
buzdağı

KAPSAMLI PYTHON KURSU İÇİN ÖVGÜLER

"No Starch Press'in daha geleneksel programlama kitaplarıyla yan yana olması gereken geleceğin klasiklerini ürettiğini görmek ilginç oldu. Kapsamlı Python Kursu bu kitaplardan biri."
— GREG LADEN, SCIENCEBLOGS

"Bazı karmaşık projeleri ele alıyor ve bu projeleri tutarlı, mantıklı ve hoş bir biçimde açıklıyor ve bu da okuyucuyu konuya çekiveriyor."
— FULL CIRCLE MAGAZINE

"İyi açıklanmış kod parçacıklarıyla iyi sunulmuş. Kitap her defasında küçük bir adım atarak sizinle birlikte çalışmakta ve ne dönüp bittiğini bütünüyle açıklayarak daha karmaşık kodlar inşa etmektedir."
— FLICKTHROUGH REVIEWS

"Kapsamlı Python Kursu ile Python'u öğrenmek oldukça pozitif bir tecrübeydi. Python'u yeni öğreniyorsanız mükemmel bir tercih."
— MIKKE GOES CODING

"Adından da belli olduğu üzere bu işi çok iyi yapıyor... Üç zorlayıcı ve eğlendirici projenin yanı sıra çok sayıda faydalı alıştırma sunuyor."
— REALPYTHON.COM

"Python ile programlamaya hızlı fakat geniş kapsamlı bir giriş olmasıyla Kapsamlı Python Kursu kütüphanenize ekleyeceğiniz ve Python'da en sonunda ustalaşmanıza yardımcı olacak bir başka harika kitap."
— TUTORIALEDGE.NET

"Kodlama tecrübesi olmayan, bütünüyle yeni başlayanlar için parlak bir seçenek. Bu derin dile katı ve karmaşık olmayan bir giriş arıyor iseniz bu kitabı tavsiye etmem gerekiyor."
— WHATPIXEL.COM

"Python ile ilgili bilmeniz gereken her şeyi ve hatta daha fazlasını tam anlamıyla içeriyor."
— FIREBEARSTUDIO.COM

KAPSAMLI PYTHON KURSU

Programlamaya Uygulamalı ve Proje Tabanlı Giriş

2. Edisyon

ERIC MATTHES

İçindekiler

İkinci Baskı için Ön Söz

Kapsamlı Python Kursunun ilk baskısına verilen tepki oldukça olumluydu. Sekiz dildeki çeviriler de dahil olmak üzere 1.000.000'dan fazla kopya basıldı. Boş zamanlarında programlama öğrenmek isteyen emeklilerin yanı sıra 10 yaşında olacak kadar genç okuyuculardan da mektuplar ve e-postalar aldım. *Kapsamlı Python Kursu;* ortaokullar, liseler ve üniversitelerde derslerde kullanılmaktadır. Daha ileri düzeyde ders kitaplarını çalışmakla yükümlü kılınan öğrenciler dersleri için *Kapsamlı Python Kursunu* refakatçi ders kitabı olarak kullanmakta ve değerli bir tamamlayıcı olduğunu düşünmektedirler. İnsanlar bu kitabı iş üzerinde becerilerini geliştirmek ve kendi yan projelerinde çalışmaya başlamak için kullanıyor. Kısacası insanlar, benim de öyle yapacaklarını ümit ettiğim şekilde kitabı geniş yelpazedeki amaçlar için kullanmaktadır.

Kapsamlı Python Kursunun ikinci bir baskısını yazma fırsatı fazlasıyla zevk vericiydi. Olgunlaşmış bir dil olmasına rağmen Python, her dilde olduğu gibi evrimleşmeye devam etmektedir. Kitabı gözden geçirmekteki amacım onu daha ince ve daha basit bir hâle getirmekti. Python 2'yi öğrenmek için artık bir neden olmamasından dolayı bu baskı sadece Python 3'e odaklanmaktadır. Pek çok Python paketinin kurulumu daha basit bir hâle geldi, dolayısıyla hazırlama ve kurulum talimatları daha basittir. Okuyucuların faydalanabileceğinin farkına vardığım birkaç konu ekledim ve Python'da işleri yürütmenin yeni ve daha basit yollarını yansıtması için bazı kısımları güncelledim. Ayrıca dilin bazı ayrıntılarının olabileceği kadar hassas bir şekilde sunulmadığı bazı kısımları anlaşılır hâle getirdim. Kendi projelerinizi inşa etmek için güvenle kullanabileceğiniz, popüler ve iyi bakımlı kütüphaneler kullanılarak bütün projeler tamamen güncellendi.

Aşağıda ikinci baskıda yapılan belirli değişikliklerin bir özetini bulacaksınız:

- Bölüm 1'de Python kurulum talimatları bütün belli başlı işletim sistemlerinin kullanıcıları için basitleştirildi. Şimdi, yeni ve profesyonel programcılar arasında popüler olan ve bütün işletim sistemlerinde düzgün çalışan metin düzenleyicisi Sublime Text'i tavsiye ediyorum.
- Bölüm 2, Python'da değişkenlerin nasıl gerçeklendiğine dair daha doğru bir açıklama içermektedir. Değişkenler, değerler için *etiketler* olarak

tanımlanır ve bu değişkenlerin Python'da nasıl davrandığını daha iyi anlaşılmasını sağlar. Kitap artık Python 3.6'da tanıtılan f-stings'i kullanmaktadır. Bu, dizgilerde değişken değerleri kullanmanın daha kolay bir yoludur. `1_000_000` gibi büyük sayıları temsil etmek için alt tire kullanımı da Python 3.6'da ortaya atılmıştır ve bu baskı bunu içermektedir. Değişkenlerin çoklu ataması daha önceleri projelerin birinde sunulmuştu ve bu tanım bütün okuyucuların yararlanması için genelleştirildi ve Bölüm 2'ye taşındı. Son olarak, bu bölümde, Python'da sabit değerleri temsil etmek için kolay anlaşılabilir bir kural yer almaktadır.

- Bölüm 6'da bir sözlükten değerleri bulup getirmek için `get()` metodunu tanıtıyorum. Bu metot bir anahtar olmadığı zaman varsayılan bir değer döndürebilmektedir.

- Uzaylı İstilası projesi (Bölüm 12-14) şimdi artık tamamen sınıf tabanlıdır. Oyunun kendisi fonksiyonlar serisi olmaktan ziyade bir sınıftır. Bu, fonksiyon çağırma sayısını ve gerekli parametre sayısını önemli ölçüde azaltarak oyunun genel yapısını büyük ölçüde basitleştirmektedir. İlk baskıya aşina olan okuyucular bu sınıf tabanlı yaklaşımın sağladığı kolaylığı takdir edecektir. Pygame şimdi bütün sistemlerde tek satırda kurulabilmekte ve okuyuculara oyunu tam ekran modunda veya pencereli modda çalıştırma seçeneği sunulmaktadır.

- Veri görselleştirme projelerinde Matplotlib kurulum talimatları bütün işletim sistemleri için daha basittir. Matplotlib'i ön plana çıkaran görselleştirmeler `subplots()` fonksiyonunu kullanmaktadır. Böylelikle siz daha karmaşık görselleştirmeler oluşturmayı öğrendikçe bunun üzerine inşa etmek daha kolay olacaktır. Bölüm 15'teki Zar Atma projesi temiz bir söz dizim ve tamamıyla kullanıcı tanımlı hâle getirilebilir çıktı barındıran iyi bakımlı bir görselleştirme kütüphanesi olan Plotly'yi kullanmaktadır.

- Bölüm 16'daki hava durumu projesi, ilk baskıda kullanılan siteye göre önümüzdeki birkaç yılda daha istikrarlı hâle gelmesi gereken NOAA'dan gelen verilere dayanmaktadır. Haritalama projesi global deprem aktivitesine odaklanmaktadır; bu projenin sonunda belirli bir zaman dilimi içerisinde bütün depremlerin konumuna odaklanmak suretiyle Dünya'nın tektonik levha sınırlarını gösteren şaşırtıcı görselleştirmelere sahip olacaksınız. Coğrafi noktalar içeren herhangi bir veri kümesini çizdirmeyi öğreneceksiniz.

- Bölüm 17, GitHub'daki açık kaynaklı projelerde Python ile ilgili aktiviteyi görselleştirmek için Plotly'yi kullanmaktadır.

- Öğrenme Günlüğü projesi (Bölüm 18-20) Django'nun en son sürümü kullanılarak inşa edilmiştir ve bu projeye Bootstrap'in en son sürümü kullanılarak stil verilmiştir. Projeyi Heroku'ya yükleme süreci `django-heroku` paketi kullanılarak basitleştirilmiştir ve süreç *settings.py* dosyalarını değiştirmekten ziyade ortam değişkenleri kullanmaktadır. Bu daha basit bir yaklaşımdır ve profesyonel programcıların modern Django projelerini nasıl dağıttıkları hususuyla uyumludur.

- Ek A, Python'u kurmada mevcut en iyi uygulamaları tavsiye etmek üzere tamamen güncellenmiştir. Ek B, Sublime Text'i kurmak için ayrıntılı talimatları ve genellikle kullanımda olan belli başlı metin düzenleyicileri ve IDE'lerin (ÇN: Integrated Development Environment: Tümleşik Geliştirme Ortamı) çoğunun kısa açıklamalarını içermektedir. Ek C kullanıcıyı yardım almak için daha yeni, daha popüler çevrim içi kaynaklara yönlendirmektedir ve Ek D, versiyon kontrolü için Git kullanma konusunda mini bir hızlandırılmış kurs vermeye devam etmektedir.

- İndeks ise *Kapsamlı Python Kursunu* gelecekteki bütün Python projeleriniz için bir referans olarak kullanmanızı sağlamak için derinlemesine güncellenmiştir.

Kapsamlı Python Kursunu okuduğunuz için teşekkürler! Herhangi bir geri bildiriminiz veya sorularınız var ise benimle irtibata geçmekten çekinmeyin.

2.3.2 Dizgilerde Değişkenleri Kullanmak

Bazı durumlarda bir değişkenin değerini bir dizgi içerisinde kullanmak isteyeceksiniz. Örneğin, iki değişkenin sırasıyla ad ve soyadı temsil etmesini ve sonra bir kişinin tam ismini görüntülemek için bu değerleri birleştirmeyi isteyebilirsiniz:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"{first_name} {last_name}"❶
print(full_name)
```

full_name.py

Bir değişkenin değerini bir dizgiye yerleştirmek için ❶'deki gibi ilk tırnak işaretinden hemen önce f harfini yerleştiriniz. Dizgi içerisinde kullanmak istediğiniz her değişkenin isminin veya isimlerinin etrafına küme parantezleri koyun. Dizgi görüntülendiğinde Python her bir değişkeninin yerine o değişkenin değerini koyacaktır.

Bu dizgilere *f-dizgileri* denir. *f* harfi *format* demektir çünkü Python küme parantezleri içerisindeki her değişkenin isminin yerine o değişkenin değerini koyarak dizgiyi formatlar. Yukarıdaki kodun çıktısı aşağıdaki gibidir:

```
ada lovelace
```

f-dizgileriyle pek çok şey yapabilirsiniz. Örneğin, aşağıda gösterildiği gibi bir değişkenle ilişkili bilgiyi kullanarak tam mesajlar oluşturmak için f-dizgilerini kullanabilirsiniz.

```
first_name = "ada"
last_name = "lovelace"
full_name = f"first_name last_name"
print(f"Hello, {full_name.title()}!")❶
```

❶'de tam isim kullanıcıyı selamlayan bir cümlede kullanılmıştır ve `title()` metodu ismi başlık biçimine dönüştürür (ÇN: Yalnızca ilk harfler büyük). Bu kod basit fakat güzel formatlanmış bir selamlama döndürür:

```
Hello, Ada Lovelace!
```

f-dizgilerini aynı zamanda bir mesaj oluşturup daha sonra bütün mesajı bir değişkene atamak için de kullanabilirsiniz:

```
first_name = "ada"
last_name = "lovelace"
full_name = f"first_name last_name"
message = f"Hello, {full_name.title()}!"❶
print(message)❷
```

Çoğunlukla nasıl davrandıkları konusunda endişelenmeden ondalık sayıları kullanabilirsiniz. Sadece kullanmak istediğiniz sayıları giriniz ve Python gerçekleştirmek istediğinizi büyük olasılıkla yapacaktır:

```
>>>0.1 + 0.1
0.2
>>>0.2 + 0.2
0.4
>>>2 * 0.1
0.2
>>>2 * 0.2
0.4
```

Ancak yanıtınızda bazen rastgele sayıda ondalık basamak olabileceğini unutmayın:

```
>>>0.2 + 0.1
0.30000000000000004
>>>3 * 0.1
0.30000000000000004
```

Bu, bütün dillerde olur ve küçük bir meseledir. Python, sonucu olabildiğince kesin bir şekilde temsil etmenin bir yolunu bulmaya çalışır; bu, bilgisayarların sayıları dahili olarak nasıl temsil etmesi gerektiği göz önüne alındığında bazen zordur. Şimdilik fazladan ondalık basamakları sadece görmezden gelin. Kısım II'de projelerde ihtiyacınız olduğunda fazladan basamaklarla uğraşmanın yollarını öğreneceksiniz.

2.4.3 Tam Sayılar ve Kayan Noktalı Sayılar

Herhangi iki sayıyı böldüğünüzde, bu iki sayı bölme işlemi tam sayıyla sonuçlansa bile her zaman sonuç olarak kayan noktalı bir sayı elde edersiniz.

```
>>>4/2
2.0
```

Eğer bir tam sayıyı ve bir kayan noktalı sayıyı bir başka işlemde karıştırırsanız sonuç olarak yine bir kayan noktalı sayı elde edersiniz:

```
>>>1 + 2.0
3.0
>>>2 * 3.0
6.0
>>>3.0 ** 2
9.0
```

Python, çıktı bir tam sayı olsa bile, kayan nokta kullanan herhangi bir işlemde Python varsayılan olarak bir kayan nokta kullanır.

2.4.4 Sayılarda Alt Tire

Uzun sayıları yazarken, büyük sayıları daha okunur hâle getirmek için alt tireler kullanarak basamakları gruplandırabilirsiniz:

```
>>>universe_age = 14_000_000_000      #kâinat_yaşı
```

Alt tireler kullanılarak tanımlanan bir sayıyı ekrana yazdırdığınızda, Python sadece basamakları ekrana yazdırır:

```
>>>print(universe_age)
14000000000
```

Python bu tür değerleri saklarken alt tireleri görmezden gelir. Basamakları üçer üçer gruplamasanız bile değer etkilenmeyecektir. Python için *1000* ile *1_000* aynıdır ve *1_000* ile *10_00* de aynıdır. Bu özellik tam sayılar ve kayan noktalı sayılar için geçerlidir ancak Python 3.6 ve sonrasında mevcuttur.

2.4.5 Çoklu Atama

Tek satır kullanarak birden çok değişkene değerler atayabilirsiniz. Bu, programlarınızı kısaltmaya yardımcı olabilir ve okunması daha kolay hâle getirir. Bu tekniği birtakım sayılara ilk değer atarken sıkça kullanacaksınız. Örneğin, aşağıda x, y ve z değişkenlerine nasıl ilk değer olarak sıfır verileceğini görmektesiniz:

```
>>>x, y, z = 0, 0, 0
```

Değişken isimlerini virgüllerle ayırmanız ve bunu değerler için de yapmanız gerekmektedir. Python her bir değeri konumlarına göre sırasıyla değişkenlere atayacaktır. Değer sayısı değişken sayısıyla eşleştiği sürece Python bunları doğru bir şekilde eşleştirecektir.

2.4.6 Sabitler

Bir sabit, bir programın hayatı boyunca değeri aynı kalan bir değişken gibidir. Python'da yerleşik sabit tipleri yoktur ancak Python programcıları bir değişkenin sabit olarak muamele görmesi ve hiçbir zaman değiştirilmemesi gerektiğini belirtmek için tamamıyla büyük harfler kullanırlar:

```
MAX_CONNECTIONS = 5000
```

Kodunuzda bir değişkene sabitmiş gibi muamele etmek istiyorsanız değişkenin isminin tamamını büyük harflerle yazınız.

Bölüm 3

LİSTELERE GİRİŞ

Bu ve bundan sonraki bölümde listelerin ne olduğunu ve bir listedeki elemanlarla nasıl çalışmaya başlayacağınızı öğreneceksiniz. Listeler ister birkaç isterse milyonlarca öğeniz olsun, bilgi kümelerini tek bir yerde saklamanızı sağlar. Listeler Python'un yeni programcıların kolayca anlayabileceği en güçlü özelliklerinden biridir ve programlamadaki pek çok önemli kavramı birbirine bağlar.

3.1 Liste Nedir?

Liste belirli bir sıralamadaki öğeler topluluğudur. Alfabedeki harfleri, 0'dan 9'a kadar olan rakamları veya ailenizdeki herkesin isimlerini içeren bir liste yapabilirsiniz. Bir listeye istediğiniz her şeyi koyabilirsiniz ve listenizdeki öğelerin belirli bir biçimde birbiriyle ilgili olması gerekmez. Listelerin genellikle birden fazla eleman içermesinden dolayı listenizin ismini `harfler`, `rakamlar` veya `isimler` gibi çoğul yapmak iyi bir fikirdir.

Python'da köşeli parantezler (`[ ]`) liste belirtir ve listedeki münferit elemanlar virgüllerle ayrılır. Aşağıda birkaç çeşit bisiklet içeren basit bir liste örneği bulunmaktadır:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles)
```

bicycles.py

Python'dan bir listeyi ekrana yazdırmasını isterseniz, köşeli parantezler de dahil olmak üzere kendi liste gösterimini döndürür:

```
['trek', 'cannondale', 'redline', 'specialized']
```

Bu, kullanıcıların görmesini istediğiniz çıktı değildir, bu nedenle bir listedeki öğelere tek tek nasıl erişileceğini öğrenelim.

3.1.1 Bir Listedeki Elemanlara Erişmek

Listeler sıralı koleksiyonlardır, dolayısıyla bir listedeki herhangi bir elemana, Python'a arzu edilen öğenin konumunu veya *indeksini* bildirerek erişebilirsiniz. Bir listedeki elemana erişmek için listenin isminden sonra köşeli parantezler içerisinde öğenin indeksini yazınız.

Örneğin bisiklet listesinden ilk bisikleti çekip çıkaralım:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0])❶
```

Bunu yapan söz dizim ❶'de gösterilmiştir. Bir listeden tek bir öğe istediğimiz zaman Python köşeli parantez olmaksızın sadece bu elemanı döndürür:

```
trek
```

Bu, kullanıcılarınızın görmesini istediğiniz sonuçtur —temiz, düzgün formatlanmış bir çıktı.

Ayrıca Bölüm 2'deki dizgi metotlarını da bu listedeki herhangi bir eleman üzerinde kullanabilirsiniz. Örneğin, `title()` metodunu kullanarak `'trek'` elemanını daha düzgün bir şekilde formatlayabilirsiniz:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[0].title())
```

Bu örnek bir önceki örnekle aynı çıktıyı verir, sadece `'Trek'` değerinin ilk harfi büyük olacaktır.

3.1.2 İndeks Konumları 1'den Değil 0'dan Başlar

Python, bir listedeki ilk öğenin konum 1'de değilde konum 0'da olduğunu kabul eder. Bu çoğu programlama dili için de aynıdır. Bunun sebebi daha alt bir seviyede liste işlemlerinin nasıl gerçeklendiğiyle ilgilidir. Beklenmedik sonuçlar alıyorsanız basit bir 1-kaydırma-hatası yapıp yapmadığınızı tespit ediniz.

Listedeki ikinci elemanın indeksi 1'dir. Bu sayma sistemini kullanarak listedeki konumundan 1 çıkarmak suretiyle listeden istediğiniz herhangi bir elemanı elde edebilirsiniz. Örneğin bir listedeki dördüncü öğeye erişmek için indeks 3'teki öğeyi talep edersiniz.

Aşağıdaki kod indeks 1 ve indeks 3'teki bisikletleri talep etmektedir:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[1])
print(bicycles[3])
```

Bu kod listedeki ikinci ve dördüncü bisikletleri döndürür:

```
cannondale
specialized
```

Listenin en son elemanına erişmek için Python'un özel bir söz dizimi vardır. İndeks -1'deki öğeyi talep etmek suretiyle Python her zaman listedeki en son elemanı döndürür:

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
print(bicycles[-1])
```

Bu kod `'specialized'` değerini döndürür. Bu söz dizim oldukça kullanışlıdır çünkü çoğu zaman listenin uzunluğunu tam olarak bilmeden listenin en son elemanlarına erişmek isteyeceksiniz. Bu kural diğer negatif indeks değerlerini de kapsar. İndeks -2 listenin sondan ikinci ve indeks -3 sondan üçüncü elemanını döndürür ve bu böyle devam eder.

3.1.3 Bir Listeden Tek Tek Değerler Kullanmak

Diğer değişkenlerde olduğu gibi bir listedeki değerleri tek tek kullanabilirsiniz. Örneğin bir listedeki değere dayalı olarak bir mesaj oluşturmak için f-dizgilerini kullanabilirsiniz.

Listeden ilk bisikleti alıp, bu değeri kullanarak bir mesaj yazmayı deneyelim.

```
bicycles = ['trek', 'cannondale', 'redline', 'specialized']
message=f"Benim ilk bisikletim {bicycles[0].title()} idi."❶
print(message)
```

❶'de `bicycles[0]`'daki değeri kullanarak bir cümle oluşturuyoruz ve bu cümleyi `message` değişkenine atıyoruz. Çıktı, listedeki ilk bisikletle ilgili basit bir cümledir:

```
Benim ilk bisikletim bir Trek idi.
```

Kendiniz Deneyin

7-1. Kiralık Araba: Kullanıcıya ne tür bir kiralık araba istediğini soran bir program yazınız. Bu araba hakkında örneğin *"Bakalım size bir Subaru bulabilecek miyim"* gibi bir mesajı ekrana yazdırınız.

7-2. Restoranda Oturulacak Yer: Kullanıcıya akşam yemeğine gelecek grupta kaç kişi olduğunu soran bir program yazınız. Yanıt sekizden fazla ise boş bir masa için beklemeleri gerektiğini söyleyen bir mesajı ekrana yazdırınız. Aksi takdirde masalarının hazır olduğunu bildiriniz.

7-3. 10'un Katları: Kullanıcıdan bir sayı girmesini isteyiniz ve bu sayının 10'un katları olup olmadığını bildiriniz.

7.2 `while` Döngülerine Giriş

`for` döngüsü bir öğe koleksiyonunu alır ve koleksiyondaki her bir öğe için bir kod bloğunu bir kez çalıştırır. Buna karşın, `while` döngüsü belirli bir koşul doğru olduğu sürece çalışır.

7.2.1 `while` Döngüsü Eylem Hâlinde

Bir sayı serisi boyunca saymak için `while` döngüsü kullanabilirsiniz. Örneğin aşağıdaki `while` döngüsü 1'den 5'e kadar sayar.

```
current_number = 1
while current_number <= 5:
  print(current_number)
  current_number += 1
```

counting.py

İlk satırda `current_number`'a (ÇN: Mevcut sayı) 1 değerini atayarak saymaya 1'den başlamış oluyoruz. Daha sonra `while` döngüsü `current_number` 5'e eşit veya 5'den küçük olduğu sürece çalışacak şekilde ayarlanır. Döngünün içerisindeki kod `current_number`'ın değerini ekrana yazdırır ve `current_number+=1` ile bu değere 1 ekler. (`+=` operatörü; `current_number = current_number + 1` şeklinde yazmanın kısa yoludur.)

`current_number <= 5` koşulu doğru olduğu sürece Python döngüyü tekrarlar. 1, 5'ten küçük olduğu için Python ekrana 1 yazdırır ve 1 eklemek suretiyle mevcut sayıyı 2 yapar. 2, 5'ten küçük olduğu için Python 2'yi ekrana yazdırır ve 1 eklemek suretiyle mevcut sayıyı 3 yapar ve bu böyle devam eder.

`current_number`'ın değeri 5'ten büyük olduğunda döngü çalışmasını durdurur ve program sona erer:

```
1
2
3
4
5
```

Her gün kullandığınız programlar kuvvetle muhtemel `while` döngüleri içeriyordur. Örneğin, bir oyun programı siz oynamak istediğiniz sürece çalışmasını sürdürmek için ve siz çıkmak istediğinizde çalışmasını durdurmak için `while` döngüsüne ihtiyaç duyar. Programlar biz söylemeden çalışmalarını durdursaydı veya biz çıkmak istediğimizde bile çalışsaydı bu programların bir özelliği kalmazdı, dolayısıyla `while` döngüleri oldukça faydalıdır.

7.2.2 Kullanıcının Ne Zaman Çıkacağını Seçmesine İzin Verme

Programın çoğunu bir `while` döngüsünün içerisine koyarak *parrot.py* programının kullanıcı istediği sürece çalışmasını sağlayabiliriz. Bir *çıkış değeri* tanımlayacağız ve kullanıcı çıkış değerini girmediği sürece programı çalışır hâlde tutacağız:

```
prompt = "\nBana bir şey söyle, sana onu tekrar edeyim:❶
prompt +="\nProgramı sona erdirmek için 'quit' giriniz."

message = ""❷
while message != 'quit':❸
  message = input(prompt)
  print(message)
```

parrot.py

❶'de kullanıcıya iki tane seçeneğini söyleyen bir istemi tanımlıyoruz: Bir mesaj girmek veya çıkış değeri girmek (bu durumda `'quit'`). Daha sonra ❷'de kullanıcının girdiklerinin kaydını tutmak için `message` değişkenini oluşturuyoruz. `message`'ı boş bir dizgi, yani " " olarak tanımlıyoruz, böylece Python'un `while` satırına ilk kez ulaştığında kontrol edecek bir şeyleri olur. Program ilk kez çalıştığında ve Python `while` ifadesine ulaştığında `message`'ın değerini `'quit'` ile karşılaştırması gerekir ancak henüz bir kullanıcı girdisi girilmemiştir. Eğer Python'un karşılaştıracak bir şeyi yoksa programı çalıştırmaya devam edemez. Bu problemi çözmek için `message`'a bir başlangıç değeri verdiğimizden emin oluyoruz. Bir boş dizgiden ibaret olmasına rağmen Python için bu bir anlam ifade edecek ve Python'un `while` döngüsünün çalışmasını sağlayan

karşılaştırma işlemini gerçekleştirmesini sağlayacaktır. ❸'te `message`'ın değeri `'quit'` olmadığı sürece `while` döngüsü çalışır.

Döngüye girerken `message` boş bir dizgiden ibarettir, dolayısıyla Python döngüye girer. `message = input(prompt)`'da Python istemi görüntüler ve kullanıcıdan girdisini girmesini bekler. Kullanıcı her neyi girdiyse bu `message`'a atanır ve ekrana yazdırılır. Daha sonra Python `while` ifadesindeki koşulu tekrardan değerlendirir. Kullanıcı `'quit'` kelimesini girmediği sürece istemci tekrar görüntülenir ve Python girdi beklemeye devam eder. Kullanıcı en sonunda `'quit'` girdiğinde Python `while` döngüsünü çalıştırmayı durdurur ve program durur:

```
Bana bir şey söyle, sana onu tekrar edeyim:
Programı sona erdirmek için 'quit' giriniz. Herkese merhaba!
Herkese merhaba!

Bana bir şey söyle, sana onu tekrar edeyim:
Programı sona erdirmek için 'quit' giriniz. Tekrar merhaba.
Tekrar merhaba.

Bana bir şey söyle, sana onu tekrar edeyim:
Programı sona erdirmek için 'quit' giriniz. quit
quit
```

Program, `'quit'` kelimesini gerçek bir mesajmış gibi ekrana yazdırması haricinde düzgün çalışır. Basit bir if testi bu durumun önüne geçer:

```
prompt = "\nBana bir şey söyle, sana onu tekrar edeyim:
prompt += "\nProgramı sona erdirmek için 'quit'giriniz. "

message = '' ''
while message != 'quit':
  message = input(prompt)
  if message != 'quit':
    print(message)
```

Program artık mesajı görüntülemeden önce hızlı bir kontrol gerçekleştirir ve sadece çıkış değeriyle eşleşmediğinde mesajı ekrana yazdırır.

```
Bana bir şey söyle, sana onu tekrar edeyim:
Programı sona erdirmek için 'quit' giriniz. Herkese merhaba!
Herkese merhaba!

Bana bir şey söyle, sana onu tekrar edeyim:
Programı sona erdirmek için 'quit' giriniz. Tekrar merhaba.
Tekrar merhaba.

Bana bir şey söyle, sana onu tekrar edeyim:
Programı sona erdirmek için 'quit' giriniz. quit
```

8-2. Sevdiğiniz Kitap: Sadece `title` isimli bir parametre kabul eden `favorite_book()` isimli bir fonksiyon yazınız. Fonksiyon, *"Sevdiğim kitaplardan biri Alice Harikalar Ülkesinde adlı kitaptır"* gibi bir mesajı ekrana yazmalıdır. Fonksiyon çağrısında kitabın başlığını argüman olarak dahil ettiğinizden emin olunuz ve fonksiyonu çağırınız.

8.2 Argümanlar Göndermek

Bir fonksiyon tanımının birden çok parametresinin olabilmesinden ötürü fonksiyon çağrısı birden çok argümana ihtiyaç duyabilir. Fonksiyonlarınıza argümanları çeşitli yollarla gönderebilirsiniz. *Konumsal argümanlar* kullanabilirsiniz. Bu argümanlar parametrelerin yazıldığı sırada olmalıdır. Bunun yanı sıra *anahtar kelime argümanları* da kullanabilirsiniz. Bu durumda her bir argüman bir değişken ismi ve bir değerden ve değerlerin bir listesiden ve sözlüğünden ibaret olur. Bunların her birine sırayla bakalım:

8.2.1 Konumsal Argümanlar

Bir fonksiyonu çağırdığınızda Python, fonksiyon çağrısındaki her bir argümanı fonksiyon tanımındaki bir parametreyle eşleştirmelidir. Bunu yapmanın en basit yolu sunulan argümanların sırasına bağlıdır. Bu şekilde eşleşen değerlere *konumsal argümanlar* denir.

Bunun nasıl işlediğini görmek için evcil hayvanlarla ilgili bilgileri görüntüleyen bir fonksiyon düşünün. Bu fonksiyon aşağıda gösterildiği gibi her bir evcil hayvanın türünü ve ismini bize söyler:

```
def describe_pet(animal_type, pet_name):❶
  """Bir evcil hayvanla ilgili bilgileri görüntüle."""
  print(f"\nBen bir {animal_type} sahibiyim.")
  print(f"{animal_type.title()} hayvanımın
        ismi {pet_name.title()}'dir.")
describe_pet('hamster', 'harry')❷
```

pets.py

❶'deki fonksiyon tanımı bu fonksiyonun hayvanın cinsine ve ismine ihtiyaç duyduğunu göstermektedir. `describe_pet()`'i çağırdığımızda hayvanın cinsini ve ismini bu sırada sunmamız gerekir. Örneğin, ❷'de `'hamster'` argümanı `animal_type` (ÇN: Hayvanın cinsi) parametresine ve `'harry'` argümanı da `pet_name` parametresine atanmaktadır. Fonksiyonun gövdesinde bu iki parametre, tasvir edilen evcil hayvan hakkındaki bilgiyi görüntülemek için kullanılmaktadır.

Çıktı Harry isminde bir hamsteri tasvir etmektedir.

```
Ben bir hamster sahibiyim.
Hamster hayvanımın ismi Harry'dir.
```

8.2.2 Çoklu Fonksiyon Çağırma

Bir fonksiyonu istediğiniz kadar çağırabilirsiniz. Farklı, ikinci bir köpeği tasvir etmek için `describe_pet()`'i sadece bir kez daha çağırmanız gerekir:

```
def describe_pet(animal_type, pet_name):
  """Bir evcil hayvanla ilgili bilgileri görüntüle."""
  print(f"\nBen bir {animal_type} sahibiyim.")
  print(f"{animal_type.title()} hayvanımın
          ismi {pet_name.title()}'dir.")

describe_pet('hamster', 'harry')
describe_pet('köpek', 'willie')
```

Bu ikinci fonksiyon çağrısında `describe_pet()`'e `'dog'` ve `'willie'` argümanlarını gönderiyoruz. Daha önce kullandığımız argüman kümesinde olduğu gibi Python, `'dog'`u `animal_type` parametresi ve `'willie'`yi de `pet_name` parametresiyle eşleştirir. Daha önce olduğu gibi fonksiyon görevini yerine getirir ancak bu sefer Willie isimli bir köpeğe ilişkin değerleri ekrana yazdırır.

```
Ben bir hamster sahibiyim.
Hamster hayvanımın ismi Harry'dir.

Ben bir köpek sahibiyim.
Köpek hayvanımın ismi Willie'dir.
```

Bir fonksiyonu birden çok kez çağırmak çok verimli bir çalışma şeklidir. Evcil hayvanı tasvir eden kod fonksiyonda sadece bir kez yazılmıştır. Daha sonra yeni bir evcil hayvan tasvir etmek istediğinizde fonksiyonu yeni evcil hayvanın bilgileriyle çağırmanız gerekir. Evcil bir hayvanı tasvir eden kod on satır kaplasa bile fonksiyonu tekrar çağırarak tek satırda yeni bir evcil hayvanı tasvir edebilirsiniz.

Fonksiyonlarınızda ihtiyacınız olduğu kadar konumsal argüman kullanabilirsiniz. Python, fonksiyonu çağırırken sunduğunuz argümanları inceler ve her birini fonksiyonun tanımındaki karşılık gelen parametreyle eşleştirir.

8.2.3 Konumsal Argümanlarda Sıralama Önemlidir

Konumsal argümanlar kullanırken bir fonksiyonu çağırdığınızda argümanların sıralamasını karıştırırsanız beklenmedik sonuçlar elde edebilirsiniz:

```
def describe_pet(animal_type, pet_name):
  """Bir evcil hayvanla ilgili bilgileri görüntüle."""
  print(f"\nBen bir {animal_type} sahibiyim.")
  print(f"{animal_type.title()}'ımın
          ismi {pet_name.title()}'dir.")

describe_pet('harry', 'hamster')
```

Bu fonksiyon çağrısında ilk önce ismi sonra hayvanın cinsini listeliyoruz. `'harry'` argümanı ilk olarak listelendiğinden, bu değer `animal_type` parametresine atanır. Benzer şekilde `'hamster'` `pet_name`'e atanmıştır. Şimdi "Hamster" isminde bir "harry"ye sahibiz.

```
Ben bir harry sahibiyim.
Harry hayvanımın ismi Hamster'dir.
```

Böyle gülünç sonuçlar elde ederseniz, fonksiyon çağrınızdaki argümanların sırasının fonksiyon tanımındaki parametrelerin sırasıyla eşleştiğinden emin olunuz.

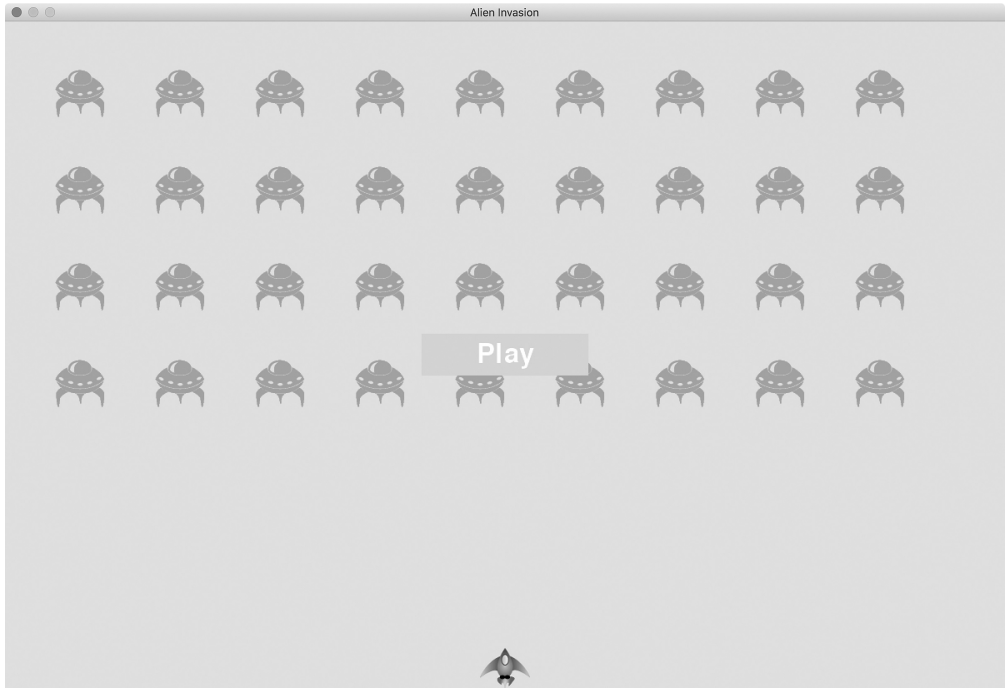
8.2.4 Anahtar Kelime Argümanlar

Anahtar kelime argümanı bir fonksiyona gönderdiğiniz isim-değer çiftidir. Argümanın içinde isim ile değeri doğrudan doğruya ilişkilendirdiğinizden dolayı argümanı fonksiyona gönderdiğinizde hiçbir karışıklık olmaz (Hamster isimli bir Harry söz konusu olmaz). Anahtar kelime argümanlar; sizi, fonksiyon çağrısında argümanları doğru sıralama endişesinden kurtarır ve fonksiyon çağrısında her bir değerin rolünü anlaşılır hâle getirir.

`describe_pet()`'i çağırmak için *pets.py*'yi anahtar kelime argümanlar kullanacak şekilde yeniden yazalım:

```
def describe_pet(animal_type, pet_name):
  """Bir evcil hayvanla ilgili bilgileri görüntüle."""
  print(f"\nBen bir {animal_type} sahibiyim.")
  print(f"{animal_type.title()}'ımın
          ismi {pet_name.title()}'dir.")

describe_pet(animal_type='hamster', pet_name='harry')
```

`describe_pet()` fonksiyonu değişmemiştir. Ancak fonksiyonu çağırdığımızda Python'a açıkça her bir argümanın hangi parametreyle eşleşmesi gerektiğini söylemekteyiz. Python fonksiyon çağrısını okuduğunda `'hamster'` argümanını `animal_type` parametresine ve `'harry'` argümanını da `pet_name`'e atayacağını bilmektedir. Çıktı doğru bir şekilde Harry isimli bir hamsterimizin olduğunu göstermektedir.



Şekil 14.1: *Oyun aktif olmadığında Play düğmesi görünür*

14.1.3 Oyunu Başlatmak

Oyuncu Play'e tıkladığında yeni bir oyuna başlamak için düğme üzerindeki fare olaylarını izlemek üzere aşağıdaki `elif` bloğunu `_check_events()`'in sonuna ekleyiniz.

```
    def _check_events(self):
      """Tuşa basmalara ve fare olaylarına yanıt ver."""
      for event in pygame.event.get():
        if event.type == pygame.QUIT:
          -kesinti-
      elif event.type == pygame.MOUSEBUTTONDOWN:❶
        mouse_pos = pygame.mouse.get_pos()❷
        self._check_play_button(mouse_pos)❸
```

alien_invasion.py

❶'de oyuncu ekranda herhangi bir yere tıkladığında Pygame `MOUSEBUTTONDOWN` (Fare Tuşu Basıldı) olayını tespit eder ancak oyunumuzu sadece Play düğmesi üzerindeki fare tıklamalarına yanıt verecek şekilde sınırlamak istiyoruz. Bunu başarmak için ❷'de fare düğmesi tıklandığında fare imlecinin x ve y koor-

dinatlarını içeren bir demet döndüren `pygame.mouse.get_pos()`'u kullanıyoruz. ❸'te bu değerleri yeni metot `_check_play_button()`'a gönderiyoruz.

Aşağıda `_check_events()`'in sonrasına yerleştirmeyi tercih ettiğim `_check_play_button()` metodu bulunmaktadır:

```
    def _check_play_button(self, mouse_pos):
        """Oyuncu Play'e tıkladığında yeni oyuna başla."""
        if self.play_button.rect.collidepoint(mouse_pos):❶
            self.stats.game_active = True
```

alien_invasion.py

❶'de fare tıklama noktasının Play düğmesinin `rect`'i ile tanımlanan bölgeyle örtüşüp örtüşmediğini kontrol için `rect` metodu `collidepoint()`'i (ÇN: Çarpışma noktası) kullanıyoruz. Eğer öyleyse `game_active`'i `True` olarak ayarlarız ve oyun başlar!

Bu noktada oyunun tamamını başlatıp oynayabilmeniz gerekir. Oyun sona erdiğinde `game_active`'in değeri `False` olmalı ve Play düğmesi yeniden görünmelidir.

14.1.4 Oyunu Resetlemek

Az önce yazmış olduğumuz Play düğmesi kodu oyuncu Play'e ilk kez tıkladığında çalışır. Ancak bu kod ilk oyun sona erdiğinde çalışmaz, çünkü oyunun sona ermesine sebebiyet veren koşullar resetlenmemiştir.

Oyuncu Play'e her bastığında oyunu resetlemek için aşağıda gösterildiği gibi oyun istatistiklerini resetlememiz, eski uzaylılar ve mermileri temizlememiz, yeni bir filo oluşturmamız ve gemiyi merkeze yerleştirmemiz gerekir:

```
    def _check_play_button(self, mouse_pos):
        """Oyuncu Play'e bastığında yeni bir oyun başlat."""
        if self.play_button.rect.collidepoint(mouse_pos):
            # Oyun istatistiklerini resetle.
            self.stats.reset_stats()❶
            self.stats.game_active = True
            # Geri kalan uzaylılar ve mermilerden kurtul.
            self.aliens.empty()❷
            self.bullets.empty()
            # Yeni bir filo oluştur ve gemiyi merkeze yerleştir.
            self._create_fleet()❸
            self.ship.center_ship()
```

alien_invasion.py

❶'de oyun istatistiklerini resetliyoruz. Bu, oyuncuya üç yeni gemi verir. Daha sonra `game_active`'i `True` olarak ayarlarız. Böylelikle bu fonksiyondaki

kod çalışmasını bitirir bitirmez oyun başlayacaktır. ❷'de `aliens` ve `bullets` grubunu boşaltıyoruz ve sonra ❸'te yeni bir filo oluşturup gemiyi merkeze yerleştiriyoruz.

Şimdi Play'e her bastığınızda oyun doğru bir şekilde resetlenecektir ve bu, oyunu istediğiniz kadar oynayabilmenizi sağlayacaktır!

14.1.5 Play Düğmesinin Etkinliğini Kaldırmak

Play düğmemizle ilgili sorun, Play düğmesi görünür olmadığında bile ekrandaki düğme bölgesinin tıklamalara yanıt vermeye devam etmesidir. Play düğmesi alanına oyun başladıktan sonra yanlışlıkla tıklarsanız oyun yeniden başlayacaktır!

Bu sorunu gidermek için oyunu sadece `game_active False` olduğunda başlayacak şekilde ayarlayınız:

```
def _check_play_button(self, mouse_pos):
 """Oyuncu Play'e tıkladığında yeni bir oyun başlat."""
 button_clicked=self.play_button.rect.collidepoint(mouse_pos)❶
 if button_clicked and not self.stats.game_active:❷
    self.stats.reset_stats()
    -kesinti-
```

alien_invasion.py

❶'de `button_clicked` (ÇN: Düğme tıklandı) bayrağı `True` veya `False` değeri saklar ve ❷'de sadece Play tıklanmış ise ve oyun o an aktif değilse oyun başlar. Bu davranışı test etmek için yeni bir oyun başlatınız ve Play düğmesinin olması gerektiği yere tekrarlı olarak tıklayınız. Her şey beklenildiği gibi çalışıyorsa Play düğmesi alanına tıklamanın oyunun oynayışı üzerine hiçbir etkisi olmayacaktır.

14.1.6 Fare İmlecini Gizlemek

Oynamaya başlamak için fare imlecinin görünür olmasını isteriz ancak oyun başladı mı imleç karşımıza çıkmaya başlar. Bu sorunu gidermek için oyun aktif olduğunda imleci görünmez yapacağız.

Bunu `_check_play_button()`'daki `if` bloğunun sonunda yapacağız:

```
def _check_play_button(self, mouse_pos):
   """Oyuncu Play'e tıkladığında yeni bir oyun başlat."""
   button_clicked = self.play_button.rect.collidepoint(mouse_pos)
   if button_clicked and not self.stats.game_active:
      -kesinti-
      # Fare imlecini gizle.
      pygame.mouse.set_visible(False)
```

alien_invasion.py

`RandomWalk` sınıfı için iki metoda ihtiyacımız olacak: `__init__()` metodu ve yürüyüşteki noktaları hesaplayacak olan `fill_walk()`. Aşağıda gösterildiği gibi `__init__()`'den başlayalım:

```
from random import choice❶

class RandomWalk:
  """Rastgele yürüyüşler üreten bir sınıf."""

  def __init__(self, num_points=5000)❷
  """Yürüyüşün niteliklerine ilk değer ata."""
  self.num_points = num_points

  # Bütün yürüyüşler (0,0)'dan başlar.
  self.x_values = [0] ❸
  self.y_values = [0]
```

random_walk.py

Rastgele kararlar verebilmek için muhtemel hareketleri bir listede saklayacağız ve ❶'de her bir adım atıldığında hangi hareketin yapılacağına karar vermek için `random` (ÇN: Rastgele) modülünden `choice()` (ÇN: Seçenek) fonksiyonunu kullanacağız. Daha sonra ❷'de yürüyüşteki varsayılan nokta sayısını 5000 yapıyoruz. Bu rakam bazı ilginç örüntüler üretmek için yeterince büyük ancak yürüyüşleri hızlı bir şekilde üretmek için yeterince küçüktür. Daha sonra ❸'te x ve y değerlerini tutmak için iki liste oluşturuyoruz ve her bir yürüyüşe (0,0)'da başlıyoruz.

15.3.2 Yönleri Seçmek

Yürüyüşümüzü noktalarla doldurmak ve her bir adımın yönünü belirlemek için aşağıda gösterildiği gibi `fill_walk()` metodunu kullanacağız. Bu metodu *random_walk.py*'ye ekleyiniz.

```
def fill_walk(self):
  """Yürüyüşteki bütün noktaları hesapla."""

  # Yürüyüş istenen uzunluğa ulaşana
  # kadar adım atmaya devam et.
  while len(self.x_values) < self.num_points:❶
    # Hangi yöne gidileceğine ve bu yönde
    # ne kadar uzağa gidileceğine karar ver.
    x_direction = choice([1, -1])❷
    x_distance = choice([0, 1, 2, 3, 4])
    x_step = x_direction * x_distance❸
```

```
        y_direction = choice([1, -1])
        y_distance = choice([0, 1, 2, 3, 4])
        y_step = y_direction * y_distance❹

        # Hiçbir yere gitmeyen hareketleri reddet.
        if x_step == 0 and y_step == 0:❺
          continue

        # Yeni konumu hesapla.
        x = self.x_values[-1] + x_step ❻
        y = self.y_values[-1] + y_step

        self.x_values.append(x)
        self.y_values.append(y)
```

random_walk.py

❶'de yürüyüş doğru sayıda nokta ile doldurulana kadar çalışan bir döngü kuruyoruz. `fill_walk()` metodunun ana kısmı Python'a dört rastgele kararı nasıl simüle edeceğini söyler: Yürüyüş sağa mı gidecek sola mı gidecek? Bu yönde ne kadar uzağa gidecek? Yukarı mı aşağı mı gidecek? Bu yönde ne kadar uzağa gidecek?

❷'de `x_direction` için bir değer seçmek adına sağa hareket için 1, sola hareket için -1 döndüren `choice[1, -1]`'u kullanıyoruz. Daha sonra `choice([0, 1, 2, 3 ,4])`, 0 ile 4 arasında rastgele bir tam sayı seçerek Python'a bu yönde (`x_distance`) ne kadar uzağa gideceğini söyler. (0'ın dahil edilmesi hem y ekseninde adımlar atmamızı sağlar hem de her iki eksende de harekete sahip adımlar atmamızı sağlar.)

❸ ve ❹'te x ve y yönlerindeki her bir adımın uzunluğunu hareketin yönünü seçilen mesafeyle çarparak belirleriz. `x_step` için pozitif bir sonuç sağa hareket, negatif bir sonuç sola hareket ve 0 ise dikey hareket anlamına gelir. `y_step` için pozitif bir sonuç yukarı hareket, negatif bir sonuç aşağı hareket ve 0 ise yatay hareket anlamına gelir. ❺'te hem `x_step`'in hem `y_step`'in değeri 0 ise yürüyüş hiçbir yere gitmez, öyleyse bu hareketi görmezden gelmek için döngüye devam ederiz.

Yürüyüş için bir sonraki x değerini elde etmek için ❻'da `x_step`'teki değeri `x_values`'da saklı son değere ekleriz ve aynı şeyi y değerleri için de yaparız. Bu değerlere sahip olduğumuzda bunları `x_values` ve `y_values`'ın sonuna ekleriz.

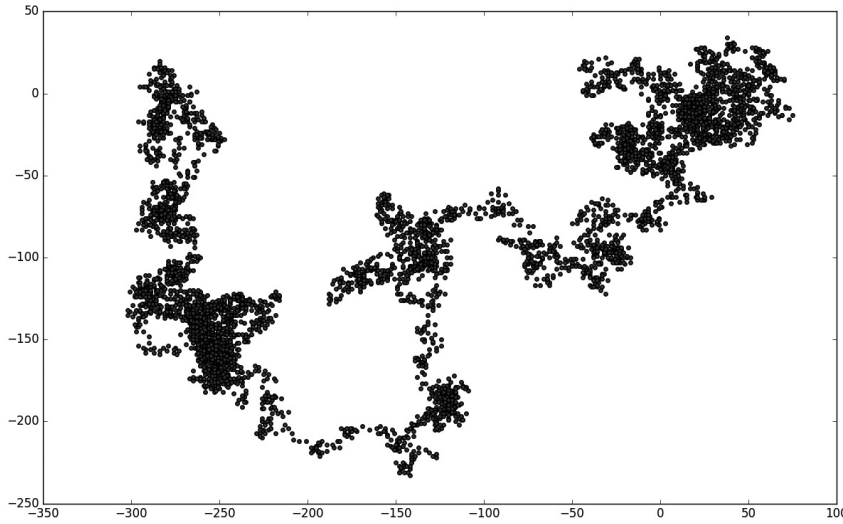
15.3.3 Rastgele Yürüyüşü Çizdirmek

İşte yürüyüşteki bütün noktaları çizdiren kod:

```
import matplotlib.pyplot as plt
from random_walk import RandomWalk
# Rastgele bir yürüyüş yap.
rw = RandomWalk()❶
rw.fill_walk()
# Yürüyüşteki noktaları çiz.
plt.style.use('classic')
fig, ax = plt.subplots()
ax.scatter(rw.x_values, rw.y_values, s=15)❷
plt.show()
```

rw_visual.py

`pyplot` ve `RandomWalk`'u içe aktararak başlıyoruz. Daha sonra ❶'de rastgele bir yürüyüş oluşturuyoruz ve bu yürüyüşü `rw`'de saklıyoruz ve `fill_walk()`'u çağırdığımıza emin oluyoruz. ❷'de yürüyüşün x ve y değerlerini `scatter()`'a besliyoruz ve uygun bir nokta büyüklüğü seçiyoruz. Şekil 15.9 sonuçta oluşan 5000 noktalı grafiği göstermektedir. (Bu kısımdaki resimler Matplotlib'in görüntüleyicisini dahil etmemektedir ancak *rw_visual.py*'i çalıştırdığınızda görüntüleyiciyi görmeye devam edeceksiniz.)



Şekil 15.9: *5000 noktalı rastgele bir yürüyüş.*

Her bir tarih için en yüksek sıcaklığı seçip aldık ve her bir değeri bir listede sakladık. Şimdi bu verinin görselleştirmesini oluşturalım.

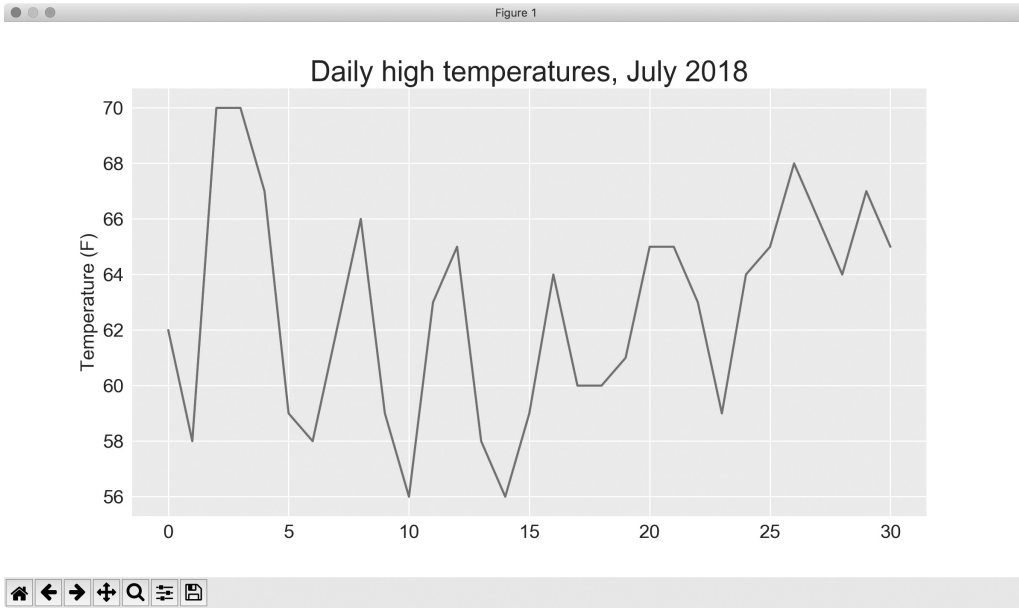
16.1.4 Bir Sıcaklık Çizelgesinde Veriyi Çizdirmek

Sahip olduğumuz sıcaklık verisini görselleştirmek için aşağıda gösterildiği gibi ilk olarak Matplotlib kullanarak günlük en yüksek sıcaklıkların basit bir grafiğini oluşturacağız:

```
import csv

import matplotlib.pyplot as plt

filename = 'data/sitka_weather_07-2018_simple.csv'
with open(filename) as f:
  -kesinti-

# En yüksek sıcaklıkları çizdir.
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(highs, c='red')❶

# Grafiği formatla
plt.title("Daily high temperatures, July 2018",
          fontsize=24)❷
plt.xlabel('', fontsize=16)❸
plt.ylabel("Temperature (F)", fontsize=16)
plt.tick_params(axis='both', which='major',
                labelsize=16)

plt.show()
```

sitka_highs.py

❶'de en yüksekleri `plot()`'a gönderiyoruz ve noktaları kırmızı olarak çizdirmek için `c = 'red'` gönderiyoruz. (En yüksekleri kırmızı, en düşükleri maviyle çizdireceğiz.) Daha sonra ❷'de bölüm 15'ten bilmeniz gerektiği üzere başlık, yazı tipi büyüklüğü ve etiketler gibi diğer birkaç formatlama ayrıntılarını belirtiyoruz. Daha tarihleri ekleyeceğimizden x eksenine etiket vermeyeceğiz ancak ❸'te varsayılan etiketleri daha okunur yapmak için `plt.xlabel()`, yazı tipi büyüklüğünü değiştirmektedir. Şekil 16.1 sonuçta oluşan grafiği göstermektedir: Temmuz 2018 Sitka, Alaska için en yüksek sıcaklıkların basit bir çizgi grafiği.



Şekil 16.1: *Temmuz 2018 Sitka, Alaska için günlük en yüksek sıcaklıkları gösteren bir çizgi grafiği*

16.1.5 `datetime` Modülü

Daha kullanışlı hâle getirmek için grafiğimize tarihler ekleyelim. Hava durumu veri dosyasındaki ilk tarih, dosyanın ikinci satırındadır:

```
"USW00025333","SITKA AIRPORT,AK US","2018-07-01","0.25",,"62","50"
```

Veri dizgi olarak okunacaktır, dolayısıyla `"2018-07-01"` dizgisini bu tarihi temsil eden bir nesneye dönüştürmenin bir yoluna ihtiyacımız vardır. `datetime` modülünden `strptime()` metodunu kullanarak 1 Temmuz 2018'i temsil eden bir nesne oluşturabiliriz. Terminal oturumunda `strptime()`'ın nasıl çalıştığını görelim:

```
>>> from datetime import datetime
>>> first_date=datetime.strptime('2018-07-01','%Y-%m-%d')
>>> print(first_date)
2018-07-01 00:00:00
```

İlk olarak `datetime` modülünden `datetime` sınıfını içe aktarıyoruz. Daha sonra ilk argümanı olarak çalışmak istediğimiz tarihi içeren dizgiyi kullanarak `strptime()` metodunu çağırıyoruz. İkinci argüman Python'a tarihin nasıl formatlandığını söyler. Bu örnekte Python, `'%Y'` argümanını dizginin ilk tireden önceki kısmının dört basamaklı bir yıl olduğu şeklinde yorumlar. `'%m-'`

```
import requests

from plotly.graph_objs import Bar❶
from plotly import offline

# API çağrısı yap ve yanıtı sakla.
url='https://api.github.com/search/repositories?q=language:python&sort=stars'
headers = 'Accept': 'application/vnd.github.v3+json'
r = requests.get(url, headers=headers)
print(f"Status code: r.status_code")❷

# Sonuçları işle.
response_dict = r.json()
repo_dicts = response_dict['items']
repo_names, stars = [], []❸
for repo_dict in repo_dicts:
  repo_names.append(repo_dict['name'])
  stars.append(repo_dict['stargazers_count'])

# Görselleştirme yap.
data = [{❹
    'type': 'bar',
    'x': repo_names,
    'y': stars,
}]

my_layout = {❺
    'title': 'Most-Starred Python Projects on GitHub',
    'xaxis': {'title': 'Repository'},
    'yaxis': {'title': 'Stars'},
}

fig = {'data': data, 'layout': my_layout}
offline.plot(fig, filename='python_repos.html')
```

python_repos_visual.py

❶'de `plotly`'den Bar (ÇN: Çubuk) sınıfını ve `offline` (ÇN: Çevrim dışı) modülünü içe aktarıyoruz. `Layout` sınıfını içe aktarmamıza gerek yoktur, çünkü Bölüm 16'daki deprem haritalama projesinde `data` listesinde yaptığımız gibi düzeni tanımlamak için sözlük yaklaşımını kullanacağız. ❷'de API çağrısı yanıtının durumunu ekrana yazdırmaya devam ediyoruz. Böylece bir sorun olup olmadığını bileceğiz. Ayrıca API yanıtını işleyen kodun bir kısmını siliyoruz çünkü artık keşif evresinde değiliz. İstediğimiz veriye sahip olduğumuzu biliyoruz.

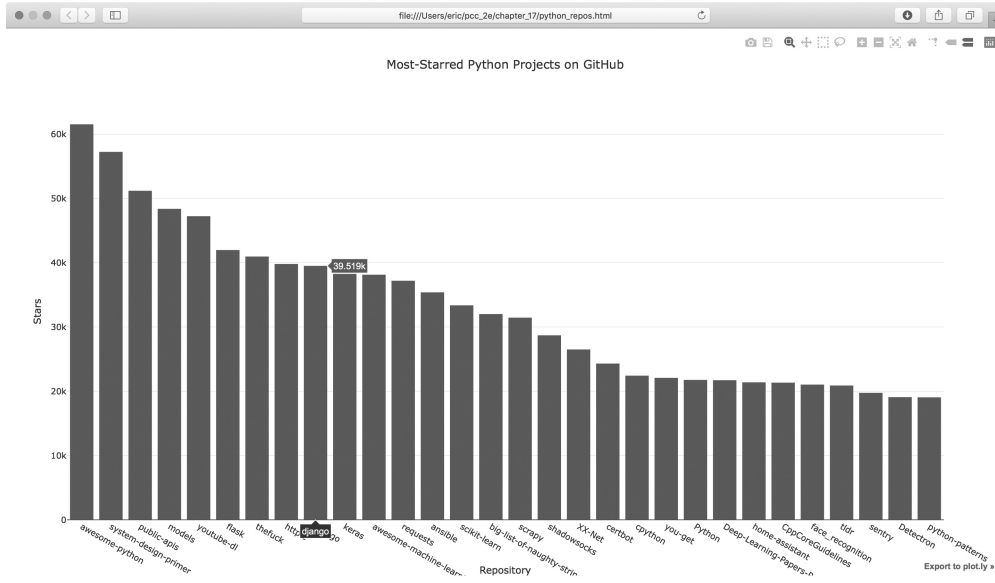
❸'te ilk çizelgeye dahil edeceğimiz verileri saklamak için iki boş liste oluşturuyoruz. Çubukları etiketlemek için her bir projenin ismine ve çubukların

yüksekliğini belirlemek için yıldızların sayısına ihtiyacımız vardır. Döngüde her bir projenin ismini ve sahip olduğu yıldız sayısını bu listelerin sonuna ekliyoruz.

Daha sonra ❹'te `data` listesini tanımlıyoruz. Bu liste Bölüm 16'da kullandığımıza benzer bir sözlük içerir. Bu sözlük grafiğin tipini tanımlar ve `x` ve `y` değerleri için veri sağlar. `x` değerleri projelerin isimleridir ve `y` değerleri her bir projeye verilen yıldız sayısıdır.

❺'te sözlük yaklaşımını kullanarak bu çizelge için düzeni tanımlıyoruz. `Layout` sınıfının bir örneğini oluşturmak yerine kullanmak istediğimiz düzen spesifikasyonları ile bir sözlük oluşturuyoruz. Çizelgenin geneli için bir başlık belirliyor ve her bir eksen için bir etiket tanımlıyoruz.

Şekil 17.1 sonuçta oluşan çizelgeyi göstermektedir. İlk birkaç projenin geri kalandan önemli ölçüde daha popüler olduğunu görebiliyoruz ancak projelerin hepsi Python ekosisteminde önemli projelerdir.



Şekil 17.1: *GitHub'ta en çok yıldız alan Python projeleri*

17.2.1 Plotly Çizelgelerini Geliştirmek

Çizelgenin stilini geliştirelim. Bölüm 16'da gördüğünüz gibi bütün stil verme yönergelerini `data` ve `my_layout` sözlüklerine anahtar-değer çiftleri olarak kaydedebilirsiniz.

`data` nesnesine yapılan değişiklikler çubukları etkiler. Aşağıda her bir çubuk için bize belirli bir renk ve net bir kenarlık veren, çizelgemiz için `data` nesnesinin değiştirilmiş bir versiyonu bulunmaktadır:

```
--kesinti--
data = [{
    'type': 'bar',
    'x': repo_names,
    'y': stars,
    'marker': {
        'color': 'rgb(60, 100, 150)',
        'line': {'width': 1.5, 'color': 'rgb(25, 25, 25)
            '}
    },
    'opacity': 0.6,
}]
--kesinti--
```

python_repos_visual.py

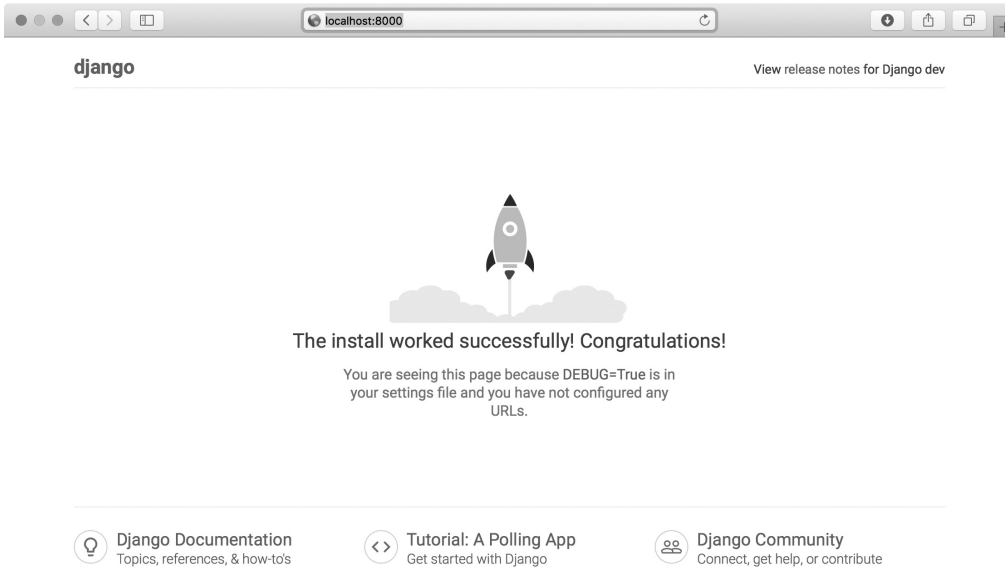
Burada gösterilen `marker` ayarları çubukların tasarımını etkiler. Çubuklar için özel bir mavi renk belirleriz ve 1.5 piksel genişliğinde koyu gri çizgiyle ana hatlarının belirleneceğini belirtiyoruz. Ayrıca çizelgenin görünümünü biraz yumuşatmak için çubukların opaklığını 0.6'ya ayarlıyoruz.

Daha sonra `my_layout`'u değiştireceğiz:

```
--kesinti--
my_layout = {
    'title': 'Most-Starred Python Projects on GitHub',
    'titlefont': {'size': 28},❶
    'xaxis': {❷
        'title': 'Repository',
        'titlefont': {'size': 24},
        'tickfont': {'size': 14},
    },
    'yaxis': {❸
        'title': 'Stars',
        'titlefont': {'size': 24},
        'tickfont': {'size': 14},
    },
}
--kesinti--
```

python_repos_visual.py

❶'de genel çizelge başlığının yazı tipini tanımlamak için `'titlefont'` anahtarını kullanıyoruz. ❷'de `'xaxis'` sözlüğünde x ekseni başlığının (`'titlefont'`) ve çentik etiketlerinin (`'tickfont'`) yazı tipi büyüklüğünü kontrol etmek için ayarlar ekliyoruz. Bunlar ayrı ayrı iç içe geçmiş tekil sözlükler olduğundan



Şekil 18.1: *Şu ana kadar her şey çalışıyor*

Kendiniz Deneyin

18-1. Yeni Projeler: Django'nun ne yaptığıyla ilgili daha iyi bir fikir elde etmek için birkaç tane boş proje oluşturunuz ve Django'nun ne oluşturduğuna bakınız. *snap_gram* veya *insta_chat* gibi basit bir ismi olan yeni bir klasör oluşturunuz (*learning_log* dizininin dışında) ve terminalde bu klasöre gidiniz ve sanal bir ortam oluşturunuz. Django'yu kurunuz ve `django-admin.py startproject snap_gram .` komutunu çalıştırınız (komutun sonuna noktayı eklediğinizden emin olunuz).

Komutun oluşturduğu dosya ve klasörlere bakınız ve bu dosya ve klasörleri Öğrenme Günlüğüyle karşılaştırınız. Django'nun yeni bir proje başlattığında ne oluşturduğuna aşina olana kadar bunu birkaç kez yapınız. Daha sonra eğer isterseniz proje dizinlerini siliniz.

18.2 Bir Uygulamayı Başlatmak

Bir Django projesi, projenin bir bütün olarak çalışmasını sağlamak için birlikte çalışan bir grup tekil *uygulama* olarak düzenlenir. Şimdilik, projemizin çalışmasının çoğunu yerine getirmek için sadece bir uygulama oluşturacağız. Bölüm 19'da kullanıcı hesaplarını yönetmek için bir başka uygulama ekleyeceğiz.

```
(ll_env)learning_log$ python manage.py createsuperuser
Username (leave blank to use 'eric'): ll_admin ❶
Email address:❷
Password:❸
Password (again):
Superuser created successfully.
(ll_env)learning_log$
```

❶'de `createsuperuser` (ÇN: superuser oluştur) komutunu verdiğinizde Django superuser için bir kullanıcı ismi girmenizi ister. Burada *ll_admin* kullanıyorum ama siz istediğiniz kullanıcı ismini girebilirsiniz. ❷'de eğer isterseniz bir e-posta adresi girebilir veya bu alanı boş bırakabilirsiniz. ❸'te şifrenizi iki kere girmeniz gerekecektir.

Not

Bazı hassas bilgiler bir sitenin yöneticilerinden gizlenebilir. Örneğin Django girdiğiniz şifreyi saklamaz, bunun yerine hash denilen, şifreden türetilmiş bir dizgiyi saklar. Şifrenizi her girdiğinizde Django girişi hashler ve saklanan hash ile karşılaştırır. Eğer iki hash eşleşirse kimliğiniz doğrulanmıştır. Hash'lerin eşleşmesini gerektirmesinin sebebi eğer bir saldırgan bir sitenin veri tabanına erişirse veri tabanının saklanmış hashlerini okuyabilir ancak şifreleri okuyamaz. Site doğru bir şekilde kurulmuş ise hash'lerden orijinal şifreleri elde etmek neredeyse imkânsızdır.

Bir Modeli Admin Sitesine Kaydetmek

Django, `User` ve `Group` gibi bazı modelleri admin sitesine otomatik olarak dahil eder ancak bizim oluşturduğumuz modellerin elle eklenmesi gerekir.

`learning_logs` uygulamasını başlattığımızda Django, *models.py* ile aynı dizinde bir *admin.py* dosyası oluşturdu. *admin.py* dosyasını açınız:

```
from django.contrib import admin

# Modellerinizi burada kaydediniz.
```

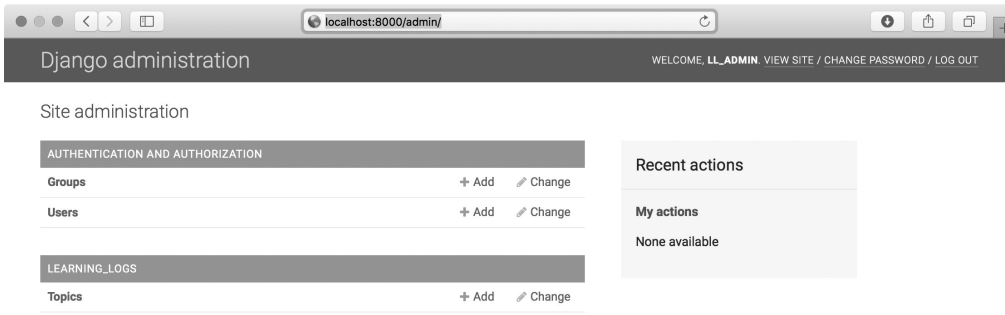
admin.py

`Topic`'i admin sitesine kaydetmek için aşağıdakini giriniz:

```
from django.contrib import admin

from .models import Topic❶

admin.site.register(Topic)❷
```

Bu kod ❶'de ilk olarak kaydetmek istediğimiz model olan `Topic`'i içe aktarır. `models`'in önündeki nokta Django'ya *models.py*'yi *admin.py* ile aynı dizinde aramasını söyler. ❷'de `admin.site.register()` kodu Django'ya modelimizi admin sitesi vasıtasıyla yönetmesini söyler.

Şimdi admin sitesine erişmek için superuser hesabını kullanın. ***http://localhost:8000/admin/*** adresine gidiniz ve az önce oluşturduğunuz superuser için kullanıcı ismi ve şifreyi giriniz. Şekil 18.2'deki gibi bir ekran görmeniz gerekmektedir. Bu sayfa, yeni kullanıcılar ve gruplar eklemenizi ve mevcut olanları değiştirmenizi sağlar. Ayrıca az önce tanımladığımız Topic modeliyle ilgili verilerle de çalışabilirsiniz.



Şekil 18.2: *Topic dahil olmak üzere admin sitesi*

Not

Tarayıcınızda web sayfasının mevcut olmadığına dair bir mesaj görürseniz Django sunucusunun terminal penceresinde çalışır vaziyette olduğundan emin olunuz. Eğer çalışmıyorsa sanal bir ortamı aktif hâle getiriniz ve `python manage.py runserver` *komutunu tekrar veriniz. Geliştirme sürecinde herhangi bir aşamada projenizi görme sorunu yaşıyorsanız açık terminalleri kapatmak ve* `runserver` *komutunu tekrar vermek iyi bir sorun gidermenin ilk adımıdır.*

Konular Eklemek

`Topic`, admin sitesine kayıt edildiğine göre ilk konumuzu ekleyelim. Topics sayfasına gitmek için Topics'e tıklayınız. Bu sayfa çoğunlukla boştur çünkü henüz yöneteceğimiz bir konu yoktur. Add Topic'e tıklayınız. Yeni bir konu eklemek için bir form görünecektir. İlk kutuya `Chess` (ÇN: Satranç) giriniz ve Save'e (ÇN: Kaydet) tıklayınız. Topics admin sayfasına geri gönderileceksiniz ve az önce oluşturduğunuz konuyu göreceksiniz.

Üzerinde çalışacağımız daha çok veri olsun diye ikinci bir konu ekleyelim. Add Topic'e tekrar tıklayınız ve `Rock Climbing`'i (ÇN: Kayalara Tırmanma) ekleyiniz. Save'e tıklayınız. Tekrardan Topics sayfasına geri gönderileceksiniz. Şimdi Chess ve Rock Climbing'i listelenmiş olarak göreceksiniz.

18.2.4 Girdi Modelini Tanımlamak

Bir kullanıcının satranç ve kaya tırmanışı hakkında öğrendiklerini kaydetmesi için kullanıcıların öğrenme günlüklerinde yapabilecekleri girdi türleri için bir model tanımlamamız gerekmektedir. Her bir girdinin belirli bir konuyla ilişkili olması gerekmektedir. Bu ilişkiye *çoktan bire ilişki* denir ve çok sayıda girdinin bir konuyla ilişkili olabileceği anlamına gelir.

İşte `Entry` (ÇN: Girdi) modeli için olan kod. Bu kodu *models.py* dosyanıza yerleştiriniz:

```
from django.db import models

class Topic(models.Model):
  --kesinti--

class Entry(models.Model):❶
  """Konu hakkında öğrenilen özel bir şey."""
  topic = models.ForeignKey(Topic,
          on_delete=models.CASCADE)❷
  text = models.TextField()❸
  date_added = models.DateTimeField(auto_now_add=True)

class Meta:❹
  verbose_name_plural = 'entries'

  def __str__(self):
    """Modelin dizgi gösterimini döndür."""
    return f"{self.text[:50]}...❺
```

models.py

❶'de `Topic`'in yaptığı gibi `Entry` Sınıfı Django'nun temel `Model` sınıfından miras alır. ❷'de ilk nitelik olan `topic` bir `ForeignKey` (ÇN: İkincil Anahtar) örneğidir. *İkincil anahtar* bir veri tabanı terimidir. Veri tabanındaki bir başka kayıta referanstır. Bu, her bir girdiyi belirli bir konuya bağlayan koddur. Her bir konuya oluşturulduğunda bir anahtar veya ID atanır. Django'nun iki veri parçası arasında bir bağlantı kurması gerektiğinde, her bir bilgi parçasıyla ilişkili olan anahtarı kullanır. Bu bağlantıları kısaca belirli bir konuyla ilişkili bütün girdileri bulup getirmek için kullanacağız. `on_delete=models.CASCADE`

Dizin