## Programlamanın Temelleri

Yorumlanan Programlama Dilleri (Interpreted Programming Languages)

- Yorumlanan diller, Python gibi, bilgisayarın işletim sistemi veya web tarayıcısı üzerinden çalışan programlardır. Bu diller, insan tarafından okunabilir kodu makine koduna çeviren bir yorumlayıcıya ihtiyaç duyar.
- Yaygın örnekler arasında JavaScript, Python ve Lua bulunmaktadır. Bu diller genellikle daha esnek ve öğrenmesi kolaydır.

Derlenen Programlama Dilleri (Compiled Programming Languages)

- Derlenen diller, müzik uygulamaları veya işletim sistemleri gibi daha karmaşık programlar oluşturmak için kullanılır. Bu diller, kaynak kodunu makine koduna çeviren bir derleyici aracılığıyla tek bir çalıştırılabilir dosya haline getirilir.
- C, C++ ve Java gibi derlenen diller, genellikle daha büyük ve karmaşık uygulamalar için tercih edilir. Örneğin, Android işletim sistemi Java ile yazılmıştır.

**Yorumlanan Diller (Interpreted Languages)**

- **Çalışma Şekli**: Kod, bir yorumlayıcı tarafından satır satır çalıştırılır. Her seferinde kodun tamamı değil, sadece o anki satır çalıştırılır.
- **Örnekler**: Python, JavaScript, Ruby.
- **Avantajlar**:
    - Daha esnek ve hızlı bir geliştirme süreci sağlar.
    - Hatalar anında tespit edilir, bu da hata ayıklamayı kolaylaştırır.
- **Dezavantajlar**:
    - Performans genellikle derlenen dillere göre daha düşüktür.
    - Her çalıştırmada yorumlayıcıya ihtiyaç duyar.

**Derlenen Diller (Compiled Languages)**

- **Çalışma Şekli**: Kod, bir derleyici tarafından makine diline çevrilir ve tek bir çalıştırılabilir dosya oluşturulur. Bu dosya daha sonra çalıştırılır.
- **Örnekler**: C, C++, Java.
- **Avantajlar**:
    - Daha yüksek performans sunar, çünkü kod önceden derlenmiştir.
    - Çalıştırma sırasında yorumlayıcıya ihtiyaç duyulmaz.
- **Dezavantajlar**:

    - Geliştirme süreci daha uzun olabilir, çünkü kodun her değişikliğinden sonra yeniden derlenmesi gerekir.
    - Hatalar, derleme aşamasında tespit edilir, bu da hata ayıklamayı zorlaştırabilir.

Yüksek Seviyeli ve Düşük Seviyeli Programlama Dilleri

- Yüksek seviyeli programlama dilleri, İngilizce benzeri bir dil kullanarak kodun anlaşılabilirliğini artırır ve kodlama sürecini hızlandırır. Örnekler arasında SQL, Pascal ve Python yer alır.
- Düşük seviyeli programlama dilleri, makine kodunu temsil eden semboller kullanır. Örnekler arasında ARM, MIPS ve X86 montaj dilleri bulunur.

Sorgu Dilleri Query

- Sorgu, bir veritabanından bilgi talep etme işlemidir. Veritabanı, sorguyu işleyerek istenen bilgiyi döndürür.
- En yaygın sorgu dili SQL'dir, ancak AQL, CQL, Datalog ve DMX gibi diğer sorgu dilleri de mevcuttur. Sorgu dilleri, veritabanından veri talep etmek veya veriyi oluşturmak, okumak, güncellemek ve silmek için kullanılır.

Montaj Dilleri Assmbly

- Montaj dilleri, makine kodunun 0 ve 1'lerini temsil eden basit semboller kullanır ve düşük seviyeli programlama dilleri arasında yer alır.
- Her CPU tipi genellikle kendi montaj diline sahiptir ve montaj dilleri, bir derleyici veya yorumlayıcı yerine bir montajcı tarafından çevrilir. Her bir montaj dili ifadesi, bir makine kodu talimatına karşılık gelir.

SQL (Structured Query Language):

- SQL, ilişkisel veritabanı yönetim sistemleri için kullanılan bir sorgu dilidir. Veriler, önceden tanımlanmış şemalarla yapılandırılmış tablolarda saklanır. Bu, verilerin satırlar ve sütunlar halinde düzenlendiği anlamına gelir.
- SQL veritabanları, veri tutarlılığı ve bütünlüğü sağlamak için güçlü bir yapı sunar. Örneğin, bir müşteri kaydı oluşturduğunuzda, bu kaydın tüm gerekli bilgileri içermesi gerekir.
- SQL veritabanları, genellikle karmaşık sorgular ve veri analizi için idealdir. Örnekler arasında MySQL, PostgreSQL ve Oracle yer alır.

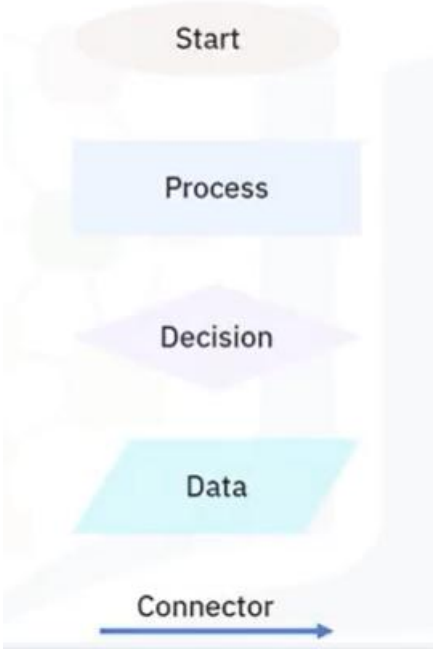
NoSQL (Not Only SQL):

- NoSQL, ilişkisel olmayan veritabanı yönetim sistemlerini ifade eder. Veriler, dinamik şemalarla yapılandırılmıştır ve genellikle belge, anahtar-değer, grafik veya sütun tabanlı formatlarda saklanır.
- NoSQL veritabanları, büyük veri setleri ve yüksek hacimli verilerle çalışmak için daha esnek bir yapı sunar. Bu, verilerin daha hızlı bir şekilde eklenip güncellenmesine olanak tanır.

- NoSQL veritabanları, genellikle ölçeklenebilirlik ve performans açısından avantaj sağlar. Örnekler arasında MongoDB, Cassandra ve Redis bulunur.

Özetle:

- SQL, yapılandırılmış verilerle çalışırken, NoSQL daha esnek ve dinamik veri yapılarıyla çalışır.
- SQL veritabanları, veri tutarlılığına odaklanırken, NoSQL veritabanları daha fazla ölçeklenebilirlik ve esneklik sunar.




Kod Organizasyon Yöntemleri

- Kod organizasyonu, kodun okunabilirliğini, bakımını ve yapılandırılmasını artırır.
- İyi planlanmış yazılım tasarımı, daha temiz ve güvenilir kod yazılmasına yardımcı olur.

Akış Şemaları ve Pseudocode

- Akış şemaları, algoritmaların adımlarını kutular ve oklarla gösteren görsel temsillerdir.
- Pseudocode, bir algoritmanın yüksek seviyeli tanımını sunar ve programlama dillerine bağımlı değildir.

Avantajlar ve Kullanım Alanları

- Akış şemaları, küçük kavramlar için faydalıdırken, pseudocode daha büyük projelerde etkilidir.
- Pseudocode, programcıların fikirlerini paylaşmalarını kolaylaştırır ve kod geliştirme aşamasını basitleştirir.

Dallanma Mantığı

- Dallanma, programın belirli koşulların karşılanıp karşılanmadığına göre farklı talimatlar izlediği bir mantık türüdür.
- Yaygın dallanma ifadeleri arasında "if", "if-then-else", "switch" ve "goto" bulunmaktadır.

Döngü Mantığı

- Döngü, belirli bir koşul sağlanana kadar sürekli olarak tekrar eden bir talimat dizisidir.
- Temel döngü ifadeleri "while", "for" ve "do-while" olarak üç gruba ayrılır; her biri farklı koşul değerlendirme yöntemleri kullanır.

Boolean İfadeleri ve Değişkenler

- Boolean ifadeleri yalnızca "doğru" veya "yanlış" değerlerini alabilen programlama ifadeleridir.
- Değişkenler, programın çalışması sırasında değişebilen ve fonksiyonlara veya alt programlara geçirilen değerlerdir.

**GoTo İfadesi: Basit Bir Açıklama**

GoTo ifadesi, programlama dillerinde belirli bir kod satırına doğrudan geçiş yapmayı sağlayan bir komuttur. Bu ifade, program akışını kontrol etmek için kullanılır ve genellikle belirli bir koşulun sağlanması durumunda başka bir kod bloğuna atlamak için tercih edilir. Ancak, GoTo ifadesinin kullanımı, kodun okunabilirliğini ve bakımını zorlaştırabileceği için dikkatli bir şekilde kullanılmalıdır.

**Bir Örnekle Açıklama:**

Diyelim ki bir programda kullanıcıdan bir sayı girmesini istiyoruz ve bu sayının pozitif olup olmadığını kontrol etmek istiyoruz. Eğer sayı negatifse, kullanıcıyı bilgilendirmek için GoTo ifadesini kullanabiliriz:

```
number = int(input("Bir sayı girin: "))

if number < 0:

   goto negative_label

print("Girdiğiniz sayı pozitif.")

negative_label:

print("Lütfen pozitif bir sayı girin.")
```

Bu örnekte, kullanıcı negatif bir sayı girerse, program negative_label etiketine atlar ve kullanıcıya pozitif bir sayı girmesi gerektiğini bildirir. Ancak, bu tür bir kullanım, kodun akışını karmaşık hale getirebilir ve bu nedenle genellikle daha yapılandırılmış kontrol yapıları (if-else gibi) tercih edilir.

**Unutmayın:**

- GoTo ifadesi, program akışını kontrol etmek için kullanılabilir, ancak aşırı kullanımı kodun karmaşıklaşmasına neden olabilir.
- Daha iyi bir kod yapısı için, dallanma ve döngü yapıları gibi daha modern kontrol yapıları tercih edilmelidir.

Tanımlayıcılar (Identifiers)

- Yazılım geliştiricileri, program bileşenlerini referans almak için tanımlayıcılar (identifiers) kullanır; bu, bir değer (value), yöntem (method), arayüz (interface) veya sınıf (class) gibi bileşenlere özel adlar atamayı içerir.
- Tanımlayıcılar iki tür veri saklayabilir: sabitler (constants) (değeri değişmeyen) ve değişkenler (variables) (değeri programın çalışması sırasında değişebilen).

Kapsayıcılar (Containers)

- Kapsayıcılar (containers), birden fazla öğeyi referans almak için kullanılır ve her bir öğe için ayrı bir değişken oluşturma ihtiyacını ortadan kaldırır.
- İki ana kapsayıcı türü vardır: diziler (arrays) (sabit boyutlu ve sıralı öğeler) ve vektörler (vectors) (dinamik boyutlu ve otomatik olarak boyutlandırılabilen).

Diziler ve Vektörler (Arrays and Vectors)

- Diziler (arrays), belirli bir veri türünde sabit sayıda öğe saklar ve sıralı bir şekilde başlar.
- Vektörler (vectors), dinamik boyutları sayesinde öğe eklenip çıkarıldıkça otomatik olarak boyutlandırılır, bu da daha fazla bellek kullanımı ve erişim süresinin uzamasına neden olabilir.

Fonksiyonlar (Functions)

- Fonksiyonlar, belirli bir görevi yerine getiren, yapılandırılmış, bağımsız ve yeniden kullanılabilir kod parçalarıdır (structured, stand-alone, reusable code).
- Fonksiyonlar, verileri girdi (input) olarak alır, işler ve sonuç olarak çıktı (output) döner. İki tür fonksiyon vardır: standart kütüphane fonksiyonları (standard library functions) ve kullanıcı tarafından tanımlanan fonksiyonlar (user-defined functions).

Nesneler (Objects)

- Nesneler, nesne yönelimli programlamanın (Object-Oriented Programming, OOP) temelini oluşturur ve verileri özellikler (attributes) ve kodu prosedürler (methods) olarak içerir.
- Yazılım nesneleri, gerçek dünya nesneleri gibi durumlar (states) ve davranışlar (behaviors) içerir. Örneğin, bir kullanıcı hesabı (user account) veya bir veritabanı tablosu (database table) bir nesne olarak düşünülebilir.

Öğrenilenler

- Fonksiyonlar, programlama dillerinde yaygın olarak tanımlanır (defined) ve çağrılır (called).
- OOP, nesneler üzerinde yoğunlaşan bir programlama metodolojisidir ve yazılım nesneleri, özellikler ve metotlar içerir.

Nesne yönelimli programlamanın (OOP) birkaç avantajı, yazılım geliştirme ve bakımını artırır:

1. **Kapsülleme (Encapsulation)**:
    - Verileri (özellikler) ve bu veriler üzerinde işlem yapan yöntemleri (fonksiyonlar) bir araya getirerek bir nesne oluşturur.
    - Nesnenin iç durumunu dış müdahalelerden ve kötüye kullanımdan korur.
2. **Kalıtım (Inheritance)**:
    - Yeni sınıfların (alt sınıflar) mevcut sınıflardan (üst sınıflar) özellikleri ve yöntemleri miras almasına olanak tanır.
    - Kodun yeniden kullanılabilirliğini artırır ve tekrar eden kodu azaltır, bu da kodun oluşturulmasını ve bakımını kolaylaştırır.
3. **Polimorfizm (Polymorphism)**:
    - Nesnelerin, üst sınıf örnekleri olarak işlenmesine olanak tanır, böylece yöntemler birbirinin yerine kullanılabilir.
    - Esneklik sağlar ve aynı yöntemin farklı nesne türlerine göre farklı davranışlar sergilemesine olanak tanır.
4. **Soyutlama (Abstraction)**:
    - Karmaşık sistemleri, gereksiz detayları gizleyerek temel özelliklere dayalı olarak modellemeyi sağlar.
    - Geliştiricilerin yüksek seviyeli işlevselliklere odaklanmasına yardımcı olarak karmaşıklığı yönetir.
5. **Daha İyi Bakım (Improved Maintainability)**:
    - OOP, düzenli bir kod yapısını teşvik eder, bu da kodun anlaşılmasını, değiştirilmesini ve genişletilmesini kolaylaştırır.
    - Kodun bir bölümündeki değişiklikler, diğer bölümler üzerinde minimal etki ile yapılabilir, bu da bakım kolaylığı sağlar.
6. **Daha İyi İşbirliği (Better Collaboration)**:
    - OOP, ekiplerin farklı sınıflar veya nesneler üzerinde aynı anda çalışmasına olanak tanır, bu da işbirliğini ve verimliliği artırır.

Bu avantajlar, OOP'yi ölçeklenebilir, sürdürülebilir ve sağlam yazılım uygulamaları geliştirmek için güçlü bir paradigma haline getirir.