PROGRAMLAMA DİLLERİ VE YAZILIM GELİŞTİRME

Temel Bilgi Teknolojileri Kullanımı

Ç. Ü. Enformatik Bölümü

Temel Kavramlar

Programlama Dili Nedir?

- Programlama dili, geliştiricilerin belirli görevleri ve işlemleri bilgisayara yaptırabilmesi için geliştirilmiş bir kodlama dilidir.
- Bu diller, insanın okuyabileceği ve anlamlandırabileceği sözdizimlerine sahiptir ve bu sözdizimleri, bilgisayar tarafından anlaşılabilir bir forma çevrilir.
- Programlama dillerinin temel amacı, bilgisayarın işleyeceği talimatları yazılı olarak ifade etmektir.

Kaynak Kod Nedir?

- Bir programlama dilinin kurallarına göre yazılmış, üzerinde değişiklik yapılabilen, programın algoritmasını içeren belgelerdir.
- Kaynak kodların çalışması için Makinenin anlayabileceği şekle getirilmesi gerekir.

```java
public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello world!");
    }
}
```

Makine Dili Nedir?

- İşlemcinin verilen komutlar doğrultusunda çalıştırılmasını sağlayan ve işlemci mimarisine göre değişen en alt seviyedeki programlama dilidir.
- Bu dil sadece 0 ve 1 ikililerinin anlamlı kombinasyonlarından meydana gelmektedir. Bunlar, işlemci tarafından çözülerek gerekli işlemin yerine getirilmesini sağlar.
- Diğer programlama dillerin gerektirdiği derleyici ya da yorumlayıcı kullanımını gerektirmediğinden donanımı doğrudan kontrol eder.

Derleme İşlemi ve Derleyici Nedir?

- Kaynak kodun makine diline çevrilmesi işlemine Derleme (Compile) adı verilir.
- Bu derleme işlemini yapan programlara Derleyici (Compiler) adı verilir.
- Kaynak kod üzerinde yazım hatası ve dilin kuralları dışında bir işlem yapılmışsa, derleyici derleme işlemi sırasında hata verir ve derleme işlemi gerçekleşmez.

Programlama Dillerinin Tarihçesi

- 1940'lar ve 1950'ler:

  İlk bilgisayarlar için makine dili ve assembler kullanılıyordu. Fortran (1957) ve COBOL (1959) gibi ilk yüksek seviyeli diller bu dönemde ortaya çıktı.

- 1960'lar - 1970'ler:

  C dili ve Pascal gibi diller ortaya çıktı. Bu diller, bilgisayar mühendisliğinde büyük bir ilerleme sağladı.

- 1980'ler - 1990'lar:

  Nesne yönelimli programlama kavramı tanıtıldı. Bu dönemde C++, Java, C# gibi nesne yönelimli diller popülerlik kazandı.

- 2000'ler - Günümüz:

  Python, JavaScript, Swift ve Go gibi modern dillerle birlikte, kullanım alanına göre optimize edilmiş diller geliştirildi. Bu dönemde programlama dillerinin yazılım geliştirme hızını ve kolaylığını artıran özellikler eklendi.

Programlama Dillerinin Sınıflandırılması

1. Programlama Dilinin Seviyesi

- Programlama dilinin seviyesi, o programlama dilinin insan algılamasına olan yakınlığıdır.

1. ***Düşük Seviyeli Diller:*** Bu diller, donanıma daha yakın ve makineye doğrudan erişim sağlayan dillerdir. Makine dili ve assembler bu gruptadır. Bellek yönetimi gibi düşük seviyeli işlemler üzerinde tam kontrol sağlar, ancak okunması ve yazılması zor olabilir.
2. ***Yüksek Seviyeli Diller:*** İnsan tarafından daha anlaşılabilir olan dillerdir. Bu diller, sistemle doğrudan etkileşime girmeden daha karmaşık işlemleri yönetmeyi sağlar. C, Python, Java, C#, vb. yüksek seviyeli dillere örnektir.

Seviyelerine Göre Programlama Dilleri

- ***Düşük Seviyeli Diller:*** Makine dilleri, Assembly
- ***Orta Seviyeli Diller:*** Hem makinaya hem de insan algılamasına yakın dillerdir. C, C++
- ***Yüksek Seviyeli Diller:*** Daha hızlı yazılım üretmeye yatkın dillerdir. Donanıma etkileri de orta seviyedeki kadar performanslı olmasa da vardır. C#, Java, Pascal
- ***Çok Yüksek Seviyeli Diller:*** Çok daha az kod yazarak, kullanıcı arayüzü kullanarak program yazılabilen dillerdir. Visual Basic, Access, Fox Pro

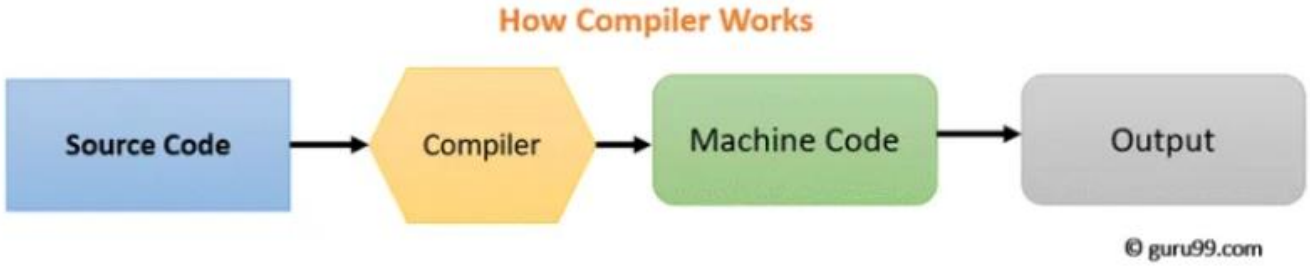
2. Programların Çalışma Türleri

Programlama dilleri, çalışma prensiplerine göre 3 şekilde incelenir:

- ***Derlenen Diller:*** Bu dillerde kaynak kod makine diline çevrilir. Bilgisayar programı makine diliyle kendisi çalıştırır.
- ***Yorumlanan Diller:*** Bu dillerde kaynak kodu çalıştıracak olan bir çalıştırıcı program bulunur. Bu program kodu satır satır yorumlayarak kendisi çalıştırır.
- ***Yarı Yorumlanmış Diller (Hibrit Diller):*** Bu diller, hem derleme hem de yorumlama aşamalarını kullanır. İlk aşamada kod, bayt koduna veya ara bir formata derlenir, sonra çalışma zamanında yorumlanır.

Çalışma Türlerine Göre Programlama Dilleri

- ***Derlenmiş Diller:*** C, C++, Rust, Go
- ***Yorumlanmış Diller:*** Python, JavaScript, Ruby, PHP
- ***Hibrit Diller:*** Java, C#, Kotlin



Şekil 1. Derlenen ve Yorumlanan Dillerin Çalışma Mantığı

3. Programlama Dillerinin Kullanım Alanları

- ***Web Geliştirme:*** Web uygulamaları ve internet tabanlı servisler geliştirmek için kullanılır. Bu diller, hem istemci tarafında (frontend) hem de sunucu tarafında (backend) kullanılabilir
- ***Mobil Geliştirme:*** Mobil cihazlar için uygulama geliştirme alanında kullanılır. Genellikle platforma özel diller veya çapraz platform çözümleri ile geliştirilir.
- ***Oyun Geliştirme:*** Oyun motorlarıyla uyumlu veya oyun geliştirme için performans odaklı diller tercih edilir. Bu diller, genellikle yüksek performans gereksinimi olan grafik ve fizik hesaplamalarında etkilidir.
- ***Veri Bilimi ve Yapay Zeka:*** Veri analizi, makine öğrenimi ve yapay zeka geliştirme alanlarında tercih edilir. Bu diller, büyük veri kümeleri üzerinde analiz yapabilen ve model geliştirmede güçlü kütüphane desteğine sahiptir.

3. Programlama Dillerinin Kullanım Alanları

- ***Sistem Programlama:*** Sistem yazılımı, işletim sistemleri ve donanıma yakın diğer uygulamaları geliştirmek için kullanılır. Genellikle düşük seviyeli olup, bellek yönetimi ve performans kontrolü sağlar.
- ***Bilimsel ve Mühendislik:*** Mühendislik ve bilimsel hesaplamalarda kullanılan diller, yüksek doğruluk ve güçlü matematiksel kütüphaneler sunar.
- ***Veritabanı Yönetimi:*** Veritabanı yönetimi, veri sorgulama ve veri işleme için özel olarak geliştirilmiş diller kullanılır.
- ***Betik (Scripting) Diller:*** Betik dilleri, özellikle otomasyon ve küçük ölçekli görevleri yerine getirmek için kullanılır. Çoğu betik dili yorumlanır ve hızlı prototipleme sağlar.

Kullanım Alanlarına Göre Programlama Dilleri

- ***Web Geliştirme:***
  - İstemci Tarafı (Frontend) : HTML, CSS, JavaScript, TypeScript
  - Sunucu Tarafı (Backend) : Python, Ruby, Node.js, ASP.NET
- ***Mobil Geliştirme:***
  - iOS : Swift, Objective-C
  - Android: Kotlin, Java
  - Çapraz Platform : Flutter (Dart), React Native (JavaScript), MAUI (C#)
- ***Oyun Geliştirme:*** C++ (unreal), C# (unity)

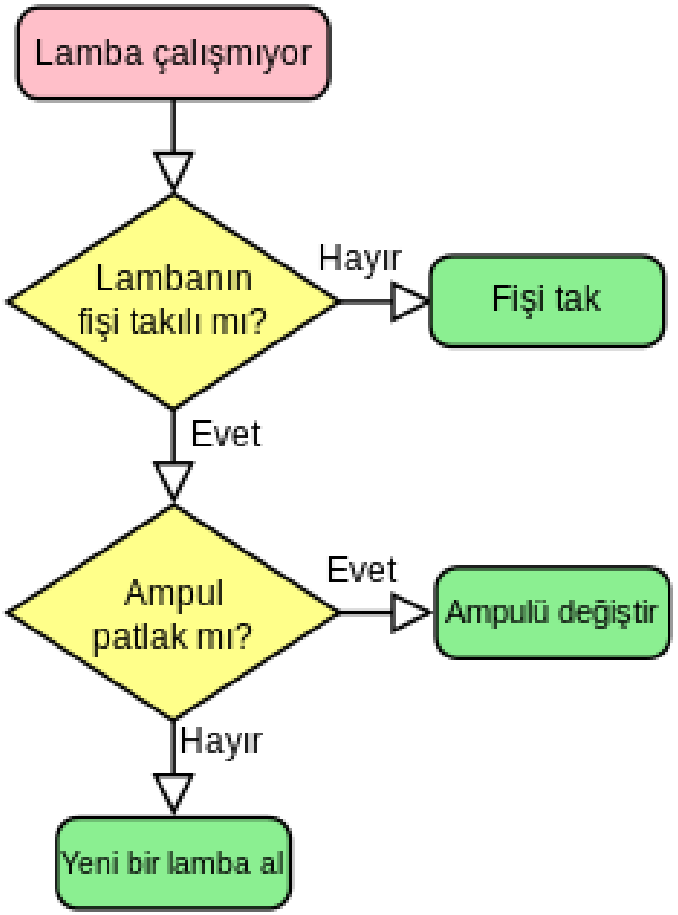
Kullanım Alanlarına Göre Programlama Dilleri

- ***Veri Bilimi ve Yapay Zeka:*** Python, Java, R, Julia
- ***Sistem Programlama:*** C, C++, Rust, Assembly
- ***Bilimsel ve Mühendislik:*** MATLAB, Fortran, Julia
- ***Veritabanı Yönetimi:*** SQL (Structured Query Language), PL/SQL (Oracle), T-SQL (Microsoft SQL Server)
- ***Betik (Scripting) Dilleri:*** Python, JavaScript, Bash, Perl

Algoritmalar ve Akış Diyagramları

Algoritma

- ***Algoritma***, belli bir problemi çözmek veya belirli bir amaca ulaşmak için tasarlanan yoldur.
- Bir işi yapmak için tanımlanan, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumunda sonlanan, sonlu işlemler kümesidir.



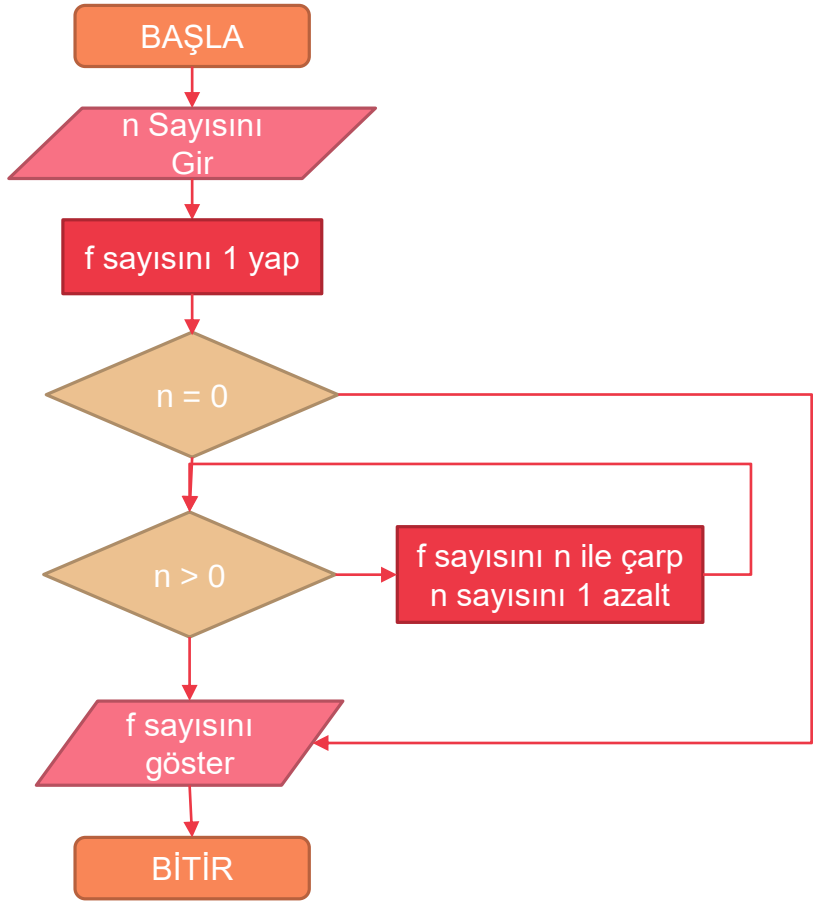
Algoritmanın Adım Adım Yazılması

- Algoritmalar, basit adımlar halinde yazılarak çözüm süreci planlanır.
  - ✓ Yan tarafta, bir sayının faktöriyelini hesaplayan algoritmanın adımları bulunmaktadır.

- Başla.
- Girdi olarak bir sayı al (n).
- Faktöriyel değişkenini 1 olarak başlat.
- Eğer n = 0 ise sonucu 1 olarak belirle.
- n > 0 iken, n sayısını faktöriyel ile çarp ve n'yi 1 azalt.
- İşlem tamamlanınca faktöriyel değişkenini döndür.
- Bitir.

Algoritmanın Akış Diyagramı ile Gösterimi

- Akış diyagramı, bir algoritmanın veya sürecin grafiksel temsilidir.
- Akış diyagramları, algoritmanın adımlarını simgeler ve bu adımların arasındaki bağlantıları oklarla gösterir.
- Bu diyagramlar, karmaşık süreçleri daha görsel hale getirerek anlaşılmasını kolaylaştırır.



Yazılım Geliştirme Süreçleri

Yazılım Geliştirme Aşamaları

- ***Gereksinim Analizi:*** Yazılımın hangi işlevleri yerine getirmesi gerektiği ve kullanıcı ihtiyaçları belirlenir. Bu aşamada kullanıcılar, paydaşlar ve geliştiricilerle yapılan görüşmelerle gereksinimler netleştirilir. Tüm gereksinimler belgelenir.
- ***Sistem Tasarımı:*** Yazılımın genel yapısı, mimarisi, veri yapıları, kullanıcı arayüzü ve sistem bileşenlerinin nasıl etkileşime gireceği planlanır. Bu aşama, sistemin tüm bileşenlerinin nasıl bir arada çalışacağına dair teknik bir rehber sunar.
- ***Kodlama (Geliştirme):*** Tasarım aşamasında belirlenen yapı ve plan doğrultusunda yazılım geliştirilir. Bu aşamada programcılar, yazılımın fonksiyonel gereksinimlerini karşılamak için kod yazarlar. Bu süreçte programlama dilleri, çerçeveler ve araçlar kullanılır.

Yazılım Geliştirme Aşamaları

- ***Test:*** Yazılımın tüm bileşenleri, işlevsellik, güvenlik, performans ve kullanıcı etkileşimi gibi çeşitli açılardan test edilir. Test aşamasında, yazılımın hatasız çalıştığından emin olmak için bir dizi test türü yapılır.
- ***Dağıtım ve Entegrasyon:*** Yazılım, kullanıcılar için dağıtılır ve entegre edilir. Bu aşama, yazılımın son kullanıcıya ulaştırılmasını ve sistemdeki diğer bileşenlerle entegrasyonunun sağlanmasını içerir.
- ***Bakım ve Destek:*** Yazılım, kullanıcılar tarafından kullanıldıkça ortaya çıkan sorunlar çözülür ve yazılım güncellemeleri yapılır. Bakım aşamasında hata düzeltmeleri, yeni özellik eklemeleri ve performans iyileştirmeleri yapılır.

Yazılım Geliştirme Yöntemleri

- ***Şelale Modeli:*** Her aşama sırasıyla yapılır. Aşamalar birbirini takip eder, bu nedenle geri dönüşler zordur. Küçük ve net bir projede iyi çalışır.
- ***Çevik (Agile) Metodolojisi:*** Esnek ve iteratif bir süreçtir. Yazılım geliştirme süreci küçük parçalara bölünerek, her bir parça üzerinde kısa döngülerle çalışılır. Bu yöntem, müşteri geri bildirimlerine dayalı olarak sürekli iyileştirme sağlar.
- ***Scrum:*** Çevik metodolojinin bir türüdür ve projeyi kısa zaman dilimlerinde (sprint) yönetir. Her sprint sonunda bir ürün parçası teslim edilir.
- ***DevOps:*** Yazılım geliştirme ve sistem operasyonları arasındaki işbirliğini artırmaya yönelik bir yaklaşımdır. Yazılımın hızlı bir şekilde geliştirilmesi ve dağıtılmasını sağlar.

Yazılım Geliştirmede Kullanılan Araçlar

- ***IDE (Integrated Development Environment):*** Visual Studio, Eclipse,Netbeans, IntelliJ IDEA gibi araçlar yazılım geliştirmeyi hızlandırır.
- ***Sürüm Kontrol Sistemleri:*** Git, SVN, Mercurial gibi araçlar, yazılımın farklı sürümlerinin yönetilmesini sağlar.
- ***Test Araçları:*** JUnit, Selenium, Postman gibi araçlar, yazılımın kalitesini test etmek için kullanılır.
- ***Yazılım Entegrasyonu ve Sürekli Dağıtım Araçları:*** Jenkins, Travis CI, Circle CI gibi araçlar, sürekli entegrasyon ve dağıtım süreçlerini yönetir.

Temel Programlama Yapıları

1. Değişkenler ve Veri Tipleri

- ***Değişkenler***, programda verilerin saklanması için kullanılan bellek alanlarıdır. Her değişken, belirli bir ***veri tipi*** ile tanımlanır. Veri tipleri, bir değişkenin alabileceği veri türünü belirtir.
- ***Veri Tipleri:***
  - ***Sayısal Veri Tipleri:*** int, float, double, decimal
  - ***Karakter ve Metin Veri Tipleri:*** char, string
  - ***Mantıksal Veri Tipi:*** bool (True/False değerlerini tutar)
  - ***Diğer Veri Tipleri:*** DateTime, object, array, enum

2. Kontrol Yapıları

- Kontrol yapıları, programın akışını belirleyen yapılardır. Bu yapılar sayesinde program, belirli koşullara göre farklı yollar izleyebilir.
- Koşul içeren bir ifade mantıksal işleme tabi tutulur ve sonuçta DOĞRU ya da YANLIŞ türünde bir sonuca ulaşılır. Program akışı bu sonuca göre devam eder.

```
if ( pass == 1024)
    Console.WriteLine("Şifre Doğru");
else
    Console.WriteLine("Hatalı Şifre! ");
```

3. Döngüler

Döngüler, belirli bir koşul sağlandığı sürece bir kod bloğunu tekrarlamak için kullanılır. Bu yapılar, işlemlerin hızlı bir şekilde tekrarlanmasını sağlar.

```
for ( int i = 1; i <= 10; i++)
    {
        Console.WriteLine("Bilgisayar");
    }
```

4. Fonksiyonlar (Metodlar)

Fonksiyonlar, belirli bir görevi yerine getiren ve genellikle bir değer döndüren kod bloklarıdır. Fonksiyonlar, programın daha modüler olmasını sağlar ve aynı kodun farklı yerlerde tekrar kullanılmasını sağlar.

```
int Topla(int a, int b)
    {
        return a + b;
    }
```

5. Diziler

Dizi, aynı türde birden fazla veriyi tek bir değişkende saklamamıza olanak tanır. Diziler, sabit bir boyuta sahiptir ve indekslerle erişilir.

```
int[] sayilar = new int[5];
    sayilar[0] = 1;
    sayilar[1] = 2;
    sayilar[2] = 3;
    sayilar[3] = 4;
    sayilar[4] = 5;
    Console.WriteLine(sayilar[2]);
```