



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 çekiyor.”

— 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 alıştırmayı 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	xvii
Teşekkür	xx
Giriş	xxi
I TEMELLER	1
1 BAŞLANGIÇ	3
1.1 Programlama Ortamınızı Kurmak	3
1.1.1 Python Versiyonları	3
1.1.2 Python Kod Parçacıklarını Çalıştırmak	4
1.1.3 Sublime Text Metin Düzenleyicisi Hakkında	4
1.2 Farklı İşletim Sistemleri Üzerinde Python	5
1.2.1 Windows Üzerinde Python	5
1.2.2 macOS Üzerinde Python	7
1.2.3 Linux Üzerinde Python	9
1.3 Hello World Programını Çalıştırmak	10
1.3.1 Doğru Python Versiyonunu Kullanması için Sublime Text'i Yapılandırmak	11
1.3.2 <i>hello_world.py</i> 'yi Çalıştırmak	11
1.4 Sorun Giderme	12
1.5 Python Programlarını Terminalden Çalıştırmak	13
1.5.1 Windows Üzerinde	13
1.5.2 Linux ve macOS üzerinde	14
1.6 Özet	15
2 DEĞİŞKENLER VE BASİT VERİ TİPLERİ	16
2.1 <i>hello_world.py</i> 'yi Çalıştırdığımızda Gerçekten Ne Oluyor?	16
2.2 Değişkenler	17

2.2.1	Değişkenlere İsim Verip Kullanmak	18
2.2.2	Değişkenleri Kullanırken İsim Hatalarından Kaçınmak . .	19
2.2.3	Değişkenler Etiketlerdir	20
2.3	Dizgiler	21
2.3.1	Metotlarla Bir Dizgide Büyük/Küçük Harf Değiştirmek .	21
2.3.2	Dizgilerde Değişkenleri Kullanmak	23
2.3.3	Sekmeler ve Yeni Satırlar ile Dizgilere Beyaz Boşluk Eklemek	24
2.3.4	Beyaz Boşluğu Ayıklamak	25
2.3.5	Dizgilerde Söz Dizim Hatalarından Kaçınmak	26
2.4	Sayılar	28
2.4.1	Tam Sayılar	29
2.4.2	Kayan Noktalı Sayılar	29
2.4.3	Tam Sayılar ve Kayan Noktalı Sayılar	30
2.4.4	Sayılarda Alt Tire	31
2.4.5	Çoklu Atama	31
2.4.6	Sabitler	31
2.5	Açıklamalar	32
2.5.1	Açıklamaları Nasıl Yazarsınız?	32
2.5.2	Ne Türden Açıklamalar Yazmalısınız?	32
2.6	Python'un Zen'i	33
2.7	Özet	35

3 LİSTELERE GİRİŞ 36

3.1	Liste Nedir?	36
3.1.1	Bir Listedeki Elemanlara Erişmek	37
3.1.2	İndeks Konumları 1'den Değil 0'dan Başlar	37
3.1.3	Bir Listedeki Elemanları Tek Tek Değerler Kullanmak	38
3.2	Elemanları Değiştirmek, Eklemek ve Silmek	39
3.2.1	Bir Listedeki Elemanları Değiştirmek	39
3.2.2	Bir Listeye Elemanlar Eklemek	40
3.2.3	Bir Listedeki Elemanları Silmek	41
3.3	Bir Listeyi Düzenlemek	46
3.3.1	sort() Metoduyla Bir Listeyi Kalıcı Olarak Sıralamak .	47
3.3.2	sorted() Fonksiyonu ile Bir Listeyi Geçici Olarak Sıralamak	47
3.3.3	Bir Listeyi Ters Sıralamada Yazdırmak	48
3.3.4	Bir Listenin Uzunluğunu Bulmak	49
3.4	Listelerle Çalışırken İndeks Hatalarından Kaçınmak	50
3.5	Özet	52

4	LİSTELERLE ÇALIŞMAK	53
4.1	Listenin Tamamı Üzerinden Döngü Kurmak	53
4.1.1	Döngü Kurmaya Yakından Bakış	54
4.1.2	for Döngüsünün İçinde Daha Fazla İş Yapmak	55
4.1.3	for Döngüsünden Sonra Bir Şeyler Yapmak	56
4.2	Girinti Hatalarından Kaçınmak	57
4.2.1	Girinti Vermeyi Unutmak	57
4.2.2	İlave Satırlara Girinti Vermeyi Unutmak	58
4.2.3	Gereksiz Yere Girinti Vermek	59
4.2.4	Döngüden Sonra Gereksiz Yere Girinti Vermek	59
4.2.5	İki Nokta Üst Üste İşaretini Unutmak	60
4.3	Sayısal Listeler Yapmak	61
4.3.1	range() Fonksiyonunu Kullanmak	61
4.3.2	Sayı Listesi Oluşturmak için range()’i Kullanmak	62
4.3.3	Bir Sayı Listesiyle Basit İstatistikler	64
4.3.4	Hesaplanmış Listeler	64
4.4	Bir Listenin Bir Kısmı ile Çalışmak	66
4.4.1	Bir Listeyi Dilimlemek	66
4.4.2	Dilim Üzerinde Döngü Kurmak	68
4.4.3	Bir Listeyi Kopyalamak	68
4.5	Demetler	72
4.5.1	Bir Demeti Tanımlamak	72
4.5.2	Bir Demetteki Bütün Değerler Üzerinden Döngü Kurmak	73
4.5.3	Bir Demetin Üzerine Yazmak	73
4.6	Kodunuza Stil Vermek	74
4.6.1	Stil Kılavuzu	75
4.6.2	Girinti	75
4.6.3	Satır Uzunluğu	76
4.6.4	Boş Satırlar	76
4.6.5	Diğer Stil Kılavuzları	76
4.7	Özet	77
5	IF İFADELERİ	78
5.1	Basit Bir Örnek	78
5.2	Koşullu Testler	79
5.2.1	Eşitliği Kontrol Etmek	79
5.2.2	Eşitliği Kontrol Ederken Büyük Küçük Harfleri Görmezden Gelmek	80
5.2.3	Eşitsizliği Kontrol Etmek	81
5.2.4	Sayısal Karşılaştırmalar	81
5.2.5	Çoklu Koşulları Kontrol Etmek	82

5.2.6	Bir Değerin Bir Listede Olduğunu Kontrol Etmek	84
5.2.7	Bir Değerin Bir Listede Olmadığını Kontrol Etmek	84
5.2.8	Boole Deyimleri	85
5.3	if İfadeleri	86
5.3.1	Basit if İfadeleri	86
5.3.2	if-else İfadeleri	87
5.3.3	if-elif-else Zinciri	88
5.3.4	Çoklu elif Blokları Kullanmak	89
5.3.5	else Bloğunu Kaldırmak	90
5.3.6	Çoklu Koşulları Test Etmek	91
5.4	Listelerle if İfadeleri Kullanmak	94
5.4.1	Özel Öğeleri Kontrol Etmek	94
5.4.2	Bir Listenin Boş Olmadığını Kontrol Etmek	95
5.4.3	Çoklu Listeler Kullanmak	96
5.5	if İfadelerine Stil Vermek	98
5.6	Özet	99
6	SÖZLÜKLER	100
6.1	Basit Bir Sözlük	100
6.2	Sözlüklerle Çalışmak	101
6.2.1	Bir Sözlükteki Değerlere Erişmek	101
6.2.2	Yeni Anahtar-Değer Çiftleri Ekleme	102
6.2.3	Boş Bir Sözlükle Başlamak	103
6.2.4	Bir Sözlükteki Değerleri Değiştirmek	104
6.2.5	Anahtar-Değer Çiftlerini Silme	105
6.2.6	Benzer Nesnelerin Sözlüğü	106
6.2.7	Değerlere Erişmek için get ()'i Kullanmak	107
6.3	Sözlük Üzerinden Döngü Kurmak	109
6.3.1	Bütün Anahtar-Değer Çiftleri Üzerinden Döngü Kurmak	109
6.3.2	Bir Sözlükteki Bütün Anahtarlar Üzerinden Döngü Kurmak	111
6.3.3	Bir Sözlüğün Anahtarları Üzerinden Belirli Bir Sıralamada Döngü Kurmak	113
6.3.4	Bir Sözlükteki Bütün Değerler Üzerinden Döngü Kurmak	114
6.4	İç İçte Yerleştirme	116
6.4.1	Sözlüklerin Bir Listesi	116
6.4.2	Sözlük İçerisinde Liste	119
6.4.3	Sözlük İçerisinde Sözlük	121
6.5	Özet	123

7	KULLANICI GİRDİSİ VE WHILE DÖNGÜLERİ	124
7.1	input() Fonksiyonu Nasıl Çalışır	125
7.1.1	Anlaşılır İstemler Yazmak	125
7.1.2	Sayısal Girdi Kabul Etmek için int()’i Kullanmak	126
7.1.3	Modulo Operatörü	128
7.2	while Döngülerine Giriş	129
7.2.1	while Döngüsü Eylem Hâlinde	129
7.2.2	Kullanıcının Ne Zaman Çıkacağını Seçmesine İzin Verme	130
7.2.3	Bayrak Kullanmak	132
7.2.4	Döngüden Çıkmak için break’i Kullanmak	133
7.2.5	Döngü İçerisinde continue Kullanmak	134
7.2.6	Sonsuz Döngülerden Kaçınmak	135
7.3	Listelerde ve Sözlüklerde while Döngüsü Kullanmak	137
7.3.1	Öğeleri Bir Listedenden Diğerine Taşımak	137
7.3.2	Belirli Değerlerin Bütün Örneklerini Bir Listedenden Silmek	138
7.3.3	Bir Sözlüğü Kullanıcı Girdisiyle Doldurmak	139
7.4	Özet	141
8	FONKSİYONLAR	142
8.1	Bir Fonksiyonu Tanımlamak	142
8.1.1	Bir Fonksiyona Bilgi Göndermek	143
8.1.2	Argümanlar ve Parametreler	144
8.2	Argümanlar Göndermek	145
8.2.1	Konumsal Argümanlar	145
8.2.2	Çoklu Fonksiyon Çağırma	146
8.2.3	Konumsal Argümanlarda Sıralama Önemlidir	146
8.2.4	Anahtar Kelime Argümanlar	147
8.2.5	Varsayılan Değerler	148
8.2.6	Eş Değer Fonksiyon Çağruları	149
8.2.7	Argüman Hatalarından Kaçınmak	150
8.3	Döndürülen Değerler	152
8.3.1	Basit Bir Değer Döndürmek	152
8.3.2	Bir Argümanı İsteğe Bağlı Hâle Getirmek	153
8.3.3	Sözlük Döndürmek	154
8.3.4	while Döngüsünde Fonksiyon Kullanmak	155
8.4	Liste Göndermek	158
8.4.1	Fonksiyon İçerisinde Bir Listeyi Değiştirmek	158
8.4.2	Bir Fonksiyonun Bir Listeyi Değiştirmesini Önlemek	161
8.5	Keyfi Sayıda Argüman Göndermek	162
8.5.1	Konumsal ve Keyfi Argümanları Karıştırmak	164
8.5.2	Keyfi Anahtar Kelime Argümanlar Kullanmak	164

8.6	Fonksiyonlarımızı Modüllerde Saklamak	167
8.6.1	Bir Modülün Tamamını İçe Aktarmak	167
8.6.2	Belirli Fonksiyonları İçeri Aktarmak	168
8.6.3	Bir Fonksiyona Takma İsim Vermek için <code>as</code> Kullanmak	169
8.6.4	Bir Modüle Takma İsim Vermek için <code>as</code> Kullanmak	169
8.6.5	Bir Modüldeki Bütün Fonksiyonları İçe Aktarmak	170
8.7	Fonksiyonlara Stil Vermek	171
8.8	Özet	172
9	SINIFLAR	174
9.1	Bir Sınıfı Oluşturmak ve Kullanmak	175
9.1.1	<code>Dog</code> Sınıfını Oluşturmak	175
9.1.2	<code>__init__</code> () Metodu	176
9.1.3	Bir Sınıftan Örnek Oluşturmak	177
9.2	Sınıflar ve Örneklerle Çalışmak	180
9.2.1	<code>Car</code> Sınıfı	180
9.2.2	Bir Niteliğe Varsayılan Değer Vermek	181
9.2.3	Nitelik Değerlerini Değiştirmek	182
9.3	Kalıtım	186
9.3.1	Alt Sınıfın <code>__init__</code> () Metodu	186
9.3.2	Alt Sınıf için Nitelik ve Metotlar Tanımlamak	187
9.3.3	Üst Sınıftaki Metotları Geçersiz Kılmak	189
9.3.4	Nitelikler Olarak Örnekler	189
9.3.5	Gerçek Dünyaya Ait Nesnelere Modellemek	191
9.4	Sınıfları İçeri Aktarmak	193
9.4.1	Tek Bir Sınıfı İçeri Aktarmak	193
9.4.2	Bir Modülde Birden Çok Sınıf Saklamak	195
9.4.3	Bir Modülden Birden Çok Sınıfı İçe Aktarmak	196
9.4.4	Bir Modülün Tamamını İçeri Aktarmak	197
9.4.5	Bir Modüldeki Sınıfların Tamamını İçe Aktarmak	197
9.4.6	Bir Modülü Bir Başka Modüle Aktarmak	198
9.4.7	Takma İsimler Kullanmak	199
9.4.8	Kendi İş Akışımızı Bulmak	199
9.5	Python Standart Kütüphanesi	200
9.6	Sınıflara Stil Vermek	202
9.7	Özet	202
10	DOSYALAR VE ÖZEL DURUMLAR	203
10.1	Bir Dosyayı Okumak	204
10.1.1	Dosyanın Tamamını Okumak	204
10.1.2	Dosya Yolları	206

10.1.3	Satır Satır Okumak	207
10.1.4	Bir Dosyadan Satırlar Listesi Oluşturmak	208
10.1.5	Bir Dosyanın İçeriğiyle Çalışmak	209
10.1.6	Büyük Dosyalar: Bir Milyon Basamak	210
10.1.7	Doğum Gününüz Pi Sayısının İçinde mi?	211
10.2	Bir Dosyaya Yazmak	212
10.2.1	Boş Bir Dosyaya Yazmak	212
10.2.2	Birden Çok Satırı Yazmak	213
10.2.3	Bir Dosyanın Sonuna Ekleme	214
10.3	Özel Durumlar (Exceptions)	215
10.3.1	ZeroDivisionError Özel Durumunu Ele Almak	216
10.3.2	<code>try-except</code> Blokları Kullanmak	216
10.3.3	Çökmeleri Engellemek için Özel Durumları Kullanmak	217
10.3.4	<code>else</code> Bloğu	218
10.3.5	FileNotFoundError Özel Durumunu Ele Almak	219
10.3.6	Metni Analiz Etmek	220
10.3.7	Birden Çok Dosyayla Çalışmak	221
10.3.8	Sessizce Başarısız Olmak	223
10.3.9	Hangi Hataların Bildirileceğine Karar Vermek	224
10.4	Veriyi Saklamak	225
10.4.1	<code>json.dump()</code> ve <code>json.load()</code> 'u Kullanmak	226
10.4.2	Kullanıcı Tarafından Üretilen Veriyi Kaydetmek ve Okumak	227
10.4.3	Yeniden Yapılandırma	229
10.5	Özet	232

11 KODUNUZU TEST ETMEK 233

11.1	Bir Fonksiyonu Test Etmek	233
11.1.1	Birim Testleri ve Test Senaryoları	235
11.1.2	Başarılı Bir Test	235
11.1.3	Başarısız Bir Test	237
11.1.4	Başarısız Bir Teste Yanıt Vermek	238
11.1.5	Yeni Testler Ekleme	239
11.2	Bir Sınıfı Test Etmek	241
11.2.1	Assert Metot Çeşitleri	241
11.2.2	Test Edilecek Bir Sınıf	242
11.2.3	AnonymousSurvey Sınıfını Test Etmek	244
11.2.4	<code>setUp()</code> Metodu	246
11.3	Özet	248

II PROJELER	250
PROJE 1: UZAYLI İSTİLASI	252
12 MERMİ ATAN BİR GEMİ	253
12.1 Projenizi Planlamak	254
12.2 Pygame’i Kurmak	254
12.3 Oyun Projesine Başlamak	255
12.3.1 Bir Pygame Penceresi Oluşturmak ve Kullanıcı Girdisine Yanıt Vermek	255
12.3.2 Arka Plan Rengini Ayarlamak	257
12.3.3 Settings Sınıfını Oluşturmak	258
12.4 Gemi Resmini Ekleme	259
12.4.1 Gemi Sınıfını Oluşturmak	261
12.4.2 Gemiyi Ekranı Çizdirmek	262
12.5 Yeniden Yapılandırma Metotları	264
12.5.1 <code>_check_events()</code> Metodu	264
12.5.2 <code>_update_screen()</code> Metodu	265
12.6 Gemiyi Sürmek	266
12.6.1 Klavye Tuşuna Basmaya Yanıt Vermek	266
12.6.2 Sürekli Hareketi Sağlamak	267
12.6.3 Hem Sola Hem Sağa Hareket Etmek	269
12.6.4 Geminin Hızını Ayarlamak	270
12.6.5 Geminin Azami Mesafesini Sınırlamak	272
12.6.6 <code>_check_events()</code> ’i Yeniden Yapılandırmak	273
12.6.7 Çıkmak için Q’ya Basmak	274
12.6.8 Oyunu Tam Ekran Modunda Çalıştırmak	274
12.7 Hızlı Bir Özet	275
12.7.1 <i>alien_invasion.py</i>	275
12.7.2 <i>settings.py</i>	276
12.7.3 <i>ship.py</i>	276
12.8 Mermi Atmak	276
12.8.1 Mermi Ayarlarını Ekleme	277
12.8.2 Bullet Sınıfını Oluşturmak	277
12.8.3 Mermileri Bir Grupta Saklamak	279
12.8.4 Mermileri Ateşlemek	279
12.8.5 Eski Mermileri Silmek	281
12.8.6 Mermi Sayısını Sınırlamak	282
12.8.7 <code>_update_bullets()</code> Metodunu Oluşturmak	283
12.9 Özet	284

13 UZAYLILAR!	285
13.1 Projeyi Gözden Geçirmek	285
13.2 İlk Uzaylıyı Oluşturmak	286
13.2.1 Uzaylı Sınıfını Oluşturmak	286
13.2.2 Alien'in Bir Örneğini Oluşturmak	288
13.3 Uzaylı Filosunu Oluşturmak	290
13.3.1 Bir Satıra Kaç Uzaylının Sığacağını Belirlemek	290
13.3.2 Bir Uzaylı Satırı Oluşturmak	291
13.3.3 <code>_create_fleet()</code> 'i Yeniden Yapılandırmak	292
13.3.4 Satırlar Ekleme	293
13.4 Filoyu Hareket Ettirmek	296
13.4.1 Uzaylıları Sağa Hareket Ettirmek	296
13.4.2 Filonun Yönü için Ayarlar Oluşturmak	298
13.4.3 Bir Uzaylının Kenara Çarpıp Çarpmadığını Kontrol Etmek	299
13.4.4 Filoyu Düşürmek ve Yönünü Değiştirmek	299
13.5 Uzaylılara Ateş Etmek	301
13.5.1 Mermi Çarpışmalarını Tespit Etmek	301
13.5.2 Test için Daha Büyük Mermiler Oluşturmak	302
13.5.3 Filoyu Tekrardan Oluşturmak	303
13.5.4 Mermileri Hızlandırmak	304
13.5.5 <code>_update_bullets()</code> 'i Yeniden Yapılandırmak	304
13.6 Oyunu Sona Erdirmek	305
13.6.1 Uzaylı ve Gemi Çarpışmalarını Tespit Etmek	306
13.6.2 Uzaylı ve Gemi Çarpışmalarına Yanıt Vermek	307
13.6.3 Ekranın Alt Tarafına Ulaşan Uzaylılar	310
13.6.4 Oyun Bitti!	311
13.6.5 Oyunun Kısımlarının Ne Zaman Çalışacağını Belirlemek	312
13.7 Özet	313
14 SKOR VERME	315
14.1 Play Düğmesini Ekleme	315
14.1.1 Button Sınıfını Oluşturmak	316
14.1.2 Düğmeyi Ekranı Çizdirmek	318
14.1.3 Oyunu Başlatmak	319
14.1.4 Oyunu Resetlemek	320
14.1.5 Play Düğmesinin Etkinliğini Kaldırmak	321
14.1.6 Fare İmlecini Gizlemek	321
14.2 Seviyeyi Yükseltmek	322
14.2.1 Hız Ayarlarını Değiştirmek	323
14.2.2 Hızı Resetlemek	325
14.3 Skor Verme	325

14.3.1	Skoru Görüntülemek	326
14.3.2	Bir Skorbord Oluşturmak	327
14.3.3	Uzaylılar Vurulduçça Skoru Güncellemek	328
14.3.4	Skoru Resetlemek	330
14.3.5	Bütün Vuruşlara Skor Verildiğinden Emin Olmak	330
14.3.6	Puan Değerlerini Artırmak	331
14.3.7	Skoru Yuvarlamak	332
14.3.8	En Yüksek Skorlar	333
14.3.9	Seviyeyi Görüntülemek	335
14.3.10	Gemi Sayısını Görüntülemek	338
14.4	Özet	343

PROJE 2: VERİ GÖRSELLEŞTİRME **344**

15 VERİ ÜRETME **345**

15.1	Matplotlib'i Kurmak	346
15.2	Basit Bir Çizgi Grafiği Çizmek	346
15.2.1	Etiket Tipini ve Çizgi Kalınlığını Değiştirmek	348
15.2.2	Grafiği Düzeltmek	348
15.2.3	Yerleşik Stilleri Kullanmak	350
15.2.4	<code>scatter()</code> ile Tek Tek Noktaları Çizdirmek ve Stil Vermek	351
15.2.5	<code>scatter()</code> ile Bir Noktalar Serisini Çizdirmek	353
15.2.6	Veriyi Otomatik Olarak Hesaplamak	354
15.2.7	Kullanıcı Tanımlı Renkler Tanımlamak	354
15.2.8	Renk Haritası Kullanmak	355
15.2.9	Grafiklerinizi Otomatik Olarak Kaydetmek	357
15.3	Rastgele Yürüyüşler	357
15.3.1	<code>RandomWalk</code> Sınıfını Oluşturmak	357
15.3.2	Yönleri Seçmek	358
15.3.3	Rastgele Yürüyüşü Çizdirmek	359
15.3.4	Birden Çok Rastgele Yürüyüş Üretmek	361
15.3.5	Yürüyüşe Stil Vermek	361
15.4	Plotly ile Zar Atmak	366
15.4.1	Plotly'yi Kurmak	367
15.4.2	<code>Die</code> Sınıfını Oluşturmak	367
15.4.3	Zarı Atmak	368
15.4.4	Sonuçları Analiz Etmek	369
15.4.5	Bir Histogram Oluşturmak	369
15.4.6	İki Zar Atmak	371
15.4.7	Farklı Büyüklüklerde Zarlar Atmak	373
15.5	Özet	376

16 VERİYİ İNDİRMEK	377
16.1 CSV Dosya Formatı	378
16.1.1 CSV Dosya Başlıklarını Ayırıştırmak	378
16.1.2 Başlıkları ve Konumlarını Ekrana Yazdırmak	379
16.1.3 Veriyi Seçip Almak ve Okumak	380
16.1.4 Bir Sıcaklık Çizelgesinde Veriyi Çizdirmek	381
16.1.5 <code>datetime</code> Modülü	382
16.1.6 Tarihleri Çizdirmek	383
16.1.7 Daha Uzun Bir Zaman Çerçevesini Çizdirmek	384
16.1.8 İkinci Bir Veri Serisi Çizdirmek	385
16.1.9 Çizelgedeki Bir Alanı Gölgelemek	387
16.1.10 Hata Kontrolü	388
16.1.11 Kendi Verinizi İndirmek	392
16.2 Global Veri Kümelerini Haritalamak: JSON Formatı	393
16.2.1 Deprem Verisini İndirmek	394
16.2.2 JSON Verisini İncelemek	394
16.2.3 Bütün Depremlerin Listesini Yapmak	397
16.2.4 Büyüklükleri Seçip Almak	397
16.2.5 Konum Verisini Seçip Almak	398
16.2.6 Dünya Haritası Oluşturmak	399
16.2.7 Çizelge Verisini Belirlemenin Farklı Bir Yolu	400
16.2.8 İşaretleyici Büyüklüğünü Kullanıcı Tanımlı Hâle Getirmek	401
16.2.9 İşaretleyici Renklerini Kullanıcı Tanımlı Hâle Getirmek .	402
16.2.10 Diğer Renk Ölçekleri	403
16.2.11 Üzerinde Gezinilen Metin Ekleme	405
16.3 Özet	407
17 API'LERLE ÇALIŞMAK	409
17.1 Bir Web API'si Kullanmak	409
17.1.1 Git ve GitHub	409
17.1.2 API Çağrısı Kullanarak Veri İstemek	410
17.1.3 Requests'i Kurmak	411
17.1.4 Bir API Yanıtını İşlemek	411
17.1.5 Yanıt Sözlüğüyle Çalışmak	413
17.1.6 En Üst Sıralardaki Depoları Özetlemek	415
17.1.7 API Hız Sınırlarını İzlemek	416
17.2 Plotly Kullanarak Depoları Görselleştirmek	417
17.2.1 Plotly Çizelgelerini Geliştirmek	419
17.2.2 Kullanıcı Tanımlı Araç İpuçları Ekleme	421
17.2.3 Grafiğimize Tıklanabilir Bağlantılar Ekleme	422
17.2.4 Plotly ve GitHub API Hakkında Daha Fazla Bilgi	424

17.3	Hacker News API'si	424
17.4	Özet	429
PROJE 3: WEB UYGULAMALARI		430
18 DJANGOYA BAŞLAMAK		431
18.1	Bir Projeyi Hazırlamak	431
18.1.1	Spesifikasyon (Spec) Yazmak	432
18.1.2	Sanal Bir Ortam Oluşturmak	432
18.1.3	Sanal Ortamı Aktif Hâle Getirmek	433
18.1.4	Django'yu Kurmak	433
18.1.5	Django'da Bir Proje Oluşturmak	434
18.1.6	Veri Tabanını Oluşturmak	435
18.1.7	Projeyi İzlemek	436
18.2	Bir Uygulamayı Başlatmak	437
18.2.1	Modelleri Tanımlamak	438
18.2.2	Modelleri Aktif Hâle Getirmek	440
18.2.3	Django Admin Sitesi	441
18.2.4	Girdi Modelini Tanımlamak	444
18.2.5	Girdi Modelini Taşıma	445
18.2.6	Entry'yi Admin Sitesine Kayıt Etmek	445
18.2.7	Django Kabuğu	447
18.3	Sayfaları Oluşturmak: Öğrenme Günlüğü Ana Sayfası	449
18.3.1	Bir URL'yi Eşleştirmek	449
18.3.2	Görünüm Yazmak	451
18.3.3	Şablon Yazmak	452
18.4	İlave Sayfalar Oluşturmak	454
18.4.1	Şablon Kalıtımı	454
18.4.2	Konular Sayfası	456
18.4.3	Tek Tek Konu Sayfaları	459
18.5	Özet	464
19 KULLANICI HESAPLARI		465
19.1	Kullanıcıların Veri Girmesini Sağlamak	465
19.1.1	Yeni Konular Ekleme	466
19.1.2	Yeni Girdiler Ekleme	470
19.1.3	new_entry URL'si	471
19.1.4	Girdileri Düzenleme	474
19.2	Kullanıcı Hesaplarını Kurmak	479
19.2.1	users Uygulaması	479
19.2.2	Oturum Açma Sayfası	480

19.2.3	Oturumu Kapatmak	483
19.2.4	Kayıt Sayfası	484
19.2.5	<code>register()</code> Görünüm Fonksiyonu	485
19.3	Kullanıcıların Kendi Verilerine İzin Vermek	488
19.3.1	<code>@login_required</code> ile Erişimi Kısıtlamak	488
19.3.2	Belirli Kullanıcılara Verileri Bağlamak	490
19.3.3	Konulara Erişimi Uygun Kullanıcılarla Kısıtlamak	494
19.3.4	Kullanıcının Konularını Korumak	495
19.3.5	<code>edit_entry</code> Sayfasını Korumak	496
19.3.6	Yeni Konuları O Anki Kullanıcıyla İlişkilendirmek	496
19.4	Özet	498
20	UYGULAMAYA STİL VERMEK VE DAĞITMAK	499
20.1	Öğrenme Günlüğüne Stil Vermek	500
20.1.1	<code>django-bootstrap4</code> Uygulaması	500
20.1.2	Öğrenme Günlüğüne Stil Vermek için Bootstrap'i Kullanmak	501
20.1.3	<code>base.html</code> 'i Değiştirmek	501
20.1.4	Jumbotron Kullanarak Ana Sayfaya Stil Vermek	506
20.1.5	Oturum Açma Sayfasına Stil Vermek	507
20.1.6	Konular Sayfasına Stil Vermek	508
20.1.7	Konular Sayfasındaki Girdilere Stil Vermek	509
20.2	Öğrenme Günlüğünü Dağıtmak	511
20.2.1	Heroku Hesabı Oluşturmak	512
20.2.2	Heroku CLI'yi Kurmak	512
20.2.3	Gerekli Paketleri Kurmak	512
20.2.4	<code>requirements.txt</code> Dosyasını Oluşturmak	513
20.2.5	Python Çalışma Zamanını Belirlemek	513
20.2.6	Heroku için <code>settings.py</code> 'yi Değiştirmek	514
20.2.7	Süreçleri Başlatmak için Procfile Oluşturmak	514
20.2.8	Projenin Dosyalarını Takip Etmek için Git Kullanmak	515
20.2.9	Heroku'ya Yükleme	517
20.2.10	Heroku Üzerinde Veri Tabanını Kurmak	519
20.2.11	Heroku Dağıtımını Geliştirmek	520
20.2.12	Canlı Projeyi Güvenli Hâle Getirmek	522
20.2.13	Değişiklikleri Commit Etmek ve Göndermek	523
20.2.14	Heroku Üzerinde Ortam Değişkenlerini Ayarlamak	524
20.2.15	Kullanıcı Tanımlı Hata Sayfaları Oluşturmak	524
20.2.16	Devam Eden Geliştirme	527
20.2.17	<code>SECRET_KEY</code> Ayarı	528
20.2.18	Heroku Üzerinde Bir Projeyi Silmek	528

20.3 Özet	530
SON SÖZ	531
A KURULUM VE SORUN GİDERME	533
A.1 Windows Üzerinde Python	533
A.1.1 Python Yorumlayıcısını Bulmak	533
A.1.2 Python’u Path Değişkenine Ekleme	534
A.1.3 Python’u Tekrar Kurmak	535
A.2 macOS Üzerinde Python	535
A.2.1 Homebrew’i Kurmak	535
A.2.2 Python’u Kurmak	536
A.3 Linux Üzerinde Python	536
A.4 Python Anahtar Kelimeleri ve Yerleşik Fonksiyonları	537
A.4.1 Python Anahtar Kelimeleri	537
A.4.2 Python Yerleşik Fonksiyonları	537
B METİN DÜZENLEYİCİLER VE IDE’LER	539
B.1 Sublime Text Ayarlarını Kullanıcı Tanımlı Hâle Getirmek	540
B.1.1 Sekmeleri Boşluklara Dönüştürmek	540
B.1.2 Satır Uzunluğu Göstergesini Ayarlamak	540
B.1.3 Kod Bloklarına Girinti Vermek ve Girinti Kaldırmak	541
B.1.4 Kod Bloklarının Etkinliğini Açıklama Satırıyla Gidermek	541
B.1.5 Yapılandırmayı Kaydetmek	541
B.1.6 Daha Fazla Kullanıcı Tanımlı Hâle Getirme	541
B.2 Diğer Metin Düzenleyiciler ve IDE’ler	542
B.2.1 IDLE	542
B.2.2 Geany	542
B.2.3 Emacs ve Vim	542
B.2.4 Atom	543
B.2.5 Visual Studio Code	543
B.2.6 PyCharm	543
B.2.7 Jupyter Notebook’lar	543
C YARDIM ALMAK	545
C.1 İlk Adımlar	545
C.1.1 Tekrar Deneyiniz	546
C.1.2 Ara Verin	546
C.1.3 Bu Kitabın Kaynaklarına Başvurunuz	546
C.2 Çevrim İçi Arama	547
C.2.1 Stack Overflow	547

C.2.2	Resmi Python Dokümantasyonu	548
C.2.3	Resmi Kütüphane Dokümantasyonu	548
C.2.4	r/learnpython	548
C.2.5	Blog Gönderileri	548
C.3	Internet Relay Chat	548
C.3.1	IRC Hesabı Oluşturmak	549
C.3.2	Katılacağınız Kanallar	549
C.3.3	IRC Kültürü	549
C.4	Slack	550
C.5	Discord	550
D	VERSİYON KONTROLÜ İÇİN GIT KULLANMAK	551
D.1	Git'i Kurmak	551
D.1.1	Git'i Windows'ta Kurmak	552
D.1.2	Git'i macOS Üzerinde Kurmak	552
D.1.3	Git'i Linux Üzerinde Kurmak	552
D.1.4	Git'i Yapılandırmak	552
D.2	Proje Oluşturmak	552
D.3	Dosyaları Görmezden Gelmek	553
D.4	Bir Depoyu Başlatmak	553
D.5	Durumu Kontrol Etmek	553
D.6	Dosyaları Depoya Ekleme	554
D.7	Commit Etmek	555
D.8	Günlüğü Kontrol Etmek	555
D.9	İkinci Commit	556
D.10	Değişikliği Geri Almak	557
D.11	Önceki Commit'leri Alıp getirmek	558
D.12	Depoyu Silmek	560

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 işlerinde 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 getirmektir. 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 gerçekleştirildiğine dair daha doğru bir açıklama içermektedir. Değişkenler, değerler için *etiketler* olarak

tanımlanır ve bu deęişkenlerin Python'da nasıl davrandığını daha iyi anlaşılmasını sağlar. Kitap artık Python 3.6'da tanımlanan f-stings'i kullanmaktadır. Bu, dizgilerde deęişken deęerleri 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. Deęişkenlerin 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 deęerleri temsil etmek için kolay anlaşılabilir bir kural yer almaktadır.

- Bölüm 6'da bir sözlükten deęerleri bulup getirmek için `get()` metodunu tanıtıyorum. Bu metot bir anahtar olmadığı zaman varsayılan bir deęer döndürebilmektedir.
- Uzaylı İstilasası 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 aşına olan okuyucular bu sınıf tabanlı yaklaşımın sağladığı kolaylığı takdir edecektir. Pygame şimdi bütün sistemlerde tek satırda kurulabilmekte ve okuyuculara oyunu tam ekran modunda veya pencere modda çalıştırma seçeneęi 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 dizimi 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. Coęrafi 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: Tümüleşik 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 görüntülediğinde 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-dizgileri*yle pek çok şey yapabilirsiniz. Örneğin, aşağıda gösterildiği gibi bir değişkenle ilişkili bilgiyi kullanarak tam mesajlar oluşturmak için *f-dizgileri*ni 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-dizgileri*ni 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 yanıtımızda 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 sayısı ile 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 gerçekleştiğiyle 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 beklemleri gerektiğini söyleyen bir mesajı ekrana yazdırınız. Aksi takdirde masalarının hazır olduğunu bildirin.

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

7.2 while Döngülerine Giriş

`for` döngüsü bir öge koleksiyonunu alır ve koleksiyondaki her bir öge 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 kullandığımız 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 listesinden ve sözlüğünden ibaret olur. Bunların her birine sırayla bakalım:

8.2.1 Konumsal Argümanlar

Bir fonksiyonu çağırduğunuzda 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 çağırduğumuzda 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 çağırarak 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 çağırduğunuzda 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

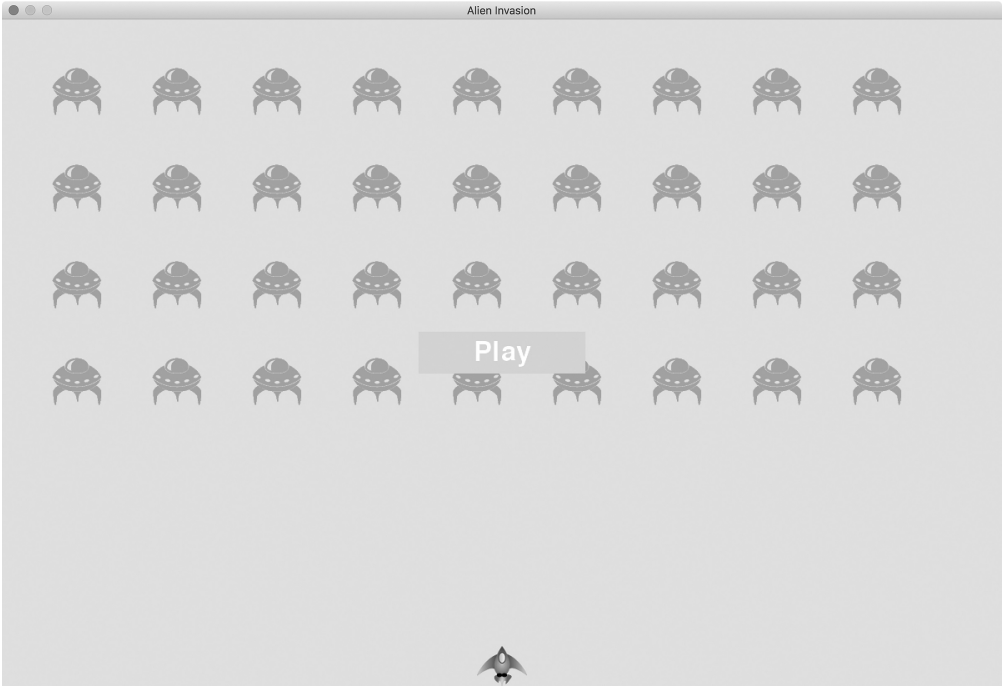
Anahtar kelime argümanı bir fonksiyona gönderdiğiniz isim-değer çiftidir. Argümanın içinde isim ile değeri doğrudan doğruya ilişkilendirdiğinizden dolayı argümanı fonksiyona gönderdiğinizde hiçbir karışıklık olmaz (Hamster isimli bir Harry söz konusu olmaz). Anahtar kelime argümanlar; sizi, fonksiyon çağrısında argümanları doğru sıralama endişesinden kurtarır ve fonksiyon çağrısında her bir değer 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 tıkladığında fare imlecinin x ve y koor-

dinatarımı içeren bir demet döndüren `pygame.mouse.get_pos()`'u kullanıyoruz. ❸'te bu değerleri yeni metod `_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 bastığımızda 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 eksen 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`'in sonuna ekleriz.

15.3.3 Rastgele Yürüyüşü Çizdirmek

İşte yürüyüşteki bütün noktaları çizdiren kod:

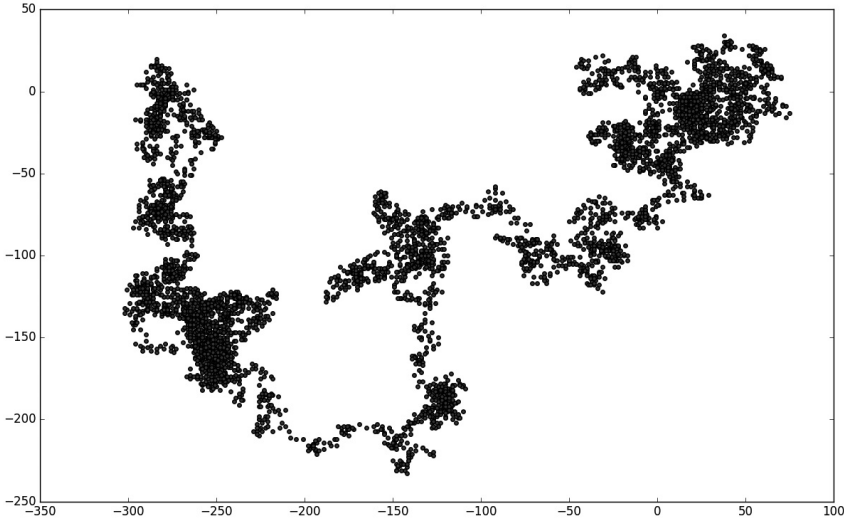
```

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 çağırdığımızda 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-

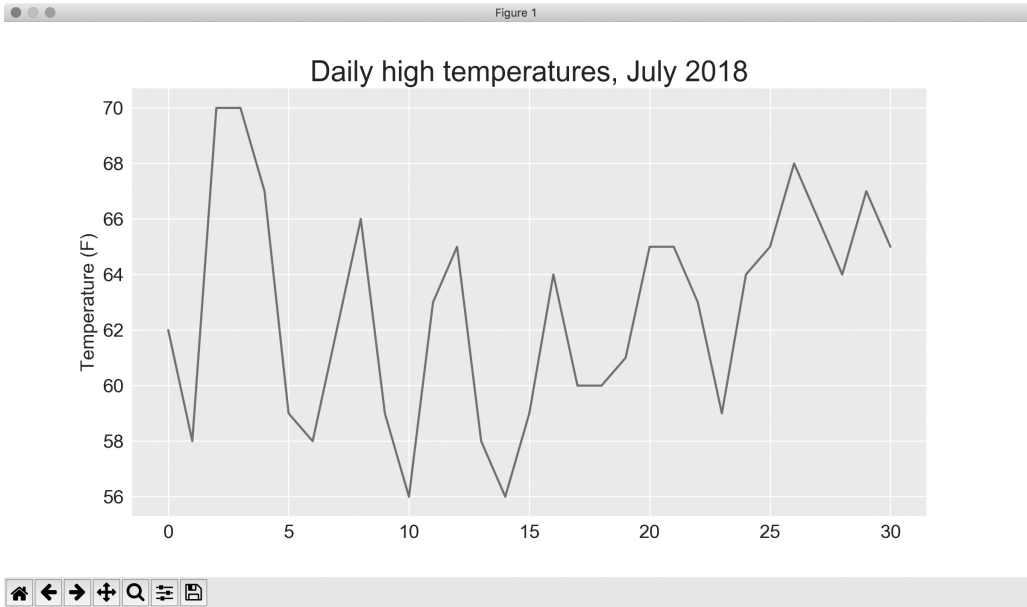
# En yüksek sıcaklıkları çizdir.
plt.style.use('seaborn')
fig, ax = plt.subplots()
ax.plot(highs, c='red')❶

# 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 belirliyoruz. 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 argümanımız 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. İstedığımız veriye sahip olduğumuzu biliyoruz.

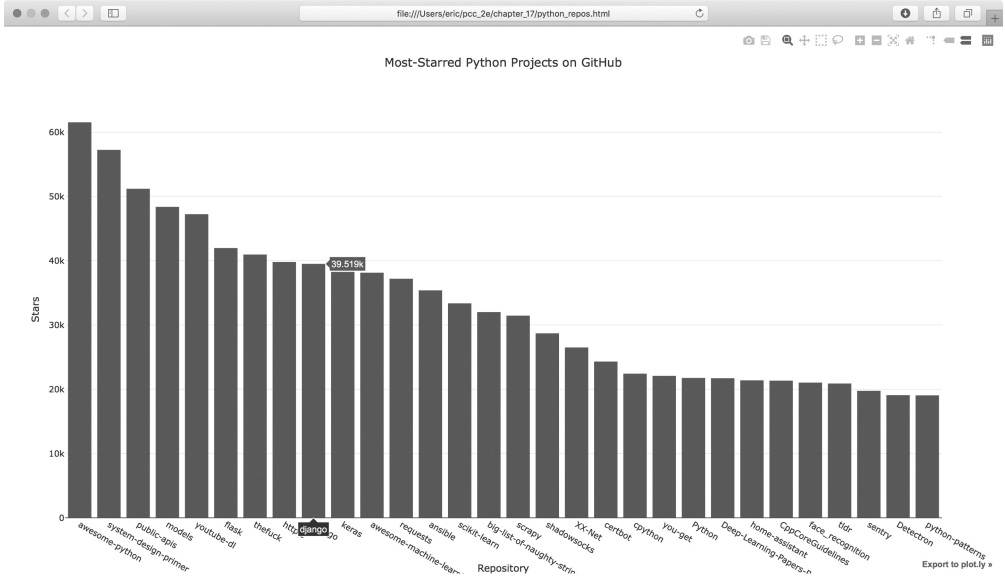
❸'te ilk çizelgeye dahil edeceğimiz verileri saklamak için iki boş liste oluşturuyoruz. Çubukları etiketlemek için her bir projenin ismine ve çubukların

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 geriden 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 gördüğümüz 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

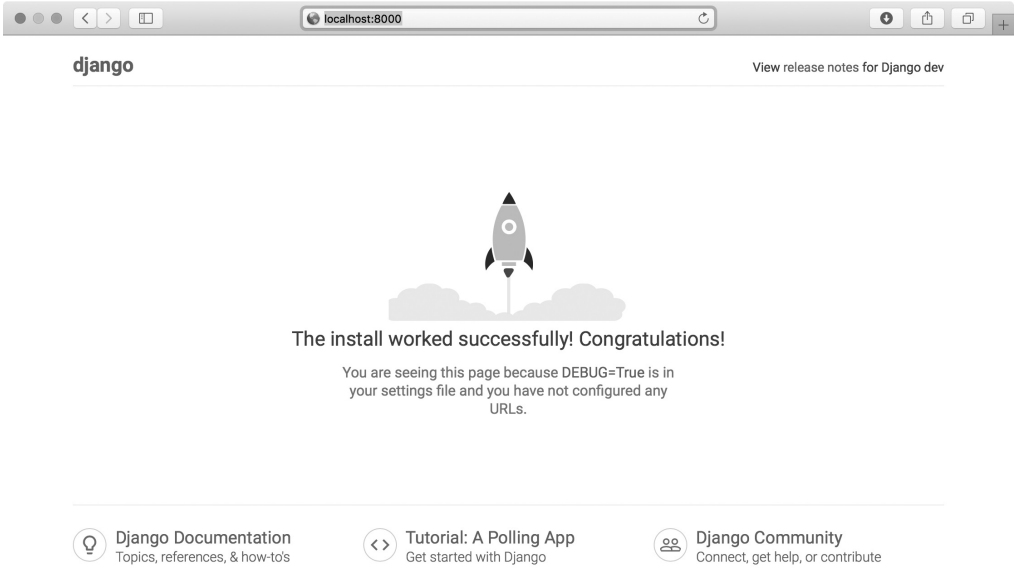
Burada gösterilen `marker` ayarları çubukların tasarımını etkiler. Çubuklar için özel bir mavi renk belirleriz ve 1.5 piksel genişliğinde koyu gri çizgiyle ana hatlarımızın belirleneceğini belirtiyoruz. Ayrıca çizelgenin görünümünü biraz yumuşatmak için çubukların opaklığını 0.6'ya ayarlıyoruz.

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 eksen 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 karşılaştırmamız. Django'nun yeni bir proje başlattığında ne oluşturduğuna aşına 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:

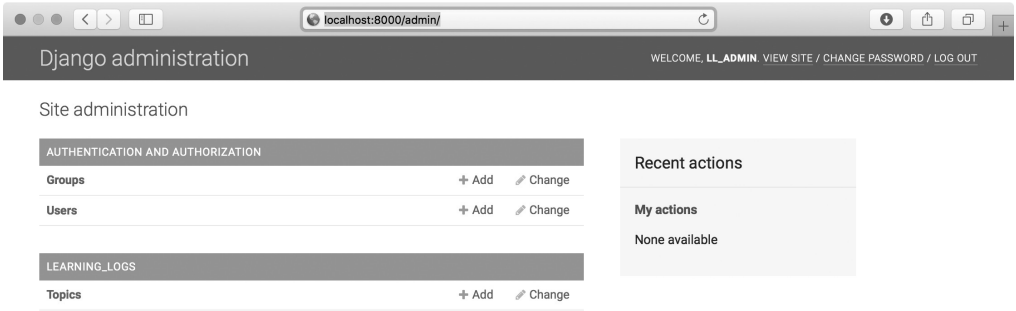
```
from django.contrib import admin

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 Ekleme

`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

- != (eşitsizlik operatörü), 81
- * (asterisk) operatörü, 162
- * (çarpma), 29
- ** (çift yıldız) operatörü, 164
- + (toplama), 29
- += operatörü, 126
- (çıkarma), 29
- { } (küme parantezleri), 101
- / (bölme), 29
- // (tam değer bölme), 290
- < (küçüktür), 82
- <= (küçük eşittir), 82
- == (eşitlik operatörü), 79–81
- > (büyüktür), 82
- [] (köşeli parantezler), 36
- % (modulo operatörü), 128, 134
- \n (yeni satır), 24
- \t (sekme), 24
- ** (üslü sayı), 29
- >= (büyük eşittir), 82

- açıklamalar, 32–33
- açıklamalar için # (kare işareti), 32
- alice.py*, 219–221
- amusement_park.py*, 87–90
- anahtar kelimeler, 537
- anahtar-değer çifti, 101–109
- and anahtar kelimesi, 82
- API (application programming interface), 410
 - çağrılar, 410
 - GitHub için, 424
 - Hacker News için, 424–428
 - hız limitleri, 416
 - sonuçları görselleştirmek, 417–424
 - veri istemek, 410
 - yantıları işlemek, 411–412
- apostrophe.py*, 26
- append() metodu, 40
- argümanlar, 144–151
- aritmetik, 29

- as anahtar kelimesi, 169
- assert metodu, 236
- assert metodu, 241

- banned_users.py*, 84
- bayraklar, 132
- beyaz boşluk, 24–26
- bicycles.py*, 36–38
- bir HTML dosyasının başlığı, 501
- birim testleri, 235
- .bmp (bitmap) görüntü dosyaları, 260
- Boole değerleri, 85, 522
- Bootstrap, 500–511
- bölme (/), 29
- büyük eşittir (>=), 82
- büyüktür (>), 82

- CamelCase, 202
- car.py*, 180
- car.py, 196
- cars.py*, 47–49, 79
- cities.py*, 133
- comment.py*, 32
- confirmed_users.py*, 137
- counting.py*, 129, 134
- CSV (comma-separated value) dosyaları, 378–392
 - dosya başlıklarını ayrıştırmak, 378–379
 - hata kontrolü, 388–392
 - veriyi okumak, 380

- çarpma (*), 29
- çıkarma (-), 29
- çıkış değeri, 130–132
- çift yıldız (**) operatörü, 164
- datetime modülü, 382–384
- death_valley_highs_lows.py*, 389–392
- def anahtar kelimesi, 143
- değil (!), 81
- değişkenler, 17–20, 31
- çoklu atama, 31

- değerler, 17
- etiket olarak, 20
- isimlendirme kuralları, 18
- sabitler, 31
- değişmez, 72
- dekoratörler, 488
- del** ifadesi, 42
- demetler, 72–74
- depremleri haritalamak, 394–407
 - büyüküğün temsili, 401
 - dünya haritası oluşturmak, 399
 - enlem-boylam sıralaması, 396
 - renk ölçekleri, 402
 - Scattergeo çizelge tipi, 399
 - üzerinde gezinilen metin, 405
 - veri
 - büyükükleri seçip almak, 397
 - geoJSON dosya formatı, 395
 - indirmek, 394–407
 - JSON verisini incelemek, 394
 - konumları seçip almak, 398
- dice_visual.py*, 371–374
- die.py*, 367
- die_visual.py*, 368–371
- dilimleme, 66
- dimensions.py*, 72–73
- div** (HTML), 503
- division_calculator.py*, 219
- dizgiler, 21–27
 - beyaz boşluklar eklemek, 24–26
 - büyük-küçük harf değişimi, 21
 - değişkenleri kullanmak, 23
 - f-dizgileri, 23
 - format()** metodu, 24
 - sekmeler eklemek, 24
 - tek ve çift tırnak, 21, 26
 - yeni satırlar eklemek, 24
- Django, 431
 - admin sitesi, 441–446
 - modelleri kaydetmek, 442, 445
 - ayrıcılıklar, 441
 - bir uygulama başlatmak, 437
 - Bootstrap, 500–511
 - açılır kapanır gezinme çubuğu, 502
 - card, 509
 - container etiketi, 505
 - django-bootstrap4** uygulaması, 500
 - formlara stil vermek, 507–508
 - gezinme çubuğu, 502–505
 - HTML başlıkları, 501–505
 - jumbotron, 506
 - dekoratörler, 488
 - dökümantasyon
 - Django, 431
 - modeller, 439
 - sorgulamalar, 449
 - şablonlar, 463
 - forms, 466–479
 - action** argümanı, 469
 - çapraz site talep sahteciliği, 469
 - doğrulama, 466
 - GET ve POST istekleri, 468
 - görüntülemek, 469
 - işlemek, 468, 472
 - ModelForm**, 466, 470
 - önceden doldurmak, 476
 - widgets, 471
 - geliştirme sunucusu, 436, 443
 - get_object_or_404()** metodu, 527
 - görünümler, 451
 - nesneleri almak, 457, 460
 - hashler (parolalar için), 442
 - HTML
 - div** etiketi, 503
 - dolgu, 506
 - main** etiketi, 505
 - sınır boşlukları, 506
 - span** etiketi, 505
 - HTTP 404 hatası, 495
 - INSTALLED_APPS**, 440, 479, 500
 - kayıt sayfası, 484–488
 - komutlar
 - createsuperuser**, 441
 - flush**, 494
 - makemigrations**, 440, 445, 492
 - migrate**, 435
 - shell**, 447
 - startapp**, 438, 479
 - startproject**, 434
 - Kullanıcı ID değeri, 491
 - kullanıcılar
 - giriş yapan kullanıcılara mesaj görün-
tölemek, 482
 - kullanıcıya oturum açmak, 485
 - UserCreationForm**, 484
 - varsayılan login görünümü, 481
 - kurmak, 433
 - localhost, 436
 - login şablonu, 481
 - modeller, 438–449, 491

- oturum açma sayfası, 480
- oturum kapatmak, 483
- proje oluşturmak, 434
- projeler ve uygulamalar, 437
- `redirect()` fonksiyonu, 467
- sayfalara erişimi kısıtlamak, 488–498
- settings.py*
 - `INSTALLED_APPS`, 440
- settings.py*
 - `INSTALLED_APPS`, 479, 500
 - `LOGIN_URL`, 489
 - `SECRET_KEY`, 528
- shell, 447, 492
- statik dosyalar, 512
- superuser, 441
- şablonlar, 452
 - block etiketi, 455
 - context sözlüğü, 457
 - çapa etiketi, 455
 - filtreler, 462
 - `for` döngüleri, 458
 - içerisinde girinti, 455
 - kalıtım, 452
 - linebreaks (satır sonları) filtresi, 462
 - şablon etiketleri, 455
- URL'ler
 - değerleri yakalamak, 461
 - isim uzayı, 455
 - şablon etiketi, 455
 - URL örüntüleri, 449–451
- URL'leri eşleştirmek, 449–451
- Üçüncü parti uygulamalar, 500
- veri tabanını taşımak, 435, 445
- veritabanları
 - basamaklı silme, 445
 - çoktan bire ilişkiler, 444
 - ikincil anahtarlar, 444
 - oluşturmak, 435
 - resetlemek, 494
 - sorgu kümeleri, 447–448, 457
 - sorgular, 461, 494
 - SQLite, 435
 - taşımak, 435, 445
 - zorunlu (boş bırakılmaz) alanlar, 492
- veriye erişimi kısıtlamak, 494–498
- veriyi kullanıcılarla ilişkilendirmek, 496
- versiyonları, 434
- yayınlanma dönemi, 434
- yeni bir proje başlatmak, 434
- docstrings, 143
- dog.py*, 175
- dog.py*, 179
- dosyalar
 - açmak, 204
 - ekleme modu, 213
 - okuma modu, 213
 - yazma modu, 213
 - dosya yolları, 206
 - göreceli, 206
 - mutlak, 206
 - kapatmak, 205
 - okumak, 204–211
 - büyük dosyalarla çalışmak, 210
 - dosyanın tamamını, 204–205
 - içerik ile çalışmak, 209
 - satır satır, 207
 - satırlar listesi oluşturmak, 208
 - yazmak
 - birden çok satır, 213
 - boş dosyalar, 212
 - ekleyerek, 214
- döndürülen değerler, 152
- electric_car.py*, 186–193
- electric_car.py* modülü, 196
- `enumerate()` fonksiyonu, 379
- epoch time, 417
- eq_explore_data.py*, 394–398
- eq_world_map.py*, 399–406
- eşitlik operatörü (`==`), 79–81
- eşitsizlik operatörü (`!=`), 81
- even_numbers.py*, 63
- even_or_odd.py*, 128
- favorite_languages.py*, 106–107
- file_reader.py*, 204–208
- `FileNotFoundError`, 219
- first_numbers.py*, 61
- fonksiyonlar, 142
 - argümanlar, 144–151
 - anahtar kelime, 147
 - hatalardan kaçınmak, 150
 - isteğe bağlı, 153
 - keyfi anahtar kelime, 164
 - keyfi sayıda, 162
 - konumsal, 145–148
 - konumsal ve keyfi argümanları karıştırmak, 164
 - listeler, 158–162
 - sıralama, 146

- çağırarak, 143–151
 - birçok kez, 146
 - eşdeğer fonksiyon çağıruları, 149–150
- döndürülen değerler, 152–155
- içerisinde liste
 - değiştirmek, 158–162
 - korumak, 162
- modüller, 167–170
 - belirli fonksiyonları içe aktarmak, 168
 - modülün tamamını içe aktarmak, 167
 - takma isim (**as**), 169
 - tüm fonksiyonları içe aktarmak (*), 170
- parametreler için varsayılan değerler, 148
- sözlük döndürmek, 154
- stil vermek, 171
- takma isim (**as**), 169
- tanım, 142
- yerleşik, 537
- foods.py*, 69–70
- for** döngüleri, 53
- formatted_name.py*, 152
- full_name.py*, 23–24
- Geany, 542
- geri izleme (traceback), 19
- GET istekleri, 468
- girinti hataları, 57–60
- Git, 409, 515
 - ayrılmış HEAD, 559
 - commitler, 410, 515
 - kontrol etmek, 558–560
 - oluşturmak, 517, 523, 526, 554, 555, 560
 - dallar, 517, 554
 - değişiklikleri geri almak, 557
 - depolar, 410
 - başlangıç durumu, 517, 554, 561
 - silme, 560
 - dosyalar
 - eklemek (add), 517, 526, 554, 560
 - görmezden gelmek, 516, 553
 - durum kontrolü, 517, 523, 553–560
 - günlüğü kontrol etmek, 555
 - HEAD, 559
 - kurmak, 515, 551
 - yapılandırma, 515, 552
- GitHub, 409
- gizli dosyalar, 516
- gövdesi
 - bir fonksiyonun, 142
 - bir HTML dosyasının, 503
- greet_users.py*, 158
- greeter.py*, 125, 142–144
- unicorn paketi, 512
- Hacker News, 424
- hava durumu verisi, 377–392
- HEAD (Git), 559
- hello_git.py*, 552–558
- hello_world.py*, 11, 16–20
- Heroku, 499
 - Bash kabuğu, 520
 - CLI kurulumu, 512
 - django-heroku** paketi, 512
 - dökümantasyon, 519
 - güvenlik, 522
 - heroku için *settings.py*'yi değiştirmek, 514, 522
 - heroku için *settings.py*'yi değiştirmek, 525
 - hesap oluşturmak, 512
 - komutlar
 - config**, 524
 - destroy**, 528
 - login**, 518
 - open**, 519
 - ps**, 518
 - rename**, 521
 - run**, 520
 - set**, 524
 - kullanıcı dostu, 521
 - kullanıcı tanımlı hata sayfaları, 524–527
 - ortam değişkenleri, 522–524
 - Procfile*, 514
 - projeler
 - canlı görüntülemek, 518
 - Heroku'ya yüklemek, 517–519, 523
 - silme, 528
 - Python çalışma zamanını belirlemek, 513
 - requirements.txt*, 513
 - superuser oluşturmak, 520–521
 - ücretsiz planın kısıtlamaları, 512, 528
 - veritabanlarını kurmak, 519
- hn_article.py*, 425
- hn_submissions.py*, 426
- Homebrew, 535
- iç içe yerleştirme, 116–122
 - derinlik, 120

- liste içerisinde sözlükler, 116–119
- sözlük içerisinde listeler, 119–121
- sözlük içerisinde sözlük, 121–122
- IDE (integrated development environment), 539
- IDLE, 542
- if ifadeleri
 - and anahtar kelimesi, 82
 - else ifadesi, 87
 - basit, 86
 - Boole deyimleri, 85
 - çoklu koşulları test etmek, 91–92
 - elif değimi, 88–90
 - kontroller
 - boş listeler, 95
 - büyük-küçük harf duyarsız eşitlik, 80
 - elemanın listede olmadığı, 85
 - eşitsizlik (!=), 81
 - eşitlik (==), 79
 - liste içindeki bir eleman, 84
 - koşullu testler, 79–84
 - or anahtar kelimesi, 83
 - sayısal karşılaştırmalar, 81–83
 - stil kılavuzları, 98
 - ve listeler, 94–96
- import *, 170
- import this, 33
- indeks hataları, 50
- input() fonksiyonu, 125–128
 - istemiciler, 125
 - sayısal girdi, 126–128
- insert() metodu, 41
- IRC (Internet Relay Chat), 548–550
- isim hataları, 19
- itemgetter() fonksiyonu, 427
- items() metodu, 110
- JSON dosyaları
 - geoJSON dosya formatı, 395
 - json.dump() fonksiyonu, 394
 - json.load() fonksiyonu, 227
 - veriyi incelemek, 394
- jumbotron, 506
- kalıtım, 186
- kayan noktalı sayılar, 29
- keys() metodu, 111
- kodu test etmek, 233–249
 - assert metodu, 241
 - başarılı testler, 235–237
 - başarısız testler, 237–239
 - birim testleri, 235
 - fonksiyonları test etmek, 233–241
 - kapsam, 235
 - setUp() metodu, 246
 - sınıfları test etmek, 241–248
 - test eklemek, 239
 - test senaryoları, 235
 - unittest modülü, 233
- koşullu testler, 79–84
- köşeli parantezler ([]), 36
- küçük eşittir (<=), 82
- küçüktür (<), 82
- küme parantezleri ({}), 101
- kümeler, 114
- language_survey.py, 243
- len() fonksiyonu, 49
- Linux
 - Merhaba Dünya'yı çalıştırmak, 11
 - Python
 - kurmak, 536
 - kurulu versiyonların kontrolü, 9
 - sorun giderme kurulumu, 12
 - Sublime Text kurulumu, 10
 - terminal
 - programları çalıştırmak, 13
 - Python oturumu başlatmak, 10
- listeler, 36
 - argüman olarak, 158–162
 - bir değer için bütün örneklerini silmek, 138
 - boş, 40
 - dilimlemek, 66–68
 - elemanlar
 - append() ile eklemek, 40
 - benzersizleri belirlemek, 114
 - değiştirmek, 39
 - del ile silmek, 42
 - erişmek, 37
 - insert() ile eklemek, 41
 - pop() ile silmek, 42–43
 - remove() ile silmek, 44
 - sonuncuya erişmek, 38
 - enumerate() fonksiyonu, 379
 - for döngüleri, 53–60
 - iç içe yerleştirme, 120, 294
 - girinti hataları, 57–60
 - hesaplanmış listeler, 64
 - indeksler, 38
 - hatalar, 50–52

- negatif, 38
- isimlendirme, 37
- kopyalama, 68–71
- len() fonksiyonu, 49
- range() fonksiyonu, 62–64
- sayısal listeler, 61–64
- sıralama, 47–50
 - reverse() metodu, 48
 - sort() metodu, 47
 - sorted() fonksiyonu, 47
- ve if ifadeleri, 94–96
- macOS
 - Merhaba Dünya'yı çalıştırmak, 11
 - Python
 - Homebrew kullanarak kurulum, 535–536
 - kurulu versiyonların kontrolü, 7
 - kurulum, 7–9
 - sorun giderme kurulumu, 12
 - Sublime Text kurulumu, 9
 - terminal
 - programları çalıştırmak, 13
 - Python oturumu başlatmak, 9
- magicians.py, 53–56
- making_pizzas.py, 167–170
- mantıksal hatalar, 58
- Matplotlib, 346–366, 381–393
 - çizgi grafiği, 346
 - çizimi formatlamak
 - alpha argümanı, 387
 - boyut, 365
 - çizgi kalınlığı, 348
 - etiketler, 348
 - gölgelendirmek, 387
 - kullanıcı tanımlı renkler, 354
 - renk haritaları, 355
 - yerleşik stiller, 350
 - eksenler
 - ax değişkeni, 347
 - axis() metodu, 354
 - kaldırmak, 364
 - fig değişkeni, 347
 - gallery, 346
 - grafikleri kaydetmek, 357
 - kurmak, 346
 - plot() metodu, 347, 348
 - pyplot modülü, 347
 - scatter grafik, 351–356
 - subplots() fonksiyonu, 347
- Merhaba Dünya, 10
- metin editörleri ve IDE'ler
 - Atom, 543
 - Emacs ve Vim, 542
 - Geany, 542
 - IDLE, 542
 - Jupyter Notebooks, 543
 - PyCharm, 543
 - Sublime Text, 4–12, 540–542
 - Visual Studio Code, 543
- modulo operatörü (%), 128, 134
- motorcycles.py, 40–45
- mountain_poll.py, 139
- mpl_squares.py, 346–351
- my_car.py, 194
- my_cars.py, 196–197
- my_electric_car.py, 196
- name.py, 22
- name_function.py, 234–239
- names.py, 234
- nesne tabanlı programlama, 174
- nitelikler, 176
- nokta gösterimi, 168, 177
- None, 108
- number_reader.py, 227
- number_writer.py, 226
- open() fonksiyonu, 204
- or anahtar kelimesi, 83
- ortam değişkenleri, 522
- Öğrenme Günlüğü Projesi, 431
 - dosyalar, 466, 471, 474, 490, 491, 494, 527
 - .gitignore, 516
 - 404.html, 525
 - 500.html, 525
 - admin.py, 442
 - base.html, 455, 459, 482, 483, 487, 501–506
 - edit_entry.html, 477
 - forms.py, 466, 470
 - index.html, 452, 506
 - learning_log/urls.py, 450, 451, 480
 - learning_logs/urls.py, 457, 460, 471, 474
 - logged_out.html, 484
 - login.html, 481–483, 507
 - models.py, 438, 444, 491

- new_entry.html*, 473
- new_topic.html*, 469
- Profile*, 514
- register.html*, 487
- requirements.txt*, 513
- runtime.txt*, 514
- settings.py*, 440, 479, 489, 514, 522, 525
- topic.html*, 461–462, 474, 477
- topics.html*, 458, 462, 470, 509
- users/urls.py*, 485
- users/urls.py*, 481
- users/views.py*, 485
- sanal ortam, 432–433
- sayfalar
 - ana sayfa, 449–453
 - girdi düzenle, 474–479
 - kayıt, 484
 - konu, 459
 - konular, 456–459
 - oturum açma, 480–483
 - oturum kapatma, 483
 - yeni girdi, 470–474
 - yeni konu, 466–470
- spesifikasyon yazmak, 432
- users uygulamaları, 479–488
- özel durumlar (exceptions), 202, 215–225
 - çökmeleri engellemek için kullanmak, 217
 - ele almak, 215
 - else** blokları, 218
 - FileNotFoundError**, 219
 - hangi hataların bildirileceğine karar vermek, 224
 - sessizce başarısız olmak, 223
 - try-except** blokları, 216
 - ZeroDivisionError**, 216
- parametreler, 144
- parrot.py*, 125, 130–133
- pass** ifadesi, 223
- PEP 8, 75–77, 98, 171
- person.py*, 154–157
- pets.py*, 138, 145–151
- pi_string.py*, 209–211
- pip, 254
 - Django kurulumu, 433
 - Matplotlib kurulumu, 346
 - Plotly kurulumu, 367
 - Pygame kurulumu, 254
 - Requests kurulumu, 411
- pizza.py*, 163–164
- plastik ördek hata ayıklaması, 546
- players.py*, 66–68
- Plotly, 346
 - Bar()** sınıfı, 370
 - çizimi formatlamak
 - çubuklar, 418
 - işaretleyici büyüklüğü, 401
 - Layout()** sınıfı, 370
 - marker rengi, 420
 - renk ölçekleri, 402
 - üzerinde gezinilen metin, 405, 421–422
 - x-ekseni, 420
 - y-ekseni, 420
 - gallery, 366
 - histogram, 369
 - kullanıcı kılavuzu, 424
 - kurmak, 367
 - offline.plot()** fonksiyonu, 370
 - Python figür referansı, 424
 - Scattergeo çizelge tipi, 399
- POST istekleri, 468
- printing_models.py*, 159–162
- Project Gutenberg, 220
- .py* dosya uzantısı, 16
- Pygame
 - arka plan renkleri, 257–258
 - boş bir pencere oluşturmak, 255
 - çarpışmalar, 301–303, 306–310, 329–331
 - çıkış yapmak, 274
 - ekran koordinatları, 262
 - ekrana yazı yazdırmak, 316
 - girdiye yanıt vermek, 257
 - fare tıklamaları, 320
 - tuş basımları, 266
 - görüntüleri ekrana çizdirmek, 262
 - görüntüleri yüklemek, 262
 - gruplar, 279
 - boşaltmak, 304
 - elemanları silmek, 282
 - içinde elemanları saklamak, 279
 - içindeki elemanları güncellemek, 281
 - içindeki tüm elemanları çizdirmek, 289
 - üzerinden döngü, 282
 - imleci gizlemek, 321
 - kurmak, 254
 - olay döngüleri, 256
 - oyunları bitirmek, 311
 - oyunları test etmek, 302
 - print()** çağırısı yapmak, 282
 - rect** nesneleri
 - sıfırdan oluşturmak, 277

- rect nesneleri, 286–289, 291–294
- renkler, 258
- tam ekran modu, 274
- yüzeyler, 256
- pyplot modülü, 347
- Python
 - >>> istemi, 4
 - anahtar kelimeler, 537
 - dökümantasyon, 546
 - kurmak
 - linux üzerine, 536
 - macOS üzerine Homebrew kullanarak, 535–536
 - Windows üzerine, 5–7, 533–535
 - kurulum
 - macOS üzerine, 7–9
 - PEP 8, 75–77, 98, 171
 - standart kütüphanesi, 200–201
 - terminal oturumları, 4
 - versiyonları, 3
 - yerleşik fonksiyonları, 537
 - yorumlayıcı, 16
 - zen'i, 33–35
- Python Enhancement Proposal (PEP), 75
- Python'un Zen'i, 33–35
- python_repos.py*, 412–416
- python_repos_visual.py*, 418–423
- r/learnpython, 548
- random_walk.py*, 358–359
- range()** fonksiyonu, 61–64
- rastgele yürüyüşler, 357–366
 - başlangıç ve bitiş noktaları, 362
 - birden çok yürüyüş üretmek, 361
 - choice()** fonksiyonu, 358
 - çizdirmek, 359
 - fill_walk()** metodu, 358
 - noktaları renklendirmek, 362
 - RandomWalk** sınıfı, 357
- read()** metodu, 205
- readlines()** metodu, 209
- remember_me.py*, 227–231
- Requests paketi, 411
- rollercoaster.py*, 127
- rw_visual.py*, 359–366
- sabitler, 31
- sanal ortam (venv), 432
- sayılar, 28–31
 - alt tire kullanımı, 31
 - aritmetik, 28
 - işlem sırası, 29
 - karşılaştırmalar, 81–83
 - kayan noktalı sayılar, 29
 - kayan noktalı sayıları ve tam sayıları karşılaştırmak, 30
 - round()** fonksiyonu, 332
 - tam değer bölme (**//**), 290
 - tam sayılar, 29
 - üslü sayılar, 28
- scatter_squares.py*, 351–357
- sekme (**\t**), 24
- sınıflar
 - çoklu örnekler, 178
 - gerçek dünya nesnelere modellemek, 191
 - içe aktarmak, 193–197
 - bir modüldeki tüm sınıfları, 197
 - birden çok sınıfı, 195–197
 - tek sınıf, 193–195
 - isimlendirme kuralları, 177
 - kalıtım, 186–193
 - __init__()** metodu, 186–187
 - alt sınıflar, 186
 - metotları geçersiz kılmak, 189
 - nitelik olarak örnekler, 189–191
 - nitelikler ve metotlar, 187
 - super()** fonksiyonu, 187
 - süper sınıflar, 187
 - üst sınıflar, 186
 - metotlar, 176
 - __init__()** metodu, 176
 - çağırarak, 178
 - nesnelere, 174
 - nitelikler, 176
 - değiştirmek, 182–185
 - erişim, 177
 - varsayılan değerler, 181
 - oluşturmak, 175–179
 - örnekler, 173
 - stil kılavuzları, 202
- sitka_highs.py*, 378–385
- sitka_highs_lows.py*, 385–388
- Slack, 550
- sleep()** fonksiyonu, 308
- sonsuz döngüler, 135
- söz dizim hataları, 26
- söz dizim vurgulama, 17
- sözlükler
 - anahtar-değer çifti, 101–109
 - ekleme, 102

- silme, 105
- boş, 103
- büyük sözlükleri biçimlendirmek, 106
- değerler
 - değiştirmek, 104
 - erişim, 101
- get() metodu, 107
- KeyError, 107
- liste içinde sıralanması, 427
- sıralama, 103
- tanım, 101
- üzerinden döngü, 109–115
 - anahtar sıralı, 113
 - anahtar-değer çifti, 109
 - anahtarlar, 111
 - değerler, 113
- split() metodu, 220
- squares.py, 63, 65
- Stack Overflow, 547
- stil kılavuzları, 75–77
 - boş satırlar, 76
 - CamelCase, 202
 - fonksiyonlar, 171
 - girinti, 75
 - if ifadeleri, 98
 - PEP 8, 75
 - satır uzunluğu, 76
 - sınıflar, 202
- strptime() metodu, 382
- Sublime Text, 4–12, 540–542
 - girinti vermek ve kaldırmak, 541
 - kodları yoruma dönüştürmek, 541
 - kurulum, 7–9
 - özelleştirmek, 540
 - Python programlarını yürütmek, 11–12
 - satır uzunluğu göstergesi, 540
 - sekmeler ve boşluklar, 540
 - yapılandırmayı kaydetmek, 541
- survey.py, 242
- süper sınıflar, 187
- takma isimler, 169
- test_name_function.py, 235–240
- test_survey.py, 244–248
- tip hatası, 72
- toplama (+), 29
- toppings.py, 81, 91–97
- try-except blokları, 216–225
- unittest modülü, 233
- Unix time, 417
- user_profile.py, 165
- Uzaylı İstilası Projesi
 - ayarları saklamak, 258
 - dinamik ayarların ilk değer ataması, 323
 - dosyalar
 - alien_invasion.py, 255
 - bullet.py, 277
 - button.py, 316
 - game_stats.py, 307
 - scoreboard.py, 326
 - settings.py, 258
 - ship.bmp, 260
 - gemi
 - görüntü bulmak, 260
 - hızı ayarlamak, 270–272
 - mesafeyi ayarlamak, 272
 - sürekli hareket, 267–272
 - mermiler, 276–284
 - ateşlemek, 279
 - ayarlar, 277
 - daha büyük yapmak, 302
 - eskileri silmek, 281
 - hızlandırmak, 304
 - sayısını sınırlamak, 282
 - uzaylılarla çarpışmak, 301–303, 329–331
 - Oynat düğmesi
 - çizdirmek, 318
 - eklemek, 313–322
 - etkinliğini kaldırmak, 321
 - fare imlecini gizlemek, 321
 - oyunu başlatmak, 319
 - oyunu resetlemek, 320
 - oyunu bitirmek, 311
 - planlama, 251
 - seviyeler
 - eklemek, 322–325
 - hız ayarını değiştirmek, 323
 - hızı resetlemek, 325
 - sınıflar
 - Alien, 286–289
 - Bullet, 277–278
 - Button, 316–317
 - GameStats, 307
 - Scoreboard, 326–327
 - Settings, 258
 - Ship, 261–262
- skor verme, 325–343
 - bütün vuruşlara, 330

- en yüksek skor, 333
- gemilerin sayısı, 338–341
- güncellemek, 328
- puan değerini artırmak, 331
- resetleme, 330
- seviye, 331–332
- skor niteliği, 325
- skoru yuvarlamak, 332–333
- uzaylılar
 - ekranın alt tarafına ulaşma, 310
 - filonun yönünü değiştirmek, 299
 - filoyu düşürmek, 299
 - filoyu oluşturmak, 290–296
 - filoyu tekrar oluşturmak, 303
 - gemiyile çarpışma, 306–310
 - kenarları kontrol etmek, 299
 - mermilerle çarpışmak, 301–303, 329–331
 - uzaylı oluşturmak, 286
- üslü sayı (**), 29
- üst sınıflar, 186
- values()** metodu, 114
- varsayılan değerler, 148
 - fonksiyon parametreleri, 148
 - sınıf nitelikleri, 181
- veri analizi, 345
- veri görselleştirme, 345
- veriyi saklamak, 225–227
- versiyon kontrol sistemleri, 551
- voting.py*, 86–88
- web çerçevesi, 431
- while** döngüleri, 130–141
 - aktif bayrağı, 132–133
 - break** ifadesi, 133
 - çıkış değeri, 130–132
 - öğeleri bir listeden diğerine taşımak, 137
 - sonsuz, 135
- Windows
 - Merhaba Dünya'yı çalıştırmak, 11
 - Python
 - kurulum, 5–7, 533–535
 - sorun giderme kurulumu, 12, 535
 - Sublime Text kurulumu, 7
 - terminal
 - programları çalıştırmak, 13
 - Python oturumu başlatmak, 7
- with** ifadesi, 205
- word_count.py*, 222–224
- write()** metodu, 213
- write_message.py*, 212–213
- yardım almak
 - çevrim içi kaynaklar, 547
 - Discord, 550
 - IRC (Internet Relay Chat), 548–550
 - plastik ördek hata açıklaması, 546
 - r/learnpython, 548
 - resmi python dökümantasyonu, 548
 - Slack, 550
 - Stack Overflow, 547
 - üç ana soru, 545
- yeni satır (\n), 24
- yeniden yapılandırma, 229–231, 264
- yerleşik fonksiyonlar, 537
- yıldız (*) operatörü, 162
- zar atmak, 366–376
 - Die** sınıfı, 367
 - farklı büyüklüklerde, 373
 - iki zar, 371
 - randint()** fonksiyonu, 367
 - sonuçları analiz etmek, 369