pythonda sınıf, miras ve metot ezme detaylı olarak yapılarını teknik terimleri, kullanım örnekleri ve metotlarını (*init*, *str* gibi) bütün olarak anlatımı

Python'da sınıflar, miras (inheritance) ve metot ezme (overriding) nesne yönelimli programlamanın temelini oluşturur; özel metotlar (dunder methods) ise sınıfların dile "entegre" olmasını sağlar. Aşağıda hem teknik terimler hem de günlük hayata yakın örneklerle, özellikle `__init__`, `__str__` gibi metotlarla birlikte bütünsel bir çerçeve var.[1] [2] [3]

Sınıf ve nesne temeli

- **Sınıf (class)**: Ortak özellik ve davranış paylaşan nesneler için tanımlanan şablon.[3]
- Nesne (instance): Bu şablondan üretilmiş somut örnek (örneğin `Araba` sınıfından üretilmiş `araba1`).[3]
- Özellik (attribute): Nesne veya sınıf üzerinde tutulan veri (örneğin `self.marka`).[3]
- Metot (method): Sınıf içinde tanımlanan fonksiyon (örneğin `def calistir(self): ...`).[3]

Basit bir sınıf:

```
class Araba:
    # sınıf düzeyinde (class attribute)
    tip = "kara aracı"

    def __init__(self, marka, model, yil):
        self.marka = marka      # örnek (instance) attribute
        self.model = model
        self.yil = yil

    def calistir(self):
        print(f"{self.marka} {self.model} çalıştı")

araba = Araba("Toyota", "Corolla", 2020)
araba.calistir()
```

Burada `__init__` kurucu (constructor), `self` ise oluşturulan nesnenin referansıdır.[4]

Miras (inheritance) yapısı

- **Miras (inheritance)**: Bir sınıfın (alt sınıf, subclass) başka bir sınıftan (üst/base class) özellik ve metot devralmasıdır.[5]
- Amaç: Kod tekrarını azaltmak, ortak davranışı merkezileştirmek, daha soyut/özel hiyerarşiler kurmak.[6]

Temel tekli miras:

```
class Arac:
    def __init__(self, marka):
        self.marka = marka

    def bilgi(self):
        return f"Marka: {self.marka}"

class Araba(Arac):              # Araba, Arac'tan miras alıyor
    def __init__(self, marka, kapı_sayisi):
        super().__init__(marka)   # üst sınıfın __init__'ini çağır
        self.kapı_sayisi = kapı_sayisi

    def bilgi(self):
        # metot ezme (overriding); üst sınıfın metodunu genişletiyoruz
        temel = super().bilgi()
        return f"{temel}, Kapı sayısı: {self.kapı_sayisi}"

a = Araba("Toyota", 4)
print(a.bilgi())
```

- `Araba(Arac)` ile Araba sınıfının taban sınıfı `Arac` olarak belirtilir.[5]
- `super()` ile miras zincirindeki bir üst sınıfın metotlarına erişilir.[7] [8]
- `bilgi` metodunu alt sınıfta tekrar tanımlayarak metot **ezme** yapılır.[9]

Çoklu miras (multiple inheritance) özeti:

```
class Elektrikli:
    def sarj_et(self):
        print("Şarj ediliyor")

class Otonom:
    def sur(self):
        print("Otonom sürüş başlatıldı")

class AkilliAraba(Elektrikli, Otonom):
    pass

a = AkilliAraba()
a.sarj_et()
a.sur()
```

Python, çoklu mirasta metot arama sırasını **MRO (Method Resolution Order)** ile belirler; sıralama `sınıf.__mro__` ile görülebilir.[10] [3]

Metot ezme (overriding) ve super()

- **Metot ezme**: Bir alt sınıfta, üst sınıftakiyle aynı isim ve imzaya sahip bir metodu yeniden yazarak davranışını değiştirmek veya genişletmektir.[9] [6]
- Çoğunlukla `__init__`, `__str__`, iş mantığı metotları gibi yerlerde kullanılır.[4]

Basit metot ezme:

```
class Hayvan:
    def ses_cikar(self):
        print("Belirsiz hayvan sesi")

class Kedi(Hayvan):
    def ses_cikar(self):          # override
        print("Miyav")

h = Hayvan()
k = Kedi()
h.ses_cikar()   # Belirsiz hayvan sesi
k.ses_cikar()   # Miyav
```

`super()` ile üst davranışı koruyup genişletme:

```
class Loglayici:
    def kaydet(self, mesaj):
        print(f"Log: {mesaj}")

class DosyaLoglayici(Loglayici):
    def kaydet(self, mesaj):
        super().kaydet(mesaj)          # önce temel davranış
        print("Mesaj dosyaya yazıldı") # ek davranış

d = DosyaLoglayici()
d.kaydet("sistem açıldı")
```

`super()` özellikle çoklu mirasta MRO'ya uygun şekilde zincirdeki doğru metodu çağırdığı için önerilir.[8] [10]

Özel metotlar (dunder methods) – init, str vb.

Python veri modeli, belirli isimlerdeki "özel metotlar" üzerinden sınıfların operatörler, yerleşik fonksiyonlar ve söz dizimiyle etkileşmesini sağlar. Bunlara genellikle **dunder (double underscore)** metotlar denir.[2] [1] [3]

init – kurucu

- Nesne oluşturulurken otomatik çağrılan başlatıcı metottur.[4]
- İmza tipik olarak `def __init__(self, ...)` şeklindedir ve **return değeri kullanılmaz** (varsayılan `None`).[4]

```
class Kisi:
    def __init__(self, ad, yas=0):
        self.ad = ad
        self.yas = yas

k = Kisi("Ali", 30)   # __init__ otomatik çalışır
```

Miras ile `__init__` ezme:

```
class Calisan(Kisi):
    def __init__(self, ad, yas, maas):
        super().__init__(ad, yas)   # üst sınıfın init'i
        self.maas = maas
```

Üst `__init__` çağrılmazsa, üst sınıfın kurduğu bazı attribute'lar hiç oluşmayabilir ve `AttributeError` alınabilir; bu nedenle gerektiğinde mutlaka çağrılmalıdır.[11] [12] [4]

str ve repr – yazdırma / temsil

- `__str__(self)`: `print(obj)` veya `str(obj)` çağrıldığında kullanılan, insan okuyabilir biçimde metin döndüren metot.[2]
- `__repr__(self)`: Resmi/diagnostik temsil; `repr(obj)` ve çoğu REPL çıktısı bunu kullanır.[2] [3]

```
class Kisi:
    def __init__(self, ad, yas):
        self.ad = ad
        self.yas = yas

    def __repr__(self):
        return f"Kisi(ad={self.ad!r}, yas={self.yas})"

    def __str__(self):
        return f"{self.ad} ({self.yas} yaşında)"

k = Kisi("Ayşe", 25)
print(k)          # __str__ -> Ayşe (25 yaşında)
print(repr(k))    # __repr__ -> Kisi(ad='Ayşe', yas=25)
```

Bu metotlar loglama, hata mesajları ve debug sürecini oldukça **kolaylaştırır**.[2]

Diğer yaygın özel metotlar (özet)

Tam liste Python veri modeli dökümantasyonunda yer alır, burada pratikte en sık görülen gruplar özetlenmiştir.[13] [2] [3]

- Örnek oluşturma/yaşam döngüsü:
  - `__new__`: Nesne gerçekten oluşturulmadan önce çağrılır; immutable tiplerden türetirken ileri seviye senaryolarda kullanılır.[3]

    - `__del__`: Yıkıcı (destructor), nesne toplanırken çağrılabilir; kaynak yönetimi için `with` ve context manager genelde daha güvenlidir.[3]
- Metin temsili:
    - `__str__`, `__repr__`, gerektiğinde `__format__`, `__bytes__`.[2] [3]
- Koleksiyon davranışı:
    - `__len__` → `len(obj)` fonksiyonunu destekler.[3]
    - `__getitem__`, `__setitem__`, `__delitem__` → `obj[key]` söz dizimini kontrol eder.[3]
    - `__iter__`, `__next__` → nesneyi yinelenebilir (iterable) yapar (`for` döngüsü gibi).[3]
    - `__contains__` → `x in obj` davranışını tanımlar.[3]
- Nitelik erişimi:
    - `__getattr__`, `__setattr__`, `__delattr__` → attribute erişimini dinamik/özel hale getirir.[3]
- Operatör aşırı yükleme (operator overloading):
    - Aritmetik: `__add__`, `__sub__`, `__mul__`, `__truediv__`, `__floordiv__`, `__mod__`, `__pow__` vb.[14] [1] [2]
    - Karşılaştırma: `__eq__`, `__ne__`, `__lt__`, `__le__`, `__gt__`, `__ge__`.[3]
    - Ters (right-hand) ve yerinde (in-place) versiyonlar: örn. `__radd__`, `__iadd__`.[14]

Örnek: uzunluk ve indeks destekleyen basit koleksiyon:

```
class NotDefteri:
    def __init__(self, baslik, notlar=None):
        self.baslik = baslik
        self._notlar = list(notlar) if notlar else []

    def __len__(self):               # len(not_defteri)
        return len(self._notlar)

    def __getitem__(self, index):    # not_defteri[index]
        return self._notlar[index]

    def __str__(self):               # print(not_defteri)
        return f"{self.baslik} ({len(self)} not)"

nd = NotDefteri("Günlük", ["Bugün python çalıştım", "Spor yaptım"])
print(len(nd))      # 2
print(nd[^1_0])        # Bugün python çalıştım
print(nd)           # Günlük (2 not)
```

Bu sayede kendi sınıflarını standart Python koleksiyonları gibi **kullanılabilir** hale getirirsin.[1] [2]

Dilersen bir sonraki adımda sadece miras ve `super()` kullanarak birkaç küçük egzersiz (örneğin küçük bir “personel sistemi” ya da “araç filosu” modeli) oluşturup birlikte çözebiliriz.

⁂

sınıf, miras ve metot ezme yi kullanan basit açıklamalı 10 farklı üniversite öğrencisinin günlük hayatından örnek açıklama satırlı kodlar

Aşağıdaki 10 örnekte her senaryoda **sınıf**, miras ve *en az bir* metot ezme (`__init__`, normal metotlar, bazılarında `__str__`) birlikte kullanılıyor. Her örnek üniversite öğrencisinin günlük hayatından basit tutuldu.[21] [22] [23] [24]

1) Ders planı – temel / alt sınıf

```
class Ders:
    # Üst sınıf: Genel bir ders
    def __init__(self, ad, kredi):
        self.ad = ad
        self.kredi = kredi

    def ozet(self):
        return f"{self.ad} ({self.kredi} kredi)"


class ZorunluDers(Ders):
    # Alt sınıf: Zorunlu ders
    def __init__(self, ad, kredi, bölüm):
        # Üst sınıfın __init__'ini kullan
        super().__init__(ad, kredi)
        self.bölüm = bölüm

    # ozet metodunu EZIYOR (override ediyor)
    def ozet(self):
        temel = super().ozet()
        return f"{temel} - Zorunlu, bölüm: {self.bölüm}"


mat = ZorunluDers("Lineer Cebir", 5, "Bilgisayar Mühendisliği")
print(mat.ozet())
```

Bu senaryoda öğrenci haftalık ders planına bakarken zorunlu derslerin açıklamasını daha detaylı görmek istiyor.[23] [21]

2) Öğrenci kulübü üyeliği

```
class Uye:
    def __init__(self, ad):
        self.ad = ad

    def rol(self):
        return "Normal üye"
```

```
class KulupBaskani(Uye):
    def __init__(self, ad, kulup_adi):
        super().__init__(ad)
        self.kulup_adi = kulup_adi

    # rol metodunu ez
    def rol(self):
        return f"{self.kulup_adi} kulübü başkanı"


ali = KulupBaskani("Ali", "Yazılım")
print(ali.ad, "-", ali.rol())
```

Burada aynı “üye” yapısından başkan, koordinatör gibi farklı roller türetiliyor; davranış `rol()` ile özelleştiriliyor.[22] [23]

3) Yemekhane kartı ve burslu kart

```
class YemekhaneKarti:
    def __init__(self, bakiye):
        self.bakiye = bakiye

    def harca(self, tutar):
        if self.bakiye >= tutar:
            self.bakiye -= tutar
            print(f"{tutar} TL harcandı. Kalan: {self.bakiye} TL")
        else:
            print("Yetersiz bakiye")


class BursluKarti(YemekhaneKarti):
    # Burslu öğrencinin günlük ücretsiz limitini modelleyelim
    def __init__(self, bakiye, günlük_limit):
        super().__init__(bakiye)
        self.günlük_limit = günlük_limit
        self.bugün_harcanan = 0

    # harca metodunu ez
    def harca(self, tutar):
        if self.bugün_harcanan + tutar <= self.günlük_limit:
            self.bugün_harcanan += tutar
            print(f"{tutar} TL burs hakkından kullanıldı. Bugün toplam: {self.bugün_harca
        else:
            # Limit aşılırsa normal kart gibi davran
            print("Burs limiti aşıldı, normal bakiyeden düşülüyor.")
            super().harca(tutar)


kart = BursluKarti(50, 20)
kart.harca(10)   # burs hakkı
kart.harca(15)   # limit aşılır, normal bakiyeden düşer
```

Öğrenci için burslu kart ile normal kart arasındaki fark tek metot (`harca`) ezilerek modellenmiş.[25] [24]

4) Ders notu hesaplama (Vize/Final/Proje)

```
class NotHesaplayici:
    def __init__(self, vize, final_):
        self.vize = vize
        self.final_ = final_

    def ortalama(self):
        return self.vize * 0.4 + self.final_ * 0.6


class ProjeliNotHesaplayici(NotHesaplayici):
    def __init__(self, vize, final_, proje):
        super().__init__(vize, final_)
        self.proje = proje

    # ortalama metodunu ez
    def ortalama(self):
        # Proje %20, vize %30, final %50 diyelim
        return self.vize * 0.3 + self.final_ * 0.5 + self.proje * 0.2


n1 = NotHesaplayici(70, 80)
n2 = ProjeliNotHesaplayici(70, 80, 90)

print("Normal ders:", n1.ortalama())
print("Projeli ders:", n2.ortalama())
```

Aynı “not hesaplama” davranışı, projeli derslerde farklı formülle ezilerek farklılaştırılıyor.[22] [23]

5) Kütüphane ödünç sistemi

```
class Kitap:
    def __init__(self, ad):
        self.ad = ad
        self.ödünçte = False

    def ödünc_al(self):
        if not self.ödünçte:
            self.ödünçte = True
            print(f"'{self.ad}' ödünç alındı")
        else:
            print("Zaten ödünçte")


class Tez(Kitap):
    # Lisans/tez gibi özel materyaller için davranış
    def ödünc_al(self):
        # Tezlerin kütüphane dışına çıkmadığını varsayalım
```

```
        print(f"'{self.ad}' sadece kütüphane içinde incelenebilir, dışarı çıkarılamaz.")


k = Kitap("Python Programlama")
t = Tez("Bitirme Tezi")

k.odünc_al()
t.odünc_al()
```

Burada tezler, temel kitap davranışını tamamen farklı bir implementasyonla **ezerek** kısıtlı kullanım sağlıyor.[26] [27]

6) Ulaşım kartı – öğrenci indirimi

```
class UlasimKarti:
    def __init__(self, bakiye):
        self.bakiye = bakiye

    def binis(self, ücret):
        if self.bakiye >= ücret:
            self.bakiye -= ücret
            print(f"Biniş yapıldı, kalan bakiye: {self.bakiye} TL")
        else:
            print("Yetersiz bakiye")


class OgrenciKarti(UlasimKarti):
    def binis(self, ücret):
        # Öğrenci indirimi: %50
        indirimli = ücret * 0.5
        print(f"Öğrenci indirimi uygulandı: {indirimli} TL")
        # Üst sınıfın binis metodunu indirimli ücretle çağır
        super().binis(indirimli)


kart = OgrenciKarti(20)
kart.binis(10)   # 10 yerine 5 TL düşer
```

Günlük hayatta kullanılan indirimli öğrenci kartı, normal karttan miras alarak sadece ücretli biniş davranışını değiştiriyor.[25] [23]

7) Öğrencinin yapılacaklar listesi (to-do)

```
class Yapilacak:
    def __init__(self, baslik):
        self.baslik = baslik
        self.tamamlandi = False

    def isaretle(self):
        self.tamamlandi = True
```

```
    def __str__(self):
        durum = "✓" if self.tamamlandi else "✗"
        return f"[{durum}] {self.baslik}"


class OnemliYapilacak(Yapilacak):
    # Önemli işlerin metin gösterimini ezelim
    def __str__(self):
        durum = "✓" if self.tamamlandi else "✗"
        return f"[{durum}] ÖNEMLİ: {self.baslik.upper()}"


odev = Yapilacak("Matematik ödevi")
proj = OnemliYapilacak("Mobil proje raporu")

odev.isaretle()
print(odev)
print(proj)
```

Burada `__str__` metodu ezilerek önemli görevlerin çıktısı daha dikkat çekici hale getiriliyor.[28] [29]

8) Online ders oturumu – kayıtlı / misafir

```
class DersOturumu:
    def __init__(self, ders_adi):
        self.ders_adi = ders_adi

    def giris_mesaji(self):
        return f"{self.ders_adi} oturumuna hoş geldiniz."


class KayitliOgrenciOturumu(DersOturumu):
    def __init__(self, ders_adi, ogr_no):
        super().__init__(ders_adi)
        self.ogr_no = ogr_no

    def giris_mesaji(self):
        temel = super().giris_mesaji()
        return f"{temel} Öğrenci no: {self.ogr_no}"


class MisafirOturumu(DersOturumu):
    # Farklı bir karşılama metodu
    def giris_mesaji(self):
        return f"{self.ders_adi} için misafir erişimi sağlandı."


k = KayitliOgrenciOturumu("Veri Yapıları", "20201234")
m = MisafirOturumu("Veri Yapıları")

print(k.giris_mesaji())
print(m.giris_mesaji())
```

Aynı ders oturumuna kayıtlı öğrenci ile misafirin giriş mesajı, metot ezme sayesinde farklılaşıyor. [27] [30]

9) Not alma uygulaması – basit / ders notu

```
class Not:
    def __init__(self, icerik):
        self.icerik = icerik

    def etiket(self):
        return "genel"


class DersNotu(Not):
    def __init__(self, icerik, ders_adi):
        super().__init__(icerik)
        self.ders_adi = ders_adi

    # etiket metodunu ez
    def etiket(self):
        return f"ders: {self.ders_adi}"


n1 = Not("Alışveriş listesi")
n2 = DersNotu("Sınav konuları: ağaçlar, grafikler", "Veri Yapıları")

print(n1.etiket(), "-", n1.icerik)
print(n2.etiket(), "-", n2.icerik)
```

Öğrenci telefonundaki not uygulamasında ders notları ile genel notlar arasında etiketleme davranışı ayrılıyor. [26] [22]

10) Öğrenci bilgi sistemi – lisans / yüksek lisans

```
class Ogrenci:
    def __init__(self, ad, numara):
        self.ad = ad
        self.numara = numara

    def ozet(self):
        return f"{self.ad} ({self.numara})"


class LisansOgrencisi(Ogrenci):
    def __init__(self, ad, numara, sinif):
        super().__init__(ad, numara)
        self.sinif = sinif

    def ozet(self):
        temel = super().ozet()
        return f"{temel} - Lisans, {self.sinif}. sınıf"
```

```
class YuksekLisansOgrencisi(Ogrenci):
    def __init__(self, ad, numara, tez_konusu):
        super().__init__(ad, numara)
        self.tez_konusu = tez_konusu

    def ozet(self):
        temel = super().ozet()
        return f"{temel} - Yüksek lisans, tez: {self.tez_konusu}"


l = LisansOgrencisi("Merve", "20205678", 3)
y = YuksekLisansOgrencisi("Burak", "20220012", "Derin Öğrenme")

print(l.ozet())
print(y.ozet())
```

Öğrenci bilgi sistemi gibi bir yapıda temel öğrenci sınıfından lisans ve yüksek lisans alt tipleri üretilip `ozet` metodu her tipte eziliyor.[21] [22]

İstersen bir sonraki adımda bu örneklerden birkaçını seçip küçük bir "mini proje" halinde birleştirebiliriz (örneğin hem `Ogrenci`, hem `Ders`, hem `Not` sınıflarını içeren küçük bir konsol uygulaması).

⁂

harici hiç bir kütüphane kullanmadan basit yaklaşım ile kullanıcının bilgilerinde telefon bilgisi formatı, mail adresi formatı içeren kapsamlı kargo gönderi ve takip uygulamanın açıklamalı sınıfları, fonksiyonları ve açıklamalar bittikten sonra tek çalışır dosyası (doğrudan kullanılan örnektir menü içermez)

Aşağıda:

1. Önce mimari ve sınıflar açıklanıyor.
2. En sonda aynı mantığın "menüsüz, doğrudan çalışan tek dosya" örneği veriliyor.

Harici kütüphane yok; sadece temel `if`, `len`, `in`, string işlemleri kullanılıyor.[43]

Genel tasarım fikri

- Amaç: Gönderici ve alıcı bilgilerini alan, telefon ve e-posta formatını kabaca kontrol eden, kargoya bir takip numarası verip durumunu gösteren basit bir uygulama.[44]
- Basit doğrulama yaklaşımı:
  - Telefon: Sadece rakam ve `+` işareti, uzunluk 10–15 karakter arası (örnek, gerçek hayatta daha karmaşık olabilir).[45] [46]
  - E-posta: İçinde bir adet `@`, en az bir `.`, `@`'ten önce/sonra boş olmama gibi basit kurallar.[47] [48]

Sınıflar ve fonksiyonlar (açıklamalı)

1) Basit doğrulama fonksiyonları

Bu fonksiyonlar genel yardımcı (utility) fonksiyon; herhangi bir sınıfa bağlı değil.

```
def is_valid_phone(phone: str) -> bool:
    """
    Basit telefon kontrolü:
    - Sadece rakam ve '+' içerebilir
    - Uzunluk 10 ile 15 arasında
    """
    if not phone:
        return False

    # Başta '+' varsa onu yok sayıp kalanının rakam olup olmadığına bak
    if phone[^3_0] == '+':
        digits = phone[1:]
    else:
        digits = phone

    if not digits.isdigit():
        return False

    return 10 <= len(digits) <= 15


def is_valid_email(email: str) -> bool:
    """
    Çok basit email kontrolü:
    - '@' ve '.' içermeli
    - '@' bir kez geçmeli
    - '@' öncesi ve sonrası boş olmamalı
    - '@' sonrası kısımda en az bir nokta olmalı
    """
    if not email or '@' not in email:
        return False

    if email.count('@') != 1:
        return False
```

```
    local, domain = email.split('@')

    if not local or not domain:
        return False

    if '.' not in domain:
        return False

    return True
```

Bu yaklaşımlar “sözde” doğrulama; gerçek sistemde daha ayrıntılı kurallar ve belki regex kullanılır ama burada mantığı basit tutmak için bu yeterli.[49] [47] [45]

2) Kullanıcı ve adres sınıfları

```
class Address:
    """
    Gönderici / alıcı adresi için basit sınıf.
    """

    def __init__(self, city: str, district: str, details: str):
        self.city = city
        self.district = district
        self.details = details

    def __str__(self) -> str:
        return f"{self.details}, {self.district}, {self.city}"


class User:
    """
    Kargo gönderen ya da alan kişi.
    Telefon ve email için basit format kontrolü yapılır.
    """

    def __init__(self, name: str, phone: str, email: str, address: Address):
        if not is_valid_phone(phone):
            raise ValueError(f"Geçersiz telefon formatı: {phone}")

        if not is_valid_email(email):
            raise ValueError(f"Geçersiz email formatı: {email}")

        self.name = name
        self.phone = phone
        self.email = email
        self.address = address

    def __str__(self) -> str:
        return (
            f"{self.name} | Tel: {self.phone} | E-posta: {self.email} | "
            f"Adres: {self.address}"
        )
```

Burada:

- Kullanıcı oluşturulurken telefon/mail yanlışsa `ValueError` fırlatılarak uygulamanın erken hata yakalaması sağlanıyor.[43]
- `__str__` ile hem `User` hem `Address` insan okunur biçimde yazdırılabiliyor.

3) Kargo ve takip sistemi

```
class Shipment:
    """
    Tek bir kargo gönderisini temsil eder.
    """

    def __init__(self, tracking_number: str, sender: User, receiver: User, weight_kg: flo
        self.tracking_number = tracking_number
        self.sender = sender
        self.receiver = receiver
        self.weight_kg = weight_kg
        self.status = "Hazırlanıyor"  # başlangıç durumu

    def update_status(self, new_status: str):
        """
        Kargo durumunu günceller.
        """
        allowed = [
            "Hazırlanıyor",
            "Kargoya verildi",
            "Dağıtım merkezinde",
            "Dağıtıma çıktı",
            "Teslim edildi",
        ]
        if new_status not in allowed:
            raise ValueError(f"Geçersiz durum: {new_status}")
        self.status = new_status

    def __str__(self) -> str:
        return (
            f"Takip No: {self.tracking_number}\n"
            f"Gönderici: {self.sender}\n"
            f"Alıcı: {self.receiver}\n"
            f"Ağırlık: {self.weight_kg} kg\n"
            f"Durum: {self.status}"
        )
```

Bu sınıf:

- Gönderici/alıcıyı `User` tipi ile ilişkilendiriyor (kompozisyon).[43]
- Durum güncelleme için basit bir durum listesi kullanıyor.

4) Kargo yöneticisi (birden çok gönderi tutmak için)

```
class ShipmentManager:
    """
    Birden fazla kargoyu tutan ve takip numarası ile arama/yönetim sağlayan sınıf.
    """

    def __init__(self):
        self._shipments = {}  # tracking_number -> Shipment

    def add_shipment(self, shipment: Shipment):
        if shipment.tracking_number in self._shipments:
            raise ValueError(f"Bu takip numarası zaten kayıtlı: {shipment.tracking_numbe
        self._shipments[shipment.tracking_number] = shipment

    def get_shipment(self, tracking_number: str) -> Shipment:
        return self._shipments.get(tracking_number)

    def update_status(self, tracking_number: str, new_status: str):
        shipment = self.get_shipment(tracking_number)
        if shipment is None:
            raise KeyError(f"Takip numarası bulunamadı: {tracking_number}")
        shipment.update_status(new_status)

    def list_all(self):
        """
        Tüm kargoları listelemek için basit yardımcı (örnek çıktı).
        """
        for s in self._shipments.values():
            print("=" * 40)
            print(s)
```

Bu yönetici ile:

- Sistemdeki tüm gönderiler tek bir yerde tutuluyor.
- Takip numarası ile arama ve durum güncelleme yapılabiliyor.[44]

Tam çalışan tek dosya örneği (menüsüz, doğrudan kullanım)

Aşağıdaki kodu `kargo_uygulamasi.py` gibi tek bir dosyaya yapıştırıp çalıştırabilirsin; çalıştığında yalnızca örnek senaryoyu oluşturur ve çıktıyı ekrana yazar, kullanıcıdan giriş almaz.

```
# ================= Yardımcı doğrulama fonksiyonları =================

def is_valid_phone(phone: str) -> bool:
    """
    Basit telefon kontrolü:
    - Sadece rakam ve '+' içerebilir
    - Uzunluk 10 ile 15 arasında
    """
    if not phone:
        return False
```

```
    if phone[^3_0] == '+':
        digits = phone[1:]
    else:
        digits = phone

    if not digits.isdigit():
        return False

    return 10 <= len(digits) <= 15


def is_valid_email(email: str) -> bool:
    """
    Basit e-posta kontrolü:
    - '@' bir kez geçmeli
    - '@' öncesi/sonrası boş olmamalı
    - '@' sonrası kısımda en az bir nokta (.) olmalı
    """
    if not email or '@' not in email:
        return False

    if email.count('@') != 1:
        return False

    local, domain = email.split('@')

    if not local or not domain:
        return False

    if '.' not in domain:
        return False

    return True


# ================= Temel veri sınıfları =================

class Address:
    """
    Gönderici / alıcı adresi.
    """

    def __init__(self, city: str, district: str, details: str):
        self.city = city
        self.district = district
        self.details = details

    def __str__(self) -> str:
        return f"{self.details}, {self.district}, {self.city}"


class User:
    """
    Kullanıcı (gönderici veya alıcı).
    Telefon ve e-posta formatı basitçe doğrulanır.
```

```
    """

    def __init__(self, name: str, phone: str, email: str, address: Address):
        if not is_valid_phone(phone):
            raise ValueError(f"Geçersiz telefon formatı: {phone}")

        if not is_valid_email(email):
            raise ValueError(f"Geçersiz e-posta formatı: {email}")

        self.name = name
        self.phone = phone
        self.email = email
        self.address = address

    def __str__(self) -> str:
        return (
            f"{self.name} | Tel: {self.phone} | E-posta: {self.email} | "
            f"Adres: {self.address}"
        )


# ================= Kargo ve yönetici sınıfları =================

class Shipment:
    """
    Tek bir kargo gönderisi.
    """

    def __init__(self, tracking_number: str, sender: User, receiver: User, weight_kg: flo
        self.tracking_number = tracking_number
        self.sender = sender
        self.receiver = receiver
        self.weight_kg = weight_kg
        self.status = "Hazırlanıyor"

    def update_status(self, new_status: str):
        """
        Kargo durumunu güncelle.
        """
        allowed = [
            "Hazırlanıyor",
            "Kargoya verildi",
            "Dağıtım merkezinde",
            "Dağıtıma çıktı",
            "Teslim edildi",
        ]
        if new_status not in allowed:
            raise ValueError(f"Geçersiz durum: {new_status}")
        self.status = new_status

    def __str__(self) -> str:
        return (
            f"Takip No: {self.tracking_number}\n"
            f"Gönderici: {self.sender}\n"
            f"Alıcı: {self.receiver}\n"
            f"Ağırlık: {self.weight_kg} kg\n"
```

```
            f"Durum: {self.status}"
        )


class ShipmentManager:
    """
    Birden çok kargoyu saklayan basit yönetici.
    """

    def __init__(self):
        self._shipments = {}  # tracking_number -> Shipment

    def add_shipment(self, shipment: Shipment):
        if shipment.tracking_number in self._shipments:
            raise ValueError(f"Takip numarası zaten var: {shipment.tracking_number}")
        self._shipments[shipment.tracking_number] = shipment

    def get_shipment(self, tracking_number: str) -> Shipment:
        return self._shipments.get(tracking_number)

    def update_status(self, tracking_number: str, new_status: str):
        shipment = self.get_shipment(tracking_number)
        if shipment is None:
            raise KeyError(f"Takip numarası bulunamadı: {tracking_number}")
        shipment.update_status(new_status)

    def list_all(self):
        for s in self._shipments.values():
            print("=" * 50)
            print(s)


# ================= ÖRNEK KULLANIM (menüsüz, doğrudan çalışan kısım) =================

def main():
    # Örnek adresler
    sender_address = Address("İstanbul", "Kadıköy", "Rıhtım Caddesi No:10")
    receiver_address = Address("Ankara", "Çankaya", "Atatürk Bulvarı No:20")

    # Örnek kullanıcılar
    sender = User(
        name="Ahmet Yılmaz",
        phone="+905551112233",
        email="ahmet.yilmaz@example.com",
        address=sender_address,
    )

    receiver = User(
        name="Ayşe Demir",
        phone="+905554445566",
        email="ayse.demir@example.org",
        address=receiver_address,
    )

    # Kargo yöneticisi
    manager = ShipmentManager()
```

```
    # Örnek kargo oluştur
    shipment = Shipment(
        tracking_number="TR123456789",
        sender=sender,
        receiver=receiver,
        weight_kg=2.5,
    )

    # Sisteme ekle
    manager.add_shipment(shipment)

    # Durumları biraz güncelle
    manager.update_status("TR123456789", "Kargoya verildi")
    manager.update_status("TR123456789", "Dağıtım merkezinde")
    manager.update_status("TR123456789", "Dağıtıma çıktı")
    manager.update_status("TR123456789", "Teslim edildi")

    # Tüm kargoları ekrana yazdır
    manager.list_all()


if __name__ == "__main__":
    main()
```

Bu yapıdan sonra istersen:

- Kullanıcıdan `input()` ile bilgiler alıp aynı sınıfları kullanarak interaktif bir menü ekleyebilir,
- Veya `ShipmentManager`'a basit dosya kaydet/yükle fonksiyonları ekleyerek veriyi kalıcı hale getirebilirsin. [50] [43]

⁂

1. https://www.datacamp.com/tutorial/python-dunder-methods
2. https://www.pythonlikeyoumeanit.com/Module4_OOP/Special_Methods.html
3. https://docs.python.org/3/reference/datamodel.html
4. https://uwpce-pythoncert.github.io/PythonCertDevel/modules/SubclassingAndInheritance.html
5. https://www.w3schools.com/python/python_inheritance.asp
6. https://www.datamentor.io/python/inheritance-method-overriding
7. https://www.programiz.com/python-programming/methods/built-in/super
8. https://realpython.com/python-super/
9. https://www.geeksforgeeks.org/python/method-overriding-in-python/
10. https://stackoverflow.com/questions/3277367/how-does-pythons-super-work-with-multiple-inheritance
11. https://stackoverflow.com/questions/753640/inheritance-and-overriding-init-in-python
12. https://www.pythontutorials.net/blog/inheritance-and-overriding-init-in-python/
13. https://stackoverflow.com/questions/51562492/full-list-of-python-special-method-names
14. https://diveintopython3.net/special-method-names.html
15. https://python.land/objects-and-classes/python-inheritance

16. https://stackoverflow.com/questions/12764995/python-overriding-an-inherited-class-method
17. https://docs.vultr.com/python/built-in/super
18. https://www.thedigitalcatonline.com/blog/2014/05/19/method-overriding-in-python/
19. https://www.geeksforgeeks.org/python/python-super/
20. https://www.reddit.com/r/learnpython/comments/12qhzi6/dataclasses_with_inheritance/
21. https://www.w3schools.com/python/python_inheritance.asp
22. https://www.programiz.com/python-programming/object-oriented-programming
23. https://www.datamentor.io/python/inheritance-method-overriding
24. https://www.pythontutorial.net/python-oop/python-overriding-method/
25. https://www.programiz.com/python-programming/inheritance
26. https://yasirbhutta.github.io/python/docs/oop-inheritance/
27. https://uwpce-pythoncert.github.io/ProgrammingInPython/modules/SubclassingAndInheritance.html
28. https://www.pythonlikeyoumeanit.com/Module4_OOP/Special_Methods.html
29. https://docs.python.org/3/reference/datamodel.html
30. https://realpython.com/inheritance-composition-python/
31. https://stackoverflow.com/questions/12764995/python-overriding-an-inherited-class-method
32. https://www.geeksforgeeks.org/python/inheritance-in-python/
33. https://www.geeksforgeeks.org/python/method-overriding-in-python/
34. https://www.reddit.com/r/learnpython/comments/s5u8f8/ideas_for_beginner_oop_projects_for_python/
35. https://www.scientecheasy.com/2023/10/method-overriding-in-python.html/
36. https://www.tutorialspoint.com/python/python_method_overriding.htm
37. https://www.reddit.com/r/learnpython/comments/16ft7vt/looking_for_real_examples_of_oop_in_python/
38. https://www.youtube.com/watch?v=dqTMKvSHUNI
39. https://www.w3resource.com/python-exercises/oop/index.php
40. https://www.youtube.com/watch?v=4Y83cUbDKZ8
41. https://www.youtube.com/watch?v=H3T-dQmwMfs
42. https://www.youtube.com/watch?v=OwAovKTBoNE
43. https://realpython.com/python3-object-oriented-programming/
44. https://www.cliffsnotes.com/study-notes/24779971
45. https://uibakery.io/regex-library/phone-number-python
46. https://stackoverflow.com/questions/16135069/how-do-i-validate-a-mobile-number-using-python
47. https://mailfloss.com/automating-email-validation-with-python-step-by-step-tutorial/
48. https://mailtrap.io/blog/python-validate-email/
49. https://www.mailercheck.com/articles/email-validation-using-python
50. https://www.youtube.com/watch?v=S7YAh0bqcR4
51. https://github.com/ayush-thakur02/object-tracking-opencv
52. https://www.instructables.com/Object-Tracking-With-Opencv-and-Python-With-Just-5/
53. https://pyimagesearch.com/2018/07/23/simple-object-tracking-with-opencv/

54. https://github.com/vizionapi/shipment-tracker-api-python
55. https://www.abstractapi.com/guides/api-functions/phone-number-validation-in-python
56. https://www.geeksforgeeks.org/python/check-if-email-address-valid-or-not-in-python/
57. https://www.youtube.com/watch?v=vtwixS_Ug0I
58. https://docs.ultralytics.com/modes/track/
59. https://www.modernagecoders.com/blog/email-validation-in-python
60. https://www.reddit.com/r/learnpython/comments/1aydw57/python_phone_number_validation/
61. https://cdn3.f-cdn.com/files/download/280010292/transportation-logistics-syste.pdf
62. https://www.youtube.com/watch?v=hf8FvV5UZdo