

Tutorial

djangogirls

İçindekiler

Giriş	1.1
Kurulum	1.2
Internet nasıl çalışır	1.3
Komut satırına giriş	1.4
Python kurulumu	1.5
Kod editörü	1.6
Python'a giriş	1.7
Django nedir?	1.8
Django kurulumu	1.9
İlk Django projen!	1.10
Django modelleri	1.11
Django admin	1.12
Yayına alın!	1.13
Django url'leri	1.14
Django views - yaratma zamanı geldi!	1.15
HTML'ye giriş	1.16
Django ORM ve QuerySets (Sorgu Setleri)	1.17
Template içerisinde dinamik veri	1.18
Django template	1.19
CSS - sayfanı güzelleştir	1.20
Template genişletmek	1.21
Uygulamanı genişlet	1.22
Django formları	1.23
Sıradanızda ne var?	1.24

Django Girls Eğitimi

[gitter](#) [join chat](#)

Bu çalışma Creative Commons Attribution-ShareAlike 4.0 International License altında lisanslanmıştır. Lisansın bir kopyasını görmek için <http://creativecommons.org/licenses/by-sa/4.0/> adresini ziyaret edin.

Çeviri

Bu döküman İngilizce'den Türkçe'ye 28 kişiden oluşan tamamen gönüllü bir ekip tarafından çevrildi: Elif T. Kuş, Şirin Sayılı, Suzan Üsküdarlı, Oğuzcan Küçükbayrak, Mehmet Ragıp Altuncu, Şahin B., Kıvanç Yazan, Adil Öztaşer, olasitarska, Oğuz Erdoğmuş, ophelia, Tuğçe Şirin, Fazilet Çilli, Özge Barbaros, Evin Pınar Örnek, Yiğit Bekir Baya, Berna Erden, Akın Toksan, Çağıl, Berat Doğan, Taha Yusuf, Helin Ece Akgül, Müge Kurtipek, Nezihe Pehlivan, Can Güler, Wert Yuio, Barış Arda Yılmaz ve Aybüke Özdemir! Emeği geçen herkese sonsuz teşekkürler! Ellerinize sağlık!

Giriş

Hiç dünyanın her geçen gün daha fazla teknoloji ile alakalı olduğunu ve bir şekilde geride kaldığınızı hissettiniz mi? Hiç web sayfalarının nasıl yapıldığını merak edip başlamak için gereken motivasyona sahip olmadığınız oldu mu? Yazılım dünyasının sizin için, kendi başınıza bir şeyler yapmayı denemek için bile, çok karmaşık olduğunu düşündünüz mü?

Öyleyse, sizin için iyi haberlerimiz var! Programlama göründüğü kadar zor değil ve size ne kadar keyifli olabileceğini göstermek istiyoruz.

Bu eğitim sihirli bir şekilde sizi bir programcıya dönüşturmeyecek. Eğer bu konuda iyi olmak istiyorsanız, aylara ve hatta yıllara dayanan bir öğrenme ve pratiğe ihtiyacınız var. Fakat biz size programmanın ya da websayfası oluşturmanın göründüğü kadar da karmaşık olmadığını göstermek istiyoruz. Farklı parçaları, değişik kısımları anlatabildiğimiz kadar iyi anlatmaya gayret edeceğiz ki artık teknoloji gözünüzü korkutmasın.

Ümit ederiz ki sizin de teknolojiyi bizim kadar sevmenizi sağlayabiliriz!

Bu eğitim ile ne öğreneceksiniz?

Eğitimi bitirdiğinizde, basit fakat çalışan bir web uygulamasına sahip olacaksınız: kendi blog uygulamanız. Size blogunuzu nasıl canlıya alabileceğinizi de göstereceğiz ki başkaları da çalışmanızı görebilsin!

Aşağı yukarı şu şekilde gözükecek:

The screenshot shows a web browser window titled "Django Girls Blog" with the URL "127.0.0.1:8000". The page has a yellow header with the "Django Girls" logo and navigation icons. Below the header, there are three blog post cards:

- Nulla facilisi** (published: 28-06-2014)
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Cras bibendum sapien interdum, posuere massa et, hendrerit leo. Nam commodo facilisis sapien vitae ornare. Integer eget purus posuere, vesti...
- Fusce vehicula feugiat augue eget consectetur** (published: 28-06-2014)
Pellentesque venenatis elit tortor, eu dictum magna accumsan in. Aenean vestibulum velit arcu, eleifend mattis purus suscipit a. Ut vitae pellentesque lorem. Integer lobortis orci in est molestie t...
- Duis quis imperdiet justo** (published: 28-06-2014)
Nulla ut metus luctus, tristique massa sit amet, venenatis eros. Aliquam hendrerit ligula nec viverra euismod. Vivamus eu sagittis diam, eget pharetra libero. Vestibulum ante ipsum primis in faucib...

Eğer eğitimi kendi başınıza takip ediyorsanız ve bir problem yaşadığınızda yardım edebilecek bir eğitmeniniz yoksa, sizin için bir chat odamız var: [gitter](#) [join chat](#). Daha önce katılmış olan kişilerden ve eğitmenlerimizden zaman zaman chat odasında bulunmalarını ve eğitimi takip eden kişilere yardımcı olmalarını istedik! Chat odasında sorularınızı sormaktan çekinmeyin!

Tamam, [hadi en baştan başlayalım...](#)

Hakkında ve katkıda bulunma

Bu eğitim [DjangoGirls](#) tarafından sürdürülürmektedir. Eğer hatalar bulursanız ve eğitimi güncellemek isterseniz lütfen katkıda bulunma kılavuzunu takip edin.

Bu eğitimi başka dillere çevirmemize yardımcı olmak ister misiniz?

Hali hazırda, çeviriler [crowdin.com](#) platformunda tutuluyor:

<https://crowdin.com/project/django-girls-tutorial>

Eğer diliniz [crowdin](#)'de listelenmemiş ise, lütfen GitHub'da dilinizle alakalı bilgi veren bir [konu açın](#) böylece dilinizi ekleyebilelim.

Eğer tutorial'ı evde yapıyorsanız

Eğer tutorial'ı [Django Girls etkinlikleri](#)nin birinde değil de evde yapıyorsanız, bu bölümü atlayabilirsiniz ve doğrudan [İnternet nasıl çalışır?](#) bölümüne gidebilirsiniz.

Çünkü burada anlatılanları tutorial boyunca zaten işliyoruz, bu kısım kurulum talimatlarının tek yerde toparlandığı ek bir sayfa sadece. Django Girls etkinlikleri, tutorial sırasında uğraşmamak için herşeyi kurduğumuz bir "kurulum akşamı"nı içeriyor. Bu sayfayı onun için kullanıyoruz.

Eğer yararlı olduğunu düşünüyorsanız, bu bölümü okuyabilirsiniz. Ama bilgisayarınıza bir şeyle kurmadan önce bir kaç şey öğrenmeye başlamak istiyorsanız, bu bölümü atlayın. Size kurulum işlerini sonra anlatacağız.

İyi şanslar!

Kurulum

Atölyede bir blog geliştireceksiniz, eğitim günü kodlamaya hazır olmanız için önceden ayarlamakta fayda olan birkaç kurulum var.

Python Yükleyin

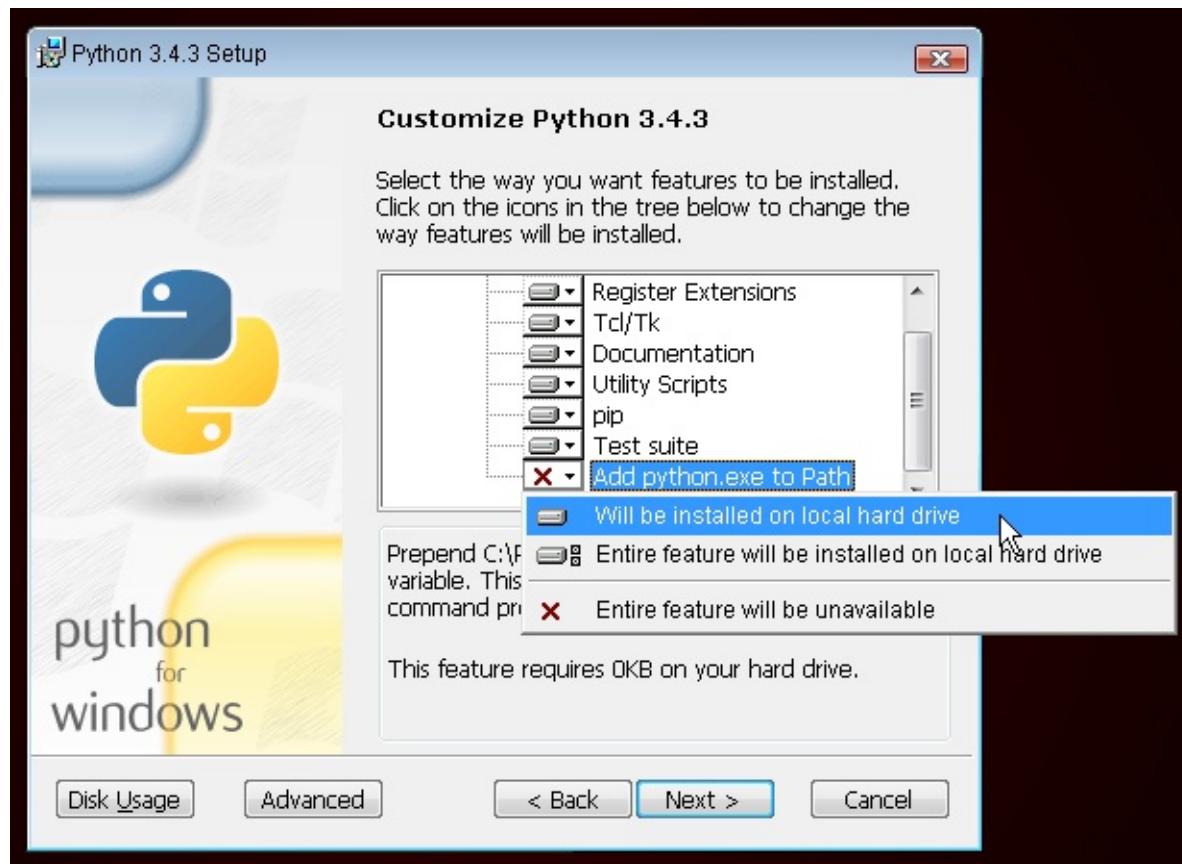
Bu bölüm Geek Girls Carrots tarafından yapılan bir eğitime dayanılarak hazırlanmıştır. (<http://django.carrots.pl/>)

Django, Python ile yazılmıştır. Django ile bir şey yapmak için Python diline ihtiyacımız var. Hadi Python kurmaya başlayalım! Biz Python 3.4 kurmak istiyoruz, eğer daha düşük bir sürümü sahipseniz, güncellemelisiniz.

Windows

Windows için Python indirmek için resmi siteyi ziyaret edebilirsiniz: <https://www.python.org/downloads/release/python-343/>. *.msi dosyasını indirdikten sonra, dosyayı çalıştırın (çift tıklayın) ve yönergeleri izleyin. Python kurulumunu yaptığınız dizinin yolunu (dizini) unutmamanız önemli. Daha sonra lazım olacak!

Dikkat: Özelleştir (Customize) olarak seçilmiş kurulum sihirbazının ikinci ekranında seçenekleri aşağıya kaydırın ve "Add python.exe to the Path" (python.exe yolunu ekle) seçeneğinin üzerine gelip "Will be installed on local hard drive" (Lokal hard diske kurulacak) seçeneğini seçin:



Linux

Muhtemelen sisteminizde Python zaten yüklüdür. Yüklü olup olmadığını (ya da hangi versiyon olduğunu) kontrol etmek için komut satırını açın ve aşağıdaki komutları girin:

```
$ python3 --version
Python 3.4.3
```

Python yüklü değilse ya da farklı bir versiyon edinmek istiyorsanız aşağıdaki adımları takip edin:

Debian veya Ubuntu

Terminale bu komutu girin:

```
$ sudo apt-get install python3.4
```

Fedora (21'e kadar)

Terminalde kullanmanız gereken komut:

```
$ sudo yum install python3.4
```

Fedora (22+)

Terminalde kullanmanız gereken komut:

```
$ sudo dnf install python3.4
```

OS X

Python kurulum dosyasını indirmek için resmi siteye gitmelisiniz: <https://www.python.org/downloads/release/python-342/>:

- Mac OS X 64-bit/32-bit yükleyici dosyasını indirin,
- `python-3.4.3-macosx10.6.pkg` dosyasına çift tıklayarak yükleyiciyi çalıştırın.

Kurulumun başarılı olup olmadığını kontrol etmek için *Terminal* uygulamasını açın ve aşağıdaki `python3` komutunu çalıştırın:

```
$ python3 --version  
Python 3.4.3
```

Herhangi bir şüpheniz varsa, kurulumda bir şeyler ters gittiyse ya da sonrasında ne yapacağınızı bilmiyorsanız eğitmene sorabilirsiniz! Bazen işler düzgün gitmiyor, bu durumda daha fazla deneyime sahip birinden yardım istemelisiniz.

Bir "virtualenv" kurun ve Django'yu yükleyin

Bu bölümün bir kısmı Geek Girls Carrots tarafından hazırlanmış eğitimlere dayanılarak hazırlanmıştır (<http://django.carrots.pl/>).

Bu bölümün bir parçası Creative Commons Attribution-ShareAlike 4.0 International License ile lisanslı [django-marcador tutorial](#)'a dayanılarak hazırlanmıştır. Django-marcador tutorial'ının hakları Markus Zapke-Gründemann'e aittir.

Virtual environment (Sanal ortam)

Django'yu yüklemeden önce kod ortamınızı düzenli tutmak için son derece yararlı bir araç yükleyeceğiz. Bu adımı atlayabilirsiniz, fakat atlamanızı tavsiye ederiz. En iyi olası kurulum ile başlamanız siz gelecekteki bir sürü sorundan koruyacaktır!

Öyleyse bir **virtual environment**(diğer adıyla *virtualenv*) kuralım. Virtualenv Python/Django kurulumunu her proje için ayrı tutup izole eder. Bu, bir websitesine yapacağınız değişikliklerin diğer geliştirdiklerinize yansımayacağı anlamına gelir. Muazzam, değil mi?

Yapmanız gereken tek şey `virtualenv` oluşturmak için bir dizin bulmak; örneğin giriş diziniz. Windows'da bu `C:\Users\isim` olabilir (`isim` kısmı kullanıcı adınız olacak şekilde).

Bu eğitim için giriş dizinizde yeni açtığımız `djangogirls` adlı bir klasör kullanacağız:

```
mkdir djangogirls  
cd djangogirls
```

`myvenv` adında bir virtualenv oluşturacağız. Genel komut aşağıdaki şekilde olacak:

```
python3 -m venv myvenv
```

Windows

Yeni bir `virtualenv` oluşturmak için konsolu açıp (nasıl yapıldığını birkaç adım önce anlatmıştık - hatırlıyorsunuz değil mi?) `C:\Python34\python -m venv myvenv` komutunu çalıştırın. Şöyleden görünmeli:

```
C:\Users\isim\djangogirls> C:\Python34\python -m venv myvenv
```

C:\Python34\python dizini önceden Python'u kurduğunuz dizin ve `myvenv` ise `virtualenv` 'inizin ismi olacaktır. İstediğiniz herhangi bir ismi kullanabilirsiniz, ama küçük harfle yazılmasına ve boşluk, aksan karakterleri (örn: å) ve özel karakterleri kullanmamaya dikkat edin. Ayrıca ismi kısa tutmak iyi bir fikir olabilir, zira bu ismi çok kullanıyor olacaksınız!

GNU/Linux ve OS X

Linux ve OS X işletim sistemlerinde `virtualenv` oluşturmak için `python3 -m venv myvenv` komutunu çalıştırın. Komut şu şekilde görünecektir:

```
~/djangogirls$ python3 -m venv myvenv
```

Burada `myvenv` sizin `virtualenv` 'inizin ismi. Dilerseniz istediğiniz herhangi bir isim kullanabilirsiniz, ama büyük harf ve boşluk kullanmamaya dikkat edin. İsmi çok fazla kullanacağınız için kısa tutmak da işinize yarayacaktır.

NOT: Ubuntu 14.04 işletim sisteminde sanal ortam yaratmaya çalışırken şu hatala karşılaşabilirsiniz:

```
Error: Command '['/home/zeynep/Slask/tmp/venv/bin/python3', '-Im', 'ensurepip', '--upgrade', '--default-pip']'
returned non-zero exit status 1
```

Bu sorunu çözmek için `virtualenv` komutunu kullanabilirsiniz.

```
~/djangogirls$ sudo apt-get install python-virtualenv
~/djangogirls$ virtualenv --python=python3.4 myvenv
```

Virtualenv ile çalışmak

Yukarıdaki komutlar `myvenv` (veya seçtiğiniz isimde) bir klasör oluşturacaktır. Bu klasörde birçok dosya ve klasör bulunur.

Windows

Şu komutu çalıştırarak virtualenv'i başlatın:

```
C:\Users\Name\djangogirls> myvenv\Scripts\activate
```

GNU/Linux ve OS X

Şu komutu çalıştırarak virtualenv'i başlatın:

```
~/djangogirls$ source myvenv/bin/activate
```

`myvenv` yerine kendi seçtiğiniz `virtualenv` ismini koymayı unutmayın!

NOT: bazen `source` komutu kullanılamıyor durumda olabilir. Böyle durumlarda onun yerine aşağıdaki komutu da kullanabilirsiniz:

```
~/djangogirls$ . myvenv/bin/activate
```

Konsolunuzda şuna benzer bir şey gördüğünüzde `virtualenv` 'in başladığını anlayabilirsiniz:

```
(myvenv) C:\Users\Name\djangogirls>
```

ya da:

```
(myvenv) ~/djangogirls$
```

En başta beliren `(myvenv)` 'e dikkat edin!

Virtualenv ile çalışırken `python` otomatik olarak doğru sürümü çalıştıracaktır. Yani `python3` yerine `python` yazabilirsiniz.

Artık bütün gerekli uygulamaları bir araya getirdiğimize göre sonunda Django'yú yükleyebiliriz!

Django'yu yüklemek

`virtualenv` 'i başlattığınız için artık Django'yu `pip` kullanarak yükleyebiliriz. Konsola `pip install django==1.9` komutunu yazın. (İki tane eşittir işaretini kullandık: `==`).

```
(myvenv) ~$ pip install django==1.9
Downloading/unpacking django==1.9
  Installing collected packages: django
    Successfully installed django
Cleaning up...
```

Windows'ta

Eğer Windows işletim sisteminde `pip` komutunu kullanırken bir hataya karşılaştıysanız proje adresinizin boşluk, aksan veya özel karakter içerip içermediğini (`c:\users\User Name\django\girls` gibi) kontrol edin. Eğer durum buyusa projenizi boşluk veya özel karakter içermeyen bir adrese taşıyın; önerimiz `c:\django\girls` olacaktır. Taşıma işleminden sonra yukarıdaki komutu tekrar deneyin.

Linux'ta

Eğer Ubuntu 12.04 işletim sisteminde `pip` komutunu çağırırken bir hata iletişiyle karşılaşılıyorsanız `python -m pip install -U --force-reinstall pip` komutunu çalıştırarak `pip` kurulumunu onarmayı deneyin.

İşte bu kadar! Sonunda Django uygulamanızı oluşturmaya hazırlınsız!

Bir kod düzenleyicisi yükleyin

Birçok farklı kod editörü var, hangi editörü kullanacağınız kişisel tercihinize bağlı. Çoğu Python programcısı PyCharm gibi karmaşık fakat son derece güçlü IDE'leri (Integrated Development Environments-Entegre Geliştirme Ortamları) kullanır. Başlangıç seviyesi için bunlar muhtemelen pek uygun olmayacağındır. Bizim önerilerimiz aynı derecede güçlü fakat çok daha basit editörler olacak.

Bizim önerilerimizi aşağıda bulabilirsiniz, fakat eğitmenlerinize onların tercihlerini sormaktan çekinmeyin - onlardan yardım almak daha kolay olacaktır.

Gedit

Gedit açık kaynaklı, ücretsiz bir editördür. Tüm işletim sistemlerinde kullanılabilir.

[Buradan indirin](#)

Sublime Text 2

Sublime Text ücretsiz deneme sürümüne sahip oldukça popüler bir editördür. Kurulumu ve kullanımı basittir, tüm işletim sistemlerinde kullanılabilir.

[Buradan indirin](#)

Atom

Atom [GitHub](#) tarafından geliştirilen oldukça yeni bir editör. Atom ücretsiz, açık kaynak kodlu, kurulumu ve kullanımı basit bir editördür. Windows, OSX ve Linux işletim sistemlerinde kullanılabilir.

[Buradan indirin](#)

Neden bir kod editörü kuruyoruz?

Neden Word ya da Notepad kullanmak yerine, özel bir kod editörü yazılımı kurduğumuzu merak ediyor olabilirsiniz.

Birinci nedeni, kodun **düz metin** olması gerekliliği. Word ve TextEdit gibi programlar [RTF \(Rich Text Format\)](#) gibi özel formatları kullanarak düz metin yerine zengin metin (fontlu ve formatlı metin) üretiyorlar.

İkinci neden, kod editörleri kod düzenlemek için özelleşmişlerdir, dolayısıyla kodu anlamına göre renklerle öne çıkarma (highlighting) veya tırnakları otomatik kapama gibi yararlı özellikler sağlar.

Bütün bunları ilerde uygulama içerisinde göreceğiz. Yakında güvenilir ihtiyar kod editörünü favori araçlarınız arasında görmeye başlayacaksınız :)

Git yükleyin

Windows

Git'i [git-scm.com](#) adresinden indirebilirsiniz. 5. adıma kadar "next"e basarak geçebilirsiniz. 5. adımda "Adjusting your PATH environment" dediği yerde, "Run Git and associated Unix tools from the Windows command-line" (en alttaki opsionu) seçin. Onun dışında, varsayılanlar iyi. Kodu çekerken Windows-stili, kodu gönderirken Unix-stili satır sonları iyidir.

MacOS

Git'i [git-scm.com](#)'den indirin ve yönergeleri izleyin.

GNU/Linux

Halihazırda yüklü değilse, git'i paket yöneticinizle indirebilirsiniz. Şunlardan birini deneyin:

```
sudo apt-get install git  
# veya  
sudo yum install git
```

GitHub hesabı oluşturun

[GitHub.com](#)'a gidin ve ücretsiz yeni bir kullanıcı hesabı oluşturun.

PythonAnywhere hesabı oluşturun

Sırada PythonAnywhere sitesinde bedava bir "Beginner" (yeni başlayan) hesabı açma işi var.

- [www.pythonanywhere.com](#)

Not Burada kullanıcı isminizi seçerken bilin ki blogunuzun URL'si `kullaniciadiniz.pythonanywhere.com` şeklinde olacak. O yüzden ya kendi rumuzunuzu(nickname) seçin ya da blogunuzun konusu ile ilgili bir isim seçin.

Okumaya başlayın

Tebrikler, ayarlarınız tamam ve hazırlısınız! Eğer atölyeden önce hala vaktiniz var ise, başlangıç bölümlerinden bazılarını okumanız yararlı olacaktır:

- [Internet nasıl çalışır](#)
- [Komut satırına giriş](#)
- [Python'a giriş](#)
- [Django nedir?](#)

İnternet nasıl çalışır

Bu bölüm Jessica McKellar'ın "How the Internet works" yani "İnternet nasıl çalışır" adlı bir konuşmasından esinlenerek oluşturuldu. (<http://web.mit.edu/jesstess/www/>).

Bahse varız ki İnternet'i her gün kullanıyorsunuz. Fakat tarayıcınıza <http://djangogirls.org> gibi bir adres girip `enter` tuşuna bastığınızda neler olduğunu biliyor musunuz?

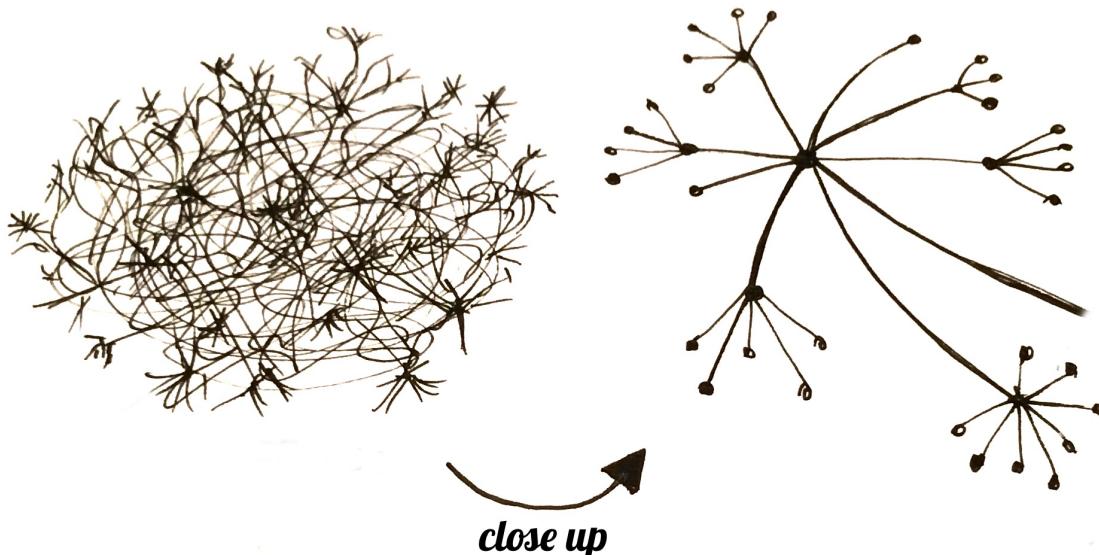
Öncelikle bir web sitesinin bir sabit diske kaydedilmiş bir grup dosya olduğunu anlamamız gereklidir. Típkı filmleriniz, müzikleriniz ya da resimleriniz gibi. Ancak, web sitelerine özel söyle bir durum var: HTML adı verilen bir bilgisayar kodu içerirler.

Eğer programlamaya aşina değilseniz HTML'yi kavramak başlangıcta zor olabilir, ama web tarayıcılarınız (Chrome, Safari, Firefox, vs.) ona bayılıyor. Web tarayıcılarınız bu kodu anlamak, komutlarını yerine getirmek ve size bu websitesini oluşturan dosyaları sunmak için tasarlanmıştır, tam da olmasını istediğiniz gibi.

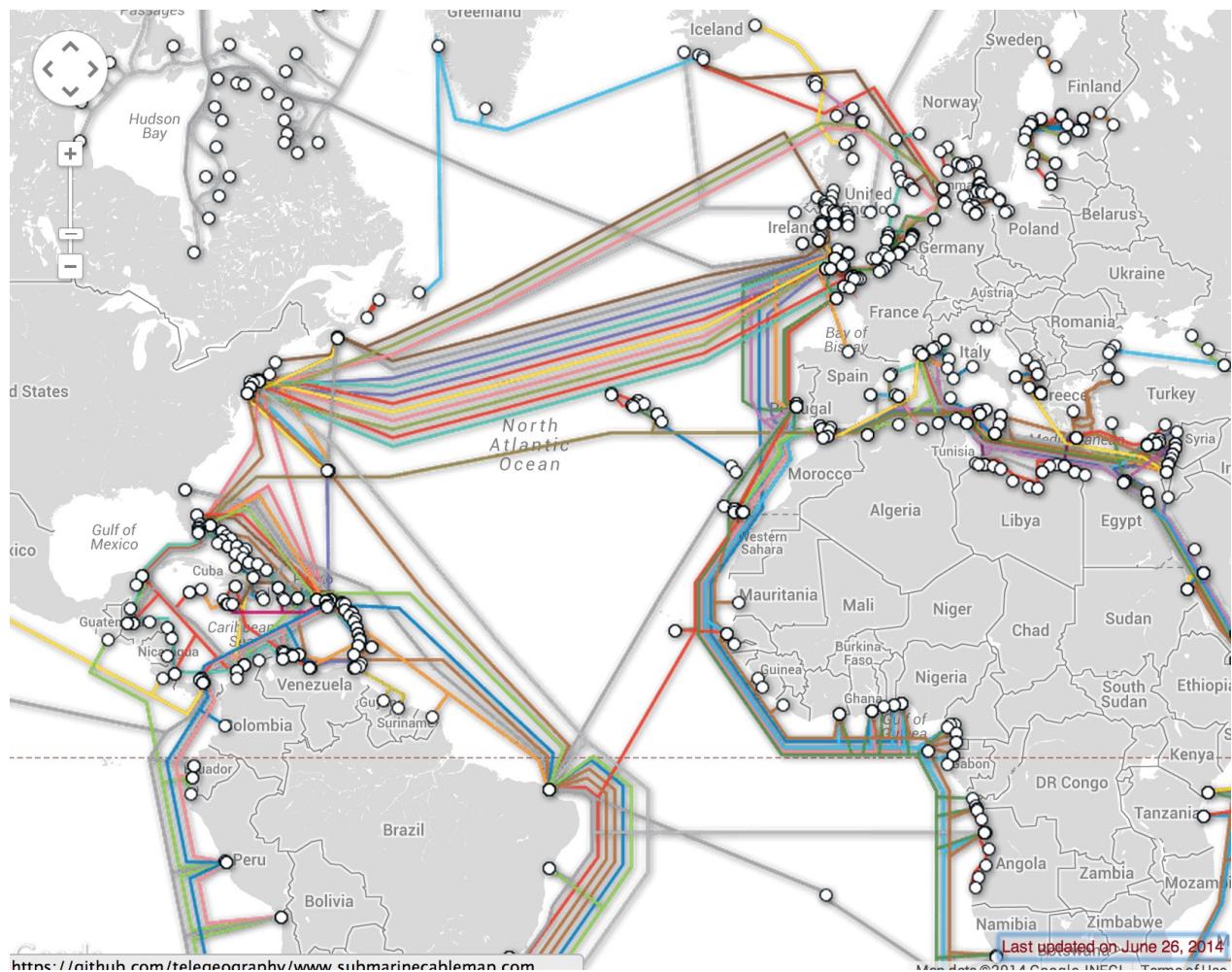
Her dosya gibi, bu HTML dosyalarını da sabit diskte bir yerde saklamamız gereklidir. Internet için, *sunucu* denilen özel ve güçlü bilgisayarlar kullanıyoruz. Bir ekranları, fareleri ya da klavyeleri yok çünkü esas amaçları veriyi saklamak ve sunmak. Bu yüzden onlara *sunucu* diyoruz -- çünkü size veri *sunuyorlar*.

Peki, fakat İnternet'in nasıl göründüğünü bilmek istiyorsunuz. Değil mi?

Size bir resmini çizdim, İnternet işte buna benzer:

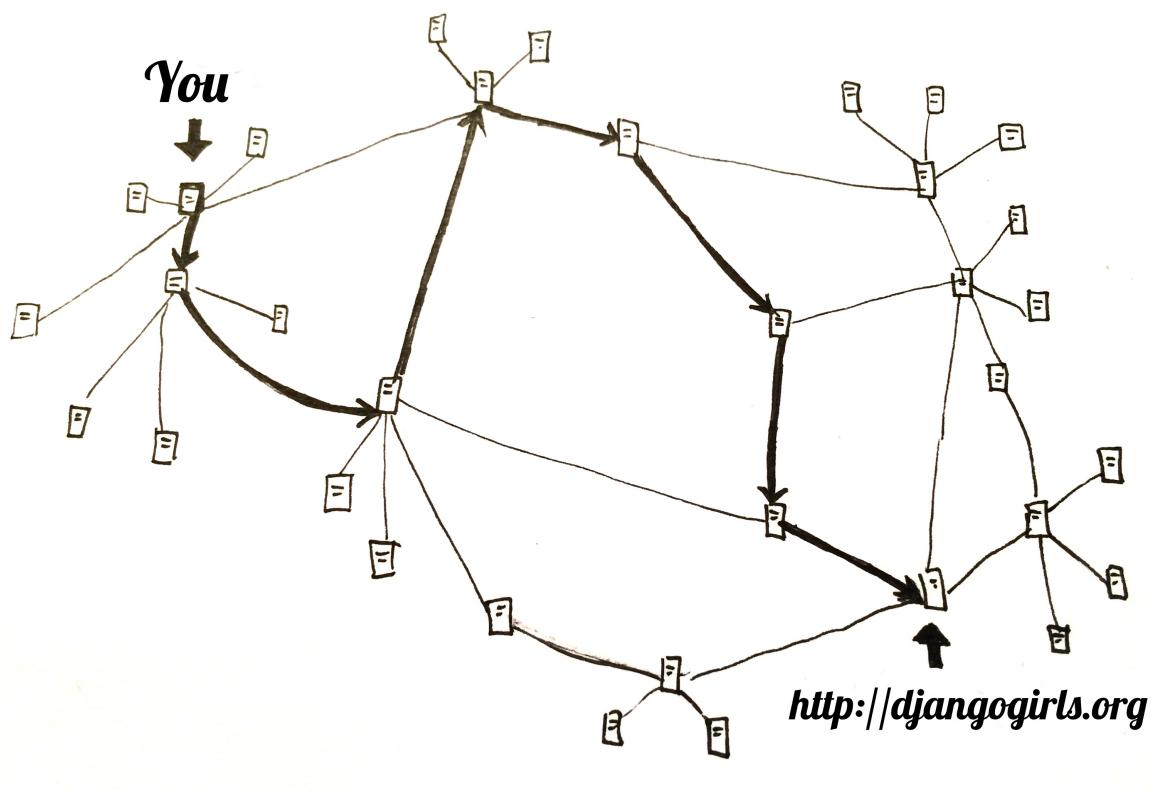


Çok karışık, değil mi? Aslında bu birbirine bağlı makinelerin(yukarıda bahsi geçen *sunucular*) birbirine bağlandığı bir ağ. Yüz binlerce makine! Dünyanın dört bir yanını saran yüz binlerce kilometrelük kablolar! İnternet'in ne kadar karışık olduğunu görmek için denizaltı kablo haritası web sitesine(<http://submarinecablemap.com>) göz atabilirisiniz. İşte web sitesinden bir ekran görüntüsü:



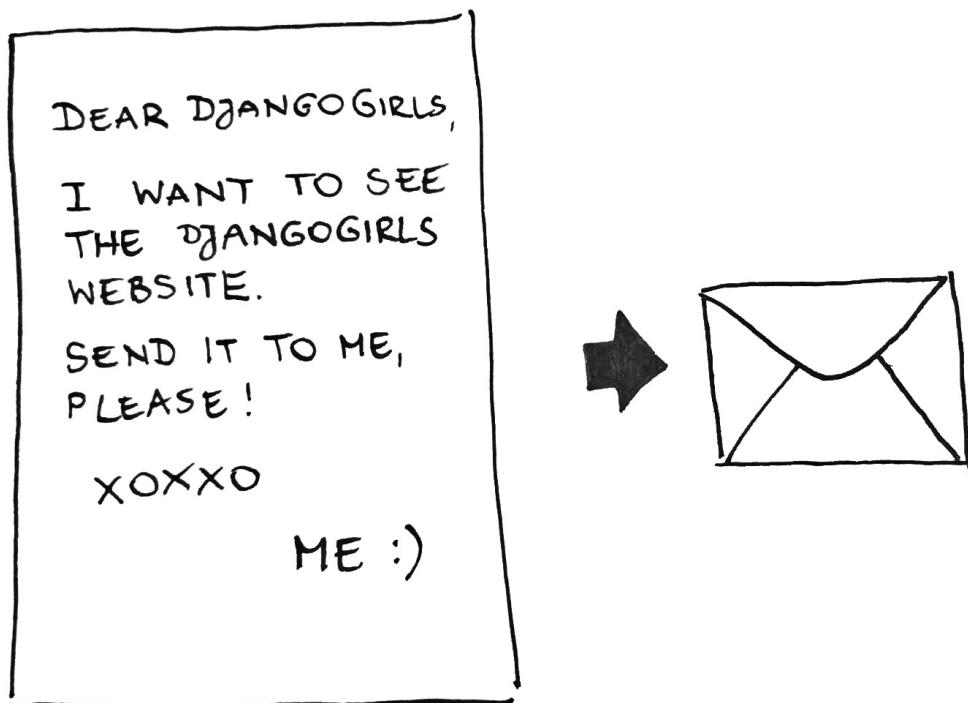
Büyük bir ağı, değil mi? Ama açıkça belli ki, İnternet'e bağlanan tüm makineler arasında kablolar olması mümkün değil. Yani, bir makineye ulaşmak için (örneğin <http://djangogirls.org> web sitesinin kayıtlı olduğu) bir çok farklı makineden istek geçmesi gereklidir.

Şöyle gözükmeakte:



Tarayıcınıza <http://djangogirls.org> yazdığınızda, "Sevgili Django Girls, ben djangogirls.org web sitesini görmek istiyorum. Lütfen bana gönderin!" diyen bir mektup gönderdiğinizizi düşünün.

Mektubunuz size en yakın postaneye gider. Daha sonra sizden biraz daha uzak bir postaneye gider, daha sonra biraz daha uzaktakine, daha uzağa ve en son gitmesi gereken yere. Tek farklı olan, aynı yere bir çok mektup(*veri paketi*) gönderirseniz, tamamen farklı postanelerden(*yönlendirici*) geçebilir. Bu, her bir postanede nasıl dağıtım yapıldığına bağlıdır.



Evet, bu kadar basit. Bir mektup yollarsınız ve cevap beklersiniz. Tabii ki, kağıt kalem yerine bayt biriminde veri kullanırsınız, fakat fikir aynı!

Sokak adı, şehir, alan kodu ve ülke adı yerine, biz IP adreslerini kullanırız. Bilgisayarınız ilk olarak DNS'den (Domain Name System - Alan Adı Sistemi) djangogirls.org adresini bir IP adresine çevirmesini ister. Bu biraz ulaşmak istediğiniz kişinin adına bakarak telefon numarası ve adresini bulabildiğiniz eski telefon rehberleri gibi çalışır.

Bir mektup gönderdiğinizde, bazı özelliklerin doğru teslim edilmesi gerekecektir: bir adres, pul vs. Ayrıca alıcının anlayacağı bir dil kullanıyorsunuz, değil mi? Aynı bir web sitesini görmek için gönderdiğiniz veri paketleri için de geçerli.

HTTP(Hypertext Transfer Protocol - HiperMetin transfer protokülü) adı verilen bir protokol kullanırız.

Basit olarak, bir web siteniz olduğunuzda, içinde yaşayacağınız bir sunucu makineniz olması gereklidir. Sunucu (bir mektupla gelen) bir istek aldığında, (başka bir mektupla) size web sitenizi gönderir.

Bu bir Django eğitimi olduğu için, Django'nun ne yaptığını soracaksınız. Bir yanıt gönderirken, herkese her zaman aynı şeyi göndermek istemezsiniz. Mektupları özellikle size yazan kişi için kişiselleştirmek çok daha iyi, değil mi? Django bu kişiselleştirilmiş, ilginç mektupları yazmanızı yardımcı olur :).

Bu kadar konuşma yeter, bir şeyler yaratma vakti!

Komut satırı arayüzüne giriş

Ha, heyecan verici, değil mi?! İlk kodunuzu birkaç dakika içinde yazacaksınız :)

Sizi yeni arkadaşınızla tanıtırıyalım: komut satırı!

Gelecek aşamalar size tüm "hacker"ların kullandığı siyah pencerenin nasıl kullanıldığını gösterecek. Başta biraz korkutucu görünebilir fakat bu sadece sizden komut bekleyen bir pencere.

Not Lütfen bu kitap boyunca 'dizin' veya 'klasör' terimlerini birbirinin yerine kullandığımızı ve aynı anlamda geldiklerini unutmayın.

Komut satırı nedir?

Genellikle **komut satırı** veya **komut satırı arabirimini** adı verilen pencere, bilgisayarlarınızdaki dosyaları görmek, düzenlemek ve yönetmek için kullanılan metin tabanlı bir uygulamadır. Típkí Windows Gezgini veya Mac'deki Finder gibi, fakat grafik arayüzü olmadan. Komut satırının diğer adları: *cmd*, *CLI*, *komut istemcisi*, *konsol* veya *terminal (uçbirim)*dir.

Komut satırı arabirimini açın

Birkaç deneme yapmak için önce komut satırı arabirimini açmamız gereklidir.

Windows

Başlat menüsü → Tüm Programlar → Donatılar → Komut Satırı.

Mac OS X

Uygulamalar → Araçlar → Terminal.

GNU/Linux

Muhtemelen Uygulamalar → Donatılar → Terminal altında olmalı, fakat sistemler arası farklılık gösterebilir. Eğer orada değilse Internet'te arayın :)

İstemci

Şu anda yüksek ihtimalle sizden komut bekleyen siyah ya da beyaz bir ekran görüyorsunuz.

Eğer Mac veya GNU/Linux kullanıyorsanız, yüksek ihtimalle `$` işaretini göreceksiniz, típkí bunun gibi:

`$`

Windows'da ise `>` işaretini göreceksiniz, bunun gibi:

`>`

Tüm komutlar bu işaret ve bir boşluktan sonra gelir fakat bunu yazmak zorunda değilsiniz. Bilgisayarınız bunu sizin için yapacaktır :)

Ufak bir not: sizin durumunuzda bu `C:\Users\ola>` veya `Olas-MacBook-Air:~ ola$` ve benzeri bir şekilde olabilir ve bu kesinlikle doğru. Bu eğitimde bunu sade ve basit bir şekilde anlatacağız.

İlk komutunuz (Yaşasın!)

Basit bir şeyle başlayalım. Aşağıdaki komutu yazın:

```
$ whoami
```

ya da

```
> whoami
```

Ve ardından `enter` tuşuna basın. Sonucumuz bu:

```
$ whoami zeynep
```

Gördüğünüz gibi, bilgisayar az önce kullanıcı adınızı yazdı. Harika, değil mi? :)

Her komutu yazmaya çalışın, kopyala-yapıştır yapmayın. Bu şekilde daha akılda kalıcı olur!

Temeller

Tüm işletim sistemleri komut satırı için birbirinden biraz farklı komutlar kullanır, bu nedenle işletim sisteminize uygun yönergeleri izlediğinizden emin olun. Deneyelim mi?

Geçerli Dizin

Nerede olduğumuzu bilmek güzel olurdu, değil mi? Bakalım. Bu komutu yazın ve `enter` tuşuna basın:

```
$ pwd  
/Users/zeynep
```

Windows'ta iseniz:

```
> cd  
C:\Users\zeynep
```

Muhtemelen makinenizde benzeri bir yazı göreceksiniz. Komut satırını açığınızda genellikle kullanıcınızın ev dizininde başlırsınız.

Not: 'pwd'nin anlamı "print working directory" yani "çalışma dizinini yazdır"dır.

Dosya ve dizinleri listele

Yani içerisinde ne var? Bilmek harika olurdu. Haydi bakalım:

```
$ ls  
Uygulamalar  
Masaüstü  
İndirilenler  
Müzik  
...
```

Windows:

```
> dir Directory of C:\Users\zeynep  
05/08/2014 07:28 PM <DIR> Uygulamalar  
05/08/2014 07:28 PM <DIR> Masaüstü  
05/08/2014 07:28 PM <DIR> İndirilenler  
05/08/2014 07:28 PM <DIR> Müzik  
...
```

Geçerli dizini değiştir

Şimdi, haydi Masaüstü dizinimize gidelim:

```
$ cd Masaüstü
```

Windows:

```
> cd Masaüstü
```

Gerçekten değişmiş mi bir bakalım:

```
$ pwd  
C:\Users\zeynep\Masaüstü
```

Windows:

```
> cd  
C:\Users\zeynep\Masaüstü
```

İşte oldu!

Profesyonel ipucu: eğer `cd D` yazarsanız ve `tab` tuşuna basarsanız, komut satırı ismin kalanını otomatik tamamlar ve gitmek istediğiniz yere daha hızlı gidersiniz. Eğer "D" ile başlayan birden çok klasör var ise, seçeneklerin listesi için `tab` tuşuna iki kez basın.

Dizin oluşturun

Uygulamalı yapmak için masaüstünüzde bir dizin oluşturmaya ne dersiniz? Bu şekilde yapabilirsiniz:

```
$ mkdir uygulama
```

Windows:

```
> mkdir uygulama
```

Bu küçük komut masaüstünüzde `uygulama` isimli bir klasör oluşturacaktır. Orada olup olmadığını kontrol etmek için `ls` veya `dir` komutlarını kullanabilirsiniz! Deneyin :)

Profesyonel ipucu: Eğer aynı komutları tekrar yazmak istemiyorsanız, `yukarı ok` ve `aşağı ok` tuşlarına basarak yazdığınız komutlar arasında geçiş yapabilirsiniz.

Alıştırma!

Sizin için ufak bir alıştırma: yeni oluşturduğunuz `uygulama` dizininde `test` adında bir dizin oluşturun. `cd` ve `mkdir` komutlarını kullanın.

Çözüm:

```
$ cd uygulama  
$ mkdir test  
$ ls  
test
```

Windows:

```
> cd uygulama  
> mkdir test  
> dir  
05/08/2014 07:28 PM <DIR>      test
```

Tebrikler! :)

Temizlik

Ortalığı dağınık bırakmak istemeyiz, haydi yaptığımız her şeyi silelim.

İlk önce masaüstüne geri dönmemiz gerek:

```
$ cd ..
```

Windows:

```
> cd ..
```

`cd` komutu ile `..` kullanmak sizin bir üst dizine götürür (Bu sizin şuanınca dizininizi tutan ana dizindir).

Nerede olduğunuzu kontrol edin:

```
$ pwd  
C:\Users\zeynep\Masaüstü
```

Windows:

```
> cd  
C:\Users\zeynep\Masaüstü
```

Şimdi `uygulama` dizinini silme vakti:

Dikkat: Dosyaları `del`, `rmdir` veya `rm` ile silme işlemi geri alınamaz, bu *silinen dosyalar sonsuza dek yok olur* anlamına gelir! Yani, bu komutları kullanırken çok dikkatli olun.

```
$ rm -r uygulama
```

Windows:

```
> rmdir /S uygulama  
uygulama, Emin misiniz <E/H>? E
```

Bitti! Gerçekten silindiğinden emin olalım:

```
$ ls
```

Windows:

```
> dir
```

Çıkış

Şimdilik bu kadar! Şimdi komut satırını güvenle kapatabilirsiniz. Bunu "hacker" tarzında yapalım, tamam mı?:)

```
$ exit
```

Windows:

```
> exit
```

Harika, değil mi?:)

Özet

İşte bazı yararlı komutların özeti:

Komut (Windows)	Komut (Mac OS - GNU/Linux)	Açıklama	Örnek
exit	exit	pencereyi kapatır	exit
cd	cd	dizin değiştir	cd test
dir	ls	dizin/dosyaları listele	dir
copy	cp	dosya kopyala	copy c:\test\test.txt c:\windows\test.txt
move	mv	dosya taşı	move c:\test\test.txt c:\windows\test.txt
mkdir	mkdir	yeni bir dizin oluştur	mkdir testdizini
del	rm	bir dosya/dizin sil	del c:\test\test.txt

Bu kullanabileceğiniz komutlardan sadece birkaçı, fakat bugün bundan daha fazlasını kullanmayacaksınız.

Eğer merak ediyorsanız, ss64.com adresinden tüm işletim sistemleri için tüm komutların kullanımına ulaşabilirsiniz.

Hazır misiniz?

Haydi Python'a giriş yapalım!

Hadi Python ile başlayalım

Nihayet buradayız!

Fakat önce, size Python'un ne olduğunu açıklayalım. Python web siteleri, oyunlar, bilimsel yazılımlar, grafikler ve çok daha fazlasını oluşturmak için kullanılan bir programlama dilidir.

Python'un kökleri 1980'li yılların sonlarına dayanmaktadır ve ana hedefi sadece makineler değil insanlar tarafından da okunabilir olmaktadır. Bu yüzden diğer programlama dillerine oranla çok daha basit görülmektedir. Öğrenmesi kolaydır ama merak etmeyin, Python gerçekten güçlündür!

Python kurulumu

Not Eğer kurulum aşamalarını zaten tamamladıysanız, tekrar yapmanıza gerek yok - direkt olarak diğer aşamaya geçebilirsiniz!

Bu bölüm Geek Girls Carrots tarafından yapılan bir eğitime dayanılarak hazırlanmıştır. (<http://django.carrots.pl/>)

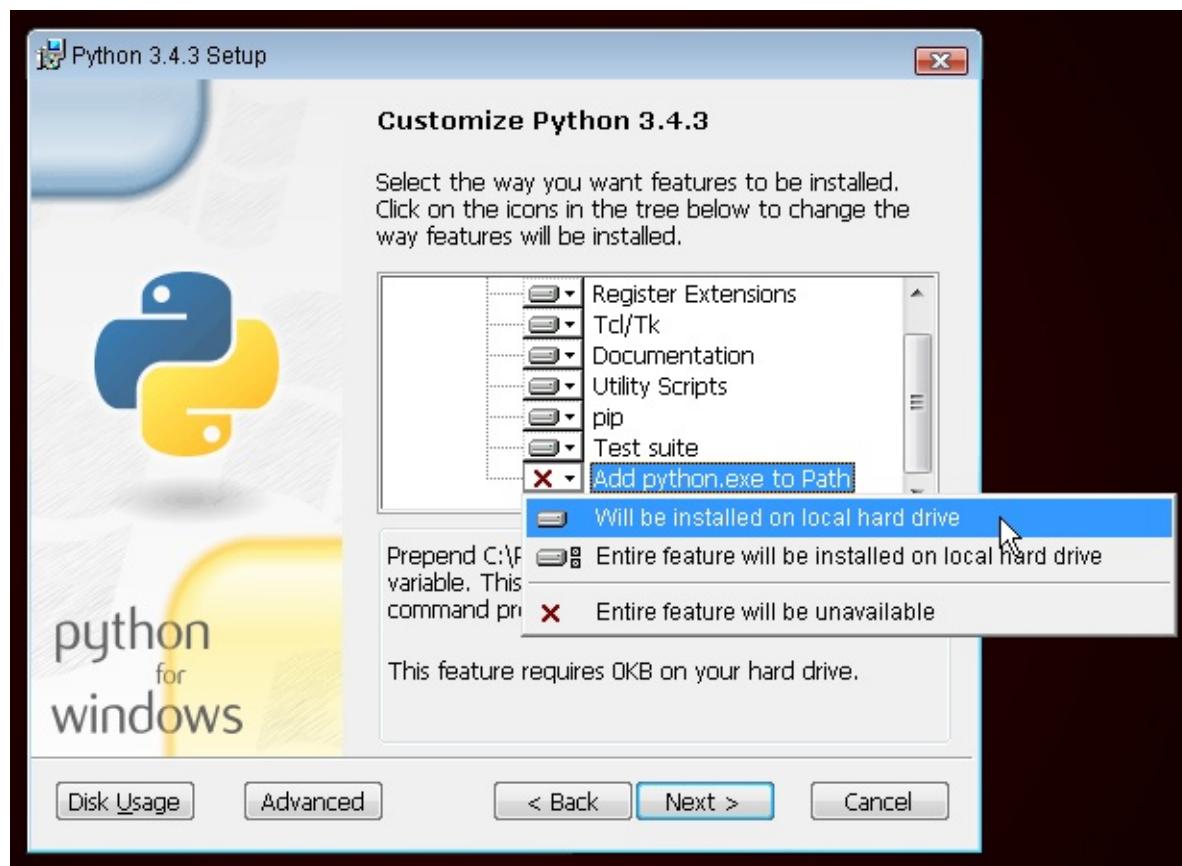
Django, Python ile yazılmıştır. Django ile bir şey yapmak için Python diline ihtiyacımız var. Hadi Python kurmaya başlayalım! Biz Python 3.4 kurmak istiyoruz, eğer daha düşük bir sürümü sahipseniz, güncellemelisiniz.

Windows

Windows için Python indirmek için resmi siteyi ziyaret edebilirsiniz: <https://www.python.org/downloads/release/python-343/>.

*.**msi** dosyasını indirdikten sonra, dosyayı çalıştırın (çift tıklayın) ve yönergeleri izleyin. Python kurulumunu yaptığınız dizinin yolunu (dizini) unutmamanız önemli. Daha sonra lazım olacak!

Dikkat: Özelleştir (Customize) olarak seçilmiş kurulum sihirbazının ikinci ekranında seçenekleri aşağıya kaydırın ve "Add python.exe to the Path" (python.exe yolunu ekle) seçeneğinin üzerine gelip "Will be installed on local hard drive" (Lokal hard diske kurulacak) seçeneğini seçin:



Linux

Muhtemelen sisteminizde Python zaten yüklüdür. Yüklü olup olmadığını (ya da hangi versiyon olduğunu) kontrol etmek için komut satırını açın ve aşağıdaki komutları girin:

```
$ python3 --version  
Python 3.4.3
```

Python yüklü değilse ya da farklı bir versiyon edinmek istiyorsanız aşağıdaki adımları takip edin:

Debian veya Ubuntu

Terminale bu komutu girin:

```
$ sudo apt-get install python3.4
```

Fedora (21'e kadar)

Terminalde kullanmanız gereken komut:

```
$ sudo yum install python3.4
```

Fedora (22+)

Terminalde kullanmanız gereken komut:

```
$ sudo dnf install python3.4
```

OS X

Python kurulum dosyasını indirmek için resmi siteye gitmelisiniz: <https://www.python.org/downloads/release/python-342/>:

- Mac OS X 64-bit/32-bit yükleyici dosyasını indirin,
- *python-3.4.3-macosx10.6.pkg* dosyasına çift tıklayarak yükleyiciyi çalıştırın.

Kurulumun başarılı olup olmadığını kontrol etmek için *Terminal* uygulamasını açın ve aşağıdaki `python3` komutunu çalıştırın:

```
$ python3 --version  
Python 3.4.3
```

Herhangi bir şüpheniz varsa, kurulumda bir şeyler ters gittiye ya da sonrasında ne yapacağınızı bilmiyorsanız eğitmene sorabilirsiniz! Bazen işler düzgün gitmiyor, bu durumda daha fazla deneyime sahip birinden yardım istemelisiniz.

Kod editörü

İlk kod satırınızı yazmak üzeresiniz, bu yüzden önce bir kod editörü edinme zamanı!

Not: Bunu daha önceki Kurulum bölümünde yaptıysanız, doğrudan bir sonraki bölüme geçebilirsiniz!

Birçok farklı kod editörü var, hangi editörü kullanacağınız kişisel tercihinize bağlı. Çoğu Python programcısı PyCharm gibi karmaşık fakat son derece güçlü IDE'leri (Integrated Development Environments-Entegre Geliştirme Ortamları) kullanır. Başlangıç seviyesi için bunlar muhtemelen pek uygun olmayacağındır. Bizim önerdiklerimiz aynı derecede güçlü fakat çok daha basit editörler olacak.

Bizim önerilerimizi aşağıda bulabilirsiniz, fakat eğitmenlerinize onların tercihlerini sormaktan çekinmeyin - onlardan yardım almak daha kolay olacaktır.

Gedit

Gedit açık kaynaklı, ücretsiz bir editördür. Tüm işletim sistemlerinde kullanılabilir.

[Buradan indirin](#)

Sublime Text 2

Sublime Text ücretsiz deneme sürümüne sahip oldukça popüler bir editördür. Kurulumu ve kullanımı basittir, tüm işletim sistemlerinde kullanılabilir.

[Buradan indirin](#)

Atom

Atom [GitHub](#) tarafından geliştirilen oldukça yeni bir editör. Atom ücretsiz, açık kaynak kodlu, kurulumu ve kullanımı basit bir editördür. Windows, OSX ve Linux işletim sistemlerinde kullanılabilir.

[Buradan indirin](#)

Neden bir kod editörü kuruyoruz?

Neden Word ya da Notepad kullanmak yerine, özel bir kod editörü yazılımı kurduğumuzu merak olabilirsiniz.

Birinci nedeni, kodun **düz metin** olması gerekliliği. Word ve TextEdit gibi programlar [RTF \(Rich Text Format\)](#) gibi özel formatları kullanarak düz metin yerine zengin metin (fontlu ve formatlı metin) üretiyorlar.

İkinci neden, kod editörleri kod düzenlemek için özelleşmişlerdir, dolayısıyla kodu anlamına göre renklerle öne çıkarma (highlighting) veya tırnakları otomatik kapama gibi yararlı özellikler sağlar.

Bütün bunları ilerde uygulama içerisinde göreceğiz. Yakında güvenilir ihtiyar kod editörünü favori araçlarınız arasında görmeye başlayacaksınız :)

Python'a Giriş

Bu bölümün bazı kısımları Geek Girls Carrots (<http://django.carrots.pl/>) öğreticisinden alınmıştır.

Biraz kod yazalım!

Python komut istemi (prompt)

Python'la oynamaya başlamadan önce bilgisayarımızda bir *komut satırı* açmamız gerekiyor. Bunu nasıl yapacağınızı artık biliyorsunuz, [Komut satırına giriş](#) bölümünde öğrenmiştiniz.

Hazır olduğunuzda, aşağıdaki talimatları takip edin.

Bir Python konsolu açmak istiyoruz; öyleyse Windows'ta `python`, Mac OS/Linux'ta `python3` yazıp, `enter` 'a basın.

```
$ python3
Python 3.4.3 (...)
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

İlk Python komutunuz!

Python komutunu çalıştırıldıktan sonra, komut istemi `>>>` şeklinde değişti. Bizim için bu, şimdi yalnızca Python dilinde komutlar kullanabileceğimiz anlamına geliyor. `>>>` yazmanıza gerek yok, Python sizin için bunu yapıyor.

Eğer herhangi bir zamanda Python komut satırından çıkmak isterseniz, yalnızca `exit()` yazmanız ya da Windows için `ctrl + z`, Mac/Linux için `ctrl + d` kısa yolunu kullanmanız yeterli. Bunu yaptığınız taktirde artık `>>>` yazısını görmeyeceksiniz.

Şu an için Python komut satırından çıkmak istemiyoruz. Bu konuda daha fazlasını öğrenmek istiyoruz. Gerçekten basit bir şeyle başlayalım. Örneğin, biraz matematik yapmayı deneyip `2 + 3` gibi bir şey yazın ve `enter` 'a basın.

```
>>> 2 + 3
5
```

Harika! Cevabın komut satırına geldiğini gördün değil mi? Python matematik biliyor! Şu gibi komutları da deneyebilirsiniz: -

```
4 * 5 - 5 - 1 - 40 / 2
```

Bunları biraz kurcalayıp eğlen, sonra tekrar burada buluşalım :).

Gördüğün üzere Python çok iyi bir hesap makinesi. Eğer başka neler yapabileceğini merak ediyorsan...

String'ler (dizeler)

Mesela ismin? İsmini tırnak işaretleri içerisinde şu şekilde yaz:

```
>>> "Zeynep"
'Zeynep'
```

İlk string'ini oluşturdun! String (katar), bilgisayar tarafından işlenebilen ve karakterlerden oluşan dizilerin genel adıdır. Bir string her zaman aynı özel karakterle başlamalı ve aynı özel karakterle bitmelidir. Tek tırnak (`'`) veya çift tırnak (`"`) olabilir (aralarında herhangi bir fark yok!). Tırnak işaretleri Python'da içlerinde olan şeyin bir string olduğunu ifade eder.

Stringler birbirlerine eklenebilir. Şunu dene:

```
>>> "Merhaba " + "Zeynep"
'Merhaba Zeynep'
```

Ayrıca stringleri bir sayı ile çarpabilirsin:

```
>>> "Zeynep" * 3
'ZeynepZeynepZeynep'
```

Eğer stringinin içerisine bir tırnak işaretini koymak istiyorsan, bunun için iki seçenek var.

Çift tırnak kullanarak:

```
>>> "Turgut Uyar'in dizeleriyiz"
"Turgut Uyar'in dizeleriyiz"
```

veya sola eğik çizgi (\) kullanarak:

```
>>> 'Turgut Uyar\'in dizeleriyiz'
'Turgut Uyar'in dizeleriyiz'
```

Hoş değil mi? İsminin tamamını büyük harf yapmak için, sadece şunu yazman yeterli:

```
>>> "Zeynep".upper()
'ZEYNEP'
```

Stringin üzerinde `upper` **fonksiyonunu** kullandın! Bir fonksiyon (`upper()` gibi), çağrıldığında(calling) Python'un bir obje (`"zeynep"`) üzerinde gerçekleştirmesi gereken bir dizi işleme denilir.

Eğer ismindeki harflerin sayısını öğrenmek istiyorsan bunun için de bir fonksiyon var!

```
>>> len("Zeynep")
6
```

Fonksiyonları neden bazen stringin sonunda bir `.` ile (`"zeynep".upper()` gibi) ve bazen de önce fonksiyonu çağrııp sonra parantez içerisine stringi yazarak kullandığımızı merak ediyor musun? Pekala, bazı durumlarda, fonksiyonlar bir takım nesnelere aittirler, mesela `upper()`, yalnızca stringler üzerinde kullanılabilir. Böyle durumlarda, bu tarz fonksiyonlara biz **method** ismini veriyoruz. Diğer durumlarda, bir fonksiyon özel olarak bir nesneye ait olmayıp, farklı çeşitlerde nesneler üzerinde de kullanılabilir, aynı `len()` gibi. İşte bu nedenle `"Zeynep"` stringini `len` fonksiyonuna bir parametre olarak veriyoruz.

Özet

Tamam, stringlerden yeterince bahsettim. Şu ana kadar şu konuları öğrendin:

- **konsol-komut istemcisi** - Python komut satırına komut(kod) yazdığında Python cevap veriyor
- **sayılar ve strings(karakter dizinleri)** - Python'da sayılar matematik ve stringler metin nesneleri için kullanılıyor
- **operators(isleçler)** + ve * gibi, değerleri birleştirerek yeni bir değer oluşturuyor
- **fonksiyonlar-islevler** - `upper()` ve `len()` gibi, nesneler üzerinde eylemler gerçekleştiriyor.

Bunlar öğreneceğiniz her programlama dilinin temelleri. Biraz daha zor bir şey için hazır mısın? İddiaya gireriz öylesin!

Hatalar

Şimdi yeni bir şey deneyelim. Bir sayının uzunluğunu, bir string'in uzunluğunu bulduğumuz gibi bulabilir miyiz? Bunu görmek için `len(304023)` yazıp `enter` a basalım:

```
>>> len(304023)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: object of type 'int' has no len()
```

İlk hatamızı aldık! Nesne türü "int" (tam sayılar, tüm sayılar) in uzunluğu olmadığını söylüyor. Şimdi ne yapabiliz? Belki de rakamı bir string olarak yazabiliz? Stringlerin bir uzunluğu var, değil mi?

```
>>> len(str(304023))
6
```

İşte yaradı! `str` fonksiyonunu `len` fonksiyonunun içinde kullandık. `str` her şeyi string'e çeviriyor.

- `str` fonksiyonu, değişkenleri **stringe** çeviriyor
- `int` fonksiyonu değişkenleri **integera** çeviriyor

Önemli: Tamsayıları yazıya çevirebiliriz, fakat yazıları(text) sayılaraya çevirememiz - `int('selamlar')` bir anlam ifade etmiyor.

Değişkenler

Programlamada en önemli konulardan biri değişkenlerdir. Değişken, daha sonra kullanmak istediğiniz bir yapıya verdığınız isimdir. Programcılar değişkenleri verileri tutmak ya da kodlarını daha okunabilir ve anlaşılabilir kılmak için kullanırlar ve böylece her şeyi sürekli akıllarında tutmaya gerek kalmaz.

`name` adında bir değişken yaratmak istediğimizi varsayalım:

```
>>> name = "Ayşe"
```

Gördünüz mü? Ne kadar kolay: `name` değişkeni "Ayşe" oldu.

Farkettiğiniz gibi, program daha öncekilerinin aksine bu kez hiçbir cevap vermedi. O zaman böyle bir değişkenin gerçekten tanımlı olduğunu nasıl bileyebiliriz? Basitçe, `name` yazıp `enter` tuşuna basalım:

```
>>> name
'Ayşe'
```

İşte bu sizin ilk değişkeniniz! `name` değişkeninin işaret ettiği şeyi her zaman değiştirebilirsiniz:

```
>>> name = "Suzan"
>>> name
'Suzan'
```

Bu değişkeni fonksiyonlar içinde de kullanabilirsiniz:

```
>>> len(name)
5
```

Harika değil mi? Tabii ki değişkenler, sayılar da dahil herhangi bir şey olabilir. Şunu deneyin:

```
>>> a = 4
>>> b = 6
>>> a * b
24
```

Peki ya değişkenin adını yanlış kullanırsak? Ne olacağını tahmin ediyor musunuz? Deneyelim!

```
>>> city = "Tokyo"
>>> ctiy
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'ctiy' is not defined
```

Bir hata! Görügünüz gibi, Python bir çok çeşit hata çeşidine sahip ve bu hatanın adı **NameError**, yani İsimlendirme Hatası. Tanımlamadığınız bir değişkenin adını kullanmaya çalışırsanız, Python size bu hatayı verir. Eğer bu hata ile daha sonra karşılaşırsanız, kodunuzdaki değişkenlerin adını doğru yazıp yazmadığınızı kontrol edin.

Bununla biraz oynayıp, neler yapabildiğinizi görün!

Yazdırma (Print) İşlevi

Şunu deneyin:

```
>>> name = 'Merve'
>>> name
'Merve'
>>> print(name)
Merve
```

Sadece `name` yazdığınız zaman, Python yorumlayıcısından 'name' değişkeninin dize olarak *temsili* döner, yani tek tırnaklar içine alınmış M-e-r-v-e harfleri. Eğer `print(name)` derseniz Python ekrana değişkenin içeriğini yazdıracaktır, bu kez tırnaklar olmaksızın, daha temiz biçimde.

Daha ileride göreceğimiz gibi `print()`, işlevlerin içindeyken bir şey yazdırmak istediğimizde ya da bazı şeyleri birden fazla satırda yazdırmak istediğimizde de kullanışlıdır.

Listeler

Python, string ve integerin yanı sıra, çok değişik türlerde nesnelere sahiptir. Şimdi, **liste** türünü tanıtacağız. Listeler tam da düşündüğünüz gibidir: diğer nesnelerin listesi olan nesne :)

Yeni bir liste yaratmakla devam edelim:

```
>>> []
[]
```

Evet, liste boş. Çok kullanışlı sayılmaz, değil mi? Hadi loto numaralarıyla liste oluşturalım. Sürekli kendimizi tekrar etmek istemeyiz, o yüzden listeyi değişkene atayalım:

```
>>> lottery = [3, 42, 12, 19, 30, 59]
```

Pekala, listeyi oluşturduk! Onunla ne yapabiliriz? Hadi listede kaç tane loto numarası olduğunu görelim. Hangi fonksiyonu kullanmanın gerekiği hakkında bir fikrin var mı? Zaten bildiğin bir fonksiyon!

```
>>> len(lottery)
6
```

Evet! `len()` listedeki nesne sayısını verir. Kullanışlı, değil mi? Belki de şu an listeyi sıralarız:

```
>>> lottery.sort()
```

Bu hiçbir cevap vermez, sadece listedeki numaraların sırasını değiştirir. Şimdi listeyi yazdıralım ve ne olduğunu görelim:

```
>>> print(lottery)
[3, 12, 19, 30, 42, 59]
```

Gördüğünüz gibi, listedeki sayılar artık küçükten büyüğe sıralı. Tebrikler!

Belki de sıralamayı ters çevirmek isteriz? Hadi yapalım!

```
>>> lottery.reverse()
>>> print(lottery)
[59, 42, 30, 19, 12, 3]
```

Kolay, değil mi? Listeye yeni bir eleman eklemek isterseniz, bu komutu yazarak yapabilirsiniz:

```
>>> lottery.append(199)
>>> print(lottery)
[59, 42, 30, 19, 12, 3, 199]
```

Sadece listedeki ilk elemanı göstermek isterseniz, **indexes** (indeksler) ile yapabilirsiniz. İndeks elemanın listede nerede olduğunu belirten numaradır. Programcılar sıfırdan başlamayı tercih ederler, bu yüzden listedeki ilk eleman listenin 0. indeksindedir, sonraki 1. indeksindedir ve böyle devam eder. Şunu deneyin:

```
>>> print(lottery[0])
59
>>> print(lottery[1])
42
```

Gördüğünüz gibi, Listedeki nesnelere listenin ismi ve köşeli parantez içindeki nesnenin indeksini kullanarak ulaşabilirsin.

Listeden eleman silmek için yukarıda öğrendiğimiz gibi **indeksleri** ve **del** komutunu kullanmanız gereklidir (del silmenin(delete) kısaltmasıdır). Bir örnekle öğretiklerimizi pekiştirelim; listeden ilk numarayı sileceğiz.

```
>>> print(lottery)
[59, 42, 30, 19, 12, 3, 199]
>>> print(lottery[0])
59
>>> del lottery[0]
>>> print(lottery)
[42, 30, 19, 12, 3, 199]
```

Kusursuz çalıştı!

Daha fazla eğlence için diğer indeksleri de deneyin: 6, 7, 1000, -1, -6 veya -1000. Denemeden önce komutların sonuçlarını tahmin etmeye çalışın. Sonuçlar mantıklı mıydı?

Bütün liste fonksiyonlarını Python dökümantasyonunun bu bölümünde bulabilirsin:

<https://docs.python.org/3/tutorial/datastructures.html>

Sözlükler (Dictionaries)

Sözlük listeye benzerdir ancak sözlük değerlerine indeks yerine anahtar ile ulaşılır. Anahtar metin veya numara olabilir. Boş bir sözlük oluşturmak için kullanılan söz dizimi şudur:

```
>>> {}
{}
```

Bu boş bir sözlük oluşturduğunuza gösterir. Yaşasın!

Şimdi, bu komutu yazmayı deneyin (kendi bilgilerinle değiştir):

```
>>> participant = {'name': 'Ayşe', 'country': 'Türkiye', 'favorite_numbers': [7, 42, 92]}
```

Bu komut ile üç anahtar-değer çiftine sahip `participant` isminde bir değişken oluşturdu:

- Anahtar `name` 'Ayşe' (`string` nesnesi) değerine işaret eder,
- `country` Türkiye (bir diğer `string`) değerine,),
- ve `favorite_numbers` [7, 42, 92] (3 numaralı bir `list`) değerine işaret eder.

Bu söz dizimi ile tek bir anahtarın içeriğini kontrol edebilirsin:

```
>>> print(participant['name'])
Ayşe
```

Gördün mü, listeye benzer. Ancak indeksini hatırlamana gerek yok - sadece ismi.

Python'a olmayan bir anahtarın değerini sorarsak ne olur? Tahmin edebiliyor musun? Hadi deneyip görelim!

```
>>> participant['age']
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyError: 'age'
```

Bir başka hata! **KeyError** hatası. Python yardımseverdir ve sana '`age`' anahtarının sözlükte bulunmadığını söyler.

Ne zaman sözlük veya liste kullanmalısın? Düşünmek için güzel bir nokta. Sonraki satırda cevaba bakmadan önce kafanızda bir çözüm oluşturun.

- Sıralı elemanlara mı ihtiyacın var? Liste ile devam et.
- İleride hızlıca (anahtarlar ile) değerlere ulaşmak istediğin için anahtarlar ile ilişkilendirilmiş değerlere mi ihtiyacın var? Sözlük kullan.

Sözlükler de listeler gibi değişimlidir (*mutable*), yani oluşturuluktan sonra değiştirilebilirler. Oluşturuluktan sonra sözlükler anahtar/değer çifti ekleyebilirsiniz, aşağıdaki gibi:

```
>>> participant['favorite_language'] = 'Python'
```

Listeler gibi, `len()` metodu sözlükteki anahtar-değer çiftlerinin sayısını bize verir. Devam edip şu komutu yazın:

```
>>> len(participant)
4
```

Umarım şu ana kadar mantıklı gelmişdir :) Sözlüklerle biraz daha eğlenceye hazır mısın? İlginç şeyler için sonraki satıra atla.

`del` komutunu sözlükten eleman silmek için kullanabilirsin. Mesela, '`favorite_numbers`' anahtarına karşılık gelen elemanı silmek istersen, sadece şu komutu yaz:

```
>>> del participant['favorite_numbers']
>>> participant
{'country': 'Türkiye', 'favorite_language': 'Python', 'name': 'Ayşe'}
```

Cıktıdan görüldüğü gibi, '`favorite_numbers`' anahtarına karşılık gelen anahtar-değer çifti silindi.

Bunun yanı sıra, sözlükteki daha önce oluşturulmuş anahtarın değerini değiştirebilirsiniz. Şunu yazın:

```
>>> participant['country'] = 'Almanya'
>>> participant
{'country': 'Almanya', 'favorite_language': 'Python', 'name': 'Ayşe'}
```

Gördüğün gibi, 'country' anahtarının değeri 'Türkiye' den 'Almanya 'ya çevrildi. :) Heyecan verici değil mi? Yaşasın! Bir başka harika şey öğrendin.

Özet

Harika! Şu an programlama hakkında birçok şey biliyorsun. Bu kısımda, şunları öğrendin:

- **hatalar** - eğer Python yazdığını komutu anlamazsa çıkan hataları nasıl okuyacağını ve anlayacağını artık biliyorsun
- **değişkenler** - daha kolay kod yazmanın sağlayan ve kodunu daha okunabilir yapan nesnelerin isimleri
- **listeler** - belirli bir sırada tutulan nesnelerin listesi
- **sözlükler** - anahtar-değer çifti olarak tutulan nesneler

Bir sonraki part için heyecanlı misiniz? :)

Karşılaştırma

Programlamanın önemli bir bölümü bir şeylerini karşılaştırmayı içerir. Karşılaştırılabilecek en kolay şey nedir? Tabii ki sayılar. Nasıl çalıştığını görelim (True = "Doğru", False= "Yanlış" demek).

```
>>> 5 > 2
True
>>> 3 < 1
False
>>> 5 > 2 * 2
True
>>> 1 == 1
True
>>> 5 != 2
True
```

Python'a birkaç sayı karşılaştırmasını söylediğimiz gibi, sadece sayıları karşılaştırmakla kalmadı, aynı zamanda metodların sonuçlarını da karşılaştırdı. Güzel değil mi?

İki sayının eşit olup olmadığını öğrenmek için neden iki tane eşittir işaretini `==` yan yana koyduk? Değişkenlere içerik verirken, tek `=` işaretini kullanıyoruz. İki sayının birbirine eşit olup olmadığını görmek için **her zaman** `==` işaretini kullanmak gerekiyor. Sayıların birbirine eşit olmaması durumunu da kontrol edebiliriz. Bunun için, yukarıdaki örnekteki gibi `!=` sembolünü kullanıyoruz.

Python' a iki görev daha verin:

```
>>> 6 >= 12 / 2
True
>>> 3 <= 2
False
```

`>` ve `<` işaretleri kolay, fakat `>=` ve `<=` ne anlama geliyor?

- `x > y` : x büyük y
- `x < y` : x küçük y
- `x <= y` : x küçük eşittir y
- `x >= y` : x büyük eşittir y

Harika! Biraz daha ister misiniz? Şunu deneyin:

```
>>> 6 > 2 and 2 < 3
True
>>> 3 > 2 and 2 < 1
False
>>> 3 > 2 or 2 < 1
True
```

Python'a istediğiniz kadar sayıyı karşılaştırınmak için verebilirsiniz, ve size hepsinin cevabını verecek. Çok akıllı değil mi?

- **and** - Mantıkta kullandığımız "ve" anlamına geliyor, yani iki taraf da True, yani doğruysa, cevap da True olacak
- **or** - Bu da "veya" anlamına geliyor, karşılaştırılan iki taraftan tek bir tanesi bile True ise bize True cevabını verecek

Portakallarla elmaları karşılaştırılabilir miyiz? Bunun Python'daki eşdeğerini deneyelim:

```
>>> 1 > 'django'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unorderable types: int() > str()
```

Gördüğünüz gibi Python tam sayılar(`int`) ve kelimeleri(yani stringleri, `str`) karşılaştırıyor. Onun yerine, **TypeError** göstererek iki farklı tipteki değişkenin karşılaştırılamayacağını söylüyor.

Boolean (Mantıksal)

Laf arasında, yeni bir Python nesne tipi öğrendiniz. Adı **boolean** olan bu tip çok kolay.

Sadece iki tane boolean nesnesi var: - True (Doğru) - False (Yanlış)

Python'un bunu anlaması için her zaman "True" (ilk harf büyük, geri kalanları küçük) yazmanız gerekiyor. `true`, `TRUE`, `tRUE işe yaramaz -- sadece True doğru.` (Aynısı "False" için de geçerli.)

Boolean'lar değişken de olabiliyor! Bakınız:

```
>>> a = True
>>> a
True
```

Ayrıca bu şekilde de yapabilirsiniz:

```
>>> a = 2 > 5
>>> a
False
```

Boolean'lar ile aşağıdaki komutları deneyerek biraz oynayın:

- `True and True`
- `False and True`
- `True or 1 == 1`
- `1 != 2`

Tebrikler! Boolean'lar programladaki en havalı özelliklerden, ve az önce onları nasıl kullanmanız gerektiğini öğrendiniz!

Kaydet!

Şimdiye kadar kodumuzu bizi sadece tek satır yazmaya limitleyen yorumlayıcı üzerinde yazdık. Normal programlar dosyalar içine kaydedilir ve programlama dilimizin **yorumlayıcısıyla** veya **derleyicisiyle** çalıştırılır. Şimdiye kadar programlarımızı Python **yorumlayıcısında** teker satır teker satır çalıştırıldı. Bundan sonraki görevlerde, birden fazla satırı

İhtiyacımız olacak, bu yüzden şunlara ihtiyacımız olacak:

- Python yorumlayıcısından çıkışın
- Seçtiğiniz kod düzenleyicisini açın
- Yeni Python dosyasına kod kaydedin
- Çalıştırın!

Kullandığımız Python yorumlayıcısından çıkmak için sadece `exit()` fonksiyonunu yazmanız yeterlidir:

```
>>> exit()
$
```

Bu sizi komut satırına geri yönlendirecektir.

Biraz önce [kod editörü](#) bölümünden bir kod editörü seçmiştık. Şimdi o editörü açmalı ve yeni bir dosya içine kod yazmalıyız:

```
print('Merhaba, Django girls!')
```

Not Kod editörlerinin en havalı özelliğini fark etmiş olmalısınız: renkler! Python konsolunda her şey aynı renkteydi, şimdi `print` fonksiyonunun stringden farklı bir renkte olduğunu görüyorsunuz. Bunun ismi "söz dizimi vurgulama" ve kod yazarken gerçekten yararlı bir özellik. Koddaki renkler sana ipucu verecektir, örneğin metin kullanım hatasında veya dildeki anahtar kelimenin yanlış yazımında (mesela fonksiyondaki `def`, aşağıda göreceğiz). Bu kod düzenleyicisi kullanma nedenlerimizden biri :)

Açıkça, artık oldukça deneyimli Python programcisisin, bu yüzden bugün öğrendiğin kodları yazmaktan çekinme.

Şimdi dosyayı tanımlayıcı bir isimle kaydetmemiz gereklidir. Dosyanın ismine **python_intro.py** diyelim ve masaüstüne kaydedelim. Dosyaya istediğimiz ismi verebiliriz, burada önemli olan kısım dosyanın **.py** uzantısı ile bitmesidir. **.py** uzantısı işletim sistemimize bu dosyanın bir **python çalıştırılabilir dosyası** olduğunu ve Python'un bu dosyayı çalıştırabileceğini belirtiyor.

Dosyayı kaydettiğimize göre artık çalıştırabiliriz! Konsoldan **Klasör değiştirme** yaparak masaüstüne ulaşın, komut satırı bölümünde öğrendiklerinizi hatırlayın.

Mac'de bu komut şunun gibi görünecektir:

```
$ cd /Users/<isminiz>/Desktop
```

Linux'ta ise bu şekilde ("Desktop" kelimesi "Masaüstü" olarak da görünebilir):

```
$ cd /home/<isminiz>/Desktop
```

Ve Windows'ta, bu şekilde olacak:

```
> cd C:\Users\<isminiz>\Desktop
```

Bir problem olursa yardım istemekten çekinmeyin.

Şimdi dosyadaki komutları çalıştırmak için Python'u kullanın:

```
$ python3 python_intro.py
Merhaba, Django girls!
```

Tamam! Bir dosyaya kaydedilen ilk Python programınızı çalıştırınız. Harika hissediyor musunuz?

Şimdi programlamanın olmazsa olmaz bir aracını öğrenme zamanı:

If...elif...else (Koşullu Akış)

Kodunuzdaki bir çok şeyi sadece belirli bir durum sağlaniyorsa çalıştırmayı isteyeceksiniz. İşte tam da bu yüzden Python'da **if deyi̇mi** isminden bir yapı bulunuyor.

python_intro.py dosyasındaki kodunuzu şununla değiştirin:

```
if 3 > 2:
```

Eğer bunu kaydedip çalıştırırsaydık şu hataya karşılaşacaktık:

```
$ python3 python_intro.py
File "python_intro.py", line 2
      ^
SyntaxError: unexpected EOF while parsing
```

Python bizden kendisine `3 > 2` durumu (veya `True`) sağlandığında neyi çalıştıracağını söylememizi bekliyor. Python'a "Çalışıyor!" yazmasını söyleyelim. **python_intro.py** dosyanızdaki kodu şununla değiştirin:

```
if 3 > 2:
    print('Çalışıyor!')
```

4 tane boşluk karakteri bıraktığımıza dikkat ettiniz mi? Bunu yaparak if cümlesine yazdığım durum doğru olduğunda neyi çalıştırması gerektiğini Python'a söylemiş oluyorum. Aslında tek bir boşlukla da yapabilirsiniz, ama hemen hemen bütün Python programcılar kodlarının temiz görünmesi için 4 boşluk bırakıyor. Tek `tab` karakteri de 4 boşluk yerine geçecektir.

Kaydedip çalıştırmayı deneyelim:

```
$ python3 python_intro.py
Çalışıyor!
```

Ya bir koşul True (Doğru) değilse?

Önceki örneklerde kod sadece koşullar sadece True olduğunda çalışıyordu. Ama Python ayrıca `elif` ve `else` ifadelerine de sahip:

```
if 5 > 2:
    print("5 gerçekten de 2'den büyütür")
else:
    print("5 2'den büyük değildir")
```

Bu kod çalışlığında aşağıdaki çıktıyı verecektir:

```
$ python3 python_intro.py
5 gerçekten de 2'den büyütür
```

Eğer 2 5'ten büyük bir sayı olsaydı ikinci komut çalışacaktır. Kolay, değil mi? Şimdi `elif` 'in nasıl çalıştığını bakalım:

```
name = 'Zeynep'
if name == 'Ayşe':
    print('Selam Ayşe!')
elif name == 'Zeynep':
    print('Selam Zeynep!')
else:
    print('Selam yabancı!')
```

ve çalıştırılınca:

```
$ python3 python_intro.py
Selam Zeynep!
```

Gördünüz mü? Eğer önceki if cümleleriniz doğru olmazsa kontrol edilmek üzere `elif` cümleleri ekleyebilirsiniz.

`if` cümlelerinden sonra istediğiniz kadar `elif` cümlesi ekleyebilirsiniz. Mesela:

```
volume = 57
if volume < 20:
    print("Çok sessiz.")
elif 20 <= volume < 40:
    print("Güzel bir fon müziği")
elif 40 <= volume < 60:
    print("Harika, her notayı duyabiliyorum")
elif 60 <= volume < 80:
    print("Parti başlasın")
elif 80 <= volume < 100:
    print("Biraz gürültülü!")
else:
    print("Kulaklarım ağrıyor! :(")
```

Python sırayla her soruyu çalıştırır ve sonucu ona göre yazar:

```
$ python3 python_intro.py
Harika, her notayı duyabiliyorum
```

Özet

Son üç alıntımda öğrendikleriniz:

- **kıyaslama yapmak** - Python'da `>`, `>=`, `==`, `<=`, `<`, `and`, `or` operatörlerini kullanarak kıyaslama yapabiliriz
- **Boolean** - İki farklı değer alabilen bir nesne tipidir: Ya `True` (doğru) olur ya da `False` (yanlış)
- **Dosya kaydetmek** - kodlarımızı dosyalara kaydederek daha büyük programları çalıştırabiliriz.
- **if...elif...else** - cümlelerini sadece belirli durumlar sağlandığında çalıştırmak istediğimiz komutlar için kullanabiliriz.

Bu bölümün son kısmının zamanı geldi!

Kendi fonksiyonlarınız!

Python'daki `len()` gibi fonksiyonları hatırlıyor musunuz? Haberler iyi - artık kendi fonksiyonlarınızı da yazabileceksiniz!

Fonksiyon Python tarafından işlenmesi gereken yönergeler dizisidir. Python'da her fonksiyon `def` anahtar kelimesi ile başlar, bir isim verilir ve bazı parameterleri olabilir. Kolay bir tane ile başlayalım. `python_intro.py` içindeki kodu aşağıdaki ile değiştirelim:

```
def hi():
    print('Merhaba!')
    print('Nasilsın?')

hi()
```

Tamam, ilk fonksiyonumuz hazır!

Fonksiyon adını neden dosyanın en altına yazdığını merak edebilirsiniz. Bunun nedeni, Python'ın dosyayı okuyup, onu yukarıdan aşağı doğru işlemesi. Yani fonksiyonumuza kullanabilmek için, onu en alt kısmda yeniden yazmalıyız.

Haydi şimdi bunu çalışıralım ve neler olacağını görelim:

```
$ python3 python_intro.py
Merhaba!
Nasilsın?
```

Bu epey kolaydı! Şimdi parametreli bir fonksiyon yazalım. Bir önceki örneği kullanabiliriz - fonksiyonumuz yine 'merhaba' desin - ama bu sefer ismini de söylesin:

```
def hi(name):
```

Gördüğünüz gibi, fonksiyonumuza `name` (isim) adında bir parametre ekledik:

```
def hi(name):
    if name == 'Ayşe':
        print('Selam Ayşe!')
    elif name == 'Zeynep':
        print('Selam Zeynep!')
    else:
        print('Selam yabancı!')

hi()
```

Unutmayın: `if` içerisindeki `print` fonksiyonundan önce dört tane boşluk var. Bunun sebebi sadece durum sağlandığında çalışmasını istememiz. Bakalım nasıl çalışıyor:

```
$ python3 python_intro.py
Traceback (most recent call last):
File "python_intro.py", line 10, in <module>
    hi()
TypeError: hi() missing 1 required positional argument: 'name'
```

Üzgünüz, bir hata. Neyse ki, Python bize oldukça yararlı bir hata mesajı veriyor. `hi()` fonksiyonun (yukarıda tanımladığımız) bir değişken kullanımını gerektirdiğini (`name` isimli) ve bizim o değişkeni fonksiyonu çağırırken iletmemeyi unuttugumuzu söylüyor. Dosyanın alt kısmında hatayı düzeltelim:

```
hi("Ayşe")
```

Ve tekrar çalışıralım:

```
$ python3 python_intro.py
Selam Ayşe!
```

Ve eğer ismi değiştirirsek ne olur?

```
hi("Zeynep")
```

Ve çalışırin:

```
$ python3 python_intro.py
Selam Zeynep!
```

Peki Ayşe veya Zeynep dışında başka bir isim yazdığımızda ne olacağını tahmin edebiliyor musunuz? Deneyin ve tahmininizin doğru olup olmadığını görün. Şunun gibi bir şey yazmalı:

```
Selam yabancı!
```

Süper değil mi? Böylece fonksiyona göndereceğiniz isim değiştiğinde aynı kodu tekrar yazmanız gereklilik kalmayacak. İşte fonksiyonlara tam da bu yüzden ihtiyacımız var - aynı kodu tekrar yazmaya gerek yok!

Hadi daha akıllıca bir şeyler yapalım -- tabii ki ikiden fazla isim var ve her isim için bir kontrol yazmak zor olurdu, değil mi?

```
def hi(name):
    print('Selam ' + name + '!')

hi("Seda")
```

Şimdi kodu çağırıralım:

```
$ python3 python_intro.py
Selam Seda!
```

Tebrikler! Az önce fonksiyonları nasıl yazacağınızı öğrendiniz! :)

Döngüler

Bu da zaten son parça. Hızlı oldu, değil mi? :)

Programcılar kendilerini tekrar etmemeyi sevmezler. Programlama tamamen işleri otomatize etmedir, bu yüzden her insanı ismiyle selam istemeyiz, değil mi? İşte burası döngülerin devreye girdiği yerdir.

Hala listeleri hatırlıyoruz değil mi? Haydi bir kızlar listesi yapalım:

```
girls = ['Seda', 'Gül', 'Pınar', 'Ayşe', 'Sen']
```

Diyelim ki hepsine merhaba demek istiyoruz. Az önce yazdığımız `hi` fonksiyonunu döngü içinde kullanabiliriz:

```
for name in girls:
```

`~for~` cümlesi `~if~` cümlesine benzer davranışır; ikisi için de dört boşluk karakterine ihtiyacımız var.

Dosyada yer alacak tam kod aşağıdadır:

```
def hi(name):
    print('Selam ' + name + '!')

girls = ['Seda', 'Gül', 'Pınar', 'Ayşe', 'Sen']
for name in girls:
    hi(name)
    print('Sıradaki')
```

Ve onu çalıştığımız zaman:

```
$ python3 python_intro.py
Selam Seda!
Sıradaki
Selam Gül!
Sıradaki
Selam Pınar!
Sıradaki
Selam Ayşe!
Sıradaki
Selam Sen!
Sıradaki
```

Gördüğünüz gibi, `for` cümlesinin içine boşluk karakteri ile koyduğunuz her şey `girls` listesi için tekrarlanıyor.

Ayrıca `for`'u `range` fonksiyonuyla beraber sayılar üzerinde de kullanabilirsiniz:

```
for i in range(1, 6):
    print(i)
```

Çalıştırırsak:

```
1
2
3
4
5
```

`range` fonksiyonu birbirini takip eden sayılarından bir liste oluşturur (bu sayıları da siz parametre olarak yazarsınız).

Sizin verdiğiniz ikinci parametrenin listede olmadığına dikkat edin. Yani `range(1, 6)` 1'den 5'e kadar sayıyor, 6 dahil edilmiyor. Çünkü "range" yarı-açık bir aralık ifade ediyor, yani ilk sayı dahil ediliyor ama sondaki sayı dahil edilmiyor.

Özet

İşte bu. **Harikasın, süpersin!** Bu bölüm biraz zordu, kendinle gurur duymalısın. Biz buraya kadar geldiğin için seninle gurur duyuyoruz!

Bir sonraki bölüme geçmeden önce kısa bir süreliğine başka birşey yapmak isteyebilirsiniz - esneyin, biraz etrafı dolaşın, gözlerinizi dinlendirin :)



Django nedir?

Django (*/dʒæŋgəʊʃ/jang-goh*) Python ile yazılmış ücretsiz ve açık kaynak bir web uygulama iskeletidir (framework). Bir web iskeleti, websitesi geliştirmeyi hızlandıran ve kolaylaştırın bir grup bileşendir.

Bir websitesi geliştirdiğinizde benzer türde bileşenlere her zaman ihtiyacınız olur: kullanıcılar için kimlik denetimi (üye olma, üye girişi, üye çıkış), websitesiz için bir yönetim paneli, formlar, dosyaları yüklemek için bir yol, vs.

Şansınıza çok önceden başka insanlar yeni bir site oluştururken web geliştiricilerinin benzer problemlerle karşı karşıya kaldığını fark etti ve bir araya gelip iskeletler (Django bunlardan biri) oluşturdu. Bu iskeletler size kullanabileceğiniz hazır bileşenler verir.

İskeletler sizi tekerleği yeniden icat etmekten kurtarır ve websitesi geliştirirken yükünüüz bir kısmını hafifletmekte yardımcı olur.

Neden bir iskelete ihtiyacınız var?

Django'nun tam olarak ne işe yaradığını anlamak için sunucular konusuna daha çok girmemiz gerekiyor. İlk bilmeniz gereken şey, sunucunuzun ondan bir web sayfası sunmasını istediğiniz bilmesi gerektiği.

Bir posta kutusunun (port) gelen mektuplar (requests) için izlendiğini düşünün. Bu bir web sunucusu tarafından yapılıyor. Sunucu mektubu okuyor ve bir web sayfası göndererek cevap veriyor. Ama bir şey gönderecekseniz, içeriğe ihtiyacınız var. Django içeriği oluşturmanıza yardımcı olan bir şeydir.

Birisini sunucunuzdan bir web sitesi istediğiinde ne olur?

Web sunucusuna bir istek geldiğinde tam olarak ne istedini çıkarmak için Django'ya geçirilir. O da önce web sayfasının adresini alır ve ne yapması gerektiğini çıkarmaya çalışır. Bu kısım Django'nun **url çözücü** (urlresolver) tarafından yapılmıyor (websitesi adresine URL - Uniform Resource Locator - deniyor, dolayısıyla *url çözücü* ismi mantıklı oluyor). Çok akıllı değil - bir kalıp listesi alıyor ve URL'yi bunlarla eşleştirmeye çalışıyor. Django kalıpları yukarıdan aşağı kontrol ediyor ve eşleşen bir şey varsa isteği (request'i) ilişkilendirilmiş işlevle (fonksiyona) geçiriyor. Bu işlev bir view'a karşılık geliyor. View kelimesi Türkçe'de görünüm anlamına gelir. Django'da özel bir terim olduğu için, biz sadece <1>view kelimesini kullanacağiz.).

Mektup dağıtan bir postacı düşünün. Sokak boyunca yürüyor ve her evin numarasını mektubun üzerindeki numara ile karşılaştırıyor. Eğer eşleşirse, mektubu oraya koyuyor. Url çözücü işte böyle çalışır!

view işlevinde her türlü ilginç şey yapılmıyor: Bir bilgi için veritabanına bakılabilir. Belki de kullanıcı veride bir şeyin değişimini istemiştir? "Lütfen iş tanımımı değiştirin" diyen bir mektup gibi. *view* buna yapmaya izniniz olup olmadığını kontrol edebilir, sonra da sizin için iş tanımınızı güncelleyip geriye "Yapıldı!" diye bir ileti gönderir. Arkasından *view* bir cevap üretir ve Django bunu kullanıcının web tarayıcısına gönderebilir.

Tabii ki yukarıdaki biraz basitleştirilmiş bir açıklama, ama şimdilik bütün teknik ayrıntıyı bilmene gerek yok. Genel bir fikrin olması yeterli.

Doğrudan çok fazla detaya girmek yerine, Django ile birşeyler oluşturacağız ve önemli kısımları yolda öğreneceğiz!

Django kurulumu

Not Eğer kurulum adımlarını zaten yaptıysanız sonraki bölüme geçebilirsiniz!

Bu bölümün bir kısmı Geek Girls Carrots tarafından hazırlanmış eğitimlere dayanılarak hazırlanmıştır (<http://django.carrots.pl/>).

Bu bölümün bir parçası Creative Commons Attribution-ShareAlike 4.0 International License ile lisanslı [django-marcador tutorial'a](#) dayanılarak hazırlanmıştır. Django-marcador tutorial'ının hakları Markus Zapke-Gründemann'e aittir.

Virtual environment (Sanal ortam)

Django'yu yüklemeden önce kod ortamınızı düzenli tutmak için son derece yararlı bir araç yükleyeceğiz. Bu adımı atlayabilirsiniz, fakat atlamanızı tavsiye ederiz. En iyi olası kurulum ile başlamanız sizin gelecekteki bir sürü sorundan koruyacaktır!

Öyleyse bir **virtual environment**(diğer adıyla *virtualenv*) kuralım. Virtualenv Python/Django kurulumunuzu her proje için ayrı tutup izole eder. Bu, bir websitesine yapacağınız değişikliklerin diğer geliştirdiklerinize yansımayacağı anlamına gelir. Muazzam, değil mi?

Yapmanız gereken tek şey `virtualenv` oluşturmak için bir dizin bulmak; örneğin giriş diziniz. Windows'da bu `C:\Users\isim` olabilir (`isim` kısmı kullanıcı adınız olacak şekilde).

Bu eğitim için giriş dizinizde yeni açtığımız `djangogirls` adlı bir klasör kullanacağız:

```
mkdir djangogirls  
cd djangogirls
```

`myvenv` adında bir virtualenv oluşturacağız. Genel komut aşağıdaki şekilde olacak:

```
python3 -m venv myvenv
```

Windows

Yeni bir `virtualenv` oluşturmak için konsolu açıp (nasıl yapıldığını birkaç adım önce anlatmıştık - hatırlıyorsunuz değil mi?) `C:\Python34\python -m venv myvenv` komutunu çalıştırın. Şöyleden görünmelii:

```
C:\Users\isim\djangogirls> C:\Python34\python -m venv myvenv
```

`C:\Python34\python` dizini önceden Python'u kurduğunuz dizin ve `myvenv` ise `virtualenv` 'inizin ismi olacaktır. İstediğiniz herhangi bir ismi kullanabilirsiniz, ama küçük harfle yazılmasına ve boşluk, aksan karakterleri (örn: å) ve özel karakterleri kullanmamaya dikkat edin. Ayrıca ismi kısa tutmak iyi bir fikir olabilir, zira bu ismi çok kullanıyor olacaksınız!

GNU/Linux ve OS X

Linux ve OS X işletim sistemlerinde `virtualenv` oluşturmak için `python3 -m venv myvenv` komutunu çalıştmak yeter. Komut şu şekilde görünecektir:

```
~/djangogirls$ python3 -m venv myvenv
```

Burada `myvenv` sizin `virtualenv` 'inizin ismi. Dilerseniz istediğiniz herhangi bir isim kullanabilirsiniz, ama büyük harf ve boşluk kullanmamaya dikkat edin. İsmi çok fazla kullanacağınız için kısa tutmak da işinize yarayacaktır.

NOT: Ubuntu 14.04 işletim sisteminde sanal ortam yaratmaya çalışırken şu hataya karşılaşabilirsiniz:

```
Error: Command '['/home/zeynep/Slask/tmp/venv/bin/python3', '-Im', 'ensurepip', '--upgrade', '--default-pip']'
returned non-zero exit status 1
```

Bu sorunu çözmek için `virtualenv` komutunu kullanabilirsiniz.

```
~/djangogirls$ sudo apt-get install python-virtualenv
~/djangogirls$ virtualenv --python=python3.4 myvenv
```

Virtualenv ile çalışmak

Yukarıdaki komutlar `myvenv` (veya seçtiğiniz isimde) bir klasör oluşturacaktır. Bu klasörde birçok dosya ve klasör bulunur.

Windows

Şu komutu çalıştırarak virtualenv'i başlatın:

```
C:\Users\Name\djangogirls> myvenv\Scripts\activate
```

GNU/Linux ve OS X

Şu komutu çalıştırarak virtualenv'i başlatın:

```
~/djangogirls$ source myvenv/bin/activate
```

`myvenv` yerine kendi seçtiğiniz `virtualenv` ismini koymayı unutmayın!

NOT: bazen `source` komutu kullanılamıyor durumda olabilir. Böyle durumlarda onun yerine aşağıdaki komutu da kullanabilirsiniz:

```
~/djangogirls$ . myvenv/bin/activate
```

Konsolunuzda şuna benzer bir şey gördüğünüzde `virtualenv` 'in başladığını anlayabilirsiniz:

```
(myvenv) C:\Users\Name\djangogirls>
```

ya da:

```
(myvenv) ~/djangogirls$
```

En başta beliren `(myvenv)` 'e dikkat edin!

Virtualenv ile çalışırken `python` otomatik olarak doğru sürümü çalıştıracaktır. Yani `python3` yerine `python` yazabilirsiniz.

Artık bütün gerekli uygulamaları bir araya getirdiğimize göre sonunda Django'yu yükleyebiliriz!

Django'yu yüklemek

`virtualenv` 'i başlattığınız için artık Django'yu `pip` kullanarak yükleyebiliriz. Konsola `pip install django==1.9` komutunu yazın. (İki tane eşittir işaretini kullandık: `==`).

```
(myvenv) ~$ pip install django==1.9
Downloading/unpacking django==1.9
  Installing collected packages: django
    Successfully installed django
Cleaning up...
```

Windows'ta

Eğer Windows işletim sisteminde pip komutunu kullanırken bir hataya karşılaşıysanız proje adresinizin boşluk, aksan veya özel karakter içerip içermediğini (`C:\Users\User Name\djangogirls` gibi) kontrol edin. Eğer durum buyusa projenizi boşluk veya özel karakter içermeyen bir adrese taşıyın; önerimiz `c:\djangogirls` olacaktır. Taşıma işleminden sonra yukarıdaki komutu tekrar deneyin.

Linux'ta

Eğer Ubuntu 12.04 işletim sisteminde pip komutunu çağırırken bir hata iletişiyle karşılaşıysanız `python -m pip install -U --force-reinstall pip` komutunu çalıştırarak pip kurulumunu onarmayı deneyin.

İşte bu kadar! Sonunda Django uygulamanızı oluşturmaya hazırlısanız!

İlk Django projen!

Bu bölümün kaynağı Geek Girls Carrots (<http://django.carrots.pl/>) eğitim materyalidir.

Bölümün parçaları Creative Commons Attribution-ShareAlike 4.0 International License ile lisanslı [django-marcador tutorial](#)'a dayanılarak hazırlanmıştır. Django-marcador tutorial'ının hakları Markus Zapke-Gründemann'e aittir.

Basit bir blog oluşturacağız!

İlk adım yeni bir Django projesi başlatmaktır. Aslında, bizim için bir Django projesinin iskeletini yaratacak Django tarafından sağlanan bazı komut dosyaları çalıştıracağız. Bu sadece daha sonra kullanacağımız dosyalar ve dizinler grubudur.

Bazı dosya ve dizinlerin isimleri Django için çok önemlidir. Oluşturmak üzere olduğumuz dosyaları yeniden adlandırmamalısınız. Onları başka bir yere taşımak da iyi bir fikir değil. Django'nun önemli şeyleri bulabilmesi için belirli yapısını koruması gereklidir.

virtualenv içindeki her şeyi çalıştırmayı unutmayın. Eğer konsolunuzda bir önek (`myvenv`) görmüyorsanız virtualenv'izi çalışır hale getirmelisiniz. **Django yükleme** bölümünün **virtualenv ile Çalışma** kısmında nasıl yapılacağını açıkladık. Windows'da `myvenv\Scripts\activate` ya da Mac OS / Linux'ta `source myvenv/bin/activate` yazmak sizin için bunu yapacaktır.

MacOS veya Linux konsolunuzda aşağıdaki komutu çalıştırmanızı; **sonuna nokta (.) koymayı unutmeyin**:

```
(myvenv) ~/djangogirls$ django-admin startproject mysite .
```

Windows'ta; **sonunda nokta (.) koymayı unutmeyin**:

```
(myvenv) C:\Users\Name\djangogirls > django-admin startproject mysite .
```

Nokta `.` bu durumda çok önemlidir çünkü; koda, Django'yı şu an bulunduğuiniz dizine kurmasını söyler. (nokta `.` şu anki dizine bir kısayoldur)

Not Yukarıdaki komutları yazarken sadece `django-admin` veya `django-admin.py` ile başlayan bölümü yazmayı unutmayın. Burada gösterilen `(myvenv) ~/djangogirls$` ve `(myvenv) C:\Users\Name\djangogirls>` kısımları, sadece, komut satırınızdaki girdilerinizi çağıracak olan komut isteği örnekleridir.

`django-admin.py` sizin için dosya ve dizinler oluşturacak bir komut dosyasıdır. Şimdi aşağıdaki gibi görünen bir dizin yapınız olmalı:

```
djangogirls
├── manage.py
└── mysite
    ├── settings.py
    ├── urls.py
    ├── wsgi.py
    └── __init__.py
```

`manage.py` site yönetimine yardımcı olan bir komut dosyasıdır. Bu dosya sayesinde, başka herhangi bir şey kurmadan bilgisayarınızda bir web sunucusunu başlatabileceğiz.

`settings.py` dosyası, web sitesinizin ayarlarını içerir.

Bir mektubu nereye götüreceğini kontrol eden postacının hakkında konuştugumuzu hatırlıyor musun? `urls.py` dosyası `urlresolver` (urlçözümleyici) tarafından kullanılan desenler listesi içerir.

Şu an için değişiklik yapmayacağımız diğer dosyaları yoksayalım. Unutmamanız gereken tek şey kazaya onları silmeyin!

Ayarları değiştirme

Hadi `mysite/settings.py` dosyasında bazı değişiklikler yapalım. Daha önceden kurduğunuz kod düzenleyicinizi kullanarak dosyayı açın.

Web sitemizin doğru bir saat sahip olması güzel olurdu. [wikipedia timezones list](#) e gidin ve ilgili saat diliminizi (TZ -time zone-) kopyalayın. (örn. `Europe/Istanbul`)

`settings.py` dosyasında `<0>TIME_ZONE` ifadesini içeren satırı bulun ve kendi seçtiğiniz zaman dilimine göre uyarlayın:

```
TIME_ZONE = 'Europe/Istanbul'
```

"Europe/Istanbul" uygun şekilde değiştirildi

Sabit dosyalar için de bir tane yol eklememiz gerekecek (Daha sonra eğitimde sabit dosyalar ve CSSlarındaki her şeyi öğreneceğiz). Dosyanın sonuna en aşağıya `STATIC_URL` girdisinin altına gidin ve `STATIC_ROOT` adında yeni bir girdi ekleyin:

```
STATIC_URL = '/static/'  
STATIC_ROOT = os.path.join(BASE_DIR, 'static')
```

Veritabanı Kurulumu

Web uygulamalarınız için farklı birçok veritabanı yazılımı vardır. Biz varsayılanı kullanacağız, `sqlite3`.

Sqlite varsayılan olduğu için zaten `mysite/settings.py` dosyamızda kurulu:

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.sqlite3',  
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),  
    }  
}
```

Blogumuzun veritabanını oluşturmak için konsolda şu komutu çalıştırın: `python manage.py migrate` (`manage.py` dosyasını içeren `djangogirls` klasöründe olmaliyiz). İşler iyi giderse şöyle bir şey görmelisiniz:

```
(myenv) ~/djangogirls$ python manage.py migrate  
Operations to perform:  
  Synchronize unmigrated apps: messages, staticfiles  
    Apply all migrations: contenttypes, sessions, admin, auth  
  Synchronizing apps without migrations:  
    Creating tables...  
      Running deferred SQL...  
    Installing custom SQL...  
    Running migrations:  
      Rendering model states... DONE  
      Applying contenttypes.0001_initial... OK  
      Applying auth.0001_initial... OK  
      Applying admin.0001_initial... OK  
      Applying contenttypes.0002_remove_content_type_name... OK  
      Applying auth.0002_alter_permission_name_max_length... OK  
      Applying auth.0003_alter_user_email_max_length... OK  
      Applying auth.0004_alter_user_username_opts... OK  
      Applying auth.0005_alter_user_last_login_null... OK  
      Applying auth.0006_require_contenttypes_0002... OK  
      Applying sessions.0001_initial... OK
```

Hepsi bu kadar! Web sunucusunu (web server) çalışma ve websitemizin çalıştığını görme zamanı!

`manage.py` dosyasının bulunduğu dizinde olmalıyız (`djangogirls` klasörü). Konsol üzerinden `python manage.py runserver` komutunu çalıştırarak web sunucusunu başlatabilirsiniz:

```
(myenv) ~/djangogirls$ python manage.py runserver
```

Eğer Windows'taysanız ve `UnicodeDecodeError` hatası varsa, bu komutu kullanın:

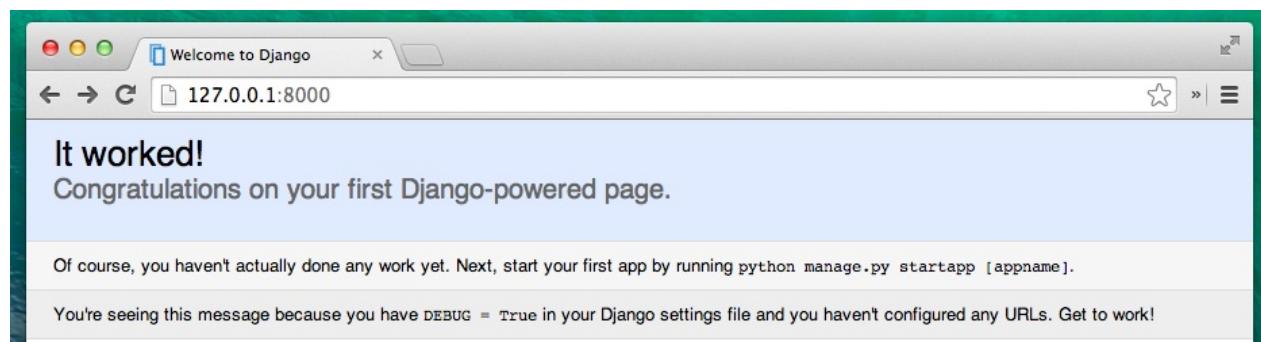
```
(myenv) ~/djangogirls$ python manage.py runserver 0:8000
```

Şimdi tek yapmanız gereken, sitenizin çalışıp çalışmadığını kontrol etmek. Tarayıcınızı (Firefox, Chrome, Safari, Internet Explorer ya da ne kullanıyorsanız) açın ve şu adresi girin:

```
http://127.0.0.1:8000/
```

Web sunucusu, siz durdurana kadar komut sistemini tutacaktır yani başka komut yazamayacaksınız. Sunucu çalışıyorken daha fazla komut girebilmek için yeni bir terminal penceresi açın ve `virtualenv`'inizi aktive edin. Web sunucusunu durdurmak için çalıştığı pencereye tekrar gelin ve CTRL+C ye -Control ve C butonlarına birlikte - basın (Windows için Ctrl+Break'e basmanız gerekiyor olabilir).

Tebrikler! ilk web siteni oluşturduğun ve web sunucusu kullanarak çalıştırın! Harika, değil mi?



Sonraki adım için hazır mısın? İçerikleri oluşturma zamanı!

Django modelleri

Şimdi blogumuzdaki bütün yazıları kaydedebileceğimiz bir şey oluşturmak istiyoruz. Ama bunu yapabilmek önce nesneler denen şeylelerden bahsetmemiz gerekiyor.

Nesneler

Programlamada `Nesneye yönelik programlama` denen bir kavram bulunuyor. Buradaki ana fikir her şeyi sıkıcı bir düzende programlama komutları ile yazmak yerine şeyleleri modelleyip birbirleri ile nasıl etkileşime geçeceklerini tanımlayabileceğimiz.

Peki bir nesne nedir? Özelliklerin ve hareketlerin bir bütünüdür. Kulağa garip geliyor olabilir ama bir örnekle açıklayacağız.

Eğer bir kediyi modellemek istiyorsak `Kedi` nesnesini oluştururuz ve bu nesne şöyle özelliklere sahip olur: `renk`, `yas`, `ruh_hali` (örneğin; iyi, kötü, uykulu ;)) ve `sahibi` (ki bu da bir `İnsan` nesnesi olur ya da eğer sokak kedisi ise bu özellik boş olabilir).

`Kedi` bazı hareketlere sahiptir: `miyavla`, `tirmala` ya da `beslen` (bu durumda kediye biraz `KediMaması` vermemiz gereklidir ki o da kendine ait özellikleri olan başka bir nesne olur. Özelliklere örnek olarak `tat` verilebilir).

```
Kedi
-----
renk
yas
ruh_hali
sahibi
miyavla()
tirmala()
beslen(kedi_maması)

KediMaması
-----
tat
```

Yani aslında ana fikir, gerçek nesneleri kod içinde özellikleriyle (`nesne özellikleri`) ve hareketleriyle (`metodlar`) tanımlamak.).

Öyleyse blog gönderilerini nasıl modelleyeceğiz? Bir blog tasarlama yapmak istiyoruz değil mi?

Cevaplamamız gereken soru: Blog gönderisi nedir? Özellikleri ne olmalıdır?

Tabii ki blog gönderimizin içeriği için yazı ve bir de başlık lazım, değil mi? Kimin yazdığını da bilsek iyi olur - dolayısı ile bir yazar da ihtiyacımız var. Son olarak, gönderinin ne zaman yaratıldığını ve yayınlandığını bilmek isteriz.

```
Post
-----
baslik
yazi
yazar
yaratilma_tarihi
yayinlanma_tarihi
```

Bir blog gönderisi ile ne tür şeyler yapılabilir? Gönderiyi yayınlayan bir `method` olması güzel olurdu, değil mi?

Bu yüzden `yayinla` metoduna ihtiyacımız olacak.

Ne elde etmek istediğimizi bildiğimize göre, haydi bunu Django'da modellemeye başlayalım!

Django modeli

Nesnenin ne olduğunu bildiğimize göre, blog gönderimiz için bir Django modeli oluşturabiliriz.

Django'da bir model özel bir çeşit nesnedir - `veritabanı` 'na kaydedilir. Veritabanı veri topluluğuna verilen isimdir. Burası, kullanıcıları, blog gönderileri gibi bilgileri saklayacağımız yerdır. Verilerimizi depolamak için SQLite veritabanını kullanacağız. Bu varsayılan Django veritabanı adaptörü - şimdilik bizim için yeterli olacaktır.

Veritabanındaki bir modeli sütunları (alan adı) ve satırları (veri) olan bir hesap çizelgesi olarak düşünebiliriz.

Uygulama oluşturma

Her şeyi derli toplu tutmak için, projemizin içinde ayrı bir uygulama oluşturacağız. Her şeyin en başından düzenli olması çok iyidir. Bir uygulama oluşturmak için aşağıdaki komutu konsolda çalıştırmanız gerekiyor (`djangogirls` dizininden `manage.py` dosyasının bulunduğu yer):

```
(myenv) ~/djangogirls$ python manage.py startapp blog
```

İçinde birkaç dosya olan yeni bir `blog` klasörü fark edeceksiniz. Projemizdeki klasörler ve dosyalar şöyle olmalı:

```
djangogirls
├── mysite
|   ├── __init__.py
|   ├── settings.py
|   ├── urls.py
|   └── wsgi.py
└── manage.py
└── blog
    ├── migrations
    |   ├── __init__.py
    |   └── __init__.py
    ├── admin.py
    ├── models.py
    ├── tests.py
    └── views.py
```

Uygulamamızı oluşturduktan sonra, Django'ya bunu kullanmasını da söylememiz lazım. Bunu `mysite/settings.py` dosyası ile yapıyoruz. `INSTALLED_APPS` dosyasını bulup `'blog'` u tam `)` karakterinin üzerine yazmamız lazım. Sonunda dosya şuna benzemelidir:

```
INSTALLED_APPS = (
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'blog',
)
```

Post (Blog gönderisi) modeli oluşturma

`blog/models.py` dosyasında `Models` deki tüm nesneler tanımlanır - burası bizim blog postunu tanımlayacağımız yerdır.

Şimdi `blog/models.py` dosyasını açalım ve içindeki her şeyi silip şu kodu yazalım:

```

from django.db import models
from django.utils import timezone

class Post(models.Model):
    yazar = models.ForeignKey('auth.User')
    baslik = models.CharField(max_length=200)
    yazi = models.TextField()
    yaratilma_tarihi = models.DateTimeField(
        default=timezone.now)
    yayinlanma_tarihi = models.DateTimeField(
        blank=True, null=True)

    def yayinla(self):
        self.yayinlanma_tarihi = timezone.now()
        self.save()

    def __str__(self):
        return self.baslik

```

`str` nin her iki tarafında 2 tane alt çizgi (`_`) kullandığınızı kontrol edin. İki alt çizgi Python dilinde sık kullanılır.

Biraz korkunç görünüyor, değil mi? Ama merak etmeyin, her şeyin ne demek olduğunu tek tek anlatacağız!

`from` veya `import` ile başlayan tüm satırlar başka yerlerden bir şeyleri projemize dahil eder. Yani, başka yerlerde tanımlanmış kodları dosyalarımıza kopyalamak yerine, bu kodların bir kısmını `from ... import ...` ile projemize dahil edebiliriz.

`class Post(models.Model):` - bu satır modelimizi tanımlar (bir `nesne` dir).

- `class` bir nesne tanımladığımızı belirten bir anahtar kelimedir.
- `Post` modelimizin ismidir. Başka bir isim de verebilirdik (yeter ki özel karakterler ve boşluk kullanmayalı). Class isimleri her zaman büyük harf ile başlamlmalıdır.
- `models.Model` Post'un bir Django Modeli olduğunu belirtir, bu şekilde Django onu veritabanında tutması gerektiğini bilir.

Şimdi daha önce bahsettiğimiz özellikleri tanımlayabiliriz: `baslik` , `yazi` , `yaratilma_tarihi` , `yayinlanma_tarihi` ve `yazar` (Türkçe karakterleri kullanmadığımızı unutmayalım). Bunun için her alanın tipini belirtmemiz lazım (metin mi? Sayı mı? Tarih mi? Başka bir nesneye referans mı, ör. Kullanıcı?).

- `models.CharField` - kısıtlı uzunlukta metin tanımlamak için kullanılır.
- `models.TextField` - bu da uzun metinleri tanımlar. Blog gönderileri için biçilmiş kaftan, değil mi?
- `models.DateTimeField` -bu da gün ve saatı tanımlamada kullanılır.
- `models.ForeignKey` -başka bir model ile bağlantı içerir.

Burada her detayı anlatmıyoruz, çünkü çok fazla vakit alır. Eğer detayları merak ederseniz veya farklı tür alanlar tanımlamak isterseniz Django'nun dokümantasyonlarına bakabilirsiniz (<https://docs.djangoproject.com/en/1.8/ref/models/fields/#field-types>).

Peki `def yayinla(self):` nedir? Daha önce bahsettiğimiz `yayinla` methodudur. `def` bir fonksiyon/method olduğunu belirtir, `yayinla` ise bu methodun adıdır. İstersen methodun ismini değiştirebilirsin. Methodlara isim verirken küçük harf kullanmaya ve boşluk yerine alt çizgi kullanmaya dikkat etmemiz gerekiyor. Örneğin ortalama fiyatı hesaplayan bir methoda `ortalama_fiyati_hesapla` ismi verilebilir.

Genellikle methodlar bir şey geri döndürür (`return` anahtar kelimesi döndür anlamına gelir). `__str__` methodunda bunun örneğini görebiliriz. Bu durumda `__str__()` methodunu çağrıdığımızda Post başlığının yazısını (`string`) elde ederiz.

Buraya kadar model hakkında anlamadığın bir şeyler varsa mentörüne sormaktan çekinme! Bu konuların biraz karmaşık olduğunun farkındayız. Özellikle hem nesneleri hem de fonksiyonları aynı anda öğrenmek kolay değil. Umarız gizemi biraz azalmaya başlamıştır!

Modeller için veritabanında tablo oluşturma

Son adımlımız yeni modelimizin veritabanına eklenmesini sağlamak. İlk önce Django'ya modelde bir takım değişiklikler yaptığımızı haber vermemiz gerekiyor (modeli yeni oluşturduk!). `python manage.py makemigrations blog` yazın. Şöyleden görünmeli:

```
(myenv) ~/djangogirls$ python manage.py makemigrations blog
Migrations for 'blog':
0001_initial.py:
- Create model Post
```

Django bize veritabanımıza uygulayabileceğimiz bir taşıma (migrasyon) dosyası oluşturdu. `python manage.py migrate blog` yazdığın zaman şunu görmelisin:

```
(myenv) ~/djangogirls$ python manage.py migrate blog
Operations to perform:
  Apply all migrations: blog
Running migrations:
  Rendering model states... DONE
  Applying blog.0001_initial... OK
```

Yaşasın! Post modelimiz artık veritabanımızda! Görsek ne güzel olur, değil mi? Gelecek bölümde Post'un nasıl göründüğünü göreceğiz!

Django admin

Modelini hazırladığımız yazıları eklemek, düzenlemek ve silmek için Django admin'i kullanacağız.

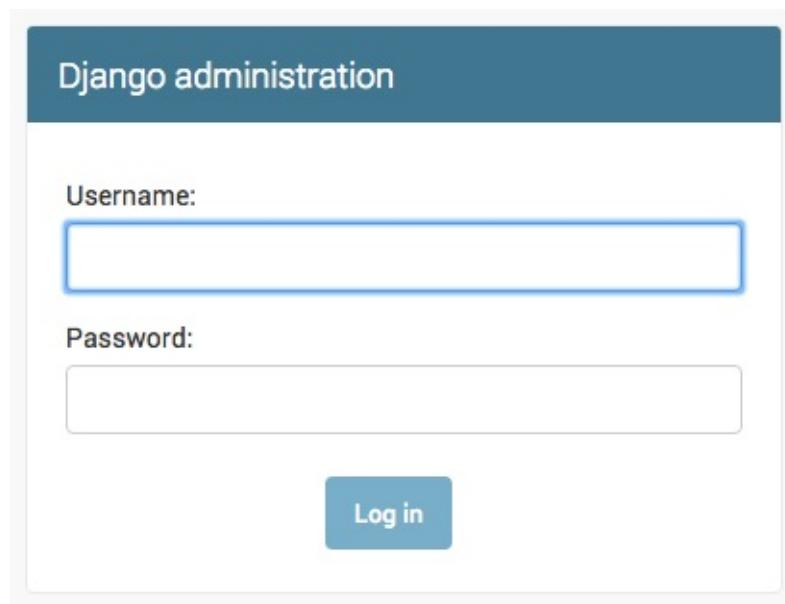
Hadi `blog/admin.py` dosyasını açalım ve içeriğini şununla değiştirelim:

```
from django.contrib import admin
from .models import Post

admin.site.register(Post)
```

Gördüğünüz gibi, bir önceki bölümde tanımladığımız Post modelini admin.py dosyamıza dahil (import) ettik. Modelimizi admin sayfasında görünür yapmak için modeli `admin.site.register(Post)` ile belirtmemiz gereklidir..

Tamam, artık admin sayfasında Post modelimize göz atabiliriz. Web sunucusunu çalıştırın komut satırında `python manage.py runserver` komutunu çalıştırmayı unutmayın. Tarayıcınızda <http://127.0.0.1:8000/admin/> adresini adres çubuğuına yazın. Aşağıdaki gibi bir giriş sayfası göreceksiniz:



Giriş yapmak için, sitede her şeyin üzerinde kontrolü olan bir kullanıcı, yani bir *superuser* oluşturmanız gerekmektedir. Komut satırında `python manage.py createsuperuser` yazın ve `enter` tuşuna basın. Giriş satırı geldiğinde, kullanıcı adınızı (küçük harfler ile ve boşluksuz), email adresinizi ve parolanızı girin. Parolayı yazarken ekranda bir şey çıkmayacaktır. Sadece yazın ve `enter` tuşuna basıp devam edin. Çıktısı aşağıdaki formatta olacaktır (kullanıcı adı ve email sizinki olacak):

```
(myvenv) ~/djangogirls$ python manage.py createsuperuser
Username: admin
Email address: admin@admin.com
Password:
Password (again):
Superuser created successfully.
```

Tarayıcınıza dönün. Oluşturduğunuz superuser'in bilgileri ile giriş yaptığınızda Django'nun admin panelini göreceksiniz.

Django administration

Site administration

AUTHENTICATION AND AUTHORIZATION

[Groups](#)

[+ Add](#) [Change](#)

[Users](#)

[+ Add](#) [Change](#)

BLOG

[Posts](#)

[+ Add](#) [Change](#)

Posts'a tıklayın ve biraz kurcalayın. Mesela üç beş tane blog yazısı ekleyin ve ne yazacağım diye düşünmeyin - zaman kazanmak için bu tutorial'dan kopyalayıp yapıştırabilirsiniz. :)

En azından iki ya da üç yazıya (ama hepsinin değil) yayılama tarihi girdiğinizden emin olun. Bunu ileriki adımlarda kullanacağız.

Django administration

WELCOME, KOJO. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home > Blog > Posts > Add post

Add post

Author:	kojo	▼	+ Add	Change
Title:	<input type="text"/>			
Text:	<input type="text"/>			
Created date:	Date: <input type="text" value="2015-12-25"/> Today Calendar	Time: <input type="text" value="20:50:01"/> Now Clock		
Published date:	Date: <input type="text"/> Today Calendar	Time: <input type="text"/> Now Clock		
<input type="button" value="Save and add another"/> <input type="button" value="Save and continue editing"/> <input type="button" value="SAVE"/>				

Eğer Django admin ile ilgili daha fazla şey öğrenmek istiyorsanız Django'nun belgelerine göz atabilirsiniz:

<https://docs.djangoproject.com/en/1.8/ref/contrib/admin/>

Şimdi enerjinizi toplamak için kendinize bir çay veya kahve alabilirsiniz. Ne de olsa ilk Django modelinizi oluşturduğunuz - biraz molayı hak ediyorsunuz. :)

Yayına alın!

Not Bir sonraki bölüm ara ara zor gelebilir. Dayanın ve bölümü bitirin; yayına alma, website geliştirme sürecinin önemli bir parçasıdır. Biraz daha uğraşmalı olan websitesini canlıya alma işine eğitmeninizin yardım edebilmesi için bu bölümü tutorial'ın ortasına yerleştirdik. Böylece eğer zaman yetmezse tutorial'ı kendi başınıza bitirebilirsiniz.

Şimdiye kadar websiteniz sadece kendi makinanızda idi, şimdi yayına nasıl alacağınızı öğreneceksiniz! Yayına alma uygulamanızı internette yayılama sürecidir, böylelikle insanlar sonunda gidip uygulamanızı görebilirler :).

Öğrendiğimiz üzere, bir websitesi bir sunucunun üstünde olmalıdır. Internette birçok sunucu sağlayıcı bulunuyor. Görece kolay bir yayına alma süreci olanlardan birini kullanacağınız: [PythonAnywhere](#). PythonAnywhere çok fazla ziyaretçisi olmayan ufak uygulamalar için ücretsiz yani sizin için kesinlikle yeterli olacaktır.

Dışarıdan kullanacağımız diğer servis bir kod barındırma hizmeti olan [Github](#). Başkaları da var, ama nerdeyse her programcının bir Github hesabı var, sizin de olacak!

Github'ı kodumuzu PythonAnywhere'e taşımak için bir atlama tahtası olarak kullanacağız.

Git

Git, birçok programcı tarafından kullanılan bir "sürüm kontrol sistemi"dir. Bu yazılım dosyaların zaman içindeki değişimlerini izler, böylelikle sonradan eski sürümlere ulaşabilirsiniz. Biraz Microsoft Word'deki "değişiklikleri izle" özelliği gibi, ama çok daha güçlü.

Git'i kurmak

Not Kurulum adımlarını halihazırda yaptıysanız, bunu tekrar yapmanıza gerek yok - bir sonraki alt bölüme geçip Git reponuzu oluşturabilirsiniz.

Windows

Git'i [git-scm.com](#) adresinden indirebilirsiniz. 5. adıma kadar "next"e basarak geçebilirsiniz. 5. adımda "Adjusting your PATH environment" dediği yerde, "Run Git and associated Unix tools from the Windows command-line" (en alttaki opsiyonu) seçin. Onun dışında, varsayılanlar iyi. Kodu çekerken Windows-stili, kodu gönderirken Unix-stili satır sonları iyidir.

MacOS

Git'i [git-scm.com](#)'den indirin ve yönergeleri izleyin.

GNU/Linux

Halihazırda yüklü değilse, git'i paket yöneticinizle indirebilirsiniz. Şunlardan birini deneyin:

```
sudo apt-get install git  
# veya  
sudo yum install git
```

Git repomuzu oluşturmak

Git, kod reposu (veya "repository") denen belli dosyaların değişikliklerini izler. Projemiz için bir tane oluşturalım. Konsolunuza açın ve `djangogirls` klasöründe aşağıdaki komutları çalıştırın:

Not Reponuzu başlatmadan önce, `pwd` (OS/Linux) veya `cd` (Windows) komutu ile bulunduğuuz dizini kontrol edin. `djangogirls` dizininde olmanız gerekiyor.

Hatırlatma: Kullanıcı adı seçerken özel Türkçe karakter kullanmayın.

```
$ git init
Initialized empty Git repository in ~/djangogirls/.git/
$ git config --global user.name "Adınız"
$ git config --global user.email you@example.com
```

Git reposunu başlatma işi, proje başına bir kere yapmamız gereken birşey (ayrıca kullanıcı adı ve eposta adresini tekrar girmenize gerek olmayacak).

Git bu dizindeki tüm dizin ve dosyalardaki değişiklikleri kaydedecek, ama takip etmemesini istediğimiz bazı dosyalar var. Bunu dizinin dibinde `.gitignore` adında bir dosya oluşturarak yapıyoruz. Editörünüzü açın ve aşağıdaki içeriklerle yeni bir dosya yaratın:

```
*.pyc
__pycache__
myvenv
db.sqlite3
.DS_Store
```

Ve "djangogirls" dizinin en üst seviyesine `.gitignore` olarak kaydedin.

Not Dosya adının başındaki nokta önemli! Eğer dosyayı oluştururken zorlanırsanız (örneğin Mac'ler Finder ile nokta ile başlayan dosya yaratmanızdan hoşlanmıyor), editörünüzdeki "Farklı Kaydet" özelliğini kullanın, kesin çalışır.

`git add` kullanmadan önce nelerin değiştiğinden emin değilseniz, `git status` komutunu kullanmakta yarar var. Bu, yanlış dosyaların eklenmesi ve gönderilmesi gibi istenmeyen sürprizlerin engelenmesine yardımcı olacak. `git status` komutu, takip edilmeyen/değişen/gönderilecek dosyalar (staged), dal durumu (branch status) gibi bilgiler verir. Çıktının aşağıdaki gibi olması gerekiyor:

```
$ git status
On branch master

Initial commit

Untracked files:
  (use "git add <dosya>..." to include in what will be committed)

    .gitignore
    blog/
    manage.py
    mysite/

nothing added to commit but untracked files present (use "git add" to track)
```

Ve son olarak değişikliklerimizi kaydediyoruz. Komut satırına gidin ve aşağıdaki komutları çalıştırın:

```
$ git add -A
$ git commit -m "Django Girls uygulamam, ilk commit"
[...]
13 files changed, 200 insertions (+)
create mode 100644 .gitignore
[...]
create mode 100644 mysite/wsgi.py
```

Kodunuzu Github'a gönderme

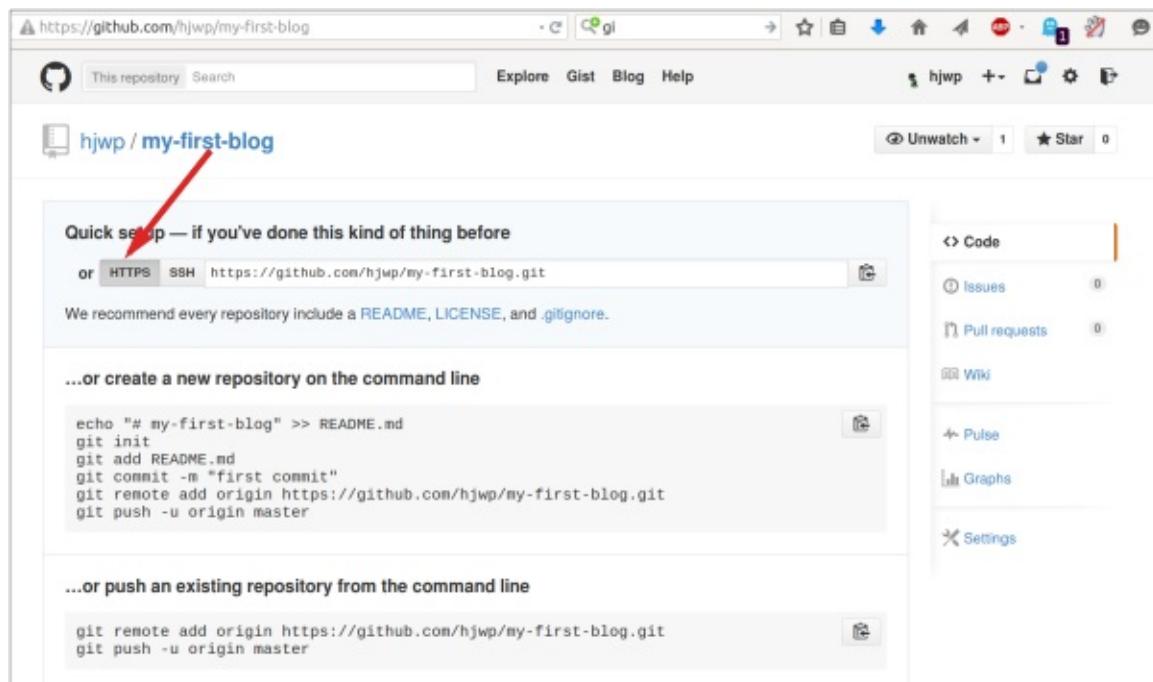
[GitHub.com](#)> adresine gidin ve kendinize yeni bedava bir kullanıcı hesabı açın. (Bunu atölye hazırlıklarında zaten yaptıysanız, harika!)

Arkasından "ilk-blogum" (veya "my-first-blog") isminde yeni bir repo oluşturun. "Initialize with a README" kutucuğunu işaretlemeden bırakın, .gitignore opsyonunu boş bırakın (onu elle yaptık) ve 'License'i 'None' olarak bırakın.

The screenshot shows the GitHub repository creation interface. The 'Repository name' field is set to 'my-first-blog'. Under 'Description (optional)', there is a text input field. Below it, there are three options: 'Public' (selected), 'Private' (unchecked), and 'Initialize this repository with a README' (unchecked). A red arrow points to the 'Initialize this repository with a README' checkbox. At the bottom, there are two dropdown menus: 'Add .gitignore: None' and 'Add a license: None'. A green 'Create repository' button is at the bottom left.

Not `ilk-blogum` ismi önemli -- başka birşey de seçebilirsiniz, ama aşağıdaki yönerelerde çok geçiyor, her seferinde değiştirmeniz gereklidir. En kolay `ilk-blogum` ismi ile devam etmek.

Bir sonraki ekranda, repo'yuzu klonlamak için gereken URL'yi göreceksiniz. "HTTPS"li versiyonunu seçin, kopyalayın. Birazdan onu komut penceresine yapıştıracağınız:



Şimdi bilgisayarınızdaki Git reposunu Github'daki repo ile ilişkilendirmemiz gerekiyor.

Aşağıdakini komut satırına yazın (`<github-kullanıcı-adınız>` kısmını Github hesabını yarattığınız sırada kullandığınız kullanıcı adı ile değiştirin, büyükür küçükür işaretlili eklemeyin):

```
$ git remote add origin https://github.com/<github-kullanıcı-adınız>/ilk-blogum.git  
$ git push -u origin master
```

Github kullanıcı adı ve şifrenizi girin, arkasından aşağıdakine benzer bir şey görmeniz gerekiyor:

```
Username for 'https://github.com': zeynep  
Password for 'https://zeynep@github.com':  
Counting objects: 6, done.  
Writing objects: 100% (6/6), 200 bytes | 0 bytes/s, done.  
Total 3 (delta 0), reused 0 (delta 0)  
To https://github.com/zeynep/ilk-blogum.git  
 * [new branch] master -> master  
Branch master set up to track remote branch master from origin.
```

Kodunuz artık Github'da. Siteye girin ve kontrol edin! İyi bir çevrede olduğunu göreceksiniz - [Django, the Django Girls Tutorial](#), ve daha birçok harika açık kaynak yazılım projesi de kodlarını Github'da tutuyor :)

Blogumuzun PythonAnywhere üzerinde kurulumu

Not En baştaki kurulum adımlarında PythonAnywhere hesabını açmış olabilirsiniz - öyleyse bu kısmı tekrar yapmanız gereklidir.

Sırada PythonAnywhere sitesinde bedava bir "Beginner" (yeni başlayan) hesabı açma işi var.

- www.pythonanywhere.com

Not Burada kullanıcı isminizi seçerken bilin ki blogunuzun URL'si `kullanıcıadınız.pythonanywhere.com` şeklinde olacak. O yüzden ya kendi rumuzunuzu(nickname) seçin ya da blogunuzun konusu ile ilgili bir isim seçin.

Kodunuzu PythonAnywhere üzerine çekmek

PythonAnywhere'de hesap açtığınızda, 'dashboard' (gösterge paneli) sayfanıza veya "Consoles" sayfasına yönlendirileceksiniz. "Bash" konsolu başlatma seçeneğini seçin -- bu bilgisayarınızdaki konsolun PythonAnywhere versiyonu.

Not PythonAnywhere Linux tabanlı, o yüzden kendi makinanız Windows ise konsol biraz farklı gözükecektir.

Reponuzun bir klonunu yaratarak kodumuzu Github'dan PythonAnywhere üzerine çekelim. Aşağıdakileri PythonAnywhere konsoluna yazın (`<github-kullanıcı-adınız>` yerine kendi Github kullanıcı adınızı yazmayı unutmayın):

```
$ git clone https://github.com/<github-kullanıcı-adınız>/ilk-blogum.git
```

Bu kodunuzun bir kopyasını PythonAnywhere üzerine indirecektir. `tree ilk-blogum` yazarak kontrol edin:

```
$ tree ilk-blogum
ilk-blogum/
└── blog
    ├── __init__.py
    ├── admin.py
    ├── migrations
    │   └── 0001_initial.py
    │       └── __init__.py
    ├── models.py
    ├── tests.py
    └── views.py
├── manage.py
└── mysite
    ├── __init__.py
    ├── settings.py
    ├── urls.py
    └── wsgi.py
```

PythonAnywhere üzerine bir virtualenv (sanal ortam) oluşturmak

Bilgisayarınızda nasıl bir virtualenv (sanal ortam) oluşturduysanız, aynı şekilde PythonAnywhere üzerinde de oluşturabilirsiniz. Bash konsoluna, aşağıdakileri yazın:

```
$ cd ilk-blogum

$ virtualenv --python=python3.4 myvenv
Running virtualenv with interpreter /usr/bin/python3.4
[...]
Installing setuptools, pip...done.

$ source myvenv/bin/activate

(myvenv) $ pip install django whitenoise
Collecting django
[...]
Successfully installed django-1.8.2 whitenoise-2.0
```

Not `pip install` birkaç dakika sürebilir. Sabır, sabır! Ama 5 dakikadan uzun sürüyorsa, birşeyler yanlış olmuştur. Eğitimmeninize sorun.

Statik dosyaların toplanması.

"whitenoise"un ne olduğunu merak ettiniz mi? "Statik dosyalar" için kullanılan bir araç. Statik dosyalar, HTML veya CSS dosyaları gibi düzenli olarak değişmeyen veya kod çalıştırmayan dosyalardır. Bu dosyalar sunucularda bilgisayarımızdakinden farklı çalışırlar. Bu yüzden onları sunucudan yaynlamak için "whitenoise" gibi bir araca ihtiyacımız var.

Tutorial'ın ilerleyen kısımlarında sitemizin CSS'ini düzenlerken statik dosyalar konusuna biraz daha fazla gireceğiz.

Şimdilik sadece sunucuda `collectstatic` diye ek bir komut çalıştıracağız. Bu komut, Django'ya sunucdaki bütün statik dosyaları toparlamasını söyler. An itibarıyle bunlar çoğunlukla admin sitesini güzelleştiren dosyalar.

```
(myvenv) $ python manage.py collectstatic

You have requested to collect static files at the destination
location as specified in your settings:

/home/zeynep/ilk-blogum/static

This will overwrite existing files! (Bu işlem halihazırda dosyalarınız üzerinde değişiklik yapar!)
Are you sure you want to do this? (Bu işlemi yapmak istediğinizden emin misiniz?)

Type 'yes' to continue, or 'no' to cancel: yes (Onaylıyorsanız 'yes', vazgeçtiyorsanız 'no' yazın)
```

"yes" yazın ve işte başladı! Bilgisayarlara sayfa sayfa yazı yazdırmayı sevmiyor musunuz? Ben hep beraberinde küçük küçük sesler çıkarırım. Trr, trr, trr...

```
Copying '/home/zeynep/ilkbogum/mvenv/lib/python3.4/site-packages/django/contrib/admin/static/admin/js/actions.min.js'
Copying '/home/zeynep/ilkbogum/mvenv/lib/python3.4/site-packages/django/contrib/admin/static/admin/js/inlines.min.js'
[...]
Copying '/home/zeynep/ilkbogum/mvenv/lib/python3.4/site-packages/django/contrib/admin/static/admin/css/changelists.css'
Copying '/home/zeynep/ilkbogum/mvenv/lib/python3.4/site-packages/django/contrib/admin/static/admin/css/base.css'
62 static files copied to '/home/zeynep/ilkbogum/static'.
```

PythonAnywhere üzerinde veritabanının oluşturulması

Bilgisayarınız ve sunucu arasında farklı olan bir başka şey daha: farklı bir veritabanı kullanıyor. Dolayısıyla bilgisayarlarınızdaki ve sunucudaki kullanıcı hesapları ve blog yazıları farklı olabilir.

Sunucudaki veritabanına aynen bilgisayardaki gibi `migrate` (taşımak) ve `createsuperuser` (yetkili bir kullanıcı oluşturmak) komutlarıyla oluşturup ilk örnek verilerle ile doldurabiliriz:

```
(myenv) $ python manage.py migrate
Operations to perform:
[...]
Applying sessions.0001_initial... OK
(myenv) $ python manage.py createsuperuser
```

Blog'umuzu web uygulaması olarak yayınılama

Artık kodumuz PythonAnywhere üzerinde, virtualenv'imiz hazır, statik dosyalar toplandı, ve veritabanı hazırlandı. Blogumuzu bir web uygulaması olarak yayılmaya hazırlız!

PythonAnywhere logosuna tıklayarak 'dashboard'a geri gidin, burda **Web** sekmesine tıklayın. En son **Add a new web app** (yeni bir web uygulaması yaratın) linkine tıklayın.

Açılan pencerede alan adınızı kontrol edin, **manual configuration** (elle konfigürasyon)'ı seçin ("Django" opsyonunu *değil*). Arkasından **Python 3.4**'ü seçin ve işlemi bitirmek için 'Next'e basın.

Not "Manual configuration" seçeneğini seçtiğinizden emin olun, "Django" seçeneğini değil. Hazır PythonAnywhere Django kurulumunu seçmek için fazla havalıyız ;-)

virtualenv'in (sanal ortamın) ayarlanması

Son bahsettiğimiz adım siz web uygulamanızın PythonAnywhere ayar ekranına getirecek. Sunucudaki uygulamanızda değişiklik yapmak istediğinizde bu ekranı kullanmanız gerekiyor.

The screenshot shows the PythonAnywhere web configuration interface for the domain `edith.pythonanywhere.com`. The 'Code' section displays the current running code status. Below it, the 'Virtualenv' section contains a text input field with the path `/home/edith/my-first-blog/myvenv`, which has a checkmark icon next to it. Two red arrows point to this checkmark and the input field.

"Virtualenv" bölümünde "Enter the path to a virtualenv" (virtualenv için dizin yolu girin) linkini tıklayın ve şunu yazın: `/home/<kullanıcı-adınız>/ilk-blogum/myvenv`. Devam etmeden önce, dizin yolunu kaydetmek için tik işaretli olan mavi kutuyu tıklayın.

Not İlgili yeri kendi kullanıcı adınızı yazın. Eğer hata yaparsanız, PythonAnywhere size bir uyarı gösterecektir.

WSGI dosyasının ayarlanması

Django, "WSGI protokolü"nü kullanarak çalışır. WSGI, PythonAnywhere'in de desteklediği Python kullanan web sitelerinin servis edilmesi için kullanılan bir standart. PythonAnywhere'in Django blogumuzu anlaması için WSGI ayar dosyasını düzenleyiyoruz.

"WSGI Configuration file" (WSGI ayar dosyası) linkine tıklayın ("Code" denen kısımda, sayfanın üst tarafında `-- adı /var/www/<kullanıcı-adınız>_pythonanywhere_com_wsgi.py` 'a benzer birşey olacak). Burdan bir edöitorye yönlendirileceksiniz.

Tüm içeriği silin ve onların yerine aşağıdakileri yazın:

```
import os
import sys

path = '/home/<kullanıcı-adınız>/ilk-blogum' # burada kendi kullanıcı adınızı yazın
if path not in sys.path:
    sys.path.append(path)

os.environ['DJANGO_SETTINGS_MODULE'] = 'mysite.settings'

from django.core.wsgi import get_wsgi_application
from whitenoise.django import DjangoWhiteNoise
application = DjangoWhiteNoise(get_wsgi_application())
```

Not `<kullanıcı-adınız>` diye geçen kısma kendi kullanıcı adınızı yazmayı unutmayın

Bu dosyanın işi, PythonAnywhere'e web uygulamamızın nerde yaşadığı ve Django ayar dosyasının adının ne olduğunu söylemek. Aynı zamanda "whitenoise" statik dosya aracını ayarlıyor.

Save (kaydet)'e basın. Arkasından **Web** sekmesine geri gidin.

Hazırız! Yeşil ve büyük **Reload** (Yeniden yükle) butonuna tıklayın. Uygulamanıza girip görebileceksiniz. Sayfanın tepesinde uygulamaya giden linki bulabilirsiniz.

Hata ayıklama önerileri

Eğer sitenize girdiğinizde bir hata görürseniz, hata ayıklama bilgileri için ilk bakacağınız yer, **error log'unuz** (hata kayıtlarınız). Buraya olan linki PythonAnywhere'deki [Web sekmesinde](#) bulabilirsiniz. Burda hata mesajları olup olmadığına bakın; en yeni mesajlar en altta. Sık karşılaşılan problemler şunlar:

- Konsolda yaptığımız adımlardan birinin unutulması: virtualenv'in oluşturulması, çalışır hale getirilmesi, içine Django'nun kurulumu, collectstatic'in çalıştırılması, veritabanının taşınması (migrate ettirilmesi).
- Web sekmesinde virtualenv dizin yolunda bir hata yapılması -- eğer problem varsa, ilgili yerde küçük kırmızı bir hata mesajı olur.
- WSGI ayar dosyasına bir hata yapmak -- ilk-blogum dizinine olan yolu doğru yazdığınızdan emin misiniz?
- Virtualenv'ınız için seçtiğiniz Python versiyonu web uygulamanız için seçtiğiniz Python versiyonu ile aynı mı? İkisinin de 3.4 olması gerekiyor.
- [Python Anywhere vikisinde \(bilgi sayfalarında\)](#) genel hata ayıklama önerileri bulunuyor.

Ve eğitmeniniz size yardıma hazır, unutmayın!

Siteniz canlıda!

Sitenizin varsayılan sayfası "Welcome to Django" diyor olmalı, aynen bilgisayarlarınızda olduğu gibi. URL'nin sonuna `/admin/` yazın, 'giriş' tuşuna bastığınızda admin sitesi açılacak. Kullanıcı adı ve şifrenizle giriş yapın, sunucuda yeni blog yazıları girebildiğini göreceksiniz.

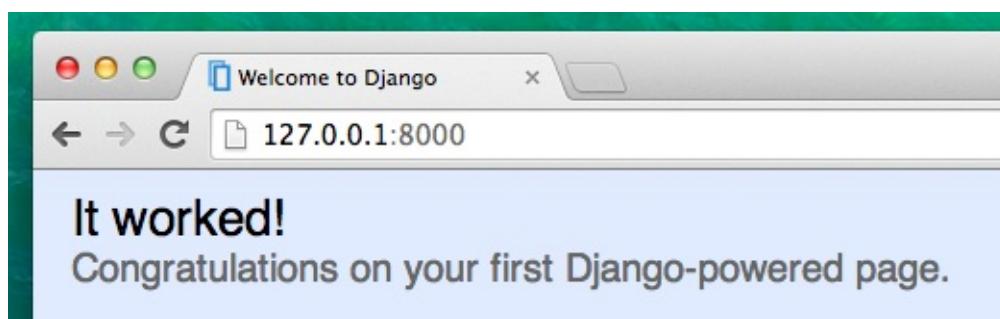
Kendinize **KOCAMAN** bir aferin diyin! Yayına alma web geliştirme işinin en uğraştırmalı kısımlarından biridir ve genelde çalışana kadar insanların birkaç gününü alır. Ama işte siteniz canlıda, gerçek internette!

Django url'leri

İlk web sayfamızı yapmak üzereyiz: blogunuzun anasayfası! Ama önce, biraz Django url'lerini öğrenmeye başlayalım.

URL nedir?

URL basitçe bir web adresidir. Bir web sayafasını her ziyaret ettiğinizde tarayıcınızın adres barında bir URL görürsünüz (evet! `127.0.0.1:8000` bir URL'dir! Ve `https://djangogirls.com` da bir URL'dir):



Internetteki her sayfanın kendi URL'inin olması gereklidir. Böylelikle bir URL açıldığında uygulama ne göstermesi gerektiğini bilir. Django'da `URLconf` (URL konfigürasyonu) denilen bir şey kullanıyoruz. URLconf, Django'ya gelen URL için doğru view'un bulunmasını sağlar.

URL'ler Django'da nasıl çalışır?

Kod editörümüzde `mysite/urls.py` dosyasını açalım ve neye benzediğine bakalım:

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    # Examples:
    # url(r'^$', 'mysite.views.home', name='home'),
    # url(r'^blog/', include('blog.urls')),

    url(r'^admin/', include(admin.site.urls)),
]
```

Gördüğünüz gibi Django bizim için bir şeyler koymuş bile.

`#` ile başlayan satırlar yorum satırlarıdır - bu satırlar Python tarafından çalıştırılmayacak manasına gelir. Çok pratik, değil mi?

Geçen bölümde gittiğimiz admin URL şimdiden burda:

```
url(r'^admin/', include(admin.site.urls)),
```

`admin` ile başlayan her URL için Django ona denk gelen bir view bulur manasına gelir. Bu şekilde bir sürü admin URLlerini ekliyoruz böylece hepsi bu küçük dosyanın içinde sıkıştırılmış bir şekilde durmuyor -- bu hali daha okunabilir ve düzenli.

Regex (Kurallı İfade)

Django'nun URL'leri view'larda nasıl eşleştirdiğini merak ediyor musunuz? Bu kısım biraz karışık. Django bunun için `regex` kullanıyor. Regex, "regular expressions"ın kısaltılmış hali ve düzenli ifadeler anlamına geliyor. Regex'in bir arama kalıbı oluşturmak için birçok (birçok!) kuralı var. Regexler ileri bir konu olduğu için nasıl çalıştığını detayına girmeyeceğiz.

Gene de kalıpları nasıl oluşturduğumuzu anlamak isterseniz, aşağıdaki bir örnek var - aradığımız kalıbı oluşturmak için kuralların sadece bir kısmına ihtiyacımız olacak, şöyle:

```
^ metnin başlangıcı için
$ metnin sonu için
\d rakamlar için
+ bir önceki karakterin en az bir kere bulunması gerektiğini belirtmek için
() kalıbin belli bir kısmını yakalamak için
```

Url tanımındaki diğer herşey birebir eşlenecek.

Şimdi `http://www.mysite.com/post/12345/` adresinde bir websitemiz olduğunu düşünelim. `12345` da gönderimizin numarası.

Her gönderi için ayrı bir view yazmak gerçekten can sıkıcı olurdu. Düzenli ifadelerle (regexlerle) url ile eşleşecek bir kalıp oluşturup gönderi numarasını çıkartabiliriz: `^post/(\d+)/$`. Parçalara bölüp ne yaptığımıza bakalım:

- `^post/` Django'ya `post/` ile başlayan her şeyi almasını söylüyor (`^`)
- `(\d+)` ise bir sayı (birden fazla rakam) olduğunu ve bu sayıyı yakalamak ve çıkarmak istediğimizi belirtiyor
- `/` ise Django'ya arkasından bir `/` karakteri gelmesi gerektiğini söylüyor
- `$` ise URL'nin sonuna işaret ediyor, yani sadece sonu `/` ile biten string'ler bu kalıpla eşleşecek

İlk Django url'iniz!

İlk URL'imizi oluşturma zamanı! '<http://127.0.0.1:8000/>'ın blogumuzun ana sayfası olmasını istiyoruz ve bize bir gönderi listesi göstermesini istiyoruz.

Aynı zamanda `mysite/urls.py` dosyasını basit tutmak istiyoruz, bunun için ana `mysite/urls.py` dosyasına `blog` uygulamamızdan url'leri import edeceğiz (içeri alacağız).

Yorum satırlarını silin (`#` ile başlayan satırları) ve ana url'ye `blog.urls` satırlarını import edecek (içeri alacak) bir satır ekleyin (`..`).

`mysite/urls.py` dosyanız şöyle olmalıdır:

```
from django.conf.urls import include, url
from django.contrib import admin

urlpatterns = [
    url(r'^admin/', include(admin.site.urls)),
    url(r'', include('blog.urls')),
]
```

Django artık '<http://127.0.0.1:8000/>'ye gelen her şeyi `blog.urls` 'ya yönlendirecek ve ordaki yönergelere bakacak.

Python'da düzenli ifadeler her zaman string'in başına `r` ekleyerek yapılır. Bu Python için string'in özel karakterler içerdigini, doğrudan Python için değil düzenli ifadeler için bir string olduğu konusunda ipucu verir.

blog.urls

`blog/urls.py` adında yeni boş bir dosya oluşturun. Harika! Şu iki satırı ekleyin:

```
from django.conf.urls import url
from . import views
```

Burada sadece Django'nun methodlarını ve `blog` uygulamasındaki tüm `view`leri içeri aktarıyoruz (uygulamamız henüz yok, ama birazdan o kısma da geleceğiz!)

Bundan sonra ilk URL kalibimizi ekleyebiliriz:

```
urlpatterns = [
    url(r'^$', views.post_list, name='post_list'),
]
```

Gördüğünüz üzere, `^$` URL'sine `post_list` adında bir `view` atıyoruz. Bu düzenli ifade `^` (başlangıç) ve `$` (bitiş)'e uyan stringlerle eşleşir - yani sadece boş string'lerle eşleşir. Bu doğru çünkü Django URL çözüçülerinde '`http://127.0.0.1:8000/`' URL'nin parçası değildir. Bu kalıp, Django'ya eğer siteye biri '`http://127.0.0.1:8000/`' adresinden gelirse gitmesi gereken yerin `views.post_list` olduğunu söylüyor.

Son kısım olan `name='post_list'` `view`'u tanımlamak için kullanılan URL'nin adı. Bu `view`'un adı ile aynı olabilir ama tamamen farklı bir şey de olabilir. Named URL'leri (isimlendirilmiş URL'leri) projenin ilerleyen kısımlarında kullanacağımız, o yüzden uygulamadaki her URL'yi isimlendirmemiz önemli. Aynı zamanda URL isimlerini tekil ve kolay hatırlanabilir yapmamız gereklidir.

Her şey tamam mı? Tarayıcınızda `http://127.0.0.1:8000/ye` gidin ve sonuçları görün.

```
return _bootstrap._gcd_import(name[level:], package, level)
File "<frozen importlib._bootstrap>", line 2254, in _gcd_import
File "<frozen importlib._bootstrap>", line 2237, in _find_and_load
File "<frozen importlib._bootstrap>", line 2226, in _find_and_load_unlocked
File "<frozen importlib._bootstrap>", line 1200, in _load_unlocked
File "<frozen importlib._bootstrap>", line 1129, in _exec
File "<frozen importlib._bootstrap>", line 1471, in exec_module
File "<frozen importlib._bootstrap>", line 321, in _call_with_frames_removed
File "/Users/dana/Dana-Files/Codes/djangogirls/blog/urls.py", line 5, in <module>
    url(r'^$', views.post_list, name='post_list'),
AttributeError: 'module' object has no attribute 'post_list'
```

Artık "It works" demiyor, di mi? Meraklanmayın, sadece bir hata sayfası, korkacak birşey yok! Aslında çok kullanışlılar:

Sayfada gördüğünüz şey: **no attribute 'post_list'**. `post_list` size bir şey hatırlatıyor mu? Bu `view`'un ismi! Bu her şey yerinde sadece henüz `view` yok manasına geliyor. Hiç merak etmeyin, oraya geleceğiz.

Django URLconfs ile ilgili daha fazla bilgi edinmek istiyorsanız resmi dokümantasyona bakabilirsiniz:

<https://docs.djangoproject.com/en/1.8/topics/http/urls/>

Django views - yaratma zamanı geldi!

Evvelki bölümde yaptığımız hatayı yok edelim :)

`view`, "uygulama mantığının" ifade edildiği yerdir. Daha önce oluşturulan `model` den bilgi alıp `template` 'e iletir. Gelecek bölümde bir template oluşturacağız. View'ler bildiğiniz Python methodlarıdır. Ancak, **Python'a Giriş** bölümünde yazdığımız methodlardan biraz daha karmaşıktır.

View'ler `views.py` doyasına yazılır. Şimdi, `blog/views.py` dosyasına `view` ekleyelim.

blog/views.py

Dosyayı açıp inceleyelim:

```
from django.shortcuts import render

# View'lar buraya yazılacak.
```

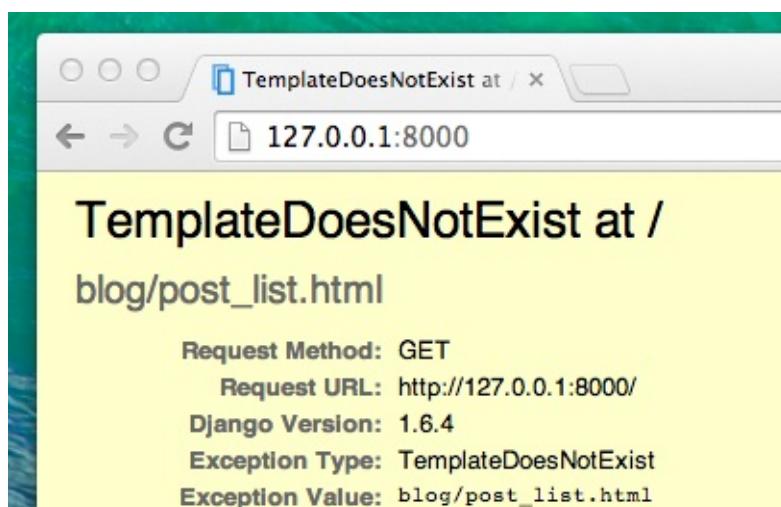
Henüz fazla bir şey görünmüyör. En basitinden `view` şöyleden olabilir.

```
def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

Burada, `request (istek)` i alıp `template blog/post_list.html` ile görüntüleyen `render` methodunu döndüren `post_list` isimli bir method yarattık.

Dosyamızı kaydedelim ve <http://127.0.0.1:8000> e gidip bakalım.

Yine hata! Okuyup anlamaya çalışalım:



Bu hatayı düzeltmek kolay: `TemplateDoesNotExist` (Template bulunamadı). Bu hatayı template oluşturarak gelecek bölümde düzeltelim!

Django view hakkında daha fazla bilgi edinmek için dokümantasyonları okuyun:
<https://docs.djangoproject.com/en/1.8/topics/http/views/>

HTML'ye giriş

Template nedir diye sorabilirsiniz.

Template, farklı bilgileri hep aynı biçimde sunmak için tekrar kullanabileceğimiz bir dosyadır - örneğin, mektup yazmanıza yardımcı olan bir template kullanabilirsiniz çünkü yazacağınız tüm mektuplar farklı mesajlar içerece ve farklı kişilere gönderilse de aynı sayfa düzene sahip olacaktır.

Bir Django template düzeni HTML adını verdığımız bir dilde tanımlanır ([İnternet nasıl çalışır](#) adlı ilk bölümde bahsettiğimiz HTML).

HTML nedir?

HTML Chrome, Firefox veya Safari gibi web tarayıcılar tarafından bir web sayfasını kullanıcıya görüntülemek için yorumlanan basit bir koddur.

HTML "HyperText Markup Language" (HiperMetin İşaretleme Dili) anlamına gelir. **HyperText (HiperMetin)** sayfalar arası bağlantıları destekleyen türden bir metin demektir. **Markup (İşaretleme)**, bir belgeyi alıp onu kodlarla işaretleyerek, nasıl yorumlanacağını (tarayıcıya) söyledik demektir. HTML kodu **etiketler** ile oluşturulur, etiketlerin her biri < ile başlar ve > ile biter. Bu etiketler biçimlendirme **öğelerini** temsil eder.

İlk template'iniz!

Bir template oluşturmak bir template dosyası oluşturmak demektir. Her şey bir dosyadır, değil mi? Bunu muhtemelen zaten fark etmişsinizdir.

Template'lar `blog/templates/blog` dizininde saklanır. Öyleyse blog klasörü altında `templates` adlı bir klasör oluşturalım. Sonra da templates klasörü altında yine `blog` adlı bir klasör oluşturalım:

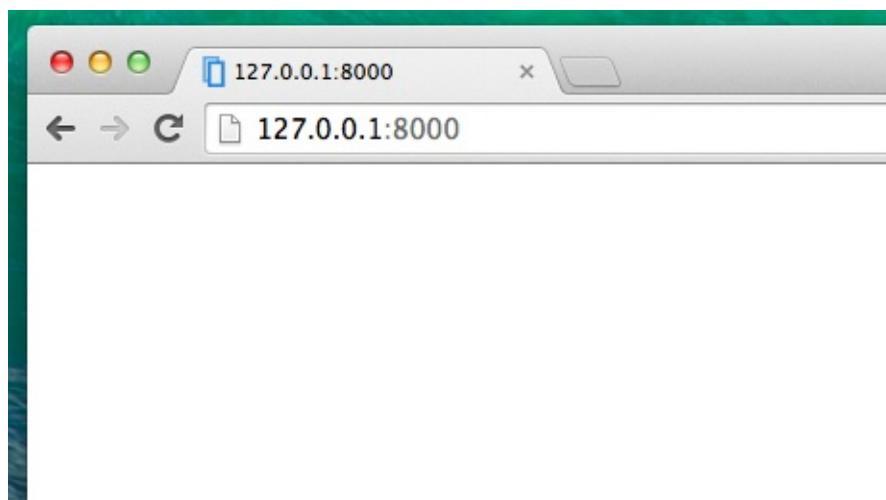
```
blog
└── templates
    └── blog
```

(Neden iki tane `blog` adlı klasöre gerek olduğunu merak etmiş olabilirsin. Daha sonra da anlaşılacığı gibi, sitemiz karmaşıklaşıkça bu şekilde isimlendirme tarzı işimizi oldukça kolaylaştırır.)

Şimdi de `blog/templates/blog` dizini içine `post_list.html` adlı bir dosya oluşturalım (şimdilik içini boş bırakalım).

Web sitemizin nasıl görüneceğine bir bakalım: <http://127.0.0.1:8000/>

Eğer `TemplateDoesNotExist` hatası alırsanız sunucuyu yeniden başlatmayı deneyin. Komut satırına gidip, Ctrl+C (Control ve C tuşlarına eş zamanlı basarak) yaptıktan sonra sunucuyu tekrar başlatmak için `python manage.py runserver` komutunu çalıştırın.

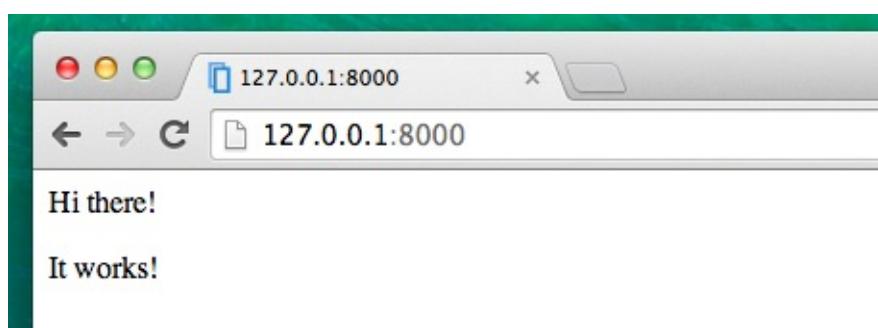


Artık hata kalmadı! Tebrikler :) Ama, web sitemiz aslında boş bir sayfadan başka bir şey yayınlamıyor, çünkü template boş. Bunu düzeltelim.

Template dosyamıza şunları ekleyelim:

```
<html>
  <p>Merhaba!</p>
  <p>Çalışıyor!</p>
</html>
```

Web siteniz şimdi nasıl görünüyor? Öğrenmek için tıklayın: <http://127.0.0.1:8000/>



Çalıştı! Tebrikler :)

- Tüm web sayfaları en temel etiket olan `<html>` etiketi ile başlar ve her zaman `</html>` ile biter. Gördüğünüz gibi, web sitesinin tüm içeriği `<html>` başlangıç etiketi ve `</html>` bitiş etiketinin arasında yer alır
- `<p>` paragraf öğelerini belirten etikettir; her paragrafin bitişinde de `</p>` olacaktır

Head ve body (Başlık ve gövde)

Aynı zamanda tüm HTML sayfaları **head** ve **body** olmak üzere iki öğeye ayrıılır.

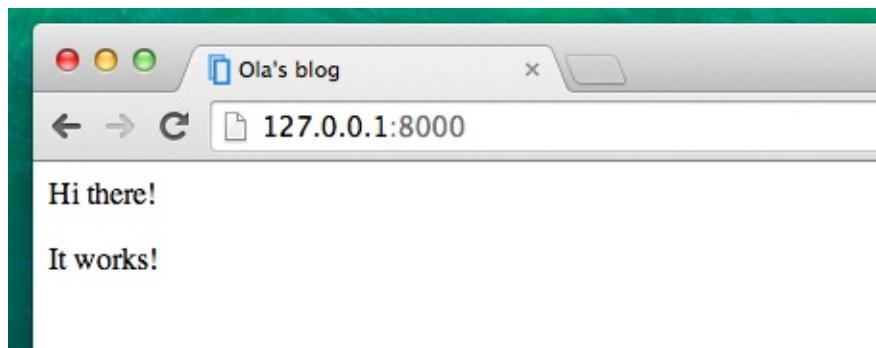
- **head** belge hakkında ekranda görüntülenmeyen bilgiler içeren öğedir.
- **body** ise ekranda gösterilen tüm öğeleri içeren öğedir.

`<head>` ögesini tarayıcıya sayfanın yapılandırmasını anlatmak için, `<body>` ögesini ise sayfada aslında ne olduğunu anlatmak için kullanırız.

Örneğin, web sayfasının (title) başlık elemanını `<head>` 'in içine böyle koyabilirsiniz:

```
<html>
  <head>
    <title>Zeynep'in blogu</title>
  </head>
  <body>
    <p>Merhaba!</p>
    <p>Çalışıyor!</p>
  </body>
</html>
```

Dosyayı kaydedin ve sayfanızı yenileyin.



Tarayıcınızın "Zeynep'in blogu" başlığını nasıl anladığını fark ettiniz mi? `<title>Zeynep'in blogu</title>` kısmını başlık olarak yorumlayarak yazıyı tarayıcının başlık kısmına yerleştirdi. (Bu yazı yer işaretleri gibi yerlerde de kullanılır).

Her açılan etiketin benzer bir *kapatıcı etiket*, / ile başlayan, ile kapatılmalıdır. Ayrıca bu etiketler iç içe yerleştirilebilir (bu da bir etiketi kapatabilmek için, içindeki tüm etiketlerin kapanmış olmasını gerektirir).

Bir şeyleri kutulara yerleştirmek gibi. Büyük bir kutuda `<html></html>` olsun; onun içinde `<body></body>` kutusu olsun, onun da içinde daha küçük kutular olsun: `<p></p>`.

Etiketleri düzgün *kapatma* ve *icice* yerleştirme kurallarına uymak çok önemli. Aksi takdirde tarayıcı belgenizi doğru yorumlayamaz ve gösteremez.

Template özelleştirme

Şimdi artık biraz eğlenip template'inizi özelleştirmeyi deneyebilirsiniz! İşte bunun için faydalı birkaç etiket:

- `<h1>Bir başlık</h1>` - ana başlığınız için
- `<h2>Bir alt başlık</h2>` - bir sonraki seviyedeki bir başlık için
- `<h3>Bir alt alt başlık</h3>` ... ve böyle `<h6>` ya kadar iner
- `metin` metni vurgular
- `metin` metni iyice vurgular
- `
` - alt satira gider (br etiketi içine bir şey konulmaz)
- `bağlantı` bir bağlantı oluşturur
- `ilk maddeikinci madde` - tıpkı bunun gibi bir liste yapar!
- `<div></div>` - sayfanın bir bölümünü tanımlar

Şimdi de tam bir template örneği:

```

<html>
  <head>
    <title>Django Girls blog</title>
  </head>
  <body>
    <div>
      <h1><a href="">Django Girls Blog</a></h1>
    </div>

    <div>
      <p>published: 14.06.2014, 12:14</p>
      <h2><a href="">İlk Blogum</a></h2>
      <p>Çok heyecanlıyım! Bu benim ilk blogum. Ne kadar zevkli bir işmiş bilgisayarlarla uğraşmak. Artık bilgi
sayar başından kalkmam. </p>
    </div>

    <div>
      <p>published: 14.06.2014, 12:20</p>
      <h2><a href="">İkinci gönderim</a></h2>
      <p>Bir varmış bir yokmuş, evvel zaman içinde Ne kadar zevkli bir işmiş bilgisayarlarla uğraşmak. Artık bi
lgisayar başından kalkmam. kalbur saman içinde, develer tellal iken, pireler berber iken; ben annemin beşiğini tıngır
mınğır sallar iken.</p>
    </div>
  </body>
</html>

```

Burada üç tane `div` bölümü oluşturduk.

- İlk `div` ögesi blogumuzun başlığını içeriyor - bir başlık ve bir bağlantından oluşuyor
- Sonraki iki `div` ögesi blog gönderilerimizi içeriyor; bunlarda bir yayın tarihi, tıklanabilir bir `h2` başlığı ve biri tarih
diğeri gönderi metnimiz için olmak üzere, iki tane `p` (paragraf) var.

Bize yaşattığı duygu:



Yaşasın! Şimdiye dek, template tam olarak sadece **aynı bilgiyi** görüntüledi - öncesinde ise template'in **farklı** bilgileri **aynı formatta** görüntülememize izin verdığinden bahsetmiştim.

Gerçekten yapmak istediğimiz ise Django adminde ekli gerçek gönderileri göstermek - ve bir sonraki adımımız da bu.

Birşey daha: dağıtım!

Bunları İnternet'te canlı olarak görmek çok güzel olur, değil mi:

Kodumuzu commit ve push komutları ile Github'a yükleyelim

İlk önce son deployment dan sonra hangi dosyaların değiştiğine bakalım. Bu komutları lokal bilgisayarımızda çalıştırılarım, PythonAnywhere'de değil:

```
$ git status
```

djangogirls dizininde olduğumuzdan emin olalım ve `git` 'e bu dizinde yapılan tüm değişiklikleri dahil etmesini söyleyelim:

```
$ git add -A .
```

Not: `-A` (hepsi için bir kısaltma - İngilizce'de "all" hepsi demek) `git` 'in silinmiş dosyaları tanır (normalde sadece yeni/güncellenmiş dosyaları tanır). Hatırlatma: `.` içinde olduğumuz klasör anlamına gelir (3. Bölüm).

Dosyalarımızı yüklemeden önce `git` 'in hangilerini yükleyeceğine (`git` 'in yükleyeceği dosyalar yeşil gösterilir) bakalım:

```
$ git status
```

Neredeyse bitirdik, şimdi bu değişikliği tarihçesine kaydetmesini söyleyelim. Commit için değişiklikleri açıklayan bir mesaj yazalım. Bu aşamada istediğimizi yazabiliriz, fakat tanımlayıcı yazılar gelecekte neler yapmış olduğumuza hatırlatması açısından faydalı olacaktır.

```
$ git commit -m "Site için HTML dosyasını değiştirdim."
```

Not Tamamlama mesajını çift tırnak içerisinde kullandığımızdan emin olalım.

Bunu tamamladıktan sonra, değişiklikleri Github'a push komutunu kullanarak yükleyelim:

```
$ git push
```

Pull ile yeni kodu PythonAnywhere e alıp web uygulamasını tekrar yükleyelim

- [PythonAnywhere consoles page](#)sayfasını ve **Bash console** u açalım (ya da yeni bir tane açalım). Sonra da çalıştırılarım:

```
$ cd ~/ilk-blogum  
$ source myvenv/bin/activate  
(myvenv)$ git pull  
[...]  
(myvenv)$ python manage.py collectstatic  
[...]
```

Kodumuzun indirilmesini izleyelim. Kodun geldiğini kontrol etmek istersek **Files (dosyalar)** sekmesini açıp PythonAnywhere'de kodumuzu görebiliriz.

- Son olarak, [Web sekmesine](#) gidip uygulamanızın **Yenile** butonuna basın.

Güncelleme hazır olmalı! Devam edelim ve tarayıcıda web sitesini yenileyelim. Değişiklikler görünüyor olmalı :)

Django ORM ve QuerySets (Sorgu Setleri)

Bu bölümde Django'nun veritabanına nasıl bağlandığını ve veriyi nasıl sakladığını öğreneceğiz. Hadi başlayalım!

QuerySet (SorguSeti) Nedir?

QuerySet (SorguSeti), esas olarak, verilen bir modelin nesnelerinin listesidir. QuerySet veritabanından veri okumamıza, veriyi filtrelememize ve sıralamamıza imkan sağlar.

En kolay örnekle öğrenmektir. Hadi deneyelim, olur mu?

Django shell (kabuk)

Yerel konsolumuza açalım (PythonAnywhere'dekini değil) ve şu komutu yazalım:

```
(myenv) ~/djangogirls$ python manage.py shell
```

Etkisi aşağıdaki gibi olmalı:

```
(InteractiveConsole)
>>>
```

Şu an Django'nun etkileşimli konsolundayız. Python istemine benziyor, ama biraz Django büyüsü eklenmiş :) Kuşkusuz burada da Python komutlarının tümünü kullanabiliriz.

Tüm nesneler

Once tüm gönderilerimizi görüntülemeyi deneyelim. Bunu aşağıdaki komut ile yapabiliriz:

```
>>> Post.objects.all()
Traceback (most recent call last):
  File "<console>", line 1, in <module>
NameError: name 'Post' is not defined
```

Aman! Bir hata meydana geldi. Bize bir gönderi olmadığını söylüyor. Doğru - önce gönderiyi almayı unuttuk!

```
>>> from blog.models import Post
```

Post modelini `blog.models` model havuzundan kolayca kodumuza dahil ettik. Şimdi bütün gönderileri tekrar göstermeye deneyelim:

```
>>> Post.objects.all()
[<Post: Gönderi 1>, <Post: Gönderi 2>]
```

Daha önce yarattığımız gönderilerin listesi! Bu gönderileri Django yönetici arayüzü kullanarak yaratmıştık. Şimdi ise Python kullanarak yeni gönderiler yaratmak istiyoruz, bunu nasıl yapabiliz?

Nesne oluşturma

Veritabanına yeni bir gönderi eklemek için:

```
>>> Post.objects.create(yazar=ben, baslik='Harika bir gönderi', yazi='Ne desem bilemedim')
```

Ancak bir eksigmiz var: `ben`. Gönderinin yazar özelliğine `User` modelinden türetilen bir nesneyi parametre olarak vermemiz gerekiyor. Nasıl verebiliriz?

Öncelikle kullanıcı modelini dahil edelim:

```
>>> from django.contrib.auth.models import User
```

Veritabanımızda hangi kullanıcılar var? Şu şekilde görebiliriz:

```
>>> User.objects.all()
[<User: zeynep>]
```

Daha önce yarattığımız ayrıcalıklı kullanıcı! Şimdi veritabanından kullanıcı nesnesi alalım:

```
ben = User.objects.get(username='zeynep')
```

Gördüğünüz gibi, `username` özelliği 'zeynep' olan `User` nesnesini `get` ile aldık. Müthiş! Tabiki, kullanıcı adını kendi kullanıcı adınıza göre ayarlamalısınız.

Gönderimizi artık kaydedebiliriz:

```
>>> Post.objects.create(yazar=ben, baslik='Harika bir gönderi', yazi='Ne desem bilemedim')
```

Yaşasın! Çalışıp çalışmadığını kontrol etmek ister misin?

```
>>> Post.objects.all()
[<Post: Gönderi 1>, <Post: Gönderi 2>, <Post: Harika bir gönderi>]
```

İşte bu kadar, listede bir gönderi daha!

Daha fazla gönderi ekle

Şimdi biraz eğlenenebiliriz ve nasıl çalıştığını görmek için daha fazla gönderi ekleyebiliriz. 2-3 tane daha ekleyin ve bir sonraki kısma devam edin.

Nesneleri filtrelemek

QuerySets in büyük bir parçası nesneleri filtreleyebilme kabiliyetidir. Diyelim ki, Zeynep tarafından yazılmış tüm gönderileri bulmak istiyoruz. `Post.objects.all()` içindeki `all` yerine `filter` kullanacağız. Parantez içine istediğimiz blog gönderilerinin sağlaması gereken şartları belirteceğiz. Örneğimizde, `yazar` `ben`'e eşitti. Django'da bu filtre şöyle yazılır: `yazar=ben`. Şu an kod parçacığımız şöyle görünüyor:

```
>>> Post.objects.filter(yazar=ben)
[<Post: Gönderi 1>, <Post: Gönderi 2>, <Post: Harika bir gönderi>, <Post: Nefis bir gönderi>]
```

Ya da belki `baslik` alanında içinde 'Nefis' kelimesini içeren tüm gönderileri görmek istiyoruz?

```
>>> Post.objects.filter(baslik__contains='Nefis')
[<Post: Nefis bir gönderi>]
```

Not `baslik` ve `contains` arasında iki tane alt çizgi (`_`) var. Django'nun ORM'i bu söz dizimini, özelliği ("baslik") ve operasyon veya filtreyi ("contains") ayırmak için kullanır. Sadece tek alt çizgi kullanırsanız, "FieldError: Cannot resolve keyword `baslik_contains`" hatası alırsınız.

Ayrıca yayınlanmış tüm gönderilerin bir listesini alabiliyoruz. Bunu geçmişte `yayinlanma_tarihi` alanı belirtmiş tüm gönderileri filtreleyerek yapıyoruz:

```
>>> from django.utils import timezone
Post.objects.filter(yayinlanma_tarihi__lte=timezone.now()) []
```

Maalesef, Python konsolundan eklediğimiz gönderi henüz yayınlanmadı. Bunu değiştirebiliriz! İlk olarak yayınlamak istediğimiz gönderinin bir örneğini alalım:

```
>>> post = Post.objects.get(baslik="Harika bir gönderi")
```

Ardından `yayinla` methodu ile gönderiyi yayinallyalım!

```
>>> post.yayinla()
```

Şimdi yayınlanmış gönderileri tekrar almaya çalışalım (3 kez yukarı yön ve ardından `enter` tuşuna basın):

```
>>> Post.objects.filter(yayinlanma_tarihi__lte=timezone.now())
[<Post: Harika bir gönderi>]
```

Nesneleri Sıralama

QuerySets ayrıca nesne listesini sıralamanızı da sağlar. Nesneleri `yaratılma_tarihi` özelliğine göre sıralamayı deneyelim:

```
>>> Post.objects.order_by('yaratılma_tarihi')
[<Post: Gönderi 1>, <Post: Gönderi 2>, <Post: Harika bir gönderi>, <Post: Nefis bir gönderi>]
```

Başına `-` ekleyerek sıralamayı tersine de çevirebiliriz:

```
>>> Post.objects.order_by('-yaratılma_tarihi')
[<Post: Nefis bir gönderi>, <Post: Harika bir gönderi>, <Post: Gönderi 2>, <Post: Gönderi 1>]
```

Sorgu Setlerini Zincirlemek

Sorgu setlerini **zincirleyerek** beraber kullanabilirsiniz:

```
>>> Post.objects.filter(yayinlanma_tarihi__lte=timezone.now()).order_by('yayinlanma_tarihi')
```

Zincirleme gerçekten çok güçlündür ve oldukça karmaşık sorgular yazmanıza imkan sağlar.

Güzel! Şimdi bir sonraki bölüm için hazırlız. Kabuğu kapatmak için, şunu yazalım:

```
>>> exit()
$
```

Template içerisinde dinamik veri

Birkaç parçayı yerine oturttuk: `Post` (gönderi) modelini `models.py` 'de tanımladık, `views.py` 'de `post_list` (gönderi listesi) var ve template ekledik. Ama gönderilerimizi HTML'de görünür kıldık mı? Çünkü bu yapmak istediğimiz şey: içeriği (veritabanında kayıtlı modellerimiz) al ve güzelceme template içerisinde göster, değil mi?

Bu tam olarak `view`'lerin yapmasını beklediğimiz şey: modelleri ve template'leri bağlamak. `post_list` `view`de göstermek istediğimiz modelleri alıp template'e iletmemiz gerekecek. Yani temelde bir `view`de template içinde neyin (hangi modelin) gösterileceğine karar veriyoruz.

Peki, bunu nasıl yapacağız?

`blog/views.py` 'I açacağız. Şu anda `post_list` `view`ü şöyle:

```
from django.shortcuts import render

def post_list(request):
    return render(request, 'blog/post_list.html', {})
```

Kodları farklı dosyalara eklemekten bahsettiğimizi hatırlıyor musunuz? Şimdi `models.py` 'de yazdığımız modeli ekleme zamanı. `from .models import Post` satırını şu şekilde ekleyeceğiz:

```
from django.shortcuts import render
from .models import Post
```

`from` 'dan sonraki nokta, *mevcut dizin* veya *mevcut uygulama* anlamına geliyor. `views.py` ve `models.py` aynı dizinde oldukları için sadece `.` ve dosyanın adı (`.py` olmadan) kullanabiliyoruz. Arkasından modelin adını (`Post`)'u dahil ediyoruz.

Sırada ne var? `Post` modelinden gönderileri almamız için `querySet` dediğimiz bir şeye ihtiyacımız var.

QuerySet (Sorgu Seti)

QuerySet'in nasıl çalıştığı konusunda bir fikriniz oluşmuştur. [Django ORM \(QuerySets\) bölümü](#)nde konuşmuştu.

Şimdi yayınlanmış ve `yayınlanma_tarihi` 'ne göre sıralanmış bir gönderi listesi istiyoruz, değil mi? Bunu QuerySets bölümünde yapmıştık zaten!

```
Post.objects.filter(yayınlanma_tarihi__lte=timezone.now()).order_by('yayınlanma_tarihi')
```

Şimdi bu kodu `blog/views.py` dosyasında `def post_list(request)` fonksiyonuna ekleyelim:

```
from django.shortcuts import render
from django.utils import timezone
from .models import Post

def post_list(request):
    posts = Post.objects.filter(yayınlanma_tarihi__lte=timezone.now()).order_by('yayınlanma_tarihi')
    return render(request, 'blog/post_list.html', {})
```

QuerySet'imiz için bir *değişken* yarattığımıza dikkat edin: `posts`. Bu QuerySet'in ismi. Bundan sonra ondan ismi ile bahsedebiliriz.

Bu kod `timezone.now()` fonksiyonunu kullanıyor, dolayısıyla `timezone` için bir 'import' eklememiz gerekiyor.

Son eksik kalan kısım `posts` QuerySet'ini template'e iletmek (nasıl göstereceğimizi bir sonraki bölümde işleyeceğiz).

render fonksiyonunda halihazırda request diye bir parametremiz var (dolayısıyla Internet üzerinden kullanıcı ile ilgili aldığımız her şey) ve bir de template dosyamız 'blog/post_list.html' var. {} şeklindeki son parametremiz, template içinde kullanılmak üzere bir şeyler ekleyebileceğimiz bir değişken. Bunlara isimler vermemiz gerekiyor ('posts' ismini kullanmaya devam edeceğiz şimdilik :)). Şöyledir olması lazımdır: {'posts': posts} . : 'dan önceki kısmın bir string olduğuna dikkat edin; etrafına tek tırnak koymamız gerekiyor ''.

Nihayetinde blog/views.py şu şekilde gelmiş olmalı:

```
from django.shortcuts import render
from django.utils import timezone
from .models import Post

def post_list(request):
    posts = Post.objects.filter(yayinlanma_tarihi__lte=timezone.now()).order_by('yayinlanma_tarihi')
    return render(request, 'blog/post_list.html', {'posts': posts})
```

İşte bu kadar! Template'e geri gidip QuerySet'leri görünür hale getirme zamanı!

Django'da QuerySet'lerle ilgili daha fazla bilgi istiyorsanız şuraya bakabilirsiniz:

<https://docs.djangoproject.com/en/1.8/ref/models/querysets/>

Django template

Bazı verileri gösterme zamanı! Django bunun için bize faydalı bazı yerleşik **template etiketleri** sunuyor.

Template etiketleri nedir?

Görüyoruz ki aslında, HTML'de Python kodu yazamayız, çünkü tarayıcılar bunu anlamaz. Tarayıcılar yalnızca HTML'den anlar. Biliyoruz ki Python daha dinamik bir dil iken, HTML oldukça statiktir.

Django template etiketleri Python benzeri yapıların HTML'ye aktarılmasını sağlar, böylece dinamik web sitelerini daha kolay ve hızlı oluşturabiliriz!

Gönderi listesi template'ini göster

Bir önceki bölümde, template'e `posts` değişkeni içinde gönderiler listesi verdik. Şimdi, bunu HTML'de göstereceğiz.

Django şablonunda bir değişken yazdırmak için, değişken adını çift kıvrımlı parantez içinde şu şekilde kullanınız:

```
 {{ posts }}
```

Bunu `blog/templates/blog/post_list.html` şablonunda deneyelim. İlkinci `<div>`'den üçüncü `</div>`'e kadar olan her şeyi `{}{ posts }{ }` ile değiştirelim. Ne olduğunu görmek için dosyayı kaydedip sayfayı yenileyelim:



Gördüğümüz sadece bu:

```
[<Post: Gönderi 2>, <Post: Gönderi 1>]
```

Yani Django bunu bir nesneler listesi olarak algılıyor. **Python'a giriş'ten** listelerin nasıl gösterildiğini hatırlıyor musun? Evet, döngüler! Bir Django template ile bunu şöyle yaparsın:

```
{% for post in posts %}
    {{ post }}
{% endfor %}
```

Bunu kendi template'imizle deneyelim.

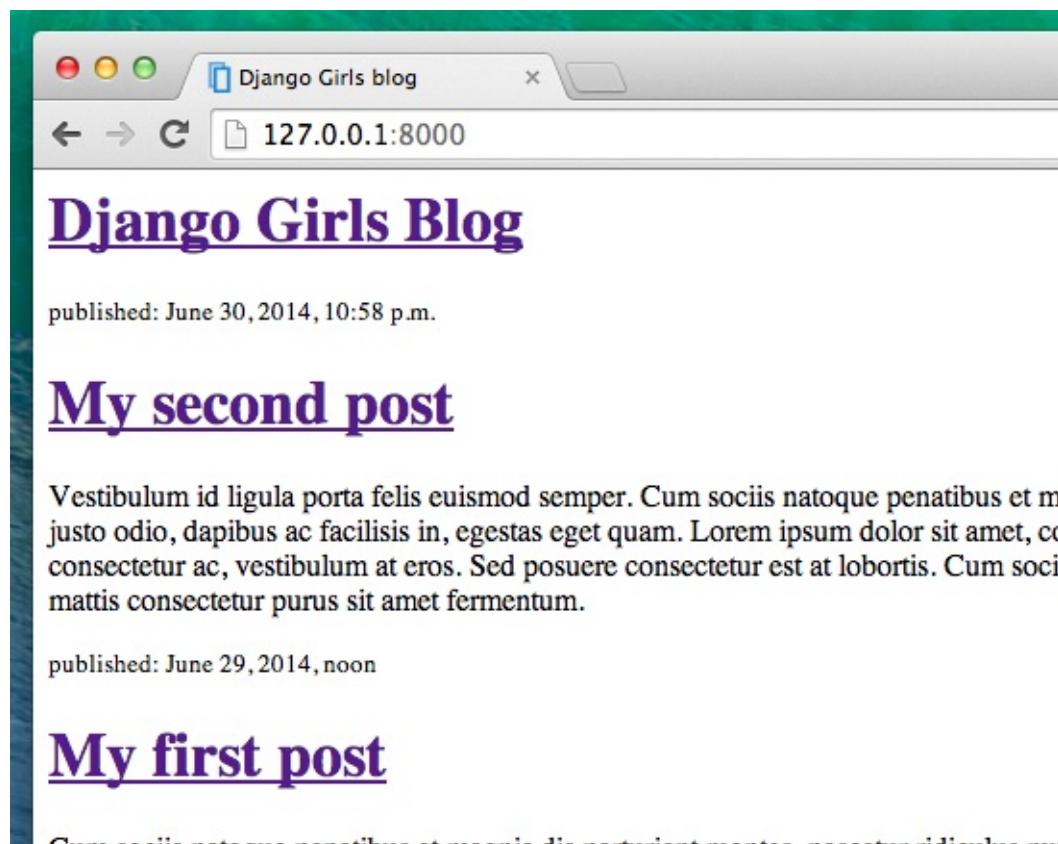


İşe yarıyor! Fakat bunların daha önce **HTML'ye giriş** bölümünde oluşturduğumuz statik gönderiler gibi görünmesini istiyoruz. HTML ve template etiketlerini karıştırabiliriz. `body` şöyle görünecektir:

```
<div>
    <h1><a href="/">Django Girls Blog</a></h1>
</div>

{% for post in posts %}
<div>
    <p>published: {{ post.yayinlanma_tarihi }}</p>
    <h1><a href="{{ post.baslik }}>{{ post.baslik }}</a></h1>
    <p>{{ post.yazi|linebreaks }}</p>
</div>
{% endfor %}

{% for %} ve {% endfor %} arasına koyduğunuz her şey listedeki her nesne için tekrarlanır. Sayfanı yenile:
```



`published: {{ post.yayinlanma_tarihi }}` ya da `published: {{ post.yazi }}` için biraz farklı bir yazım kullandığımızı farkettin mi? Böylece `Post` modelinde tanımlanan alanlardaki verilere ulaşıyoruz. Ayrıca `|linebreaks` (satırsonu), gönderilerin metnini, satır sonlarını paragraflara çeviren bir filtreden geçiriyor.

Bir şey daha

Web sitemizin İnternet'te hâlâ çalıştığını görmek iyi olacak, değil mi? PythonAnywhere yükleyelim yine. Adımları hatırlayalım...

- İlk önce kodumuzu Github'a push komutu ile yükleyelim

```
$ git status  
[...]  
$ git add -A .  
$ git status  
[...]  
$ git commit -m "Veritabanındaki postları görebilmek için template'i değiştirdim."  
[...]  
$ git push
```

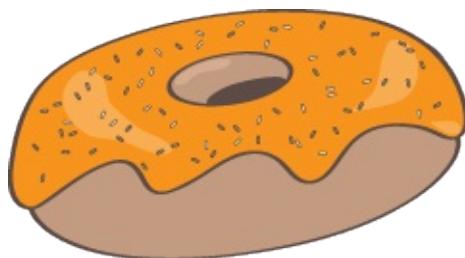
- [PythonAnywhere](#)'e bağlanalım ve **Bash konsolu**'na gidelim (veya yeni bir konsol açalım) ve şunu çalıştıralım:

```
$ cd ilk-blogum  
$ git pull  
[...]
```

- Ve son olarak da [Web tab](#) sekmesine gidip web uygulamamızdaki **Reload**'a basalım. Şimdi güncellememiz yayında olmalı!

Tebrikler! Şimdi devam edelim ve Django admininde yeni bir gönderi eklemeyi deneyelim (yayınlanma_tarihi eklemeyi unutmayalım!), sonrasında gönderinin görünüp görünmediğini görmek için sayfayı yenileyelim.

Şiir gibi çalışıyor, değil mi? Gurur duyabiliriz! Şimdi bilgisayar başından bir süre kalkalım, çünkü bir molayı hak ettik. :)



CSS - güzelleştir!

Blogumuz hala epey çirkin gözüküyor, değil mi? Güzelleştirme zamanı! Bunun için CSS kullanacağız.

CSS Nedir?

Basamaklı Stil Sayfaları (Cascading Style Sheets - CSS) bir websayfasının görünüm ve biçimlendirmelerini tanımlamak için kullanılan bir işaretleme (markup) dilidir (HTML gibi). CSS'i websayfanızın makyajı olarak görebilirsiniz ;).

Fakat tekrar en baştan başlamak istemiyoruz, değil mi? Bir kez daha, programcılar tarafından önceden hazırlanmış ve internette ücretsiz olarak yayınlanmış bir şeyi kullanacağız. Bilirsiniz tekerleği yeniden icat etmek eğlenceli değildir.

Hadi Bootstrap kullanalım!

Bootstrap güzel websayfaları geliştirmek için kullanılan en popüler HTML ve CSS çerçevesidir(framework) :
<http://getbootstrap.com/>

Twitter yazılımcıları tarafından geliştirilmeye başlanmış ve şu anda dünyanın her yerinden gönüllüler tarafından geliştirilmektedir.

Bootstrap kurulumu

Bootstrap kurmak için `.html` dosyanızda `<head>` kısmına şunları eklemeniz gereklı
(`blog/templates/blog/post_list.html`):

```
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
<link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
```

Bu, projemize hiçbir yeni dosya eklemez. Yalnızca internet üzerinde var olan dosyalara işaret eder. Şimdi websitenizi açın ve sayfayı yenileyin. İşte oldu!



Django'da statik dosyalar

Son olarak **statik dosyalar** diye bahsettiğimiz şeylere daha yakından bakalım. Statik dosyalar, CSS ve resimlerindir -- dinamik olmayan, bu nedenle içerikleri istek bağlamından bağımsız ve her kullanıcı için aynı dosyalar.

Django'da statik dosyaları nereye koymalı

Sunucuda `collectstatic` komutunu çalıştığımız zaman gördüğün gibi, Django dahili "admin" uygulaması için statik dosyaların nerede olduğunu biliyor. Şimdi de bizim kendi `blog` uygulamamız için bazı statik dosyalar eklememiz gerekiyor.

Bunu blog uygulamamızın içerisinde `static` isimli bir klasör oluşturarak yapacağız:

```
djangogirls
└── blog
    ├── migrations
    └── static
        └── mysite
```

Django uygulama klasörlerinizin altındaki "static" isimli tüm klasörleri otomatik olarak bulacak ve içindekileri statik dosya olarak kullanabilecektir.

İlk CSS dosyanız!

Şimdi web sayfamıza kendi stilimizi eklemek için bir CSS dosyası oluşturalım. `static` klasörü içinde `css` adlı yeni bir klasör oluşturalım. Şimdi de `css` klasörü içinde `blog.css` adlı yeni bir dosya oluşturalım. Hazır mısınız?

```
djangogirls
└── blog
    └── static
        └── css
            └── blog.css
```

Şimdi CSS yazma zamanı! `blog/static/css/blog.css` dosyasını kod editöründe açın.

Biz burada özelleştirme ve CSS'in detaylarına çok fazla girmeyeceğiz, çünkü gerçekten kolay ve workshop'tan sonra kendinizi bu konuda geliştirebilirsiniz. Web sitelerini CSS'le güzelleştirmek hakkında bilmen gerekenleri öğrenmek için, [Codecademy HTML & CSS kursunu](#) özellikle öneriyoruz.

Ancak az da olsa yapalım. Acaba başlığımızın rengini mi değiştiresek? Bilgisayarlar renkleri anlamak için özel kodlar kullanır. Bu kodlar `#` ile başlar ve arkasından 6 harf (A-F) ve numaralarda (0-9) gelir. Renk kodlarını örneğin burada bulabilirsin: <http://www.colorpicker.com/>. Ayrıca [önceyen tanımlanmış renkleri](#) de kullanabilirsin, `red` (kırmızı) ve `green` (yeşil) gibi.

`blog/static/css/blog.css` dosyanıza şu kodu eklemelisiniz:

```
h1 a {
    color: #FCA205;
}
```

`h1 a` bir CSS Seçicisidir (Selector). Bu demek oluyor ki biz stilimizi, bir `h1` ögesi içerisinde olan tüm `a` öğelerine (örneğin kodumuzun içerisinde `<h1>link</h1>` gibi bir şey olduğunda) uyguluyoruz. Bu durumda, rengi `#FCA205` yani turuncu yapmasını söylüyoruz. Elbette, buraya kendi arzu ettiğin rengi koymabilirsin!

Bir CSS dosyasında, HTML dosyasındaki öğeler için stil belirleriz. Öğeler, ögenin ismi (örn. `a`, `h1`, `body`), sınıf ozniteliği (attribute) ya da `id` ozniteliği ile tanımlanırlar. Sınıf ve id (kimlik), bir elemente senin tarafından verilen isimlerdir. Sınıflar bir öğe grubunu tanımlar, id'ler ise belirli bir öğeye işaret ederler. Örneğin şu aşağıdaki etiket CSS tarafından, `a` etiket adı, `external_link` sınıfı ya da `link_to_wiki_page` id'si kullanılarak tanımlanabilir:

```
<a href="http://en.wikipedia.org/wiki/Django" class="external_link" id="link_to_wiki_page">
```

Daha fazla bilgi için [w3schools'da CSS seçenekleri](#)ni okuyabilirsin.

Sonrasında, ayrıca HTML şablonumuza (template) bir takım CSS eklemeleri yaptığımızı bildirmemiz gerekiyor.

`blog/templates/blog/post_list.html` dosyasını açın ve en başına şu satırı ekleyin:

```
{% load staticfiles %}
```

Burada yaptığımız yalnızca statik dosyaları yüklemek. :) Sonrasında, `<head>` ve `</head>`, tagları arasına, Bootstrap CSS dosyalarına yönelik bağlantılarından sonra (web tarayıcıımız dosyaları yazıldıkları sırasıyla okuduğundan, bizim dosyamızdaki kodlar Bootstrap dosyasının içerisindekileri geçersiz kılabılır), şu satırı ekleyin:

```
<link rel="stylesheet" href="{% static 'css/blog.css' %}">
```

Az evvel şablonumuza (template) CSS dosyamızın nerede olduğunu söylemiş olduk.

Dosyanız şu şekilde gözüküyor olmalı:

```
{% load staticfiles %}

<html>
  <head>
    <title>Django Girls blog</title>
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
    <link rel="stylesheet" href="{% static 'css/blog.css' %}">
  </head>
  <body>
    <div>
      <h1><a href="/">Django Girls Blog</a></h1>
    </div>

    {% for post in posts %}
      <div>
        <p>yayınlanma tarihi: {{ post.yayinlanma_tarihi }}</p>
        <h1><a href="{{ post.baslik }}>{{ post.baslik }}</a></h1>
        <p>{{ post.yazi|linebreaks }}</p>
      </div>
    {% endfor %}
  </body>
</html>
```

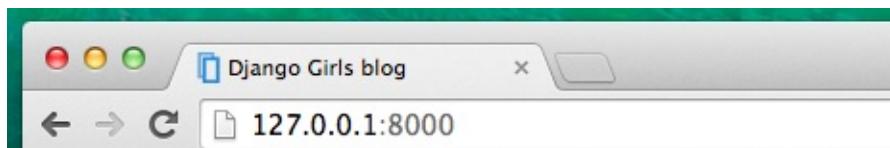
Tamamdır, dosyayı kaydedip sayfayı yenileyebilirsiniz.



Güzel! Şimdi de sitemizi biraz rahatlatıp sol kenar boşluğunu (margin'i) artırsak mı? Hadi deneyelim!

```
body {  
    padding-left: 15px;  
}
```

Bunu CSS dosyanıza ekleyin, dosyayı kaydedin ve nasıl çalıştığını görelim!



Django Girls Blog

published: June 30, 2014, 10:58 p.m.

My second post

Vestibulum id ligula porta felis euismod semper. Cum sociis natoque p

Belki de başlığımızın yazı tipini özelleştirebiliriz? Aşağıdaki satırı `blog/templates/blog/post_list.html` dosyasının içinde `<head>` bölümüne yapıştırın:

```
<link href="http://fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext" rel="stylesheet" type="text/css">
```

Bu satır *Lobster* adlı bir fontu Google Fonts (<https://www.google.com/fonts>) sitesinden sayfamıza aktarır.

Şimdi `blog/static/css/blog.css` dosyamızdaki `h1 a` deklarasyon bloğunun içine (`{ } ve }` kodları arasında) `font-family: 'Lobster';` satırını ekleyip sayfayı yenileyin:

```
h1 a {  
    color: #FCA205;  
    font-family: 'Lobster';  
}
```



Harika!

Yukarıda bahsettiğimiz üzere, CSS'te class (sınıf) diye bir kavram var. Class'lar, temel olarak HTML kodunuzun bir parçasına isim vermenize yarar ve yalnızca o parçanın stilini değiştirirken diğer parçaların etkilenmemesini sağlar. İki div'iniz var diyelim; fakat çok farklı şeyler yapıyorsa (örneğin biri başlık diğerinin metni) ve bu nedenle de aynı şekilde gözükmelerini istemiyorsanız, sınıflar müthiş yararlıdır.

Devam edelim ve HTML kodumuzun bir kısmına isim verelim. Başlığı içeren `div`'e `page-header` isimli bir class ekleyelim:

```
<div class="page-header">
    <h1><a href="/">Django Girls Blog</a></h1>
</div>
```

Şimdi de gönderi metnini içeren `div`'e `post` isimli bir sınıf ekleyelim.

```
<div class="post">
    <p>yayınlanma tarihi: {{ post.yayinlanma_tarihi }}</p>
    <h1><a href="">{{ post.baslik }}</a></h1>
    <p>{{ post.yazi|linebreaks }}</p>
</div>
```

Şimdi farklı seçimlere (selectors) bildirim (deklarasyon) blokları ekleyeceğiz. `.` ile başlayan seçimciler sınıflara işaret eder. Web'de, aşağıdaki kodu anlamانıza yardımcı olacak pek çok güzel CSS öğreticisi ve açıklama mevcut. Simdilik sadece bu kodu kopyalayıp `blog/static/css/blog.css` dosyamıza yapıştıralım:

```
.page-header {
    background-color: #ff9400;
    margin-top: 0;
    padding: 20px 20px 20px 40px;
}

.page-header h1, .page-header h1 a, .page-header h1 a:visited, .page-header h1 a:active {
    color: #ffffff;
    font-size: 36pt;
    text-decoration: none;
}

.content {
    margin-left: 40px;
}

h1, h2, h3, h4 {
    font-family: 'Lobster', cursive;
}

.date {
    float: right;
    color: #828282;
}

.save {
    float: right;
}

.post-form textarea, .post-form input {
    width: 100%;
}

.top-menu, .top-menu:hover, .top-menu:visited {
    color: #ffffff;
    float: right;
    font-size: 26pt;
    margin-right: 20px;
}

.post {
    margin-bottom: 70px;
}

.post h1 a, .post h1 a:visited {
    color: #000000;
}
```

Sonrasında blog postlarımızı gösteren HTML kodunun etrafını class deklarasyonları ile saralım. Aşağıdaki kodları değiştirin:

```
{% for post in posts %}
<div class="post">
    <p>yayınlanma tarihi: {{ post.yayinlanma_tarihi }}</p>
    <h1><a href="">{{ post.baslik }}</a></h1>
    <p>{{ post.yazi|linebreaks }}</p>
</div>
{% endfor %}
```

bununla değiştirelim:

```
<div class="content container">
    <div class="row">
        <div class="col-md-8">
            {% for post in posts %}
                <div class="post">
                    <div class="date">
                        {{ post.yayinlanma_tarihi }}
                    </div>
                    <h1><a href="">{{ post.baslik }}</a></h1>
                    <p>{{ post.yazi|linebreaks }}</p>
                </div>
            {% endfor %}
        </div>
    </div>
```

Bu dosyaları kaydedin ve web sayfanızı yenileyin.



Heyo! Harika görünüyor, değil mi? Yapıştırdığımız bu kodları anlamak pek de zor değil. Büyük bir kısmını sırf okuyarak anlayabilirsiniz.

CSS ile biraz oynamaktan korkmayın ve bazı şeyleri değiştirmeyi deneyin. Bir şeyi bozarsanız tasalanmayın, yaptığınızı her zaman geri alabilirsiniz!

Her durumda atölye sonrası ödevi olarak ücretsiz [Codeacademy HTML & CSS Kursu](#)'nu, websayfanızı CSS ile güzelleştirmeyi öğrenmek için almanızı tavsiye ediyoruz.

Sonraki bölüm için hazır mısınız? :)

Template genişletmek

Django'nun size sunduğu başka bir güzellik de **template genişletmektir**. O da ne demek? Şu demek, HTML dosyanızın bazı bölgümlerini birden fazla sayfanızda kullanabilirsiniz.

Böylece aynı bilgi/yerleştirmeyi kullanmak istediğinizde her dosyada kendinizi tekrar etmenize gerek kalmaz. Ve bir değişiklik yapmak istediğinizde bunu bütün template dosyalarında yapmanız gereklidir!

Temel template oluşturun

Temel template web sitenizin bütün sayfalarında genişletebileceğiniz en temel template'inizdir.

Şimdi `blog/templates/blog/` klasörü içinde `base.html` adlı bir dosya oluşturalım:

```
blog
└── templates
    └── blog
        └── base.html
        └── post_list.html
```

Sonra bunu açalım ve `post_list.html` dosyasındaki her şeyi aşağıdaki gibi bu `base.html` 'ye kopyalayalım:

```
% load staticfiles %
<html>
  <head>
    <title>Django Girls blog</title>
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
    <link href='//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext' rel='stylesheet' type='text/css'>
  </head>
  <body>
    <div class="page-header">
      <h1><a href="/">Django Girls Blog</a></h1>
    </div>

    <div class="content container">
      <div class="row">
        <div class="col-md-8">
          {% for post in posts %}
            <div class="post">
              <div class="date">
                {{ post.yayinlanma_tarihi }}
              </div>
              <h1><a href="{{ post.baslik }}>{{ post.baslik }}</a></h1>
              <p>{{ post.yazi|linebreaks }}</p>
            </div>
          {% endfor %}
        </div>
      </div>
    </body>
  </html>
```

Sonra, `base.html` dosyasındaki `<body>` 'nizi (`<body>` ve `</body>` arasında kalan her şeyi) şununla değiştirin:

```
<body>
    <div class="page-header">
        <h1><a href="/">Django Girls Blog</a></h1>
    </div>
    <div class="content container">
        <div class="row">
            <div class="col-md-8">
                {% block content %}
                {% endblock %}
            </div>
        </div>
    </div>
</body>
```

Aslında sadece `{% for post in posts %}{% endfor %}` arasındaki her şeyi şununla değiştirmiştir olduk:

```
{% block content %}
{% endblock %}
```

Peki bu ne anlama geliyor? Az önce bir template etiketi olan `block`'u kullanarak bir blok oluşturduk. Diğer template'leri bu bloğun içine HTML ekleyerek `base.html` 'yi genişletebilirsiniz. Bunun nasıl yapıldığını da hemen göstereceğiz.

Şimdi bunu kaydedin ve tekrar `blog/templates/blog/post_list.html` dosyanızı açın. Body içinde kalanlar hariç her şeyi silin. Ve ayrıca `<div class="page-header"></div>` bölümünü de silin. Dosyanız şöyle görünecektir:

```
{% for post in posts %}
    <div class="post">
        <div class="date">
            {{ post.yayinlanma_tarihi }}
        </div>
        <h1><a href="">{{ post.baslik }}</a></h1>
        <p>{{ post.yazi|linebreaks }}</p>
    </div>
{% endfor %}
```

Ve şimdi şu satırı sayfanın başına ekleyin:

```
{% extends 'blog/base.html' %}
```

Bu şu anlama geliyor: `post_list.html` dosyasında `base.html` template'i genişletiyoruz. Sadece bir şey kaldı: Her şeyi (en son eklediğimiz satır hariç) `{% block content %}` ve `{% endblock content %}` arasına koyun. Şunun gibi:

```
{% extends 'blog/base.html' %}

{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                {{ post.yayinlanma_tarihi }}
            </div>
            <h1><a href="">{{ post.baslik }}</a></h1>
            <p>{{ post.yazi|linebreaks }}</p>
        </div>
    {% endfor %}
{% endblock content %}
```

İşte bu! Sitenizin hala düzgün çalışıp çalışmadığını kontrol edin :)

`blog/base.html` dosyası olmadığını söyleyen bir `TemplateDoesNotExist` hatası alıyorsanız, ve `runserver` konsolda çalışmaya devam ediyorsa Ctrl ve C butonlarına aynı anda basarak durdurmayı çalışın. Ve `python manage.py runserver` komutu ile yeniden başlatmayı deneyin.

Uygulamayı genişlet

Websitemizi oluşturmak için gerekli adımların hepsini tamamladık: bir modelin, url'nin, view'ün ve template'in nasıl yazılacağını biliyoruz. Websitemizi nasıl güzelleştirebiliriz onu da biliyoruz.

Pratik zamanı!

Blog'umuzda lazım olan ilk şey bir gönderiyi göstermek için bir sayfa, değil mi?

Halihazırda bir `Post` modelimiz var, dolayısıyla `models.py` dosyasına bir şey eklememize gerek yok.

Bir gönderinin detayı için bir şablon linki oluşturun

`blog/templates/blog/post_list.html` dosyasına bir link (bağlantı) ekleyerek başlayacağız. Şu ana kadar yaptıklarımızın şöyle gözükmeli olması lazım:

```
{% extends 'blog/base.html' %}

{% block content %}
    {% for post in posts %}
        <div class="post">
            <div class="date">
                {{ post.yayinlanma_tarihi }}
            </div>
            <h1><a href="">{{ post.baslik }}</a></h1>
            <p>{{ post.yazi|linebreaks }}</p>
        </div>
    {% endfor %}
{% endblock content %}
```

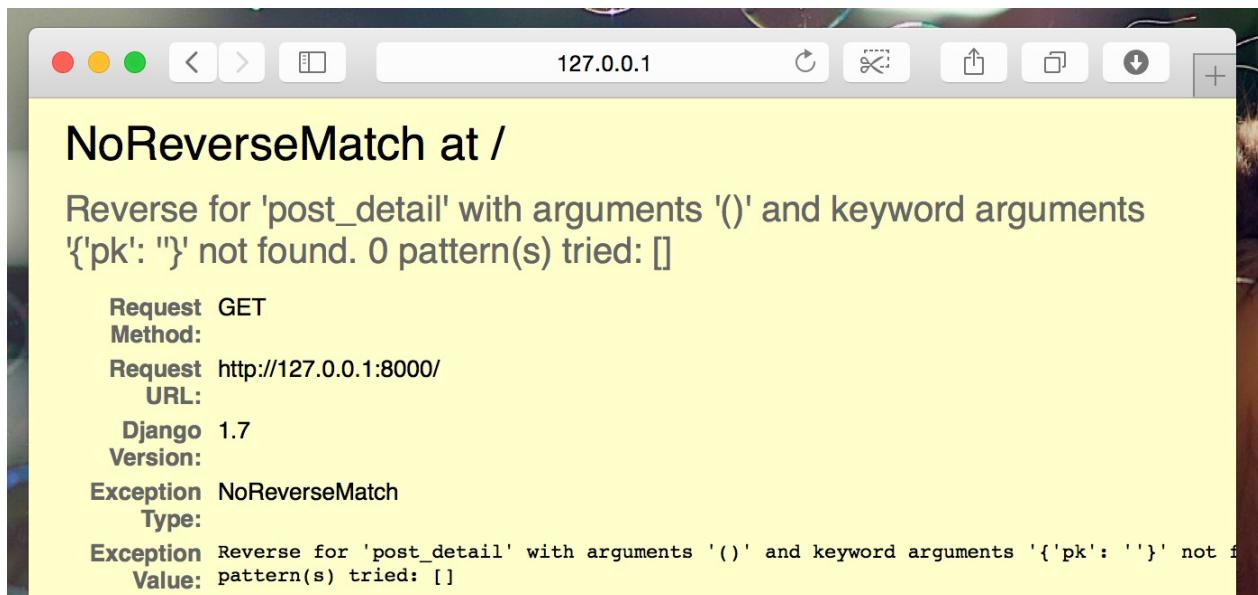
Gönderi listesindeki bir gönderinin başlığından bir gönderinin detay sayfasına bir link (bağlantı) olsun istiyoruz. `<h1>{{ post.baslik }}</h1>` 'i gönderinin detay sayfasına link verecek şekilde değiştirelim:

```
<h1><a href="{% url 'post_detail' pk=post.pk %}">{{ post.baslik }}</a></h1>
```

Gizemli `{% url 'post_detail' pk=post.pk %}` satırını anlatma zamanı. Şüphelendığınız üzere, `{% %}` notasyonu, Django template tags (şablon etiketleri) kullandığımız manasına geliyor. Bu sefer bizim için URL oluşturacak bir template etiketi kullanacağız!

`blog.views.post_detail`, oluşturmak istediğimiz `post_detail` view'una bir yol. Lütfen dikkat: `blog` uygulamamızın adı (`blog` dizini); `views`, `views.py` dosyasının adından geliyor ve son kısım - `post_detail` - `view`'ün adından geliyor.

Şimdi <http://127.0.0.1:8000/>'ye gittiğimizde bir hata alacağız (beklediğimiz bir şey, çünkü URL'miz veya `post_detail` için bir `view`'ümüz yok). Şöyleden görünenecektir:



Bir gönderinin detayı için URL oluşturun

`post_detail` view'ümüz için `urls.py` 'un içinde bir URL oluşturalım!

İlk gönderimizin detayının şu URL'de gösterilmesini istiyoruz: <http://127.0.0.1:8000/post/1/>

`blog/urls.py` dosyasında `post_detail` adında bir Django `view`'una işaret eden bir URL yapalım. Bu `<1>view` bir gönderinin tümünü gösterecek. `blog/urls.py` dosyasına `url(r'^post/(?P<pk>[0-9]+)/$', views.post_detail, name='post_detail'),` satırını ekleyin. Dosyanın şu hale gelmiş olması gerekiyor:

```
from django.conf.urls import include, url
from . import views

urlpatterns = [
    url(r'^$', views.post_list, name='post_list'),
    url(r'^post/(?P<pk>[0-9]+)/$', views.post_detail, name='post_detail'),
]
```

Şu kısım `^post/(?P<pk>[0-9]+)/$` korkutucu gözükmüyor, ama endişelenmeyin, açıklayacağız:

- Gene `^` ile başlıyor - "başlangıç".
- `post/` sadece URL'nin başlangıçtan sonra `post` ve `/`. ifadelerinin geçmesi gerektiği anlamına geliyor. Simdilik iyi gidiyor.
- `(?P<pk>[0-9]+)` - bu kısım biraz daha karışık. Buranın anlamı şu: Django bu alana yerleştirdiğimiz her şeyi alacak ve onu `pk` adında bir değişken olarak view'e aktaracak. `[0-9]` bize eşleşenlerin sadece rakam (yani harf olamaz) olabileceğini söylüyor (0 ile 9 arasındaki her şey). `+` en az bir veya daha fazla rakam olması gerektiğini ifade ediyor. Yani `http://127.0.0.1:8000/post//` eşleşmez ama `http://127.0.0.1:8000/post/1234567890/` eşleşir!
- `/` - gene bir '/e ihtiyacımız var
- `$` - "son"!

Bu şu demek, eğer tarayıcınıza `http://127.0.0.1:8000/post/5/` yazarsanız, Django `post_detail` adında bir `view` aradığınızı anlar ve `pk` eşittir `5` bilgisini `view`'e aktarır.

`pk`, `primary key` 'in (tekil anahtarın) kısaltılmış hali. Bu isim sıkılıkla Django projelerinde kullanılır. Değişkeninize istediğiniz ismi verebilirsiniz (hatırlayın: küçük harfler ve boşluk yerine `_`!). Örneğin `(?P<pk>[0-9]+)` yerine `post_id` değişkenini kullanabilirdik. O zaman şöyle olurdu: `(?P<post_id>[0-9]+)`.

Tamam, `blog/urls.py` dosyasına yeni bir URL kalıbı ekledik! Sayfayı tazeleyelim: <http://127.0.0.1:8000/> Bam! Yeni bir hata daha! Beklenildiği üzere!

```

    return _bootstrap._gcd_import(name[level:], package, level)
File "<frozen importlib._bootstrap>", line 2231, in _gcd_import
File "<frozen importlib._bootstrap>", line 2214, in _find_and_load
File "<frozen importlib._bootstrap>", line 2203, in _find_and_load_unlocked
File "<frozen importlib._bootstrap>", line 1200, in _load_unlocked
File "<frozen importlib._bootstrap>", line 1129, in _exec
File "<frozen importlib._bootstrap>", line 1448, in _exec_module
File "<frozen importlib._bootstrap>", line 321, in _call_with_frames_removed
File "/home/hel/code/djangogirls/workthrough/blog/urls.py", line 6, in <module>
    url(r'^post/(?P<pk>[0-9]+)/$', views.post_detail, name='post_detail'),
AttributeError: 'module' object has no attribute 'post_detail'

```

Bir sonraki adımları ne olduğunu hatırlıyor musunuz? Tabii ki: view'ü eklemek!

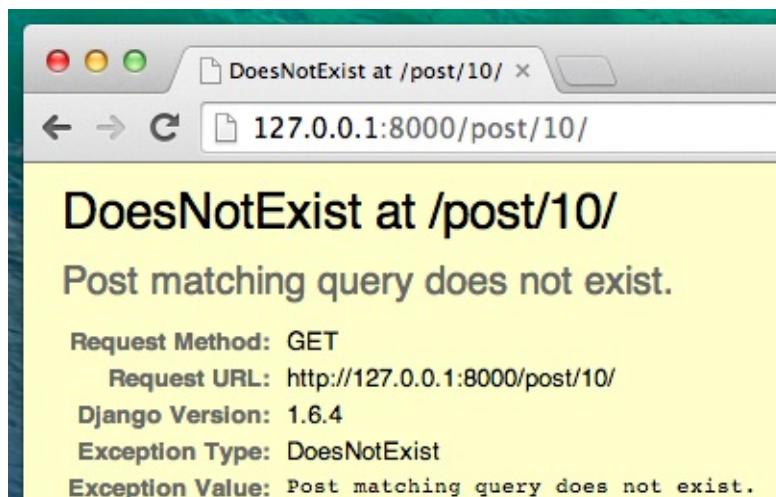
Gönderi detayı için bir view ekleyin

Bu sefer view'ümüze `pk` adında bir parametre ekleyeceğiz. `view`'ümüzin onu yakalaması gerekiyor, değil mi? Fonksiyonumuzu `def post_detail(request, pk):` olarak tanımlayacağız. Dikkat edin, url'lerde kullandığımız ismin birebir aynısını kullanmamız gerekiyor (`pk`). Bu değişkeni kullanmamak yanlışdır ve hataya sebep olacaktır!

Şimdi sadece ve sadece bir blog gönderisini almak istiyoruz. Bunu yapmak için querysets'i şu şekilde kullanabiliriz:

```
Post.objects.get(pk=pk)
```

Ama bu kodun bir problemi var. Eğer gelen `primary key` (`pk` - tekil anahtar) ile bir `Post` (gönderi) yoksa, çok çirkin bir hatamız olacak!



Bunu istemiyoruz! Ama tabii Django'da bunu ele alan bir şey var: `get_object_or_404`. Eğer verilen `pk` ile bir `Post` bulunamazsa, çok daha güzel bir sayfa gösterilecek (`Sayfa bulunamadı 404 sayfası`).



İyi haber şu, kendi `sayfa bulunamadı` sayfasını yapabilir ve istediğiniz kadar güzelleştirebilirsiniz. Ama şu anda çok önemli değil, o yüzden bu kısmı atlayacağız.

Evet, `views.py` dosyamıza bir `view` ekleme zamanı!

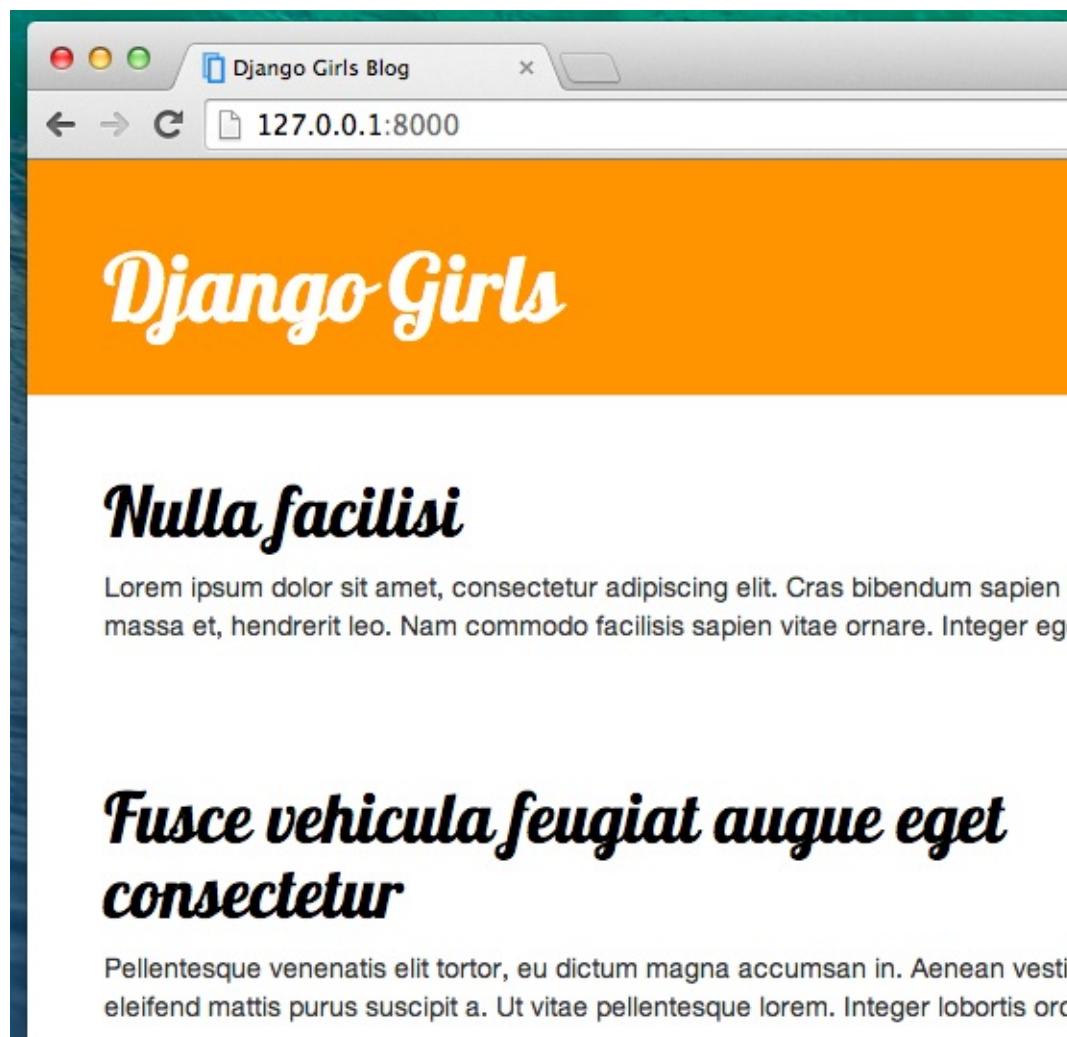
`blog/views.py` dosyasını açıp aşağıdaki kodu ekleyelim:

```
from django.shortcuts import render, get_object_or_404
```

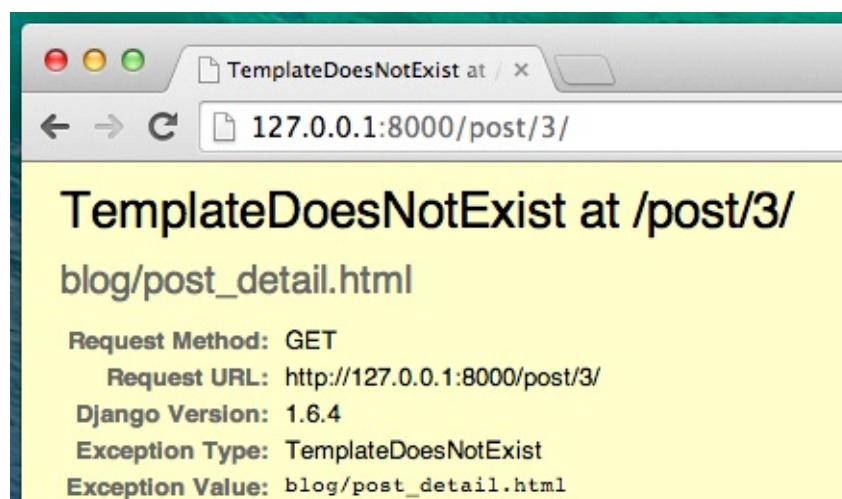
Bunu diğer `from` satırlarının yakınına eklememiz gerekiyor. Dosyanın sonuna `view`'ümüzü ekleyeceğiz:

```
def post_detail(request, pk):
    post = get_object_or_404(Post, pk=pk)
    return render(request, 'blog/post_detail.html', {'post': post})
```

Evet. <http://127.0.0.1:8000/> sayfasını tazeleme zamanı



Çalıştı! Fakat blog gönderisi başlığındaki bir bağlantıya tıkladığınızda ne oluyor?



Of hayır! Başka bir hata! Ama onu nasıl halledeceğimizi biliyoruz, di mi? Bir template eklememiz gerekiyor!

Gönderi detayı için bir template oluşturun

blog/templates/blog dizininde `post_detail.html` adında bir dosya oluşturacağız.

Şöyleden görünmeli:

```
{% extends 'blog/base.html' %}

{% block content %}
<div class="post">
    {% if post.yayinlanma_tarihi %}
        <div class="date">
            {{ post.yayinlanma_tarihi }}
        </div>
    {% endif %}
    <h1>{{ post.baslik }}</h1>
    <p>{{ post.yazi|linebreaks }}</p>
</div>
{% endblock %}
```

Bir kere daha `base.html` dosyasını genişleteceğiz. `content` bloğunda bir gönderinin varsa yayınlama tarihini, başlığını ve metnini göstermek istiyoruz. Ama daha önemli şeyleri konuşmalıyız, değil mi?

`{% if ... %} ... {% endif %}` bir şeyi kontrol etmek istediğimizde kullanabileceğimiz bir template etiketidir (**Python'a giriş** bölümünden `<1>if ... else ..</code>` 'i hatırladınız mı?). Bu senaryoda gönderinin `yayinlanma_tarihi` 'nin boş olup olmadığına bakmak istiyoruz.

Tamam, sayfamızı tazeleyip `sayfa bulunamadı` hatasının gidip gitmediğine bakabiliriz.



Heyo! Çalışıyor!

Bir şey daha: deployment (yayına alma) zamanı!

Sitenizin hala PythonAnywhere'de çalışıp çalışmadığını bakmakta fayda var, değil mi? Yeniden taşımayı deneyelim.

```
$ git status
$ git add -A .
$ git status
$ git commit -m "Detaylı blog gönderileri için CSS'e ilaveten view ve template eklendi."
$ git push
```

- Sonra bir [PythonAnywhere Bash konsol](#) una gidip:

```
$ cd ilk-blogum  
$ source myvenv/bin/activate  
(myvenv)$ git pull  
[...]  
(myvenv)$ python manage.py collectstatic  
[...]
```

- Nihayet, [Web tab](#)ına gidip **Reload** edelim.

O kadar! Tebrikler :)

Django Forms

Günlüğümüzde son olarak yapmak istediğimiz şey, günlük yazılarını eklemek ve düzenlemek için güzel bir yapı oluşturmak. Django'nun `admin` arayüzü çok havalı, ama özelleştirilmesi ve güzelleştirilmesi oldukça zor. Bunu `forms` (formlar) kullanarak kendi arayüzümüz üzerinde mutlak bir güce sahip olacağımız - neredeyse hayal ettiğimiz her şeyi yapabiliriz!

Django formlarının güzel yanı, hem sıfırdan bir form tanımlayabilmemiz hem de sonuçları modele kaydedecek bir `ModelForm` oluşturabilmemizdir.

Tam olarak yapmak istediğimiz şey: `Post` modelimiz için bir form oluşturmak.

Django'nun diğer önemli parçaları gibi, formların da kendi dosyası var: `forms.py`.

`blog` dizinimizde bu isimde bir dosya oluşturmalıyız.

```
blog
└── forms.py
```

Tamam, hadi dosyayı açalım ve aşağıdaki kodu yazalım:

```
from django import forms

from .models import Post

class PostForm(forms.ModelForm):

    class Meta:
        model = Post
        fields = ('baslik', 'yazi',)
```

Önce Django formları (`from django import forms`) ve tabii ki `Post` modelimizi içe aktarmalıyız (`from .models import Post`).

`PostForm`, tahmin etmiş olabileceğiniz gibi, formumuzun ismi. Django'ya bu formun bir `ModelForm` olduğunu belirtmeliyiz. Bunu `forms.ModelForm` sayesinde Django bizim için yapacaktır.

Sırada Django'ya bu formu (`model = Post`) oluşturmak için hangi modelin kullanılması gerektiğini anladığımız `class Meta` var).

Son olarak, formumuzda hangi alan(lar)ın bulunması gerektiğini söyleyebiliriz. Bu senaryoda sadece `baslik` ve `yazi` alanlarının gösterilmesini istiyoruz - `yazar` şu anda giriş yapmış olması gereken kişidir (yani siz!) ve biz ne zaman yeni bir yazı oluşturursak `yaratılma_tarihi` otomatik olarak (örn. kod içinde) ayarlanmalıdır, değil mi?

Ve hepsi bu kadar! Şimdi tek yapmamız gereken formu bir `view` içinde kullanıp, template (şablon) içinde göstermek.

Bir kez daha: sayfaya bir bağlantı, bir URL, bir view ve bir template üreteceğiz.

Formun bulunduğu sayfaya bağlantı oluşturma

Şimdi `blog/templates/blog/base.html` şablonunu açma zamanı. Öncelikle `page-header` adlı `div` ögesinin içine bir bağlantı ekleyeceğiz:

```
<a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
```

Yeni `view`'i `post_new` olarak isimlendirdik.

Yukarıdaki satırı ekledikten sonra html dosyanız böyle görünmeli:

```
{% load staticfiles %}

<html>
  <head>
    <title>Django Girls blog</title>
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap.min.css">
    <link rel="stylesheet" href="//maxcdn.bootstrapcdn.com/bootstrap/3.2.0/css/bootstrap-theme.min.css">
    <link href='//fonts.googleapis.com/css?family=Lobster&subset=latin,latin-ext' rel='stylesheet' type='text/css'>
  >
    <link rel="stylesheet" href="{% static 'css/blog.css' %}">
  </head>
  <body>
    <div class="page-header">
      <a href="{% url 'post_new' %}" class="top-menu"><span class="glyphicon glyphicon-plus"></span></a>
      <h1><a href="/">Django Girls Blog</a></h1>
    </div>
    <div class="content container">
      <div class="row">
        <div class="col-md-8">
          {% block content %}
          {% endblock %}
        </div>
      </div>
    </div>
  </body>
</html>
```

Dokümanı kaydedip <http://127.0.0.1:8000> sayfasını yeniledikten sonra, siz de tanıdık `NoReverseMatch` hatasını görüyorsunuz, değil mi?

URL

`blog/urls.py` dosyasını açalım ve yeni bir satır ekleyelim:

```
url(r'^post/new/$', views.post_new, name='post_new'),
```

Ve kodun son hali şu şekilde görünecektir:

```
from django.conf.urls import include, url
from . import views

urlpatterns = [
    url(r'^$', views.post_list, name='post_list'),
    url(r'^post/(?P<pk>[0-9]+)/$', views.post_detail, name='post_detail'),
    url(r'^post/new/$', views.post_new, name='post_new'),
]
```

Sayfayı yeniledikten sonra, `AttributeError` şeklinde bir hata görürüz, bunun sebebi de henüz `post_new` view'ını (görünümünü) kodlamamış olmamız. Şimdi bu dosyayı da ekleyelim.

post_new view

Şimdi `blog/views.py` dosyasını açıp aşağıdaki satırları diğer `from` satırlarının olduğu yere ekleyelim:

```
from .forms import PostForm
```

ve bizim `view`'ımız:

```
def post_new(request):
    form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

Yeni bir `Post` formu oluşturmak için `PostForm()` fonksiyonunu çağırın ve template'ye iletmek gereklidir. Bu view'a geri döneceğiz, fakat öncesinde form için hızlıca bir şablon oluşturalım.

Template

Öncelikle `blog/templates/blog` dizininde `post_edit.html` isimli bir dosya oluşturmalıyız. Bir formu çalışır hale getirmek için birkaç şeye ihtiyacımız var:

- form'u görüntülemeliyiz. Bunu basitçe bu şekilde yapabiliriz: `{{ form.as_p }}`.
- yukarıdaki örnek satır HTML form etiketi içine alınmalı: `<form method="POST">...</form>`
- bir `Kaydet` butonuna ihtiyacımız var. Bunu bir HTML butonu ile yapıyoruz: `<button type="submit">Kaydet</button>`
- son olarak, hemen `<form ...>` etiketinden sonra `{% csrf_token %}` satırını eklememiz gereklidir. Formlarımızın güvenliğini sağladığı için bu çok önemlidir! Bunu koymayı unutup formu kaydedersek Django hata verecektir:

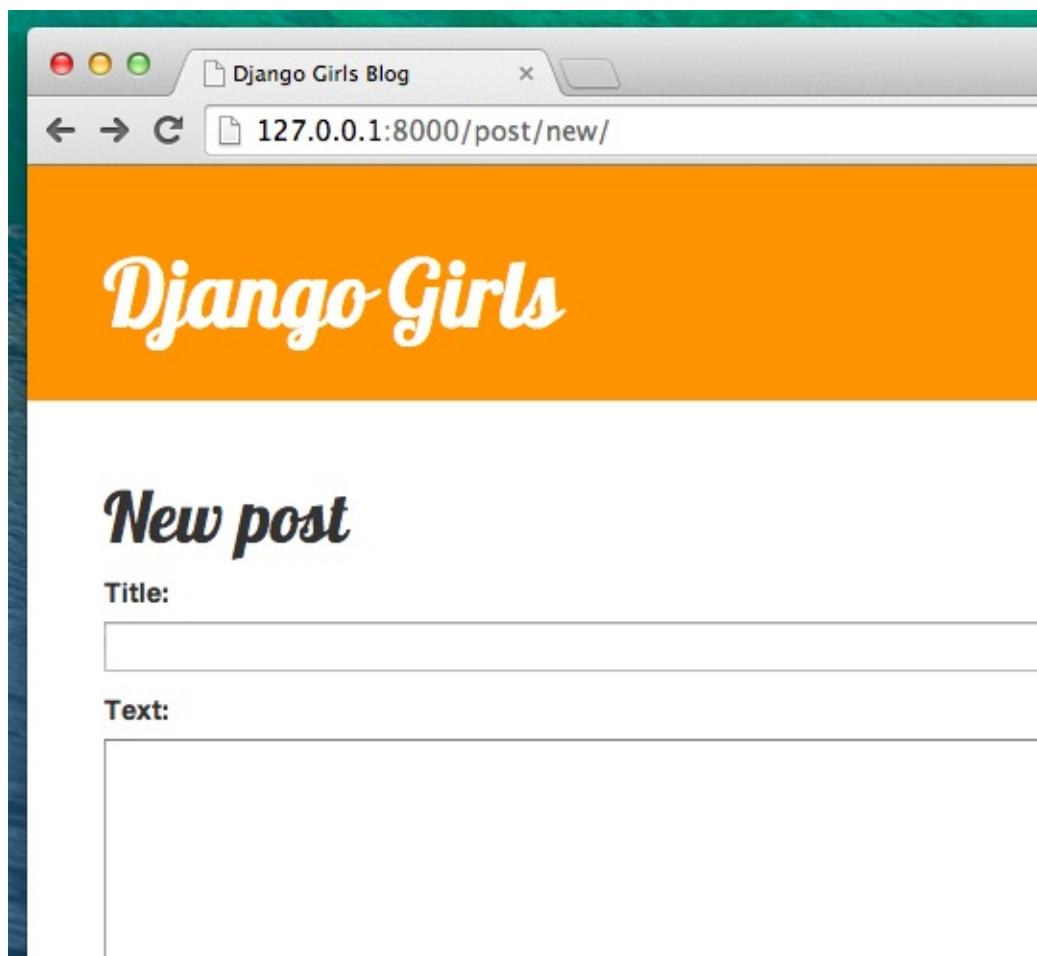


Peki, şimdi de `post_edit.html` içindeki HTML kodunun nasıl görünmesi gerekiğine bakalım:

```
{% extends 'blog/base.html' %}

{% block content %}
    <h1>Yeni gönderi</h1>
    <form method="POST" class="post-form">{% csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Kaydet</button>
    </form>
{% endblock %}
```

Yenileme zamanı! Hey! Formun görüntülendi!



Ama, bir dakika! `baslik` ve `yazi` alanlarına bir şey yazıp kaydettiğimizde ne olacak?

Hiçbir şey! Yine aynı sayfaya döndük ve bütün yazdıklarımız kayboldu... ve yeni bir gönderi de eklenmemiş. Yanlış giden ne?

Yanıt: hiçbir şey. Sadece `view`'ımızda biraz daha iş yapmamız gerekiyor.

Formu kaydetme

Tekrar `blog/views.py` dosyasını açalım. Şuanki `post_new` view'i içinde sadece bunlar var:

```
def post_new(request):
    form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})
```

Formu kaydettiğimizde tekrar aynı `view`'a yönlendirileceğiz, ama bu sefer `request` nesnesinde, daha doğrusu `request.POST` (isimlendirmesinin, bizim blog uygulamasının "post" modeli ile alakası yok, gerçekte veri gönderdiğimiz (İngilizcede "post" işlemi) için isimlendirme bu şekilde) içerisinde, daha fazla verimiz olacak. HTML dosyasında `<form>` tanımımızdaki `method="POST"` değişkenini hatırlıyor musun? Formdan gelen tüm alanlar şimdi `request.POST` 'un içerisinde. `POST` 'un ismini değiştirmememiz lazım (`method` için geçerli diğer değer sadece `GET` 'dir, ama şimdi ikisi arasındaki farkın ne olduğunu anlatacak kadar vaktimiz yok).

Oluşturduğumuz `view`'da iki ayrı durumu halletmemiz gerek. Birincisi: sayfaya ilk eriştiğimiz ve boş bir form istediğimiz zaman. İkincisi: henüz yazdığınız tüm form verileriyle `view`'a geri döndüğümüz zaman. Yani bir koşul eklememiz gerekiyor (bunun için `if` kullanacağız).

```
if request.method == "POST":  
    [...]  
else:  
    form = PostForm()
```

Şimdi boşluğu [...] doldurma zamanı geldi. `method`, `POST` ise o zaman `PostForm` u verilerle oluşturmamız lazım, değil mi? Bunu yapmak için:

```
form = PostForm(request.POST)
```

Çok kolay! Şimdi de formu (yani tüm gerekli alanların doldurulduğu ve hatalı değerlerin kaydedilmeyeceğini) kontrol etmemiz lazım. Bunu da `form.is_valid()` ile yapıyoruz.

Formun doğruluğunu kontrol ediyoruz ve doğru ise kaydedebiliriz!

```
if form.is_valid():  
    post = form.save(commit=False)  
    post.yazar = request.user  
    post.yayinlanma_tarihi = timezone.now()  
    post.save()
```

Temel olarak, burada iki şey yaptık: formu `form.save` ile kaydettik ve bir yazar ekledik (`PostForm` 'da bir `yazar` tanımlı olmadığı ve bu zorunlu bir alan olduğu için!). `commit=False`, `Post` modelini henüz kaydetmek istemediğimizi belirtir - ilk önce yazarı eklemek istiyoruz. Genellikle `form.save()`, `commit=False` olmadan kullanacaksınız, ama bu durumda, bunu kullanmamız gerekiyor. `post.save()` değişiklikleri (yazarı ekleme) koruyacaktır ve yeni bir blog gönderisi oluşturulur!

Hemen `post_detail` sayfasına gidip yeni yaratmış olduğumuz blog postunu görsek harika olur, değil mi? Bunu yapabilmek için önemli bir şey daha lazım:

```
from django.shortcuts import redirect
```

Bunu dosyanın en başına ekleyelim. Şimdi yeni yarattığımız blog postu için `post_detail` sayfasına gidebiliriz.

```
return redirect('blog.views.post_detail', pk=post.pk)
```

`blog.views.post_detail` gitmek istediğimiz görünümün ismidir. Unutmayalım ki bu view için bir `pk` değişkeni lazım. Bu değeri görüntümlere aktarmak için `pk=post.pk` yazarız. Burada `post` yeni yarattığımız blog postudur!

Çok şey söyledik ama herhalde `view` u tümüyle bir görmek isteriz artık, değil mi?

```

def post_new(request):
    if request.method == "POST":
        form = PostForm(request.POST)
        if form.is_valid():
            post = form.save(commit=False)
            post.yazar = request.user
            post.yayinlanma_tarihi = timezone.now()
            post.save()
            return redirect('blog.views.post_detail', pk=post.pk)
    else:
        form = PostForm()
    return render(request, 'blog/post_edit.html', {'form': form})yazar

```

Bakalım çalışacak mı? <http://127.0.0.1:8000/post/new/> sayfasına gidip bir `baslik` ve `yazi` ekleyelim, sonra da kayd edelim... ve işte! Yeni blog postu eklenmiş ve `post_detail` sayfasına yönlendirildik!

Postu kaydetmeden önce yayın tarihine değer atandığını fark etmiş olabilrsin. Daha sonra *yayınla butonu* nu **Django Girls Tutorial: Ek konular** da anlatacağız.

Süper!

Form doğrulama

Şimdi de Django formlarının ne kadar havalı olduğunu görelim. Bir blog postunun `baslik` ve `yazi` alanları olmalı. `Post` modelimizde bu alanlar mecburi değil demedik (`yayinlanma_tarihi` demiş olduğumuz gibi), dolayısı ile Django bu alanlara değer atanması gerektiğini varsayar.

`baslik` veya `yazi` olmadan formu kaydetmeye çalışın. Ne olacak, tahmin et!

![Form doğrulama][3]

[3]: images/form_validation2.png

Django tüm alanlara doğru tür değerlerin atandığını bizim için kontrol ediyor. Ne güzel, değil mi?

> Yakın zamanda Django'nun admin arayüzüni kullandığımız için, sistem bizi hala oturumda varsayıyor. Bazi durumlar bizim oturumdan çıkışmamızına neden olabilir (web tarayıcısını kapatmak, veritabanını tekrar başlatmak, vb). Eğer oturumda olan kullanıcı olmadığı için post yaratmadı hata alırsak admin sayfası olan <http://127.0.0.1:8000/admin> gidip tekrar bir oturum açmalıyız. Bu durumu geçici de olsa da halleder. Kalıcı çözüm, ana tutorialdan sonra **Ödev: Web sitene gidenlik ekleme!** bölümünde anlatılacak.

![Oturum hatası][4]

[4]: images/post_create_error.png

Form düzenleme

Artık yeni bir form oluşturmayı biliyoruz. Peki, mevcut bir formu güncellemek için ne yapmalı? Demin yaptığımıza çok benzıyor. Şimdi, hızlıca bir kaç şey yaratalım (anlamadığın bir şey olduğu zaman, mentöre veya önceki bölümlere danışabilirsin, çünkü bu adımları daha önce yaptık).

`blog/templates/blog/post_detail.html` dosyasını açıp şu satırı ekleyelim:

```

```python


```

ki template buna benzesin:

```
{% extends 'blog/base.html' %}

{% block content %}
<div class="post">
 {% if post.yayinlanma_tarihi %}
 <div class="date">
 {{ post.yayinlanma_tarihi }}
 </div>
 {% endif %}

 <h1>{{ post.baslik }}</h1>
 <p>{{ post.yazi|linebreaks }}</p>
 </div>
{% endblock %}
```



blog/urls.py dosyasına şu satırı ekleyelim:

```
url(r'^post/(?P<pk>[0-9]+)/edit/$', views.post_edit, name='post_edit'),
```

Daha önce kullandığımız blog/templates/blog/post\_edit.html template'i tekrar kullanacağız, tek eksik bir view.

Şimdi blog/views.py dosyasını açıp en sonuna şu satırı ekleyelim:

```
def post_edit(request, pk):
 post = get_object_or_404(Post, pk=pk)
 if request.method == "POST":
 form = PostForm(request.POST, instance=post)
 if form.is_valid():
 post = form.save(commit=False)
 post.yazar = request.user
 post.yayinlanma_tarihi = timezone.now()
 post.save()
 return redirect('blog.views.post_detail', pk=post.pk)
 else:
 form = PostForm(instance=post)
 return render(request, 'blog/post_edit.html', {'form': form})
```

Bu nerdeyse bizim post\_new view'e benziyor, değil mi? Ama, tam da değil. İlk önce: Url'den ekstra bir pk parameteresi yolladık. Sonra: güncellemek istediğimiz Post modelini get\_object\_or\_404(Post, pk=pk) ile alıp, bir form yarattığımızda bu postu bir örnek kopya (instance) olarak hem formu kaydettığımızde yolluyoruz:

```
form = PostForm(request.POST, instance=post)
```

hem de güncellemek istediğimiz postu görmek için form açtığımız zaman yolluyoruz:

```
form = PostForm(instance=post)
```

Haydi, deneyelim. Bakalım çalışacak mı? post\_detail sayfasına gidelim. Sağ üst köşede bir edit (güncelleme) butonu olmalı:

A screenshot of a web browser window titled "Django Girls Blog". The address bar shows "127.0.0.1:8000/post/3/". The main content area displays a blog post with the title "Nulla facilisi" and the date "June 28, 2014, 3:48 p.m.". A small edit icon is visible next to the date. Below the title is a large amount of placeholder text (Lorem ipsum).

Butona tıklaysak blog postunu görmemiz lazım:

A screenshot of a web browser window titled "Django Girls Blog". The address bar shows "127.0.0.1:8000/post/3/edit/". The main content area displays a blog post with the title "Nulla facilisi". Below the title is a "New post" section. It contains two form fields: "Title:" with the value "Nulla facilisi" and "Text:" with a large text area containing placeholder text.

İstediğimiz gibi başlık ve yazımı değiştirebilir ve sonra da kaydedebilriz!

Tebrikler! Uygulaman gittikçe tamamlanıyor!

Django formları hakkında daha fazla bilgi bulmak için <https://docs.djangoproject.com/en/1.8/topics/forms/> adresindeki dokümanlara bakabilirsin

## Güvenlik

Bir bağlantıya (link) tıklayarak yeni bir blog oluşturabilmek harika! Ancak, şu haliyle siteye gelen herkes bir blog yaratıp kaydedebilir. Bu da istenilen bir durum değil. Butonun sadece sana görünmesini sağlayalım.

`blog/templates/blog/base.html` dosyasında yarattığımız `page-header` `div` ve anchor etiketlerini (tags) bulalım. Şuna benziyor olmalı:

```

```

Şimdi bir `{% if %}` etiketi daha ekleyeceğiz ki link sadece admin olarak oturum açmış kişilere görünse. Simdilik, bu kişi sadece sensin! `<a>` etiketini şöyle değiştirelim:

```
{% if user.is_authenticated %}

{% endif %}
```

Bu `{% if %}` linkin tarayıcıya ancak kullanıcı oturum açmış ise gönderilmesini sağlar. Bu yeni post yaratılmasını kesin olarak engellemese de iyi bir başlangıç. Güvenlik konusu ek derslerde daha çok ele alınacak.

Oturum içi olduğumuz için, şimdi sayfayı yenilersek, farklı bir şey görmeyeceğiz. Sayfayı farklı bir tarayıcıda veya incognito bir pencerede yükleyelim. O zaman bu link görünmeyecek!

## Bir şey daha: deployment (yayına alma) zamanı!

Bakalım PythonAnywhere'de çalışacak mı? Tekrar yayına alalım!

- İlk önce kodumuzu commit edelim, sonra Github'a push edelim

```
$ git status
$ git add -A .
$ git status
$ git commit -m "Web sitesine güncelleme ve yaratma için view eklendi."
$ git push
```

- Sonra bir [PythonAnywhere Bash konsol](#) una gidip:

```
$ cd ilk-blogum
$ source myvenv/bin/activate
(myvenv)$ git pull
[...]
(myvenv)$ python manage.py collectstatic
[...]
```

- Nihayet, [Web tab](#)ına gidip **Reload** edelim.

O kadar! Tebrikler :)

## Sıradan var?

Kendini tebrik et! **Kesinlikle harikasın.** Seninle gurur duyuyoruz! <3

### Şimdi ne yapmak gereklidir?

Bir ara ver ve rahatla. Gerçekten çok büyük bir şey yaptın.

Bundan sonra şunları yaptığına emin ol:

- Güncel kalmak için Django Girls'ü [Facebook](#) ya da [Twitter](#) dan takip et

### Daha geniş kapsamlı kaynaklar önerilebilir misiniz?

Evet! Öncelikle [Django Girls Tutorial: Extensions](#) isimli diğer kitabı dene.

Sonra ise aşağıda listelenen diğer kaynakları deneyebilirsin. Hepsini öneriyoruz!

- [Django'nun resmi eğitimi](#)
- [Kodlamaya Yeni Başlayan \(New Coder\) eğitimi](#)
- [Code Academy Python kursu](#)
- [Code Academy HTML & CSS kursu](#)
- [Django Carrots eğitimi](#)
- [Learn Python The Hard Way kitabı](#)
- [Getting Started With Django](#) görüntülü dersleri
- [Two Scoops of Django: Best Practices for Django 1.8 kitabı](#)