Python and Django Assignments and Lab Exercise

PYTHON

Installing Python for Linux –

The steps are:

i) First you have to update your APT Repository. You have to type the command in your terminal:

$ apt-get update

ii) Now, install Python by typing the command –

$ apt-get install python3.6

iii) Now, verify Python by typing

$ python

For Python3 type the command goes something like this.

$ python3

Installing Python in Macintosh systems –

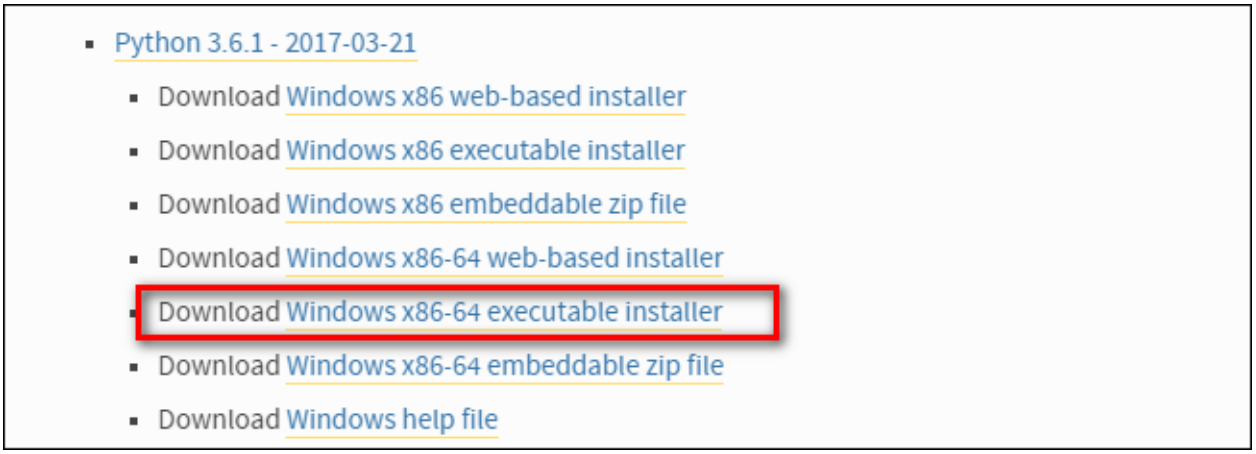
You do not need to install or configure anything else for using Python 2. These instructions are documented in the installation of Python 3. The version of Python which ships with OS-X becomes good for learning.

But if you want to have a stable release for Python, you will need to install GCC first. GCC can be obtained by downloading Xcode (https://developer.apple.com/xcode/), the smaller Command Line Tools (https://developer.apple.com/downloads/) (for this you must have an Apple account) or even the smaller OSX-GCC-Installer (https://github.com/kennethreitz/osx-gcc-installer#readme) package.

Installing Python in Windows 10

Python doesn't come pre-packaged with Windows, but that doesn't mean Windows users won't find the flexible programming language useful. For this go to the official website of Python (https://www.python.org/downloads/windows/).

From there, you have to download your system compatible latest Python released version. It comes with both Windows x86-64 web-based installer as well as Windows x86 embeddable zip file / Windows x86 executable installer. Once the file is downloaded, you have to double click to install it.



As soon as you run the setup file of the Python installer, you will see a screen where you will have to choose a radio button either "Install for all users" or "Install just for me". Click Next > Next and let it install Python interpreter into your system.

Now to check whether Python has been installed in your system, you have to open command prompt in your system and type the following command –

python -v

Python 2 vs. Python 3 –

Even though Python 3 is the most recent version and generation of this language, many programmers still exercise Python 2.7 which is the final update to Python version 2, and was released in 2010. Programming languages continually grow as developers extend its functionality as well as iron out quirks which will cause problems for the developers. Python 3 was introduced

in the year 2008 with the aim of creation of a programming language that is easier to apply and change the easy to handle strings for matching the demands placed on the language today.

Python 3.0 is basically unlike to its previous releases because it is the first Python release which is not well-suited with its older versions. Programmers typically do not have to be concerned regarding the minor updates (e.g. from 2.6 to 2.7) as they frequently change the internal workings of Python and do not need programmers for changing their syntax. The modification between Python 2.7 and Python 3.0 is much more noteworthy — syntaxes that work in Python 2.7 may require to be written in a changed way to work in Python 3.0.

Basic Hello World Program –

```
>>> print('Hello World!')
Hello World!
>>> 2 + 5
7
>>> print('Welcome to Real Python!')
```

Fibonacci Series using Function in Python

```
def recur_fibo(n):
  if n <= 1:
     return n
  else:
     return(recur_fibo(n-1) + recur_fibo(n-2))
# take input from the user
nterms = int(input ("How many terms? "))
# check if the number of terms is valid
if nterms <= 0:
  print("Please enter a positive integer")
else:
  print("Fibonacci sequence:")
  for i in range(nterms):
```

```
print(recur_fibo(i))
```

Different ways of creating String in Python

```
# all of the following are equivalent
my_string = 'Hello'
print(my_string)
my_string = "Hello"
print(my_string)
my_string = '''Hello'''
print(my_string)
# triple quotes string can extend multiple lines
my_string = """Hello, welcome to
     the world of Python"""
print(my_string)
```

Allows negative indexing for its sequences using Python Program

```
str = 'programming'
print('str = ', str)

#first character
print('str[0] = ', str[0])

#last character
print('str[-1] = ', str[-1])
```

```
#slicing 2nd to 5th character
print('str[1:5] = ', str[1:5])

#slicing 6th to 2nd last character
print('str[5:-2] = ', str[5:-2]
```

Multi-line string:

1). Adding black slash at the end of each line.

Eg:

```
>>> text1='hello\
user'
>>> text1
'hellouser'
>>>
```

2)Using triple quotation marks:-

Eg:

```
>>> str2='''welcome
to
SSSIT'''
>>> print str2
welcome
to
SSSIT
>>>
```

String Formatting in Python –

```
print(out a greeting to that user.)


# This prints out "Hello, John!"
name = "John"
print("Hello, %s!" % name)


To use two or more argument specifiers, use a tuple (parentheses):
# This prints out "John is 23 years old."
name = "John"
age = 23
print("%s is %d years old." % (name, age))
```

Implement Lists in Python –

```
>>> list=['aman',678,20.4,'saurav']
>>> list1=[456,'rahul']
>>> list


list = ['physics', 'chemistry', 1997, 2000];
print "Value available at index 2 : "
print list[2]
list[2] = 2001;
print "New value available at index 2 : "
print list[2]
```

```
list1 = ['physics', 'chemistry', 1997, 2000];
print list1
del list1[2];
print "After deleting value at index 2 : "
print list1
```

Write a program to compare the details of input from the list and display it.

```
country=["India", "Russia", "Srilanka", "China", "Brazil"]
is_member = input("Enter the name of the country: ")
if is_member in country:
print(is_member, " is the member of BRICS")
else:
print(is_member, " is not a member of BRICS")
```

Indentation in Conditional statements –

```
for i in range(1,11):
    print(i)
    if i == 5:
        break
```

Multiple Ifs in a Python Program

```
num = 3
if num > 0:
    print(num, "is a positive number.")
```

```
print("This is always printed.")


num = -1
if num > 0:
   print(num, "is a positive number.")
print("This is also always printed.")
```

Nested if statements –

```
num = float(input("Enter a number: "))
if num >= 0:
   if num == 0:
      print("Zero")
   else:
      print("Positive number")
else:
   print("Negative number")
```

Looping using While statement –

```
n = 10
# initialize sum and counter
sum = 0
i = 1

while i <= n:
   sum = sum + i
```

```
    i = i+1    # update counter

# print the sum
print("The sum is", sum)
```

Python program using for Loop

```
numbers = [6, 5, 3, 8, 4, 2, 5, 4, 11]

# variable to store the sum
sum = 0

# iterate over the list
for val in numbers:
        sum = sum+val

# Output: The sum is 48
print("The sum is", sum)
```

Range function in Python –

```
print(range(10))

print(list(range(10)))

print(list(range(2, 20, 3)))
```

Tuples in Python –

```
t1 = ('MIT', 'Illinois', 1974, 1982);

t2 = (9, 7, 5, 3, 1, 2, 4 );

print "t1[0]: ", t1[0];

print "t2[1:5]: ", t2[1:5];
```

```
tpl = ('JavaScript', 'Go', 'ASP', 'CSharp');
print tpl;
del tpl;
print "After deleting tup : ";
print tpl;
```

Iterating through Sequence (Lists & Tuples) –

```
list = [1, 3, 5, 7, 9]
# Using for loop
for i in list:
   print(i)
```

Python program for Enumerate() –

```
l1 = ["eat","sleep","repeat"]
# printing the tuples in object directly
for ele in enumerate(l1):
   print ele
print
# changing index and printing separately
for count,ele in enumerate(l1,100):
   print count,ele
```

Write a Python code to illustrate generator, yield() and next().

```
def generator():
   t = 1
   print 'First result is ',t
```

```
        yield t

        t += 1
        print 'Second result is ',t
        yield t

        t += 1
        print 'Third result is ',t
        yield t

    call = generator()
    next(call)
    next(call)
next(call)
```

Dictionary using Python Program –

```
    d = {'Name': 'Karlos', 'Salary': 850000, 'Job': 'Researcher'}
    print "d['Name'] is: ", d['Name']
    print "d['Age'] is: ", d['Salary']

    d = {'Name': 'Karlos', 'Height': 6, 'Job': 'Researcher'}
    del d['Name']; # this will remove entry having the key 'Name'
    dict.clear();      #  this will remove each and every entry of your dictionary
    del dict ;         # this will delete the complete dictionary
    print "dict['Age']: ", dict['Age']
    print "dict['School']: ", dict['School']
```

Implement the concept of dictionary to store a some username and password and display it

```
gaurav = {'username': 'karlos ray', 'online': True, 'password': 9876}

abhi = {'username': 'abhiS', 'online': False, 'password': 019284}
```

Sets in Python –

```
Set = set(["a", "b", "c"])


print("Set: ")

print(Set)

# Adding element to the set

Set.add("d")

print("\nSet after adding: ")

print(Set)
```

Functions in Python –

```
# A simple Python function to check

# whether x is even or odd

def evenOdd( x ):

   if (x % 2 == 0):

      print "even"

   else:

      print "odd"
```

```
evenOdd(2)
evenOdd(3)
```

Creating a file using Python -

```
# Python code to create a file
file = open('geek.txt','w')
file.write("This is the write command")
file.write("It allows us to write in a particular file")
file.close()
```

Opening a file in binary mode

```
f = open('my_file', 'w+b')
byte_arr = [120, 3, 255, 0, 100]
binary_format = bytearray(byte_arr)
f.write(binary_format)
f.close()
```

Use of Pickle in Python –

```
import pickle

def storeData():
    # initializing data to be stored in db
    Omkar = {'key' : 'Omkar', 'name' : 'Omkar Pathak',
    'age' : 21, 'pay' : 40000}
    Jagdish = {'key' : 'Jagdish', 'name' : 'Jagdish Pathak',
    'age' : 50, 'pay' : 50000}
```

```
    # database
    db = {}
    db['Omkar'] = Omkar
    db['Jagdish'] = Jagdish

    # Its important to use binary mode
    dbfile = open('examplePickle', 'ab')

    # source, destination
    pickle.dump(db, dbfile)
    dbfile.close()

def loadData():
    # for reading also binary mode is important
    dbfile = open('examplePickle', 'rb')
    db = pickle.load(dbfile)
    for keys in db:
        print(keys, '=>', db[keys])
    dbfile.close()

if __name__ == '__main__':
    storeData()
loadData()
```

Exception handling Program in python –

```
try:
   linux_interaction()
except AssertionError as error:
   print(error)
else:
   try:
      with open('file.log') as file:
         read_data = file.read()
   except FileNotFoundError as fnf_error:
      print(fnf_error)
finally:
print('Cleaning up, irrespective of any exceptions.')
```

Program for Regular Expressions –

```
import re
pattern = '^a...s$'
test_string = 'abyss'
result = re.match(pattern, test_string)
if result:
  print("Search successful.")
else:
print("Search unsuccessful.")
```

Use of Class - OOP in Python

```
class Employee:
```

```
'Common base class for all employees'
empCount = 0

def __init__(self, name, salary):
  self.name = name
  self.salary = salary
  Employee.empCount += 1

def displayCount(self):
 print "Total Employee %d" % Employee.empCount

def displayEmployee(self):
  print "Name : ", self.name,  ", Salary: ", self.salary
```

Another Example –

```
class Dog:

  # A simple class
  # attribute
  attr1 = "mamal"
  attr2 = "dog"

  # A sample method
  def fun(self):
     print("I'm a", self.attr1)
     print("I'm a", self.attr2)
```

```
# Driver code
# Object instantiation
Rodger = Dog()

# Accessing class attributes
# and method through objects
print(Rodger.attr)
Rodger.fun()
```

Getter and Setter in Class –

```
class Celsius:
    def __init__(self, temperature = 0):
        self.set_temperature(temperature)

    def to_fahrenheit(self):
        return (self.get_temperature() * 1.8) + 32

    # new update
    def get_temperature(self):
        return self._temperature

    def set_temperature(self, value):
        if value < -273:
            raise ValueError("Temperature below -273 is not possible")
    self._temperature = value
```

Private Methods in Python –

```
# demonstrate private methods

# Creating a Base class
class Base:

    # Declaring public method
    def fun(self):
        print("Public method")

    # Declaring private method
    def __fun(self):
        print("Private method")

# Creating a derived class
class Derived(Base):
    def __init__(self):

        # Calling constructor of
        # Base class
        Base.__init__(self)

    def call_public(self):
```

```
        # Calling public method of base class
        print("\nInside derived class")
        self.fun()

    def call_private(self):

        # Calling private method of base class
        self.__fun()

# Driver code
obj1 = Base()

# Calling public method
obj1.fun()

obj2 = Derived()
obj2.call_public()
```

Inheritance in Python –

```
class Parent:       # define parent class
  parentAttr = 100
  def __init__(self):
    print "Calling parent constructor"

  def parentMethod(self):
```

```
    print 'Calling parent method'

  def setAttr(self, attr):
    Parent.parentAttr = attr

  def getAttr(self):
    print "Parent attribute :", Parent.parentAttr

class Child(Parent): # define child class
  def __init__(self):
    print "Calling child constructor"

  def childMethod(self):
    print 'Calling child method'
```

DJANGO

Make a Simple View

```
from django.http import HttpResponse
def hello(request):
  text = """<h1>welcome to my app !</h1>"""
  return HttpResponse(text)
```

Function based view in Python

```
# import the standard Django Model
# from built-in library
from django.db import models
```

```
# declare a new model with a name "GeeksModel"
class GeeksModel(models.Model):

    # fields of the model
    title = models.CharField(max_length = 200)
    description = models.TextField()

    # renames the instances of the model
    # with their title name
    def __str__(self):
        return self.title
```

Django's QueryDict program using Python –

```
def list_orders(request):
    """
    orders/index
    """
    args = request.GET.copy()
    default = {'state': Order.STATE_QUEUED}
    if len(args) < 2:  # search form not submitted
        f = request.session.get("order_search_filter", default)
        args = QueryDict('', mutable=True)
        args.update(f)
    request.session['order_search_filter'] = args
    data = prepare_list_view(request, args)
    return render(request, "orders/index.html", data)
```

Rendering of Template –

```
from django.shortcuts import render
```

```
# Create your views here.
def geeks_view(request):
    # render function takes argument  - request
    # and return HTML as response
    return render(request, "home.html")

from django.contrib import admin
from django.urls import path, include
urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include("geeks.urls")),
]
```

RequestContext to render_to_response Program

```
from django.shortcuts import render_to_response
from django.template import RequestContext

def my_view(request):
    # View code here...
    return render_to_response('my_template.html',
                              my_data_dictionary,
                              context_instance=RequestContext(request))
```

Creating Django forms in Python

```
from django import forms
from .models import Post
class PostForm(forms.ModelForm):
    class Meta:
        model = Post
        fields = ('title', 'text',)
```

Example of Django's Form Validation

```
from django.db import models
# model named Post
class Post(models.Model):
    Male = 'M'
    FeMale = 'F'
    GENDER_CHOICES = (
    (Male, 'Male'),
    (FeMale, 'Female'),
    )
    # define a username filed with bound  max length it can have
    username = models.CharField( max_length = 20, blank = False,
                                null = False)
    # This is used to write a post
    text = models.TextField(blank = False, null = False)

    # Values for gender are restricted by giving choices
    gender = models.CharField(max_length = 6, choices = GENDER_CHOICES,
                              default = Male)
```

```
    time = models.DateTimeField(auto_now_add = True)
```

REST ATI with Django

```
from django.urls import include, path
from rest_framework import routers
from tutorial.quickstart import views
router = routers.DefaultRouter()
router.register(r'users', views.UserViewSet)
router.register(r'groups', views.GroupViewSet)
# Wire up our API using automatic URL routing.
# Additionally, we include login URLs for the browsable API.
urlpatterns = [
   path('', include(router.urls)),
   path('api-auth/', include('rest_framework.urls', namespace='rest_framework'))
]
```

Writing and Running Tests in Django –

```
from django.test import TestCase
from myapp.models import Animal

class AnimalTestCase(TestCase):
   def setUp(self):
      Animal.objects.create(name="lion", sound="roar")
      Animal.objects.create(name="cat", sound="meow")
```

```
    def test_animals_can_speak(self):
        """Animals that can speak are correctly identified"""
        lion = Animal.objects.get(name="lion")
        cat = Animal.objects.get(name="cat")
        self.assertEqual(lion.speak(), 'The lion says "roar"')
        self.assertEqual(cat.speak(), 'The cat says "meow"')
```

Doctests in Django –

```
# models.py
from django.db import models
class Animal(models.Model):
    """
    An animal that knows how to make noise
    # Create some animals
    >>> lion = Animal.objects.create(name="lion", sound="roar")
    >>> cat = Animal.objects.create(name="cat", sound="meow")
    # Make 'em speak
    >>> lion.speak()
    'The lion says "roar"'
    >>> cat.speak()
    'The cat says "meow"'
    """
    name = models.CharField(max_length=20)
    sound = models.CharField(max_length=20)
    def speak(self):
```

```
        return 'The %s says "%s"' % (self.name, self.sound)
```

Using Models Fields in Django –

```
from django.db import models


class Musician(models.Model):
   first_name = models.CharField(max_length=50)
   last_name = models.CharField(max_length=50)
   instrument = models.CharField(max_length=100)


class Album(models.Model):
   artist = models.ForeignKey(Musician, on_delete=models.CASCADE)
   name = models.CharField(max_length=100)
   release_date = models.DateField()
   num_stars = models.IntegerField()
```

Program for Filter QuerySets in Django

filters.py

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

class UserFilter(django_filters.FilterSet):
   class Meta:
      model = User
      fields = ['username', 'first_name', 'last_name', ]
```

The view is as simple as:

views.py

```
from django.contrib.auth.models import User
from django.shortcuts import render
from .filters import UserFilter

def search(request):
    user_list = User.objects.all()
    user_filter = UserFilter(request.GET, queryset=user_list)
    return render(request, 'search/user_list.html', {'filter': user_filter})
```

Then a route:

urls.py

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

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

And finally the template:

user_list.html

```
{% extends 'base.html' %}

{% block content %}
  <form method="get">
    {{ filter.form.as_p }}
    <button type="submit">Search</button>
  </form>
  <ul>
  {% for user in filter.qs %}
    <li>{{ user.username }} - {{ user.get_full_name }}</li>
  {% endfor %}
  </ul>
{% endblock %}
```

Django Lookups based on Models –

```
from __future__ import unicode_literals
from selectable.base import ModelLookup
from selectable.registry import registry
from .models import Fruit
class FruitLookup(ModelLookup):
   model = Fruit
   search_fields = ('name__icontains', )
registry.register(FruitLookup)
```

Deleting Objects in Django –

```
>>> a = Album.objects.get(pk = 2)
>>> a.delete()
>>> Album.objects.all()
```

Deleting multiple objects in Python –

```
>>> Album.objects.filter(genre = "Pop").delete()
>>> Album.objects.all()
```

Retrieving related records in Django –

```
>>> a = Album(title = "Abbey Road", artist = "The Beatles", genre = "Rock")
>>> a.save()
>>> a = Album(title = "Revolver", artist = "The Beatles", genre = "Rock")
>>> a.save()
```

To retrieve all the objects of a model, we write the following command:

```
>>> Album.objects.all()
```

Program for Django Q objects –

```
class Q(tree.Node):
 AND = 'AND'
 OR = 'OR'
 default = AND

 def __init__(self, *args, **kwargs):
  super(Q, self).__init__(children=list(args) + list(six.iteritems(kwargs)))


 def _combine(self, other, conn):
  if not isinstance(other, Q):
   raise TypeError(other)
  obj = type(self)()
  obj.connector = conn
  obj.add(self, conn)
  obj.add(other, conn)
  return obj

 def __or__(self, other):
  return self._combine(other, self.OR)
```

```
def __and__(self, other):
  return self._combine(other, self.AND)

def __invert__(self):
  obj = type(self)()
  obj.add(self, self.AND)
  obj.negate()
  return obj
```

Cookies program in Django –

```
from django.template import RequestContext
def login(request):
  username = "not logged in"
  if request.method == "POST":
    #Get the posted form
    MyLoginForm = LoginForm(request.POST)
  if MyLoginForm.is_valid():
    username = MyLoginForm.cleaned_data['username']
  else:
    MyLoginForm = LoginForm()
  response = render_to_response(request, 'loggedin.html', {"username" : username},
    context_instance = RequestContext(request))
  response.set_cookie('last_connection', datetime.datetime.now())
  response.set_cookie('username', datetime.datetime.now())
  return response
```

Configuring the URL Routes –

```
from django.conf.urls import url
from django.contrib import admin
from django.contrib.auth import views as auth_views

urlpatterns = [
   url(r'^login/$', auth_views.login, name='login'),
   url(r'^logout/$', auth_views.logout, name='logout'),
   url(r'^admin/', admin.site.urls),
]
```

Authentication Decorator in Django –

```
from functools import wraps
from urllib.parse import urlparse

from django.conf import settings
from django.contrib.auth import REDIRECT_FIELD_NAME
from django.core.exceptions import PermissionDenied
from django.shortcuts import resolve_url
[docs]def user_passes_test(test_func, login_url=None,
redirect_field_name=REDIRECT_FIELD_NAME):
   """
   Decorator for views that checks that the user passes the given test,
   redirecting to the log-in page if necessary. The test should be a callable
   that takes the user object and returns True if the user passes.
```

```
    """
    def decorator(view_func):
        @wraps(view_func)
        def _wrapped_view(request, *args, **kwargs):
            if test_func(request.user):
                return view_func(request, *args, **kwargs)
            path = request.build_absolute_uri()
            resolved_login_url = resolve_url(login_url or settings.LOGIN_URL)
            # If the login url is the same scheme and net location then just
            # use the path as the "next" url.
            login_scheme, login_netloc = urlparse(resolved_login_url)[:2]
            current_scheme, current_netloc = urlparse(path)[:2]
            if ((not login_scheme or login_scheme == current_scheme) and
                    (not login_netloc or login_netloc == current_netloc)):
                path = request.get_full_path()
            from django.contrib.auth.views import redirect_to_login
            return redirect_to_login(
                path, resolved_login_url, redirect_field_name)
        return _wrapped_view
    return decorator


[docs]def login_required(function=None,
redirect_field_name=REDIRECT_FIELD_NAME, login_url=None):
    """
    Decorator for views that checks that the user is logged in, redirecting
    to the log-in page if necessary.
    """
```

```
    actual_decorator = user_passes_test(
        lambda u: u.is_authenticated,
        login_url=login_url,
        redirect_field_name=redirect_field_name
    )
    if function:
        return actual_decorator(function)
    return actual_decorator


[docs]def permission_required(perm, login_url=None, raise_exception=False):
    """
    Decorator for views that checks whether a user has a particular permission
    enabled, redirecting to the log-in page if necessary.
    If the raise_exception parameter is given the PermissionDenied exception
    is raised.
    """
    def check_perms(user):
        if isinstance(perm, str):
            perms = (perm,)
        else:
            perms = perm
        # First check if the user has the permission (even anon users)
        if user.has_perms(perms):
            return True
```

```
            # In case the 403 handler should be called raise the exception
            if raise_exception:
                raise PermissionDenied
            # As the last resort, show the login form
            return False
        return user_passes_test(check_perms, login_url=login_url)
```

Asynchronous messaging in Python –

```
from django.contrib import messages
from async_messages import get_messages
class AsyncMiddleware(object):


    def process_response(self, request, response):
        """
        Check for messages for this user and, if it exists,
        call the messages API with it
        """
        if hasattr(request, "session") and hasattr(request, "user") and
request.user.is_authenticated():
            msgs = get_messages(request.user)
            if msgs:
                for msg, level in msgs:
                    messages.add_message(request, level, msg)
        return response
```

Caching a View in Django –

```
from django.views.decorators.cache import cache_page

@cache_page(60 * 15)

def viewArticles(request, year, month):

  text = "Displaying articles of : %s/%s"%(year, month)

  return HttpResponse(text)
```