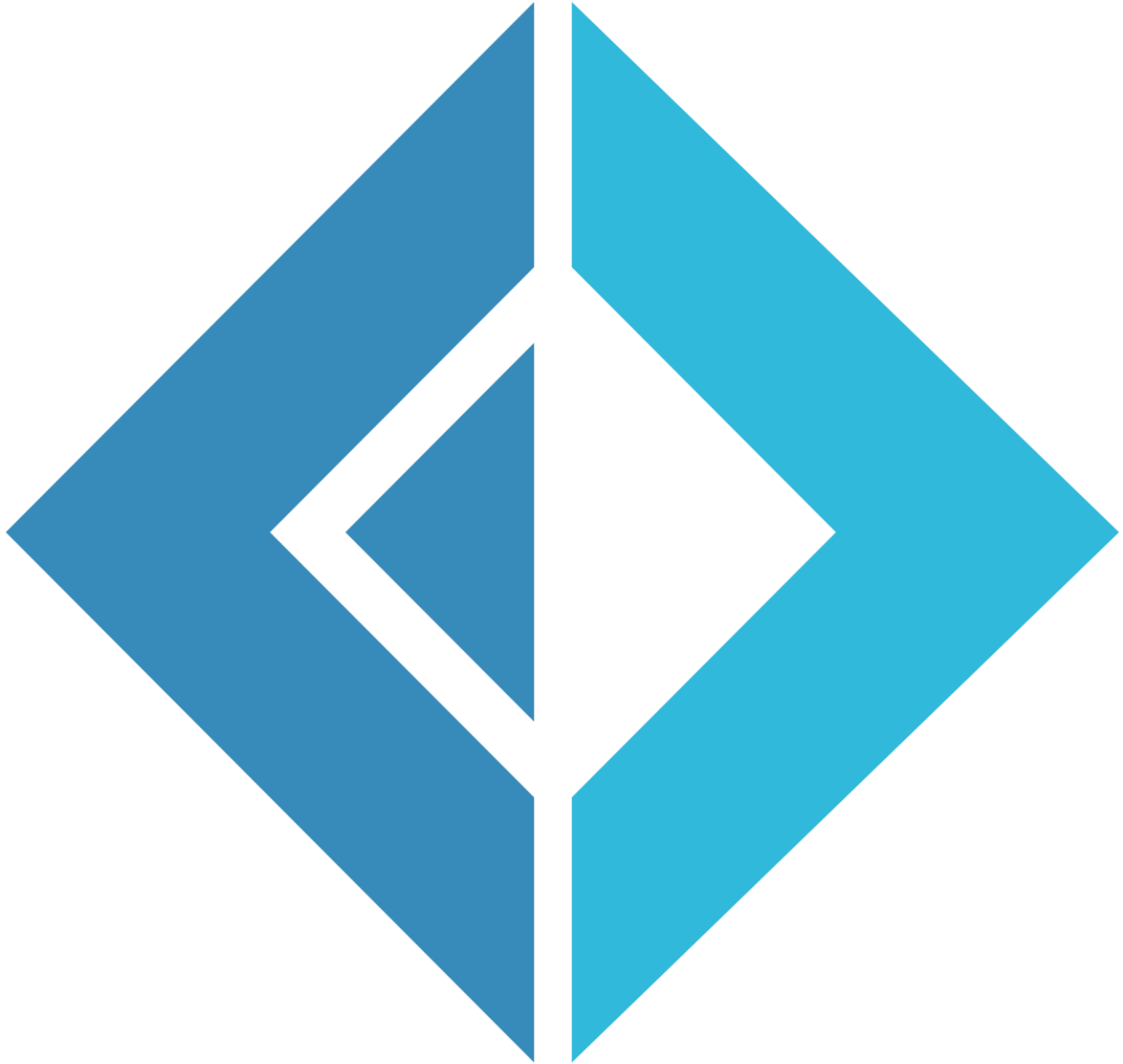


# **F# 3.1 Ebook**



**Burak Özdemir**

### Önsöz

## 1. F# PROGRAMLAMA DİLİ

- 1.1 Kısaca Fonksiyonel Programlama
- 1.2 F#'ın Tarihçesi
- 1.3 F#'ın Fonksiyonel Programlamadaki Yeri
- 1.4 F# ile Yapılmış Projeler
- 1.5 F# İçin Kaynaklar ve Topluluk Siteleri
- 1.6 IDE(Uygulama Geliştirme Ortamı) Kurulumu ve Kullanımı
- 1.7 F# ile İlk Uygulama

## 2. F# İLE FONKSİYONEL PROGRAMLAMAYA GİRİŞ

- 2.1 Kullanıcı ile Etkileşim I/O
- 2.2 Veri Tipleri
- 2.3 Koşul Yapıları
- 2.4 Döngüler
- 2.5 Fonksiyonlar
- 2.6 Koleksiyonlar
- 2.7 Tuples(Demetler) ve Records(Kayıtlar)
- 2.8 Hata Yakalama
- 2.9 F# ile Nesne Yönelimli Programlama

## 3. ÖRNEK UYGULAMA

- 3.1 Uygulama Tanımı
- 3.2 Geliştirme Aşaması

## 4. KAYNAKLAR

## ÖNSÖZ

---

Bu dökümanın amacı, herhangi bir sebeple Fonksiyonel programlamaya özellikle F#’a ilgi duyan ve bu programlama dilini öğrenmek isteyen yazılım severlere bu dili olabildiğince hızlı ve etkili bir biçimde öğretmektir. Bu dökümandaki bilgileri anlamak için herhangi bir programlama tecrübeniz olmasına gerek yoktur. Not defteri ve Browser gibi araçları kullanma bilgisine sahipseniz bu dökümanda anlatılan bilgileri anlamakta fazla zorlanmayacağınıza emin olabilirsiniz

Bu dökümanda, F# programlama diline ilişkin konular ardı ardına ve birbiriyle alakalı olarak anlatılmıştır. Dolayısıyla bu dökümandan en iyi derecede performans almak istiyorsanız, konuları en baştan en sona kadar **sıra atlamadan** takip etmeniz gerekmektedir.

Bu projede bize öncülük eden saygıdeğer hocam Öğr. Gör. Muhammed Tekin’e teşekkür ederiz.

Bu döküman Creative Commons lisansı altındadır. Buna göre, buradaki bütün belgelerden herkes ücretsiz olarak yararlanabilir. Eğer isterseniz burada gördüğünüz belgelerin çıktısını alabilir, arkadaşlarınızla gönül rahatlığıyla paylaşabilirsiniz. Ancak bu belgeleri başka bir yerde kullanacaksanız, dökümanı kaynak olarak göstermeli ve bu belgeleri kesinlikle satmamalısınız. Arzu ederseniz belgeleri çoğaltıp ücretsiz olarak dağıtabilirsiniz.



## 1.1 Kısaca Fonksiyonel Programlama

Öncelikle fonksiyonel programlama, zannedilebileceği gibi "işlevsel programlama" değil "fonksiyon tabanlı programlama"ya daha yakın bir kavramdır.

Fonksiyonel programlama disiplini yazılımı modellerken yazılımın fonksiyonlardan oluştuğunu temel alır ve veri yapılarının içeriğini değiştirmek yerine onlardan yeni veri yapıları türeten fonksiyonlar hayal eder. Tasarım ve geliştirme süreci bu düşünceyle işler

Bu yüzden bildiğimiz içini degistirdigimiz degişken kavramı fonksiyonel programlamada var olsa bile kullanı-mından mumkun oldugunca kaçınılır. Zira bu degişken orucunun en büyük avantajı programın state(durum) konusundaki cakişmalarını minimize etmeyi saglamasi ve multithreading(çoklu işlem) gibi işleri çakişmalardan kaçınarak çok kolay şekilde yapabiliyor olmasıdır.

Tarihteki ilk fonksiyonel programlama dili Lisp'tir. Günümüzdeki bilinen örnekleri ise OCaml ve Haskell'dir. Ayrıca bu kitapta bizim değineceğimiz .Net dünyasındaki gelecek vaad eden dil ise OCaml'dan türemiş olan F#'tir.

## 1.2 F#'ın Tarihçesi

F# Microsoft Research tarafından geliştirilern ve .Net Framework üzerinde çalışan fonksiyonel programlama dilidir. Bildiğiniz gibi günümüzden en popüler yazılım dili olarak Microsoft'un yazılım dili C# kullanılmaktadır ve popüler olan bu dil C# 6.0 sürümüne yükseldi ve .Net Framework mimarisi ile oldukça geniş bir tabanda zenginleştirildi.

Ancak gün geldi artık C# yazılım geliştiriciler için tek bir alanda yetersiz kalmaya başladı.Bu alan finans-muhasebenin temelini oluşturan matematiksel işlemlerdir ve bu matematiksel işlemler için büyük sayılı ve çok sayıda karmaşık formüller için C#'ın hızı yetersiz kalarak özellikle işi para olan firmalara zaman sarfiyatını doğurdu.

Durum böyle olunca sevgili Microsoft'un yazılım mühendisleri masaya oturdular ve karmaşık matematiksel işlemlerin daha hızlı çözümlenebileceği bir dil oluşturmaya karar verdiler ve 2005 yılında F#'ı geliştirdiler.

"Peki neden adı F# ?" oldu gibi bir soru gelebilir aklınıza,cevap şöyle: Özellikle finansal işlemlerde kullanılan bir dilin adı da finansla ilgili olsun dediler ve Functional# olsun istediler ve F# kısaltmasıyla piyasaya sürdüler.

C# ile karşılaştırıldığında cidden matematiksel işlemlerde oldukça hızlı olduğunu göreceğiniz bu dilin kullanımı syntax (söz dizimi-yazım biçimi) C#'tan farklı olsa da pek zorlayıcı olduğunu söyleyemeyiz.

Nedendir bilinmez ancak Microsoft bi ara F# programlama dili için bir ara desteğini kaldırdığını yani o dilde herhangi bir geliştirme olmayacağını duyurmuştu.İşin daha enteresan yanı ise F# artık 3.1 versiyonuna yükseldi ve bu konuda piyasada artık bu dilin öğrenilmesi konusunda Türkçe olmasada İngilizce dilinde pek çok eğitim kitabı var. Umarız kitabımız F# için Türkçe kaynak arayanlar için faydalı olur.

## 1.3 F#'ın Fonksiyonel Programlamadaki Yeri

F# birçok sıkıntılı hesaplama problemlerini çözmek için en iyi yaklaşımların başında gelmektedir. F#, fonksiyonel programlama dışında imperative programming ve object-orientated yaklaşımlarına uyumluluk gösterir. Strongly typed özelliği mevcuttur, bu sayede programcıların belirsiz durumlar olmadığı sürece değişken tiplerini açıkça

belirtmelerine gerek yoktur. C#’ta buna örnek olarak var keyword’ü gösterilebilir (tam karşılığı olmasa da). Bunun dışında inferred typing özelliğini de destekler.

F# temel olarak OCalm (object-orientated fonksiyonel programlama) dilinden modellendi. Ve .NET ile güçlendirildi. Generics kısmını hiçbir kod değiştirmeden destekler. Hatta IL (Intermediate Language) kodunu destekler. F# derleyicisi, sadece herhangi bir CLI (Common Language Infrastructure) için çalıştırılabilir dosyalar üretmez, ayrıca CLI olan her platformda çalışabilir. Bunun anlamı F# sadece Windows işletim sistemi ile sınırlı değildir. Ayrıca Linux ve Mac OS X üzerinde de çalışabilir.

#### F#’ın Başlıca Özellikleri;

- Type(Tip) güvenlidir. Yani C# ve Java gibi dillerde olduğu gibi F#’ta belleğe direk erişemez, ve bellek adreslerini kullanarak çeşitli işlemleri yapamaz. Belleğe sadece veri tiplerini kullanarak erişebilir.
- Thread güvenlidir. F# ile yazılan kod parçaları birden fazla thread tarafından kullanıldığında tutarlılıklarını kaybetmezler.
- Nesne yönelimli programlama dillerine göre daha az kod ile aynı işi yapabilir.
- Matematiksel ve İstatiksel yazılımlar için daha uygundur.

### 1.4 F# ile Yapılmış Projeler



#### FSharp.Charing

Google Charts görselleştirici kütüphanesinin F# ile kullanımını kolaylaştırmak için yazılmış bir API(Uygulama Arayüzü)’dir.

Ayrıntılı bilgiye aşağıdaki adresten ulaşabilirsiniz;

<https://github.com/fslaborg/FSharp.Charing>



#### ExcelFinancialFunctions

Yüksek seviyeli uygunluk testleriyle Excel’deki fonksiyonları F#’a implemente etmeye yarayan bir araçtır.

Ayrıntılı bilgiye aşağıdaki adresten ulaşabilirsiniz;

<http://fsprojects.github.io/ExcelFinancialFunctions/>



#### VimSpeak

VimSpeak, Vim adlı text editörünü verdiğiniz ses komutlarıyla kontrol etmeye yardımcı olan bir yapay zeka uygulamasıdır.

Ayrıntılı bilgiye aşağıdaki adresten ulaşabilirsiniz;

<https://github.com/AshleyF/VimSpeak>



#### Suave

Suave hafif ve kolay kolay bloke olmayan bir web sunucu yazılımıdır. Linux, OSX ve Windows’ta çalışabilir.

Ayrıntılı bilgiye aşağıdaki adresten ulaşabilirsiniz;

<http://suave.io>

## 1.5 F# İçin Kaynaklar ve Topluluk Siteleri

F# öğrenmeye ek kaynak olarak kendi resmi sitesini kullanabilirsiniz;

<http://fsharp.org/>

F# kullanıcılarının bilgi alışverişinde bulunduğu en popüler web sitesi;

<http://c4fsharp.net>

## 1.6 IDE(Uygulama Geliştirme Ortamı) Kurulumu ve Kullanımı

F# ile ilk programımızı yazmak için öncelikle programlama işmemini kolaylaştıracak birkaç araca ihtiyacınız olacak.



Programınızı F#'ın sağladığı kolaylık sayesinde herhangi bir Text Editörü hatta NotePad kullanarak bile yazabilirsiniz.

Ancak sizlere programcılık maceranızda kolaylık sağlaması açısından F# geliştirmede en çok kullanılan IDE(Uygulama Geliştirme Ortamı)'yi tanıtacağız. Bu araçlar işletim sisteminize göre değişiklik göstermektedir;

Eğer Windows kullanıyorsanız;

1. Seçenek: **Visual Studio Express**

Bu adresten indirebilirsiniz: <https://www.visualstudio.com>

2. Seçenek: **Xamarin Studio**

Bu adresten indirebilirsiniz: <http://xamarin.com/studio>

Linux kullanıyorsanız;

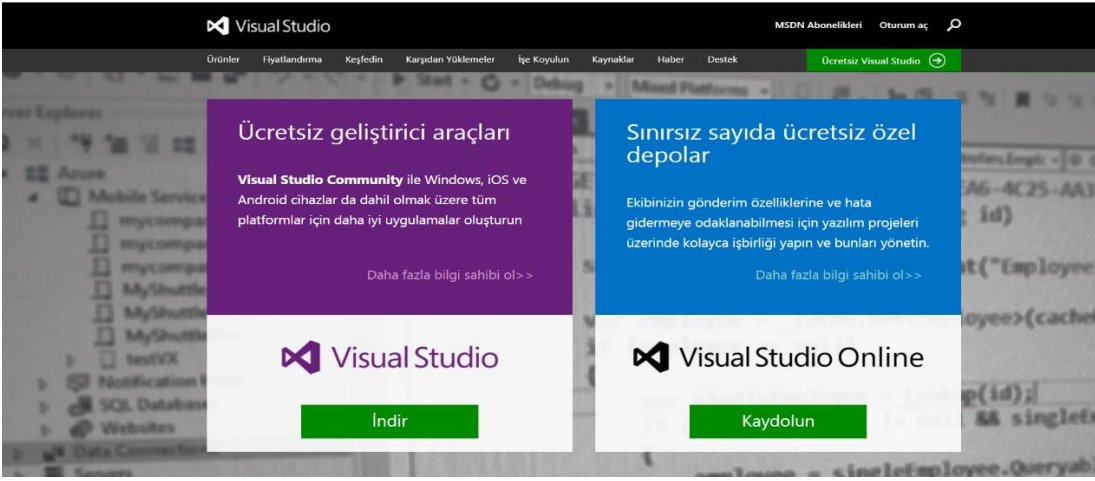
Buradaki adamları takip edebilirsiniz: <http://fsharp.org/use/linux/>

Mac kullanıyorsanız;

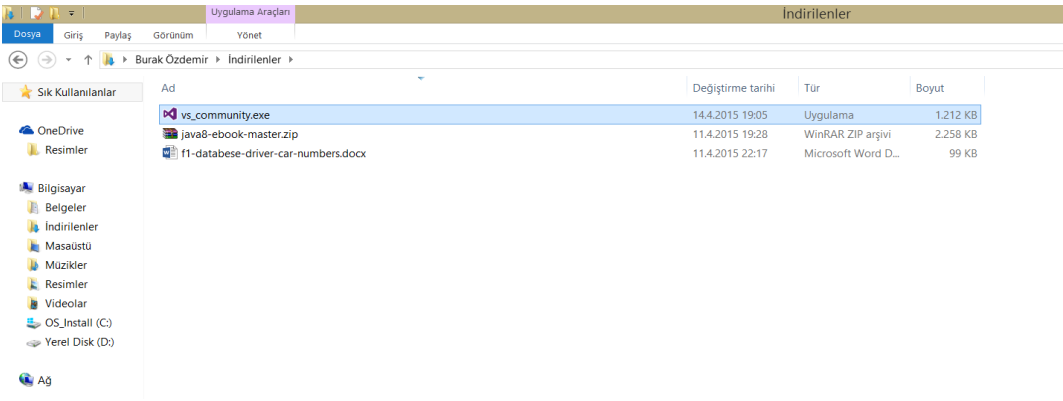
Buradaki adamları takip edebilirsiniz: <http://fsharp.org/use/linux/>

Biz kitabımızda Windows & Visual Studio üzerinden gideceğimizden ötürü size Visual Studio Express'in kurulumunu göstereceğiz.

Öncelikle yukarıdaki adrese giriyoruz karşınıza aşağıdaki gibi bir sayfa gelecek. Bu sayfayı gördükten sonra sol alt kısımda yer alan "İNDİR" butonuna tıklayıp kurulum dosyasını bilgisayarınıza indirin.



İndirme işlemi gerçekleştikten sonra inen dosyanın(vs\_community.exe) bulunduğu dizine gidip dosyayı çalıştırın.



Karşınızda gri bir pencere göreceksiniz. Bu pencerede bilgisayarımızda kurulacak diskte 9.16 Gb yer olması gerektiğini ve Lisans koşullarını kabul edeceğimiz bir de Visual Studio'nun geri bildirim kalite ve kullanımına yardımcı olacağımız bir gruba katılıp katılmadığımızı soran bir seçeneğimiz mevcut gerekli yerleri işaretliyoruz.



Yukarıdaki işlemi tamamladıktan sonra karşınıza Tools(Uygulama geliştirme araçları) seçeneklerini yapacağınız bir pencere gelecek istediklerinizi işaretleyip Install butonuna tıklıyoruz ve yüklememiz başlıyor.

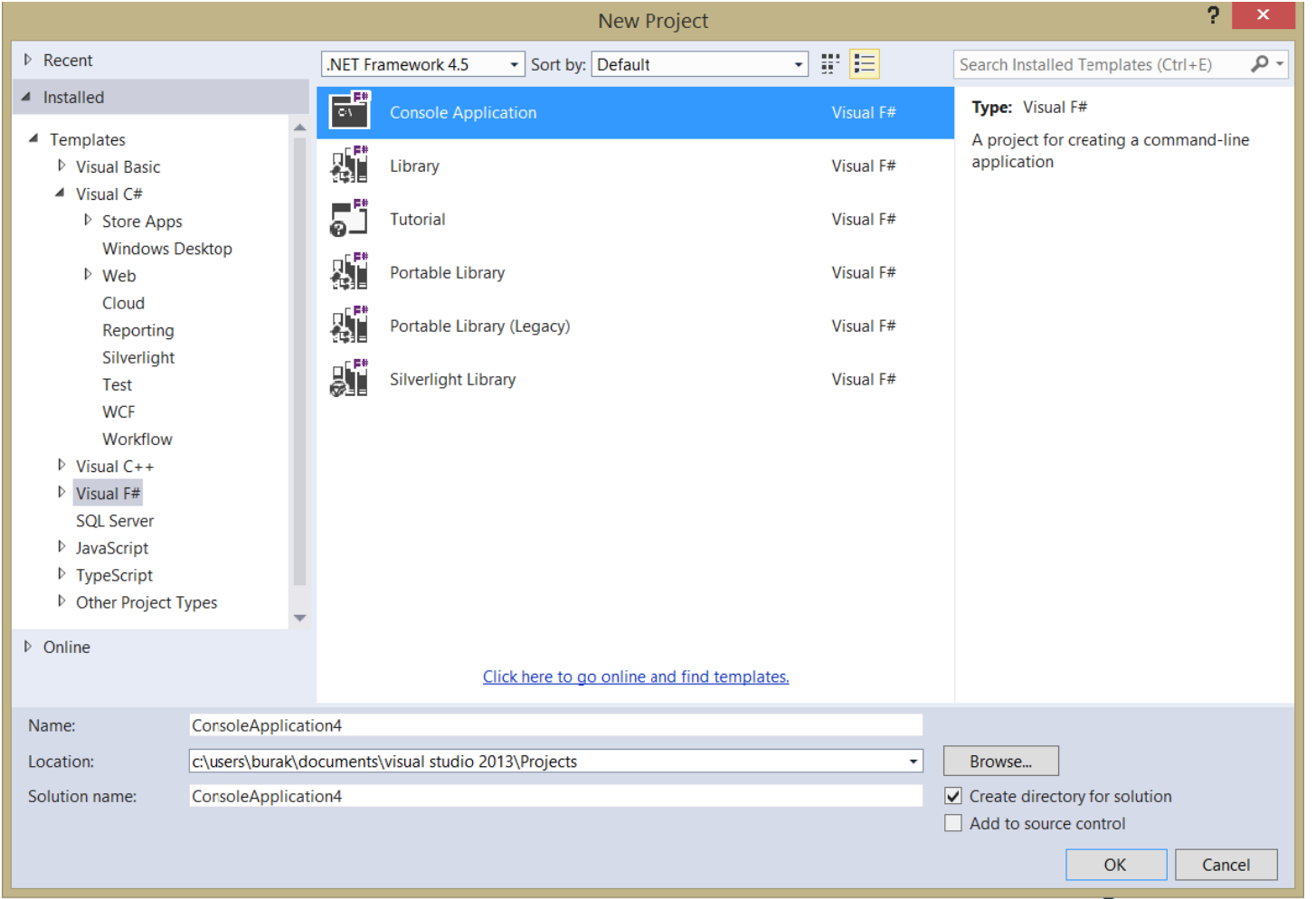


Yükleme işlemi bittikten sonra işlemin tamamlandığına dair ekran gelmekte ve bilgisayarımızı yeniden başlatmaktayız. Tüm işlemlerin ardından artık ilk programınızı yazmaya hazırsınız.

## 1.6 F# ile İlk Uygulama

Öncelikle Visual Studio'yu açalım. İlk projemizi oluşturmak için aşağıdaki adımları takip edin, File→New Project seçeneklerinin ardından açılan yeni pencerede sol taraftaki Visual F# sekmesini seçip ortada çıkan Console Application seçeneğini seçerek OK butonuna basınız.(Şekil 1.0)





Şekil 1.0

Projemizi oluşturduktan sonra şöyle bir ekran geliyor karşımıza;

```
Program.fs
1 // Learn more about F# at http://fsharp.net
2 // See the 'F# Tutorial' project for more help.
3
4 [<EntryPoint>]
5 let main argv =
6     printfn "%A" argv
7     0 // return an integer exit code
8
```

Eğer daha önce F# dışında herhangi bir programlama dili öğrendiyseniz her programın bir başlangıç metodu olduğunu bilirsiniz ve genellikle bu isim "Main" (ana) olarak bilinir. Main metodları ise program çalışmaya başladığı anda ilk ne yapılacağını temsil eder. Yani bir nevi programın başlangıç noktasıdır diyebiliriz.

F# programlama dilinde de bu gelenek sürdürülmüş ve bir main metodu yazılmış ancak metod isminin bir üst kod satırında EntryPoint (giriş noktası) ibaresi eklenmiş. Bunun anlamı program başladığında ilk bu kod bloğu çalışsın istenilmiş.

Burada kafamıza parlak bir fikir geliyor; madem EntryPoint özniteliği programın başlangıç noktasını ifade ediyorsa Main metodunun adını değiştirsek bile program yine oradan başlayabilir. Bu düşünce kesinlikle doğru programlarımız [<EntryPoint>] ifadesi nerede yazıldıysa oradan başlar.

Bu programlama dilinde birçok ifadenin başında "let" sözcüğü kullanılıyor ve eğer bu sözcük herhangi birşeyin kısaltılması değilse İngilizcedeki "izin ver" fiiline denk gelmektedir.

Neyse konuyu çok karıştırmadan önce her yeni programlama dilinde bir gelenek haline gelmiş olan "Merhaba Dünya" uygulamasını yazalım, hem bu sayede ilk kodlarımızı yazmaya başlamış oluruz.

```
Program.fs  X
1 // Learn more about F# at http://fsharp.net
2 // See the 'F# Tutorial' project for more help.
3
4 //EntryPoint=Programın Başlangıç Noktası
5 [<EntryPoint>]
6 let main argv = //Main isimli metottur.
7     printfn "Merhaba Dünya"//printfn metodu ekrana
8                             //bir metin yazdırır
9     0                       //Program 0 değerini
10                             //döndürür
11
```

Kodlarımızı incelediğimizde EntryPoint konusu haricinde, az önce bahsettiğimiz let sözcüğü ve printfn metodunu görüyoruz. Eğer C++ ile ilgilendiyseniz printfn metodunun ekrana bir metin yazdırmak olduğunu biliyorsunuz demektir. C# biliyorsanız bunun karşılığı Console.WriteLine() metodudur.

Metot sonunda ise 0 (sıfır) ibaresi yer almaktadır. Main metodunun başında ne void ne de string int gibi ilkel değişken tipleri yer almadığı için değer döndüren bir metot olup olmadığını ilk bakışta anlamak zor geliyor ancak şimdiden söyleyelim F# programlama dilinde değişken tipleri diye birşey yok.

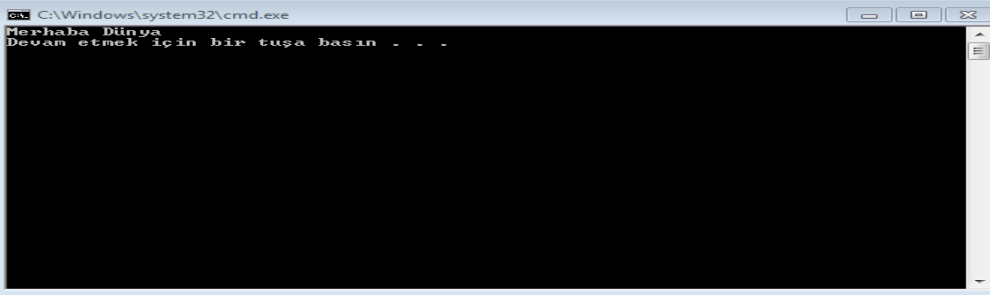
Diğer popüler programlama dillerinde( C#, Java, C++, .. gibi) veri tiplerini tanımlamak için kullanılan int, string, char, double gibi keywordlerde F# ta tercih edilmemiş. Tüm bu veri tipi tanımlayıcılarının yerine "let" keywordunu kullanabiliyoruz.

Eğer JavaScript biliyorsanız nasıl ki tek tip değişken tipi "var" (varians-değişken) varsa burada da ona benzer bir yapı var.

Ayrıca F# 'da dikkatinizi çektiyse ne süslü parantezler ne de satır sonunda noktalı virgül işareti yer almaktadır. Bu benzerlikle biraz da Visual Basic ve Python' u andırıyor.

Sözü yine uzatmadan programımızı çalıştıralım:

```
Program.fs  X
1 // Learn more about F# at http://fsharp.net
2 // See the 'F# Tutorial' project for more help.
3
4 //EntryPoint=Programın Başlangıç Noktası
5 [<EntryPoint>]
6 let main argv = //Main isimli metottur.
7     printfn "Merhaba Dünya"//printfn metodu ekrana
8                             //bir metin yazdırır
9     0                       //Program 0 değerini
10                             //döndürür
11
```



Görüldüğü gibi programımız gayet düzgün bir şekilde çalışıyor ve konsol ekranına verdiğimiz komut olan "Merhaba Dünya" ifadesini yazdırıyor.

## F# İLE FONKSİYONEL PROGRAMLAMAYA GİRİŞ

### 2.1 Kullanıcı ile Etkileşim I/O

F# bir .NET dili olduğundan .NET'in kullanıcı etkileşiminde kullandığı tüm metotları F# ile de kullanabiliriz.

#### Input(Veri Girişi)

F#'ta veri girişi işlemi C#'a benzer şekilde yapılmaktadır.

```
//.NET fonksiyonlarından faydalanmak için open System ifadesini ekliyoruz
open System
```

```
let YetiskinUcreti = 10
```

```
let CocukUcreti = 7
```

```
Console.WriteLine("Yetişkin bilet adedini giriniz:")
```

```
let YetiskinBiletSayisi = Int32.Parse(Console.ReadLine())
```

```
Console.WriteLine("Cocuk bilet adedini giriniz:")
```

```
let CocukBiletSayisi = Int32.Parse(Console.ReadLine())
```

```
let ToplamMaliyet = YetiskinUcreti * YetiskinBiletSayisi + CocukUcreti*CocukBiletSayisi
```

```
Console.WriteLine("Toplam bilet ücreti={0}",ToplamMaliyet)
```

C#'tan farklı olarak metotların sonuna “;” işareti eklemediğimize dikkat edelim.

Yeri gelmişken şunuda belirtmekte fayda var: F#'ta kod bloklarını ayırmak için “{” ve “}” işareti kullanılmaz. Derleyici kod girintilerinden yararlanarak işlemleri kendi tayin eder.

### 2.2 Veri Tipleri

F# bir .NET ailesi dili. Bu yüzden, temel veri tipleri ve referans'ları C# ile çok benziyor.

F# bir strongly typed dildir. Bu nedenle tür dönüşümlerinde oluşan bir hata compile time(programın derlenme zamanı)'da meydana gelir. Bu sayede bu tür hatalar, yazılım geliştirmenin erken evrelerinde kontrol edilip müdahale edilebilir.

C# ile F# arasındaki farklardan biri de; F# örnekleri açıkça tip belirtilmesine ihtiyaç duymaz, bu yüzden genellikle atanan değerden tipi anlayabilir. C# geliştiricileri için bu durum var keyword'ünü anımsatır. Fakat let ile var anahtar sözcükleri arasında bazı temel farklılıklar da mevcuttur.

#### Boolean Tipler

Tıpkı diğer dillerde olduğu gibi boolean tipler sadece iki değere sahip olabilir; 1(true) veya 0(false). Bir örnek verecek olursak;

```
let giris_Yapti = false
```

```
//Giriş yapıp yapmadığını kontrol etmeye yarayan bir blok. Giriş yapıldıysa 1(true) yapılmadıysa 0(false) değeri atanacak.
```

Ayrıca boolean tipleri karşılaştırmada kullanılan birkaç operatör vardır. Bunlar;

Operatör	İşlev
not	Olumsuzluk
	Veya
&&	Ve

### Nümerik Tipler <sup>1</sup>

F#’taki nümerik tipler .NET ortamında kullanılan nümerik tiplerle aynıdır. Bunlar;

Tip Kısaltması	.Net Tipi	Değer Aralığı
byte	System.Byte	0-255
sbyte, int8	System.SByte	-128 - 127
int16	System.Int16	-32,768 32,767
uint16	System.UInt16	0 - 65,535
İnt, int32	System.Int32	$-2^{31}$ to $2^{31}-1$
UInt, int32	System.UInt32	0 to $2^{32}-1$
decimal	System.Decimal	$-2^{96}-1$ to $2^{96}-1$
Float, double	System.Double	0 to $2^{64}-1$

### Nümerik Tipleri Birbirine Dönüştürme

Modern programlamada birçok kez değişkenlerde tür dönüşümüne ihtiyaç duyulur. Örneğin float türündeki sayılarla (5.5 veya 2.98 gibi) integer türündeki sayılar arasında ( 1 veya 3 gibi) matematiksel işlem yapmamız gerektiğinde tür dönüşümü yapmamız gerekir. Aslında bahsettiğimiz tam olarak tür dönüşümü değildir, sadece bir değişkenin değişik türdeki hâlinin başka bir değişkene atanmasıdır. Örnek verecek olursak;

```
let martAyıSıcakliklar = [ 33.0; 30.0; 33.0; 38.0; 36.0; 31.0; 35.0;
                          42.0; 53.0; 65.0; 59.0; 42.0; 31.0; 41.0;
                          49.0; 45.0; 37.0; 42.0; 40.0; 32.0; 33.0;
                          42.0; 48.0; 36.0; 34.0; 38.0; 41.0; 46.0;
                          54.0; 57.0; 59.0 ]
let toplamMartAyıSıcakliklar = List.sum martAyıSıcakliklar
let ortalama = toplamMartAyıSıcakliklar / float martAyıSıcakliklar.Length
```

//Yukarıda mart ayındaki tüm sıcaklıkları bir dizide tuttuk ardından ortalamayı hesapladık. float martAyıSıcakliklar.Length işlemini yapmamızın nedeni toplamMartAyıSıcakliklar değişkeninin float olması martAyıSıcakliklar.Length değişkeninin ise integer olmasıdır.

### Kendi Veri Tipimizi Tanımlamak <sup>2</sup>

Kendi tipimizi tanımlamak yani bir class(sınıf) oluşturmak istiyoruz. Bunu eğer C#’a aşınaysak şu şekilde yaptığımızı hatırlayacaksınız;

// C#

```

using System;
public class Person
{
    public Person(int id, string name, int age)
    {
        Id = id;
        Name = name;
        Age = age;
    }
    public int Id { get; private set; }
    public string Name { get; private set; }
    public int Age { get; private set; }
}

```

Bu işlemi F#’ta şu şekilde gerçekleştirebiliriz;

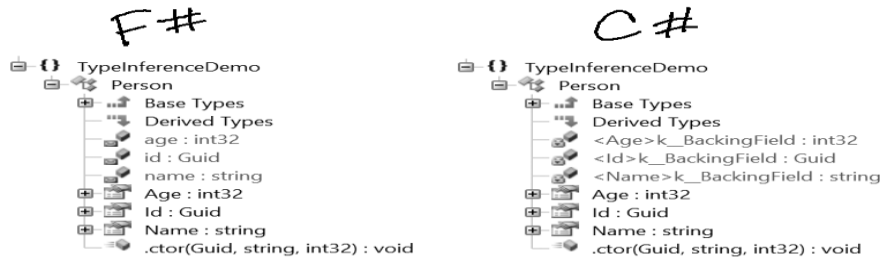
//F#

```

type Person (id : int, name : string, age : int) =
member x.Id = id
member x.Name = name
member x.Age = age

```

Gördüğünüz gibi F#’ta bir tip tanımlama C#’a göre ne kadar kolay ☺ Bu iki dilin derlendikten sonra nasıl görüldüğüne göz atalım;



Pek bir fark yok değil mi ?

## Enumlar

Enum(Enumaration) tanımlamaları programlama dillerinde belirli değerlere karşılık gelen sabit değerlerin temsili için kullanılırlar, haftanın günleri, yönler vs gibi. Geliştiriciye sağladığı faydalar ise kod okunabilirliğini arttırması, muhtemel değer kümesinin daraltılması ve tip güvenliğini sağlaması dolayısıyla hata payını en aza indirmesidir.

Geliştirme yaparken bazen program akışını yönlendiren belirli durumlar oluşur, biz de bu durumları saptar, sayısal değerler ile ilişkilendirip kodu şekillendiririz.

F#’ta Enumların temel syntax’ı aşağıdaki gibidir;

```

type enum-adi =
| deger1 = sayisalkarsiligi1
| deger2 = sayisalkarsiligi2
-- end --

```

---

Bir örnekle pekiştirecek olursak;

---

```
type HaftaninGunleri =  
| Pazartesi = 0  
| Sali = 1  
| Carsamba = 2  
| Persembe = 3  
| Cuma = 4  
| Cumartesi = 5  
| Pazar = 6
```

---

Peki enum tipimizi tanımladık bunu nasıl kod bloğumuzda kullanacağız diye düşünüyorsanız, çok basit;

```
let tatil = HaftaninGunleri.Pazar
```

```
//tatil adında bir değişken oluşturup buna daha önceden oluşturduğumuz enum tipimizin Pazar değerini atadık. Buda sayısal olarak 6 ya denk geliyor.
```

## 2.3 Koşul Yapıları

Programlarda sık kullanılan bir komut kalıplarıdır. **If** eğer, **then** öyleyse, **else** öyle değilse anlamına gelir. İskeleti şu şekildedir;

```
if boolean-ifade then yapılacak-işlem [ else diğer-işlem ]
```

Aşağıdaki örnek koşul yapılarının F# dilinde nasıl gerçekleştirildiğini göstermektedir;

```
let test x y =  
    if x = y then "a eşittir"  
    elif x < y then "den küçüktür"  
    else "dan büyüktür"  
  
printfn "%d %s %d." 10 20 (test 10 20)  
  
printfn "Adın ne? "  
let ad = System.Console.ReadLine()  
  
printfn "Yaşın kaç? "  
let yasString = System.Console.ReadLine()  
let yas = System.Int32.Parse(yasString)  
  
if yas < 10  
then printfn "Sen sadece %d yaşındasın ve F# öğreniyorsun. Vay canına!" yas
```

Ekran çıktısı:

```
10 20 den küçüktür.  
Sen sadece 9 yaşındasın ve F# öğreniyorsun. Vay canına!
```

## 2.4 Döngüler<sup>3</sup>

F#’ta iki adet döngü bulunur: **while** ve **for**. İlk olarak **for** döngüsü ile başlayalım.

### For...to Döngüsü

Programlama dillerinde bir değişken üzerinde verilen iki değer arasında ve verilen aralıklarla işlem yapmamıza yarayan bir yapıdır. **For** döngüsü programlarımızın birden fazla sayıda çalışmasını sağlar. F# dilinde iki tip **for** döngüsü vardır.

İlk **for** döngüsünün temel yapısı şu şekildedir:

```
for degisken = [ baslangic | bitis ] do
    body-ifadeleri
```

Aşağıdaki örnek **for** döngüsünün F# dilinde nasıl gerçekleştirildiğini göstermektedir;

```
// 1 den 10'a kadar olan sayıları ekrana yazdıran bir for döngüsü
```

```
let fonksiyon1() =
    for i = 1 to 10 do
        printf "%d " i
    printfn ""
```

```
// 1 den 10'a kadar olan sayıları tersten yazdıran bir for döngüsü
```

```
let fonksiyon2() =
    for i = 10 downto 1 do
        printf "%d " i
    printfn ""
```

```
fonksiyon1()
fonksiyon2()
```

Ekran çıktısı:

```
1 2 3 4 5 6 7 8 9 10
10 9 8 7 6 5 4 3 2 1
```

### For...in Döngüsü

**for...to** gibi döngü komutudur ama çalışma prensibi daha farklıdır. **For...in** döngüsü herhangi bir dizi, liste ve enumerable veriler üzerinde gezinmemize olanak sağlar. Syntax'ı şu şekildedir:

```
for degisken in enumerable-veriler do
    body-ifadeleri
```

Aşağıdaki örnek **for...in** döngüsünün F# dilinde nasıl gerçekleştirildiğini göstermektedir;

```
// Liste üzerinde dönme
```

```
let liste1 = [ 1; 5; 100; 450; 788 ]
for i in liste1 do
    printfn "%d" i
```

Ekran çıktısı:

```
1
5
100
450
788
```

### While Döngüsü

*While* keywordu, Türkçede ‘... iken, ... sırasında’ gibi anlamlara gelir. F# programlama dilinde *while* bir döngüdür. *For* döngüsünde olduğu gibi *while* döngüsü sayesinde de programlarımızın sürekli olarak çalışmasını sağlayabiliriz.

Basit bir *while* döngüsünün temeli şu şekildedir:

```
while boolean-ifade do
    body-ifadeleri
```

Örnek:

```
// While döngüsü
let mutable i = 0
while i < 5 do
    i <- i + 1
    printfn "i = %d" i
```

Ekran Çıktısı:

```
i = 1
i = 2
i = 3
i = 4
i = 5
```

## 2.5 Fonksiyonlar

F#’ta *let* keywordu yardımıyla kolayca fonksiyon tanımı yapılabilir

Burada gönderilen sayının negatifini döndüren bir fonksiyon;

```
let negatifYap x = x * -1
```

Bu fonksiyona parametre göndermek ise şu şekilde olur;  
negatifYap 12 veya negatifYap(12) şeklinde parametre gönderebiliriz.

Sayının karesini alan bir fonksiyon;

```
let kareAl x = x * x
```

Gönderilen sayıyı ekrana printfn fonksiyonu yardımıyla basan bir fonksiyon;

```
let yazdir x = printfn "Gonderilen sayi: %d" x
```

Fonksiyonları bir arada kullanmak;

```
let birlestir x =
    yazdir (negatifYap (kareAl x))
```

### Pipe ve Composition Operatörleri

Pipe operatörü (*|>*) fonksiyonları ve argümanları biraraya getirmede kullanılırlar. Örnek vericek olursak;

```
let birlestir x =
    x |> kareAl |> negatifYap |> yazdir
```



Yukarıda örnek verdiğimiz chaining işleminde önceki bölümde göstermiş olduğumuz fonksiyonları zincirlemiş olduk. Gönderilen x sayısının öncelikle kareAl yardımıyla karesini alacak ardından karesi alındıktan sayı negatifYap yardımıyla negatif hale getirilip en son yazdir fonksiyonumuz yardımıyla ekrana bastırılacaktır.

Aynı işlemi Composition operatörü(>) ile de yapabiliriz. Burada Pipe operatöründen farklı olarak parametre göndermemize gerek yoktur.

let birlestir =

```
kareAl >> negatifYap >> yazdir
```

### Recursive(Öz Yinelemeli) Fonksiyonlar

Kısaca, içerisinde kendisini çağıran fonksiyonlara öz yinelemeli fonksiyonlar denir. Belirli bir şarta kadar aynı işlemleri tekrarlayan yapılarda öz yinelemeli yordamlar kullanılabilir. Bu fonksiyonlarda yordamı her defasında tekrar tekrar çağırdığımızda yordamın yeni bir kopyası oluşturulmaz. Bu nedenle kullandığı değişkenlerin değerini bulabilmesi için bunları yığına atar ve hatırlamak üzere geri çağırabilir. Buda yığının extra şişmesi demektir. Fonksiyon kendini kadar çok çağırırsa yığında o kadar şişer.

Bu tip fonksiyonların kullanım yerleri daha çok arama algoritmaları ve kendini tekrar eden matematiksel işlemlerdir. Bir diğer deyişle öz yinelemeli fonksiyonlar, fonksiyonel programlamanın olmazsa olmazıdır.

Öz yinelemeli fonksiyonları anlatırken klasik örnek haline gelen faktöriyel hesaplamaya birde F# gözüyle bakalım;

```
let rec faktoriyel x =  
    if x < 1 then 1  
    else x * faktoriyel (x - 1)
```

Yukarıdaki faktoriyel metodumuz gelen sayı(x) 0 dan büyük olma şartıyla sürekli kendini çağırıyor. 0 değeri geldiğinde ise kendini çağırmaı bırakıyor.



Öz yinelemeli fonksiyon tanımlarken normal fonksiyondan farklı olarak let keywordunun ardından bir de rec(recursive) keywordunun geldiğine dikkat etmelisiniz.

### Pattern(Desen) Eşleştirme

Match keywordu yardımıyla gerçekleştirilir.

Daha önce anlattığımız öz yinelemeli fonksiyonlar üzerinden örneklendirecek olursak;

```
let rec fib n =  
    match n with  
    | 0 -> 0  
    | 1 -> 1  
    | _ -> fib (n - 1) + fib (n - 2)
```

Yukarıda verilen fib metodu pattern eşleştirme ve öz yinelemeli fonksiyon yardımıyla fibonacci dizisi oluşturmaktadır. Fib metoduna gönderilen n sayısı 0 ile eşleşirse 0, 1 ile eşleşirse 1 eğer her ikisi ile eşleşmez ise fib(n-1) + fib(n-2) metodlarını döndürecektir.

## 2.6 Koleksiyonlar

### Listeler

Liste içinde aynı tip verileri birbiri ardına saklayan bir veri tipidir. Dizilerden farklı olarak boyutu önceden belli değildir.

Listelerin kullanımı;

```
// Liste elemanlarını ayırmada “;” işareti kullanılır...
let list1 = [ "a"; "b" ]

// :: yardımıyla “c” harfini list1 listesinin başına ekliyoruz...
let list2 = "c" :: list1

// @ yardımıyla listeleri birleştiriyoruz...
let list3 = list1 @ list2

// :: yardımıyla listenin tüm elemanları üzerinde pattern eşleştirme işlemi yapıyoruz...
let rec sum list =
match list with
| [] -> 0
| x :: xs -> x + sum xs
```

## Diziler

Diziler verileri sıralı bir şekilde tutmak için kullanılırlar. Listelere göre dez avantajları sabit boyutlu olmalarıdır.

Dizilerin kullanımı;

```
// Listelere tanımlarken kullandığımız [] kalıbından farklı olarak [| |] kalıbıyla tanımlanırlar.
let array1 = [| "a"; "b" |]

// Dizinin ilk elemanına şu şekilde erişebiliriz.
let first = array1.[0]
```

## 2.7 Tuples(Demetler) ve Records(Kayıtlar)

### Tuples(Demetler)

Demetler bize tek bir tip içerisinde birden fazla farklı tipteki verileri saklama olanağı sağlar.

Kullanım şekli;

```
// Tanımlanma şekli
let x = (1, "Hello")

// 3 adet veri tutmak.
let y = ("one", "two", "three")
```

### Records(Kayıtlar)

Records bize numaralandırılmış verileri tutmada kolaylık sağlar. Örneklendirecek olursak;

```
// Records tipi tanımlamak
type Person = { Name : string; Age : int }

// Yukarıda tipten bir nesne oluşturmak
let paul = { Name = "Paul"; Age = 28 }

// Oluşturduğumuz nesneyi güncellemek.
let paulsTwin = { paul with Name = "Jim" }
```

## 2.8 Hata Yakalama

F#’ta hatalar projemizin derleme yada çalışma aşamasında gerçekleşir. Bunlar yazılımcının hataları olabildiği gibi kullanıcının hataları da olabilir. Örneğin kullanıcıdan sayısal bir değer istenmişken kullanıcının buraya metinsel bir ifade yazması ya da kullanıcı kaynaklı hataya neden olacaktır. Bu yüzden yazılımcı olası bu ihtimalleri düşünerek önlemleri almalı ve programın kararlılığını sağlamalıdır. Aynı zamanda yazılımcının syntax yani yazım hataları yapması yada mantıksal hatalar yapması söz konusudur. Hataların tespit edilmesinde en çok zorlanılan yer ise mantıksal hatalardır. Bu konu ile; olası hataları, bunların tespit edilmesi ve bu hatalara karşı önlemlerin alınması durumlarını inceliyor olacağız.

```
let bolmeHata x y =  
    if y = 0 then  
        failwith "Bölen 0 olamaz"  
    else x / y
```

Yukarıdaki örnekte bölen(y) 0 olduğu zaman program “Divisor cannot be zero” hatası verecektir. Hata yakalama işlemi try/with ifadeleriyle gerçekleşir. Try/with ifadeleri daha temiz kod yazma olanağı sağlar.

```
let bolme x y =  
    try  
        Some (x / y)  
    with :? System.DivideByZeroException ->  
        printfn "0 a bölme hatası"
```

Önceden tanımlı hataları kullanabileceğiniz gibi kendiniz de bir hata tanımlayabilirsiniz.

```
exception InnerError of string  
exception OuterError of string  
  
let hataKontrol x y =  
    try  
        try  
            if x = y then raise (InnerError("ic"))  
            else raise (OuterError("dis"))  
            with InnerError(str) ->  
                printfn "%s hatası" str  
        finally  
            printfn "Bu satir her zaman yazdirilacak."
```

## 2.9 F# ile Nesne Yönelimli Programlama

Nesne yönelimli programlama (Object Oriented Programming) yazılım dünyasında karmaşıklığı ya da boyutu artan yazılımların kolayca ve kısa sürede geliştirilebilmesi için özellikle tüm yazılım projelerinde kullanılmaktadır. Tüm modern programlama dilleri tarafından desteklenmez. Nesneye yönelik programlama da 3 temel yapı üzerine durulur. Bunlar :

- Veri Soyutlama
- Kalıtım
- Çok Biçimlilik

Öncelikle sınıf (class) ve nesne (object) nedir bunlara değinelim.

Sınıf (Class) , bir bütünü oluşturacak şekilde ortak özelliklere sahip olan üyelerin bir araya getirdikleri bütün olarak tanımlanabilir. Bunları daha sonra örneklendireceğiz.

Nesne (Object) ise, etrafımızda gördüğümüz ve tabiri eğer uygunsa “şey” olarak tanımladığımız varlıktır. Örnek verirsek; kalem, kağıt, defter, ben, onlar, Ahmet vs bunların hepsi birer nesnedir diyebiliriz. Nesnelerin iki temel bileşeni bulunur. Bunlar Özellik (Property) ve Davranış (Behavior)’dır. Nesne tanımlandığı sınıfın özelliklerini taşımaktadır.

Buradan anladığımız gibi, sınıf nesneyi tanımlayan bir veri türüdür denilebilir. Nesne ise bir sınıftan türettiğimiz ve onun özelliklerini taşıyan bir modeldir ve daha sonra tekrar tekrar kullanılacak olan parçalardır.

OOP, yazılım geliştirme sürecinde son derece etkili bir yöntemdir ve günümüz projelerinde vazgeçilmez olmayı sürdürmektedir. Bir yazılım geliştirmenin 3 temel aşaması bulunmaktadır, bunlar Tasarlama, Geliştirme ve Test Etme

olarak tanımlanır. İlk aşamanın iyi olması sonucu geliştirme ve test etme süreçleri de hızlı ve doğru-kolay bir şekilde daha rahat gelişir, bu durum da tasarımda OOP'nin kullanılması sonucu gerçekleşebilir.

Aşağıdaki örnekte sizlere fonksiyonel tabanlı bir dil olan F#'ta nesne yönelimli programlama mantığını izah etmeye çalışacağız. Yorum satırında (1) ile gösterilen yerler local let tanımlamaları, (2) properties(nitelikler), (3) metotlar, ve (4) ise statik üyeleri belirtmektedir.

```
type Vector(x : float, y : float) =  
    let mag = sqrt(x * x + y * y) // (1)  
    member this.X = x // (2)  
    member this.Y = y  
    member this.Mag = mag  
    member this.Scale(s) = // (3)  
        Vector(x * s, y * s)  
    static member (+) (a : Vector, b : Vector) = // (4)  
        Vector(a.X + b.X, a.Y + b.Y)
```

## Kalıtım

F#'ta kalıtım yaparken, C# dilinde yaptığımız gibi ":" işareti kullanmıyoruz. Türetmek istediğimiz sınıf içerisinde inherit keywordu yardımıyla base classı belirtiyoruz.

```
type Animal() =  
    member __.Rest() = ()  
  
type Dog() =  
    inherit Animal()  
    member __.Run() =  
        base.Rest()  
  
//Casting işlemi(türler arası dönüşüm) :> operatörüyle gerçekleşir.  
let dog = Dog()  
let animal = dog :> Animal
```

## BÖLÜM 3

### ÖRNEK UYGULAMA

Şimdiye kadar öğrendiğimiz bilgilerle herhangi bir F# uygulaması geliştirebilecek seviyeye geldiğimizi düşünüyorum. Bu bölümde geliştireceğimiz örnek uygulama F# serüvenimizde bize avantaj sağlaması açısından F#'ın en çok tercih edildiği konu olan istatistik üzerine olacaktır.

### 3.1 Uygulama Tanımı

Uygulamamız F#'ın kütüphanelerinden ve işlevselliğinden yararlanarak, istatistiksel verilerin klasik formlardan soyutlanması ve bu verilerin farklı ve yaratıcı bir şekilde grafiksel olarak sunulmasına yardımcı olacaktır.

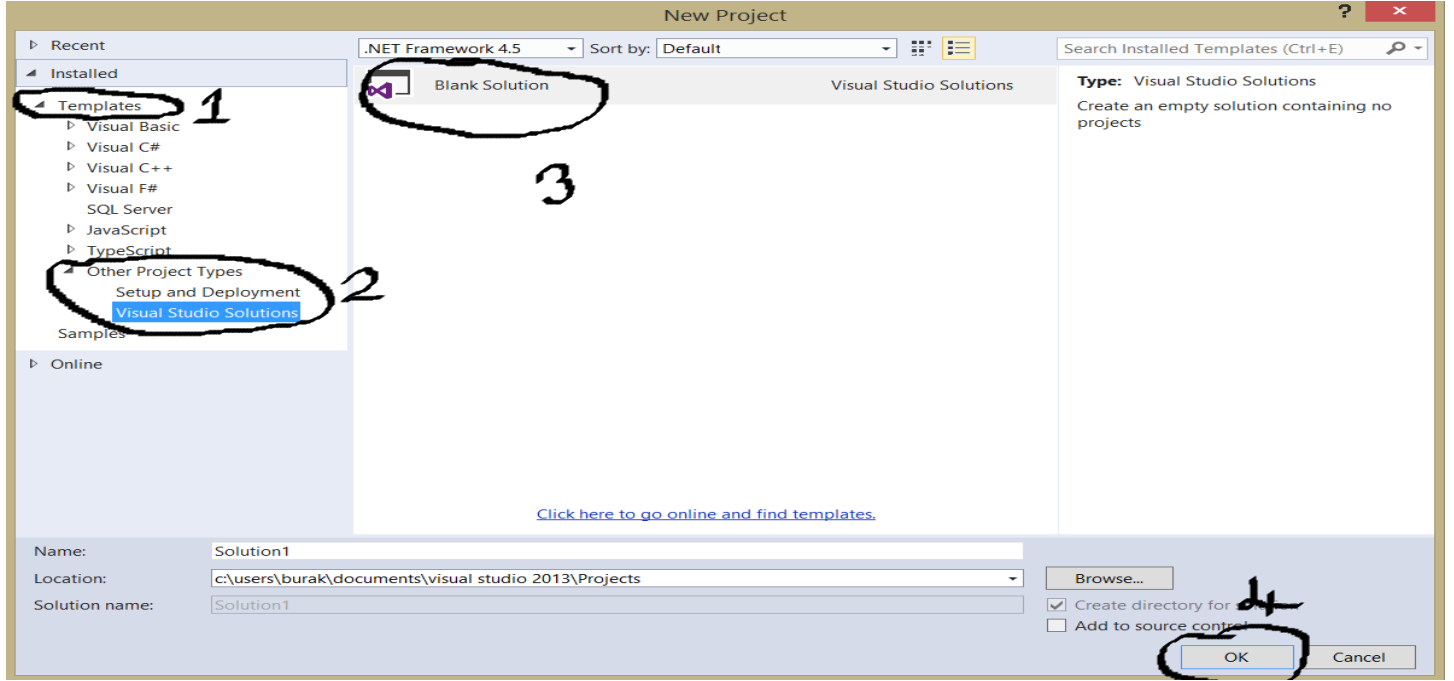
Uygulamanın amaçlarından biri de normalde klasik formatta sunulan karmaşık ve kafa karıştıran verileri, kolay algılanabilir grafik arayüzler ile rahatça anlaşılır hale getirmektir.

İş ve finans, yönetim ve dijital medya gibi meslek gruplarının özellikle nimetlerinden bol bol faydalandığı veri görselleştirme sanatının örneklerine uygulamamız yardımıyla değineceğiz.

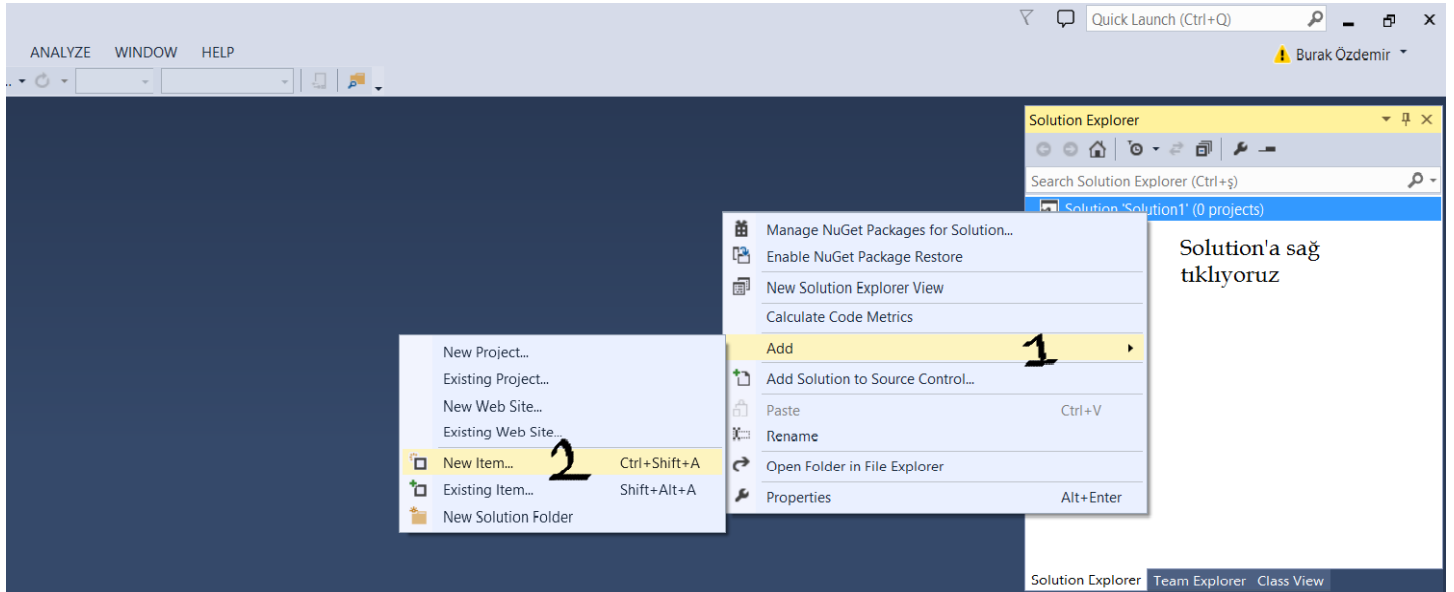
### 3.2 Geliştirme Aşaması

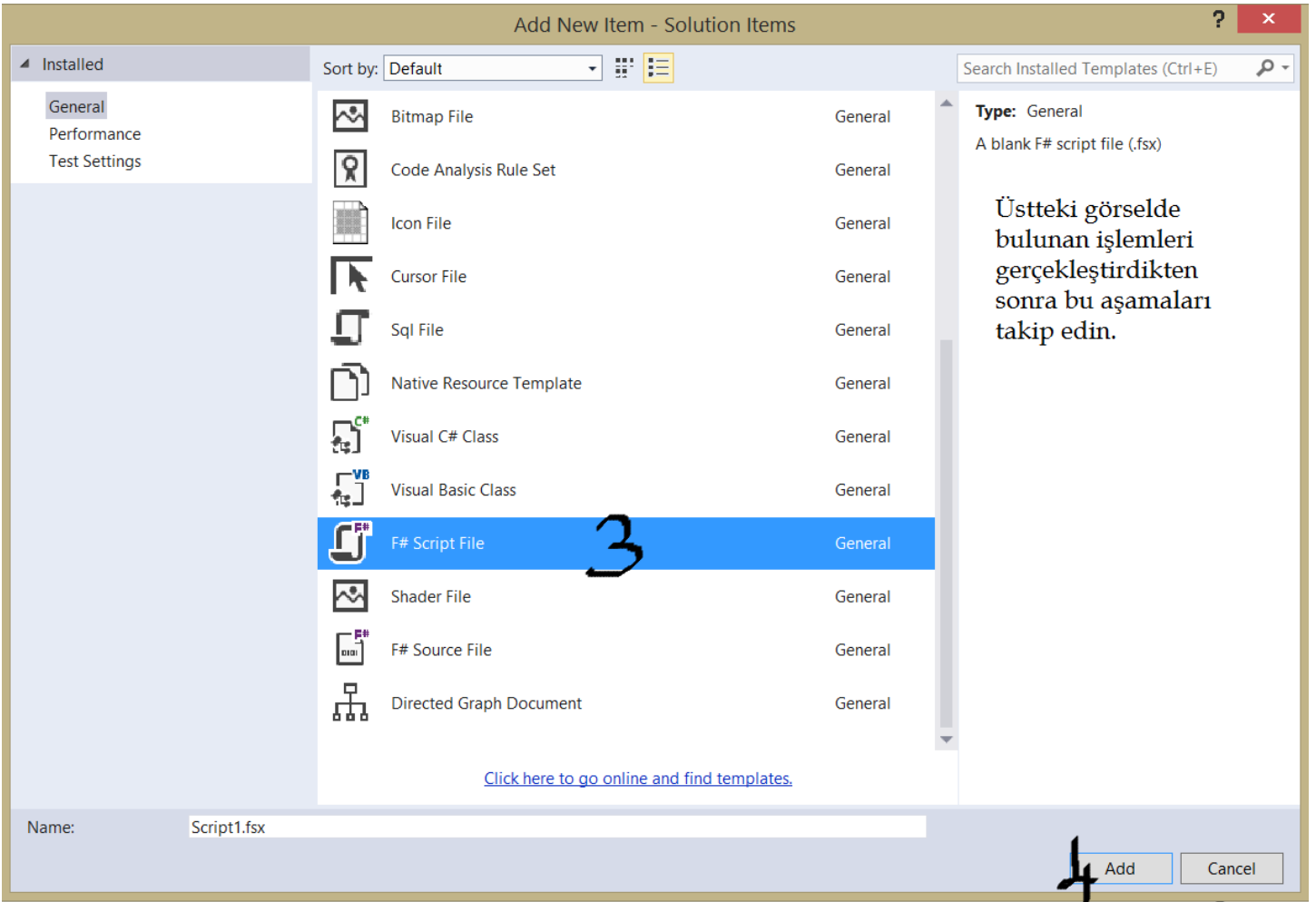
Uygulamamızın çalışması için F#’ın istatistik alanındaki en büyük kütüphanelerinden olan Fsharp Charting adındaki kütüphaneyi yüklememiz gerekmektedir.

Kütüphaneyi yüklemeden önce Visual Studio’yu açalım. Visual Studio’yu açtıktan sonra File menüsüne gelip New’e onun ardından da New Project’e tıklayın. Açılan pencerede Installed kısmında bulunan Other Project Types bölümüne gelip Visual Studio Solutions’u seçip ardından Blank Project’e tıkladıktan sonra “OK” butonuyla yeni solutionumuzu oluşturalım.



Solution’u oluşturduktan sonra bu solutiona F# script dosyası eklememiz gerekmektedir. Bu script dosyasını aşağıdaki adımları takip ederek ekleyebiliriz.



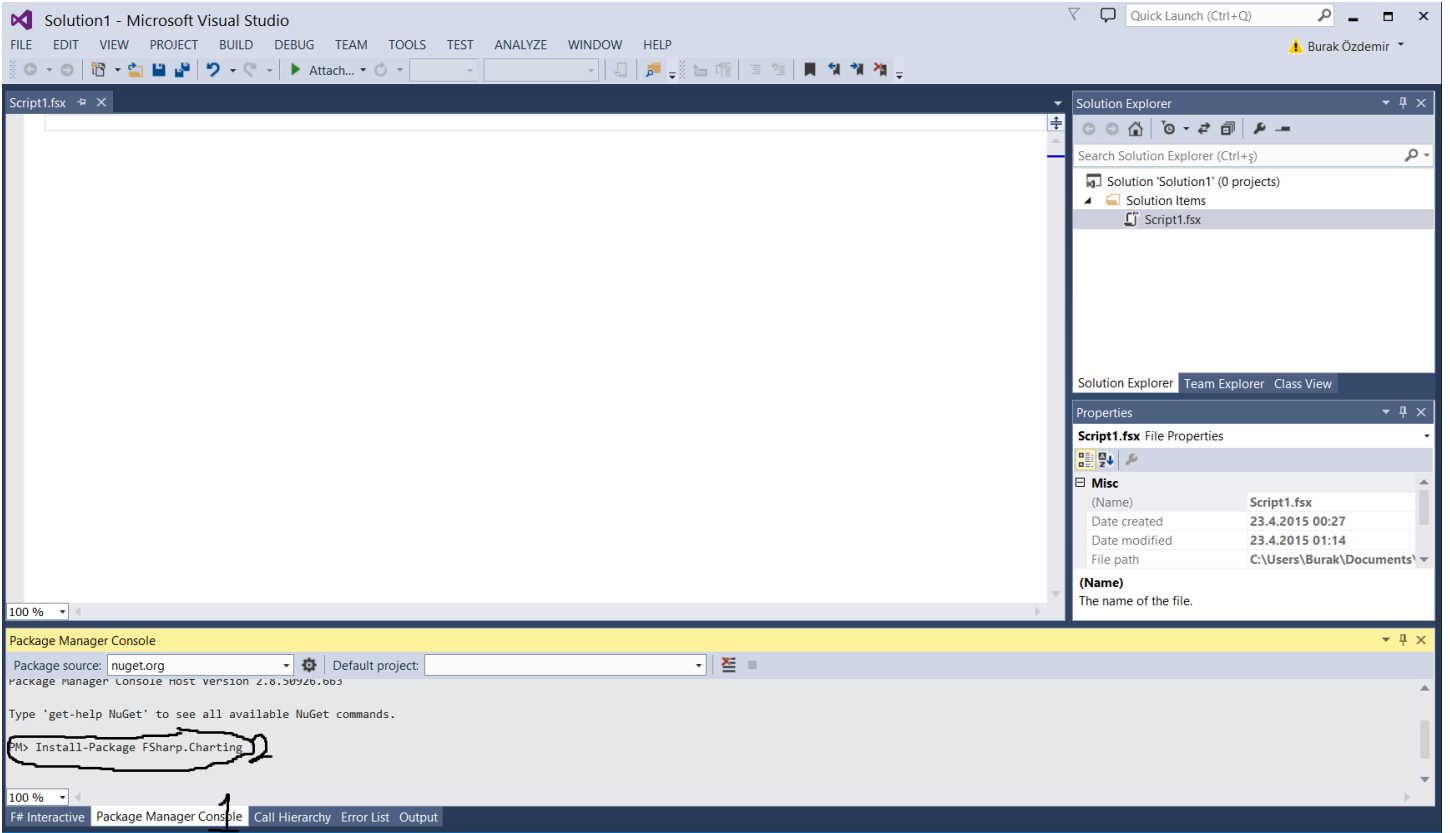


Uygulamamız için gerekli olan ilk aşamayı gerçekleştirdik. Sıra geldi kütüphanemizi NuGet paket yöneticisi yardımıyla indirmeye.

Solution'u oluşturduktan sonra NuGet paket yöneticisi aracılığıyla aşağıdaki komutu veriyoruz ve kütüphanemizi solutionumuza ekliyoruz.

```
PM> Install-Package FSharp.Charting
```

NuGet paket yöneticisi aracılığıyla komut verme işlemi şu şekilde yapılmaktadır;



FSharp Charting kütüphanesini uygulamamızda kullanabilmemiz için öncelikle script dosyamızda kütüphaneyi referans göstermemiz gerekmektedir. Bu işlemi şu şekilde yapabiliriz:

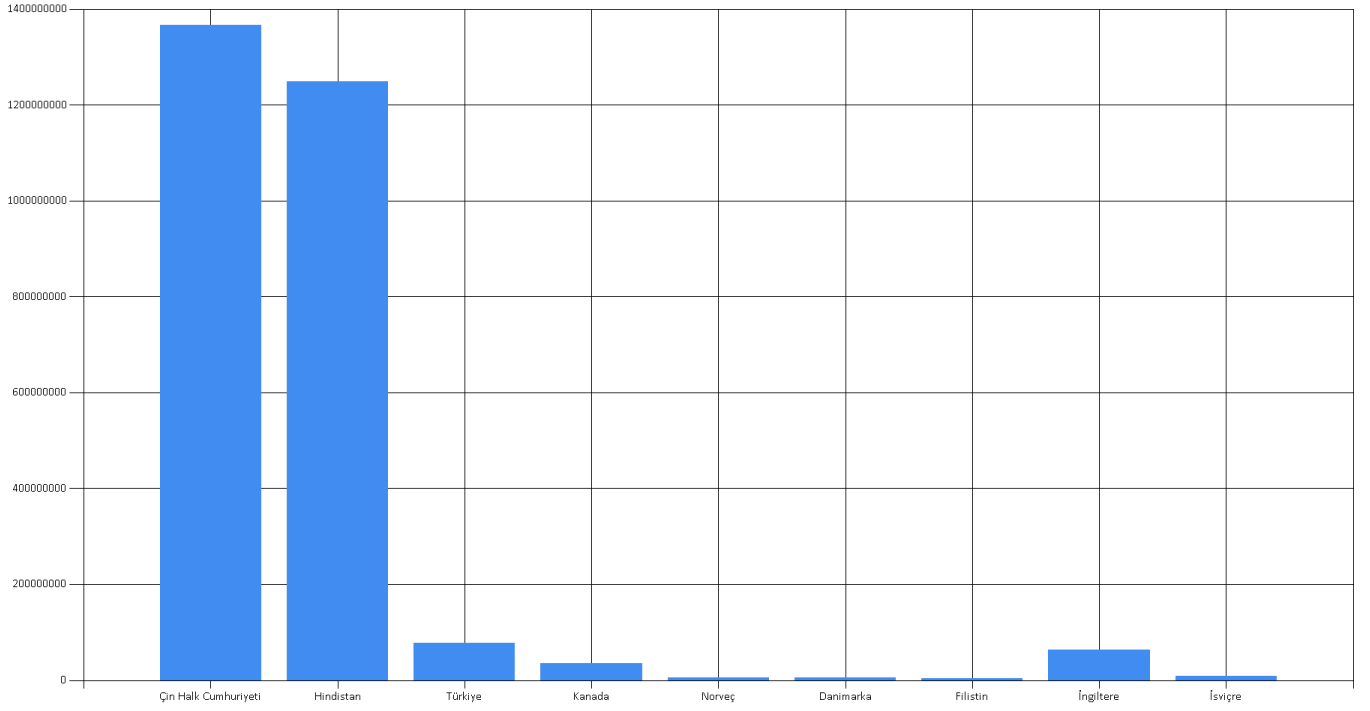
```
#load "packages/FSharp.Charting.0.90.10/FSharp.Charting.fsx"
```

#### Ülke Nüfuslarının Grafiksel Hale Getirilmesi <sup>4</sup>

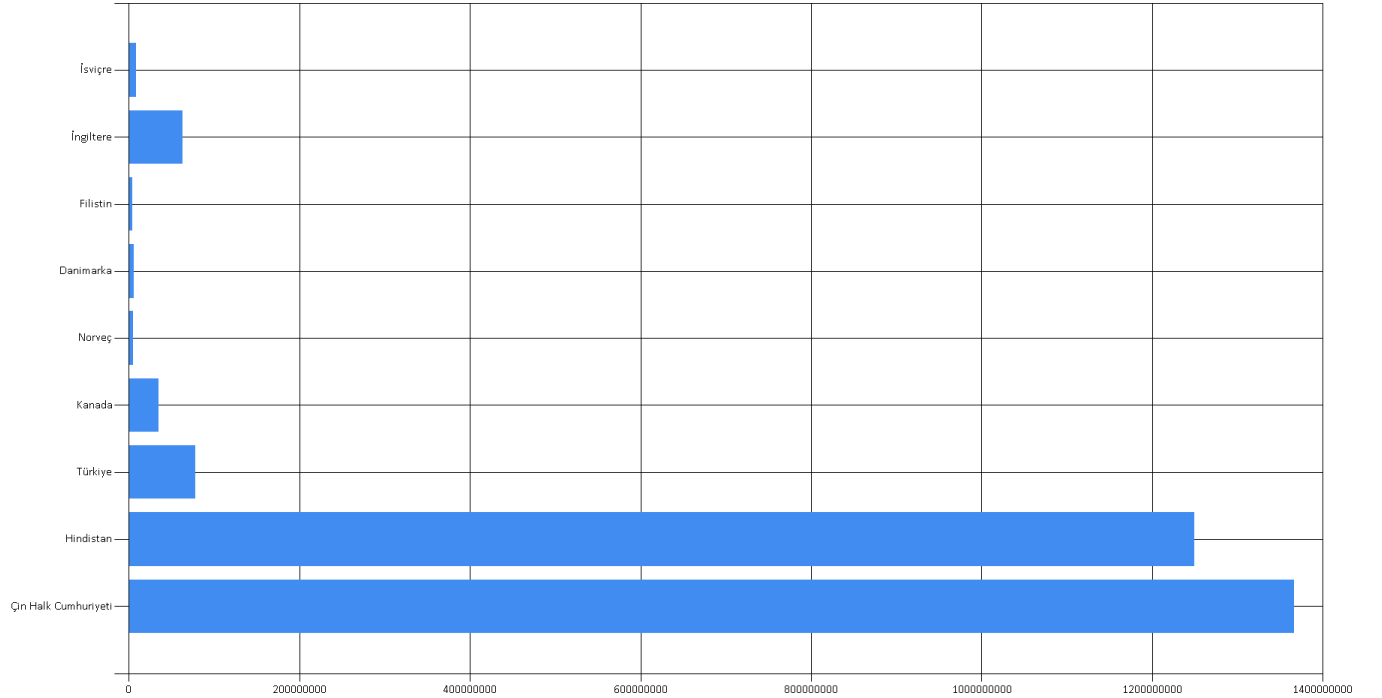
```
#load "packages/FSharp.Charting.0.90.10/FSharp.Charting.fsx"
open FSharp.Charting
open System
```

```
let ulkeNufuslari =
    ["Çin Halk Cumhuriyeti", 1366630000;
     "Hindistan", 1249030000;
     "Türkiye", 77965904;
     "Kanada", 35141542;
     "Norveç", 5063709;
     "Danimarka", 5605836;
     "Filistin", 4420549;
     "İngiltere", 63181775;
     "İsviçre", 8058100 ]
Chart.Column ulkeNufuslari
```

Uygulamayı çalıştırdıktan sonra şu grafiksel veriler ekrana çıkacaktır:



Uygulamanın en son satırında yer alan “Chart.Column ülkeNufusları” ifadesinin yerine “Chart.Bar ülkeNufusları” ifadesini koyarsak elde edeceğimiz grafik şu şekilde olur:



## 2014 Yerel Seçim Verilerinin Grafikselle Hale Getirilmesi

```
#load "packages/FSharp.Charting.0.90.10/FSharp.Charting.fsx"
open FSharp.Charting
open System

open FSharp.Charting
open System
```



```

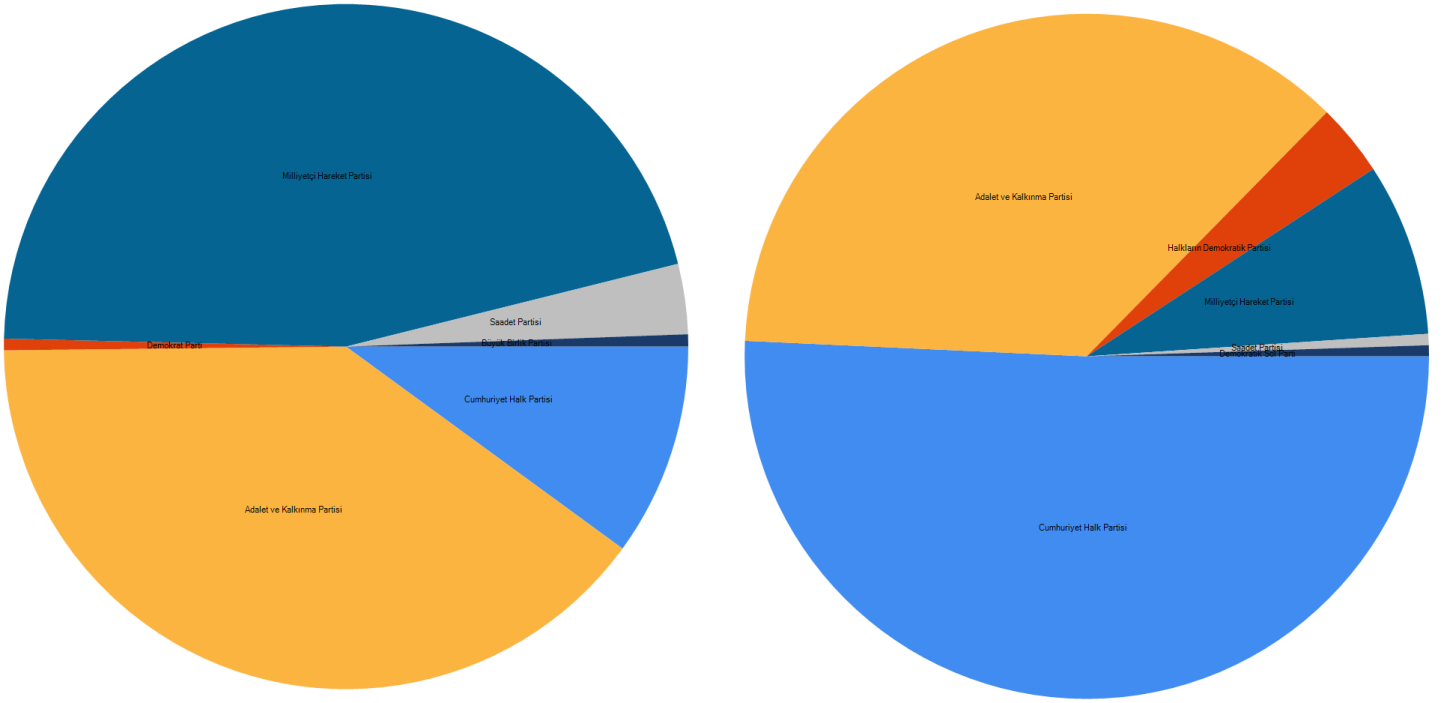
let istanbulYerelSecim =
[
    "Cumhuriyet Halk Partisi", 3426602;
    "Adalet ve Kalkınma Partisi", 4096221;
    "Halkların Demokratik Partisi", 413315;
    "Milliyetçi Hareket Partisi", 339346;
    "Saadet Partisi", 122869;
    "Büyük Birlik Partisi", 49126
]
let ankaraYerelSecim =
[
    "Cumhuriyet Halk Partisi", 1387139;
    "Adalet ve Kalkınma Partisi", 1411583;
    "Halkların Demokratik Partisi", 24993;
    "Milliyetçi Hareket Partisi", 249324;
    "Saadet Partisi", 14642;
    "Büyük Birlik Partisi", 47230
]
let izmirYerelSecim =
[
    "Cumhuriyet Halk Partisi", 1319084;
    "Adalet ve Kalkınma Partisi", 953043;
    "Halkların Demokratik Partisi", 88867;
    "Milliyetçi Hareket Partisi", 212178;
    "Saadet Partisi", 13855;
    "Demokratik Sol Parti", 13491
]
let trabzonYerelSecim =
[
    "Cumhuriyet Halk Partisi", 112048;
    "Adalet ve Kalkınma Partisi", 268350;
    "Bağımsız Türkiye Partisi", 1492;
    "Milliyetçi Hareket Partisi", 52606;
    "Saadet Partisi", 18912;
    "Büyük Birlik Partisi", 2559
]
let ispartaYerelSecim =
[
    "Cumhuriyet Halk Partisi", 11955;
    "Adalet ve Kalkınma Partisi", 47422;
    "Demokrat Parti", 647;
    "Milliyetçi Hareket Partisi", 54504;
    "Saadet Partisi", 3945;
    "Büyük Birlik Partisi", 686
]
let giresunYerelSecim =
[
    "Cumhuriyet Halk Partisi", 28087;
    "Adalet ve Kalkınma Partisi", 23697;
    "Halkların Demokratik Partisi", 144;
    "Milliyetçi Hareket Partisi", 7229;
    "Saadet Partisi", 864;
    "Büyük Birlik Partisi", 348
]
let bosSonuc =
[
    "Geçersiz il adı. Sadece 'ist, ank,izm,tra,isp,gir' girebilirsiniz", 1;
]

Console.WriteLine("Secim sonucunu gormek istediğiniz ili giriniz.(İstanbul için ist, Ankara için ank)")
let secilenIl = Console.ReadLine();

let secim = function
| "ist" -> Chart.Pie istanbulYerelSecim
| "ank" -> Chart.Pie ankaraYerelSecim
| "izm" -> Chart.Pie izmirYerelSecim
| "tra" -> Chart.Pie trabzonYerelSecim
| "isp" -> Chart.Pie ispartaYerelSecim
| "gir" -> Chart.Pie giresunYerelSecim
| _ -> Chart.Pie bosSonuc (* Herhangi bir il seçilmediği takdirde boş sonuç gosterilecektir. *)

```

Yukarıda kodları verilen uygulamayı çalıştırdıktan sonra aşağıdaki dilimli daire grafikleri ekrana çıkacaktır:



## BÖLÜM 4

### KAYNAKLAR

Bu dökümanın yazılmasında referans olarak alınan kaynaklar:

1. Programming F# 3.0 <sup>1</sup>
2. F# For C# Developers <sup>2</sup>
3. F# Programming Wikibook <sup>3</sup>
4. FSharp.Charting Library <sup>4</sup>

<sup>1</sup> <https://books.google.com/books?isbn=1449326048>

<sup>2</sup> <https://books.google.com/books?isbn=0735670226>

<sup>3</sup> [http://en.wikibooks.org/wiki/F\\_Sharp\\_Programming](http://en.wikibooks.org/wiki/F_Sharp_Programming)

<sup>4</sup> <http://fslab.org/FSharp.Charting/index.html>