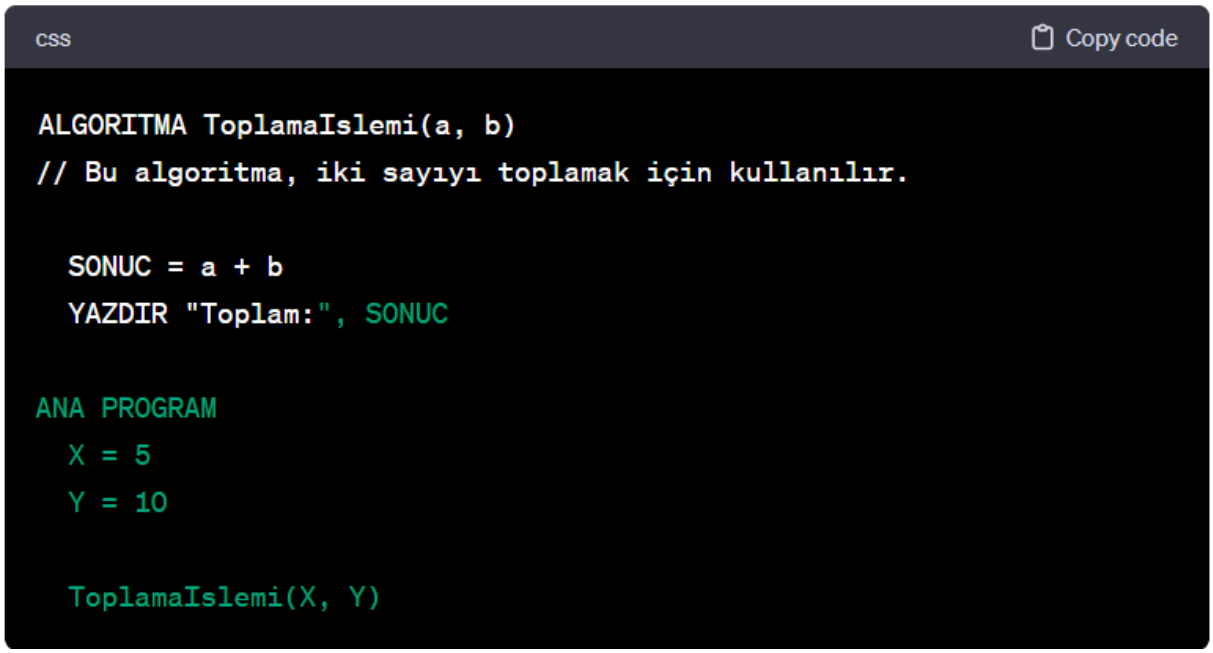
Pseudo Code

Pseudo code, genellikle bir algoritmanın veya programın mantığı veya adımları hakkında açıklamalar içeren, programlama diline özgü olmayan, insanlar tarafından anlaşılabilir bir dilde yazılmış bir taslağı ifade eder. Pseudo code, bir programın temel mantığını anlatmak için kullanılır ve genellikle daha sonra gerçek bir programlama diline çevrilecek kodun ana hatlarını sunar.

Pseudo code, program geliştirme sürecinin başlarında kullanılır ve daha sonra bu taslağı gerçek bir programlama dilinde kodlamak daha kolay olabilir. Ayrıca, farklı programcılar arasında iletişimi kolaylaştırabilir, çünkü programın mantığına odaklanırken spesifik bir dilin ayrıntılarından kaçınır.

Pseudo code'un bir örneği şu şekilde olabilir:



Bu örnekte, `ToplamaIslemi` adlı bir algoritma, iki sayıyı toplamak için kullanılır ve `ANA PROGRAM` bölümünde bu algoritma çağrılır. Bu pseudo code, bir programın temel mantığını anlatan basit bir örnektir.

Biraz daha detaylı bir kullanım örneği:

plaintext Copy code

```
ALGORITMA FaktoriyelHesapla(n)
// Bu algoritma, bir sayının faktöriyelini hesaplar.

  EĞER n < 0 İSE
    HATA VER "Negatif sayıların faktöriyeli tanımsızdır."
  DİĞERİ DURUMDA
    FAKTÖRİYEL = 1
    İ i = 1'den n'ye kadar ARTAR
      FAKTÖRİYEL = FAKTÖRİYEL * i
    İ i = i + 1
    YAZDIR n, "'nin faktöriyeli:", FAKTÖRİYEL

ANA PROGRAM
  // Kullanıcıdan bir sayı al
  YAZDIR "Faktöriyelini hesaplamak istediğiniz sayıyı girin:"
  OKU SAYI

  // Faktöriyel hesaplama işlemini çağır
  FaktoriyelHesapla(SAYI)
```

Bu örnekte, **FaktoriyelHesapla** adlı bir algoritma tanımlanmıştır. Bu algoritma, kullanıcıdan alınan bir sayının faktöriyelini hesaplar. Pseudo code, sayının negatif olup olmadığını kontrol eder ve negatifse bir hata mesajı verir. Negatif olmadığı durumda, faktöriyel hesaplaması bir döngü içinde gerçekleştirilir.

'**ANA PROGRAM**' bölümünde, kullanıcıdan bir sayı alınır ve daha önce tanımlanan algoritma bu sayının faktöriyelini hesaplamak için çağrılır. Bu örnek, pseudo code'un gerçek bir programı anlama ve tasarlama sürecine yardımcı olma amacını göstermektedir.

Hata Yönetimi

Python'da hata yönetimi için **try**, **except**, **else** ve **finally** bloklarını kullanabilirsiniz. Bu yapılar, programınızın çalışma sırasında oluşabilecek hatalara karşı daha güvenli ve kontrol edilebilir bir şekilde yaklaşmanıza yardımcı olur. İşte bir örnek:

python Copy code

```python
def bolme_islemi(a, b):
    try:
        # Bölme işlemi gerçekleştir
        sonuc = a / b
    except ZeroDivisionError:
        # Sıfıra bölme hatası
        print("Hata: Sıfıra bölme hatası!")
    except TypeError:
        # Tür hatası
        print("Hata: Geçersiz veri türü!")
    else:
        # Hata oluşmadıysa bu blok çalışır
        print("Sonuç:", sonuc)
    finally:
        # Her durumda çalışan blok
        print("İşlem tamamlandı veya hata oluştu.")

# Test senaryoları
bolme_islemi(10, 2)  # Normal durum
bolme_islemi(10, 0)  # Sıfıra bölme hatası
bolme_islemi("10", 2)  # Tür hatası
```

Bu örnekte, **bolme_islemi** adlı bir fonksiyon tanımlanmıştır. Bu fonksiyon, iki sayı arasında bölme işlemi gerçekleştirir. **try** bloğu içinde işlem gerçekleştirilir, ancak bir hata oluşursa ilgili **except** bloğu çalışır. **else** bloğu, hata oluşmadığında çalışır ve **finally** bloğu ise her durumda çalışır, yani hata olsun veya olmasın.

Bu örnekte, sıfıra bölme hatası ve tür hatası durumlarına özel **except** blokları eklenmiştir. Bu sayede program, bu tür hatalarla karşılaştığında düzgün bir şekilde işlem yapabilir ve kullanıcıya uygun bir hata mesajı verebilir.