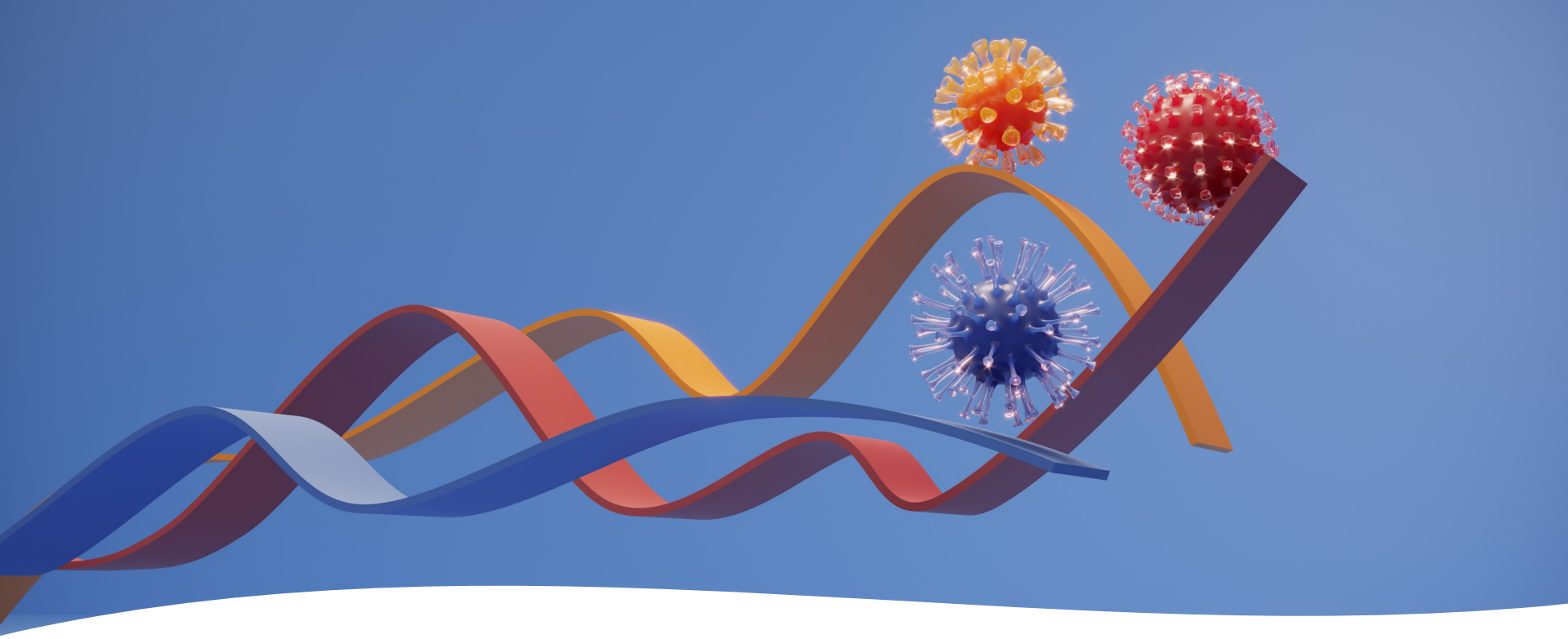
**2024-2025 EĞİTİM VE ÖĞRETİM YILI**

**PROGRAMLAMAYA GİRİŞ VE ALGORİTMA DERSİ**

**4. ÜNİTE**

# ALGORİTMA VE AKIŞ DİYAGRAMINI TEST ETME



# Algoritma ve Akış Diyagramlarının Mantıksal İncelemesi

Tasarlanan algoritma ve akış diyagramlarının, kodlama işlemine geçmeden önce dikkatlice gözden geçirilmesi gerekir.

Kaynak: onuraltuntas.net

# Algoritma ve Akış Diyagramlarının Mantıksal İncelemesi

Bu süreçte, algoritmanın tüm ihtimalleri dikkate alarak her durumda doğru sonuçları üretip üretmediği test edilmelidir.

Kaynak: onuraltuntas.net

# Algoritma ve Akış Diyagramlarının Mantıksal İncelemesi

Örneğin, bir yolculuk rotasını planlarken tüm alternatif yolları düşünmek gibi, algoritmada da her olasılığı göz önünde bulundurmak önemlidir.

Kaynak: onuraltuntas.net

Kaynak: onuraltuntas.net

# Algoritmaları Test Etmek İçin Kullanılan Araçlar:

**Python Tutor:**

**Flowgorithm:**

**PseudoEditor:**

**Visual Logic:**

# ÖRNEK ALGORİTMA

**Sayı Pozitif mi, Negatif mi?**

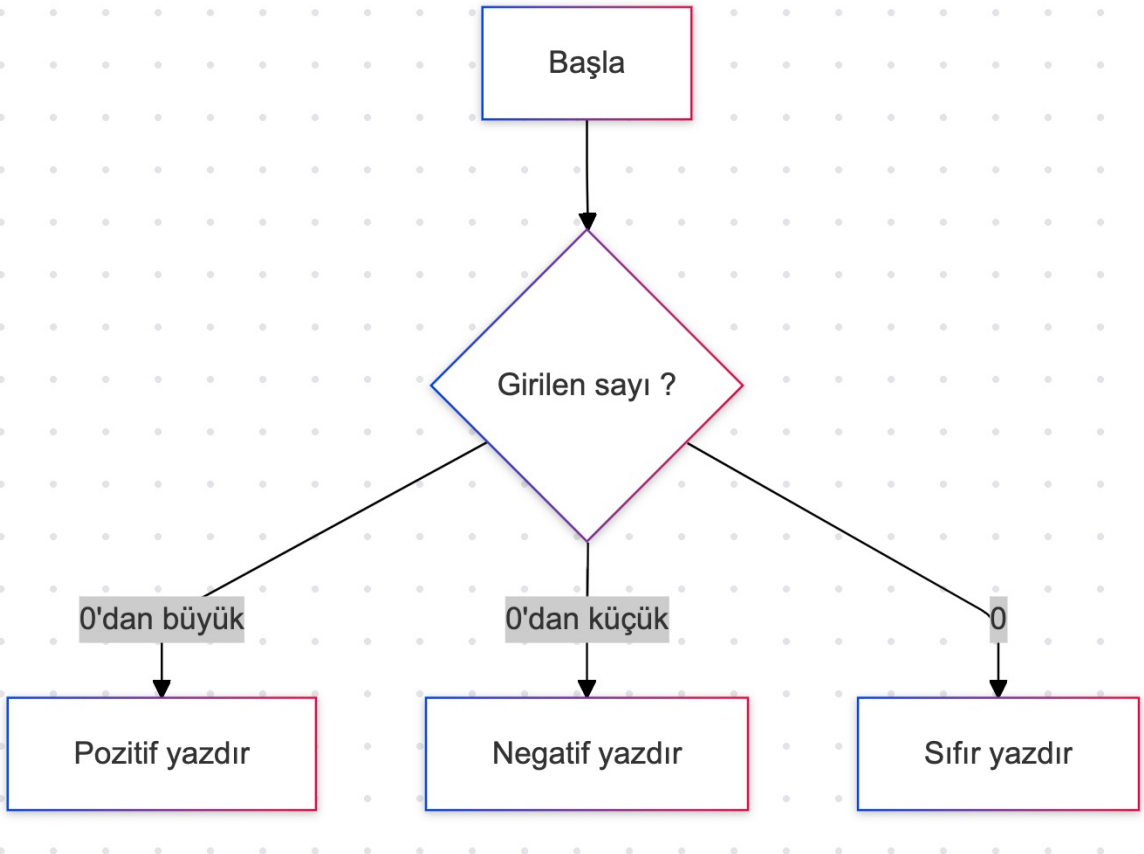
**Algoritma:** Bir sayının pozitif, negatif veya sıfır olup olmadığını belirleyen bir algoritma yazın.

•

**Adımlar:**

1. adım: Kullanıcıdan bir sayı girilmesi istenir.
2. adım: Girilen sayı 0'dan büyükse, "Pozitif" yazdır.
3. adım: Girilen sayı 0'dan küçükse, "Negatif" yazdır.
4.adım: Sayı 0 ise, "Sıfır" yazdır.

# ÖRNEK AKIŞ DİYAGRAMI




Kaynak: onuraltuntas.net

# Python tutor uygulamaları

https://pythontutor.com/render.html#mode=edit

Adresine giderek algoritmalarınızı python koduyla yazıp test ediniz.

```
# Kullanıcıdan sayı girişi alınır
sayi = int(input("Bir sayı girin: "))

# Sayıyı kontrol etme
if sayi > 0:
    print("Pozitif")
elif sayi < 0:
    print("Negatif")
else:
    print("Sıfır")
```

# Hata Ayıklama (Debugging)

- Hata ayıklama, yazılım geliştirme sürecinde kodda oluşabilecek mantık hatalarını veya beklenmeyen davranışları bulup düzeltme sürecidir.

Kaynak: onuraltuntas.net

# Hata Ayıklama (Debugging)

## HATA AYIKLAMA YÖNTEMLERİ

**1- Yazıcı Komutları Kullanma:** Kodda belirli noktalarda print veya System.out.println gibi komutlar kullanarak, değişkenlerin değerlerini veya programın akışını gözlemlemek.

Kaynak: onuraltuntas.net

# HATA AYIKLAMA YÖNTEMLERİ

## Hata Ayıklama (Debugging)

**2- Debugging Araçları Kullanma:** IDE'ler genellikle bir "debugger" içerir. Bu araçlar, kodu adım adım çalıştırmaya, kırılma noktaları (breakpoints) oluşturmaya ve değişkenlerin anlık durumunu izlemeye olanak tanır.

Kaynak: onuraltuntas.net

# Hata Ayıklama (Debugging)

## HATA AYIKLAMA YÖNTEMLERİ

3- **Kod İnceleme:** Programcılar, yazdıkları kodu veya ekip arkadaşlarının kodunu gözden geçirerek hataları tespit edebilirler.

Kaynak: onuraltuntas.net

# HATA YAKALAMA YÖNTEMLERİ

## Hata Ayıklama (Debugging)

Hata yakalama, programın çalışma sırasında beklenmedik durumları ele alarak, programın çökmesini önlemek amacıyla kullanılan bir tekniktir.

Kaynak: onuraltuntas.net

# Hata Ayıklama (Debugging)

## HATA YAKALAMA YÖNTEMLERİ

Programlama dillerinde hata yakalamak için yaygın olarak try, catch, finally blokları kullanılır.

- **try Blokları:** İçinde hata çıkabilecek kod parçaları yer alır.
- **catch Blokları:** Hata oluştuğunda devreye giren ve hatayı ele alan kısımdır.
- **finally Blokları:** Hata olsun veya olmasın, her zaman çalıştırılacak kodlar burada yer alır.

Kaynak: onuraltuntas.net

# Hata Ayıklama (Debugging)

## Hatanın Türünü ve Yerini Belirleme

Hatalar genellikle üç ana türe ayrılabilir:

**1- Sözdizimi Hataları (Syntax Errors):**

1. Dilin kurallarına aykırı yazılan kodlardır. Örneğin, bir parantezin kapanmaması veya noktalı virgül eksikliği gibi.

2. Bu hatalar genellikle algoritmada ortaya çıkar, çünkü algoritmanın düzgün bir dille ifade edilmemesi sonucu oluşur.

Kaynak: onuraltuntas.net

# Hata Ayıklama (Debugging)

## Hatanın Türünü ve Yerini Belirleme

**2- Mantık Hataları (Logical Errors):**

•Program çalışır, ancak beklenen sonuca ulaşmaz. Örneğin, yanlış matematiksel işlemler veya koşul ifadeleri.

•Bu tür hatalar hem algoritmada hem de akış diyagramında olabilir. Algoritma mantıksal olarak yanlış tanımlandıysa, akış diyagramı da yanlış olur ve sonuçlar tutarsız çıkar.

# Hata Ayıklama (Debugging)

## Hatanın Türünü ve Yerini Belirleme

**3- Çalışma Zamanı Hataları (Runtime Errors):**

•Program çalışırken ortaya çıkan hatalardır. Örneğin, sıfıra bölme veya bir dosyanın bulunamaması gibi.

•Bu tür hatalar genellikle algoritmada değil, kodun spesifik koşullarında veya kullanıcı girişlerinde ortaya çıkar.

Kaynak: onuraltuntas.net

# Hata Ayıklama (Debugging)

**Algoritmadaki Hatalar:** Algoritma, adım adım bir çözüm yolunu tanımlar. Eğer algoritma yanlış adımlar içeriyorsa, bu hatalar doğrudan sonucu etkiler.

**Hata Örneği:** Eğer bir algoritmada bir döngü yanlış bir koşula bağlıysa, beklenenden fazla veya az dönebilir ve yanlış sonuç üretir.

**Örnek Algoritma Hatası:** Bir toplama işlemi yerine çarpma işlemi yapılıyorsa.

# Hata Ayıklama (Debugging)

**Akış Diyagramındaki Hatalar:** Akış diyagramı, algoritmanın görsel bir temsilidir. Eğer algoritma hatalıysa, bu diyagramda da kendini gösterir. Ancak bazen görsel gösterim sırasında da hatalar olabilir

**Hata Örneği:** Koşullu dallanmanın (decision box) yanlış yere konumlandırılması.

**Örnek Diyagram Hatası:** Bir karar kutusunda "Eğer x > 5" yerine yanlışlıkla "Eğer x = 5" yazılması, sonucu etkileyebilir.

# Hata Ayıklama (Debugging)

Hataları bulmak için aşağıdaki yöntemler kullanılabilir:

- **Adım Adım İzleme (Step-by-Step Debugging):** Kodun her satırı tek tek çalıştırılarak değişkenlerin durumları gözlemlenir.

- **Akış Diyagramının Kontrolü:** Diyagramın her adımı mantıklı bir şekilde incelenir ve olası tüm çıkışlar test edilir.

- **Test Durumları Oluşturma:** Farklı veri setleri ile algoritma ve diyagram test edilerek her durumda doğru sonuç verip vermediği kontrol edilir.

# Hata Ayıklama (Debugging)

**Hatalı Algoritmanın Düzeltilmesi ve Çözüm Yöntemleri**

**1.Hataların Tespiti:** Hatalı adımlar, beklenmeyen sonuçlar veya yanlış mantık tespit edilir ve liste haline getirilir.

**2.Hataların Analizi:** Hatanın neden kaynaklandığı belirlenir. Bu bir mantık hatası mı, sözdizimi hatası mı, yoksa yanlış karar mı?

**3.Çözüm Yöntemlerinin Belirlenmesi:** Her hata için uygun düzeltme yöntemi belirlenir. Mantık hataları için algoritmanın mantığı gözden geçirilir, sözdizimi hataları için dil kurallarına uygun düzenlemeler yapılır.

**4.Hataların Düzeltilmesi:** Belirlenen çözümler uygulanır ve algoritma test edilerek doğru sonuç verip vermediği kontrol edilir.

Kaynak: onuraltuntas.net

# Örnek: Python kodu

**Hatalı Algoritma:** Bir dizideki sayıların ortalamasını bulmaya çalışan bir algoritma düşünün. Ancak, toplama işlemi yerine yanlışlıkla çarpma yapılmış.

```
dizi = [10, 20, 30, 40, 50]
toplam = 1  # Hatalı: toplam başlangıç değeri 0 olmalıydı
for sayi in dizi:
    toplam = toplam * sayi  # Hatalı: toplama işlemi yerine çarpma yapılıyor
ortalama = toplam / len(dizi)
print("Ortalama:", ortalama)
```

# Hataların Listelenmesi



- toplam değişkeni başlangıçta 0 yerine 1 olarak atanmış.
- toplam = toplam * sayi satırında, toplama yerine çarpma işlemi yapılmış.

- **Çözüm Yöntemleri:**

- toplam başlangıç değeri 0 olarak değiştirilmeli.
- toplam = toplam * sayi ifadesi, toplam = toplam + sayi olarak düzeltilmeli.

# Düzeltilmiş Algoritma

- dizi = [10, 20, 30, 40, 50]
- toplam = 0  # Doğru: toplam başlangıç değeri 0
- for sayi in dizi:
- toplam = toplam + sayi  # Doğru: toplama işlemi yapılıyor
- ortalama = toplam / len(dizi)
- print("Ortalama:", ortalama)

# Düzeltilmiş Algoritma

- dizi = [10, 20, 30, 40, 50]
- toplam = 0  # Doğru: toplam başlangıç değeri 0
- for sayi in dizi:
- toplam = toplam + sayi  # Doğru: toplama işlemi yapılıyor
- ortalama = toplam / len(dizi)
- print("Ortalama:", ortalama)