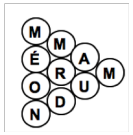
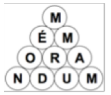
Python - Introduction

1. Historique et principes
2. Environnement de développement
3. Introduction à Python : syntaxe
    - Python avancé
4. Introduction à Scipy (Pandas)
5. Introduction à Matplotlib
    - Autres librairies graphiques
6. Introduction à Scikit-Learn
7. Préparation de données
8. DASK
9. Création d'une API Rest Django
10. Test Driven Development en Django
11. Gestion des environnements
12. Annexes





Création d'une API pour le boston dataset

```
print(boston["DESCR"])
```

```
Boston House Prices dataset

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000 sq.ft.
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's
```

Introduction à Django

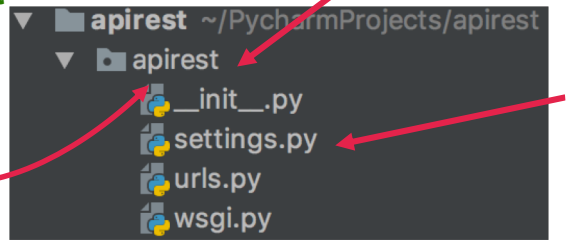
```
pip install django
pip install djangorestframework
pip install pygments

django-admin.py startproject apirest
```

1) Créer un répertoire pour le projet

2) Créer un sous-répertoire Du même nom

2.1) Dossier de paramètres du projet

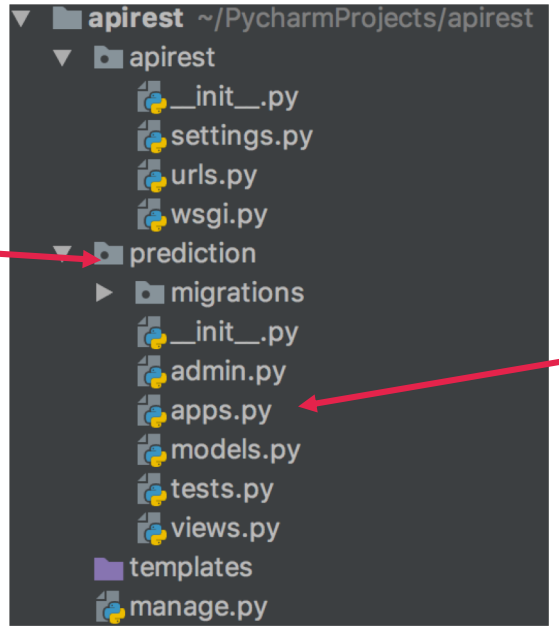
2.2) Fichier de paramètres globaux



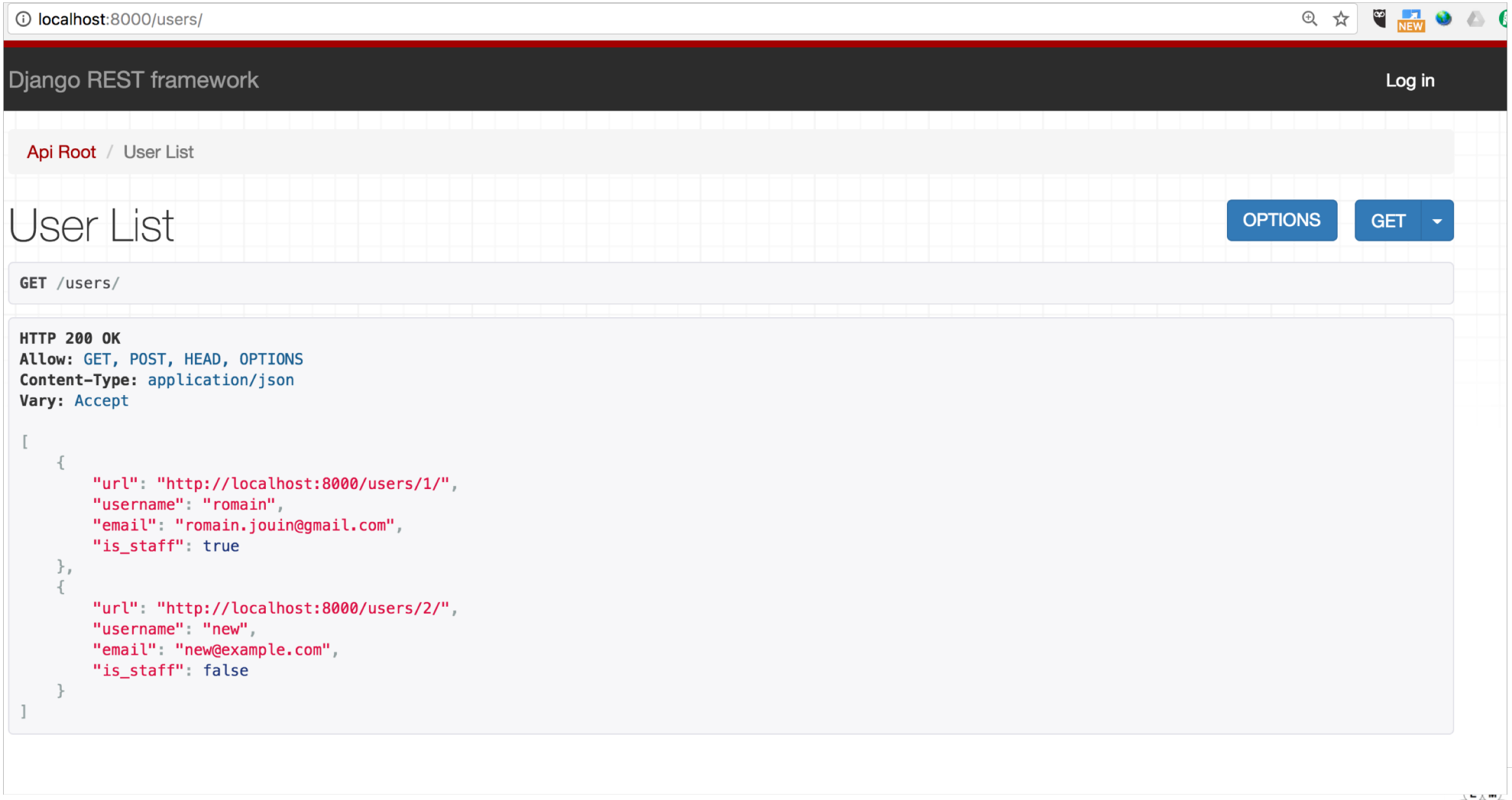

```
cd apirest
python manage.py startapp prediction
```

3) Créer un répertoire pour l'app « prédiction »

3.1) Fichier de paramétrage local de l'app



Django Rest Framework



Django Rest Framework

The API guide is your complete reference manual to all the functionality provided by REST framework.

Requests
Responses
Views
Generic views
Viewsets
Routers
Parsers
Renderers
Serializers
Serializer fields
Serializer relations
Validators
Authentication
Permissions
Throttling
Filtering
Pagination
Versioning

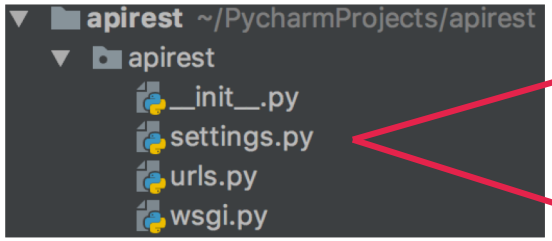
Content negotiation
Metadata
Schemas
Format suffixes
Returning URLs
Exceptions
Status codes
Testing
Settings

Serializers

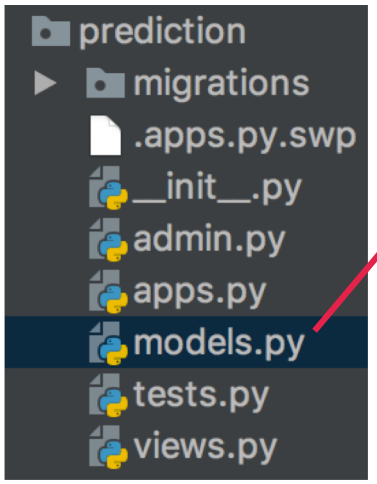
> *Expanding the usefulness of the serializers is something that we would like to address. However, it's not a trivial problem, and it will take some serious design work.*
>
> *— Russell Keith-Magee, Django users group*

Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into `JSON`, `XML` or other content types. Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.

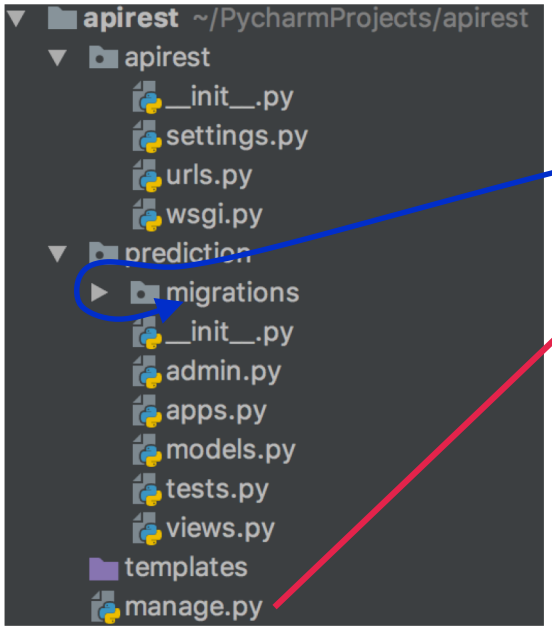
The serializers in REST framework work very similarly to Django's `Form` and `ModelForm` classes. We provide a `Serializer` class which gives you a powerful, generic way to control the output of your responses, as well as a `ModelSerializer` class which provides a useful shortcut for creating serializers that deal with model instances and querysets.




```
INSTALLED_APPS = [
'django.contrib.admin'          ,
'django.contrib.auth'           ,
'django.contrib.contenttypes'   ,
'django.contrib.sessions'       ,
'django.contrib.messages'       ,
'django.contrib.staticfiles'    ,
'rest_framework'                ,
'prediction.apps.PredictionConfig' ,
]
```

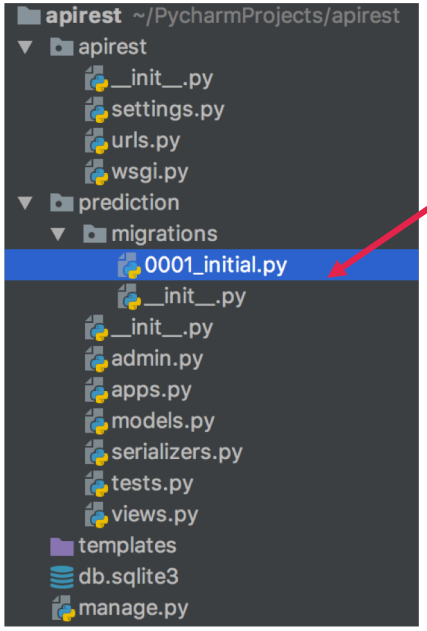



```
from django.db       import models
class House(models.Model):
    CRIM    = models.FloatField()
    ZN      = models.FloatField()
    INDUS   = models.FloatField()
    CHAS    = models.FloatField()
    NOX     = models.FloatField()
    RM      = models.FloatField()
    AGE     = models.FloatField()
    DIS     = models.FloatField()
    RAD     = models.FloatField()
    TAX     = models.FloatField()
    PTRATIO = models.FloatField()
    B       = models.FloatField()
    LSTAT   = models.FloatField()
    MEDV    = models.FloatField(null=True)
    created = models.DateTimeField(auto_now_add=True)

    class Meta:
        ordering = ['created']
```

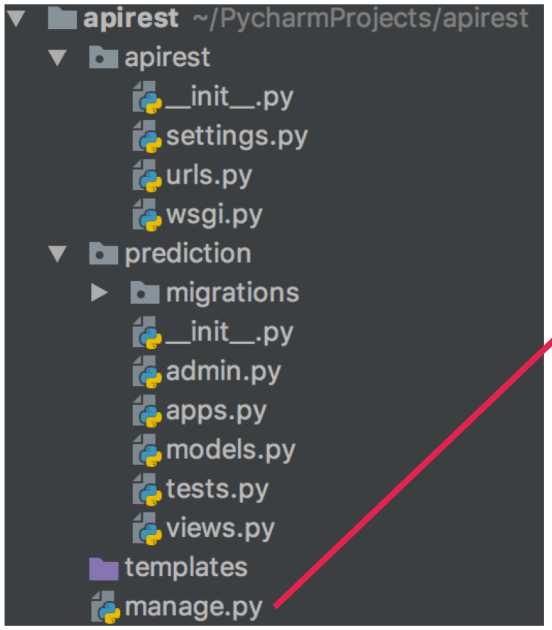


Permet d'agir sur tout le projet
Permet de préparer la création d'une base de donnée :
python manage.py makemigrations prediction

```
python manage.py makemigrations prediction
Migrations for 'prediction':
  prediction/migrations/0001_initial.py
    - Create model House
```



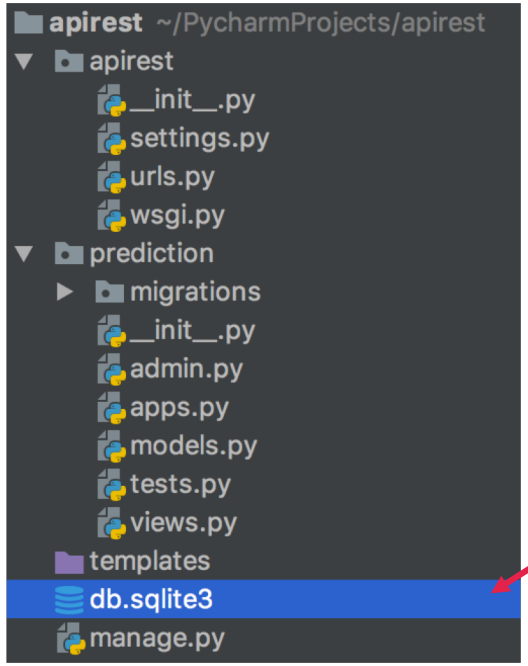
La liste des instructions de migration a été créé dans des fichiers dans le dossier migrations



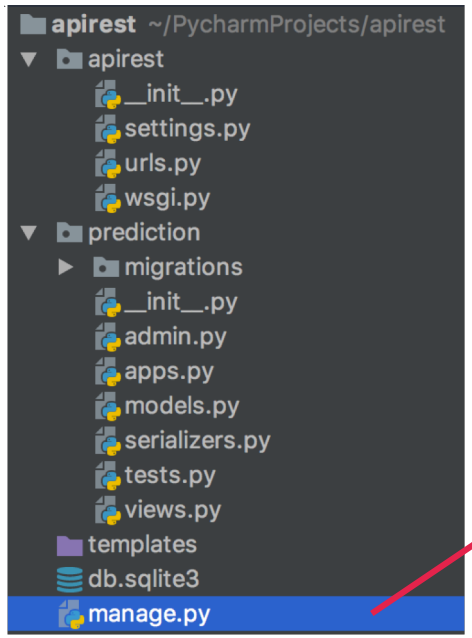
Permet d'agir sur tout le projet
Permet de créer une base de donnée :
python manage.py migrate

```
python manage.py migrate
```

```
Operations to perform:
  Apply all migrations: admin, auth, contenttypes, prediction, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying prediction.0001_initial... OK
  Applying sessions.0001_initial... OK
```



Une base de donnée à été créée (en SQLlite par défaut)



Permet :

- **D'agir sur tout le projet (startapp)**
- **De créer une base de donnée (makemigrations / migrate)**
- **De tester en ligne de commande**
  - **python manage.py shell**

```
python manage.py shell
Python 3.5.3 |Anaconda 4.4.0 (x86_64)| (default, Mar  6 2017, 12:15:08)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]:
```

```
from prediction.models          import House
house = House( CRIM      = 0.00632 ,
        ZN        = 18.0   ,
        INDUS     = 2.31   ,
        CHAS      = 0.0    ,
        NOX       = 0.538  ,
        RM        = 6.575  ,
        AGE       = 65.2   ,
        DIS       = 4.09   ,
        RAD       = 1.0    ,
        TAX       = 296.0  ,
        PTRATIO   = 15.3   ,
        B     = 396.9  ,
        LSTAT     = 4.98   )
house.save()
```

Copier un texte d'un fichier dans le shell
=> **%paste**

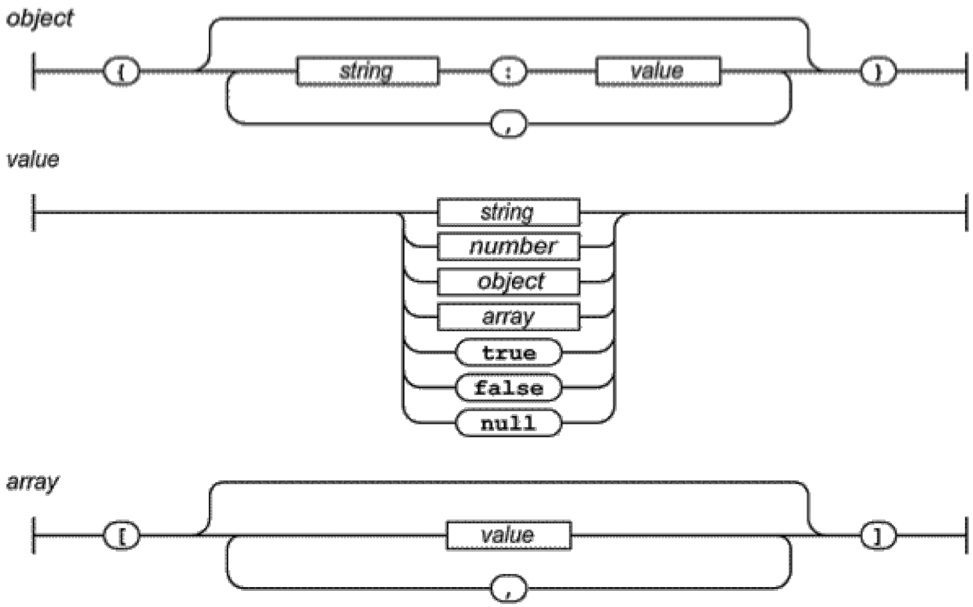
```
from prediction.models          import House
house = House( CRIM      = 0.00632 ,
        ZN        = 18.0   ,
        INDUS     = 2.31   ,
        CHAS      = 0.0    ,
        NOX       = 0.538  ,
        RM        = 6.575  ,
        AGE       = 65.2   ,
        DIS       = 4.09   ,
        RAD       = 1.0    ,
        TAX       = 296.0  ,
        PTRATIO   = 15.3   ,
        B     = 396.9  ,
        LSTAT     = 4.98   )
house.save()
```

Copier un texte d'un fichier dans le shell
=> **%paste**

```
Python 3.5.3 |Anaconda 4.4.0 (x86_64)| (default, Mar  6 2017, 12:15:08)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: %paste
from prediction.models          import House
from prediction.serializers     import HouseSerializer
from rest_framework.parsers     import JSONParser
from rest_framework.renderers   import JSONRenderer

house = House(  CRIM      = 0.00632 ,
                ZN        = 18.0    ,
                INDUS     = 2.31    ,
                CHAS      = 0.0     ,
                NOX       = 0.538   ,
                RM        = 6.575   ,
                AGE       = 65.2    ,
                DIS       = 4.09    ,
                RAD       = 1.0     ,
                TAX       = 296.0   ,
                PTRATIO   = 15.3    ,
                B         = 396.9   ,
                LSTAT     = 4.98    ,
                #MEDV     =
)
house.save()
## -- End pasted text --
```

Formalisme JSON :

clef : valeur

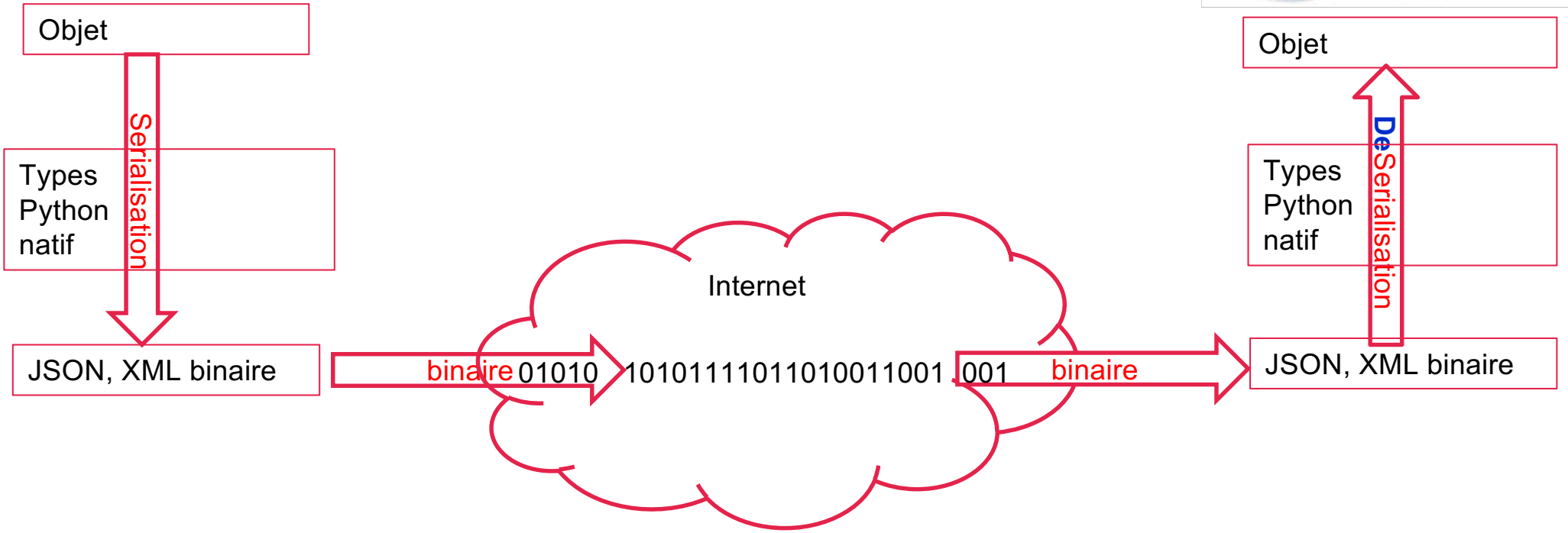



```
{
   « nom »        :   « jouin »                ,
   « prénom »     :   « romain »               ,
   « adulte »     :   true                     ,
   « adresse »    :   { « ville » : « paris » ,
                        « cp »    : « 75013 »}

}
```

Sérialisation et désérialisation

Sérialiser	**Convertir un objet en chaine d'octets**	Permet le stockage ou le transfert via le réseau
Désérialiser	**Convertir une chaine d'octets en objet**	Permet de reconstituer l'objet reçu / stocké
Persistence	**Enregistrer une chaine d'octets sur un disque**	Permet de redémarrer un programme dans un état sauvé
Marshalling	**Envoyer une chaine d'octets sur un réseau**	Permet de distribuer des objets entre des PC
Caching	**Positionner un objet en mémoire**	Permet d'accéder rapidement à la donnée
Partionning	**Diviser une matrice entre *n* machine**	Permet d'optimiser l'usage des ressource

Formalisme JSON :



Formalisme JSON :

querysets
model instances

Serialisation

JSON, XML or other content types

```
house = House(  CRIM     = 0.00632 ,
                ZN       = 18.0    ,
                INDUS    = 2.31    ,
                CHAS     = 0.0     ,
                NOX      = 0.538   ,
                RM       = 6.575   ,
                AGE      = 65.2    ,
                DIS      = 4.09    ,
                RAD      = 1.0     ,
                TAX      = 296.0   ,
                PTRATIO  = 15.3    ,
                B        = 396.9   ,
                LSTAT    = 4.98    ,
                #MEDV     =
)
house.save()
```

Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes

native Python datatypes

```
In [4]: %paste
serializer = HouseSerializer(house)
serializer.data

## -- End pasted text --
Out[4]:
ReturnDict([('CRIM', 0.02731),
            ('ZN', 0.0),
            ('INDUS', 7.07),
            ('CHAS', 0.0),
            ('NOX', 0.469),
            ('RM', 6.421),
            ('AGE', 78.9),
            ('DIS', 4.9671),
            ('RAD', 2.0),
            ('TAX', 242.0),
            ('PTRATIO', 17.8),
            ('B', 396.9),
            ('LSTAT', 9.14),
            ('MEDV', None)])
```

JSONRenderer

```
return
JsonResponse(serializer.dat
a  , status=201)
```

bytes = b'...'
a sequence of octets (integers between 0 and 255)

Formalisme JSON :

querysets
model instances

DeSerialisation

JSON, XML or other content types

```
house.save()
```

```
def create(self, validated_data):
    """
    Create and return a new `House` instance, given the validated data.
    """
    return House.objects.create(**validated_data)
```

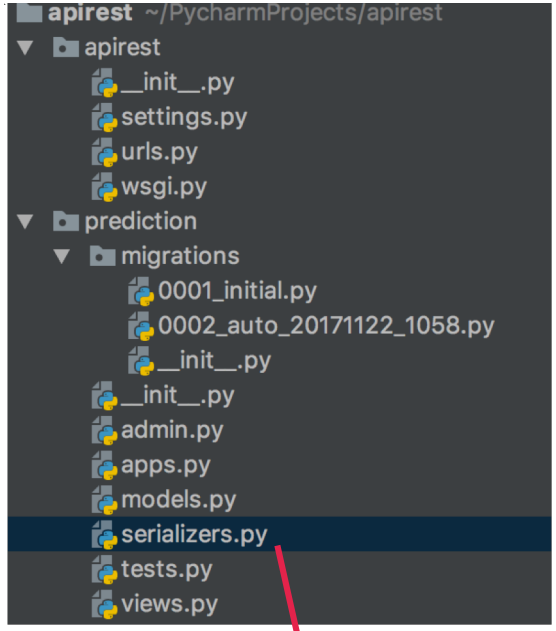
bytes = b'...'
a sequence of octets (integers between 0 and 255)

native Python datatypes

Serializers also provide deserialization, allowing parsed data to be converted back into complex types, after first validating the incoming data.

parse and validate data

```
class HouseSerializer(serializers.Serializer):
    CRIM    = serializers.FloatField()
    ZN      = serializers.FloatField()
    INDUS   = serializers.FloatField()
    CHAS    = serializers.FloatField()
    NOX     = serializers.FloatField()
    RM      = serializers.FloatField()
    AGE     = serializers.FloatField()
    DIS     = serializers.FloatField()
    RAD     = serializers.FloatField()
    TAX     = serializers.FloatField()
    PTRATIO = serializers.FloatField()
    B       = serializers.FloatField()
    LSTAT   = serializers.FloatField()
    MEDV    = serializers.FloatField(allow_null=True)
```



On crée un fichier serializers.py qui contiendra des classes pour transformer des objets python en objets JSON:

```python
from rest_framework     import serializers
from prediction.models  import House

class HouseSerializer(serializers.Serializer):
    CRIM    = serializers.FloatField()
    ZN      = serializers.FloatField()
    INDUS   = serializers.FloatField()
    CHAS    = serializers.FloatField()
    NOX     = serializers.FloatField()
    RM      = serializers.FloatField()
    AGE     = serializers.FloatField()
    DIS     = serializers.FloatField()
    RAD     = serializers.FloatField()
    TAX     = serializers.FloatField()
    PTRATIO = serializers.FloatField()
    B       = serializers.FloatField()
    LSTAT   = serializers.FloatField()
    MEDV    = serializers.FloatField(allow_null=True)

    def create(self, validated_data):
    """Create and return a new `House` instance, given the validated data."""
        return House.objects.create(**validated_data)

    def update(self, instance, validated_data):
    """Update and return an existing `House` instance, given the validated data."""
        instance.CRIM       = validated_data.get('CRIM'     , instance.CRIM     )
        instance.ZN         = validated_data.get('ZN'       , instance.ZN       )
        instance.INDUS      = validated_data.get('INDUS'    , instance.INDUS    )
        instance.CHAS       = validated_data.get('CHAS'     , instance.CHAS     )
        instance.NOX        = validated_data.get('NOX'      , instance.NOX      )
        instance.RM         = validated_data.get('RM'       , instance.RM       )
        instance.AGE        = validated_data.get('AGE'      , instance.AGE      )
        instance.DIS        = validated_data.get('DIS'      , instance.DIS      )
        instance.RAD        = validated_data.get('RAD'      , instance.RAD      )
        instance.TAX        = validated_data.get('TAX'      , instance.TAX      )
        instance.PTRATIO    = validated_data.get('PTRATIO'  , instance.PTRATIO  )
        instance.B          = validated_data.get('B'        , instance.B        )
        instance.LSTAT      = validated_data.get('LSTAT'    , instance.LSTAT    )
        #instance.MEDV      = validated_data.get('MEDV'     , instance.MEDV     )
        instance.save()
        return instance
```

La classe serializers fournie par le package API REST transforme des objets python en objets JSON:

```
from prediction.models          import House
house = House( CRIM      = 0.00632 ,
        ZN        = 18.0   ,
        INDUS     = 2.31   ,
        CHAS      = 0.0    ,
        NOX       = 0.538  ,
        RM        = 6.575  ,
        AGE       = 65.2   ,
        DIS       = 4.09   ,
        RAD       = 1.0    ,
        TAX       = 296.0  ,
        PTRATIO   = 15.3   ,
        B     = 396.9  ,
        LSTAT     = 4.98   )
house.save()
```

Copier un texte d'un fichier dans le shell
=> **%paste**

```
Python 3.5.3 |Anaconda 4.4.0 (x86_64)| (default, Mar  6 2017, 12:15:08)
Type "copyright", "credits" or "license" for more information.

IPython 5.3.0 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra details.

In [1]: %paste
from prediction.models          import House
from prediction.serializers     import HouseSerializer
from rest_framework.parsers     import JSONParser
from rest_framework.renderers   import JSONRenderer

house = House(  CRIM      = 0.00632 ,
                ZN        = 18.0    ,
                INDUS     = 2.31    ,
                CHAS      = 0.0     ,
                NOX       = 0.538   ,
                RM        = 6.575   ,
                AGE       = 65.2    ,
                DIS       = 4.09    ,
                RAD       = 1.0     ,
                TAX       = 296.0   ,
                PTRATIO   = 15.3    ,
                B         = 396.9   ,
                LSTAT     = 4.98    ,
                #MEDV     =
)
house.save()
## -- End pasted text --
```

```
serializer = HouseSerializer(house)
serializer.data
```

Le **serializer** a transformé la maison en dictionnaire de donnée avec des tuypes de données python bruts.

```
In [4]: %paste
serializer = HouseSerializer(house)
serializer.data

## -- End pasted text --
Out[4]:
ReturnDict([('CRIM', 0.02731),
            ('ZN', 0.0),
            ('INDUS', 7.07),
            ('CHAS', 0.0),
            ('NOX', 0.469),
            ('RM', 6.421),
            ('AGE', 78.9),
            ('DIS', 4.9671),
            ('RAD', 2.0),
            ('TAX', 242.0),
            ('PTRATIO', 17.8),
            ('B', 396.9),
            ('LSTAT', 9.14),
            ('MEDV', None)])
```

```
content = JSONRenderer().render(serializer.data)
content
```

```
In [9]: %paste
content = JSONRenderer().render(serializer.data)
content

## -- End pasted text --
Out[9]: b'{"CRIM":0.02731,"ZN":0.0,"INDUS":7.07,"
CHAS":0.0,"NOX":0.469,"RM":6.421,"AGE":78.9,"DIS"
:4.9671,"RAD":2.0,"TAX":242.0,"PTRATIO":17.8,"B":
396.9,"LSTAT":9.14,"MEDV":null}'
```

Le **JSONRenderer** a transformé les types de données python bruts en format JSON.

```
from django.utils.six import BytesIO
stream = BytesIO(content)
data = JSONParser().parse(stream)
```

Le **BytesIO** a transformé le json en un dico de **types de données python bruts.**

```
In [10]: %paste
from django.utils.six import BytesIO

stream = BytesIO(content)
data = JSONParser().parse(stream)

## -- End pasted text --

In [11]: data
Out[11]:
{'AGE': 78.9,
 'B': 396.9,
 'CHAS': 0.0,
 'CRIM': 0.02731,
 'DIS': 4.9671,
 'INDUS': 7.07,
 'LSTAT': 9.14,
 'MEDV': None,
 'NOX': 0.469,
 'PTRATIO': 17.8,
 'RAD': 2.0,
 'RM': 6.421,
 'TAX': 242.0,
 'ZN': 0.0}
```

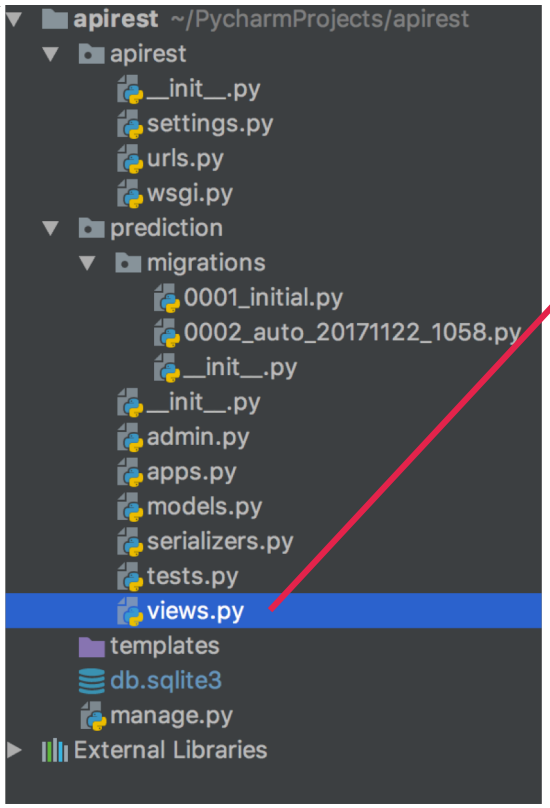
```
house = HouseSerializer(data=data)
```

```
def create(self, validated_data):
    """
    Create and return a new `House` instance, given the validated data.
    """
    return House.objects.create(**validated_data)
```

Le **HouseSerialiser** a créé une instance de la classe House. Ainsi on a tranformé le dico de data en instance de classe.

```
if house.is_valid():
    house.save()
```

```
In [9]: %paste
house = HouseSerializer(data=data)
if house.is_valid():
    house.save()

## -- End pasted text --

In [10]:
```



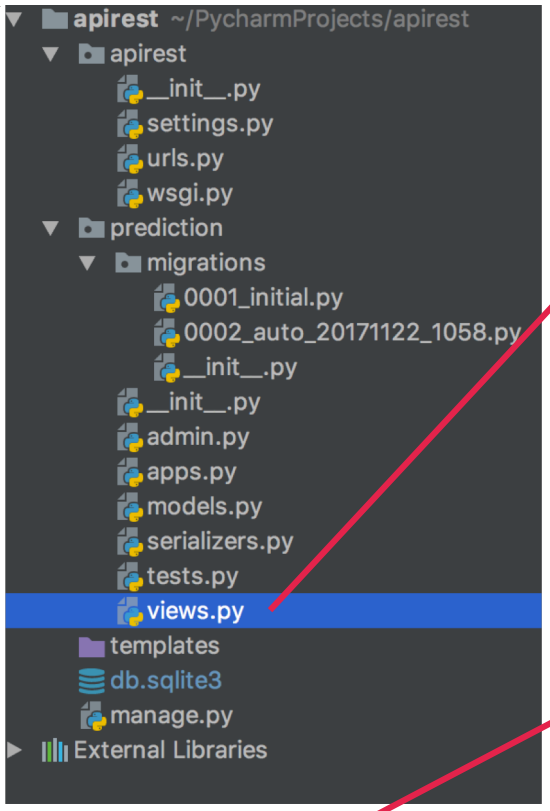
Le fichier views permet d'éxécuter un code sur l'appel d'une url au niveau du serveur. Les codes sont contenus dans des vues

```
from django.http                    import HttpResponse
from django.http                    import JsonResponse
from django.views.decorators.csrf   import csrf_exempt
from rest_framework.renderers       import JSONRenderer
from rest_framework.parsers         import JSONParser
from prediction.models              import House
from prediction.serializers         import HouseSerializer


@csrf_exempt
def house_list(request):

    if request.method == 'GET':
        houses   = House.objects.all()
        serializer  = HouseSerializer(houses, many=True)
        return JsonResponse(serializer.data, safe=False)

    elif request.method == 'POST':
        data        = JSONParser().parse(request)
        serializer  = HouseSerializer(data=data)
        if serializer.is_valid():
            serializer.save()
            return JsonResponse(serializer.data  , status=201)
        return     JsonResponse(serializer.errors, status=400)
```

La vue house_list permet de retourner toutes les maisons contenues dans la base sur une requete GET, ou d'en créeer une sur une requête POST.

Sur GET, on récupère tous les objets house, on les sérialise avec la classe HouseSerializer avant de les convertir en Json

Sur POST, on désérialise le JSON reçu en objet python, et on utilise HouseSerializer pour en faire une instance de la classe House que l'on enregistre ensuite.



Le fichier views permet d'éxécuter un code sur l'appel d'une url au niveau du serveur. Les codes sont contenus dans des vues

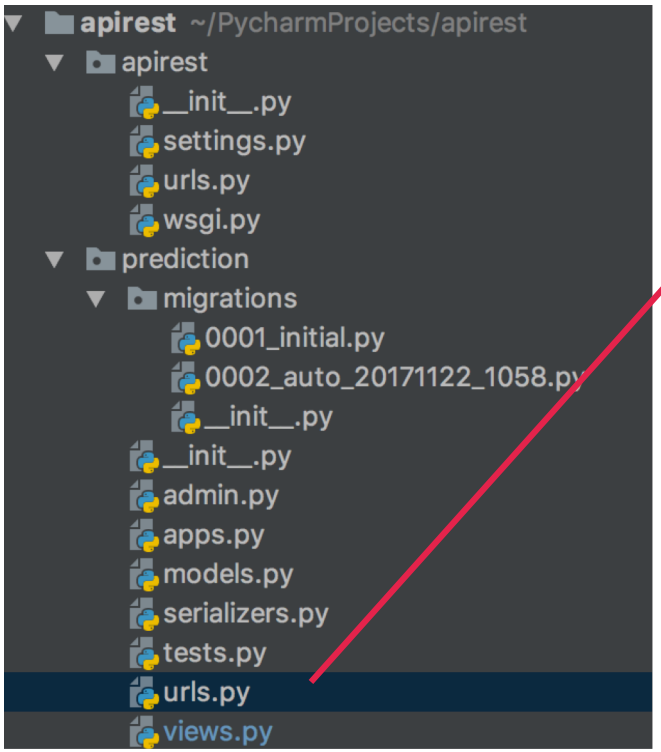
```python
@csrf_exempt
def house_detail(request, pk):


  try:
        house = House.objects.get(pk=pk)
    except House.DoesNotExist:
        return HttpResponse(status=404)

    if request.method == 'GET':
        serializer = HouseSerializer(house)
        return JsonResponse(serializer.data)

    elif request.method == 'PUT':
        data = JSONParser().parse(request)
        serializer = HouseSerializer(house, data=data)
        if serializer.is_valid():
            serializer.save()
             return JsonResponse(serializer.data)
        return JsonResponse(serializer.errors, status=400)

    elif request.method == 'DELETE':
         house.delete()
        return HttpResponse(status=204)
```

La vue house_detail permet de retourner ou d'updater les informations d'une maison particulière, désignée par son identifiant (pk).

Sur GET, on sérialise la maison avec HouseSerializer avant de la convertir en Json et de la renvoyer

Sur PUT, on désérialise le JSON reçu en objet python, et on utilise HouseSerializer pour updater l'instance de la classe House que l'on avait récupéré avec le pk.

Sur DELETE, on supprimel'instance de la classe House que l'on avait récupéré avec le pk.



Le fichier urls permet de créer / valider / désigner les urls qui sont accesibles sur le serveur.

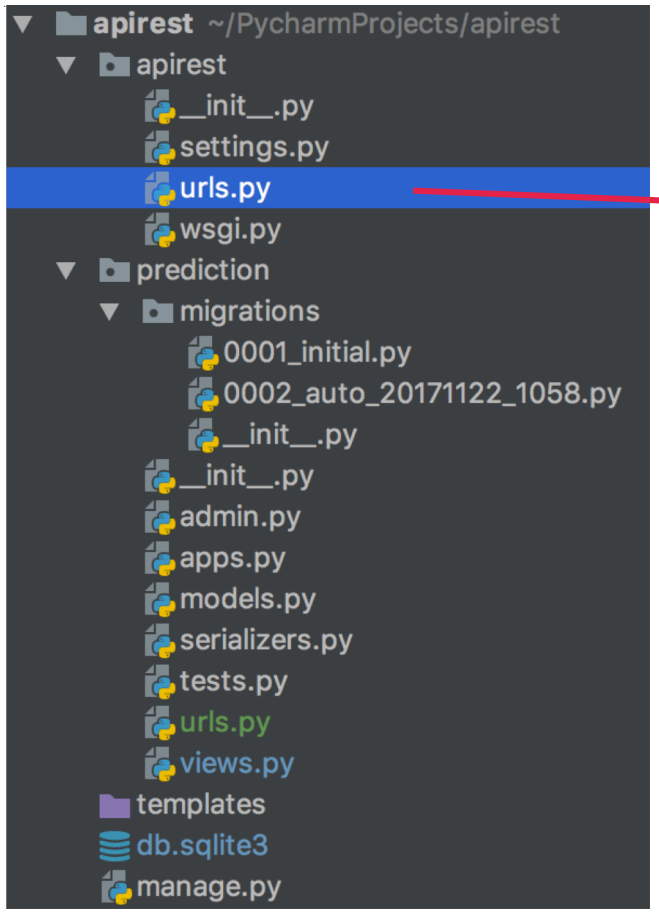
Les urls comprenant houses/

seront renvoyées vers la fonction house_liste du fichier views

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

urlpatterns = [
  url(r'^houses/$'          , views.house_list ),
  url(r'^house/(?P<pk>[0-9]+)/$' , views.house_detail),
]
```

Les urls comprenant house/un_chiffre

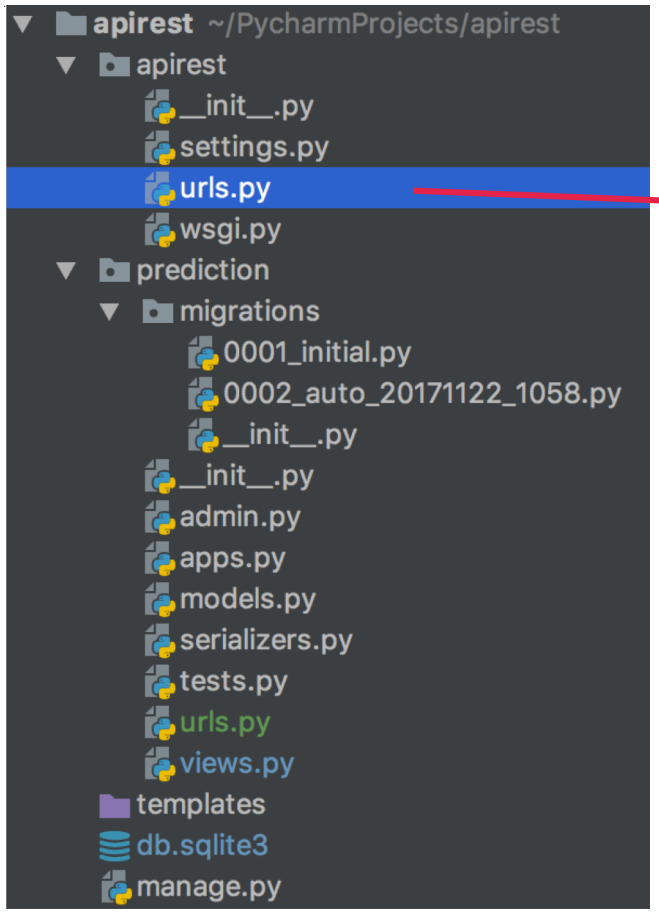
seront renvoyés vers la fonction house_detail du fichier views



Le fichier urls du dossier apirest permet de rassembler toutes les urls des sous dossiers, et de les rendre disponible à l'utilisateur.

Sans préfixe on rajoute les urls définies dans l'app prédiction

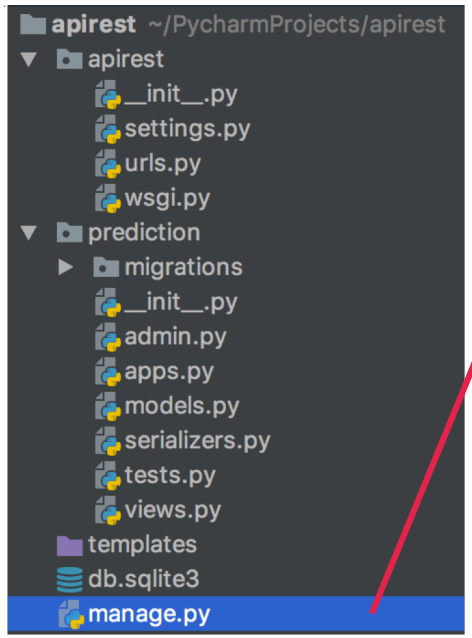
```
from django.conf.urls import url
from django.conf.urls import include

urlpatterns = [
    url(r'^', include('prediction.urls')),
]
```



Le fichier urls du dossier apirest permet de rassembler toutes les urls des sous dossiers, et de les rendre disponible à l'utilisateur.

Sans préfixe on rajoute les urls définies dans l'app prédiction

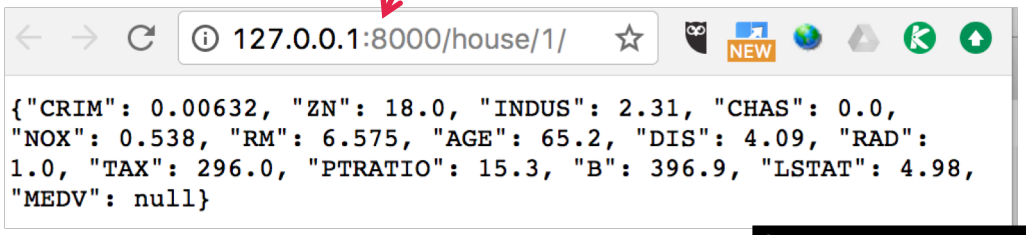
```
from django.conf.urls import url
from django.conf.urls import include

urlpatterns = [
    url(r'^', include('prediction.urls')),
]
```



Permet :

- D'agir sur tout le projet (startapp)
- De créer une base de donnée (makemigrations / migrate)
- De tester en ligne de commande (shell)
- De lancer le serveur de l'application :
  - python manage.py runserver

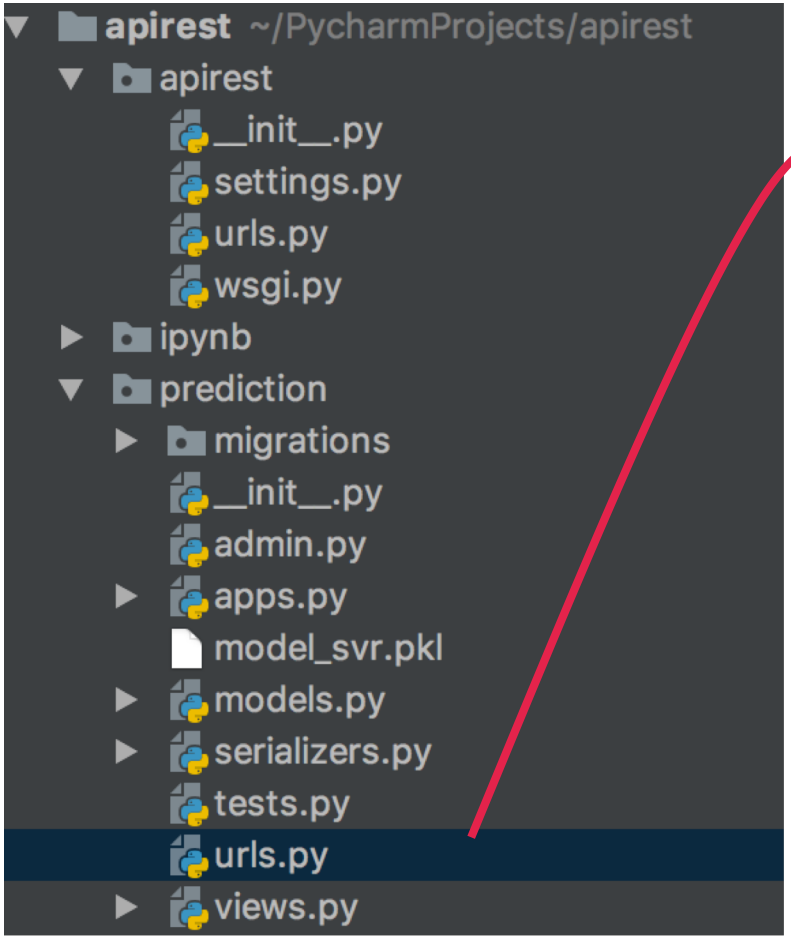
```
python manage.py runserver
Performing system checks...

System check identified no issues (0 silenced).
November 22, 2017 - 13:39:15
Django version 1.11.3, using settings 'apirest.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.

[22/Nov/2017 13:40:08] "GET /house/1/ HTTP/1.1" 200 194
```




```
$ curl http://127.0.0.1:8000/house/1/
{"CRIM": 0.00632, "ZN": 18.0, "INDUS": 2.31, "CHAS": 0.0, "NOX": 0.53
8, "RM": 6.575, "AGE": 65.2, "DIS": 4.09, "RAD": 1.0, "TAX": 296.0, "
PTRATIO": 15.3, "B": 396.9, "LSTAT": 4.98, "MEDV": null}(python35) ro
```

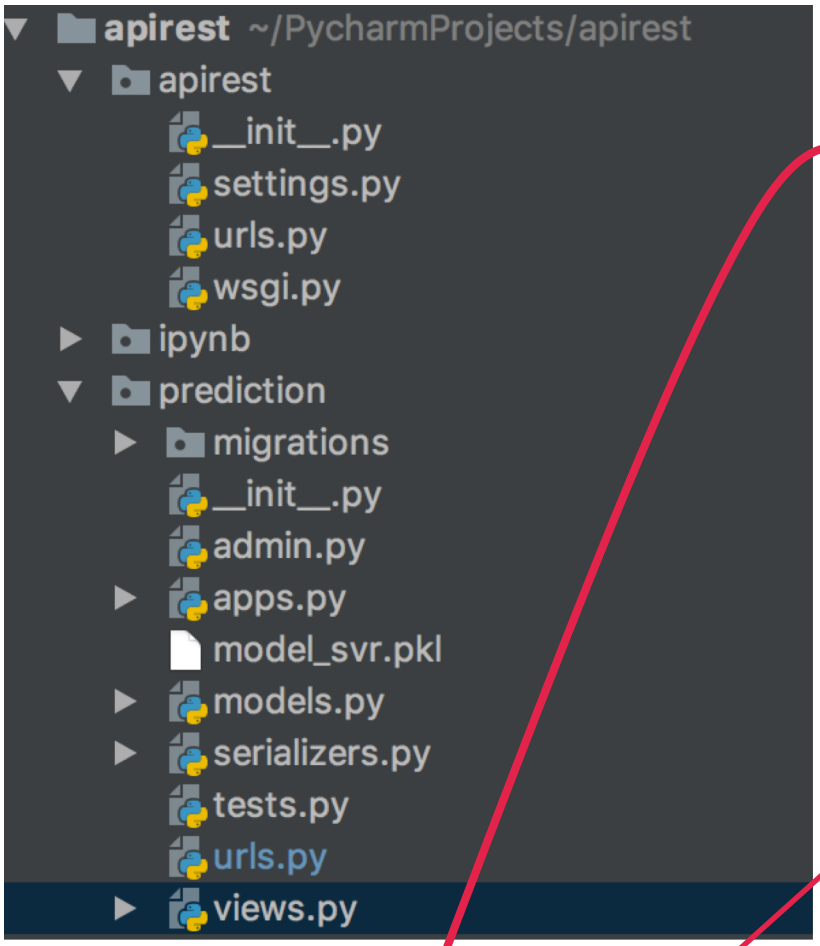
```
curl -X POST -H "Content-Type: application/json" -d '{
                                "CRIM"       : "0.02731"    ,
                                "ZN"      : "0.0"      ,
                                "INDUS"       : "7.07"   ,
                                "CHAS"        : "0.0"      ,
                                "NOX"     : "0.469"  ,
                                "RM"      : "6.421"  ,
                                "AGE"     : "78.9"   ,
                                "DIS"     : "4.9671" ,
                                "RAD"     : "2.0"      ,
                                "TAX"     : "242.0"  ,
                                "PTRATIO"   : "17.8"   ,
                                "B"       : "396.9"  ,
                                "LSTAT"       : "9.14"   ,
                                "MEDV"        : "-1"  }' http://127.0.0.1:8000/houses/
```

```
curl -X POST -H "Content-Type: application/json" -d '{    "CRIM"         : "0.02731"    ,
>                                                         "ZN"        : "0.0"      ,
>                                                         "INDUS"        : "7.07"    ,
>                                                         "CHAS"         : "0.0"       ,
>                                                         "NOX"        : "0.469"    ,
>                                                         "RM"        : "6.421"    ,
>                                                         "AGE"         : "78.9"    ,
>                                                         "DIS"         : "4.9671"    ,
>                                                         "RAD"         : "2.0"       ,
>                                                         "TAX"         : "242.0"    ,
>                                                         "PTRATIO"    : "17.8"    ,
>                                                         "B"         : "396.9"    ,
>                                                         "LSTAT"        : "9.14"    ,
>                                                         "MEDV"         : "-1"  }' http://127.0.0.1:8000/houses
/
{"CRIM": 0.02731, "ZN": 0.0, "INDUS": 7.07, "CHAS": 0.0, "NOX": 0.469, "RM": 6.421, "AGE": 78.9, "DIS": 4.9671,
"RAD": 2.0, "TAX": 242.0, "PTRATIO": 17.8, "B": 39roromaromainromainrorororororororororororororororororororororororororo
```



Rajouter une url pour les prédictions

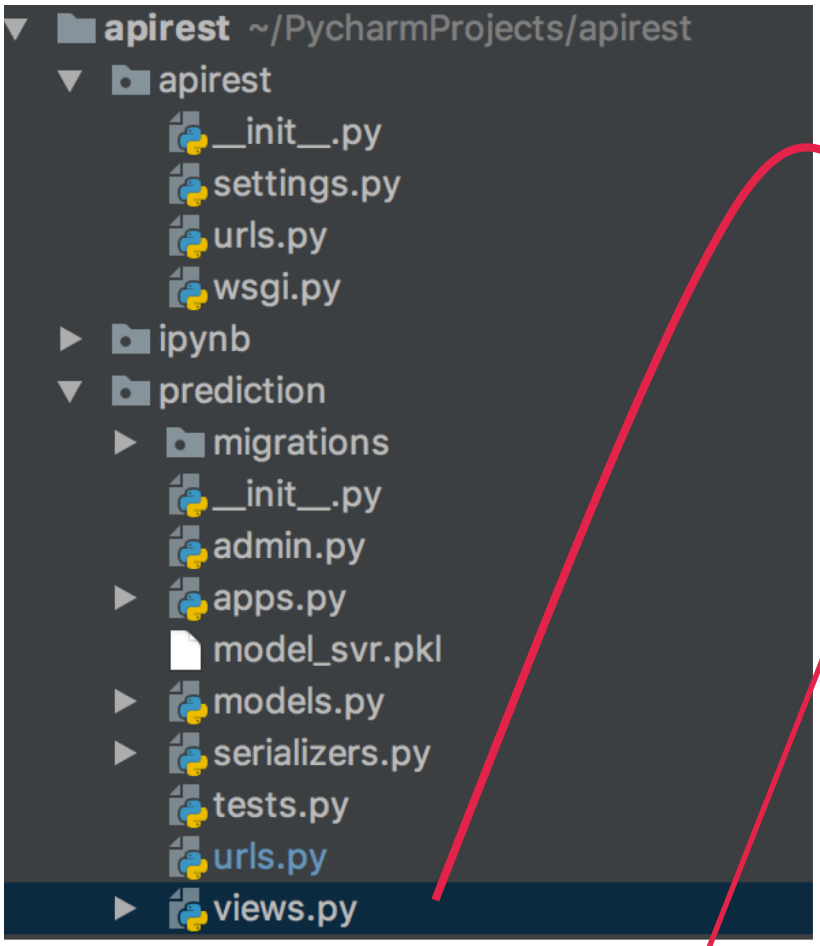
```
from django.conf.urls   import url
from prediction         import views

urlpatterns = [
  url(r'^predict/$'        , views.predict),
  url(r'^houses/$'         , views.house_list ),
  url(r'^house/(?P<pk>[0-9]+)/$' , views.house_detail),
]
```



Une vue pour les prédictions, qui ne gère que les POST

```python
@csrf_exempt
def predict(request):
    """
    Renvoie une house avec la MEDV completee
    (Attend une MEDV innexistante)
    """
    if request.method == 'GET':
        return JsonResponse(serializer.errors, status=400)

    elif request.method == 'POST':
        data = JSONParser().parse(request)
        serializer = HouseSerializer(data=data)
        if serializer.is_valid():
            data["MEDV"] = predict_medv(data)
            serializer = HouseSerializer(data=data)
            if serializer.is_valid():
                serializer.save()
                return JsonResponse(serializer.data, status=201)
        return JsonResponse(serializer.errors, status=400)
```

1) On désérialise les données reçues



Une vue pour les prédictions, qui ne gère que les POST

```python
def predict_medv(unscaled_data):
    from sklearn.externals import joblib
    colonnes        = ["CRIM", "ZN", "INDUS", "CHAS", "NOX", "RM",
                       "AGE", "DIS", "RAD", "TAX", "PTRATIO", "B",
                       "LSTAT"]
    path_to_model   = "./ipynb/model_svr.pkl"
    path_for_scaler = "./ipynb/scaler.pkl"
    unscaled_data   = [unscaled_data[colonne] for colonne in colonnes]
    model           = joblib.load(path_to_model)
    scaler          = joblib.load(path_for_scaler)
    donnees_scalees = scaler.transform(unscaled_data)
    medv            = model.predict(donnees_scalees)
    return medv
```

1) On va remettre dans l'ordre les clefs du JSON que l'on reçoit via une liste ordonnée des colonnes attendes par le modèle.

2) On utilise joblib pour charger le modèle et puis pour faire une prédiction.

```
curl -X POST -H "Content-Type: application/json" -d '{    "CRIM"         : "0.027
                                      "ZN"          : "0.0"        ,
                                      "INDUS"         : "7.07"     ,
                                      "CHAS"          : "0.0"        ,
                                      "NOX"         : "0.469"      ,
                                      "RM"          : "6.421"      ,
                                      "AGE"           : "78.9"     ,
                                      "DIS"           : "4.9671"     ,
                                      "RAD"           : "2.0"          ,
                                      "TAX"           : "242.0"    ,
                                      "PTRATIO"     : "17.8"       ,
                                      "B"           : "396.9"      ,
                                      "LSTAT"         : "9.14"     ,
                                      "MEDV"          : "-1"  }' http://127.0.0.1:8000/predic
```

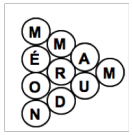
La valeur MEDV a été prédie

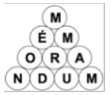
```
{"CRIM": 0.02731, "ZN": 0.0, "INDUS": 7.07, "CHAS": 0.0, "NOX": 0.469, "RM": 6.421, "AGE": 78.9, "DIS":
 4.9671, "RAD": 2.0, "TAX": 242.0, "PTRATIO": 17.8, "B": 396.9, "LSTAT": 9.14, "MEDV": 25.5666572770508
4}romain@MacBook-Pro-de-oursin:~$
```

Repo = https://github.com/romainjouin/django_api_rest

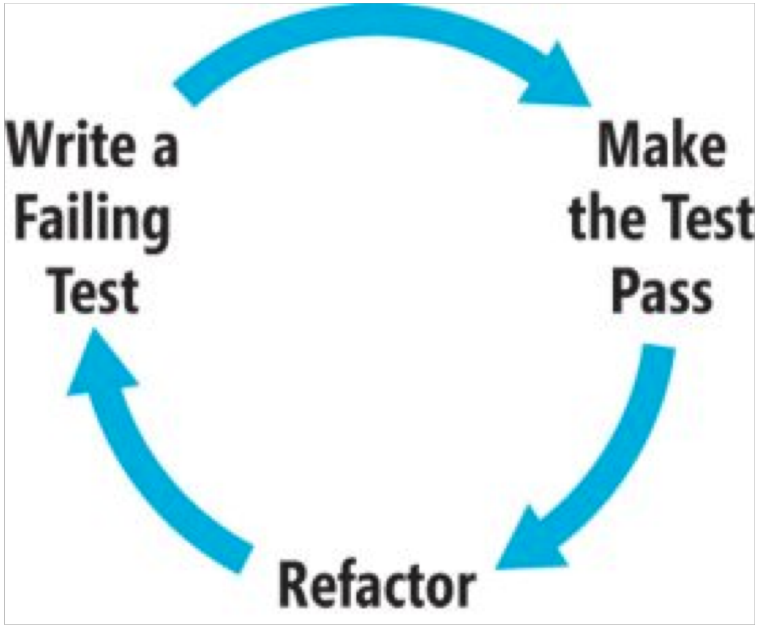
Python - Introduction

1. Historique et principes
2. Environnement de développement
3. Introduction à Python : syntaxe
    - Python avancé
4. Introduction à Scipy (Pandas)
5. Introduction à Matplotlib
    - Autres librairies graphiques
6. Introduction à Scikit-Learn
7. Préparation de données
8. DASK
9. Création d'une API Rest Django
10. Test Driven Development en Django
11. Gestion des environnements
12. Annexes





TDD : principes



TDD : module python de base

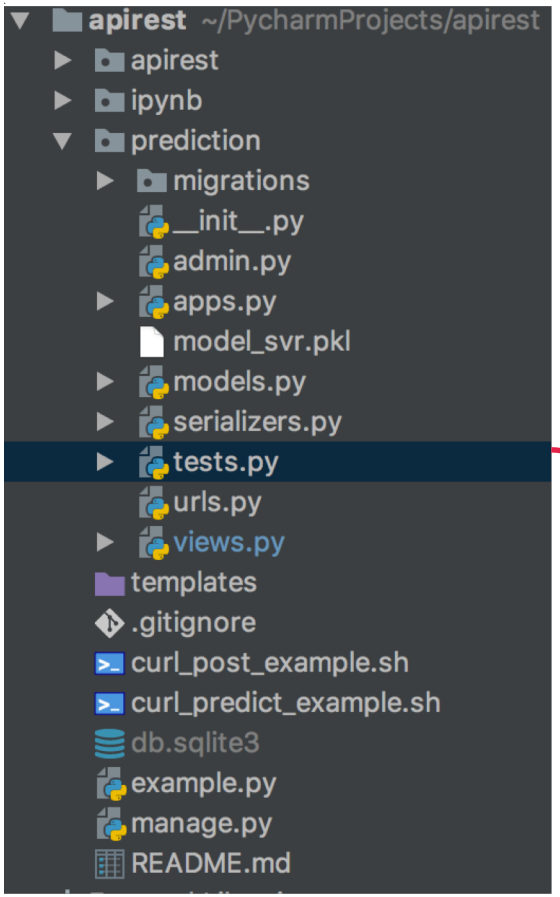
https://docs.python.org/3/library/unittest.html

Les fonctions setUp et tearDown sont appelées au début et à la fin du test (permettent de setter l'environnement de test et de le détruire)

```
from selenium import webdriver
import unittest

class NewVisitorTest(unittest.TestCase):  #1

    def setUp(self): #2
        self.browser = webdriver.Firefox()

    def tearDown(self):  #3
        self.browser.quit()

    def test_can_start_a_list_and_retrieve_it_later(self):  #4
        # Edith has heard about a cool new online to-do app. She goes
        # to check out its homepage
        self.browser.get('http://localhost:8000')

        # She notices the page title and header mention to-do lists
        self.assertIn('To-Do', self.browser.title)  #5
        self.fail('Finish the test!')  #6

        # She is invited to enter a to-do item straight away
        [...rest of comments as before]

if __name__ == '__main__':  #7
    unittest.main(warnings='ignore')  #8
```

Les fonctions commençant par test sont détectées automatiquement et appelées par le module unittest

La fonction unittest.main exécute les tests contnues dans le fichier

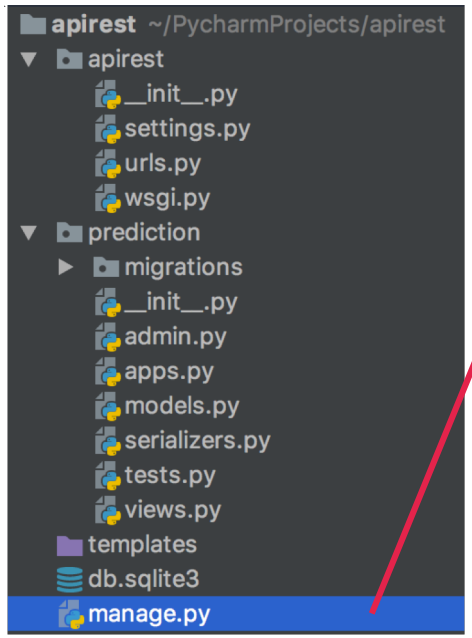


TDD : module de test avancé par django

https://docs.python.org/3/library/unittest.html

https://docs.djangoproject.com/fr/1.11/topics/testing/

```
from django.test import TestCase

# Create your tests here.
class SmokeTest(TestCase):
    def test_bad_maths(self):
        self.assertEqual(1+1,3)
```

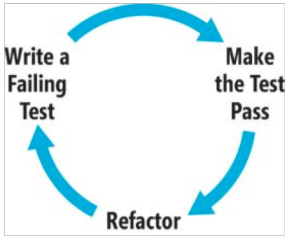


Permet :

- D'agir sur tout le projet (startapp)
- De créer une base de donnée (makemigrations / migrate)
- De tester en ligne de commande (shell)
- De lancer le serveur de l'application : (runserver)
- De lancer les tests: (test)
  - python manage.py test

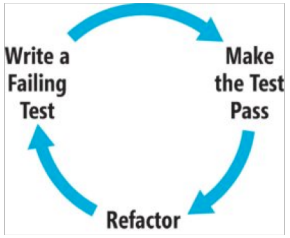
```
python manage.py test
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
F
======================================================================
FAIL: test_bad_maths (prediction.tests.SmokeTest)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/Users/romain/PycharmProjects/apirest/prediction/tests.py", line 6, in test_bad_maths
    self.assertEqual(1+1,3)
AssertionError: 2 != 3

--------
[22/Nov/2017 13:40:08] "GET /house/1/ HTTP/1.1" 200 194
Ran 1 test in 0.001s

FAILED (failures=1)
Destroying test database for alias 'default'...
```

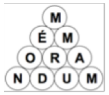
On a respecté la boucle du TDD : fail first !



```python
""" Python core """
import random
""" Django Framework"""
from django.core.urlresolvers   import resolve
from django.http                import HttpRequest
from django.utils.six           import BytesIO
""" DjangoRestFramework"""
from rest_framework.test        import APITestCase
from rest_framework.parsers     import JSONParser
""" Django specific """
from prediction.views           import house_list
from prediction.views           import house_detail
from prediction.models          import House
from prediction.serializers     import HouseSerializer
class Test_Houses_database(APITestCase):
    def setUp(self):
    for i in range(10):
      house = House(CRIM       =  0.02731 * (1 + random.random() ),
                    ZN         =  0.0     * (1 + random.random() ),
                    INDUS      =  7.07    * (1 + random.random() ),
                    CHAS       =  0.0     * (1 + random.random() ),
                    NOX        =  0.469   * (1 + random.random() ),
                    RM         =  6.421   * (1 + random.random() ),
                    AGE        =  78.9    * (1 + random.random() ),
                    DIS        =  4.9671  * (1 + random.random() ),
                    RAD        =  2.0     * (1 + random.random() ),
                    TAX        =  242.0   * (1 + random.random() ),
                    PTRATIO    =  17.8    * (1 + random.random() ),
                    B          =  396.9   * (1 + random.random() ),
                    LSTAT      =  9.14    * (1 + random.random() ),
            )
      house.save()
```

On utilise la fonction setUp pour populer la base de donnée





```python
class Test_Houses_database(APITestCase):

    def test_url_pattern_for_houses_exists(self):
        found = resolve("/houses/")
        self.assertEqual(found.func, house_list)

    def test_url_pattern_for_one_specific_house_exists(self):
        found = resolve("/house/1/")
        self.assertEqual(found.func, house_detail)

    def test_houses_url_return_10_houses(self):
         request         = HttpRequest()
         request.method = "GET"
        response         = house_list(request)
        stream           = BytesIO(response.content)
        data             = JSONParser().parse(stream)
        houses           = HouseSerializer(data=data, many=True)

        if houses.is_valid():
            assert 10 == len(houses.validated_data)

    def test_get_an_house(self):
        response    = self.client.get('/house/1/')
        stream      = BytesIO(response.content)
        data        = JSONParser().parse(stream)
        house       = HouseSerializer(data=data, many=True)

        if house.is_valid():
            assert 1 == len(house.validated_data)
```

Les fonctions commençant pa test sont détectées automatiquement

On utilise les fonctions assert pour ont détectées automatiquement

On utilise HttpRequest pour simuler une demande client

L'objet HttpRequest est passé en paramètre de la fonction

La réponse est un objet Byte que l'on doit désérialiser et transformer en objet house.

```
Creating test database for alias 'default'...
System check identified no issues (0 silenced).
....
----------------------------------------------------------------------
Ran 4 tests in 0.032s

OK
Destroying test database for alias 'default'...
```

Python - Introduction

1. Historique et principes
2. Environnement de développement
3. Introduction à Python : syntaxe
    - Python avancé
4. Introduction à Scipy (Pandas)
5. Introduction à Matplotlib
    - Autres librairies graphiques
6. Introduction à Scikit-Learn
7. Préparation de données
8. DASK
9. Création d'une API Rest Django
10. Test Driven Development en Django
11. Gestion des environnements
12. Annexes