Créer une application django dans pycharm

Par Denis Mashutin - Delphine Massenhove (traducteur)

Date de publication : 30 août 2023

Ce tutoriel vous guide à travers les étapes de la création d'une application Django simple qui indique la température de l'air là où vous vous trouvez et permet de prendre connaissance des conditions météo à différents endroits.

Pour réagir au contenu de cet article, un espace de dialogue vous est proposé sur le forum.
Commentez

Introduction

L'idée à la base de Django est de permettre aux développeurs de créer des applications rapidement. La maîtrise de ce framework permet de réduire considérablement le chemin entre le concept et l'obtention d'une application web opérationnelle. Si vous voulez aller encore plus vite, vous pouvez apprendre à créer des applications Django dans PyCharm.

Ce tutoriel vous guide à travers les étapes de la création d'une application Django simple qui indique la température de l'air là où vous vous trouvez et permet de prendre connaissance des conditions météo à différents endroits.

Grâce à ce tutoriel, vous apprendrez comment :

- créer un projet Django dans PyCharm ;
- écrire des modèles, des vues et des gabarits ;
- faire des appels d'API et traiter les réponses ;
- vous connecter à des bases de données et y importer des données.

Pour obtenir l'intégralité du code de l'application, vous pouvez cloner le **référentiel**. Pour plus d'informations sur le clonage, consultez la **documentation PyCharm**.

I - Prérequis

Ce tutoriel s'adresse aux développeurs qui ont déjà plusieurs années d'expérience avec Python. Par conséquent, nous partons du principe que Python est déjà installé sur votre ordinateur. Mais si ce n'est pas le cas, pas de souci ! Vous pouvez télécharger et installer la version de Python de votre choix lorsque vous commencerez votre premier projet dans PyCharm.

La prise en charge de Django est une fonctionnalité professionnelle, vous devez utiliser **PyCharm Professional**. Il y a une période d'essai gratuit de 30 jours pour les nouveaux utilisateurs, et les étudiants et enseignants bénéficient d'une **licence gratuite**. Ce tutoriel a été conçu dans **PyCharm 2023.1**, avec la **nouvelle interface utilisateur** activée.

Pour en savoir plus et accéder aux instructions d'installation pour les différents systèmes d'exploitation, consultez la **documentation de PyCharm**.

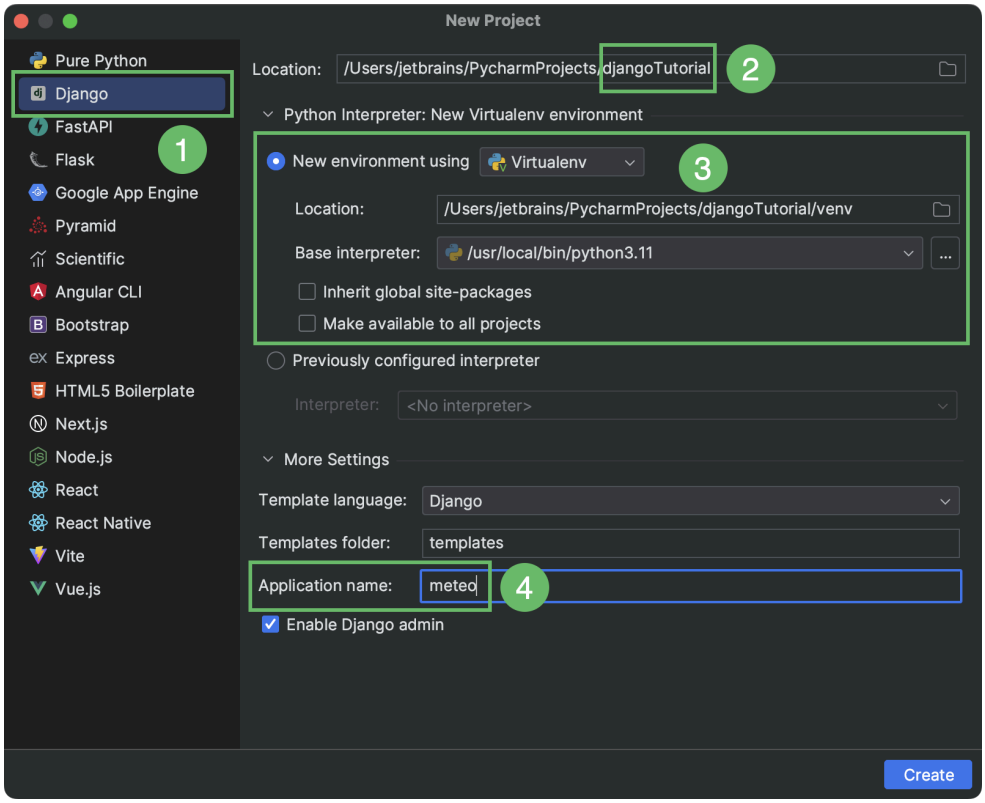
II - Créer une application opérationnelle

Les premières étapes nous permettront d'obtenir une première version basique et opérationnelle de notre application.

II-1 - Créer un projet Django dans PyCharm

Pour créer votre projet, lancez PyCharm et cliquez sur **New Project**. Si PyCharm est déjà lancé , sélectionnez **File | New Project** dans le menu principal.

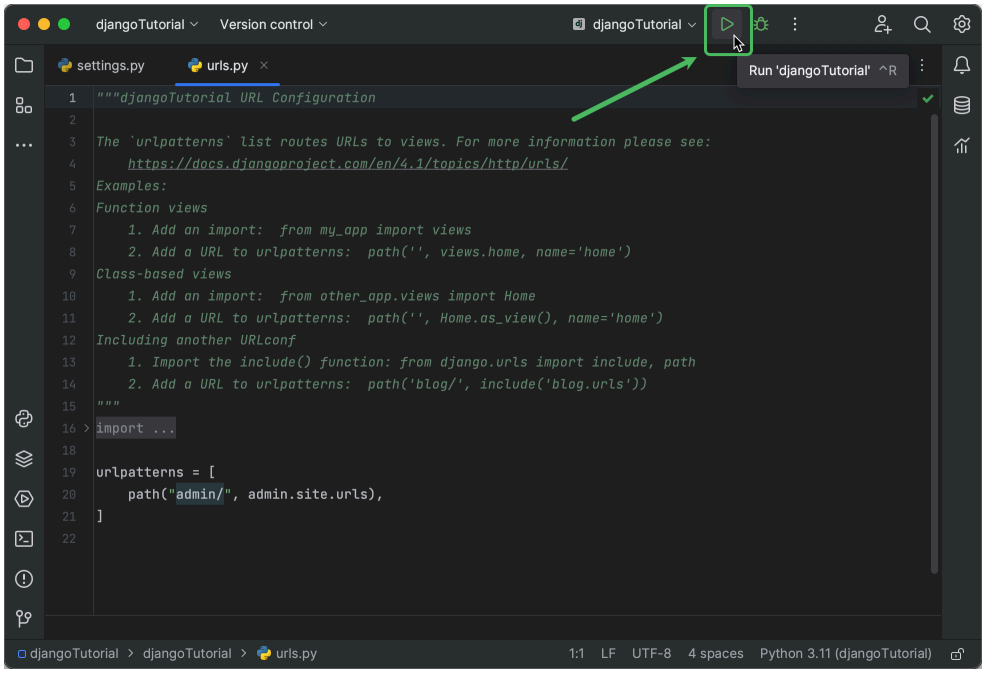
Dans la fenêtre **New Project**, spécifiez ce qui suit :



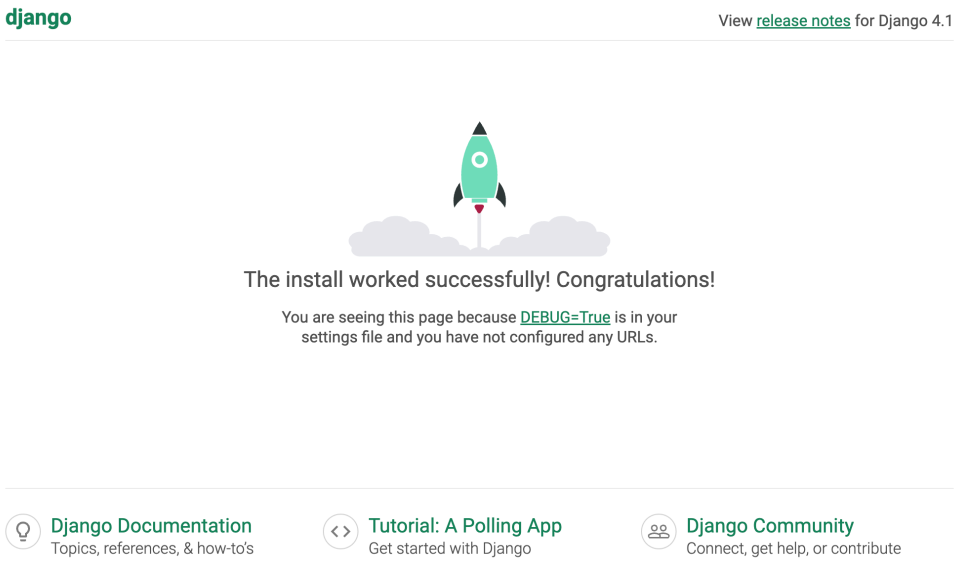
- Choisissez **Django** comme type de projet ;
- Saisissez le nom du répertoire dans lequel votre projet se trouvera. Cela sera également le nom de votre projet ;
- Créez un environnement virtuel pour votre nouveau projet Django dans lequel PyCharm installera vos dépendances. Pour ce tutoriel, nous allons sélectionner l'option **virtualenv ;**
- PyCharm est maintenant prêt pour la création d'une application Django dans votre projet. Veillez à nommer votre application dès maintenant.

Pour commencer, cliquez sur **Create** afin que PyCharm crée la structure de fichiers et installe Django et les autres dépendances requises.

Cliquez sur l'icône **Run** en haut de la fenêtre pour démarrer le serveur Django :



La fenêtre d'outils **Run** s'ouvre. Cliquez sur le lien pour ouvrir la fenêtre du navigateur :

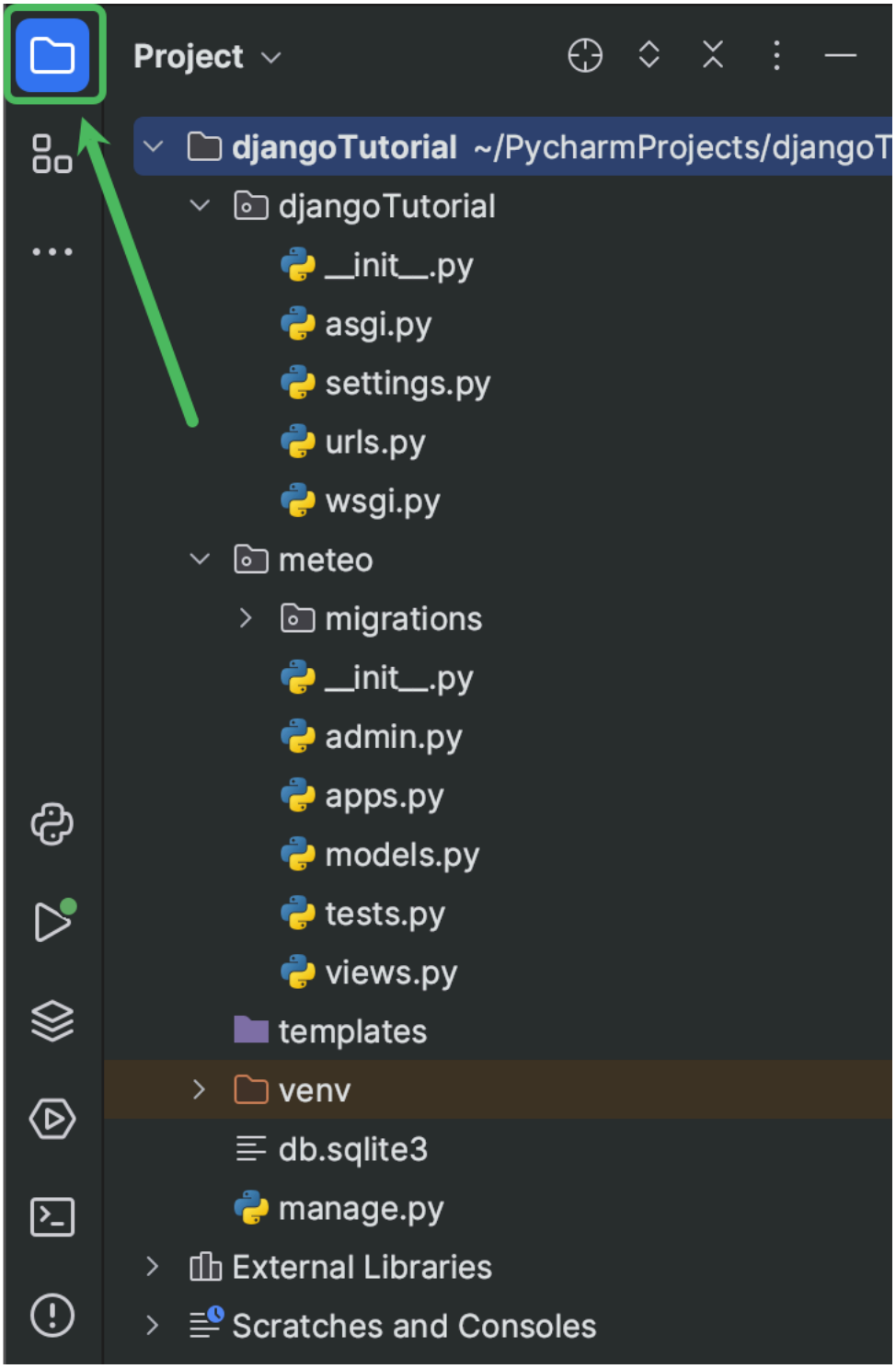


Et voilà, il n'a fallu que quelques minutes pour obtenir un serveur Django exécutable dans PyCharm. C'est un bon début, mais le meilleur reste à venir.

II-2 - Créer votre première vue

Il s'agit maintenant de créer la logique de l'application. Dans Django, cela se fait en écrivant des classes ou des fonctions dans **views.py**.

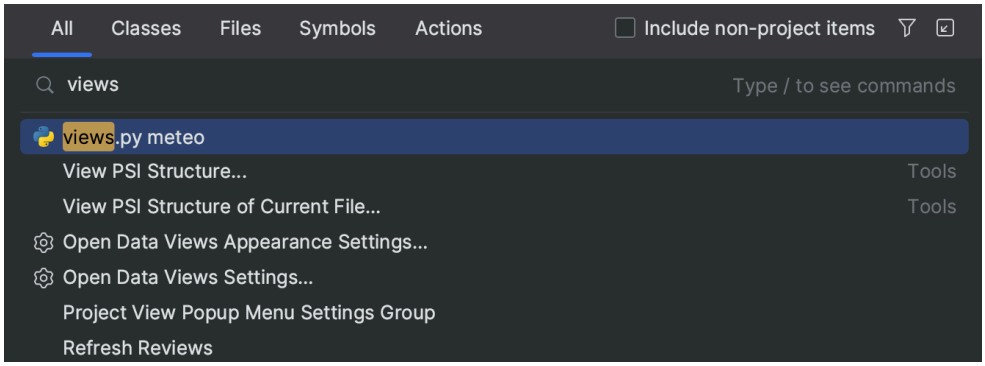
Vous pouvez examiner la structure de votre projet à tout moment en cliquant sur l'icône dossier dans le coin supérieur gauche de la fenêtre de PyCharm ou en appuyant sur ⌘1 **/ Alt+1** :



Ce tutoriel fournit des raccourcis qui permettent de gagner beaucoup de temps, par exemple, appuyez deux fois sur ⇧ **(Maj)** pour ouvrir la fenêtre **Search Everywhere**. Ce raccourci vous permet littéralement de tout trouver, notamment les fichiers, paramètres et actions de votre projet.

Nous allons l'utiliser pour ouvrir rapidement **views.py**.

Saisissez *views*, placez le curseur sur **views.py meteo** et appuyez sur **Entrée** :



Un onglet d'éditeur avec **views.py** s'ouvre. Commençons par écrire la fonction temp_here, qui renverra la température qu'il fait actuellement là où nous nous trouvons.

Collez le code suivant dans l'éditeur :

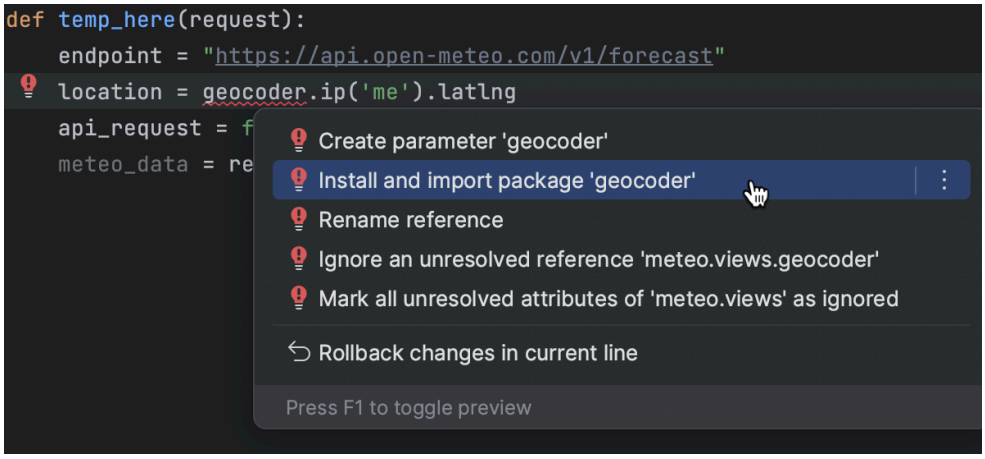
```
1. import requests
2. def temp_here():
3.     location = geocoder.ip('me').latlng
4.     endpoint = "https://api.open-meteo.com/v1/forecast"
5.     api_request = f"{endpoint}?
latitude={location[0]}&longitude={location[1]}&hourly=temperature_2m"
6.     return requests.get(api_request).json()
```

Que se passe-t-il alors ? Tout d'abord, nous importons la bibliothèque *requests*, qui est requise pour faire des appels d'API. Si l'instruction d'importation est soulignée par une ligne ondulée rouge, cela signifie que le paquet n'est pas disponible dans l'interpréteur Python sélectionné.

Survolez-la avec le curseur et sélectionnez **Install package requests**.

Pour obtenir la température actuelle, temp_here fait un appel à l'**API Weather Forecast**. C'est une API gratuite qui ne nécessite pas de clé d'API. Tout ce que nous avons besoin de connaître est le point de terminaison (**https://api.open-meteo.com/v1/forecast**) et les coordonnées. Pour ces dernières, nous allons utiliser **Geocoder**, une bibliothèque Python très simple qui permet de relever les coordonnées de différents endroits.

Placez le caret sur geocoder, qui est mis en évidence par une ligne ondulée rouge, et appuyez sur ⌥Entrée / **Alt +Entrée** pour voir les correctifs rapides disponibles. Sélectionnez **Install and import package 'geocoder'** :

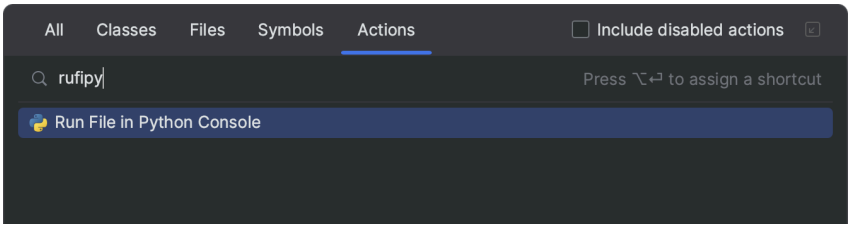


PyCharm installe le paquet et ajoute l'instruction d'importation au début du fichier.

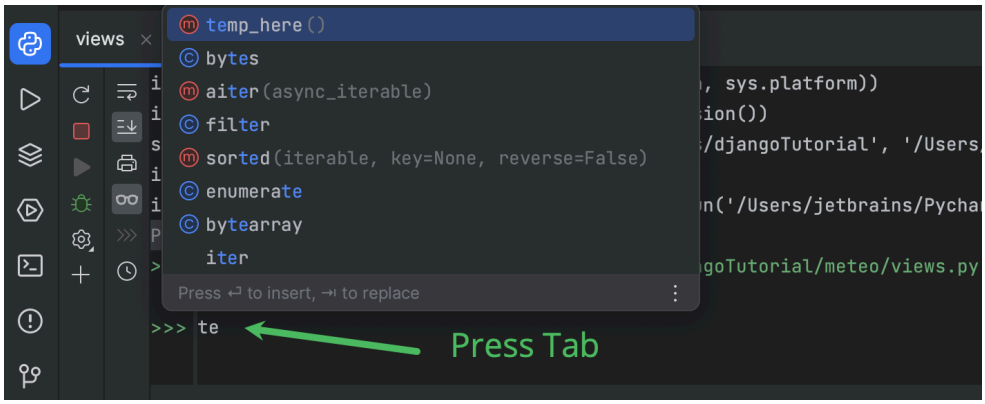
Que pensez-vous de faire une requête de test pour vous assurer que tout fonctionne comme prévu ? Le moyen le plus simple de le faire est d'appeler la fonction dans la console Python.

Utilisons un autre raccourci pour gagner du temps : **Find action**. Pas besoin de parcourir des menus avec la souris ou de mémoriser des douzaines de raccourcis, il suffit d'appuyer sur ⇧⌘A / **Ctrl+Maj+A** et de rechercher **Run File in Python Console**.

Il n'est pas nécessaire de saisir les mots en entier, vous pouvez utiliser une suite de lettres comme '*rufipy*' et obtenir malgré tout le résultat voulu :



Lorsque vous exécutez la commande, PyCharm charge les importations et la définition de fonction dans la console. Maintenant, appelez `temp_here`. Vous pouvez appuyer sur la touche **Tab** pour la saisie semi-automatique du code :



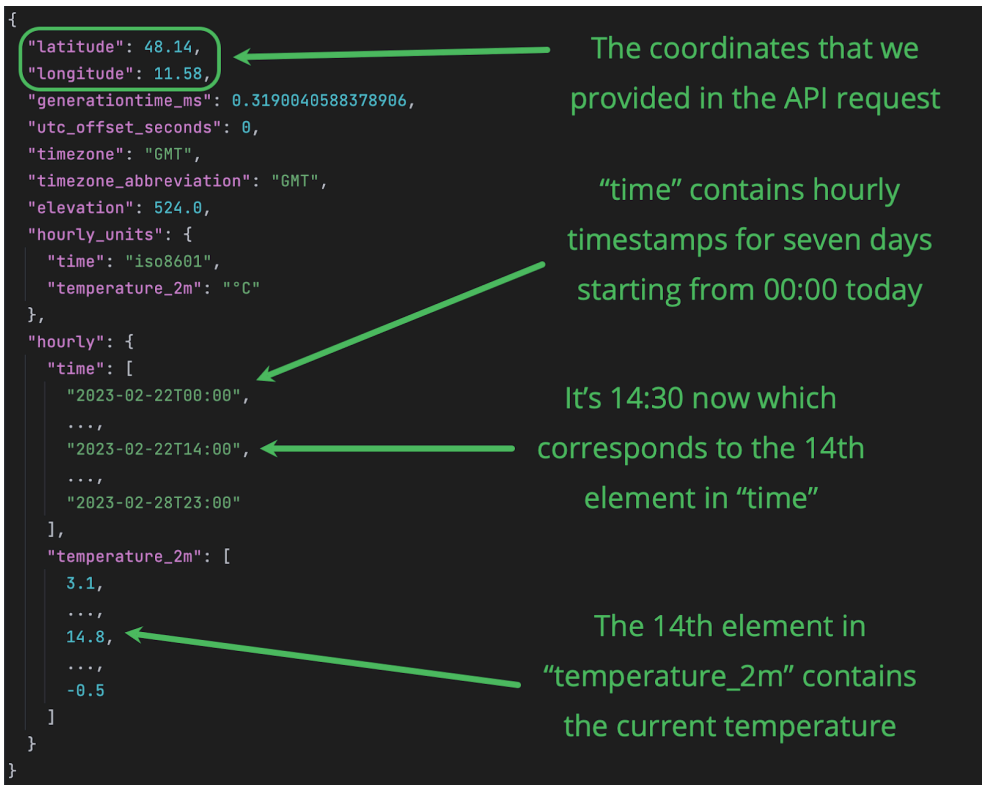
Examinez la réponse de l'API dans la sortie de la console. C'est une chaîne très longue, mais vous en trouverez une représentation claire ci-dessous.

Si vous souhaitez explorer la réponse vous-même, faites comme suit :

- Cliquez sur la sortie de la console 3 fois et copiez-la dans le presse-papiers en appuyant sur ⌘C **/ Ctrl+C ;**
- Créez un fichier en appuyant sur ⇧⌘N **/ Ctrl+Alt+Maj+Insert** et sélectionnez le type de fichier **JSON** (commencez simplement à saisir 'js…') ;
- Collez la réponse et appliquez l'action **Reformat Code**. Vous pouvez utiliser **Find action** ou appuyer sur ⌥⌘L **/ Ctrl+Alt+L**.

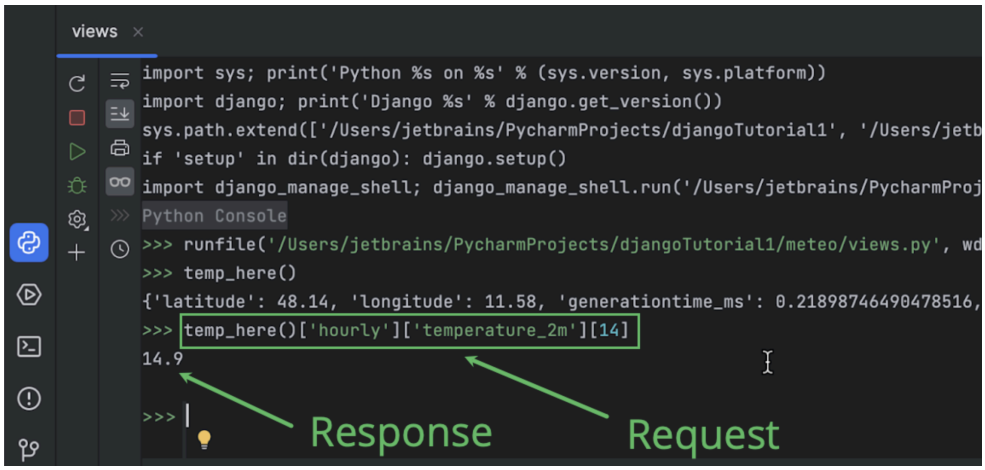
Les informations requises sont contenues dans l'élément *hourly* sous la clé *temperature_2m*. Cette clé pointe vers une liste de valeurs.

Pour obtenir la température actuelle, nous devons passer l'heure actuelle en indice. Par exemple, s'il est 14 h 30, nous prendrons le 14^{e} élément de la liste.



Vérifions si temp_here()['hourly']['temperature_2m'][14] fournit les informations requises.

Remplacez 14 par la valeur correspondant à l'heure qu'il est actuellement et saisissez le code dans la console :



Nous obtenons 14,9 °C pour notre position. Et vous ?

Nous allons modifier la fonction afin qu'elle puisse extraire la température actuelle à partir de la réponse de l'API :

```
1. def temp_here():
2.     location = geocoder.ip('me').latlng
3.     endpoint = "https://api.open-meteo.com/v1/forecast"
4.     api_request = f"{endpoint}?
latitude={location[0]}&longitude={location[1]}&hourly=temperature_2m"
5.     now = datetime.now()
6.     hour = now.hour
7.     meteo_data = requests.get(api_request).json()
8.     temp = meteo_data['hourly']['temperature_2m'][hour]
9.     return temp
```

N'oubliez pas d'importer datetime :

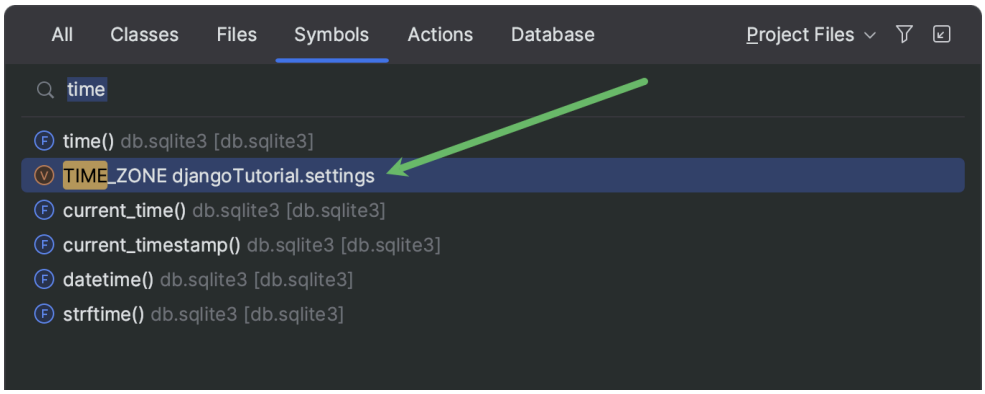
Cliquer sur ce lien pour lancer l'animation

Cliquez sur **Rerun** dans le coin supérieur gauche de la barre d'outils de la console Python pour recharger la définition de fonction mise à jour, puis appelez de nouveau temp_here :

Si le résultat est différent de celui que vous aviez avant de modifier le code de la fonction temp_here, cela peut être dû à une valeur TIME_ZONE incorrecte dans **settings.py**.

Pour plus d'informations, consultez la **documentation Django**.

Pour accéder rapidement à ce paramètre, appuyez deux fois sur ⇧ (Maj), passez à Symbols en appuyant plusieurs fois sur la touche **Tab**, puis commencez à saisir '*time*…'.



Nous allons maintenant transformer temp_here en fonction d'affichage. Afin d'être reconnue en tant que vue par Django, cette fonction doit accepter l'objet **HttpRequest** comme premier paramètre, généralement appelé request. Elle doit également renvoyer l'objet **HttpResponse**.

Voici ce à quoi **views.py** doit ressembler :

```
1. from datetime import datetime
2.
3. import geocoder as geocoder
4. import requests
5. from django.http import HttpResponse
6.
7.
8. def temp_here(request):
9.     location = geocoder.ip('me').latlng
10.     endpoint = "https://api.open-meteo.com/v1/forecast"
11.     api_request = f"{endpoint}?
latitude={location[0]}&longitude={location[1]}&hourly=temperature_2m"
12.     now = datetime.now()
13.     hour = now.hour
14.     meteo_data = requests.get(api_request).json()
15.     temp = meteo_data['hourly']['temperature_2m'][hour]
16.     return HttpResponse(f"Here it's {temp}")
```

Comme vous pouvez le voir, nous n'avons pas changé grand-chose : temp_here accepte désormais request comme argument et renvoie HttpResponse avec une chaîne.

Si vous préférez utiliser des degrés Fahrenheit au lieu de degrés Celsius, il suffit d'ajouter le paramètre temperature_unit dans la requête de l'API :

```
api_request = f"{endpoint}?
latitude={location[0]}&longitude={location[1]}&hourly=temperature_2m&temperature_unit=fahrenheit"
```

Si vous préférez les degrés Celsius, ignorez simplement cette modification.

II-3 - Configurer les URL

Pour configurer l'accès à notre application depuis un navigateur, mettez à jour **urls.py**. Appuyez deux fois sur **Maj** et saisissez les url à rechercher, puis ouvrez-les comme indiqué ci-dessus.

Ajoutez la ligne suivante à urlpatterns. Vous pouvez utiliser l'action **Reformat Code** ⌥⌘L **/ Ctrl+Alt+L** pour restaurer facilement les retraits après le collage :

path("", include('meteo.urls')),

N'oubliez pas d'importer include depuis django.urls.include.

meteo.urls est désormais signalé comme une référence non résolue par une ligne jaune ondulée, car le fichier n'existe pas encore. Cela sera corrigé à l'étape suivante.

Les projets Django contiennent souvent plusieurs applications. Bien que cela ne soit pas le cas ici, il faut prendre en compte le futur développement du projet. C'est pourquoi nous créons un fichier **urls.py** pour chaque application dans le dossier correspondant et les incluons tous dans le fichier **urls.py** du projet.

Nous allons donc créer **urls.py** dans le dossier de l'application meteo.

Faites un clic droit sur le répertoire **meteo** dans la fenêtre d'outils **Project**.

Sélectionnez New > Python File et tapez urls.

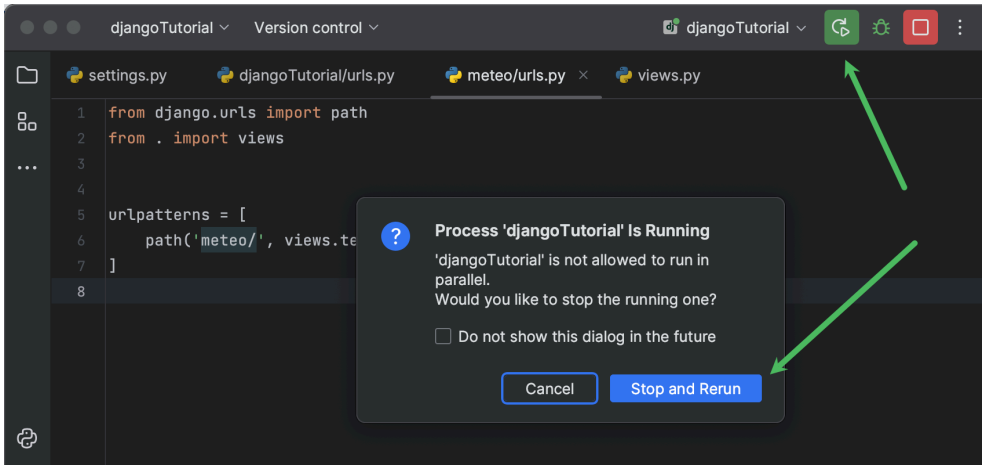
Le fichier que vous venez de créer s'ouvre. Remplissez-le avec le code suivant :

```
1. from django.urls import path
2. from . import views
3.
4.
5. urlpatterns = [
6.     path('meteo/', views.temp_here, name='temp_here'),
7. ]
```

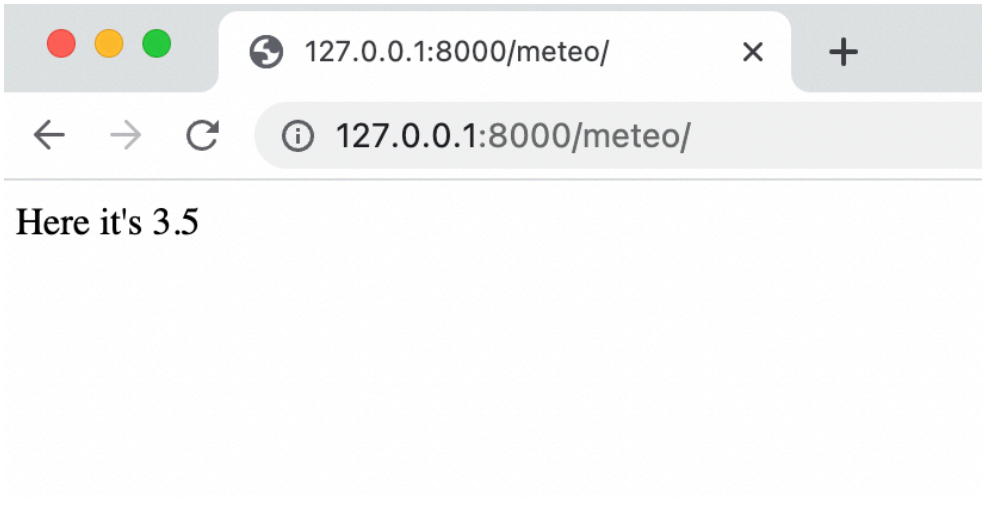
Désormais, lorsqu'on ouvre *<server_address>/meteo* dans le navigateur, la fonction temp_here de **views.py** est appelée et le navigateur affiche le rendu du résultat renvoyé par la fonction.

II-4 - Cela fonctionne !

Redémarrez le serveur Django en cliquant sur le bouton **Rerun** dans le coin supérieur droit et confirmez l'action :

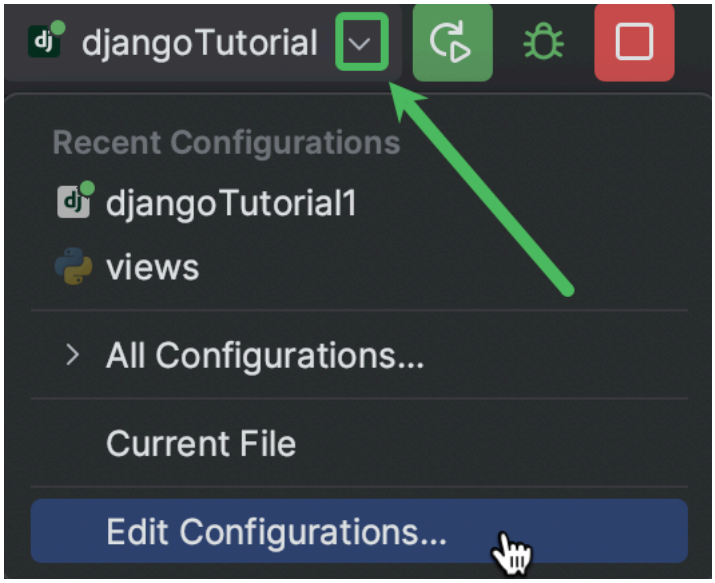


Ouvrez *http://127.0.0.1:8000/meteo/* dans votre navigateur. Vous devriez voir quelque chose de semblable à ce qui suit :

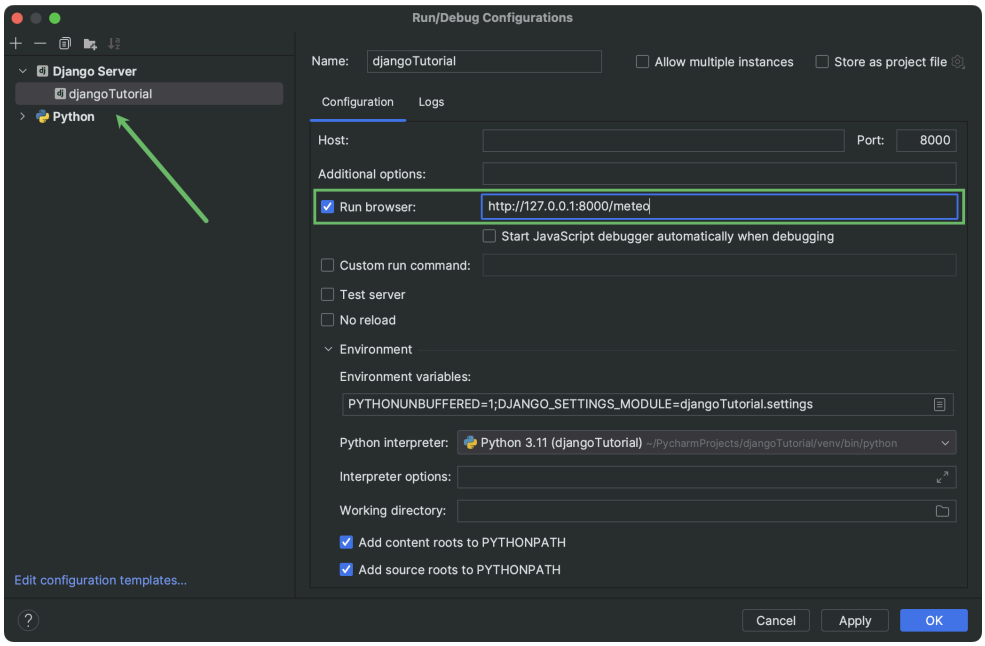


Si vous n'avez pas envie d'ouvrir le navigateur et de saisir l'adresse, ou d'actualiser la page manuellement à chaque fois que vous redémarrez le serveur Django, vous pouvez configurer PyCharm pour qu'il le fasse à votre place.

Ouvrez la liste déroulante dans le widget **Run** et sélectionnez **Edit configurations** :



- Sélectionnez la configuration de votre projet dans le volet de gauche, cochez la case **Run browser** et ajoutez *meteo* à l'URL :



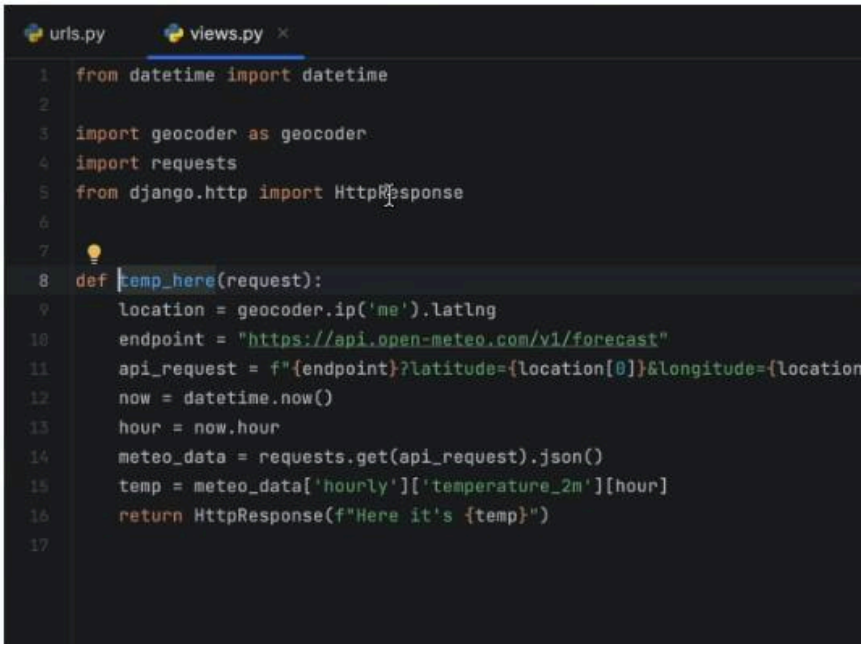
Cliquez sur **OK** pour appliquer les modifications.

Techniquement, l'application fonctionne. Toutefois, son rendu dans le navigateur n'est pas très esthétique et elle ne donne toujours pas la possibilité d'obtenir la météo d'une localisation aléatoire. Dans les prochaines étapes, nous allons ajouter un gabarit et importer des données pour résoudre ces problèmes.

III - Améliorer l'expérience

III-1 - Ajouter un gabarit

Revenons à **views.py** et modifions à nouveau la fonction temp_here. Si vous êtes toujours dans **meteo/urls.py**, vous pouvez y accéder en un rien de temps. Maintenez ⌘ **/ Ctrl**, survolez temp_here avec le curseur et cliquez dessus lorsqu'il se transforme en hyperlien :



Ajoutez une ligne avant l'instruction return et saisissez template = loader.

Appuyez sur ⌥Entrée/ **Alt+Entrée** et utilisez un correctif rapide pour importer loader depuis django.template.loader :

Cliquer sur ce lien pour lancer l'animation

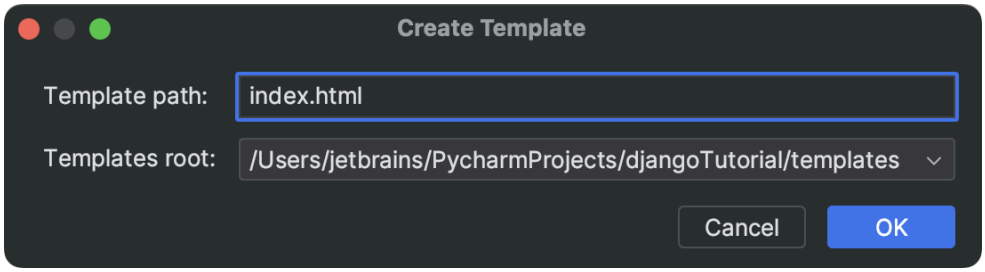
Utilisez ensuite get_template() pour charger **index.html** en tant que gabarit :

template = loader.get_template('index.html')

Remplacez maintenant l'instruction return par les deux lignes suivantes :

```
context = {'temp': temp}
   return HttpResponse(template.render(context, request))
```

Index.html est mis en évidence par une ligne jaune ondulée, car il n'existe pas encore. Survolez-le avec le curseur et sélectionnezCreate template index.html.



Cliquez sur **OK**. PyCharm va créer **index.html** et l'ouvrir pour modification.

Le fichier est maintenant vide. Nous allons utiliser un **modèle dynamique** pour le remplir avec un gabarit de code html. Tapez *html:5* et appuyez sur la touche **Tab** :

Le contenu visible d'une page html est situé entre les balises <body></body>.

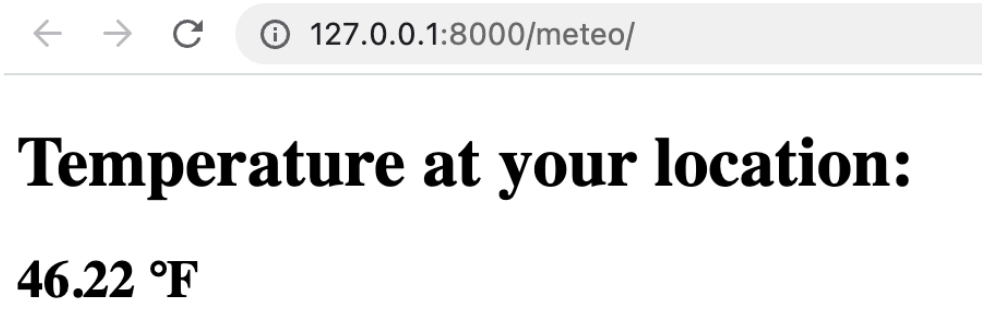
Insérez-y le code suivant :

```
<h1>Temperature at your location:</h1>
<h2>{{ temp }} #</h2>
```

Ici, *temp* est une variable qui est transférée au gabarit à partir des vues. Les noms et les valeurs des variables doivent être enregistrés dans un dictionnaire et transmis lors du rendu d'un gabarit. C'est ce que nous avons fait en affectant {'temp' : temp} à la variable de contexte dans **views.py** ci-dessus.

La représentation Unicode du symbole de degrés Celsius est &#8451. Pour les degrés Farenheit, utilisez &#8457.

Maintenant, redémarrez le serveur Django pour voir les modifications et vous assurer que l'application fonctionne comme prévu. Si vous avez modifié la configuration d'exécution comme expliqué ci-dessus, la fenêtre du navigateur doit s'ouvrir automatiquement :



III-2 - Créer une base de données et un modèle

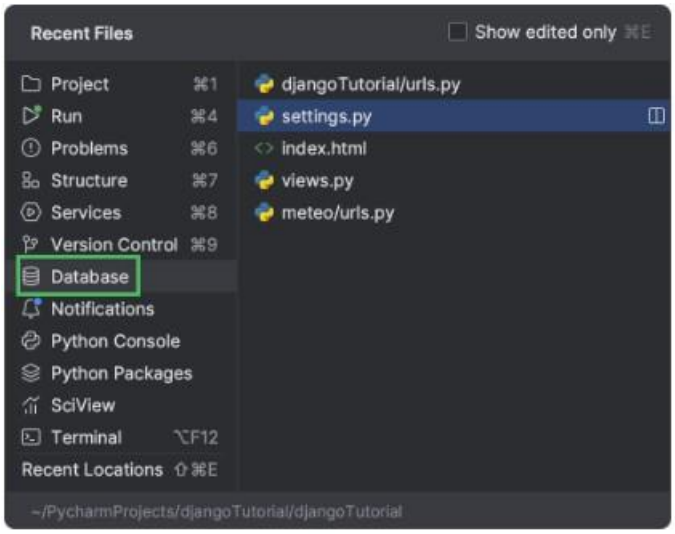
Pour obtenir de l'API les données sur la température à une certaine position, il faut lui fournir les coordonnées de cette position.

Nous allons utiliser la **base de données World cities** de **Juanma Hernández** avec la licence **CC BY 4.0**. Téléchargez la base de données et extrayez **worldcities.csv** de l'archive. Vous devez vous inscrire sur **Kaggle** si vous n'y avez pas de compte.

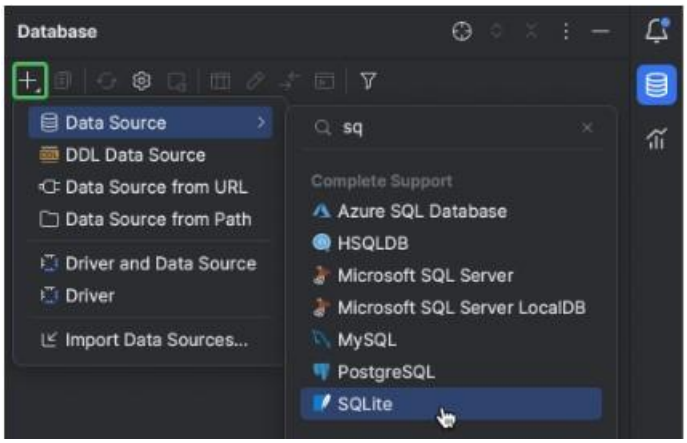
Nous devons également ajouter la base de données **db.sqlite3**, créée automatiquement par Django, à notre projet PyCharm.

Pour ce faire :

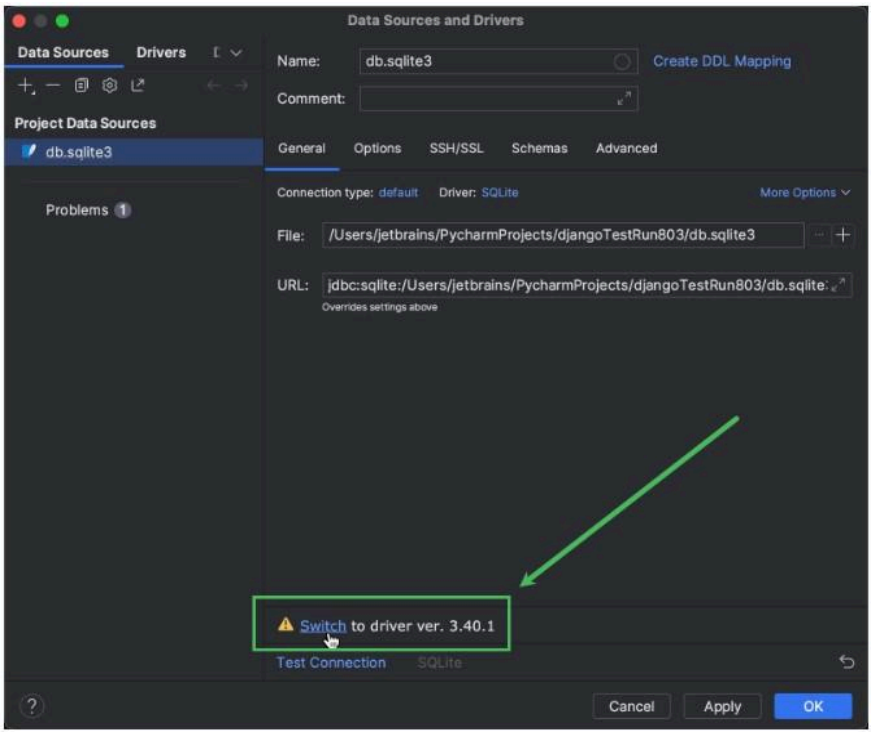
- Ouvrez la fenêtre d'outils **Database** en cliquant sur l'icône de base de données à gauche. Vous pouvez également accéder à l'ensemble des fenêtres d'outils en appuyant sur ⌘E **/ Ctrl+E** :



- Cliquez sur **+** dans le coin supérieur gauche de la fenêtre d'outils, puis sélectionnez **Data Source > SQLite**. Vous pouvez commencer à taper *sq*… pour atteindre l'option voulue plus vite :



- Cliquez sur … près du champ **File** et recherchez le fichier **db.sqlite3** dans votre dossier de projet.
- Il peut être nécessaire d'installer, de mettre à jour ou de changer le pilote de base de données. En cas d'avertissement en bas de la fenêtre, cliquez sur le lien pour exécuter l'action requise :

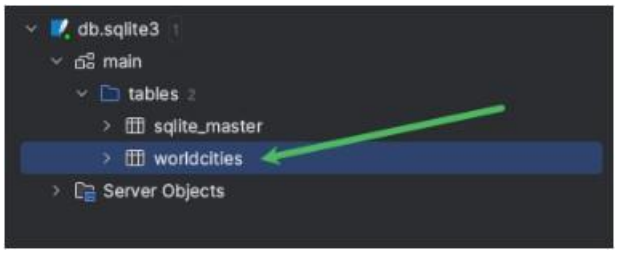


- Cliquez sur OK.

Pour importer des données dans la base, faites glisser et déplacez **worldcities.csv** vers **db.sqlite3** dans la fenêtre d'outils **Database**. Dans la boîte de dialogue qui s'ouvre, supprimez les colonnes inutiles pour ne garder que **city**, **lat**, **lng**, **country** et **id** :

Cliquer sur ce lien pour lancer l'animation

Désormais, la base de données **db.sqlite3** contient le tableau **worldcities** :



Vous pouvez faire un double-clic dessus pour en afficher le contenu dans l'éditeur.

Django utilise des modèles pour interagir avec les bases de données. Nous allons créer un modèle pour activer la lecture des données depuis la table **worldcities**.

- Lancez la console de tâches **manage.py** en appuyant sur ⌥R / **Ctrl+Alt+R**.
- Saisissez `inspectdb worldcities` et appuyez sur **Entrée**.
- Copiez la classe `Worldcities` de la sortie de la console dans **meteomodels.py**.
- Remplacez `null=True` par `primary_key=True` pour le champ id.

Vous devriez obtenir quelque chose comme cela :

```
1. from django.db import models
2. class Worldcities(models.Model):
```

```
3.      city = models.TextField(blank=True, null=True)
4.      lat = models.FloatField(blank=True, null=True)
5.      lng = models.FloatField(blank=True, null=True)
6.      country = models.TextField(blank=True, null=True)
7.      id = models.IntegerField(blank=True, primary_key=True)
8.      class Meta:
9.          managed = False
10.          db_table = 'worldcities'
```

III-3 - Ajouter des fonctionnalités

Jusqu'à présent, nous n'avions qu'une seule fonction (temp_here) dans **views.py** qui renvoyait des informations sur la température actuelle à notre position. Nous allons ajouter une autre fonction pour afficher la température de la position demandée. Elle doit également obtenir les données de température actuelle depuis l'API. Par conséquent, conformément aux bonnes pratiques de Python, nous devons déplacer cette fonctionnalité vers une fonction distincte.

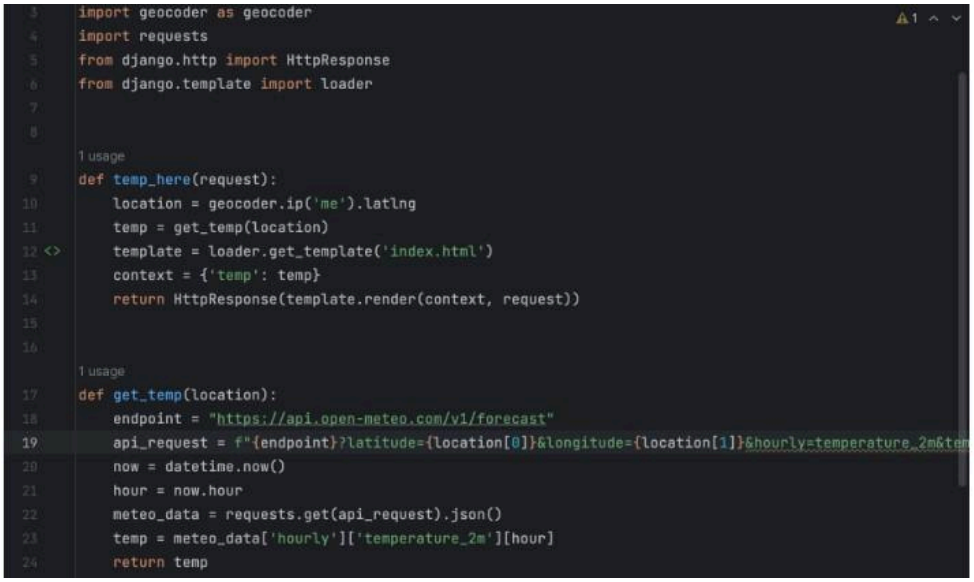
Dans PyCharm, cette opération se fait simplement avec la refactorisation **Extract method**.

Sélectionnez les lignes à déplacer vers une autre fonction et appuyez sur ⌥⌘M / **Ctrl+Alt+M**. Vous pouvez également utiliser **Find Action** :

Cliquer sur ce lien pour lancer l'animation

Spécifiez get_temp dans le champ **Method name** et cliquez sur OK. Nous obtenons ainsi une fonction get_temp qui accepte location comme seul paramètre.

La fonction temp_here l'appelle pour recevoir la température à la position actuelle.



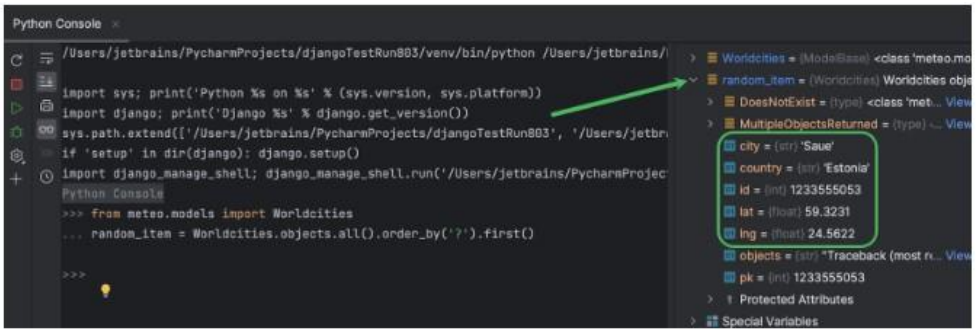
Nous allons créer la fonction temp_somewhere. Elle doit transférer les coordonnées d'une ville aléatoire à get_temp.

Commençons par un peu de prototypage dans la console Python. Ouvrez-la et collez le code suivant (si ImportError se produit, fermez et rouvrez la console) :

```
from meteo.models import Worldcities
random_item = Worldcities.objects.all().order_by('?').first()
```

Nous devons obtenir les données de la table **worldcities**. Pour ce faire, nous devons importer le modèle **Worldcities** depuis **models.py**.

Ensuite, nous allons randomiser tous les objets avec order_by('?') et obtenir le premier objet avec first(). L'onglet **Variables** se trouve dans à droite de la fenêtre d'outils **Python console**. Développez le nœud **random_item** pour voir le contenu de random_item :

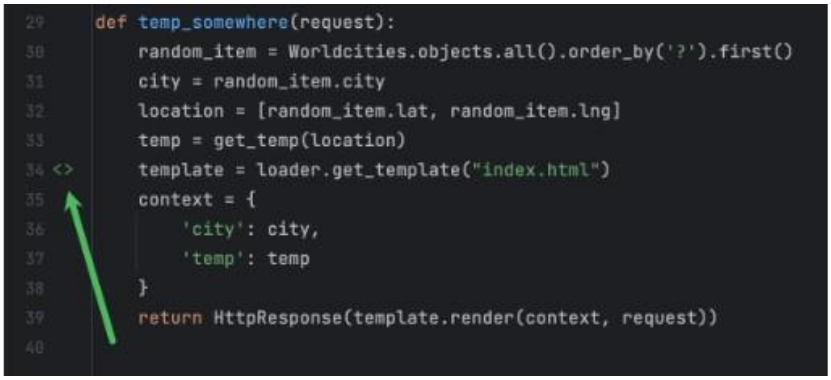


Le code complet de la fonction temp_somewhere devrait ressembler à cela :

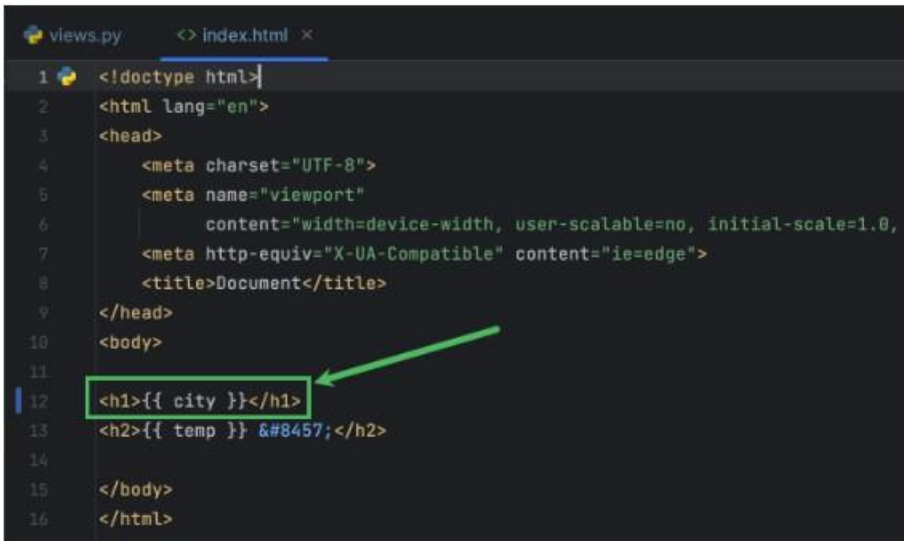
```
1. def temp_somewhere(request):
2.     random_item = Worldcities.objects.all().order_by('?').first()
3.     city = random_item.city
4.     location = [random_item.lat, random_item.lng]
5.     temp = get_temp(location)
6.     template = loader.get_template("index.html")
7.     context = {
8.         'city': city,
9.         'temp': temp
10.     }
11.     return HttpResponse(template.render(context, request))
```

En plus de la température, context contient maintenant le nom de la ville. Nous allons modifier le gabarit de façon à ce qu'il restitue aussi ces informations.

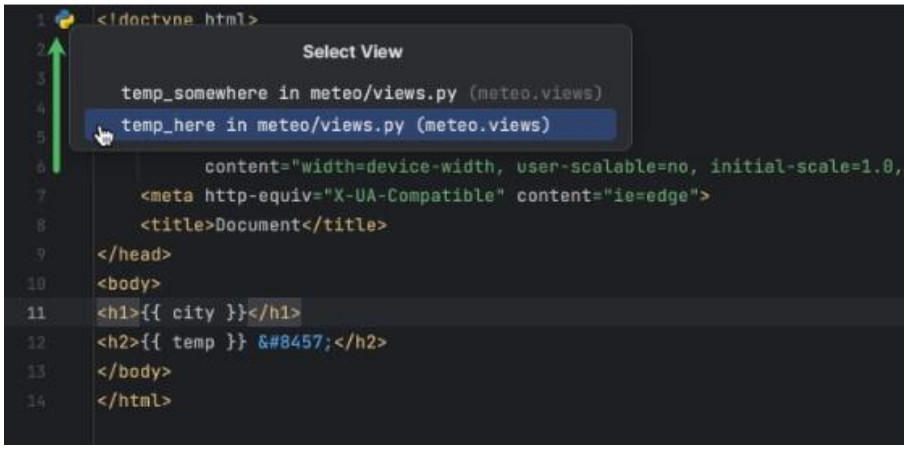
Vous avez peut-être remarqué l'icône <> dans la gouttière. Cliquez dessus pour passer rapidement à **index.html** :



Remplacez le texte entre les balises <h1></h1> par {{ city }} :



Cliquez sur l'icône de gouttière pour accéder rapidement à la fonction temp_here :



Nous devons la modifier de façon à ce qu'elle transfère également la variable city :

```
def temp_here(request):
    location = geocoder.ip('me').latlng
    temp = get_temp(location)
    template = loader.get_template("index.html")
    context = {
        'city': 'Your location',
        'temp': temp
    }
    return HttpResponse(template.render(context, request))
```

La fonction temp_here est appelée lorsque la page *https://127.0.0.1/meteo* est ouverte dans le navigateur, que nous avons configuré dans **meteo/urls.py**.

Revenez à cet endroit et spécifiez que temp_somewhere doit être appelé en cas d'accès à *https://127.0.0.1/meteo/discover* :

```
1. urlpatterns = [
```

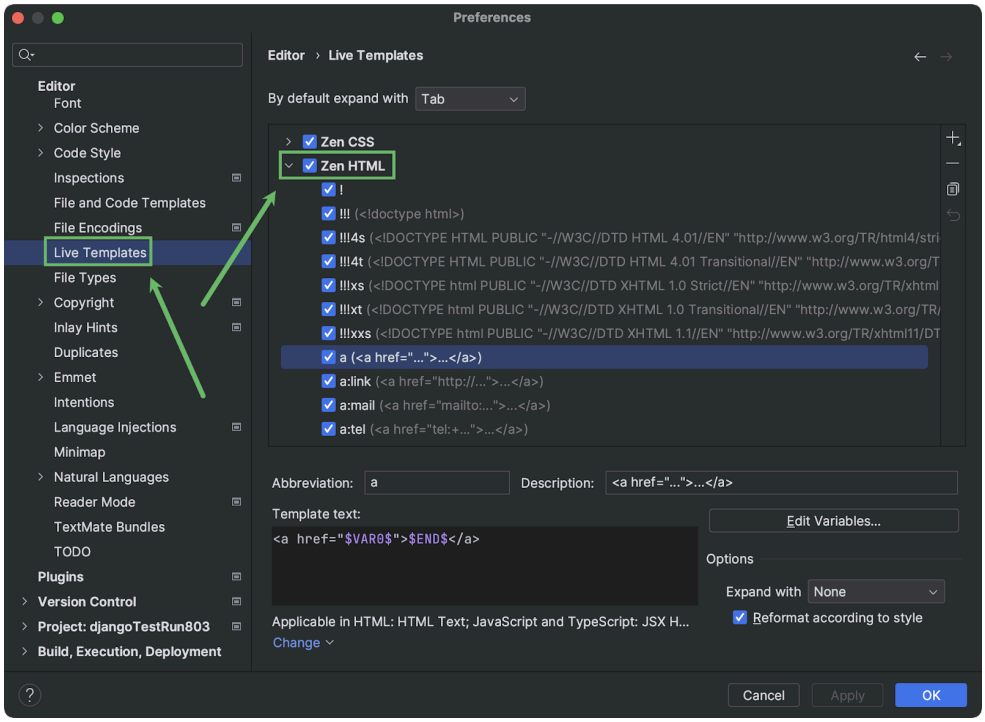
```
2.      path('meteo/', views.temp_here, name='temp_here'),
3.      path('meteo/discover', views.temp_somewhere, name='temp_somewhere'),
4. ]
```

Revenez à **index.html** et ajoutez des liens à la page **/discover** et à la page d'accueil. Cette dernière affichera toujours la température à la position actuelle. Les modèles dynamiques et la saisie semi-automatique du code permettent de gagner beaucoup de temps :

Cliquer sur ce lien pour lancer l'animation

Pour utiliser des modèles dynamiques pour les paragraphes et les liens dans des fichiers html, saisissez *p* ou *a*, puis appuyez sur la touche de **Tab**.

Pour voir la liste complète des modèles dynamiques disponibles, ouvrez les paramètres (⌘ / **Ctrl+Alt+S**), allez à **Editor > Live Templates**, puis développez le nœud **Zen HTML** dans le volet de droite :



Vous pouvez redémarrer à nouveau le serveur et vous amuser avec l'application.

IV - Ne vous arrêtez pas là

L'application est fonctionnelle, mais peut encore être améliorée. Le point d'amélioration le plus évident a priori est l'aspect visuel, mais il est aussi possible d'ajouter des fonctionnalités.

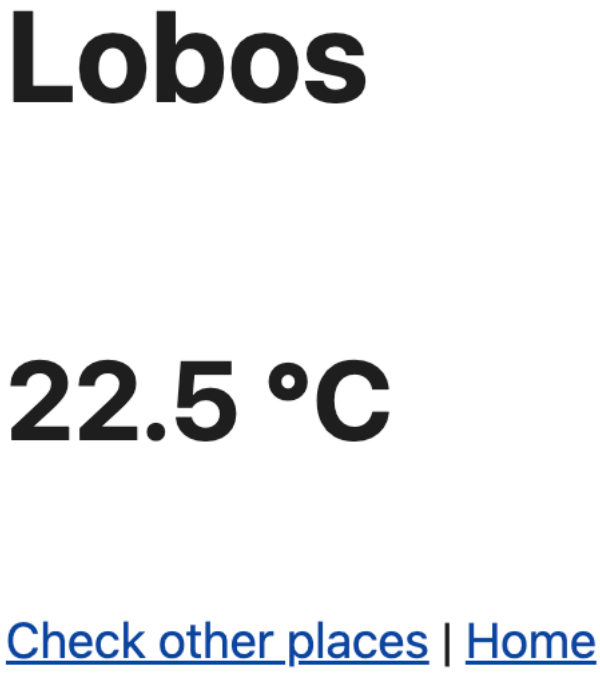
IV-1 - Utiliser CSS

L'utilisation de CSS est la solution la plus simple et rapide pour améliorer l'aspect de votre application Django. Vous pouvez par exemple utiliser **Simple.CSS**.

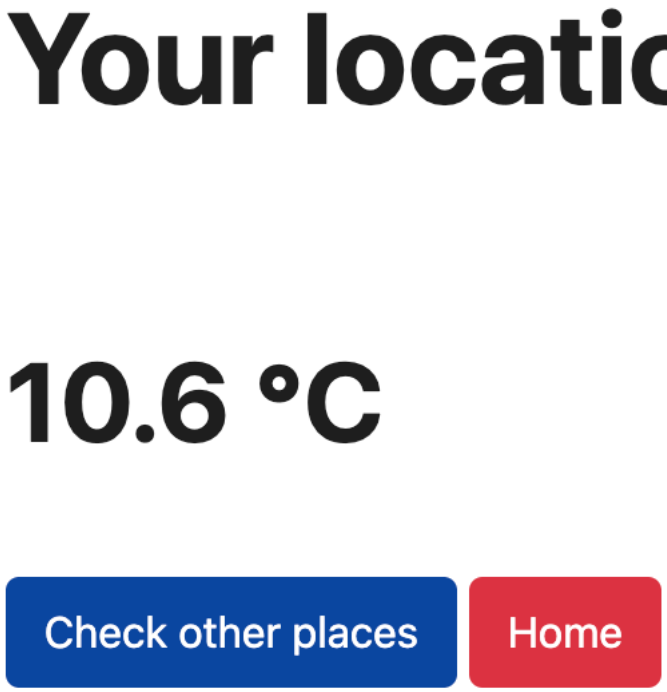
Placez la ligne suivante quelque part entre les balises <head></head> dans **index.html** :

```
<link rel="stylesheet" href="https://cdn.simplecss.org/simple.min.css">
```

Voici à quoi ressemble l'application ensuite :



Remplacer les liens par des boutons permet d'améliorer encore un peu plus la présentation :



Pour ce faire, remplacez les liens entre les balises dans **index.html** par le code html suivant :

```
<div style="display: flex;">
    <form action="./discover">
        <input type="submit" value="Check other places"/>
    </form>
     
    <form action=".">
        <input style="background: #dc3545" type="submit" value="Home"/>
    </form>
</div>
```

Vous pouvez voir les résultats de la modification de **index.html** sans avoir à quitter PyCharm pour aller dans un navigateur. Survolez le coin supérieur droit de la fenêtre de l'éditeur et sélectionnez **Built-in Preview** :

Cliquer sur ce lien pour lancer l'animation

IV-2 - Utiliser Bootstrap

Bootstrap est un outil frontend puissant et gratuit. Pour l'appliquer rapidement à votre application Django, modifiez **index.html** de la façon suivante (pour plus d'informations, consultez le **guide de prise en main de Bootstrap**) :

- Ajoutez la balise link dans la section head du fichier html (remplacez l'entrée simple.css insérée lors de l'étape précédente) :

```
<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/css/
bootstrap.min.css" rel="stylesheet" integrity="sha384-GLhlTQ8iRABdZLl6O3oVMWSktQOp6b7In1Zl3/
Jr59b6EGGoI1aFkw7cmDA6j6gD" crossorigin="anonymous">
```

2 Ajoutez le code suivant entre les balises <body></body> :

```
1. <div class="container text-center">
2.     <h1>{{ city }}</h1>
3.     <h2>{{ temp }} #</h2>
4.     <div class="btn-group" role="group">
5.         <button type="button" class="btn btn-outline-primary" onclick="location.href='./
discover'">Check other places
6.         </button>
7.         <button type="button" class="btn btn-danger" onclick="location.href='.'">Home</button>
8.     </div>
9. </div>
10. <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0-alpha1/dist/js/
bootstrap.bundle.min.js" integrity="sha384-w76AqPfDkMBDXo30jS1Sgez6pr3x5MlQ1ZAGC+nuZB
+EYdgRZgiwxhTBTkF7CXvN" crossorigin="anonymous"></script>
```

Regardez le résultat :

Bootstrap offre de nombreuses possibilités de personnalisation de vos applications. Vous trouverez toutes les informations nécessaires dans la **documentation**.

IV-3 - À vous de jouer

Nous avons ajouté le champ country provenant de la base de données **worldcities** au modèle Worldcities, mais n'avons pas encore utilisé ces données dans l'application. Vous pouvez maintenant tester vos compétences en essayant de créer une application qui affiche les du pays en plus des villes. Vous devriez obtenir un résultat similaire à ceci :

V - Résumé

Dans ce tutoriel, nous avons vu comment :

- créer des projets Django dans PyCharm ;
- créer des vues et des gabarits ;
- faire des appels d'API et traiter les réponses ;
- se connecter à des bases de données et importer des données ;
- améliorer l'aspect de l'application en utilisant CSS et Bootstrap.

VI - Remerciements Developpez.com

Nous tenons à remercier **Malick** pour la mise au gabarit et **Escartefigue** pour la relecture orthographique.