

Görsel Programlama

Ortam kurulumu, Multiplatform proje - Debugging - Dart
devtools - Dart sentaksı

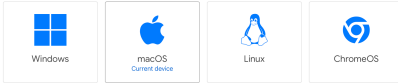
Emir ÖZTÜRK

Flutter Kurulumu

- flutter.dev
- Get Started
- Windows
- MacOS
- Linux
- Chrome OS

Choose your development
platform to get started

[Get started](#) > [Install](#)



[Developing in China](#)

If you want to use Flutter in China, check out [using Flutter in China](#). If you're not developing in China, ignore this notice and follow the other instructions on this page.

如果你正在中国的网络环境下配置 Flutter，请参考 [在中国网络环境下使用 Flutter 文档](#)。

Unless stated otherwise, the documentation on this site reflects the latest stable version of Flutter. Page last updated on 2020-01-31. [View source](#) or [report an issue](#).

Flutter'ın kurulumu için flutter.dev sitesine gidilebilir.

Kullanılacak ortama göre istenilen binary'nin indirilmesi gerekmektedir.

Flutter'ın bir çok işletim sistemi için derlenmiş versiyonları bulunmaktadır.

Flutter Kurulumu

Windows

- Vscode ile otomatik kurulum
- Manuel indirme
 - Path'e ekleme işlemleri
- <https://docs.flutter.dev/get-started/install/windows/desktop>

Install the Flutter SDK

To install the Flutter SDK, you can use the VS Code Flutter extension or download and install the Flutter bundle yourself.

[Use VS Code to install](#) [Download and install](#)

Use VS Code to install Flutter

To install Flutter using these instructions, verify that you have installed [Visual Studio Code 1.46](#) or later and the [Flutter extension](#) for VS Code.

Prompt VS Code to install Flutter

1. Launch VS Code

2. To open the **Command Palette**, press **Ctrl+Shift+P**

3. In the **Command Palette**, type **Flutter**

4. Select **Flutter: New Project**

5. VS Code prompts you to locate the Flutter SDK on your computer.

1. If you have the Flutter SDK installed, click **Locate SDK**.

2. If you do not have the Flutter SDK installed, click **Download SDK**.

This option sends you the Flutter install page if you have not installed Git for Windows as directed in the [development tools prerequisites](#).

6. When prompted **Which Flutter template?**, ignore it. Press **Enter**. You can create a test project after checking your development setup.

Flutter.dev'den indirilen zip dosyası istenilen yere açılır. Sitede verilen örnekte c:

\src\flutter altına açılması istense de farklı bir yerde oluşturmak mümkündür.

Flutter.zip'in açıldığı klasör Binary dosyalara erişmek amacı ile path'e

eklenmelidir. Windows, Linux ve macOS için bu ekleme işleminin

gerçekleştirilmesi gerekse de Linux ve macOS için bir paket yöneticisi

kullanılıyorsa bu işlemi paket yöneticisi üstlenecektir.

Path'e ekleme işleminde ana klasörün içerisindeki bin klasörünün kullanılması

gerekmektedir.

Bütün işlemler bittiğinde terminal (Windows için terminal, powershell veya cmd)

kullanılarak işlemin doğru yapıldığı kontrol edilebilir. Bunun için terminalde flutter yazıldığında eğer komut bulunamıyorsa ekleme işlemi yapılamamıştır demektir. Flutter doctor ile kurulum için gereken araçların bilgisayarda mevcut olup olmadığı kontrol edilir. Örneğin android ortamına yapılacak bir geliştirmede android studio'ya, iOS tarafında yapılacak bir geliştirme için ise Xcode'a ihtiyaç duyulmaktadır.

Flutter Kurulumu MacOS

- Flutter.zip ya da brew install flutter
- Flutter.zip indirildiyse klasöre açma
- Klasörü Path'e ekleme (export)
- Vscode üzerinden kurulum
- Brew install flutter



MacOS ya da Linux dağıtımlarında uygulama kurmak için çeşitli paket yöneticileri kullanılabilir. MacOS için brew kullanılabilir (brew install flutter). Brew ile flutter kurulumu yapıldığında ayrıca path'e bir ekleme işlemi yapılmamaktadır. Eğer yine de manuel bir şekilde kurulum yapılmak isteniyorsa export komutu ile path'e flutter'ın yolu eklenebilir.

Flutter Kurulumu Linux

- Flutter.zip ya da snap install flutter --classic
- Flutter.zip indirildiyse klasöre açma
- Klasörü Path'e ekleme (export)
- Vscode üzerinden kurulum
- Flutter doctor
- Android Studio

Install Flutter using snapd

The easiest way to install Flutter on Linux is by using [snapd](#). Once you have snapd, you can [install Flutter using the following command](#):

```
$ snap install --classic flutter
```

Note: Once the snap is installed, you can use the following command to set up the PATH:

```
$ flutter --add-path
```

Install Flutter manually

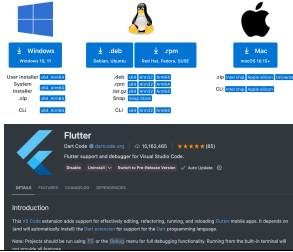
If you don't have [snapd](#), or can't use it, you can install Flutter manually.

Linux'taki kurulum için snap kullanılabilir. MacOS'te olduğu gibi Linux'ta da snap kullanıldığında bir export işlemi yapılmamaktadır. Ayrıca yine diğer platformlarda olduğu gibi flutter doctor ile olası eksiklerin görülmesi sağlanabilir.

Vscode

Download Visual Studio Code
Free and built on open source. Integrated Git, debugging and extensions.

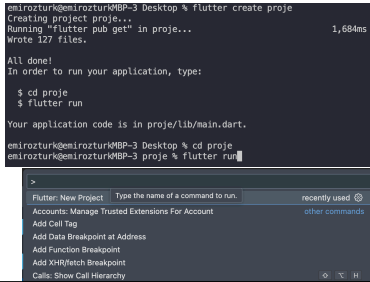
- Editör
- Eklenti kurulumu
- Flutter



Flutter geliştirme için bir çok ortam kullanılabilir (Android studio, IntelliJ Idea, Vscode, vi, vim). Vscode'da bulunan flutter eklentileri geliştirme aşamasında belirli kolaylıklar sağlamaktadır. Ayrıca Widget'ların belirli kalıplarla oluşturulabilmesi için widget snippets eklentisi de kurulabilir.

Yeni Proje Oluşturma

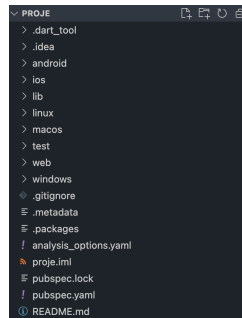
- flutter create proje_adi
- Cd proje_adi
- flutter run
- Vscode ile
- Ctrl shift p
- Flutter new project



Flutter'da yeni proje oluşturmak için vscode'da ctrl shift p ile "Command Palette" açıldıktan sonra flutter yazıp Create project seçilebilir. Terminalden ise flutter create komutu kullanılarak proje açılabilir. Create komutundan sonra proje adı verilmelidir. Proje adında büyük harf geçmesine izin verilmemektedir. Projenin verilen adında bir klasör oluşturulduğu için proje dosyalarına cd proje_adi ile erişilebilir. Projenin çalıştırılması için ise flutter run kullanılabilir.

Klasör Yapısı

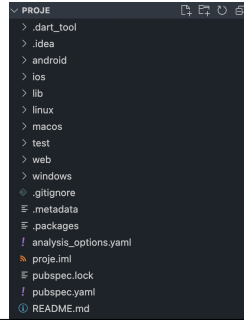
- Platform klasörleri
 - android
 - ios
 - linux
 - macos
 - web
 - windows



Flutter create komutu kullanıldığında derleme yapılacak her ortam için bir klasör oluşturulur. Windows, android, macOS, linux, iOS gibi klasörler otomatik olarak oluşturulan klasörlerdir.

Klasör Yapısı

- Geliştirme klasörleri
 - lib
 - test
- Geliştirme dosyaları
 - pubspec.yaml



Lib içerisinde ise yazdığımız kodlar bulunmaktadır. Bunun dışında paket ve kaynak yönetimi için pubspec.yaml kullanılmaktadır. Ayrıca uygulama testlerinin yazılması için bir test klasörü de oluşturulmaktadır.

İlk Uygulama

```
import 'package:flutter/material.dart';

void main() => runApp(const MyApp());

class MyApp extends StatelessWidget {
  const MyApp({Key? key}) : super(key: key);
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: const MyHomePage(title: 'Flutter Demo Home Page'),
    ); // MaterialApp
  }
}
```

Flutter uygulaması oluşturulduğunda bir çok dilde olduğu gibi bir main fonksiyonu ile başlar.

Uygulama çalıştırılırken Stateless ve Stateful olmak üzere iki farklı Widget türü oluşturulabilir. Stateless ve Stateful widget farkı ilerleyen haftalarda açıklanacaktır.

Bir sınıfın main metodunda runApp ile çağırılabilmesi için Widget sınıflarından birini inherit etmesi gerekmektedir.

Stateless bir Widget üzerinde build metodu override edilmeli ve bir Widget döndürülmelidir. Örnekte gösterilen MaterialApp, bir başlığı ve bir gövdesi olan bir yapıyı döndürmektedir.

İlk Uygulama

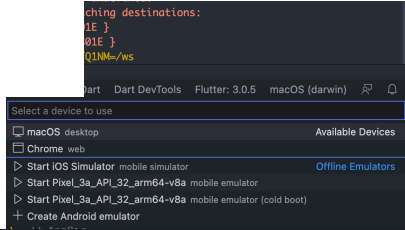
```
class MyHomePage extends StatefulWidget {  
  const MyHomePage({Key? key, required this.title}) : super(key: key);  
  final String title;  
  State<MyHomePage> createState() => _MyHomePageState();  
}
```

```
class _MyHomePageState extends State<MyHomePage> {  
  int _counter = 0;  
  void _incrementCounter() {setState(() { _counter++;});}  
  @override  
  Widget build(BuildContext context) {  
    return Scaffold(  
      appBar: AppBar(  
        title: Text(widget.title),  
      ), // AppBar  
      body: Center(  
        child: Column(  
          mainAxisAlignment: MainAxisAlignment.center,  
          children: <Widget>[  
            const Text("You have pushed the button this many times:"),  
            Text("${_counter}", style: Theme.of(context).textTheme.headline4),  
          ], // <Widget>[]  
        ), // Column  
      ), // Center  
      floatingActionButton: FloatingActionButton(  
        onPressed: _incrementCounter, tooltip: 'Increment',  
        child: const Icon(Icons.add),  
      ), // FloatingActionButton  
    ); // Scaffold  
  }  
}
```

Stateful bir widget tanımlandığında bu widget'ın createState metodunun override edilmesi gerekmektedir. Bu durumda bu metodun döndüreceği “State” in de ayrıca bir sınıf olarak yazılması gerekir. Bir sınıfı state olarak tanımlamak için State sınıfından extend etmek gerekmektedir. State sınıfı generic olup StatefulWidget olan sınıf türünde tanımlanmalıdır. Build metodu stateless widget ile aynıdır. Widget döndürülür ve context alınır. Bunun dışında state değişimleri için setState() metodu kullanılmaktadır.

Farklı Ortamlarda Çalışma

- Native
- Web
- Emulator



Derleme esnasında farklı ortamlara erişebilmek amacı ile vscode üzerinde sağ alttan platform seçimi yapılabilmektedir. Bu seçim yapılırken mevcut emulatörler, bağlı cihazlar veya kurulumu yapılan ortamların desteklediği platformlar listelenir.

Debugging

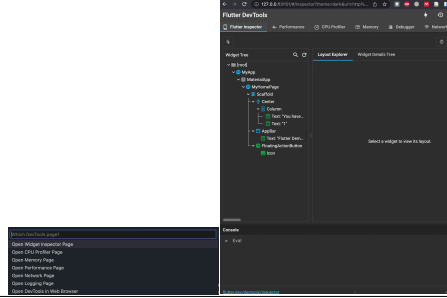
- Print debugging
- Breakpoint
- Çalışma anında durma
- Mevcut değişkenleri inceleme

```
17 }  
18 class MyHomePage extends StatefulWidget {  
19   const MyHomePage({Key? key, required this.title}) : super(key: key);  
20   final String title;  
21   State<MyHomePage> createState() => _MyHomePageState();  
22 }  
23 class _MyHomePageState extends State<MyHomePage> {  
24   int _counter = 0;  
25   void _incrementCounter() {setState(() { _counter++;});}  
26   @override  
27   Widget build(BuildContext context) {  
28     return Scaffold(  
29       appBar: AppBar(  
30         title: Text(widget.title),  
31       ), // AppBar  
32       body: Center(  
33         child: Column(  
34           mainAxisAlignment: MainAxisAlignment.center,  
35           children: <Widget>[  
36             const Text("You have pushed the button this many times:"),  
37             Text("${_counter}", style: Theme.of(context).textTheme.headline4),  
38           ], // <Widget>[]  
39         ), // Column  
40       ), // Center  
41       floatingActionButton: FloatingActionButton(  
42         onPressed: _incrementCounter, tooltip: 'Increment',  
43         child: const Icon(Icons.add),  
44       ), // FloatingActionButton  
45     ); // Scaffold  
46   }  
47 }
```

Debugging için en basit olan yöntem belirli çıktıları konsola log olarak düşmektir. Bunun dışında diğer ide veya editörlerde olduğu gibi flutter'da da breakpointler ile debugging işleminin gerçekleştirilmesi mümkündür. İşlemler tetiklendiğinde çalışma durdurulmakta ve o anki değişkenlerin durumları incelenebilmektedir.

Flutter Devtools

- Widget tree
- Performans
- Cpu
- Bellek
- Debugging
- Ağ
- Loglama
- Uygulama boyutu



Devtools kullanılarak belirli açılardan uygulama performansı ölçülebilmektedir. Oluşturulan widget'ların hiyerarşik yapısı bir ağaç olarak listelenebilir, cpu, bellek, çalışma zamanı, ağda harcanan zaman gibi etmenler devtools ile takip edilebilmektedir. Ayrıca uygulamanın boyutunun küçültülmesi istendiğinde de devtoolstan kaynak kontrolü yapılabilir.

Dart Sentaksı

Değişkenler

- var
 - int, String...
- final
- enum
- Nullable
 - ?

Değişken tanımlama için diğer dillerde olduğu gibi değişkenin türü belirtilebilmektedir. Bunun yerine değişkenin türü atama işleminden dolayı çözülebileceğinden var kullanmak da mümkündür. Strictly typed dillerden farklı olarak var değişkenlerin aynı satırda atanması zorunluluğu bulunmamaktadır. Eğer bir değişkenin değiştirilmesi gerekmiyorsa ve ilk atama sonrası değişmiyorsa var yerine final kullanılabilir. Dart diğer dillerde olduğu gibi enum'ları desteklemektedir. Enum'lar verilen sayısal değerler için bir etiket olarak kullanılabilir. Ayrıca Dart dilinde değişkenler ? Karakteri ile nullable olarak tanımlanabilirler. Primitif değişkenler null olmayıp ilk tanımlamada bir değer verilmezse varsayılan değerleri alırlar. Nullable tanımlanan değişkenler ise null olabilir ve değer ataması sonradan yapılabilir.

Dart Sentaksı Koşullar, Switch, Döngüler

- C sentaksı ile aynı (if, switch, for)
- Foreach döngüsü bulunuyor
 - for(var d in değişkenler)print(d);

Koşul, switch ve döngü için sentaks c sentaksı ile aynı şekilde kullanılabilir. Bunun dışında iteratif objelerin üzerinde kullanılabilmesi amacı ile diğer dillerdeki foreach döngüsüne benzer bir döngü de bulunmaktadır. Foreach yerine yine for anahtar kelimesi ile tanımlama yapılabilir ve for(var değişken in liste) sentaksı ile kullanılabilir.

Dart Sentaksı Fonksiyonlar

- C sentaksı ile aynı
- Anonim fonksiyonlar
 - Bulundukları yerde tanımlanabilir
 - Tek satırda () =>
 - Çok satırda () {}
- Fonksiyonlar parametre olarak aktarılabilir
 - Widgetlar arası aktarımda ihtiyaç duyulmakta

Fonksiyon tanımı da aynı şekilde c sentaksındaki gibidir. Bunun dışında anonim fonksiyonlar tanımlanabilir. Bu fonksiyonlar tek satırlı ise parametre alımı ve işlem için (param)=>işlem; şeklinde tanımlanabileceği gibi birden fazla satırdan oluşuyorsa da (param){işlem1;işlem2;...} şeklinde kullanılabilir. Fonksiyonlar bir parametre olarak başka fonksiyonlara da verilebilir. Bu özellik bir çok farklı alanda kullanılabilirle beraber (örneğin sıralama algoritmasının değerlere fonksiyon olarak verilmesi) ön yüz için birden fazla pencereye sahip olan uygulamalarda bir pencerenin sahip olduğu bir değerin değiştirilmesi için bir fonksiyonun başka pencerede tetiklenmesi gerektiğinde o fonksiyonun parametre olarak ilk pencereye aktarılması gibi durumlarda kullanılmaktadır.

Dart Sentaksı

Sınıflar

- C# ve Java sentaksı ile aynı
- Alanların private yapılması için alt tire (_)
- Yapıcı, getter ve setter için özel sentaks

Sınıf tanımı için class anahtar kelimesi ile tanımlama yapılmaktadır. Alanların private yapılması için değişkenlerin _ ile başlaması gerekmektedir. Herhangi bir kalıtım işleminde extends kelimesi kullanılır. Yapıcı tanımında klasik değişken ataması yapılabileceği gibi bazı özel yazım stilleri de bulunmaktadır.

Dart Sentaksı

Yapıcı

- Java veya C# stili kullanılabilir.
- İsimlendirilmiş argümanlar kullanılabilir
 - {}
- İsimlendirilmiş argümanlar zorunlu istenebilir
 - Required
- İsimlendirilmiş argümanlar sonda olmalıdır.
 - (Argüman,{argüman})

Yapıcının gövdesiz bir şekilde yazılması ve parametre alımı esnasında this anahtar kelimesi ile private değişkenlere atanması mümkündür. Ayrıca yapıcıda isimli değişkenler için {} karakterleri arasında tanımlama yapılabilmektedir. Bu tarz isimli tanımlamalar zorunlu tanımlamaların sonunda yapılmalıdır. Tüm parametreler isimlendirilmiş yapılırsa seçimlik olarak atanabileceği için argümanlara zorunlu ise required parametresi eklenmelidir.

Dart Sentaksı

Yapıcı

- Argüman ismi farklı verilecekse yapıcı sonrasında : ile eşitleme
 - (Argüman, argüman):_degisken = Argüman,_degisken2 = argüman
- İsimlendirilmiş yapıcı kullanılabilir
 - SınıfAdı.metotAdı():atamalar
- Başka bir yapıcının çağırılması
 - SınıfAdı.metotAdı():this(parametreler)
- Factory metotlar
 - Statik nesneleri döndürmek için
 - factory SınıfAdı(){}

Argümanın ismi değişken isminden farklı ise parametre olarak alınan argüman : karakterinden sonra atanabilir.

Aynı sentaks isimli yapıcılar için kullanılabilir.

SınıfAdı.istenilenMetotAdı():atamalar şeklinde isimli yapıcı kullanmak mümkündür.

Başka bir yapıcı ya da temel yapıcı çağırılacaksa yine :’den sonra this ile çağırma işlemi gerçekleştirilebilir.

Factory metotlar için özel bir sentaks da bulunmaktadır. Bunun için factory SınıfAdı(){} verildikten sonra bir nesne örneği parantezler içerisinde

verilebilmektedir.

Dart Sentaksı

Getter - Setter

- Getter
 - Dönüş türü get ÖzellikAdı => _özellikAdı
 - int _degisken;
 - int get Degisken => _degisken;
- Setter
 - set ÖzellikAdı(AtamaTürü value) => _degisken = value

Getter ve setter için özel sentaks kullanılabilmektedir. Bunun için get yazdıktan sonra getter ismi => döndürülecek özel değer şeklinde bir yazım kullanılabilir. Setter için ise set setter ismi(atama türü value) => private değer = value şeklinde bir yazım yapılabilir.

Dart Sentaksı

Diğer Özellikler

- Operatör aşırı yükleme
- Kopya alıcı fonksiyon
- Kalıtım
- Soyut sınıflar
 - Extends
- Arayüzler (abstract class)
 - Implements
- Jenerikler ve Koleksiyonlar

Ayrıca Dart üzerinde operatör aşırı yükleme ve kopya alıcı fonksiyon desteği de bulunmaktadır. Kalıtım desteği bulunmakta, arayüzler için ise soyut sınıflar implements kelimesi ile kullanılmaktadır. Ayrıca yine diğer dillerde olduğu gibi jenerik ve koleksiyon desteği de bulunmaktadır.