2024-2025 EĞİTİM VE ÖĞRETİM YILI

PROGRAMLAMA DİLLERİ MODÜLÜ (PYTHON)

3. ÜNİTE

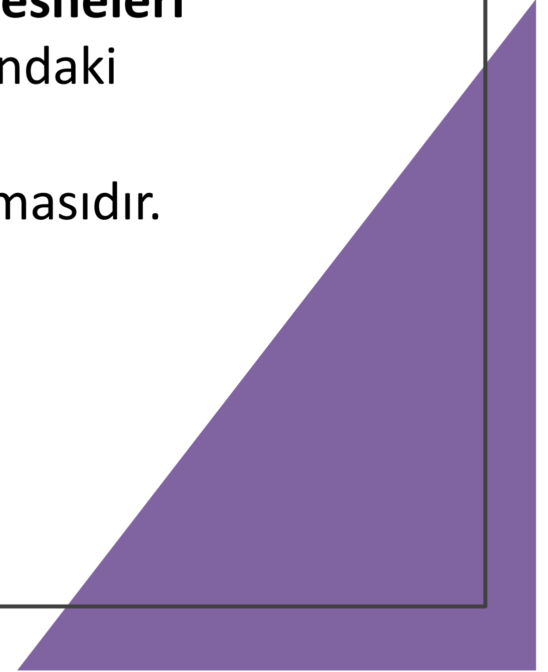
NESNE YÖNELİMLİ PROJE GELİŞTİRME

Kaynak: onuraltuntas.net

Kaynak: onuraltuntas.net

NESNE YÖNELİMLİ PROGRAMLAMA

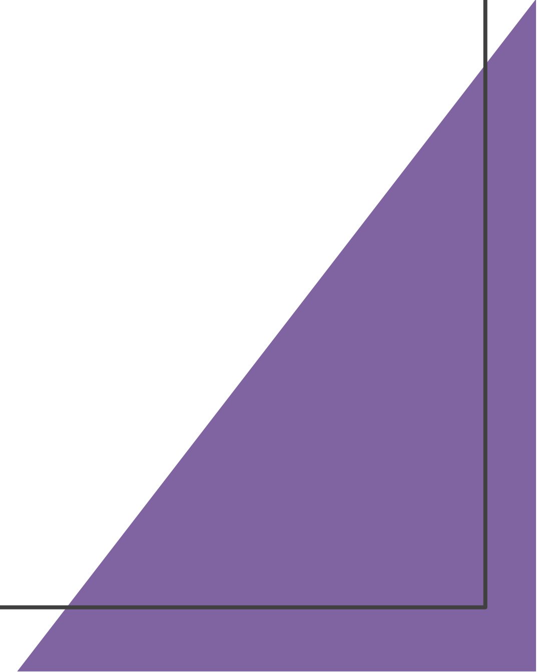
Nesne Yönelimli Programlama (Object-Oriented Programming - OOP), yazılım geliştirme sürecinde **nesneleri** ve bu nesnelerin arasındaki ilişkileri temel alan bir programlama paradigmasıdır.



Kaynak: onuraltuntas.net

NESNE YÖNELİMLİ PROGRAMLAMA

- Python, nesne yönelimli bir dildir ve bu yaklaşımda, gerçek hayattaki varlıkları programlama dünyasına aktarmak için **sınıflar** (class) ve **nesneler** (object) kullanılır.



Kaynak: onuraltuntas.net

NESNE KAVRAMI

- Nesne, bir sınıfın örneğidir ve **veriler** ile bu verilere uygulanacak **davranışlardan** oluşur. Örneğin, bir "Araba" sınıfını ele alırsak, bu sınıftan üretilen bir nesne, bir arabayı temsil eder.

Bir nesnenin iki temel öğesi vardır:

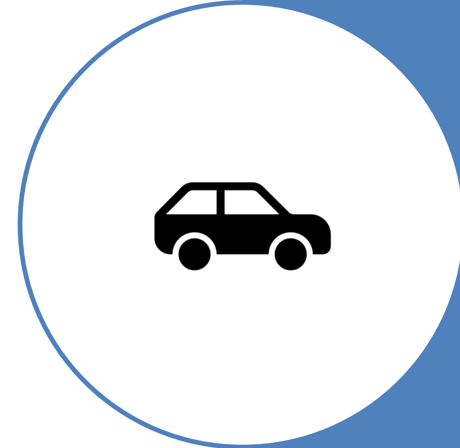
1. **Methods (Metotlar)**: Nesnenin gerçekleştirebileceği davranışlardır. Metotlar, sınıfın içinde tanımlanan **fonksiyonlardır** ve genellikle nesnenin niteliklerini işlemek veya onlara erişmek için kullanılır.

Örneğin, "Araba" nesnesi için "hızlan", "yavaşla" veya "kornaya bas" gibi davranışlar metotlardır.

Kaynak: onuraltuntas.net

Bir nesnenin iki temel öğesi vardır:

1. **Attributes (Nitelikler)**: Nesnenin sahip olduğu özelliklerdir. Bu özellikler, sınıf içinde tanımlanır ve bir nesnenin durumunu ifade eder. Örneğin, bir "Araba" nesnesinin rengi, markası, modeli gibi bilgiler niteliklerdir.



Python'da nitelikler, sınıf içinde tanımlanan **değişkenlerdir**.

Kaynak: onuraltuntas.net

SINIF YAPISI (SADECE İNCELEYİN, AYRINTILAR DAHA SONRA)

```python
# Sınıf tanımı
class Araba:
    # Yapıcı metot (__init__) ile niteliklerin tanımlanması
    def __init__(self, marka, model, renk):
        self.marka = marka  # Nitelik
        self.model = model  # Nitelik
        self.renk = renk    # Nitelik

    # Metot tanımı
    def bilgileri_goster(self):
        print(f"Marka: {self.marka}, Model: {self.model}, Renk: {self.renk}")

    def kornaya_bas(self):
        print(f"{self.marka} {self.model} kornaya bastı! Bip bip!")

# Nesne oluşturma
araba1 = Araba("Toyota", "Corolla", "Kırmızı")

# Niteliklere erişim
print(araba1.renk)  # "Kırmızı"

# Metot çağırma
araba1.bilgileri_goster()  # Marka: Toyota, Model: Corolla, Renk: Kırmızı
araba1.kornaya_bas()       # Toyota Corolla kornaya bastı! Bip bip!
```

Kaynak: onuraltuntas.net

Kaynak: onuraltuntas.net

Attribute ve Metotların Detayları

Nitelikler (Attributes):

- self.marka, self.model, ve self.renk gibi sınıfın yapıcı metodu (__init__) içinde tanımlanır.

- Her nesne, kendi niteliklerine sahiptir ve bu nitelikler, nesneye özel değerler taşır.

Kaynak: onuraltuntas.net

Attribute ve Metotların Detayları

- **Metotlar (Methods)**:
  - Sınıf içinde tanımlanan fonksiyonlardır ve genellikle nesne ile ilgili işlemleri gerçekleştirir.

  bilgileri_goster(self) , kornaya_bas(self)

  - İlk parametreleri her zaman self olmalıdır, bu parametre, metotların nesneye erişmesini sağlar.

Nesne Yönelimli Programlama İle Yordamsal Programlama Arasındaki Fark

- İkisi de yazılım geliştirme için kullanılan farklı yaklaşımlardır.
- İkisi arasındaki temel farklar, programların organizasyonu, veri ve davranışların ayrımı, ve çözüm yöntemlerindeki bakış açısıyla ilgilidir.

Yaklaşım

- Programlar, bir dizi talimat veya prosedürden (fonksiyonlardan) oluşur.
- Veri ve işlemler (davranışlar) birbirinden bağımsızdır.

Yordamsal Programlama (Procedural Programming)

Amaç

- Problemi küçük ve bağımsız alt görevler (fonksiyonlar) hâline getirerek çözmek.

Veri Yönetimi

- Veriler genellikle **global değişkenler** ile saklanır ve bu değişkenler, programın farklı yerlerinden erişilebilir. Bu, veri bütünlüğü sorunlarına yol açabilir.

Yordamsal Programlama (Procedural Programming)

Avantajları:

- Basit ve küçük projelerde etkili.
- Anlaması ve uygulaması kolaydır.

Dezavantajları:

- Büyük ve karmaşık projelerde kodun okunabilirliği ve yönetilebilirliği zorlaşır.
- Veri ve davranışların ayrılığı nedeniyle, bir değişiklik yapmak programın diğer bölümlerini etkileyebilir.

Kaynak: onuraltuntas.net

Yordamsal Programlama (Procedural Programming)

Avantajları:

- Basit ve küçük projelerde etkili.
- Anlaması ve uygulaması kolaydır.

Dezavantajları:

- Büyük ve karmaşık projelerde kodun okunabilirliği ve yönetilebilirliği zorlaşır.
- Veri ve davranışların ayrılığı nedeniyle, bir değişiklik yapmak programın diğer bölümlerini etkileyebilir.

Yordamsal Programlama (Procedural Programming) Örnek Kod Yapısı

```python
# Yordamsal yaklaşım
def toplama(a, b):
    return a + b

def carpma(a, b):
    return a * b

# Fonksiyonların çağrılması
x = 5
y = 10
print("Toplama:", toplama(x, y))  # 15
print("Çarpma:", carpma(x, y))    # 50
```

Kaynak: onuraltuntas.net

Yaklaşım

Programlar, **nesneler** ve bu nesneler arasındaki ilişkiler üzerinden organize edilir.

Nesneler, hem verileri (**nitelikler**) hem de davranışları (**metotlar**) bir arada tutar.

Amaç

Gerçek hayattaki varlıkları yazılım dünyasına yansıtmak ve her bir varlık için bir sınıf (class) tanımlamak.

Veri Yönetimi

Veriler nesneye özgüdür ve nesneler dışarıdan izole edilmiştir. Bu, veri bütünlüğünü artırır.

Nesne Yönelimli Programlama (Object-Oriented Programming - OOP)

Avantajları

Büyük ve karmaşık projelerde daha düzenli ve yeniden kullanılabilir bir yapı sunar.
Kod tekrarını önler, çünkü sınıflar bir kez yazılır ve tekrar tekrar kullanılabilir.
Kolay bakım ve genişletilebilirlik sağlar.

Dezavantajları

Küçük projelerde gereksiz yere karmaşık olabilir.
OOP'nin yapısını öğrenmek başlangıçta daha zordur.

Kaynak: onuraltuntas.net

Nesne Yönelimli Programlama (Object-Oriented Programming - OOP) Örnek Kod Yapısı

```python
# Nesne yönelimli yaklaşım
class Matematik:
    def __init__(self, a, b):
        self.a = a
        self.b = b

    def toplama(self):
        return self.a + self.b

    def carpma(self):
        return self.a * self.b

# Nesne oluşturma
matematik = Matematik(5, 10)
print("Toplama:", matematik.toplama())  # 15
print("Çarpma:", matematik.carpma())    # 50
```

Kaynak: onuraltuntas.net

Dilin Kurallarına Uygun Bir Nesne Tanımlaması

Sınıf Tanımlama (Class Definition)

class anahtar kelimesi kullanılarak bir sınıf tanımlanır.

Sınıf isimleri genellikle **PascalCase** ile yazılır (örneğin: Araba, KullaniciBilgisi).

ÖRNEK

```
class Araba:
```

Dilin Kurallarına Uygun Bir Nesne Tanımlaması

Yapıcı Metot (Constructor)

Sınıfın bir nesnesi oluşturulduğunda otomatik olarak çağrılan, genellikle nesneye ait niteliklerin başlangıç değerlerini belirlemek için kullanılan özel bir metottur.

Python'da bu metot **__init__** adını taşır ve ilk parametresi **self** olmalıdır.

ÖRNEK

```
def __init__(self, marka, model, yil):
    ................
    ................
```

Dilin Kurallarına Uygun Bir Nesne Tanımlaması

Nitelikler (Attributes)

Nitelikler, sınıfa ait verileri saklar.

self kullanılarak sınıf içinde tanımlanır ve nesneye özel hale getirilir.

Gerektiğinde niteliklere dışarıdan erişim ve düzenleme yapılabilir.

ÖRNEK

self.marka = marka

self.model = model

Kaynak: onuraltuntas.net

Dilin Kurallarına Uygun Bir Nesne Tanımlaması

Metotlar (Methods)

Metotlar, nesneye özel davranışları (işlevleri) tanımlar.

İlk parametresi self olmalıdır; bu, nesneye erişimi sağlar.

ÖRNEK

```
def bilgileri_goster(self):
```

Kaynak: onuraltuntas.net

Dilin Kurallarına Uygun Bir Nesne Tanımlaması

Yazım Kuralları

Python'un yazım kurallarına (örneğin, snake_case kullanımı) ve kodlama standartlarına (PEP 8) uyulmalıdır. (Araştırınız)

Erişim Düzeni

Gerektiğinde nitelik ve metotlar için **özel (private)** veya **korumalı (protected)** erişim düzeyleri uygulanabilir.

- Özel nitelikler/metotlar: __isim şeklinde tanımlanır.
- Korumalı nitelikler/metotlar: _isim şeklinde tanımlanır.

Kaynak: onuraltuntas.net

Dilin Kurallarına Uygun Bir Nesne Tanımlaması

`_isim` ile `__isim` Arasındaki Fark

Özellik	`_isim`	`__isim`
Erişim Seviyesi	Korumalı (Protected). Uyarı amaçlıdır.	Özel (Private). Doğrudan dışarıdan erişim engellenir.
Alt Sınıflarda Kullanım	Alt sınıflar tarafından erişilebilir.	Alt sınıflar için de gizlidir.
Örnek	`_marka`, `_model`	`__marka`, `__model`

Dilin Kurallarına Uygun Bir Nesne Tanımlaması

Dil Kurallarına Uygunluk İçin Dikkat Edilmesi Gerekenler

Anlamlı İsimlendirme:

- Sınıf, nitelik ve metot adları anlamlı ve tanımlayıcı olmalıdır.

Yapılandırılmış Kodlama:

- Kod, okunabilir ve düzenli bir şekilde yazılmalıdır.
- Gereksiz tekrarlar (redundancy) olmamalıdır.

Yazım ve Biçim Kuralları:

- Python'un girinti (indentation) kurallarına uygun yazılmalıdır (her blok için 4 boşluk).
- Yorumlar açıklayıcı olmalı ve kodla çelişmemelidir.

DİL KURALLARI ÖRNEK KOD YAPISI (UYGULAYINIZ)

```
class Araba:
    # Yapıcı metot (__init__)
    def __init__(self, marka, model, yil):
        self.marka = marka  # Genel nitelik
        self.model = model  # Genel nitelik
        self.__yil = yil    # Özel nitelik (dışarıdan doğrudan erişim yok)

    # Niteliklere erişim için getter metotları
    def get_yil(self):
        return self.__yil

    # Niteliklerin değiştirilmesi için setter metotları
    def set_yil(self, yeni_yil):
        if yeni_yil > 1885:  # Mantıksal kontrol (ilk otomobilin üretim yılı)
            self.__yil = yeni_yil
        else:
            print("Geçersiz yıl!")

    # Genel bir metot
    def bilgileri_goster(self):
        return f"Marka: {self.marka}, Model: {self.model}, Üretim Yılı: {self.__yil}"

# Nesne oluşturma
araba1 = Araba("Toyota", "Corolla", 2020)

# Metotların ve niteliklerin kullanımı
print(araba1.bilgileri_goster())  # Marka: Toyota, Model: Corolla, Üretim Yılı: 2020
araba1.set_yil(2022)           # Yılı güncelle
print(araba1.get_yil())        # 2022
```

Kaynak: onuraltuntas.net

Kaynak: onuraltuntas.net

Dosya Nesneleri Nedir?

Bilgilerin kalıcı olarak saklanması veya okunması gereken durumlarda kullanılır. Dosya nesneleri, bir dosyayla çalışmayı mümkün kılar.

Örneğin:

- Verileri bir dosyaya yazmak (kayıt/log tutma).
- Bir dosyadan veri okumak (örneğin, ayar dosyalarını yüklemek).

Kaynak: onuraltuntas.net

Dosya Nesneleri Gerekli Olan Projeler

Dosya nesneleri genellikle **veri depolama ve yönetimi** gerektiren projelerde tercih edilir:

Veri Günlüğü ve Loglama

- Uygulamaların hata günlüklerini saklamak için dosya nesneleri gerekir.
- Örnek: Bir web sunucusunun hata raporları, bir oyun uygulamasının skor kayıtları.

Veritabanı Alternatifi Olarak Kullanım

- Basit projelerde bir veritabanı yerine düz metin dosyaları, JSON veya CSV dosyaları kullanılarak veri saklanabilir.
- Örnek: Bir öğrenci bilgi sistemi, günlük görev yönetimi uygulaması.

Metin veya Veri İşleme

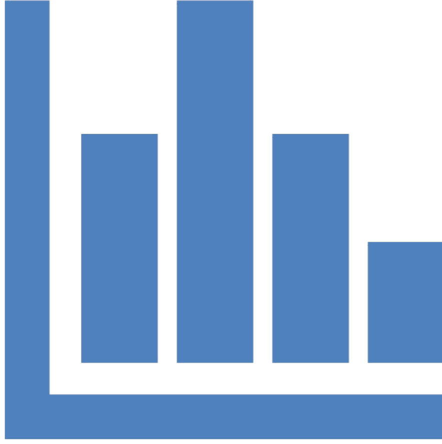
- Kullanıcıdan alınan bilgileri dosyalara kaydetmek ya da dosyalardan okuyarak işlem yapmak.
- Örnek: Bir metin düzenleyici, şifre yöneticisi uygulaması.

Oyunlar

- Oyun kayıtları, oyuncu skorları veya ayar dosyalarını saklamak için.
- Örnek: Bir satranç oyunu kaydettiği oyunları bir dosyada tutabilir.

Kaynak: onuraltuntas.net

Grafik Nesneleri Nedir?



Grafiksel görseller veya kullanıcı arayüzü tasarımlarında kullanılır.

Çizim yapmak, şekiller oluşturmak, görselleştirme yapmak ya da kullanıcılarla etkileşimli grafikler geliştirmek için kullanılır.

Kaynak: onuraltuntas.net

Grafik Nesneleri Gerekli Olan Projeler

Grafik nesneleri genellikle **görselleştirme, kullanıcı arayüzü tasarımı ve çizim** gerektiren projelerde tercih edilir:

Eğitim ve Öğrenim Uygulamaları

- Geometrik şekiller, grafikler veya etkileşimli görseller ile eğitim içerikleri oluşturma.
- Örnek: Matematik eğitim yazılımında üçgen, çember veya histogram çizimleri.

Veri Görselleştirme

- Büyük veri setlerinin analiz sonuçlarını görsel olarak sunmak.
- Örnek: Bir satış grafiği uygulaması, hava durumu verileri için çizelgeler.

Tasarım ve Çizim Uygulamaları

- Dijital çizim araçları veya tasarım programlarında çizim nesneleri oluşturmak.
- Örnek: Bir logo tasarım uygulaması.

Oyun Geliştirme

- Oyunlardaki karakterler, arka planlar ve diğer görsel öğelerin oluşturulması.
- Örnek: 2D platform oyunları için karakter ve engel çizimleri.

Bilimsel ve Teknik Uygulamalar

- Teknik çizimler, modeller veya diyagramlar için grafik nesneler kullanılır.
- Örnek: Elektrik devre tasarım uygulamaları, mühendislik simulasyonları.

Aynı Sınıfa Tanımlanan Verilerde Değişiklik Yapmanın Sağladığı Kolaylıklar ve Örnek Projeler

Kaynak: onuraltuntas.net

Tek Merkezden Değişiklik Yapma Kolaylığı

Senaryo:

Bir e-ticaret uygulaması geliştiriyorsunuz. Her ürün için bir Urun sınıfınız var ve bu sınıfta ürünün fiyatı, adı ve stok miktarı tutuluyor.

Kolaylık:

Sınıf içinde değişiklik yaparak tüm ilgili nesneleri güncelleyebilirsiniz. Örneğin:

- Fiyat hesaplama yöntemi değiştiğinde, yalnızca sınıfta değişiklik yapmanız yeterlidir.
- Programın diğer bölümleri bu sınıfa bağlı olduğu için kodun geri kalanında güncelleme yapmanıza gerek kalmaz.

Aynı Sınıfa Tanımlanan Verilerde Değişiklik Yapmanın Sağladığı Kolaylıklar ve Örnek Projeler

Kaynak: onuraltuntas.net

Kapsülleme (Encapsulation) ile Güvenli Veri Güncelleme

Senaryo:
Bir bankacılık sisteminde müşteri bilgilerini (Musteri) saklıyorsunuz. Bu sınıfta müşteri adı, hesap bakiyesi ve PIN kodu bulunuyor.

Kolaylık:
Kapsülleme sayesinde verilere doğrudan erişim yerine metotlarla erişilir:

- get_bakiye() ve set_bakiye() gibi metotlar, verilerin doğruluğunu ve güvenliğini sağlar.
- Örneğin, negatif bir bakiye değerinin atanmasını engelleyebilirsiniz.

Aynı Sınıfa Tanımlanan Verilerde Değişiklik Yapmanın Sağladığı Kolaylıklar ve Örnek Projeler

Kaynak: onuraltuntas.net

Kodun Tekrar Kullanılabilirliğini Artırma

Senaryo:
Bir oyun geliştirdiniz ve karakter özelliklerini (Karakter) bir sınıfta topladınız. Her karakterin adı, gücü ve can puanı bulunuyor.

Kolaylık:
Tüm karakterler aynı sınıftan türetildiği için bir özelliği (örneğin, can puanı hesaplama) değiştirdiğinizde tüm karakterler otomatik olarak etkilenir. Tek tek her nesne için değişiklik yapmanız gerekmez.

Aynı Sınıfa Tanımlanan Verilerde Değişiklik Yapmanın Sağladığı Kolaylıklar ve Örnek Projeler

Kaynak: onuraltuntas.net

Kodun Okunabilirliği ve Yönetilebilirliği

Senaryo:
Bir araba kiralama uygulamasında her araç için bir Araba sınıfı tanımladınız. Bu sınıfta aracın marka, model ve günlük kira fiyatı gibi özellikleri var.

Kolaylık:
Veriler bir sınıfta toplandığı için sistemdeki her araç nesnesinin özellikleri merkezi bir noktadan düzenlenebilir.
Bu, büyük projelerde kodun daha anlaşılır ve düzenli olmasını sağlar.

Aynı Sınıfa Tanımlanan Verilerde Değişiklik Yapmanın Sağladığı Kolaylıklar ve Örnek Projeler

Kaynak: onuraltuntas.net

Kalıtımla (Inheritance) Verileri Güncelleme Kolaylığı

Senaryo:
Bir ulaşım uygulaması yapıyorsunuz. Arac temel sınıfından Otomobil ve Otobus sınıfları türettiniz. Arac sınıfında motor hacmi ve yakıt türü gibi genel özellikler var.

Kolaylık:
Eğer Arac sınıfında bir değişiklik yaparsanız, bu değişiklik Otomobil ve Otobus sınıflarını da otomatik olarak etkiler.
Kodun farklı yerlerinde aynı bilgiyi değiştirmek yerine, temel sınıfta yapılan bir değişiklik yeterli olur.

Aynı Sınıfa Tanımlanan Verilerde Değişiklik Yapmanın Sağladığı Kolaylıklar ve Örnek Projeler

Kaynak: onuraltuntas.net

Metotlarla Dinamik Değişiklik Yapma

Senaryo:
Bir öğrenci yönetim sistemi için bir Ogrenci sınıfı oluşturuldu. Sınıfta öğrenci adı, notlar ve mezuniyet durumu gibi özellikler var.

Kolaylık:
Öğrenci mezuniyet durumu gibi bir bilgiyi hesaplamak için sınıfın bir metodunu kullanabilirsiniz (mezuniyet_durumu_hesapla()).
Not güncellendiğinde mezuniyet durumu otomatik olarak değişir. Her güncellemede elle kontrol yapmanıza gerek kalmaz.

Aynı Sınıfa Tanımlanan Verilerde Değişiklik Yapmanın Sağladığı Kolaylıklar ve Örnek Projeler

Kaynak: onuraltuntas.net

Esneklik ve Ölçeklenebilirlik

Senaryo:
Bir restoran sipariş sistemi tasarlıyorsunuz. Siparis sınıfında yemek adı, miktar ve toplam fiyat gibi özellikler var.

Kolaylık:
Sınıfta yapılan bir değişiklik sistemi daha esnek hale getirir. Örneğin, indirim oranı eklemek isterseniz, Siparis sınıfına bir nitelik veya metot eklemeniz yeterlidir. Kodun geri kalanı bu değişiklikten etkilenmez.

Örnek çalışma yapınız.

Bir sınıf oluşturarak bu sınıflar içinde nesneler oluşturunuz.

Python'un Açık Kaynak Yapısı ve Geniş Kütüphane Desteği

Python, açık kaynaklı bir programlama dili olduğu için dünya çapında birçok geliştirici tarafından katkı sağlanarak geniş bir kütüphane ekosistemi oluşturulmuştur.

Bu kütüphaneler sayesinde sıfırdan her şeyi yazmak gerekmez; ihtiyaç duyulan birçok özellik hazır olarak sunulmaktadır.

Python’da en çok kullanılan standart kütüphaneler arasında **time** ve **random** modülleri bulunur.

Kaynak: onuraltuntas.net

time Modülü ve Örnek Kullanımı

time modülü, zamanla ilgili işlemler yapmak için kullanılır. Tarih ve saat bilgilerine erişmek, işlemleri belli bir süre durdurmak veya zaman ölçümleri yapmak için bu modülden yararlanılır.

Örnek: Bir Programın Çalışma Süresini Ölçmek

```python
# Başlangıç zamanı
start_time = time.time()

# Örnek işlem
for i in range(1000000):
    pass

# Bitiş zamanı
end_time = time.time()

# Geçen süreyi hesaplama
elapsed_time = end_time - start_time
print(f"Program {elapsed_time:.2f} saniye sürdü.")
```

Kaynak: onuraltuntas.net

Örnek: İşlemi Belirli Süre Bekletmek

```python
import time

print("Geri sayım başlıyor...")
for i in range(5, 0, -1):
    print(i)
    time.sleep(1)  # Her bir sayı için 1 saniye bekle
print("Süre doldu!")
```

Kaynak: onuraltuntas.net

random Modülü ve Örnek Kullanımı

random modülü, rastgele sayı üretme ve seçim yapma işlemleri için kullanılır. Özellikle oyun geliştirme, simülasyonlar ve test verileri oluşturmak için oldukça faydalıdır.

Örnek: Rastgele Sayı Üretmek

```
import random

rastgele_sayi = random.randint(1, 100)  # 1 ile 100 arasında rastgele bir sayı
print(f"Üretilen rastgele sayı: {rastgele_sayi}")
```

Örnek: Bir Listeden Rastgele Seçim Yapmak

```
import random

meyveler = ["Elma", "Armut", "Muz", "Kiraz"]
secim = random.choice(meyveler)  # Listeden rastgele bir eleman seç
print(f"Rastgele seçilen meyve: {secim}")
```

Örnek: str Sınıfının Kullanımı

```python
# Bir string oluşturulması
metin = "Python Programlama"

# String sınıfının metodlarını kullanmak
print(metin.upper())  # Tüm harfleri büyük yapar
print(metin.lower())  # Tüm harfleri küçük yapar
print(metin.split())  # Kelimeleri liste halinde ayırır
print(metin.replace("Python", "Java"))  # Belirtilen kelimeyi değiştirir
```

Kaynak: onuraltuntas.net

Modül Kavramı

Python'da **modül**, birden fazla fonksiyon, sınıf ve değişkeni bir araya toplayan ve başka programlarda kullanılabilir hale getiren dosyalardır.

Modüller sayesinde kod tekrarını önler, düzenli ve modüler bir yapı kurarız.

Python'da modüller, genellikle bir .py dosyası şeklindedir.

Modülün Programa Dahil Edilmesi

Python'da bir modülü programa dahil etmek için import anahtar kelimesi kullanılır.

Örnek: Hazır Bir Modül Kullanımı

```python
# Matematiksel işlemler için math modülünü içe aktarma
import math

# math modülüne ait bazı komutlar
print(math.sqrt(16))  # Karekök hesaplama
print(math.pi)        # Pi sayısı
print(math.factorial(5))  # Faktöriyel hesaplama
```

Kendi Modülümüzü Oluşturma

Python'da kendi modülünüzü yazabilirsiniz. Örneğin, matematik.py adında bir dosya oluşturup içine özel fonksiyonlar ekleyelim.

Adım 1: Modül Dosyasını Oluşturma (matematik.py)

```
# matematik.py
def toplama(a, b):
    return a + b

def carpma(a, b):
    return a * b
```

Adım 2: Modülü Başka Bir Programda Kullanma

```
# Ana program
import matematik

print(matematik.toplama(3, 5))  # Çıktı: 8
print(matematik.carpma(4, 6))   # Çıktı: 24
```

ÖRNEK PROJE

Python dilinde ileri düzey nesne yönelimli programlama (OOP) tekniklerini kullanarak yazılmış bir örnek program

Bu örnek, **kapsülleme**, **kalıtım** ve **çok biçimlilik** gibi OOP ilkelerini içermektedir.

Örnek Program: Bir Hayvanat Bahçesi Yönetim Sistemi

Bu programda, hayvanların sınıfları oluşturulacak ve farklı hayvan türleri arasında kalıtım kullanılacaktır.

Ayrıca, kapsülleme tekniği ile veriler gizlenecek ve çok biçimlilik (polimorfizm) ile aynı işlevi farklı türlerdeki nesnelerle gerçekleştireceğiz.

ÖRNEK PROJE

```python
# Hayvan sınıfı (Ana sınıf)
class Hayvan:
    def __init__(self, isim, yas):
        self.__isim = isim  # Kapsülleme ile gizlenen nitelik
        self.__yas = yas  # Kapsülleme ile gizlenen nitelik

    # Getter metodları
    def get_isim(self):
        return self.__isim

    def get_yas(self):
        return self.__yas

    # Polimorfizm örneği: Her hayvanın ses çıkarması gerektiği varsayılıyor
    def ses_cikar(self):
        pass  # Bu metod alt sınıflarda (kalıtım yoluyla) geçersiz kılınacak
```

Devamı var...

Kaynak: onuraltuntas.net

ÖRNEK PROJE

````
#### **2. Kalıtım ile Alt Sınıflar Oluşturma**

Şimdi, `Hayvan` sınıfından türetilen **Kedi** ve **Köpek** sınıflarını oluşturacağı

```python
# Kedi sınıfı (Alt sınıf)
class Kedi(Hayvan):
    def __init__(self, isim, yas, tur):
        super().__init__(isim, yas)  # Ana sınıfın __init__ metodunu çağırır
        self.__tur = tur  # Kedinin türü

    def get_tur(self):
        return self.__tur

    # Polimorfizm: Kedinin ses çıkarması
    def ses_cikar(self):
        return "Miyav"
````

Devamı var...
````

ÖRNEK PROJE

```python
# Köpek sınıfı (Alt sınıf)
class Kopek(Hayvan):
    def __init__(self, isim, yas, tur):
        super().__init__(isim, yas)  # Ana sınıfın __init__ metodunu çağırır
        self.__tur = tur  # Köpeğin türü

    def get_tur(self):
        return self.__tur

    # Polimorfizm: Köpeğin ses çıkarması
    def ses_cikar(self):
        return "Hav hav"
```

Devamı var...

ÖRNEK PROJE

Çok Biçimlilik (Polimorfizm)

Şimdi, Hayvan sınıfından türetilen **Kedi** ve **Köpek** nesnelerini oluşturup, her iki nesnenin ses_cikar metodunu çağırarak çok biçimliliği (polimorfizmi) gösterelim.

```python
# Ana program
def hayvan_sesi_cikar(hayvan):
    print(f"{hayvan.get_isim()} adlı {hayvan.get_tur()} hayvanı şu sesi çıkarıyor:

# Kedi ve Köpek nesneleri
kedi = Kedi("Minik", 3, "Persian")
kopek = Kopek("Karabas", 5, "Alman Çoban Köpeği")

# Çok biçimlilik: Farklı nesnelerin aynı metodu farklı şekilde çalıştırması
hayvan_sesi_cikar(kedi)  # Minik adlı Persian hayvanı şu sesi çıkarıyor: Miyav
hayvan_sesi_cikar(kopek)  # Karabas adlı Alman Çoban Köpeği hayvanı şu sesi çıkarı
```

List (LİSTELER)

Python’daki **liste** veri türü, birden fazla öğeyi bir arada tutmak için kullanılan bir veri yapısıdır.

Listeler sıralı (ordered), değiştirilebilir (mutable) ve tekrar edilebilir (duplicate) öğeleri saklayabilir.

Listeler, farklı veri türlerinden öğeleri bir arada tutabilir, yani bir listenin içerisinde sayılar, metinler, hatta diğer listeler bir arada yer alabilir.

List (LİSTELER)

Liste Özellikleri:

1.Sıralı (Ordered): Listede öğeler sırasıyla depolanır ve sıralama korunur.

2.Değiştirilebilir (Mutable): Liste, program çalışırken değiştirilebilir. Öğe ekleme, çıkarma, değiştirme gibi işlemler yapılabilir.

3.Tekrar Edilebilir (Duplicate): Aynı öğe bir listede birden fazla kez bulunabilir.

4.Esneklik: Bir listenin öğeleri, farklı veri türlerinde olabilir (örneğin, sayı, metin, boolean vb.).

LİSTE TANIMLAMASI VE KULLANIMI

```python
# Liste örneği
meyveler = ["Elma", "Armut", "Kiraz", "Muz"]  # Metinlerden oluşan bir liste

# Sayılarla oluşturulmuş bir liste
sayilar = [1, 2, 3, 4, 5]

# Farklı veri türleri içeren bir liste (esneklik)
karisik = [3, "Python", True, 5.5, ["liste", "içinde", "liste"]]

# Listeyi yazdıralım
print(meyveler)  # Çıktı: ['Elma', 'Armut', 'Kiraz', 'Muz']
print(sayilar)  # Çıktı: [1, 2, 3, 4, 5]
print(karisik)  # Çıktı: [3, 'Python', True, 5.5, ['liste', 'içinde', 'liste']]
```

Kaynak: onuraltuntas.net

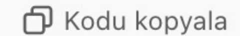
Liste İşlemleri:

1. Listeye Eleman Ekleme (`append`)

python


```python
meyveler.append("Çilek")  # Listeye "Çilek" ekler
print(meyveler)  # Çıktı: ['Elma', 'Armut', 'Kiraz', 'Muz', 'Çilek']
```

2. Liste Elemanını Değiştirme (İndeks ile)

python 


```python
meyveler[1] = "Karpuz"  # 1. indeksteki öğeyi "Karpuz" ile değiştirir
print(meyveler)  # Çıktı: ['Elma', 'Karpuz', 'Kiraz', 'Muz', 'Çilek']
```

3. Liste Elemanını Silme (`remove`)

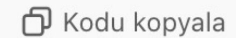
python


```python
meyveler.remove("Muz")  # "Muz" öğesini listeden çıkarır
print(meyveler)  # Çıktı: ['Elma', 'Karpuz', 'Kiraz', 'Çilek']
```

Liste İşlemleri:

4. Listeyi Sıralama (`sort`)

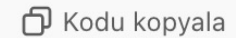
python




```python
meyveler.sort()  # Listeyi alfabetik sıralar
print(meyveler)  # Çıktı: ['Çilek', 'Elma', 'Karpuz', 'Kiraz']
```

5. Liste Uzunluğunu Bulma (`len`)

python




```python
uzunluk = len(meyveler)  # Listenin uzunluğunu döndürür
print(uzunluk)  # Çıktı: 4
```

Kaynak: onuraltuntas.net

TUPLE (DEMETLER)

Python'daki **tuple (demet)** veri türü, **list** veri türüne çok benzer ancak bazı önemli farklar vardır.

Benzerlikler:

Sıralı (Ordered): Tıpkı listeler gibi, tuple'lar da sıralıdır. Öğeler belirli bir sırayla saklanır ve bu sıralama korunur.

Farklı Veri Türlerini Saklama: Listeler gibi, tuple'lar da farklı veri türlerinden öğeler içerebilir.

İçeriğine Erişim: Hem listelerde hem de tuple'larda öğelere indeks numarasıyla (0'dan başlayan bir sıra ile) erişilebilir.

TUPLE (DEMETLER)

Python'daki **tuple (demet)** veri türü, **list** veri türüne çok benzer ancak bazı önemli farklar vardır.

Farklar:

Değiştirilemezlik (Immutable): Listeler değiştirilebilirken (yeni eleman ekleme, çıkarma, değiştirme), tuple'lar değiştirilemez. Yani, tuple'da bir öğe eklemek veya çıkarmak mümkün değildir.

Kapanma: Listeler [] köşeli parantezler ile tanımlanırken, tuple'lar () parantezleri ile tanımlanır.

Tuple Tanımlaması ve Kullanımı:

```python
# Tuple (Demet) tanımlaması
meyveler = ("Elma", "Armut", "Kiraz", "Muz")

# Liste ile benzerlik: Tuple'a da indeks ile erişilebilir
print(meyveler[1])  # Çıktı: Armut
print(meyveler[2])  # Çıktı: Kiraz
```

Tuple'ın Değiştirilemezliği:

Listelerde öğeler değiştirilebilirken, tuple'da bu yapı değiştirilemez. Aşağıdaki örnekte, tuple'da veri değiştirmeye çalıştığımızda hata alırız.

```
# Değiştirilemezlik örneği
meyveler = ("Elma", "Armut", "Kiraz", "Muz")
# Bu satır hata verir çünkü tuple değiştirilemez:
# meyveler[1] = "Karpuz"  # TypeError: 'tuple' object does not support item as
```

Tuple'da Veri Ekleyemez veya Çıkartamazsınız

```python
# Tuple oluşturma
meyveler = ("Elma", "Armut", "Kiraz", "Muz")

# Öğeleri eklemeye çalışmak:
# meyveler.append("Çilek")  # Hata verir, çünkü tuple'da append metodu yok

# Öğeleri çıkarmaya çalışmak:
# meyveler.remove("Armut")  # Hata verir, çünkü tuple'da remove metodu yok
```

Kaynak: onuraltuntas.net

Tuple'da Veri Okuma

Tuple'daki verilere sadece okuma amacıyla erişebilirsiniz, ancak veri üzerinde herhangi bir değişiklik yapamazsınız.

```
# Tuple içeriği sadece okunabilir
print(meyveler[0])  # Çıktı: Elma
print(meyveler[3])  # Çıktı: Muz
```

BİR TUPLE ÖRNEĞİ OLUŞTURUNUZ!

SET (KÜMELER)

Set (küme) veri türü, Python'da sırasız, yinelenen öğelere izin vermeyen ve değiştirilebilir bir veri türüdür. Bir kümenin temel özellikleri şunlardır:

1.Sırasızdır (Unordered): Küme elemanlarının belirli bir sıralaması yoktur.

2.Yinelenen Elemanlara İzin Vermez: Aynı öğe birden fazla kez eklenemez.

3.Değiştirilebilir (Mutable): Eleman eklenip çıkarılabilir.

4.Hızlıdır: Elemanların varlığını kontrol etme veya ekleme gibi işlemler hızlı bir şekilde gerçekleştirilir.

SET (KÜMELER)

ÖRNEK

```
my_set = {1, 2, 3, 4}
print(my_set)  # Çıktı: {1, 2, 3, 4}
```

SET (KÜMELER)

YİNELENEN ÖGELERİN ATILMASI

```
duplicate_set = {1, 2, 2, 3, 4, 4}
print(duplicate_set)  # Çıktı: {1, 2, 3, 4}
```

SET (KÜMELER)

ELEMAN EKLEME ÇIKARMA

```
my_set = {1, 2, 3}
my_set.add(4)  # Yeni eleman ekler
print(my_set)  # Çıktı: {1, 2, 3, 4}

my_set.remove(2)  # Belirli bir elemanı siler
print(my_set)  # Çıktı: {1, 3, 4}
```

SET (KÜMELER)

KESİŞİM BİRLEŞİM İŞLEMLERİ

```
set1 = {1, 2, 3}
set2 = {3, 4, 5}

intersection = set1 & set2  # Kesişim
print(intersection)  # Çıktı: {3}

union = set1 | set2  # Birleşim
print(union)  # Çıktı: {1, 2, 3, 4, 5}
```

SET (KÜMELER)

ELEMAN VARLĞINI KONTOL ETME

```
my_set = {1, 2, 3}
print(2 in my_set)  # Çıktı: True
print(5 in my_set)  # Çıktı: False
```

Dictionary (sözlük)

Dictionary (sözlük), Python'da **anahtar-değer (key-value)** çiftleri ile çalışan bir veri türüdür.

Bu yapıda, her anahtar bir değere karşılık gelir ve bu anahtarlar benzersiz olmalıdır.

Sözlükler, veri erişimini hızlı bir şekilde sağlamak için kullanılır.

Dictionary (sözlük)

ÖRNEK

```python
# Bir öğrencinin bilgilerini tutan sözlük
ogrenci_bilgileri = {
    "ad": "Ali",
    "soyad": "Yılmaz",
    "yas": 17,
    "dersler": ["Matematik", "Fizik", "Kimya"]
}

# Sözlüğü yazdırma
print(ogrenci_bilgileri)
```

Dictionary (sözlük)

UYGULAMA

Bir kitabın adını, yazarını, basım yılını ve türünü tutan bir sözlük tanımlayın ve yazdırın.

Dosya Nesneleri ve Veri Depolama

Python'programında dosya nesneleri, bir çalışması sırasında verileri uzun süreli olarak depolamak veya var olan verilerle işlem yapmak için kullanılır.

Bu yöntem sayesinde verilerin kalıcılığı sağlanır.

Programın hafızasında tutamayacağı kadar büyük veriler, dosyalarda saklanabilir.

Dosyalar aracılığıyla veriler okunabilir, işlenebilir ve düzenlenebilir.

Metin Dosyalarında İşlemler

Dosya Açma

Python'da dosyalar open() fonksiyonu ile açılır. Dosya açma modları:

'r': Okuma (varsayılan)

'w': Yazma (var olan dosyayı siler, yeniden yazar)

'a': Ekleme (mevcut dosyaya veri ekler)

'x': Yeni bir dosya oluşturur (varsa hata verir)

ÖRNEKLER

```python
# Yeni bir dosya oluşturup veri yazma
with open("veri.txt", "w") as dosya:
    dosya.write("Merhaba, bu bir Python dosya yazma örneğidir.\n")
    dosya.write("Bu veri dosyada saklanacak.\n")
print("Veriler dosyaya yazıldı.")
```

ÖRNEKLER

```python
# Dosyadaki verileri okuma
with open("veri.txt", "r") as dosya:
    veri = dosya.read()
print("Dosyadaki içerik:")
print(veri)
```

ÖRNEKLER

```python
# Mevcut dosyaya veri ekleme
with open("veri.txt", "a") as dosya:
    dosya.write("Bu veri sonradan eklendi.\n")
print("Yeni veri dosyaya eklendi.")
```

ÖRNEKLER

```
# Dosyayı satır satır okumak
with open("veri.txt", "r") as dosya:
    for satir in dosya:
        print(satir.strip())  # Satır sonu boşlukları temizlenir
```

ÖRNEKLER – DOSYAYI SİLME

```python
import os

if os.path.exists("veri.txt"):
    os.remove("veri.txt")
    print("Dosya silindi.")
else:
    print("Dosya bulunamadı.")
```

Kaynak: onuraltuntas.net

LİSTELERİ DOSYAYA YAZDIRMA

```python
# Veriler listesi
veriler = ["Trabzonspor", "Python", "Yapay Zeka", "Programlama", "Kodlama"]

# Dosyaya yazma işlemi
dosya_adi = "veriler.txt"

try:
    # Dosyayı yazma modunda açıyoruz
    with open(dosya_adi, "w", encoding="utf-8") as dosya:
        for veri in veriler:
            dosya.write(veri + "\n")  # Her bir öğeyi yeni satıra yazıyoruz

    print(f"Veriler başarıyla '{dosya_adi}' dosyasına yazıldı!")
except Exception as e:
    print("Dosyaya yazma işlemi sırasında bir hata oluştu:", e)
```

DOSYADAKİ VERİYİ LİSTEYE TANIMLAMA

Dosya İçeriği

```
Trabzonspor
Python
Kodlama
```

Kod

```python
# Dosyadan okuma ve listeye aktarma
with open("veriler.txt", "r", encoding="utf-8") as dosya:
    veriler = [satir.strip() for satir in dosya]

print(veriler)
```

Dosya işlemlerinde **with as**, **tell**, ve **seek** metotları

```python
# Dosya yazma
with open("ornek.txt", "w", encoding="utf-8") as dosya:
    dosya.write("Trabzonspor\nPython\nKodlama\n")

# Dosya okuma (tell ve seek kullanımı)
with open("ornek.txt", "r", encoding="utf-8") as dosya:
    print("Başlangıç konumu:", dosya.tell())  # İlk konumu verir
    print(dosya.readline().strip())          # İlk satırı okur
    print("Şu anki konum:", dosya.tell())     # Okuma sonrası konumu verir
    dosya.seek(0)                            # Başlangıca döner
    print("Başlangıca döndük:", dosya.readline().strip())  # İlk satırı tekrar oku
```

ÇIKTI

```
Başlangıç konumu: 0
Trabzonspor
Şu anki konum: 12
Başlangıca döndük: Trabzonspor
```

PROJE TASARLAYIN

Sınıf ve nesneleri kullanarak özgün bir proje tasarlayın.
Proje aşamasında aşağıdakileri uygulayın

Dosya nesneleriyle işlemler yapın.

Hata ayıklama yöntemlerini uygulayın

Tkinter grafik arayüzü oluşturun.

Projenizi tanıtın, yaşadığınız problemleri anlatın!

Python’da grafik arayüzü

Python'da grafik arayüzleri (GUI - Graphical User Interface) oluşturmak için çeşitli hazır kütüphaneler ve araçlar mevcuttur.

Bu araçlar, kullanıcıların görsel öğelerle etkileşime geçebileceği uygulamalar geliştirmeyi kolaylaştırır.

Python’da grafik arayüzü

Python'da en yaygın kullanılan GUI kütüphanelerinden bazıları şunlardır:

Tkinter

PyQt / PySide

Kivy

wxPython

Tkinter örnek kullanım

```
import tkinter as tk

def button_click():
    label.config(text="Merhaba, Python!")

root = tk.Tk()
root.title("Tkinter Örneği")

label = tk.Label(root, text="Hoşgeldiniz!", font=("Arial", 14))
label.pack(pady=20)

button = tk.Button(root, text="Tıklayın", command=button_click)
button.pack(pady=10)

root.mainloop()
```

Kaynak: onuraltuntas.net

Tkinter örnek kullanım

```
import tkinter as tk

def button_click():
    label.config(text="Merhaba, Python!")

root = tk.Tk()
root.title("Tkinter Örneği")

label = tk.Label(root, text="Hoşgeldiniz!", font=("Arial", 14))
label.pack(pady=20)

button = tk.Button(root, text="Tıklayın", command=button_click)
button.pack(pady=10)

root.mainloop()
```

Kaynak: onuraltuntas.net

Kaynak: onuraltuntas.net

PYTHON 3.ÜNİTESİ VE KONULARIN TAMAMI SONA ERDİ!