsilini almak için kullanılır.

print deyimi ve str() yerleşik işlevi __ öğesini çağırır ve repr() yerleşik işlevi nesneyi görüntülemek için (__*repr*__) öğesini çağırır.

Fark nedir?

str(): bir nesnenin kullanıcı Friend olarak bir sürümünü döndürür

repr(): bir nesnenin geliştirici Friende olarak bir sürümünü döndürür.

Örnek: Bir datetime nesnesi oluşturun ve dize temsillerini inceleyin.

import datetime

today = datetime.datetime.now()

print(str(today))–> '2021 09 15 09:26:40.073045' print(repr(today)) -> 'datetime.datetime(2021,9,15,9,26,40,73045)'

Kapsülleme ve Bilgi Gizleme

Nesne yönelimli programlamanın iki önemli ilkesi vardır.

Kapsülleme: veri özniteliklerinin bir araya getirilmesi ve bunlar üzerinde işlem yapma yöntemleri.

Verileri ve davranışları bir sınıfa kapsüllemek, bir sınıf içinde **bilgi gizlemeyi** etkinleştirmemize olanak tanır.

Bilgi gizleme Uygulama ayrıntılarını gizlerken genel bir arabirim sağlama işlemi.

Nesnelerin Özel Nitelikleri

Bir nesnenin verilerine nasıl erişildiğini ve güncellendiğini kontrol etmek için, örnek değişkenleri özel hale getirilmelidir.

Bir özniteliği özel yapma kuralı, başında çift alt çizgi `__VarName`

Özel adlandırma kuralını kullanarak, kullanıcılar özel verilere doğrudan sınıfın dışından erişemez, bunun yerine genel arabirimi (sınıftaki yöntemler) kullanarak erişir.

Getter ve Setter Yöntemleri

Her sınıf, özel veri üyelerinin veri kapsüllemesine genel erişime izin veren bir arabirim tanımlar.

Bir sınıfın tasarımcıları, özel veriler için hangi erişimin verileceğine karar verir.

Bir nesnenin veri üyelerine doğrudan sınıfın dışından erişmek yerine, erişim şu şekilde sağlanır:

- **Get yöntemleri:** özel veri üyelerinin değerlerini alır.
- **Set yöntemleri:**özel veri üyelerinin değerlerini güncelleştirmek için.

Nesneler, get ve set yöntemleri ile değiştirilebilir/değiştirilemez hale getirilebilir.

Getter ve Setter Yöntemleri

```
class Friend( object ):
def __init__( self, name, phone, year ):
    self.__fname = name
    self.__fphone = phone
    self.__year = 0
    self.set_year( year )
    def get_fname(self):
    return self.__fname
```

Getter ve Setter Yöntemleri

```
def set_fname( self, new_name ):

    self.__fname = new_name

def get_fphone(self):

    return self.__fphone

def set_fphone( self, new_phone ):

    self.__fphone = new_phone
```

Getter ve Setter Yöntemleri

```
def get_year(self):
    return self.__year
def set_year( self, new_year ):
    if new_year > 1900:
        self.__year = new_year
```

Getter ve Setter Yöntemleri

```
def (__str(__(self):

return f'Name: {self.__fname} Phone: {self.__fphone} Friend
Since: {self.__year}'

def (__repr(__(self):

return f'Name: {self.__fname} Phone: {self.__fphone} Friend
Since: {self.__year}'
```

Getter ve Setter Yöntemleri

```
f1 = Friend('Jane Doe','555-1234',2015)
f2 = Friend('John Smith','555-9876',2017)
print(f1)
print(f2)
f1.set_year( 2020 )
print(f1)
f2.set_year( -2020 )
print(f2)
```

Car Örneği

Aşağıdaki özniteliklere sahip bir Araba sınıfı oluşturun.

- Plaka No
- Marka
- Model
- Yıl
- Fiyatı

Aşağıdaki yöntemleri tanımlayın:

- int()
- Her özniteliğin değerlerini döndüren ve güncelleyen yöntemler
- repr/str methods

Car Örneği

4 Araba oluşturan, bunları bir listede saklayan ve aşağıdakileri yapan bir komut dosyası yazın.

- Arabaların listesini görüntüle
- Bir plaka numarası girin ve eşleşen arabayı görüntüleyin, araba yoksa hata
- 2014'ten sonra üretilen tüm otomobilleri gösterir.

Car Örneği

from Car import *

c1 = Car('06-AB-1234', 'Renault', 'Megane', 2018, 120300)

c2 = Car('34-DE-9876', 'Renault', 'Megane', 2015, 71000)

c3 = Car('07-DE-4563', 'Toyota', 'Carolla', 2010, 45000)

c4 = Car('09-XY-2145', 'Volkswagen', 'Passat', 2013, 89600)

```
#add cars to list car_list = [c1,c2,c3,c4]
```

```
#print('List of cars:',car_list)
```

```
#input a plate num and display if a car matches
```

Car Örneği

pNo = input('Enter plate number to search:')

found = False

for car in car_list:

```
if car.getplateNo() == pNo:

    print(car)

    found = True
```

Car Örneği

if not found:

```
print('No matching car')
#display the cars produced after 2014
```

print(‘Cars 2014 model or later:’)

Car Örneği

```
for car in car_list:

if car.getyear() >2014:

    print(car)

#display average price of all cars in the list
```

Car Örneği

```
sum_p = 0

for car in car_list:

sum_p += car.getprice()

average = sum_p / len(car_list)

print(f'Average price of cars: {average:.2f}')
```

Yerleşik İşlevsellik ve Özel Yöntemler

Yerleşik sıralama yöntemini kullanarak Arabalar listesini sıralarsak ne olur? Belirli bir türdeki nesneleri sıraladığımızda, sıralama düzeni nesnenin niteliklerine bağlıdır. Genellikle sıralama düzeni uygulamaya özgüdür, belirli bir türdeki nesneleri nasıl sıralamak isteriz? Arabaların plakaya göre sıralanması gerektiğini varsayalım. Kendi özel sıralama davranışımızı tanımlamak için yerleşik özel yöntemler uygulamamız gerekir.

list.sort() fonksiyonu, liste nesnelerini yerleşik olarak kullanarak (*__lt__*) metodunu karşılaştırır.

Arabalar için, bir arabayı plakalarına göre diğerinden daha küçük olarak tanımlamak için (*__lt__*) uygulayacağız.

Özel Yöntemler

Sınıfın bir örneğine standart bir Python operatörü (+,-,/,==,<,>...) uygulandığında otomatik olarak çağrılacak yöntemleri tanımlayabilir ve uygulayabiliriz. Bu, yöntemlerin adlarıyla çağrılmasından ziyade nesnelerin daha doğal bir şekilde kullanılmasına izin verir.

Örnek İki nesnenin eşit olup olmadığını test etmek için bir isequal() yöntemi uygulayabilir ve bunu aşağıdaki gibi kullanabiliriz.

if obj1.isequal(obj2)

Bunun yerine operatörü kullanabiliriz ve bu, operatörü kullanarak iki nesneyi karşılaştırdığımızda otomatik olarak çağrılan özel yöntem (__*eq*__)tanımlanarak elde edilir.

Yaygın Özel Yöntemler

Expression	Method Name
x + y	__add__(self,y)
x - y	__sub__(self,y)
x * y	__mul__(self,y)
x / y	__truediv__(self,y)
x // y	__floordiv__(self,y)
x % y	__mod__(self,y)
x ** y	__pow__(self,y)
x == y	__eq__(self,y)
x != y	__ne__(self,y)
x < y	__lt__(self,y)
x <= y	__le__(self,y)
x > y	__gt__(self,y)
x >= y	__ge__(self,y)

Kalıtım

Kalıtım, mevcut sınıfları (üst/temel/süper sınıflar) genişleterek yeni sınıflar (alt/türetilmiş/alt sınıflar) tanımlamak için kullanılabilir. Alt sınıf, kendi özelliklerinden ve yöntemlerinden bazılarını eklemenin ve bazı üst sınıf yöntemlerini geçersiz kılmanın (değiştirmenin) yanı sıra üst sınıfın özelliklerini ve yöntemlerini miras alır. Sınıf nesnesi hiyerarşinin en üstündedir, yani her yeni sınıf, nesne sınıfından otomatik olarak veri ve davranışları devralır.

Alt sınıf

- Nesne sınıfındaki tüm yöntemler / nitelikler dahil olmak üzere üst sınıfından öznitelikleri devralır
- Yeni özellikler ekleyebilir.
- Üst sınıfın yöntemlerini / niteliklerini geçersiz kılabilir veya değiştirebilir.

Alt sınıf

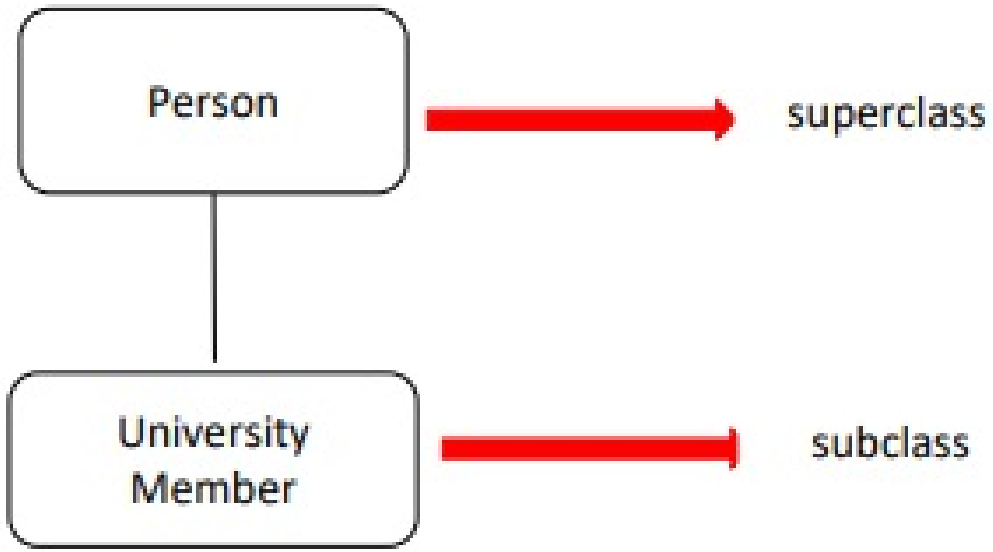

Figure 4: Altsınıf Örneği

Alt Sınıfları Tanımlama

Her yeni sınıf otomatik olarak nesne sınıfının bir alt sınıfıdır ve varsayılan davranışları ve işlevleri devralır. Sınıf adından sonra parantez içinde, verilerini ve işlevselliğini devralacağı üst sınıfı belirterek bir alt sınıf oluşturun.

Örnek

class myClass(superclass):

Alt sınıf, yeni veya güncelleştirilmiş öznitelikler ve/veya yöntemler ekleyerek üst sınıfı genişletir.

Yöntem Geçersiz Kılma

Alt sınıf, yöntemleri üst sınıftan devralır. Devralınan bir yöntemin davranışını değiştirmek isterseniz, bunu geçersiz kılabilirsiniz. Alt sınıfta yeni bir uygulama belirterek miras alınan yöntemleri geçersiz kılabiliriz. Alt sınıftaki yöntemler, üst sınıf yönteminin işlevselliğinin yerini alır ve farklı davranışlar uygular.

Bir nesne üzerinde bir yöntem çağırdığımızda, önce nesnenin sınıfı eşleşen bir yöntem için incelenir. Bulunursa, yöntem yürütülür. Alt sınıfta eşleşen bir yöntem yoksa, bir eşleşme bulunana kadar üst sınıflar aranır

Class: UniversityMember

UniversityMember sınıfı, Person sınıfını genişletir. Sınıf, üst sınıfı aşağıdakilere göre genişletir.

- idNum adında yeni bir öznitelik ekler.
- (___*init*___) yöntemini geçersiz kılar.
- (___*it*___) yöntemini geçersiz kılar.

str(p1) kodunda ilk olarak UniversityMember sınıfında (___*str*___)bir metot olup olmadığını kontrol eder. Olmadığı için, daha sonra üst sınıfında (___*str*___)yöntem olup olmadığını kontrol eder. Var, bu yüzden bu yöntemi kullanıyor. p1.getIdNum() önce UniversityMember sınıfında bir getIdNum yöntemi olup olmadığını kontrol eder. Var, bu yüzden bu 07_UniversityMember çağırıyor.

Class: UniversityMember (Version 2)

Bu uygulamada Üniversite Üyelerini karşılaştırıyoruz. p1, p2 ve p3'ün tümü UniversityMember türünde olduğundan, ilk iki karşılaştırma değerlendirilirken UniversityMember sınıfında tanımlanan (___*it*___) yöntemi çağrılır, bu nedenle sıralama kimlik numaralarına göre yapılır.

Üçüncü karşılaştırmada, $<$ operatörü farklı türdeki işlenenlere uygulanır. P4 $<$ P1, p4.(___*it*___)($P1$) 'in kısaltmasıdır. Böylece Person sınıfının (___*it*___) yöntemi kullanılacak ve isimlere göre karşılaştırma yapılacaktır.

Yazdırmayı denersek ne olur ('p1 $<$ p4', p1 $<$ p4)

Çoklu Kalıtım Düzeyleri

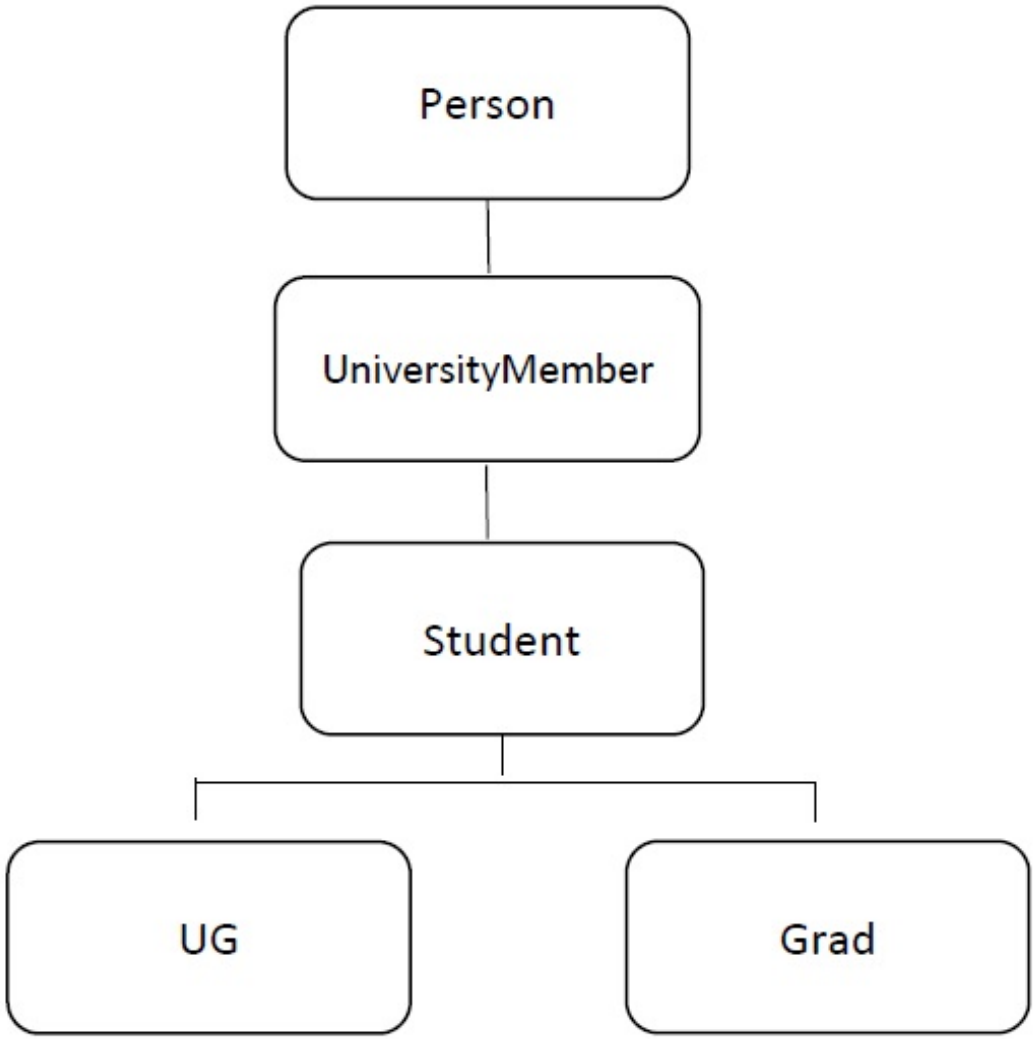

Figure 5: Çoklu Kalıtım Düzeyleri

Ayrılmış Word geçişi

Sınıfın, üst sınıfından miras alınanlar dışında hiçbir özniteliği olmadığını gösterir. Neden yeni öznitelikleri olmayan bir sınıf oluşturuyoruz?

isinstance yerleşik işlevi

isinstance(object, type), yalnızca ve yalnızca ilk bağımsız değişken ikinci bağımsız değişkenin bir örneğiyse True döndürür. örneğin. isinstance([1,2], list) True

Bilgi Gizleme ve Kalıtım

Bir alt sınıf, üst sınıfının gizli bir niteliğini kullanmaya çalıştığında ve AttributeError oluştuğunda.

Kalıtım alınan özniteliklere erişmek için, alt sınıfın gizli verilere erişmek için get ve set yöntemlerini çağırması gerekir.

Bu nedenle bilgi gizlemeyi kullanmak zor olabilir, bu nedenle bazı Python programcıları genellikle gizli öznitelik sözdizimini __ kullanmamayı tercih eder.