

Django

Table des matières

I.	IDE	3
II.	Python	4
1.	Installation	4
2.	Invites Python.....	5
	Avoir de l'aide.....	5
3.	Variables	6
4.	Listes.....	7
5.	Dictionnaire	7
6.	Conditions	8
7.	Boucles	8
8.	Fonction	9
9.	Programmation Objet en Python	10
10.	Modules.....	11
a.	Créer ses propres modules et packages	11
b.	Utiliser des modules existants.....	12
c.	Distribute.....	12
11.	Bases de données	13
a.	MySQL.....	13
b.	Redis	13
III.	Django	14
1.	Installation	14
2.	Créer un projet	14
3.	Tester le site	15
4.	Configuration « settings.py ».....	15
a.	Moteur de templates.....	15
b.	Bases de données	16
5.	Création d'application	16
6.	Routes	17
7.	« views.py ».....	18
8.	Templates ou « Gabarits »	19
a.	Templates.....	19
a.	Faire des liens	19

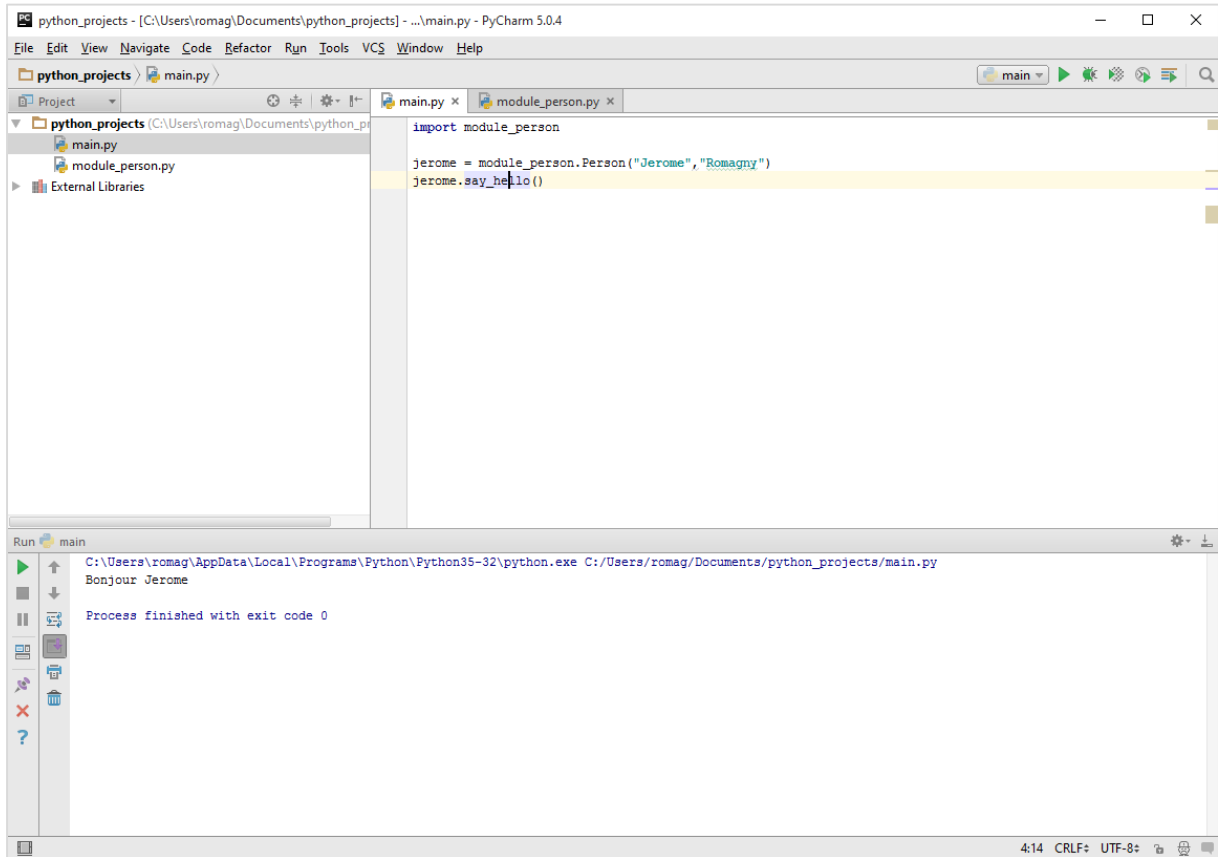
b. « static »	20
9. Modèles de l'application	20
a. Création d'un modèle	20
b. Relations	21
10. Formulaires	22
a. Création d'un formulaire	22
b. « views.py » de l'application	22
c. Afficher un formulaire dans un template	23
11. CRUD	25
12. Messages	26
13. Administration	27
a. Créer un « superuser » (administrateur)	27
b. Accéder à l'administration	27
c. Gérer des modèles depuis l'administration	29
14. Authentification	30
a. Autorisation	30
b. Templates personnalisés pour la connexion	30
c. Connexion avec les réseaux sociaux (Facebook, Google, etc.)	35

Python le langage

Django : **Framework MVT** (Model View Template) pour le **développement Web**

I. IDE

PyCharm de JetBrains



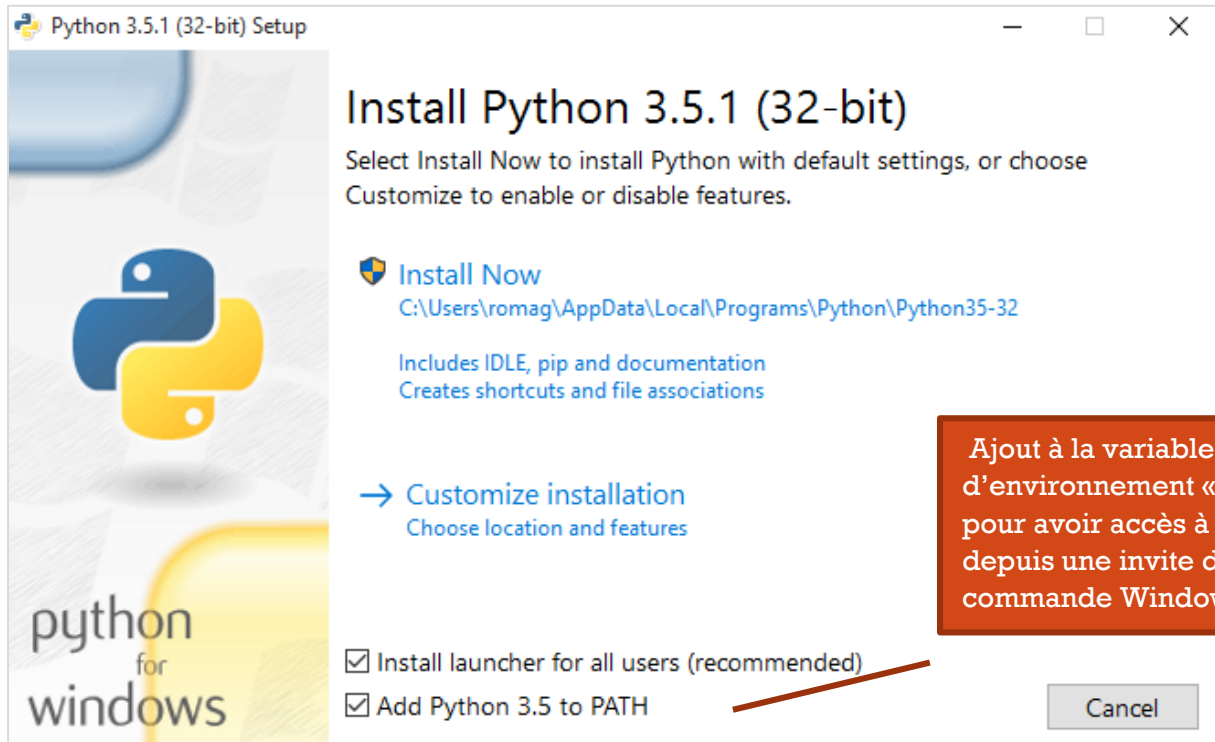
II. Python

Documentation

1. Installation

Installer la dernière version de [Python](#)

Exemple pour **Windows**



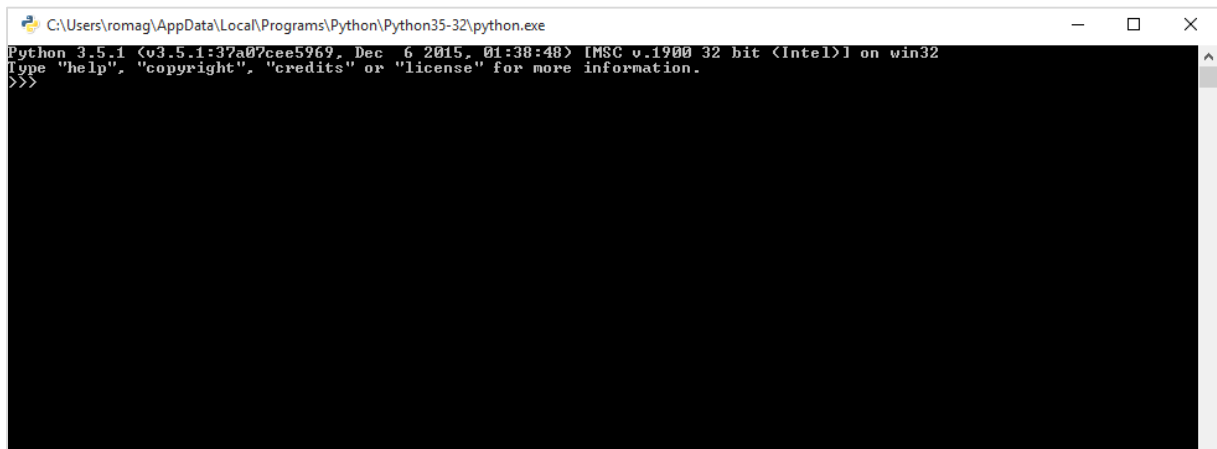
Ajout à la variable d'environnement « PATH » pour avoir accès à python depuis une invite de commande Windows

Tester si l'installation s'est bien passée : ouvrir une invite de commande « cmd » et taper «python », la version de Python devrait s'afficher.

De plus un menu « **Python** » avec des raccourcis est ajouté à « **tous les programmes** »

2. Invites Python

Soit entrer « **python** » dans le menu « **Exécuter** » de Windows, soit cliquer sur le raccourci « Python 3.5 » du menu démarrer



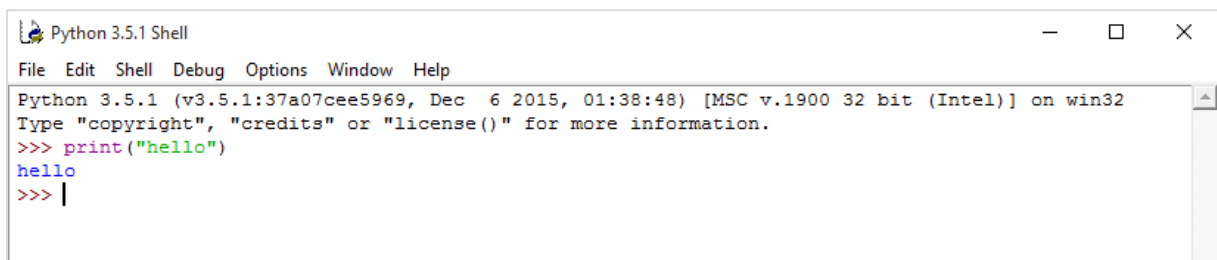
```

C:\Users\romag\AppData\Local\Programs\Python\Python35-32\python.exe
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>

```

Shell

Cliquer sur le raccourci **IDLE** du menu démarrer



```

Python 3.5.1 Shell
File Edit Shell Debug Options Window Help
Python 3.5.1 (v3.5.1:37a07cee5969, Dec 6 2015, 01:38:48) [MSC v.1900 32 bit (Intel)] on win32
Type "copyright", "credits" or "license()" for more information.
>>> print("hello")
hello
>>> |

```

Avoir de l'aide

Taper « `help()` »

Puis dans un second temps le terme recherché

3. Variables

Nombre

```
myint = 10
```

On peut utiliser « += », « -= » etc. en raccourci

```
myint +=20
myint -=5
```

Float

```
myfloat=2.99
```

Chaine de caractère

```
mystring = 'Attention aux \' apostrophes!'
```

Concaténation de chaines avec +

```
firstname = 'Jerome'
message = 'Bonjour ' + firstname
```

Autre possibilité avec « % »

```
firstname = 'Jerome'
message = 'Bonjour %s' % firstname
```

Booléen

```
mybool = True
```

Afficher la **valeur** d'une variable avec « **print()** ». Exemple print(myint)

Afficher le **type** de la variable avec « **type()** ». Exemple type(myint)

```
>>> myint=10
>>> print(myint)
10
>>> type(myint)
<class 'int'>
>>> |
```

Commentaires

Sur une ligne avec #

```
>>> myint=10 #un commentaire
```

Sur plusieurs lignes

```
"""
Commentaire
"""
```

4. Listes

Création d'une liste

```
mylist = ["un", "deux", "trois"]
```

Obtention des membres utilisables avec « dir() »

```
--> mylist = ["un", "deux", "trois"]
--> dir(mylist)
['_add_', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Ajout d'un élément en fin de liste avec « append »

```
mylist.append("quatre")
```

Insertion d'un élément avec « insert ».

```
mylist.insert(0, "zéro")
```

Accès à un membre par son index

```
mylist[1]="2!"
```

Suppression avec « remove »

```
mylist.remove("zéro")
```

Pile avec « pop »

```
>>> mylist.pop()
'quatre'
>>> print(mylist)
['un', 'deux', 'trois']
```

5. Dictionnaire

Création

```
mydic={"key1":"value1", "key2":"value2"}
```

Ajout d'un élément

```
mydic["key3"]="value3"
```

Accès à un membre et modification

```
mydic["key2"]="Nouvelle valeur"
```

Suppression d'un élément

```
del(mydic["key3"])
```

Membres disponibles : clear, copy, fromkeys, get, items, keys, pop, popitem, setdefault, update, values

6. Conditions

Ne pas oublier les « : » , et faire un retour arrière pour ramener « else » en début de ligne

```
mybool = True
if mybool==True:
    print("True!")
else:
    print("False!")
```

Avec **IDLE**

```
>>> mybool=True
>>> if mybool==True:
        print("True!")
else:
        print("False!")
```

```
True!
>>> |
```

Note pour else if employer « **elif** »

7. Boucles

Sur **liste**

```
mylist = ["un", "deux", "trois"]
for elem in mylist:
    print(elem)
```

Avec **IDLE**

```
>>> mylist = ["un", "deux", "trois"]
>>> for elem in mylist:
        print(elem)
```

```
un
deux
trois
```

Sur **dictionnaire**

```
>>> mydic={"key1":"value1", "key2":"value2"}
>>> for value in mydic.values():
        print(value)
```

```
value2
value1
```

On pourrait faire aussi une boucle sur les **clés** avec « **mydic.keys()** »

Boucle sur les **items**

```
>>> for elem in mydic.items():
    print(elem)
```

```
('key2', 'value2')
('key1', 'value1')
```

Ou encore

```
>>> for k,v in mydic.items():
    print("Clé:" + k + " / valeur:" + v)
```

```
Clé:key2 / valeur:value2
Clé:key1 / valeur:value1
```

8. Fonction

```
def say_hello(firstname):
    print("Bonjour " + firstname)

say_hello("Marie")
```

Avec **IDLE**

```
>>> def say_hello(firstname):
    print("Bonjour " + firstname)
```

Création de la
fonction

```
>>> say_hello("Marie")
Bonjour Marie
>>> |
```

Utilisation

Autre exemple avec return

```
>>> def get_message(firstname):
    return "Bonjour " + firstname
```

```
>>> result=get_message("Jerome")
>>> print(result)
Bonjour Jerome
```

9. Programmation Objet en Python

Création d'une classe

- Menu « File »... « New File »
- Saisie du script

```
class Person:
```

```
    def __init__(self,firstname,lastname):
        self.firstname = firstname
        self.lastname = lastname
```

Constructeur

```
    def say_hello(self):
        print("Bonjour " + self.firstname)
```

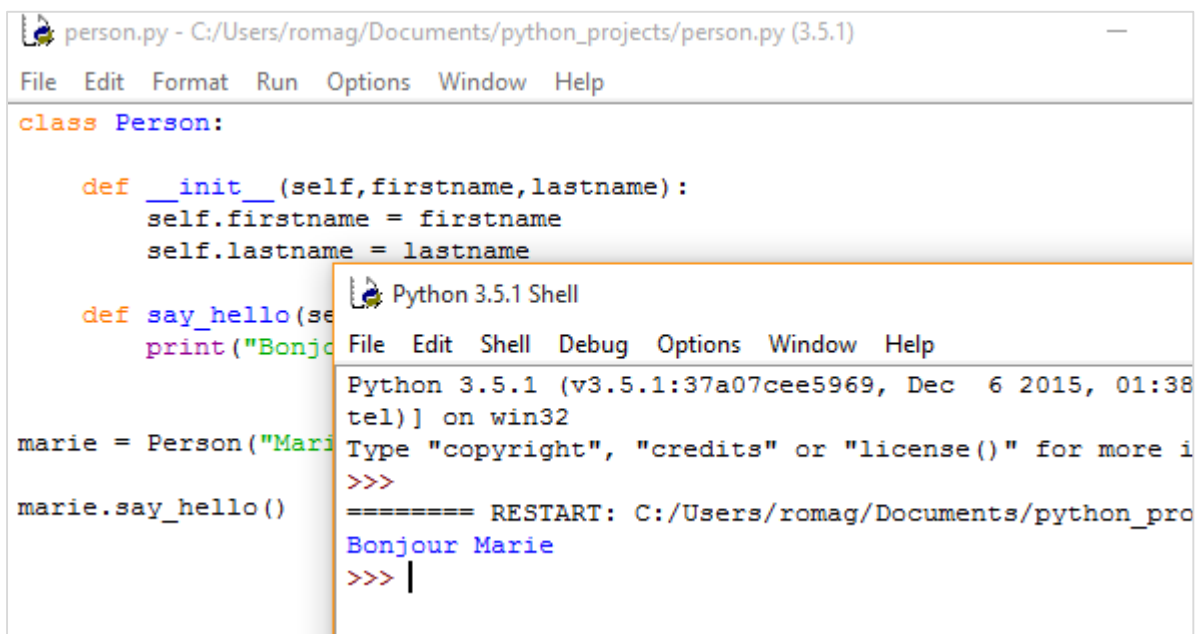
Une fonction

Paramètre « obligatoire »
« self »

```
marie = Person("Marie", "Bellin")
marie.say_hello()
```

Instance de la classe et
exécution de la méthode

- Sauvegarde du fichier au format « .py »
- Exécution avec « Run Module » (du menu « Run » de la nouvelle fenêtre) ou F5



Héritage : création d'une classe « Member » (dans le même fichier) qui hérite de « Person »

```
class Member(Person):
    pass #pas de membre, on passe c'est juste une demo
```

Exceptions

```
try:
    #code
except ValueError:
    print("Valeur invalide")
except:
    print("Erreur")
finally:
    #code
```

10. Modules

a. Créer ses propres modules et packages

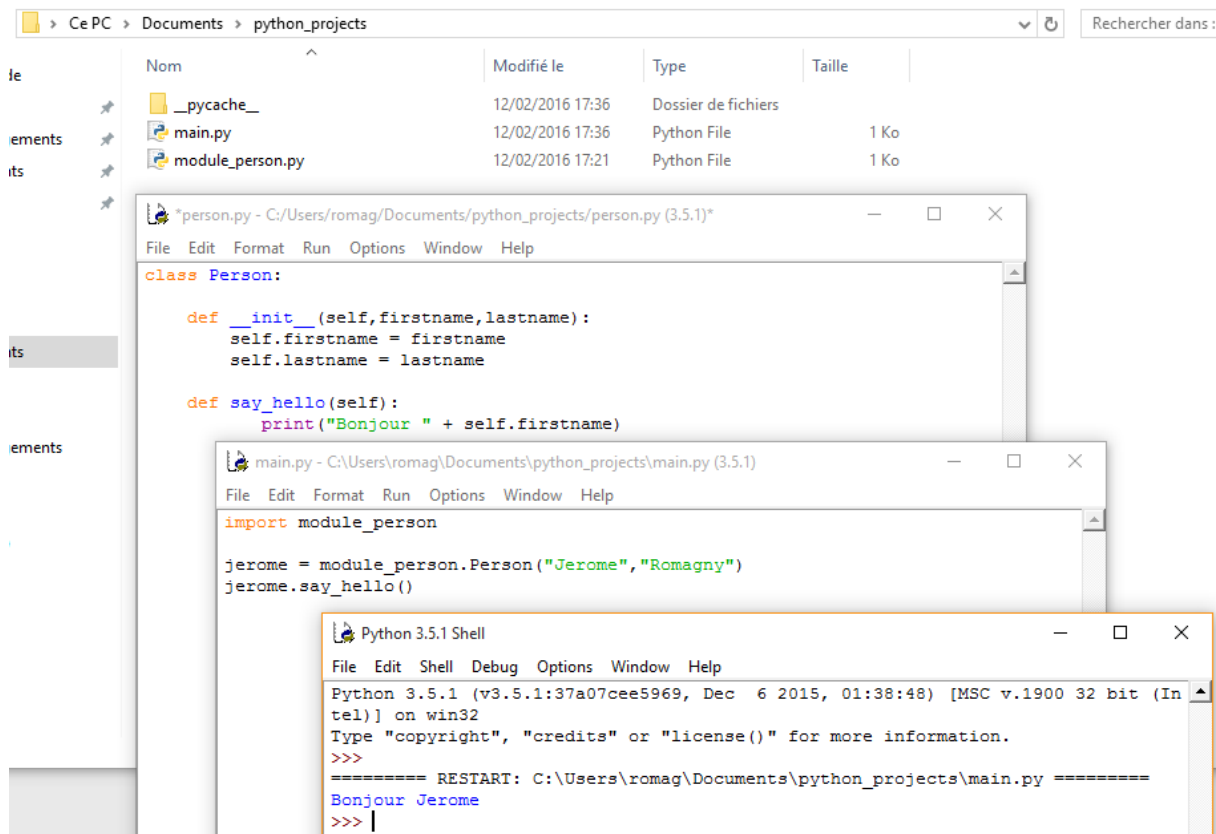
Exemple :

- On sauvegarde la classe dans un dossier avec un nom « module_demo.py »
- On crée un second fichier que l'on sauvegarde dans le même dossier et on importe le module : le nom correspond au nom du fichier

```
import module_person
```

Pour utiliser la classe on fait

```
jerome = module_person.Person("Jerome", "Romagny")
```



Dans un sous dossier, il faudrait indiquer le chemin. Par exemple si le module « module_person » était sauvegardé dans un sous dossier « mes_modules » (un « package »), pour l'importer on ferait

```
import my_package.module_person
```

Ou

```
from my_package import module_person
```

b. Utiliser des modules existants

Exemples

Programmation système (dossiers, fichiers,...)

Référence

```
import os
```

Pour les **expressions régulières**

Référence

```
Import re
```

c. Distribute

Documentation

Pour installer facilement des packages.

Installation

Télécharger la dernière version

Ensuite depuis une invite de commande, naviguer jusqu'au dossier contenant l'archive dézippé

```
python setup.py install
```

Installation des packages dans

```
C:\Users\[user] \AppData\Local\Programs\Python\Python35-32\Lib\site-packages
```

Installation de package

Avec « **easy_install** » depuis une invite de commande

```
easy_install <nom_du_package>
```

Puis il suffit de faire un import dans son fichier python pour l'utiliser

```
import <nom_du_package>
```

11. Bases de données

a. MySQL

Se connecter à une base de données MySQL

Télécharger et installer le [connector pour python](#)

Puis il suffit d'importer le package dans ses fichiers Python

```
import mysql.connector
```

[Documentation](#)

b. Redis

[Documentation](#)

III. Django

1. Installation

- a. Installer la dernière version de [Python](#)
- b. Installation de Django

[Documentation](#)

Exemple sur **Windows**

Pip est normalement installé avec Python mais il faut l'**upgrade**

```
python -m pip install --upgrade pip
```

Installation de **Django**

```
pip install django
```

Si on veut créer un environnement virtuel pour son projet

Installation de **virtualenv**

[Documentation](#)

Depuis une invite de commande, naviguer jusqu'au dossier du projet puis

```
pip install virtualenv
```

Puis naviguer jusqu'au dossier où créer le projet Django et taper la commande

```
virtualenv django_demo
```

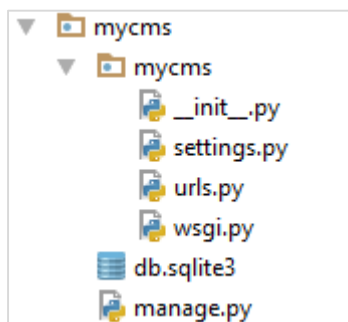
2. Créer un projet

Depuis une invite de commande naviguer jusqu'au dossier du projet

```
django-admin startproject mycms
```

Pour utiliser le dossier courant exister ajouter un « . »

```
django-admin startproject mycms .
```



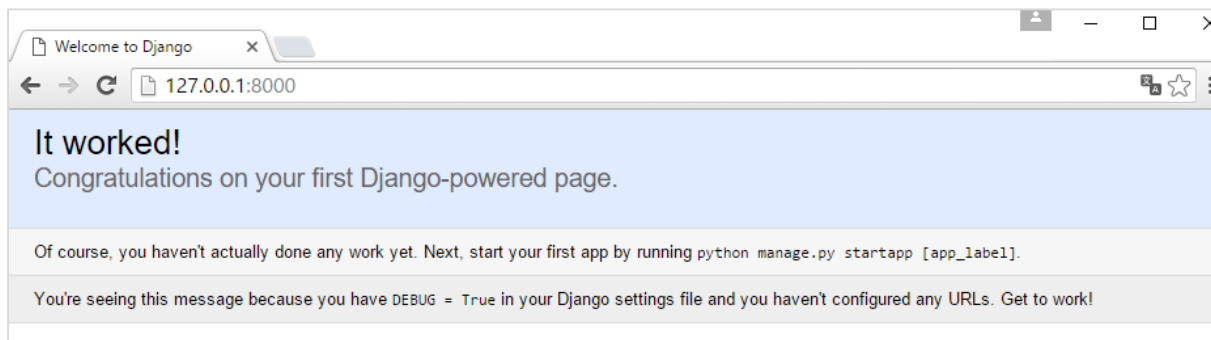
3. Tester le site

Lancer un serveur avec `manage.py`

(Sur le port 8000)

```
manage.py runserver
```

« `http://127.0.0.1:8000/` » Ou « `http://localhost:8000/` »



Il est possible d'indiquer le port

```
manage.py runserver 127.0.0.1:8080
```

CTRL + C pour **arrêter** le serveur

4. Configuration « `settings.py` »

Le fichier « `settings.py` » sert à configurer le projet. Le dossier de base, les applications installées (« `INSTALLED_APPS` »), les middlewares, le moteur de templates, la base de données, etc.

a. Moteur de templates

[Documentation](#)

```
TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
```

2 moteurs de templates supportés de base :

- Django « `django.template.backends.django.DjangoTemplates` »
- [Jinja2](#) « `django.template.backends.jinja2.Jinja2` »

b. Bases de données

Documentation

Moteur de **base de données** par défaut : SQLite.

Avec d'autres moteurs de bases de données, il faudra renseigner le nom de la base de données, un user, un password.

Migration

La migration met à jour la base de données et ajoute par défaut les tables pour l'administration et l'authentification.

```
manage.py migrate
```

5. Création d'application

Un projet peut avoir plusieurs applications. Une « application » est en fait une sorte de « module ».

a. On crée une application

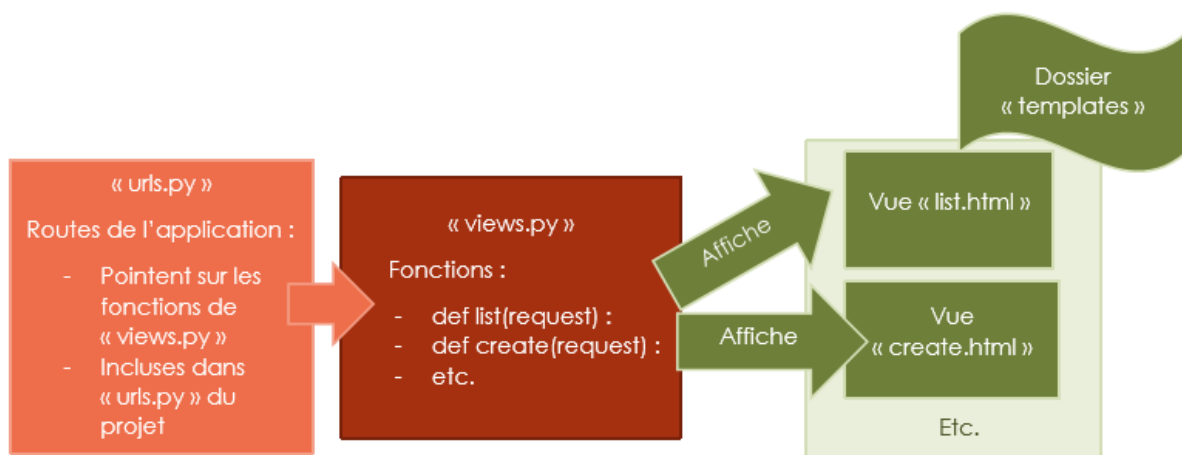
Exemple application nommée « articles »

```
manage.py startapp articles
```

b. Ajout à « settings.py » de la nouvelle application

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'articles'
]
```

Organisation



6. Routes

Toutes les routes sont définies dans le fichier « **urls.py** » du projet.

a. On définit les routes de l'application

Exemple pour l'application « articles », on ajoute à la racine un fichier « **urls.py** », on y définit les routes pour cette application

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.list),
    url(r'^view/([0-9]+)/$', views.view),
    url(r'^create/$', views.create),
    url(r'^edit/([0-9]+)/$', views.edit),
    url(r'^delete/([0-9]+)/$', views.delete)
]
```

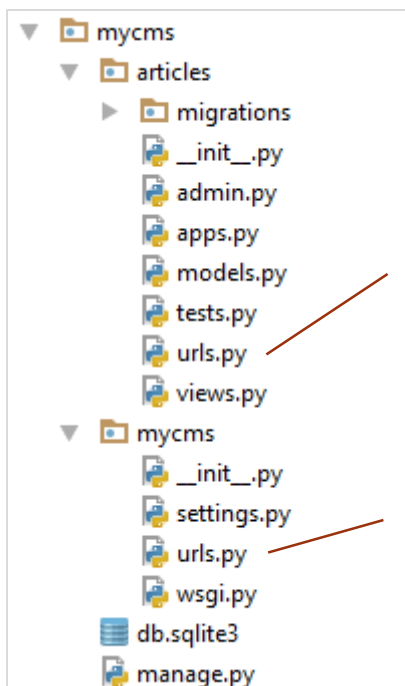
b. On inclut les routes de l'application dans « **urls.py** » du projet

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

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

Route de base,

Automatiquement « /articles/ » sera ajouté
en début des routes incluses pour
« articles.urls »



Ajout d'un fichier « **urls.py** » dans
l'application « articles » dans lequel on
définit les routes pour cette application

On inclut ensuite les routes dans « **urls.py** »
du projet.

7. « views.py »

C'est une sorte de « **contrôleur** » qui permet soit d'afficher un template, soit retourner une réponse http

Exemple réponse http

```
from django.http import HttpResponseRedirect

def index(request):
    return HttpResponseRedirect("Hello!")
```

Exemple affichage de template

```
from django.shortcuts import render

def home(request):
    return render(request, "home.html")
```

Récupération de **paramètre** passé dans la **route** et passage de paramètre au **template**.

Exemple

Dans « **urls.py** » de l'application on a une route acceptant un paramètre

```
url(r'^view/([0-9]+)/$', views.view),
```

Dans « **view.py** » de l'application

On récupère l'id dans la route et on passe l'article récupéré depuis la base de données au template

```
def view(request, id):
    article = get_object_or_404(Article, id=id)
    return render(request, "view.html", {'article': article})
```

8. Templates ou « Gabarits »

a. Templates

Créer un dossier « **templates** » dans l'**application** (exemple « articles ») et y ajouter les templates html

Création d'une **Master Page** « base.html »

```
<!DOCTYPE html>
{% load staticfiles %}
<html>
<head>
  <meta charset="UTF-8">
  <title>Django demo</title>
  <link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}" />
</head>
<body>
  {% block content %}{% endblock %}
</body>
</html>
```

Création d'un **template**

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

{% block content %}
<p>Le contenu.</p>
{% endblock %}
```

Variables définies entre accolades

```
{{ title }}
```

Boucles

```
{% if articles|length > 0%}
  {% for article in articles %}
    <article>
      <h2><a href="/articles/view/{{ article.id }}">{{ article.title}}</a></h2>
      <div>{{ article.content }}</div>
    </article>
  {% endfor %}
</div>
{% else %}
<div class="alert alert-warning text-center">
  Voulez-vous ajouter <a href="/articles/create">le premier article</a>?
</div>
{% endif %}
```

a. Faire des liens

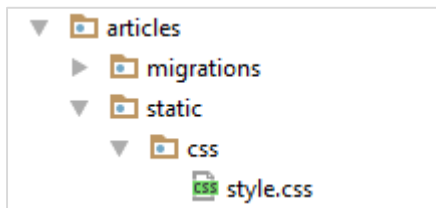
```
<a href="/articles/create">Ajouter un nouvel article</a>
<a href="/articles/edit/{{ article.id }}">Modifier</a>
<a href="/articles/delete/{{ article.id }}">Supprimer</a>
```

b. « static »

[Documentation](#)

Dans la Master Page « base.html ». Exemple la feuille de Styles de l'application « articles » sera chargée (relancer le serveur)

```
{% load staticfiles %}
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Django demo</title>
  <link rel="stylesheet" type="text/css" href="{% static 'css/style.css' %}" />
</head>
```



9. Modèles de l'application

[Documentation](#)

a. Création d'un modèle

Va servir à la fois de modèle pour la création de la table dans la base et pour la lecture, modification de données depuis l'application (ORM)

```
from django.db import models

class Article(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField(null=True)
    created = models.DateTimeField(auto_now_add=True, auto_now=False)

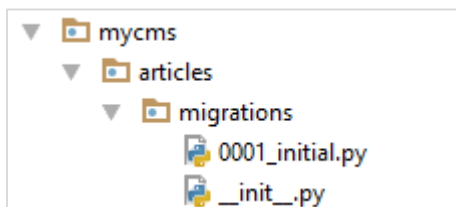
    def __str__(self):
        return self.title
```

Champ avec insertion automatique de la date

Création d'une migration

```
manage.py makemigrations articles
```

C'est un peu l'équivalent de « ToString » en .NET. C'est-à-dire que cette fonction retourne une chaîne de caractère pour la classe



Voir le SQL généré pour la future création de tables

```
manage.py sqlmigrate articles 0001
```

Création des tables dans la base de données (définie dans « settings.py »)

```
manage.py migrate
```

Pour voir sa base **SQLite**, plusieurs possibilités (extension pour Firefox, etc.), on peut aussi utiliser une tool comme [SQLiteStudio](#) ou [Datagrip](#) de JetBrains

b. Relations

Relations One to One, One to Many, Many to Many

Exemple un article a un « auteur » (en fait correspond à l'utilisateur connecté).

Dans le modèle

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

class Article(models.Model):
    title = models.CharField(max_length=100)
    content = models.TextField(null=True)
    created = models.DateTimeField(auto_now_add=True, auto_now=False)
    user = models.ForeignKey(User)

    def __str__(self):
        return self.title
```

Lors de l'ajout d'un article dans « views.py » de l'application « articles »

On affecte l'utilisateur connecté à l'utilisateur de l'article

```
def create(request):
    if request.user.is_authenticated():
        form = ArticleForm(request.POST or None)
        if form.is_valid():
            new_article = Article(
                title = form.cleaned_data['title'],
                content = form.cleaned_data['content'],
                user = request.user
            )
            new_article.save()
            messages.success(request, 'Article ajouté.', 'alert alert-success')

            return redirect('/articles/view/' + str(new_article.id))
        return render(request, "create.html", { 'form': form })
    else:
        return redirect('/auth/login/')
```

Faire une migration et une mise à jour de la table si besoin.

10. Formulaires

Documentation

a. Création d'un formulaire

Attribut widget

Ajout d'un fichier « forms.py » dans l'application et définition du formulaire.

```
from django import forms

from articles.models import Article

class ArticleForm(forms.ModelForm):
    class Meta:
        model = Article
        fields = ('title', 'content')
        title = forms.CharField(error_messages={'required': 'Vous devez renseigner un titre'}, label="Titre", max_length=100, widget=forms.TextInput(attrs={'placeholder': 'Saisissez le titre', 'class': 'form-control'}))
        content = forms.CharField(error_messages={'required': 'Vous devez ajouter un contenu'}, widget=forms.Textarea(attrs={'class': 'form-control'}), label="")
```

Formulaire lié au modèle « Article » :

- forms.ModelForm
- model
- fields

Définition des champs du formulaire :

- Input de type texte avec « forms.CharField »
- Textarea avec « forms.CharField(widget=forms.Textarea)

On peut également définir :

- le **label** pour chaque champ
- avec « **attrs** » :
 - o définir le « **placeholder** »
 - o des **classes CSS**
- Pour la **validation** de formulaire on peut définir des **messages personnalisés** selon l'erreur avec « **error_messages** »

b. « views.py » de l'application

Formulaire pour ajouter un nouvel élément en base de données

Les imports

```
from django.shortcuts import render, redirect, get_object_or_404

from articles.models import Article

from articles.forms import ArticleForm
```

Si on a besoin de récupérer l'id généré en base de données (pour un redirect vers la vue détails de l'article inséré par exemple)

```
def create(request):
    form= ArticleForm(request.POST or None)
    if form.is_valid():
        new_article = Article(
            title = form.cleaned_data['title'],
            content = form.cleaned_data['content']
        )
        new_article.save()
        return redirect('/articles/view/' + str(new_article.id))
    return render(request, "create.html", { 'form':form })
```

Sinon

```
def create(request):
    form= ArticleForm(request.POST or None)
    if form.is_valid():
        form.save()
        return redirect('/articles/')
    return render(request, "create.html", { 'form':form })
```

On peut tester si la requête est de méthode « POST » avec

```
if request.POST:
```

Formulaire pour modifier un élément existant

```
def edit(request, id):

    article = get_object_or_404(Article, id=id)
    form= ArticleForm(request.POST or None, instance=article)
    if form.is_valid():
        form.save()
        return redirect('/articles/view/' + str(article.id))
    return render(request, "edit.html", { 'form':form })
```

c. Afficher un formulaire dans un template

On peut faire très simple

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

{% block content %}
<h1>Modifier l'article</h1>

<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="Mettre à jour">
</form>

{% endblock %}
```

Il est possible également de **personnaliser** le code généré

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

{% block content %}

<section>
  <div class="page-header">
    <h1>Ajouter un nouvel article</h1>
  </div>
  <div class="col-md-12">
    <form method="post" class="form-horizontal">
      {% csrf_token %}
      {% for field in form %}
        <div class="fieldWrapper">
          {{ field.label_tag }}
          {{ field }}
          {{ field.errors }}
        </div>
      {% endfor %}
      <input type="submit" class="btn btn-default" value="Publier">
    </form>
  </div>
  <div class="text-danger">
    {{ form.title.errors }}
    {{ form.content.errors }}
  </div>
</section>

{% endblock %}
```

Affichage du label, puis du camp et enfin des erreurs du champ

Affichage des erreurs du formulaire

Affichage des erreurs

Django demo Accueil Blog

Ajouter un nouvel article

Titre:

Vous devez renseigner un titre

Vous devez ajouter un contenu

- Vous devez renseigner un titre
- Vous devez ajouter un contenu

11. CRUD

Exemple

Routes « urls.py » de l'application « articles »

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^$', views.list),
    url(r'^view/([0-9]+)/$', views.view),
    url(r'^create/$', views.create),
    url(r'^edit/([0-9]+)/$', views.edit),
    url(r'^delete/([0-9]+)/$', views.delete)
]
```

« views.py »

Utilisation du modèle et du formulaire vus précédemment

```
from django.shortcuts import render, redirect, get_object_or_404

from articles.models import Article

from articles.forms import ArticleForm

def list(request):
    articles = Article.objects.all()
    return render(request, "list.html", {'articles': articles})

def view(request, id):
    article = get_object_or_404(Article, id=id)
    return render(request, "view.html", {'article': article})

def create(request):
    form= ArticleForm(request.POST or None)
    if form.is_valid():
        new_article = Article(
            title = form.cleaned_data['title'],
            content = form.cleaned_data['content']
        )
        new_article.save()
        return redirect('/articles/view/' + str(new_article.id))
    return render(request, "create.html", { 'form':form })

def edit(request, id):
    article = get_object_or_404(Article, id=id)
    form= ArticleForm(request.POST or None, instance=article)
    if form.is_valid():
        form.save()
        return redirect('/articles/view/' + str(article.id))
    return render(request, "edit.html", { 'form':form })

def delete(request, id):
    article = get_object_or_404(Article, id=id)
    article.delete()
    return redirect('/articles/')

```

Liste

Django demo Accueil Blog

Articles

[Ajouter un nouvel article](#)

Premier article

Posté le 14 Feb 2016

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec scelerisque, justo in dignissim tempor, sem felis ultrices augue, ac fringilla est elit eget lacus. Suspendisse aliquet convallis augue, vitae eleifend nisl viverra sit amet. Duis non quam urna. Nunc ullamcorper dolor id tortor congue, vitae molestie nisl dictum. Maecenas commodo bibendum tortor vitae ornare. Quisque at volutpat neque, id hendrerit ligula.

Second article

Posté le 14 Feb 2016

Sed condimentum ex eget metus pharetra, ut condimentum augue faucibus. Morbi augue mi, interdum et porta at, mollis ac ipsum. Integer ut risus lobortis, aliquet turpis quis, varius ipsum. Nunc ut justo accumsan, vestibulum dolor et, facilisis mauris. Praesent aliquet neque ullamcorper odio euismod elementum. In sagittis felis erat, eget commodo mi aliquam laoreet. Nunc mollis convallis dui, a luctus mi luctus vitae. Vivamus pellentesque neque id odio bibendum dignissim. Donec ut odio nibh.

Détails

Premier article

Posté le 14 Feb 2016

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec scelerisque, justo in dignissim tempor, sem felis ultrices augue, ac fringilla est elit eget lacus. Suspendisse aliquet convallis augue, vitae eleifend nisl viverra sit amet. Duis non quam urna. Nunc ullamcorper dolor id tortor congue, vitae molestie nisl dictum. Maecenas commodo bibendum tortor vitae ornare. Quisque at volutpat neque, id hendrerit ligula.

[Modifier](#) [Supprimer](#)

Formulaires ajout et modification

Django demo Accueil Blog

Modifier l'article

Titre:

Premier article

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec scelerisque, justo in dignissim tempor, sem felis ultrices augue, ac fringilla est elit eget lacus. Suspendisse aliquet convallis augue, vitae eleifend nisl viverra sit amet. Duis non quam urna. Nunc ullamcorper dolor id tortor congue, vitae molestie nisl dictum. Maecenas commodo bibendum tortor vitae ornare. Quisque at volutpat neque, id hendrerit ligula.

[Mettre à jour](#)

Ou si aucun article

Django demo Accueil Blog

Articles

Voulez-vous ajouter le premier article?

12. Messages

Documentation

Exemple dans une fonction de « views.py » .Ne pas oublier l'import

```
from django.contrib import messages
```

```
def delete(request, id):
    article = get_object_or_404(Article, id=id)
    article.delete()
    messages.success(request, 'Article supprimé.', 'alert alert-success')
    return redirect('/articles/')

```

Fonctions de messages : « success », « warning », « error », « info », « debug ».

Dans la Master Page « base.html », on affiche les messages

```
{% if messages %}
    {% for message in messages %}
        <div {% if message.tags %} class="{{ message.tags }}" {% endif %}>{{ message }}</div>
    {% endfor %}
{% endif %}

```

Django demo Accueil Blog

Article ajouté.

Exemple de message

13. Administration

Documentation

a. Créer un « superuser » (administrateur)

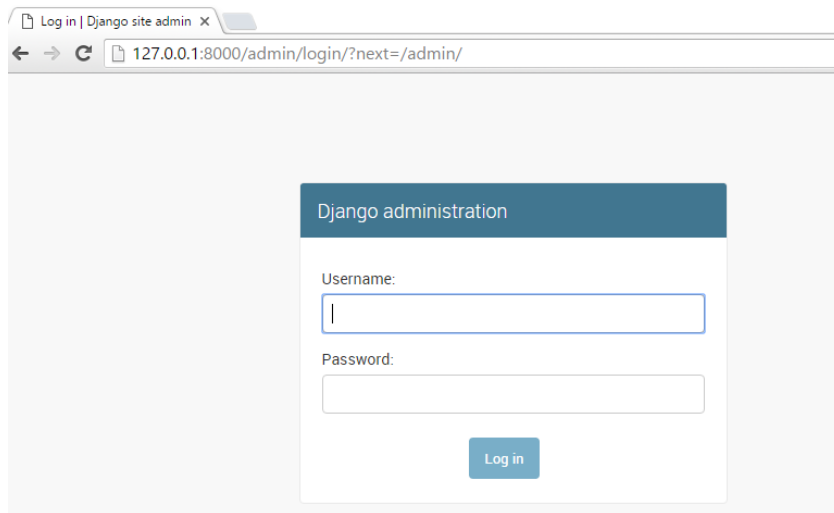
Depuis une invite de commande

```
manage.py createsuperuser
```

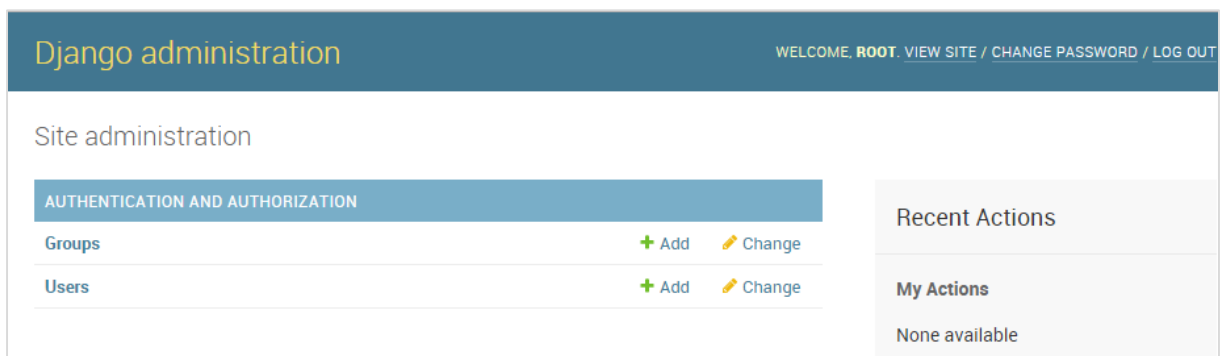
Donner un username, un email et un password.

b. Accéder à l'administration

Se rendre à la route «/admin/ » (cela pourra être « http://127.0.0.1:8000/admin/») et se connecter.



De base on peut gérer les utilisateurs (ajout, modification, suppression), les groupes et permissions



Ajout d'un utilisateur

The screenshot shows the 'Add user' form in the Django administration interface. The header bar is dark blue with 'Django administration' on the left and 'WELCOME, ROOT. VIEW SITE / CHANGE PASSWORD / LOG OUT' on the right. Below the header, a breadcrumb trail reads 'Home > Authentication and Authorization > Users > Add user'. The main content area is white and titled 'Add user'. It contains a message: 'First, enter a username and password. Then, you'll be able to edit more user options.' There are three input fields: 'Username:' with the value 'marie', 'Password:', and 'Password confirmation:'. Below the 'Username' field is a small note: 'Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.' Below the 'Password' and 'Password confirmation' fields is a note: 'Enter the same password as before, for verification.' At the bottom right, there are three buttons: 'Save and add another', 'Save and continue editing', and 'SAVE'.

... « save and continue editing ». On va pouvoir définir les permissions de l'utilisateur :

- « active »
- « **Staff status** » autorise l'utilisateur à se connecter à l'interface d'admin. On peut **sélectionner un par un ses permissions** dans la partie « User permissions »
- « **Superuser status** » donne automatiquement **toutes les permissions** à l'utilisateur

The screenshot shows the 'Permissions' section of the Django administration interface. It has a blue header bar with the title 'Permissions'. Below the header, there are three sections, each with a checkbox and a description: 1. 'Active' with a checked checkbox and the description 'Designates whether this user should be treated as active. Unselect this instead of deleting accounts.' 2. 'Staff status' with a checked checkbox and the description 'Designates whether the user can log into this admin site.' 3. 'Superuser status' with an unchecked checkbox and the description 'Designates that this user has all permissions without explicitly assigning them.'

c. Gérer des modèles depuis l'administration

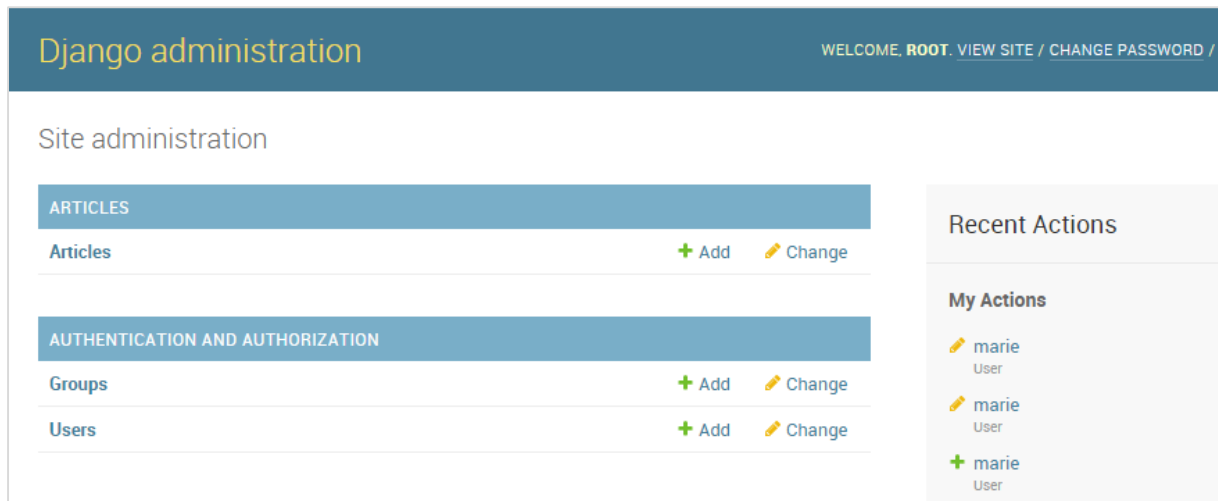
Dans « admin.py » de l'application, enregistrer des modèles administrables

```
from django.contrib import admin

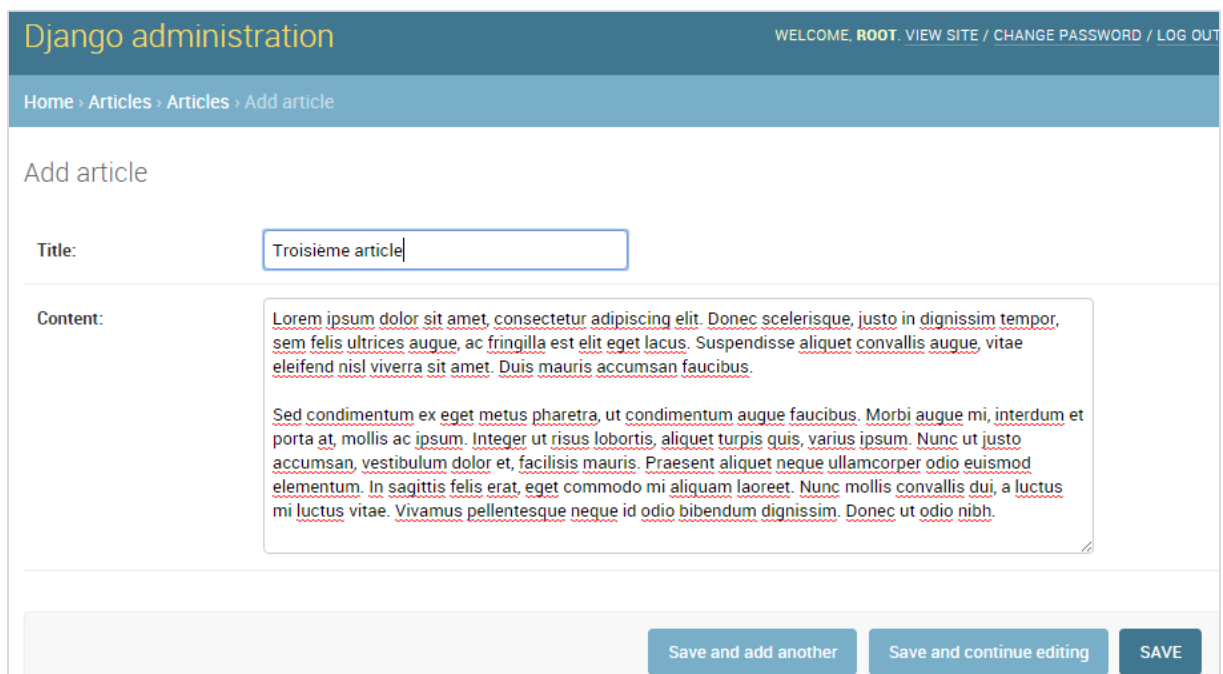
from articles.models import Article

admin.site.register(Article)
```

Il est désormais possible d'ajouter, modifier, supprimer des articles depuis l'interface d'administration



Exemple ajout d'un article



On a également la liste des articles.

14. Authentification

Documentation

a. Autorisation

Sécuriser une route

Vérifier si un utilisateur est authentifié, sinon on le redirige vers la page de « login »

```
def create(request):
    if request.user.is_authenticated():
        # etc.
        return render(request, "create.html", { 'form': form })
    else:
        return redirect('/admin/login/')
```

On peut utiliser la page pour se connecter avec « /admin/login/ »

N'afficher les boutons modifier/ supprimer (dans la vue « Détails ») que si l'utilisateur connecté correspond à l'utilisateur ayant écrit l'article

```
{%if user.is_authenticated and user.id == article.user.id %}
<div class="pull-right">
  <a href="/articles/edit/{{ article.id }}" class="btn btn-default">Modifier</a>
  <a href="/articles/delete/{{ article.id }}" class="btn btn-primary">Supprimer</a>
</div>
{% endif %}
```

b. Templates personnalisés pour la connexion

Exemple on crée une nouvelle application nommée « users » dédiée à l'authentification

Routes à l'application (ajout d'un fichier « urls.py » à « users »)

```
from django.conf.urls import url

from . import views

urlpatterns = [
    url(r'^login/$', views.auth_login),
    url(r'^register/$', views.auth_register),
    url(r'^logout/$', views.auth_logout)
]
```

.. que l'on inclut dans « urls.py » du projet

```
urlpatterns = [
    url(r"'^$', include('core.urls')),
    url(r'^admin/', admin.site.urls),
    url(r'^auth/', include('users.urls')),
    url(r'^articles/', include('articles.urls'))
]
```

Création des **formulaires** de connexion et d'inscription.

Ce sont des formulaires non liés à des modèles.

```
from django import forms

class RegisterForm(forms.Form):
    username = forms.CharField(error_messages={'required': 'Vous devez renseigner un
username'}, label="Username", max_length=100, widget=forms.TextInput(attrs={'placeholder':
'Username', 'class' : 'form-control'}))
    email = forms.CharField(error_messages={'required': 'Vous devez renseigner un
email'}, label="Email", max_length=100, widget=forms.EmailInput(attrs={'placeholder':
'Email', 'class' : 'form-control'}))
    password = forms.CharField(error_messages={'required': 'Vous devez renseigner un mot
de
passe'}, label="Password", max_length=100, widget=forms.PasswordInput(attrs={'placeholder':
'Mot de passe', 'class' : 'form-control'}))
    confirmpassword = forms.CharField(error_messages={'required': 'Vous devez confirmer le
mot de
passe'}, label="Password", max_length=100, widget=forms.PasswordInput(attrs={'placeholder':
'Confirmer le mot de passe', 'class' : 'form-control'}))

    def clean_password(self):
        if self.data['password'] != self.data['confirmpassword']:
            raise forms.ValidationError('Les mots de passe ne correspondent pas')
        return self.data['password']

    def clean(self, *args, **kwargs):
        self.clean_password()
        return super(RegisterForm, self).clean(*args, **kwargs)

class LoginForm(forms.Form):
    username = forms.CharField(error_messages={'required': 'Vous devez renseigner un
username'}, label="Username", max_length=100, widget=forms.TextInput(attrs={'placeholder':
'Username', 'class' : 'form-control'}))
    password = forms.CharField(error_messages={'required': 'Vous devez renseigner un mot
de
passe'}, label="Password", max_length=100, widget=forms.PasswordInput(attrs={'placeholder':
'Password', 'class' : 'form-control'}))
```

Vérification que les
mots de passe
correspondent

« views.py » de « users »

```

from django.contrib.auth import authenticate, login, logout
from django.shortcuts import render, redirect
from django.contrib import messages

from django.contrib.auth.models import User

from users.forms import RegisterForm, LoginForm

def auth_login(request):
    form= LoginForm(request.POST or None)
    if form.is_valid():
        username = form.cleaned_data["username"]
        password = form.cleaned_data["password"]
        user = authenticate(username=username, password=password)
        if user:
            login(request, user)
            messages.success(request, 'Vous êtes connecté!', 'alert alert-success')
            return redirect("/")
        else:
            return redirect('/auth/login/')
    return render(request, "login.html", { 'form': form })

def auth_register(request):
    form= RegisterForm(request.POST or None)
    if form.is_valid():
        username = form.cleaned_data["username"]
        email = form.cleaned_data["email"]
        password = form.cleaned_data["password"]
        User.objects.create_user(username, email, password)
        messages.success(request, 'Vous avez bien été enregistré!', 'alert alert-
success')
        return redirect('/')
    return render(request, "register.html", { 'form': form })

def auth_logout(request):
    logout(request)
    return redirect('/auth/login/')

```

On récupère les informations de formulaire, puis on utilise les fonctions « authenticate » puis « login » pour connecter l'utilisateur

On récupère les informations de formulaire, puis on utilise la fonction « create_user » pour enregistrer un nouvel utilisateur

Templates (dossier un dossier « templates » de « users »)

« register.html »

On peut personnaliser le formulaire, mais pour la clarté je donne le code le plus simple

```

{% extends "base.html" %}

{% block content %}
<h1>Inscription</h1>

<form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <input type="submit" value="S'inscrire">
</form>

{% endblock %}

```


The screenshot shows a web browser window with the address bar displaying "127.0.0.1:8000/auth/register/". The page has a dark header with "Django demo" and navigation links "Accueil" and "Blog". On the right side of the header are links "S'inscrire" and "Se connecter". The main content area is titled "Inscription" and contains a registration form with the following fields: "Username", "Email", "Mot de passe", and "Confirmer le mot de passe". Each field has a corresponding input box. At the bottom of the form, there is a blue button labeled "S'inscrire" and a link "ou Se connecter".

« login.html »

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

{% block content %}
<div class="col-md-offset-4 col-md-4">
  <h1>Connexion</h1>
  <hr>
  <form method="post">
    {% csrf_token %}
    {{ form.as_p }}
    <div class="text-center form-group">
      <button type="submit" class="btn btn-primary">Se
connecter</button>
      &nbsp; ou &nbsp;
      <a href="/auth/register">S'inscrire</a>
    </div>
  </form>
</div>
{% endblock %}
```

The screenshot shows the Django login page titled "Connexion". It features a form with two input fields: "Username:" containing the text "patrick" and "Password:" with masked characters ".....". Below the password field is a blue button labeled "Se connecter" and a link "ou S'inscrire". The page has the same dark header as the registration page, with "Django demo" and navigation links "Accueil" and "Blog". On the right side of the header are links "S'inscrire" and "Se connecter".

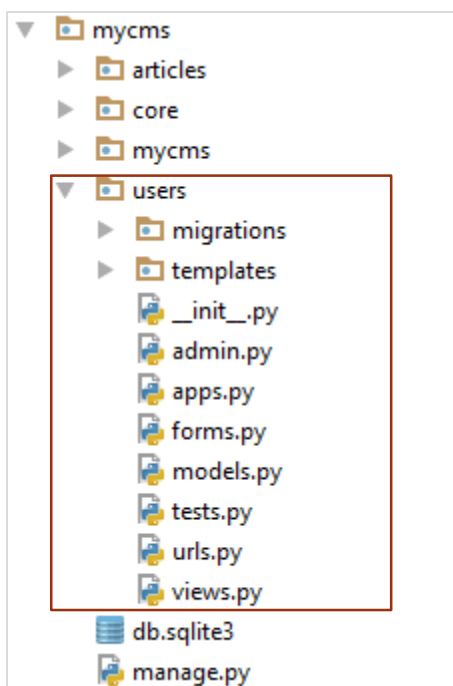
Modification du **Menu** selon que l'utilisateur est connecté ou non. Dans la Master Page
« **base.html** »

```
<div class="navbar-inverse">
  <div class="container">
    <div class="navbar-header">
      <button type="button" class="navbar-toggle" data-toggle="collapse" data-
target=".navbar-collapse">
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
        <span class="icon-bar"></span>
      </button>
      <a href="#" class="navbar-brand">Django demo</a>
    </div>
    <div class="navbar-collapse collapse">
      <ul class="nav navbar-nav">
        <li><a href="/">Accueil</a></li>
        <li><a href="/articles/">Blog</a></li>
      </ul>
      {%if user.is_authenticated %}
      <ul class="nav navbar-nav navbar-right">
        <li><a href="#">Bonjour {{ user.username }}</a></li>
        <li class="divider-vertical"></li>
        <li><a href="/auth/logout">Se déconnecter</a></li>
      </ul>
      {% else %}
      <ul class="nav navbar-nav navbar-right">
        <li><a href="/auth/register">S'inscrire</a></li>
        <li class="divider-vertical"></li>
        <li><a href="/auth/login/">Se connecter</a></li>
      </ul>
      {% endif %}
    </div>
  </div>
</div>
```

Utilisateur connecté

Utilisateur non
connecté

L'application « users »



c. Connexion avec les réseaux sociaux (Facebook, Google, etc.)

Documentation

Installation, Configuration Django Framework, ...

Installation de « python-social-auth »

```
pip install python-social-auth
```

Faire une migration pour mettre à jour la base de données

```
manage.py migrate
```

Configuration « settings.py »

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    social.apps.django_app.default,
    'core',
    'users',
    'articles'
]
```

```
AUTHENTICATION_BACKENDS = (
    'social.backends.facebook.FacebookOAuth2',
    'social.backends.google.GoogleOAuth2',
    'django.contrib.auth.backends.ModelBackend',
)
```

Facebook, Google

```
TEMPLATE_CONTEXT_PROCESSORS = (
    'social.apps.django_app.context_processors.backends',
    'social.apps.django_app.context_processors.login_redirect',
)
```

```
SOCIAL_AUTH_URL_NAMESPACE = 'social'
SOCIAL_AUTH_LOGIN_REDIRECT_URL = '/'
```

Facebook

Création d'une application Facebook

- Ajouter une plateforme « Website » et définir l'url sur <http://localhost:8000/> par exemple
- Dans onglet « App review » passer l'application en « public »

The screenshot shows the Facebook Developer console interface. On the left is a sidebar with navigation links: Dashboard, Settings (highlighted), App Review, App Details, Roles, Open Graph, Alerts, and Localize. The main content area has three tabs: Basic, Advanced, and Migrations. The 'Basic' tab is active, displaying the following fields:

- App ID:** 204062339946126
- App Secret:** Masked with dots, with a 'Show' button.
- Display Name:** djangolocal
- Namespace:** Empty text field.
- App Domains:** Empty text field.
- Contact Email:** romagny13@yahoo.fr

Below these fields is a 'Website' section with a 'Quick Start' button and a 'Site URL' field containing <http://localhost:8000/>.

```
SOCIAL_AUTH_FACEBOOK_SCOPE = ['email']
SOCIAL_AUTH_FACEBOOK_PROFILE_EXTRA_PARAMS = {
    'locale': 'fr_FR',
    'fields': 'id, name, email'
}
SOCIAL_AUTH_FACEBOOK_KEY = '2040623...456789'
SOCIAL_AUTH_FACEBOOK_SECRET = '01234...bcdefg'
```

Google+

Créer un [projet Google](#)

- Donner un « nom de produit »
- Activer l'API « Google + »
- Identifiants : « ID client OAuth » ..
 - o En origine autorisée « <http://localhost:8000> »
 - o En URI de redirection autorisée « <http://localhost:8000/complete/google-oauth2/> »

API	Gestionnaire d'API	Identifiants						
<div> <div>Présentation</div> <div>Identifiants</div> </div>		<div> <div>← Télécharger JSON Réinitialiser le code secret Supprimer</div> <div>ID client pour Application Web</div> <table border="1"> <tr> <td>ID client</td> <td>854695551565-9cvh2vh4aquvr0uefd3la6jbo3ssmuui.apps.googleusercontent.com</td> </tr> <tr> <td>Code secret du client</td> <td>23tFag-yj...oSldqg7de</td> </tr> <tr> <td>Date de création</td> <td>14 févr. 2016 à 23:09:15</td> </tr> </table> <div>Nom</div> <div>Client Web 1</div> <div>Restrictions</div> <p>Saisir les origines JavaScript, rediriger les URI ou effectuer les deux actions</p> <div>Origines JavaScript autorisées</div> <p>À utiliser pour les requêtes effectuées depuis un navigateur. Il s'agit de l'URI d'origine de l'application cliente. Ne peut pas contenir de caractère générique (http://*.example.com) ni de chemin (http://example.com/subdir).</p> <div> http://localhost:8000 </div> <div> http://www.example.com </div> <div>URI de redirection autorisés</div> <p>À utiliser pour les requêtes effectuées depuis un serveur Web. Il s'agit du chemin de votre application vers lequel les utilisateurs sont redirigés après s'être authentifiés avec Google. Le code d'autorisation d'accès sera ajouté au chemin. Il doit être associé à un protocole. Ne peut pas contenir de fragments d'URL ni de chemins relatifs. Ne peut pas être une adresse IP publique.</p> <div> http://localhost:8000/complete/google-oauth2/ </div> </div>	ID client	854695551565-9cvh2vh4aquvr0uefd3la6jbo3ssmuui.apps.googleusercontent.com	Code secret du client	23tFag-yj...oSldqg7de	Date de création	14 févr. 2016 à 23:09:15
ID client	854695551565-9cvh2vh4aquvr0uefd3la6jbo3ssmuui.apps.googleusercontent.com							
Code secret du client	23tFag-yj...oSldqg7de							
Date de création	14 févr. 2016 à 23:09:15							

```
SOCIAL_AUTH_GOOGLE_OAUTH2_KEY = '854695...apps.googleusercontent.com'
SOCIAL_AUTH_GOOGLE_OAUTH2_SECRET = '23tFag-yj...oSldqg7de'
```

Routes « *urls.py* » du projet

```
urlpatterns = [
    url(r'^$', include('core.urls')),
    url(r'^admin/', admin.site.urls),
    url(r'^auth/', include('users.urls')),
    url(r'^articles/', include('articles.urls')),
    url(r'', include('social.apps.django_app.urls', namespace='social'))
]
```

Liens de connexion

Dans le template « login.html » par exemple

```
<a href="{% url 'social:begin' 'facebook' %}">Facebook</a>
<a href="{% url 'social:begin' 'google-oauth2' %}">Google+</a>
```

On peut mettre en forme ces liens avec [bootstrap-social](#) par exemple.

```
<a class="btn btn-block btn-social btn-facebook" href="{% url
'social:begin' 'facebook' %}">
  <span class="fa fa-facebook"></span>
  Facebook
</a>
<a class="btn btn-block btn-social btn-google" href="{% url
'social:begin' 'google-oauth2' %}">
  <span class="fa fa-google"></span>
  Google+
</a>
```