**Ders 1 - Dart’ın Temel Özellikleri :**

Dart, Google tarafından geliştirilen bir programlama dilidir. Dart, genel amaçlı bir dil olup, web uygulamaları, mobil uygulamalar, masaüstü uygulamaları ve IoT cihazları gibi birçok alanda kullanılabilir. İşte Dart programlama dilinin bazı özellikleri:

1. Sınıf Tabanlı: Dart, nesne yönelimli bir programlama dilidir. Sınıflar ve nesneler kullanarak programlama yapılır.
2. Tiplendirme: Dart, isteğe bağlı olarak tip kontrolü yapabilen bir dil olduğundan, statik olarak veya dinamik olarak tiplendirilebilir.
3. Açık Kaynaklı: Dart, açık kaynaklı bir projedir. Bu nedenle, topluluk geliştiricileri tarafından kullanılabilir ve geliştirilebilir.
4. Hızlı: Dart, hızlı çalışan bir dil olarak bilinir. Dart'ın JIT (Just-In-Time) ve AOT (Ahead-Of-Time) derleyicileri sayesinde, uygulamalar hızlı bir şekilde çalışır.
5. Kolay Öğrenilebilir: Dart, C, C++, Java ve JavaScript gibi diğer popüler dillerden özellikler alır ve bu nedenle kolay öğrenilebilir.
6. Asenkron Programlama: Dart, asenkron programlama yapmak için async ve await anahtar kelimelerini içerir. Bu, uzun süre çalışan işlemleri etkili bir şekilde yönetmenizi sağlar.
7. Platform Bağımsız: Dart, farklı platformlarda çalışabilen bir dildir. Web, mobil, masaüstü ve IoT gibi birçok platformda çalışabilir.
8. Güncel: Dart, düzenli olarak güncellenen bir dil olarak bilinir. Bu nedenle, yeni özellikler eklenir ve hatalar düzeltilir.

**Ders 2 - Main Metodu :**

Dart dilinde, herhangi bir uygulamayı çalıştırmak için main() metodu gereklidir. Main() metodu, uygulamanın giriş noktasını temsil eder ve uygulamanın çalışması için gereken kodları içerir. Dart dilinde main() metodu şu şekilde yazılır:

void main() {

// uygulama kodları burada yer alır

}

Main() metodunun başında **void** anahtar kelimesi yer alır, bu anahtar kelimesi metotun bir değer döndürmediğini ifade eder. Parantezlerin içine herhangi bir parametre yazılmadığında, main() metodu herhangi bir parametre almaz. Uygulamanın başlatılması ve çalıştırılması sırasında, main() metodu otomatik olarak çağrılır ve uygulamanın çalışması başlar.

**Ders 3 - Değişkenler :**

Dart dilinde değişkenler, değerleri saklamak için kullanılan isimlendirilmiş bellek konumlarıdır. Değişkenler, veri tiplerine sahiptir ve bu veri tiplerine uygun değerler saklarlar. Dart dilinde değişkenlerin tanımlanması ve kullanımı şu şekildedir:

void main() {

  // değişken tanımlama

//veri\_tipi degisken\_adi = deger;

// örnek değişken tanımlamaları

  int sayi = 10;

  double ondalik\_sayi = 3.14;

  String metin = "Merhaba Dünya!";

  bool durum = true;

  print(sayi);

  print(ondalik\_sayi);

  print(metin);

  print(durum);

}

Yukarıdaki örnekte, değişkenlerin tanımlanmasında, önce veri tipi belirtilir, sonra değişkenin adı ve son olarak da değeri belirtilir. Dart dilinde kullanılan bazı veri tipleri şunlardır:

* **int**: tamsayıları temsil eder.
* **double**: ondalıklı sayıları temsil eder.
* **String**: metinleri temsil eder.
* **bool**: mantıksal değerleri temsil eder (true veya false).

Ayrıca, **var** anahtar kelimesi ile değişkenler tanımlanabilir. **var** kullanıldığında, değişkenin veri tipi, atanacak değerden otomatik olarak belirlenir. Örneğin:

 var isim = "Ahmet";

  var yas = 25;

**Ders 4 – Dart Veri Tipleri :**

Dart dilinde kullanılan veri tipleri şunlardır:

1. Numbers: Tamsayılar (int) ve ondalık sayılar (double) olmak üzere iki tür sayısal veri tipi vardır.
2. Strings: Metinsel ifadeleri temsil eder. Tek tırnak veya çift tırnak içinde tanımlanabilir.
3. Booleans: Mantıksal ifadeleri temsil eder. Sadece true veya false değerlerini alabilir.
4. Lists: Dizileri temsil eder. Aynı türden veya farklı türlerde elemanlar içerebilir.
5. Sets: Listeler gibi farklı türden veya aynı türden elemanlar içerebilir. Fakat, set'ler elemanları benzersizdir.
6. Maps: Key-value çiftleri şeklinde verileri temsil eder. Key'ler ve value'lar herhangi bir veri tipinde olabilir.
7. Runes: Unicode karakterleri temsil eder.
8. Symbols: Programcıların kodlarını yazarken kullanabileceği sembolik ifadeleri temsil eder.

**Ders 5 – final ve const :**

Dart dilinde **final** ve **const** anahtar kelimeleri, değişkenlerin değerlerinin değiştirilmesini önlemek için kullanılır.

**final** anahtar kelimesi, değeri tanımlandıktan sonra değiştirilemeyen değişkenler için kullanılır. **final** anahtar kelimesi ile tanımlanan bir değişken, yalnızca bir kez değer atanabilir. Örneğin:

final int sayi = 10;

Yukarıdaki örnekte, **sayi** değişkeni, **final** anahtar kelimesi ile tanımlandığı için yalnızca bir kez değer atanabilir ve daha sonra değeri değiştirilemez.

**const** anahtar kelimesi de **final** gibi davranır, ancak **const** değişkenleri derleme zamanında sabit olarak belirlenir. **const** anahtar kelimesi ile tanımlanan bir değişken, yalnızca derleme zamanında değer atanabilir. Örneğin:

const pi = 3.14;

Yukarıdaki örnekte, **pi** değişkeni, **const** anahtar kelimesi ile tanımlandığı için yalnızca derleme zamanında değer atanabilir ve daha sonra değeri değiştirilemez.

Genel olarak, **final** anahtar kelimesi ile tanımlanan değişkenler çalışma zamanında değer atanırken, **const** anahtar kelimesi ile tanımlanan değişkenler derleme zamanında değer atanır ve programın çalışma hızını artırır.

**Ders 6 – if-else :**

Dart dilinde **if** ve **else** ifadeleri, koşullu karar yapıları oluşturmak için kullanılır. **if** ifadesi, belirtilen koşulun doğru olması durumunda belirtilen kod bloğunu çalıştırır, **else** ifadesi ise koşul yanlış olduğunda çalışacak kod bloğunu belirtir.

Aşağıdaki örnek, kullanıcının yaşına göre belirli bir mesajı gösterir:

void main() {

  int yas = 18;

  if (yas >= 18) {

    print("Yaşınız 18 veya daha büyük.");

  } else {

    print("Yaşınız 18'den küçük.");

  }

}

Yukarıdaki örnekte, **yas** değişkeni 18'e eşit olduğu için **if** bloğu çalışır ve "Yaşınız 18 veya daha büyük." mesajı görüntülenir. Eğer **yas** değişkeni 18'den küçük olsaydı, **else** bloğu çalışacaktı ve "Yaşınız 18'den küçük." mesajı görüntülenirdi.

Ayrıca, **else if** ifadesi de kullanılarak daha fazla koşul belirtilerek daha fazla karar bloğu eklemek mümkündür. Aşağıdaki örnek, kullanıcının notuna göre farklı mesajlar gösterir:

void main() {

  int not = 75;

  if (not >= 90) {

    print("Notunuz AA");

  } else if (not >= 80) {

    print("Notunuz BA");

  } else if (not >= 70) {

    print("Notunuz BB");

  } else if (not >= 60) {

    print("Notunuz CB");

  } else if (not >= 50) {

    print("Notunuz CC");

  } else {

    print("Dersten kaldınız.");

  }

}

Yukarıdaki örnekte, **if** bloğu ilk olarak **not >= 90** koşulunu kontrol eder. Eğer koşul doğru ise "Notunuz AA" mesajı görüntülenir. Eğer koşul yanlış ise **else if** bloğuna geçilir ve diğer koşullar teker teker kontrol edilir. Eğer hiçbir koşul sağlanmazsa **else** bloğu çalışır ve "Dersten kaldınız." mesajı görüntülenir.

**Ders 7 – switch-case :**

Dart dilinde, switch ifadesi, bir değişkenin farklı değerleri için farklı işlemler yapmak için kullanılır. switch ifadesi, bir koşuldan çok farklı değerler için farklı işlemler yapmak için kullanılabilir. Aşağıdaki örnek, kullanıcının seçtiği renge göre farklı mesajlar görüntüler:

void main() {

  String renk = 'kırmızı';

  switch (renk) {

    case 'kırmızı':

      print('Kırmızı renk seçildi.');

      break;

    case 'mavi':

      print('Mavi renk seçildi.');

      break;

    case 'yeşil':

      print('Yeşil renk seçildi.');

      break;

    default:

      print('Geçersiz renk seçildi.');

  }

}

Yukarıdaki örnekte, **switch** ifadesi, **renk** değişkeninin değerini kontrol eder ve belirtilen **case** ifadelerini karşılaştırır. Eğer **renk** değişkeni **'kırmızı'** ise, **"Kırmızı renk seçildi."** mesajı görüntülenir. Eğer **renk** değişkeni **'mavi'** ise, **"Mavi renk seçildi."** mesajı görüntülenir. Eğer **renk** değişkeni **'yeşil'** ise, **"Yeşil renk seçildi."** mesajı görüntülenir. Eğer **renk** değişkeni hiçbir **case** ifadesine uymazsa, **default** ifadesi çalışır ve **"Geçersiz renk seçildi."** mesajı görüntülenir.

Her **case** ifadesinin sonunda **break** anahtar kelimesi kullanılması gerektiğine dikkat edin. Bu, karşılaşılan durumdan çıkıldığını ve diğer **case** ifadelerine bakılmayacağını belirtir. Ayrıca, **default** ifadesi, hiçbir **case** ifadesine uymayan durumlar için bir yedek seçenek olarak kullanılabilir.

**Ders 8 – for - döngüsü :**

Dart dilinde, **for** döngüsü, belirli bir aralıktaki değerler üzerinde tekrarlanan işlemler yapmak için kullanılır.

**for** döngüsü, aşağıdaki gibi üç bileşenden oluşur:

cssCopy code

for (başlatıcı; koşul; artırım) { // işlemler }

* **başlatıcı**: Döngü başlamadan önce bir kere çalışır ve genellikle döngü değişkenlerinin başlangıç değerleri atanır.
* **koşul**: Döngü her çalıştığında kontrol edilir. Koşul sağlandığı sürece döngü çalışmaya devam eder.
* **artırım**: Döngü sonunda her defasında bir kere çalışır ve genellikle döngü değişkenlerinin artırılması gibi işlemler yapılır.

Aşağıdaki örnek, **for** döngüsünün kullanımını göstermektedir:

void main() {

  for (int i = 0; i < 5; i++) {

    print(i);

  }

}

Yukarıdaki örnekte, **for** döngüsü, **i** değişkenini **0** olarak başlatır, **i** değişkeni **5** olduğu sürece çalışır ve her çalıştığında **i** değişkenini **1** artırır. Döngü her çalıştığında **i** değişkeni ekrana yazdırılır.

**for** döngüsü, bir dizi üzerinde de çalıştırılabilir. Aşağıdaki örnekte, **for** döngüsü bir **List** üzerinde çalıştırılmaktadır:

void main() {

  List<int> liste = [1, 2, 3, 4, 5];

  for (int eleman in liste) {

    print(eleman);

  }

}

Yukarıdaki örnekte, **for** döngüsü, **liste** adlı bir **List** üzerinde çalıştırılır. Döngü, her seferinde **liste** içindeki bir elemanı alır ve **eleman** değişkenine atar. Döngü her çalıştığında **eleman** değişkeni ekrana yazdırılır.

**Ders 9 – while ve do-while döngüsü :**

Dart dilinde while ve do-while döngüleri, belirli bir koşul sağlandığı sürece tekrarlanan işlemler yapmak için kullanılır.

**While Döngüsü**

while döngüsü, döngü koşulu doğru olduğu sürece işlemleri tekrarlar. Döngü koşulu yanlış olduğunda, döngü sona erer. Aşağıdaki örnek, while döngüsünün kullanımını göstermektedir:

void main() {

  int i = 0;

  while (i < 5) {

    print(i);

    i++;

  }

}

Yukarıdaki örnekte, **while** döngüsü, **i** değişkeni **5** olana kadar çalışır. Her seferinde, **i** değişkeni ekrana yazdırılır ve **i** değişkeni **1** artırılır.

**Do-While Döngüsü**

**do-while** döngüsü, döngü koşulunu en sona yazarak, işlemlerin en az bir kez yapılmasını garanti eder. Aşağıdaki örnek, **do-while** döngüsünün kullanımını göstermektedir:

void main() {

  int i = 0;

  do {

    print(i);

    i++;

  } while (i < 5);

}

Yukarıdaki örnekte, **do-while** döngüsü, **i** değişkeni **5** olana kadar çalışır. Her seferinde, **i** değişkeni ekrana yazdırılır ve **i** değişkeni **1** artırılır. **do-while** döngüsü, koşul doğru olduğu sürece çalıştığı için, koşul sağlanmadığı halde döngü en az bir kere çalışır.

**Ders 10 – break ve continue :**

Dart dilinde **break** ve **continue** ifadeleri, döngülerde kullanılan kontrol ifadeleridir.

**break** ifadesi, döngü içindeki işlemi tamamlar ve döngüden çıkar. Örneğin, aşağıdaki örnekte, **while** döngüsü, **i** değişkeni **3** olduğunda durdurulur ve döngüden çıkar:

void main() {

  int i = 0;

  while (i < 5) {

    print(i);

    i++;

    if (i == 3) {

      break;

    }

  }

}

Yukarıdaki örnekte, **while** döngüsü **i** değişkeni **3** olduğunda durdurulur ve döngüden çıkar.

**continue** ifadesi ise, döngü içindeki işlemi atlayarak döngünün bir sonraki adımına devam eder. Örneğin, aşağıdaki örnekte, **for** döngüsü, **i** değişkeni **3** olduğunda atlanır ve döngünün bir sonraki adımına geçilir:

void main() {

  for (int i = 0; i < 5; i++) {

    if (i == 3) {

      continue;

    }

    print(i);

  }

}

Yukarıdaki örnekte, **for** döngüsü **i** değişkeni **3** olduğunda atlanır ve döngünün bir sonraki adımına geçilir. Bu nedenle **3** sayısı yazdırılmaz ve diğer sayılar yazdırılır.

**break** ve **continue** ifadeleri, döngülerde kullanılarak programın kontrolünü sağlamada oldukça faydalıdır.

**Ders 11 – null safety ve late kavramı :**

Dart dilinde, null safety özelliği ile birlikte late anahtar kelimesi de kullanılmaktadır.

null safety, kodun daha güvenli hale getirilmesini sağlar. Bu özellik sayesinde, değişkenlerin null olup olmadıklarını kontrol etmek ve null değerlerle ilgili hataları önlemek daha kolay hale gelir. null safety özelliği varsayılan olarak Dart 2.12 ve sonraki sürümlerde kullanılmaktadır.

late anahtar kelimesi ise değişkenlerin değerlerinin ileride atanacağını belirtmek için kullanılır. Böylece, değişkenin ilk değeri atanmadan kullanılması durumunda hata alınmaz.

Aşağıdaki örnekte, null safety ve late anahtar kelimelerinin kullanımı gösterilmektedir:

late String lateExample;

String? nullableExample;

void main() {

  // nullableExample = null;  // Hata! nullableExample null olabilir.

  lateExample = "Late example";

  print(lateExample); // Geç atama yapıldığı için hata alınmaz.

}

Yukarıdaki örnekte, **nullableExample** değişkeni null olabilirken, **lateExample** değişkeninin geç atanacağı belirtilmiştir. **lateExample** değişkeni kullanılmadan önce atanmadığı halde, program hata vermez. Ancak **nullableExample** değişkeni null olduğunda, kullanılmadan önce atanmamış olması hata verecektir.

**Ders 12 – List Yapısı :**

Dart programlama dilinde List yapısı, birden fazla veri elemanını tutmak için kullanılır. List yapısı, bir sıralı koleksiyondur, yani elemanlar sırayla indekslenir ve bu indekslerle erişilebilir.

Dart'ta List yapısı iki türde tanımlanabilir:

1. Fixed-Length List (Sabit Uzunluklu Liste): Belirli bir uzunluğa sahip bir liste tanımlarız. Bu liste boyutu tanımlanırken belirlenir ve daha sonra değiştirilemez. Bu türdeki bir liste aşağıdaki gibi tanımlanır:

dartCopy code

List<int> fixedList = List(5);

1. Growable List (Genişletilebilir Liste): Boyutu önceden belirlenmemiş bir listeyi ifade eder. Yani, liste boyutu dinamik olarak değiştirilebilir. Bu türdeki bir liste aşağıdaki gibi tanımlanabilir:

dartCopy code

List<int> growableList = [];

List yapısı, Dart'ta standart kütüphanenin bir parçasıdır, bu nedenle kullanmak için ekstra bir kütüphane yüklemenize gerek yoktur. Liste elemanlarına, indeks numarasıyla erişilebilir. Örneğin:

void main() {

  var myList = [1, 2, 3, 4, 5];

  print(myList[0]); // 1

  print(myList[3]); // 4

  // eleman ekleme

  myList.add(6);

  myList.addAll([7, 8]);

// eleman silme

  myList.remove(5);

  myList.removeAt(0);

// eleman filtreleme

  var filteredList = myList.where((element) => element % 2 == 0).toList();

// eleman sıralama

  myList.sort();

}

**Ders 13 – List Metodları :**

Dart List yapısı, birçok farklı metoda sahiptir. Aşağıda, en yaygın kullanılan Dart List metotlarından bazıları açıklanmıştır:

add(): Bir eleman ekler.

addAll(): Birden fazla eleman ekler.

insert(): Belirli bir indekse eleman ekler.

remove(): Belirtilen elemanı listeden siler.

removeAt(): Belirtilen indeksteki elemanı listeden siler.

clear(): Listeyi temizler.

length: Listenin uzunluğunu döndürür.

indexOf(): Belirtilen elemanın indeksini döndürür.

lastIndexOf(): Belirtilen elemanın son indeksini döndürür.

contains(): Belirtilen elemanın listede olup olmadığını kontrol eder.

forEach(): Listenin her bir elemanı üzerinde bir işlem yapar.

map(): Listenin her bir elemanı üzerinde bir işlem yapar ve yeni bir liste döndürür.

where(): Belirli bir koşulu sağlayan elemanları içeren yeni bir liste döndürür.

sort(): Listeyi belirtilen bir kurala göre sıralar.

sublist(): Belirtilen aralıktaki elemanları içeren yeni bir liste döndürür.

void main() {

  // Listeyi oluştur

  List<String> myFruits = ['elma', 'armut', 'muz'];

  // Eleman ekle

  myFruits.add('çilek');

  print(myFruits); // ['elma', 'armut', 'muz', 'çilek']

  // Birden fazla eleman ekle

  myFruits.addAll(['kivi', 'portakal']);

  print(myFruits); // ['elma', 'armut', 'muz', 'çilek', 'kivi', 'portakal']

  // Belirli bir indekse eleman ekle

  myFruits.insert(1, 'şeftali');

  print(myFruits); // ['elma', 'şeftali', 'armut', 'muz', 'çilek', 'kivi', 'portakal']

  // Belirtilen elemanı listeden sil

  myFruits.remove('elma');

  print(myFruits); // ['şeftali', 'armut', 'muz', 'çilek', 'kivi', 'portakal']

  // Belirtilen indeksteki elemanı listeden sil

  myFruits.removeAt(2);

  print(myFruits); // ['şeftali', 'armut', 'çilek', 'kivi', 'portakal']

  // Listeyi temizle

  myFruits.clear();

  print(myFruits); // []

  // Liste uzunluğunu döndür

  List<int> myNumbers = [1, 2, 3];

  print(myNumbers.length); // 3

  // Belirtilen elemanın indeksini döndür

  List<String> myColors = ['kırmızı', 'mavi', 'yeşil'];

  print(myColors.indexOf('mavi')); // 1

  // Belirli bir koşulu sağlayan elemanları içeren yeni bir liste döndür

  List<int> myValues = [1, 2, 3, 4, 5];

  var filteredList = myValues.where((element) => element % 2 == 0).toList();

  print(filteredList); // [2, 4]

  // Listeyi belirtilen bir kurala göre sırala

  List<int> mySortList = [5, 3, 1, 4, 2];

  mySortList.sort();

  print(mySortList); // [1, 2, 3, 4, 5]

}

**Ders 14 – Liste Üzerinde Gezinme :**

void main() {

  List<String> myFruits = ['elma', 'armut', 'muz'];

  // 1. Yöntem: For döngüsü kullanarak gezinme

  for (int i = 0; i < myFruits.length; i++) {

    print(myFruits[i]);

  }

  // 2. Yöntem: For-in döngüsü kullanarak gezinme

  for (String fruit in myFruits) {

    print(fruit);

  }

  // 3. Yöntem: ForEach metodu kullanarak gezinme

  myFruits.forEach((fruit) => print(fruit));

  // 4. Yöntem: Iterator kullanarak gezinme

  var iterator = myFruits.iterator;

  while (iterator.moveNext()) {

    print(iterator.current);

  }

}

Yukarıdaki kod örneğinde, öncelikle **List<String>** türünde bir **myFruits** liste tanımlanır. Daha sonra, bu listedeki öğeleri dört farklı yöntemle gezinir ve ekrana yazdırılır:

1. For döngüsü kullanarak gezinme: Listenin uzunluğuna göre bir **for** döngüsü kullanılır ve listenin her bir öğesi, öğe indeksi kullanılarak ekrana yazdırılır.
2. For-in döngüsü kullanarak gezinme: Listenin her bir öğesi, **for-in** döngüsü kullanılarak doğrudan döngü değişkeni olarak belirtilir ve ekrana yazdırılır.
3. ForEach metodu kullanarak gezinme: **forEach** metodu, listedeki her bir öğe için belirtilen işlemi gerçekleştirir. Bu örnekte, her bir öğe doğrudan ekrana yazdırılır.
4. Iterator kullanarak gezinme: Iterator, listedeki her bir öğeye sırayla erişir ve **while** döngüsü kullanılarak bu öğeler ekrana yazdırılır.