DRY nedir ? (Software Principles olarak adı geçiyor)

**DRY**, yazılım geliştirme prensiplerinden biridir ve **"Don't Repeat Yourself"** (Kendini Tekrar Etme) anlamına gelir. Bu prensip, yazılım kodunun tekrarını önlemeyi ve kodun daha temiz, bakımı kolay ve hataya açık olmayan bir yapıda olmasını hedefler.

**DRY Prensibinin Temel Amacı**

* **Tekrarı Önlemek**: Aynı kodu birden fazla yerde yazmak yerine, ortak bir fonksiyon veya modül oluşturarak tekrarı önlemek.
* **Bakım Kolaylığı**: Kodda bir değişiklik yapılması gerektiğinde, sadece bir yeri değiştirerek tüm sistemde etkili olmasını sağlamak.
* **Hata Riskini Azaltmak**: Tekrar eden kod, hata yapma olasılığını artırır. DRY prensibi, bu riski azaltır.
* **Okunabilirlik**: Tekrar eden kod, kodun okunabilirliğini azaltır. DRY prensibi, daha temiz ve anlaşılır bir kod yapısı sağlar.

**DRY Prensibine Uymamanın Sonuçları**

* **WET (Write Everything Twice)**: DRY'nin tam tersi bir yaklaşım. Aynı kodu birden fazla yerde yazmak, bakımı zorlaştırır ve hata riskini artırır.
* **Kod Karmaşıklığı**: Tekrar eden kod, projenin büyümesiyle birlikte karmaşık hale gelir.
* **Zaman Kaybı**: Aynı kodu birden fazla yerde güncellemek, zaman kaybına neden olur.

**DRY Prensibi Nasıl Uygulanır?**

1. **Fonksiyonlar ve Metotlar**: Tekrar eden kod parçalarını bir fonksiyon veya metot haline getirin.
   * Örnek:

python

Copy

# DRY olmayan kod

print("Merhaba, Ahmet!")

print("Merhaba, Mehmet!")

print("Merhaba, Ayşe!")

# DRY uyumlu kod

def selam\_ver(isim):

print(f"Merhaba, {isim}!")

selam\_ver("Ahmet")

selam\_ver("Mehmet")

selam\_ver("Ayşe")

1. **Modüller ve Kütüphaneler**: Ortak kullanılan fonksiyonları ayrı bir modül veya kütüphane haline getirin.
   * Örnek:

python

Copy

# utils.py

def toplama(a, b):

return a + b

# main.py

from utils import toplama

print(toplama(3, 5))

1. **Kalıtım (Inheritance)**: Nesne yönelimli programlamada, ortak özellikleri bir üst sınıfta toplayın.
   * Örnek:

python

Copy

class Hayvan:

def ses\_cikar(self):

print("Ses çıkarıyor...")

class Kopek(Hayvan):

def ses\_cikar(self):

print("Hav hav!")

class Kedi(Hayvan):

def ses\_cikar(self):

print("Miyav!")

1. **Veritabanı Normalizasyonu**: Veritabanı tasarımında, tekrar eden verileri önlemek için normalizasyon kurallarını uygulayın.

**DRY Prensibinin Sınırları**

* **Aşırı Genelleme**: Her şeyi tekilleştirmeye çalışmak, kodun anlaşılırlığını bozabilir. Bazen küçük tekrarlar, kodun okunabilirliği için faydalı olabilir.
* **Performans**: Bazı durumlarda, DRY prensibi performans sorunlarına neden olabilir. Örneğin, çok sık çağrılan bir fonksiyon, gereksiz hesaplamalara yol açabilir.

**Örnek: DRY vs WET**

**WET (DRY Olmayan Kod):**

python

Copy

def alan\_hesapla\_dikdortgen(uzunluk, genislik):

return uzunluk \* genislik

def alan\_hesapla\_kare(kenar):

return kenar \* kenar

def alan\_hesapla\_ucgen(taban, yukseklik):

return (taban \* yukseklik) / 2

**DRY Uyumlu Kod:**

python

Copy

def alan\_hesapla(sekil, \*\*kwargs):

if sekil == "dikdortgen":

return kwargs['uzunluk'] \* kwargs['genislik']

elif sekil == "kare":

return kwargs['kenar'] \*\* 2

elif sekil == "ucgen":

return (kwargs['taban'] \* kwargs['yukseklik']) / 2

# Kullanım

print(alan\_hesapla("dikdortgen", uzunluk=5, genislik=10))

print(alan\_hesapla("kare", kenar=4))

print(alan\_hesapla("ucgen", taban=6, yukseklik=3))

**Sonuç**

DRY prensibi, yazılım geliştirmede temiz, bakımı kolay ve hataya açık olmayan kod yazmanın önemli bir adımıdır. Ancak, her durumda aşırıya kaçmadan, kodun okunabilirliğini ve performansını da göz önünde bulundurarak uygulanmalıdır