**M V VEFVEFVVV-895463DJANGO**

Masaüstünde klasör yarat : **blog**

Open cmd in there (Klasör adres çubuğuna gel ve cmd yaz orada açar cmd yi)

**C:\Users\Demir\Desktop\blog>** **pip install virtualenv**

**C:\Users\Demir\Desktop\blog>** **virtualenv venv** //venv isimli bir klasör yarattı

**C:\Users\Demir\Desktop\blog>** **venv\Scripts\activate** // aktif ettik sanal zımbırtıyı

**(venv) C:\Users\Demir\Desktop\blog>** //En başta env görüyorsan olmuştur.

**(venv) C:\Users\Demir\Desktop\blog>deactivate //** yazarak kapatıyoruz sanal zımbırtıyı.

* Sanal python ortamını kurmamdaki amaç ana python dosyamdan ayırmam projemi. Bir izalasyon yaratmam.

**C:\Users\Demir\Desktop\blog>** **venv\Scripts\activate** // aktif ettik sanal zımbırtıyı

**(venv) C:\Users\Demir\Desktop\blog> pip install django**

**(venv) C:\Users\Demir\Desktop\blog> django-admin startproject blog .** // proje yaratıldı.

* Şimdi pycharmda blog adlı projemizi açalım ve settings en altta dil te zaman Europe/Istanbul yapalım
* dJango bize sunucu desteğide sağlıyor.

**(venv) C:\Users\Demir\Desktop\blog> dir** // baktık manage dosyası burada mı diye.

04/04/2020 22:25 <DIR> .idea

04/04/2020 22:25 <DIR> blog

04/04/2020 22:08 645 manage.py

**(venv) C:\Users\Demir\Desktop\blog>python manage.py runserver** // Serverı çalıştırdık

… Django version 3.0.5, using settings 'blog.settings'

Starting development server at <http://127.0.0.1:8000/> …

* <http://127.0.0.1:8000/> ADRESİNİ TARAYICINA GİR VE MUHTEŞEM SONUCU GÖR 😊
* Komut satırından **Ctrl+c** ile server sonlandırılabilir.

**(venv) C:\Users\Demir\Desktop\blog>python manage.py runserver 8080**

* 8080 Port numarasından erişilebilirsin.
* … Starting development server at <http://127.0.0.1:8080/> …

**SUPERUSER CREATE\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**> venv\Scripts\activate //** Daha önce yaratıtğımız venv sanal projesini aktif ettik.

**> python manage.py runserver**  // çalıştırdık.

You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin, auth, contenttypes, sessions.

Run 'python manage.py migrate' to apply them. //dedi

\* Ctrl + c ile çıkıp. '

**> python manage.py migrate** // dedik. Ve admin sayfası vs. aktif dilmiş oldu.

**(venv) C:\Users\Demir\Desktop\blog> venv\Scripts\activate**

**(venv) C:\Users\Demir\Desktop\blog> python manage.py runserver**

* <http://127.0.0.1:8000/admin> yazıp admin paneline girdik.
* Bu arada komut satıtımızda server açık ama biz başka işlemlerde yapacağız sanal makinada o yüzden yeni bir cmd daha açıp orada işlerimizi yapıyoruz. Örneğin bir sanal user yaratacağız.

**(venv) C:\Users\Demir\Desktop\blog> python manage.py createsuperuser**

Kullanıcı adı (leave blank to use 'demir'): yunus

E-posta adresi:

Password:

* Siteden gir bak yetkilerine.
* Gruplar/yetkiler//öğrenci/öğretmen vs
* Kişi ekle sil güncelle vs.

**Uygulama yaratma\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**(venv) C:\Users\Demir\Desktop\blog> python manage.py startapp post**

**Post/models** içine

class Post(models.Model):  
 title = models.CharField(max\_length=120) # Max\_lengt belirtilmek zorundaymış.  
 text = models.TextField()  
 published\_date = models.DateTimeField()

**blog/settings/installedapps** altına bildiriyoruz yazdığımız modeli belirtiyoruz

‘post’,

**(venv) C:\Users\Demir\Desktop\blog> python manage.py makemigrations** // ve migrate altına yeni bir dosya oluştu artık migrate edebiliriz. Sql’e çevirdi yazdıkalarımızı

* Şimdi bakacağız.

**(venv) C:\Users\Demir\Desktop\blog>python manage.py sqlmigrate post 0001**

BEGIN; --CREATE TABLE "post\_post" ("id" integer NOT NULL PRIMARY KEY AUTOINCREMENT, "title" varchar(120) NOT NULL, "text" text NOT NULL, "published\_date" datetime NOT NULL);

COMMIT;

Post/admin

from .models import Post  
#from post.models import Post  
  
admin.site.register(Post)

**Admin panelini özelleştirmek\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**> venv\Scripts\activate**

**> python manage.py runserver**

* <http://127.0.0.1:8000/admin>
* yunus 123

class PostAdmin(admin.ModelAdmin):  
  
 list\_display = ["title", "published\_date"] # Ekranda neler görünsün.  
 list\_display\_links = ["title", "published\_date"]  
 list\_filter = ["published\_date"]  
 search\_fields = ["title", "text"]  
 # list\_editable = ["title"]  
  
 class Meta:  
 model = Post  
  
admin.site.register(Post, PostAdmin)

**Views**

**Post view’e aşağıdaki kodu yazıyoruz. Yani bu demekki bu viewe gelinince hoşgeldiniz yazısı dönülsün.**

from django.shortcuts import render,HttpResponse  
  
def home\_view(request):  
 return HttpResponse('<h>Hoşgeldiniz</h>')

**Daha sonra bu viewi blog/urls e tanıtıyoruz. Hangi url yazılınca bu gözüksün diye.**

from post.views import home\_view

path('', home\_view)

kök dizine tıklyınca bu metod gelsin diyoruz. Yani

>venv\Scripts\activate

>python manage.py runserver

Starting development server at <http://127.0.0.1:8000/>

Artık buraya girdiğinde hoşgeldiniz yazısı gözükmüş oluyor.

**Urls:**

**>python manage.py startapp home**

Home isimli bir uygulama oluşturduk. Ve viewine yukarıdaki view’e yaptığımızı yaptık.

**Post dediğimiz de bir ana sayfa olcak onun alt sayfaları olacak bunların tutulacağı urls dosyası lazım bu otomatikman gelmedi bunu elimizle oluşturyoruz. Ve olası alt url’leri oluşturuyoruz.**

from django.urls import path  
urlpatterns = [  
 path('index', post\_index),  
 path('details', post\_details),  
…  
]

**Şimdi bunların view’lerini oluşturacağız.**

from django.shortcuts import render,HttpResponse  
  
def post\_index(request):  
 return HttpResponse('<h>İndex Page</h>')  
  
def post\_detail(request):  
 return HttpResponse('<h>Detail Page</h>')

…

**Tekrar post/urls gidip**

from .views import \*

**blog/urls e post/urls i tanıt.**

from django.urls import path, include

path('post/', include('post.urls') ),

**Template:**

* Blog/settings/templats metodu içine az sonra oluşturacağımız templates dosyasının yolunu gösteriyoruz. BASE\_DIR içinde.

TEMPLATES = [  
 {…  
 'DIRS': [os.path.join(BASE\_DIR, "templates")],

…

* Blog / sağ click new directory (template). Template sağ click new htlm (home)
* Home /views dizinine gidip

def home\_view(request):  
 return render(request, 'home.html', {})

Dinamik hale getirmek:

* Home/ views
* def home\_view(request):  
   if request.user.is\_authenticated:  
   context = {'isim' : 'yunus'}  
   else:  
   context = {'isim': 'Misafir'}  
   return render(request, 'home.html', context)
* burada context değişkeniyle karşı tarafa bir değişken göndermiş olduk karşı taraftan ise.

Templates/home {{isim}} diyerek karşıladık.

# **Django Shell ve Database API**

1. **Open shell**

**(myvenv) ~/djangogirls$ python manage.py shell**

* Şu an Django'nun etkileşimli konsolundayız. Önce tüm post'larımızı görüntülemeyi deneyelim. Bunu aşağıdaki komut ile yapabiliriz:

1. **In [3]: from post.models import Post**
2. **In [4]: Post.objects.all()**

**Out[4]: <QuerySet [<Post: sg>, <Post: sdfsdfdfsfsfs>]>**

**In [5]: from django.utils import timezone**

**In [6]: suan = timezone.now()**

1. **Create:**

**In [8]: Post.objects.create(title = "Deneme123", text = "deneme123", published\_date = suan)**

**Out[8]: <Post: Deneme123>**

**In [9]: Post.objects.all()**

**Out[9]: <QuerySet [<Post: sg>, <Post: sdfsdfdfsfsfs>, <Post: Deneme123>]>**

1. **Create 2:**

**In [10]: nesne = Post()**

**In [11]: nesne.title = "Deneme1"**

**In [12]: nesne.text = "Deneme1"**

**In [13]: nesne.published\_date = "suan"**

**In [14]: Post.objects.all()**

**Out[14]: <QuerySet [<Post: sg>, <Post: sdfsdfdfsfsfs>, <Post: Deneme123>]>** //KAYDETMEMİŞ KAYDETMEK İÇİN SAVE DİYEEECEĞİZ.

**In [18]: nesne.save()**

**In [19]: Post.objects.all()**

**Out[19]: <QuerySet [<Post: sg>, <Post: sdfsdfdfsfsfs>, <Post: Deneme123>, <Post: Deneme1>]>**

**In [20]: nesne.title = "Guncellenen baslik"**

**In [21]: nesne.save()**

**In [22]: Post.objects.all()**

**Out[22]: <QuerySet [<Post: sg>, <Post: sdfsdfdfsfsfs>, <Post: Deneme123>, <Post: Guncellenen baslik>]>**

1. **Delete:**

**In [23]: a = Post.objects.get(id=3)**

**In [24]: a**

**Out[24]: <Post: Deneme123>**

**In [25]: a.delete()**

**Out[25]: (1, {'post.Post': 1})**

**In [27]: Post.objects.all()**

**Out[27]: <QuerySet [<Post: sg>, <Post: sdfsdfdfsfsfs>, <Post: Guncellenen baslik>]>**

1. **Filter:**

**In [30]: Post.objects.filter(title\_\_contains = "sg")** // büyük küçük duyarlı ama sqlite zaten

değil.

**Out[30]: <QuerySet [<Post: sg>]>**

**In [30]: Post.objects.filter(title\_\_icontains = "sg")** // büyük küçük suyarsız getir.

**In [34]: Post.objects.filter(published\_date\_\_lte = suan )** //least than equal şuan

**Out[34]: <QuerySet [<Post: sg>, <Post: sdfsdfdfsfsfs>, <Post: Guncellenen baslik>, <Post: Deneme123>]>**

**In [39]: Post.objects.order\_by('published\_date')**

**Out[39]: <QuerySet [<Post: sg>, <Post: sdfsdfdfsfsfs>, <Post: Guncellenen baslik>, <Post: Deneme123>]>** exit()

**Django nedir?**

Django, (framework). Bir web iskeleti, web sitesi geliştirmeyi hızlandıran ve kolaylaştıran bir grup bileşendir. So it is The library of libraries

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

Django'nun **url çözücüsü** (urlresolver) tarafından yapılıyor (web sitesi adresine URL -Uniform Resource Locator- deniyor, dolayısıyla url çözücü ismi mantıklı oluyor). Çok akıllı değildir -kalıpların bir listesini alır ve URL'yi eşleştirmeye çalışır. Django kalıpları üstten alta doğru denetler ve eşleşen bir şey varsa isteği ilgili işleve (view) aktarır. Tüm ilginç şeyler view fonksiyonunda yapılır:

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

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

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

Bir mektubu nereye götüreceğini kontrol eden postacının hakkında konuştuğumuzu hatırlıyor musun? urls.py dosyası urlresolver tarafından kullanılan kalıpların bir listesini içerir.

**Web sunucusunu başlatmak**

**manage.py** dosyasının bulunduğu dizinde olmalıyız.

**python manage.py runserver**  komutunu çalıştırarak web sunucusunu başlatabilirsiniz:

--

**Peki bir nesne nedir?**

Nesne bir özellikler ve hareketler bütünüdür. Eğer bir kediyi modellemek istiyorsak, Kedi nesnesini oluştururuz ve bu nesne şöyle özelliklere sahip olur: renk, yas, ruh\_hali. Ayrıca Kedi bazı hareketlere sahiptir: miyavla ya da beslen (bu durumda kediye biraz KediMamasi vermemiz gerekir ki o da kendine ait özellikleri olan başka bir nesne olarak tanımlanabilir, bu durumda onun özellikleri arasında da tat yer alabilir). Yani aslında ana fikir, gerçek nesneleri kod içinde özellikleri (nesne özellikleri) ve hareketleri (metodlar) ile tanımlamak.

Öyleyse blog gönderilerini nasıl modelleyeceğiz? Bir blog tasarlamak istiyoruz, degil mi? Cevaplamamız gereken soru: Bir 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 de yazara ihtiyacımız var. Son olarak, gönderinin ne zaman yaratıldığını ve yayınlandığını da bilmek isteyebiliriz.

**Post**

**--------**

**Title, text, author, created\_date, published\_date**

Bir blog gönderisi ile ne tür şeyler yapılabilir? Gönderiyi yayınlayan bir method olması güzel olurdu, değil mi? Bu yüzden **publish** metoduna ihtiyacımız olacak.

**Django'da modeller özel bir çeşit nesnedir ve Veri tabanına kaydedilir.** Veri tabanında, kullanıcılar, blog gönderileri, vs. ile ilgili bilgileri saklarız. Verilerimizi depolamak için SQLite veri tabanını kullanacağız. Veri tabanındaki bir modeli, sütunları (alan adı) ve satırları (veri) olan bir hesap çizelgesi olarak düşünebilirsiniz**.**

### **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 çalıştırmamız gerekiyor:

(myvenv) C:\Users\Name\djangogirls> python manage.py startapp blog

Uygulamamızı oluşturduktan sonra, Django'ya bunu kullanmasını söylememiz lazım. Bunu **mysite/settings.py** içinde INSTALLED\_APPS'ı içine 'blog', yazacağız:

**mysite/settings.py**

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'blog',

]

* str'nin her iki tarafında da 2 tane alt çizgi (\_) kullandığımızdan emin olalım. İki alt çizgi Python dilinde sık kullanılır, bazen "dunder" olarak da anılır ("double-underscore"un kısaltması).

models.CharField - belirli bir uzunluktaki metinleri tanımlamak için kullanılır.

models.TextField - bu da uzun metinleri tanımlar.

models.DateTimeField - bu da gün ve saati tanımlamada kullanılır.

models.ForeignKey - başka bir modele referans tanımlar.

### **Modeller için veri tabanında tablo oluşturma**

Son adımımız yeni modelimizin veri tabanına eklenmesini sağlamak. Öncelikle Django'nun modelimizde bazı değişiklikler yaptığımızı bilmesini sağlamalıyız. (Daha yeni oluşturduk!) Komut satırına gidelim ve **python manage.py makemigrations blog** yazalım.

**Not:** Düzenlediğiniz dosyaları kaydetmeyi unutmayın. Aksi takdirde, bilgisayarınız önceki sürümü çalıştırarak beklenmedik hatalar verebilir.

Django bize veritabanımıza uygulayabileceğimiz bir taşıma (migration) dosyası oluşturdu.

**python manage.py migrate blog**

Django admin

Modellediğimiz gönderilere ekleme, düzenleme ya da silme işlemi yapmak için Django admini kullanacağız. Şimdi **blog/admin.py** dosyasını açarak içeriğini aşağıdaki şekilde değiştirelim:

**blog/admin.py**

**from django.contrib import admin**

**from .models import Post**

**admin.site.register(Post)**

Gördüğünüz gibi, bir önceki bölümde tanımladığımız gönderi modelini **admin.py** dosyamıza dahil **(import)** ettik. Admin sayfasında modelimizi görünür kılabilmek için, modeli **admin.site.register(Post)** ile kaydetmemiz gerekiyor.

Giriş yapabilmek için, sitedeki her şey üzerinde kontrolü olan superuser - bir kullanıcı hesabı oluşturmanız gerekiyor. Komut satırına aşağıdakileri yazıp devam edin.

**(myvenv) C:\Users\Name\djangogirls> python manage.py createsuperuser**

# **Yayına alın!**

**GİT**

Hatırlatma: Kullanıcı adı seçerken 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 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(txt sonra txt’sini silersin):

.gitignore

\*.pyc

\*~

\_\_pycache\_\_

myvenv

db.sqlite3

/static

.DS\_Store

Ve onu .gitignore ismi ile ilgili yere kaydedin.

komut-satırı

$ git status

On branch master

No commits yet

Untracked files:

(use "git add <file>..." to include in what will be committed)

.gitignore

blog/

manage.py

mysite/

requirements.txt

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:

komut-satırı

$ git add --all .

$ git commit -m "Django Girls uygulamam, ilk commit"

[...]

13 files changed, 200 insertions(+)

create mode 100644 .gitignore

[...]

create mode 100644 mysite/wsgi.py

## Kodunu GitHub'a yollama (Push)

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

Aşağıdakini komut satırına yazın **(< your-github-username>)** kısmını Github hesabını yarattığınız sırada kullandığınız kullanıcı adı ile değiştirin, büyüktür küçüktür işaretlerini eklemeyin):

komut-satırı

$ git remote add origin https://github.com/<github-kullanıcı-adınız>/my-first-blog.git

$ git push -u origin master

(fatal falan verirse zaten var diye: git remote rm origin)

Github kullanıcı adı ve şifrenizi girin, ardından 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/my-first-blog.git

\* [new branch] master -> master

Branch master set up to track remote branch master from origin.

Kodunuz artık Github'da. Hemen gidip bak! İyi bir çevrede olduğunu göreceksiniz - [Django](https://github.com/django/django" \t "_blank), the [Django Girls Tutorial](https://github.com/DjangoGirls/tutorial" \t "_blank), ve daha birçok harika açık kaynak yazılım projesi de kodlarını Github'da tutuyor. :)

### …or create a new repository on the command line

echo "# deneme\_1" >> README.md

git init

git add README.md

git commit -m "first commit"

git remote add origin https://github.com/yunusemre002/deneme\_1.git

git push -u origin master

### …or push an existing repository from the command line

git remote add origin https://github.com/yunusemre002/deneme\_1.git

git push -u origin master

### …or import code from another repository

You can initialize this repository with code from a Subversion, Mercurial, or TFS project.

[**Import code**](https://github.com/yunusemre002/deneme_1/import)

# **Blogumuzun PythonAnywhere üzerinde kurulumu**

Localde değişim yap 🡪 Githuba at 🡪 Canlıya çek. (Pythonanywhere)

1. [www.pythonanywhere.com](http://www.pythonanywhere.com) sitesine git hesap aç. **Not** Burada kullanıcı adınızı seçerken, bloğunuzun URL'sinin YourUserName.pythonanywhere.com, şeklini alacak

## PythonAnywhere API token oluşturma

Bu sadece bir kez yapmanız gereken bir şey. PythonAnywhere'e kaydolduğunuzda, dashboard sayfasına yönlendirileceksiniz. "Account" sayfanızın sağ üstüne yakın link'i bulun ve "API token" isimli sekmeyi seçerek "Create new API token" butonuna basın.

## Sitemizin PythonAnywhere üzerinde yapılandırılması

Ana [PythonAnywhere Dashboard](https://www.pythonanywhere.com/" \t "_blank) a logosuna tıklayarak dönelim ve bir "Bash" console başlatalım – bu bilgisayarınızdaki komut satırının PythonAnywhere versiyonudur.

PythonAnywhere command-line

**$ pip3.6 install --user pythonanywhere**

**… Succesfully …**

**$ pa\_autoconfigure\_django.py** [**https://github.com/<github-kullanıcı-adınız>/my-first-blog.git**](https://github.com/%3cgithub-kullanıcı-adınız%3e/my-first-blog.git)

* Kodunuz GitHub'dan çekiliyor. Tıpkı kendi bilgisayarındaki gibi PythonAnywhere üzerinde bir virtualenv oluşturuluyor. Yayına almak için gerekli ayarlar (settings) dosyası güncelleniyor. Yine manage.py migrate komutu ile PythonAnywhere üzerinde veritabanı oluşturuluyor. Sabit dosyaların (bunları daha sonra öğreneceğiz) oluşturuluyor. Ve web uygulamanızın API sinin PythonAnywhere tarafından sunulması için ayarlar yapılıyor

1. PythonAnywhere üzerinde yaratılan veritabanı ile kendi bilgisayarımızda yarattığımız veritabanı birbirinden tamamen ayrı -- dolayısı ile bu iki veritabanında saklanmış olan postlar ve kullanıcılar da farklı olabilir.Bu nedenle, aynen kendi bilgisayarımızda yapmış olduğumuz gibi, createsuperuser ile bir admin kullacısını oluşturmamız gerekiyor. PythonAnywhere otomatik olarak virtualenv i başlatmış olduğu için, bu kullanıcıyı hemen oluşturabiliriz:

PythonAnywhere komut satırı

**(<kullanici-adiniz>.pythonanywhere.com) $ python manage.py createsuperuser**

## Şimdi Canlı Yayındasınız!

Web siteniz şimdi İnternet üzerinden erişilebilir olmalı! PythonAnywhere "Web" tabına tıklayarak linki kopyalayın. Bu linki istediğiniz herkes ile paylaşabilirsiniz :)

**Not** Basit tutmak amacıyla siteyi yayına alırken güvenlik açısından ideal olmayan bir kaç seçim yaptık. Güvenlik tavsiyeleri için [Django deployment checklist](https://docs.djangoproject.com/en/2.0/howto/deployment/checklist/" \t "_blank) i inceleyin.

Not Üçlü tırnaklar (''' or """) arasındaki satırlara docstrings denir

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

## Bir şey daha: deployment (yayına alma) zamanı!

### Kodumuzu commit ve push komutları ile Github'a yükleyelim

**Cmd**

**$ git status**

**$ git add --all .**

* **Not** --all demek eğer git'in silinmiş dosyaları varsa onları tanımayı sağlar (varsayılan olarak sadece yeni/değiştirilmiş dosyaları tanır). Hatırlatma: . içinde olduğumuz klasör anlamına gelir.
* Dosyalarımızı yüklemeden önce git'in hangilerini yükleyeceğine (git'in yükleyeceği dosyalar yeşil gösterilir) bakalım:

**$ git status**

Commit için değişiklikleri açıklayan bir mesaj yazalım.

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

* Bunu bir kez yapıp, değişikliklerimizi GitHub'a yükledik (ittik):

**$ git push**

### Pull ile yeni kodu PythonAnywhere e alıp web uygulamasını tekrar yükleyelim

* [PythonAnywhere consoles page](https://www.pythonanywhere.com/consoles/) sayfasını ve **Bash console** u açalım (ya da yeni bir tane açalım). Sonra da çalıştıralım:

PythonAnywhere komut satırı

**$ cd ~/<your-pythonanywhere-username>.pythonanywhere.com**

**$ git pull**

**[...]**

* (Açı parantezleri olmadan <your-pythonanywhere-username>'i gerçek PythonAnywhere kullanıcı isminizle değiştirmeyi unutmayın).
* Ve kodunuzun indirilmesini izleyin. Kontrol etmek istersen, **Files** sekmesine gidip kodunu PythonAnywhere üzerinde görebilirsin (Konsol sayfasındaki menü butonundan diğer PythonAnywhere sayfalarına ulaşabilirsin).
  + Son olarak, [Web sekmesi](https://www.pythonanywhere.com/web_app_setup/)ne gidip uygulamanızın **Yenile** butonuna basın.

**Django ORM ve QuerySets (SorguSetleri)**

Django'nun veritabanına nasıl bağlandığını ve veriyi nasıl sakladığını öğreneceğiz.

**QuerySet (SorguSeti) Nedir?**

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

**Django shell (kabuk)**

komut-satırı