TP Django

1 Introduction

L'objectif de ce TP est de créer une interface au générateur d'emplois du temps (c.f. Semaine 7). Pour ce faire, nous allons créer un site web qui permette de lister des classes, des cours et des salles, puis qui génère et affiche un emplois du temps à l'aide du générateur de la séance 7.

Afin que tout le monde parte d'une base commune, les fichiers fournis contiennent déjà un générateur, qui ressemble *très fortement* à celui que vous avez écrit en séance 7.

Pour créer ce site web, nous n'allons pas tout faire à la main, ce qui serait beaucoup trop compliqué. À la place, nous allons utiliser *Django*, un utilitaire qui va se charger de la plupart du travail, et nous n'aurons qu'à compléter les parties spécifiques à notre site.

1.1 Présentation des fichiers

L'archive fournie contient *beaucoup* de fichiers, car elle contient tous les fichiers nécessaires pour paramétrer et servir le site web ainsi qu'un package python qui contient le générateur. La liste des fichiers fournis est montrée en Figure 1. Nous n'aurons pas à toucher à la majorité de ces fichiers. En effet, le site est déjà configuré et, dans ce TP, nous ne ferons que :

- Ajouter une nouvelle page
- Créer un template pour une page
- Ajouter un formulaire d'envoi de données
- Interfacer les données de la base de données et le générateur d'emplois du temps.

En pratique, en amont de ces étapes, vous auriez à configurer le site de manière générale, préparer la base de données, etc... J'ai déjà fait ces étapes pour vous.

Dans ce TP, nous travaillerons donc principalement sur deux fichiers et un dossier :

- Le fichier `WebEDT/edt/urls.py`, qui permet de déclarer quelle fonction appeler lorsqu'une page est demandée. Par exemple : lorsque l'utilisateur·ice demande la page `index.html` (la page d'accueil du site), on va écrire une fonction qui *génère*, à la volée, le fichier HTML (`index.html`) de cette page.
- Le fichier `WebEDT/edt/views.py`, qui contient justement ces fonctions. Par exemple, la fonction `index` est la fonction qui génère le fichier `index.html`.
- Toutefois, générer ex-nihilo des fichiers HTML est pénible, d'autant que le plus souvent, pour une page donnée, on va générer quasiment le même fichier à chaque fois. Par exemple, si on veut afficher la liste des cours, on va toujours avoir une liste, mais c'est le contenu de la liste qui va varier. Pour se simplifier la vie, on va utiliser des *templates*, qui sont essentiellement des fichiers HTML avec des trous à remplir, que l'on remplira à l'aide de variables Python. Ces templates sont dans le sous-dossier `WebEDT/edt/templates/edt/`.

1.2 Préparation du TP

Nous allons commencer par installer le package du générateur, qui se trouve dans le dossier `EdT_Solver` fourni. Pour y faire, rendez-vous simplement dans le sous-dossier `EdT_Solver` et lancez la commande :

```
$ pip install .
```

```
.
+-- EdT_Solver/     # Le package du Solver
+-- WebEDT
    +-- db.sqlite3  # Le fichier de base de données
    +-- edt
    |   +-- admin.py
    |   +-- apps.py
    |   +-- __init__.py
    |   +-- migrations/
    |   +-- models.py   # Le fichier contenant le modèle de la base de données
    |   +-- __pycache__/
    |   +-- templates   # Les templates de pages de réponses
    |   |   +-- edt
    |   |       +-- base.html
    |   |       +-- classe.html
    |   |       +-- cours.html
    |   |       +-- curriculum.html
    |   |       +-- edt.html
    |   |       +-- index.html
    |   |       +-- salles.html
    |   +-- tests.py
    |   +-- urls.py     # Le fichier indiquant les fonctions à appeler
    |   +-- views.py    # Le fichier contenant les fonctions à appeler
    +-- manage.py   # Le programme python à lancer
    +-- WebEDT/     # La configuration globale du site
```

Figure 1: Liste des fichiers fournis.

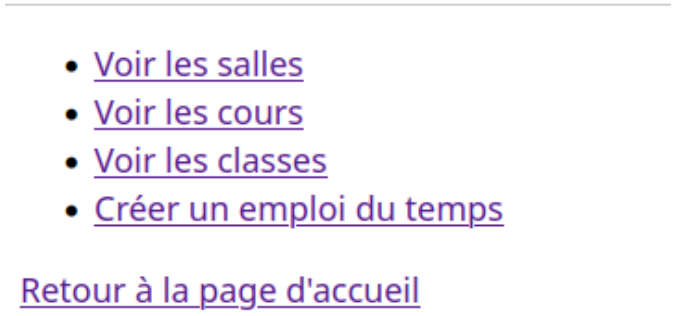


Figure 2: Page d'accueil du site de Génération d'emplois du temps.

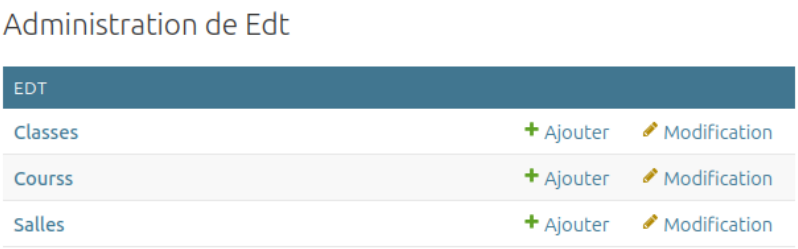


Figure 3: Page d'administration de la base de données.

1.3 Essai du site web

Une fois que les prérequis sont satisfaits, vous pouvez déjà lancer le site web, qui a déjà pas mal de choses en place :

```
$ cd /chemin/vers/archive/WebEDT/
$ python manage.py runserver
....
Django version 5.1.3, using settings 'WebEDT.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

Comme l'indique le message affiché, en lançant cette commande, un serveur Web est lancé sur votre ordinateur, à l'adresse : `127.0.0.1` (l'adresse locale de votre ordinateur[1]), sur le port 8000. En ouvrant votre navigateur et en tapant `http://127.0.0.1:8000/` dans la barre d'adresse, vous devriez donc arriver sur le site du générateur d'emplois du temps (voir Figure 2).

1.4 Base de données

Le site utilise de manière sous-jacente une base de données SQL (le fichier `db.sqlite3`). Nous ne le manipulerons pas directement, mais via Django. Toutefois, pour bien visualiser les choses, il peut être utile d'aller consulter son contenu. Django vous donne accès à une interface de visualisation (type administrateur·ice). Pour l'utiliser, rendez vous sur `http://127.0.0.1:8000/admin/` et utilisez les identifiants `admin`/`admin`. Voir Figure 3.

⚠ Il va sans dite que c'est une très mauvaise pratique d'utiliser `admin`/`admin` comme identifiants. Ne mettez surtout pas un tel site en ligne.

[1] `https://fr.wikipedia.org/wiki/Localhost`

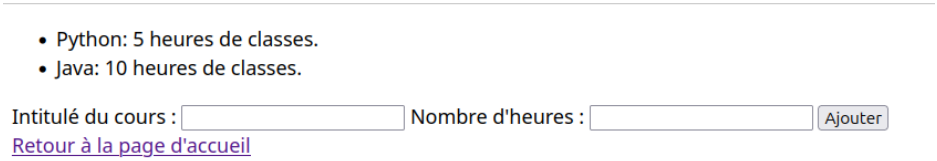


Figure 4: Exemple de page de gestion de cours. Dans la Section 2.2, nous nous chargerons de créer la liste en haut de la page. Dans la Section 2.3, nous nous chargerons de l'ajout du formulaire de la partie inférieure de la page.

2 Exercices

Le site fourni n'est pas complet. L'objectif de ce TP est de compléter les parties manquantes.

2.1 Ajouter une page web.

Les pages de gestion des salles et des classes sont déjà implémentées, et permettent respectivement de voir et d'ajouter des salles disponibles ou des classes. Cependant, la page de gestion des cours n'existe pas encore : si vous essayez de cliquer sur le lien, vous obtenez une erreur.

Remarque : Le site est paramétré en mode *debug*. Les erreurs que vous voyez sont donc verbeuse et contiennent beaucoup d'information, pour vous aider lors des exercices. En pratiques, en cas d'erreur, vous obtiendrez des erreurs HTTP classiques (e.g. 404 si vous tentez d'accéder à la page des cours, qui n'existe pas encore).

Remarque (Page du site) : Un site se compose de différentes pages, qui correspondent à chaque fois à un fichier HTML (statique ou généré à la volé) demandé par le navigateur au serveur. Le nom de ce fichier est ajouté à la suite de l'adresse : par exemple, si on tape `http://127.0.0.1:8000/salles` dans la barre de recherche, on demande au serveur le fichier (au format HTML) nommé `salles`.

Notre première étape est donc d'aller informer le programme que nous souhaitons ajouter une nouvelle page. Pour se faire, il faut aller compléter le fichier `urls.py`. Il va falloir rajouter une ligne pour indiquer qu'on veut rajouter une page à l'url `http://127.0.0.1:8000/cours`. Les lignes existantes devraient vous mettre sur la voie à suivre :

```
    path("classes", views.classe, name="classes"),
```

Le premier champ (`"classes"`) est le nom de la page à ajouter. Le second champs (`views.classe`) est le nom de la fonction qui va générer la page (ici, il s'agit donc de la fonction `classe` dans le module `views`). Le champs `name` permet de donner un nom à l'url créée, ce qui nous permettra plus tard d'utiliser cette url en la demandant par son nom, plutôt qu'en l'écrivant à la main.

Question 1 *Ajoutez une page `cours`, générée par la fonction `cours` (dans le fichier `views.py`).*
Je vous conseille de donner un nom à cette page, par exemple `"cours"`.

2.2 Créer et afficher une page

La fonction `cours` dans le fichier `views.py` existe déjà mais ne fait rien. À terme, cette page devra afficher les cours existants et permettre l'ajout de nouveaux cours, mais dans l'immédiat, on va se contenter de gérer l'affichage des cours existant.

La fonction `cours` prend un seul argument, une requête, qui correspond à la la requête HTTP reçue par le serveur de la part du client. Ici, nous n'avons pas besoin d'utiliser le contenu de cette requête, que nous pouvons donc ignorer.

La réalisation de cette nouvelle page va se faire en deux étapes. Dans un premier temps, nous allons créer un *template*, qui est une sorte de fichier HTML à trous. Dans un second temps, nous allons faire en sorte que Python remplisse ces trous pour former un fichier HTML valide.

Remarque : Nous parlons ici de créer un fichier HTML, mais il ne s'agit que d'un fichier temporaire, en mémoire, créé à la volée à chaque fois que la page du site est appelée. Il ne s'agit donc pas d'un fichier enregistré sur le disque ou visible par ailleurs.

Créer un template de page. Pour créer notre template, nous allons faire une hypothèse, qui est que nous disposons d'un *contexte*, qui contiendra les informations de chaque cours. En l'occurrence, nous aurons accès à une liste `cours`, qui contient des cours individuels; et chaque cours individuel a un champs `intitule_cours` et `quota_horaire`.

Le langage de template est très proche du langage HTML, mais avec certains ajouts qui vont permettre d'indiquer comment compléter les trous à l'aide du contexte.

Question 2 *Dans un premier temps, complétez le template `templates/edt/cours.html` afin d'insérer une liste vide. Vous pouvez utiliser les balises HTML `<ul>` et `</ul>`.*

Pour compléter la liste, nous allons utiliser un des ajouts du langage de template pour demander d'ajouter un élément à la liste pour chaque cours existant. Il s'agit de l'équivalent des boucles `for` de Python, mais dans le langage de templates.

Les lignes : `{% for XXX in YYY%}` et `{% endfor %}` permettent de créer une telle boucle for. Par ailleurs, il est possible d'insérer la valeur de variables du contexte en les encapsulant dans des doubles accolades : `{{ nom_de_variable }}`.

Remarque : Vous pouvez naturellement explorer les autres fichiers de templates et vous en inspirer pour comprendre comment utiliser les commandes et les variables.

Indice : le template de la page des classes est un bon choix.

Question 3 *Complétez la liste à l'aide d'une boucle `for` afin d'ajouter un item pour chaque cours (utilisez la liste `cours`). Vous pouvez utiliser les balises HTML `<li>` et `</li>` pour créer un item.*

Contexte de la page. Maintenant que le template est créé, il nous faut demander à la fonction `cours` du fichier `views.py` de l'utiliser pour générer le texte de la page à envoyer au client. L'enjeu principal est de créer le contexte dont on parlait en début de sous-section.

Un contexte est simplement un dictionnaire où les clefs sont les noms des variables que l'on utilise dans le template, et les valeurs sont les objets python associés. Dans notre cas, le contexte sera donc un dictionnaire avec une seule entrée, nommée `cours`, et qui sera associée à une liste de cours.

Pour trouver la liste de cours à mettre dans le contexte, nous allons devoir interroger la base de données, afin d'extraire les informations qui nous intéressent. Nous pourrions utiliser des commandes SQL à la main pour interroger la base de données; mais Django nous facilite la vie en le faisant pour nous.

Si nous regardons le fichier `models.py`, nous voyons qu'il définit des classes. Ces classes sont utilisées pour créer le schéma de la base de données. Par ailleurs, ces classes héritent de la classe `models.Model`, qui ajoute des fonctions pour accéder à la base de données.

Dans notre cas, nous voulons accéder aux cours. Nous allons donc demander à obtenir tous les objets stockés dans la table de Cours.

Remarque : Je vous invite à consulter le panneau d'administration du site, qui vous permet de visualiser l'état de la base de données : `http://127.0.0.1:8000/admin/`.

Question 4 *En vous inspirant de ce qui est fait pour les autres pages du site, créez le contexte du template de la page des `cours`.*

Utiliser le contexte pour afficher la page. Nous avons maintenant tous les éléments en main pour créer une page pour afficher les cours. Il ne reste plus qu'à demander à Django de compléter le template en utilisant le contexte. Le résultat doit être renvoyé par la fonction `cours(request)` que nous sommes en train d'écrire, et Django se chargera de la transmettre au client.

Question 5 *En utilisant la fonction `render()`*[2] *et en vous inspirant des fonctions déjà écrites, finissez d'écrire la fonction `cours()`.*

Testez le bon fonctionnement de votre code en vous rendant sur `http://127.0.0.1:8000/cours`

[2] `https://docs.djangoproject.com/fr/5.1/topics/http/shortcuts/#render`

2.3 Ajouter un formulaire et modifier la base de données

Dans ce qui précède, nous avons vu comment ajouter une nouvelle page pour envoyer des données (i.e. une page HTML) à un·e visiteur·se du site. Maintenant, nous aimerions faire l'inverse : que l'utilisateur·ice puisse nous envoyer des données, par exemple de nouveaux cours, de nouvelles salles, etc..

La base de code fournie contient déjà les éléments pour ajouter des salles et des classes, et nous allons vouloir gérer l'ajout de cours. Pour cela, on va ajouter un *formulaire* à la page de cours. Pour cela, il y a trois étapes à suivre :

- Préparer une nouvelle page qui recevra une requête avec les éléments du cours à ajouter;
- Insérer un nouveau cours dans la base de données en fonction de la requête;
- Dans la page de cours, ajouter le formulaire sur la page HTML, et effectuer une requête pour les envoyer.

Préparer une page pour recevoir la requête. La première étape consiste à créer une nouvelle page (on l'appellera `ajouter_cours`).

Question 6 *Suivez les étapes de la Section 2.1 pour ajouter cette page. La fonction à appeler est `ajout_cours`, et donnez-lui le nom `name="ajout_cours"`.*

Ceci étant fait, nous allons maintenant pouvoir travailler sur la fonction `ajout_cours`. La différence avec ce que nous avons fait jusqu'à présent est que nous allons utiliser la *requête* passée en argument lors de l'appel de la fonction.

Le protocole HTTP propose plusieurs types de requête (la plus connue étant *GET*, qui permet de demander une page). Nous allons ici utiliser un autre type de requête : les requêtes *POST*[3].

Pour se faire, Django va nous simplifier la vie en gérant tout l'aspect technique pour nous. De notre point de vue, notre fonction `ajout_cours` va recevoir un objet `request` en argument; cet objet contient les éléments de la requête *POST* envoyée par l'utilisateur·ice. Ainsi, de notre point de vue, une requête *POST* est simplement un dictionnaire (i.e. un ensemble de couples clefs-valeurs). Tout ce que nous avons à faire, c'est demander les éléments de ce dictionnaire qui nous intéressent.

Remarque : En pratique, il se peut que des éléments soient manquant (e.g. en cas de bug). Dans notre cas, nous ferons l'hypothèse que tout se passe bien.

L'objet `request` reçu en argument contient un champ `POST` qui est justement le dictionnaire évoqué à l'instant.

Question 7 *Extraire le titre du nouveau cours (clef `intitule_cours`) et son quota horaire (clef `quota_heures`) de la requête* POST.

Modifier la base de données pour ajouter un cours. Maintenant que nous avons les deux éléments qui caractérisent un cours, nous allons vouloir mettre à jour la base de données. De la même manière que dans la Section 2.2, Django nous facilite la vie et nous n'avons pas à manipuler la base de données à la main. À la place, nous pouvons simplement appeler le constructeur de `Cours` avec les paramètres adéquats.

La subtilité est que Django ne va mettre à jour la base de donnée immédiatement. L'idée est que lorsque nous créons un objet `Cours`, Django nous renvoie la requête SQL (encapsulée dans un objet) à faire sur la base de données. Pour lancer la requête, il suffit d'appeler la méthode `save`.

Remarque : Il ne faut pas confondre la requête *POST* reçue par le serveur Web, et transmise à la fonction `ajout_cours`; et la requête *SQL* que Django créé de manière transparente, et que nous utilisons. D'une certaine manière, la fonction `ajout_cours` *traduit* la requête *POST* en une requête *SQL*.

Remarque : La méthode `save` peut lancer une exception (e.g. les intitulés de cours doivent être uniques, donc une erreur est déclenchée si nous tentons d'ajouter un cours déjà existant). Il nous faut donc imbriquer le code d'ajout dans un bloc `try/except`.

Question 8 *En vous inspirant des fonctions déjà écrites, écrivez le corps de la fonction `ajout_cours` qui ajoute un cours à la base de données.*

[3] `https://en.wikipedia.org/wiki/POST_(HTTP)`

Ajouter un formulaire à la page de cours. Au point où nous sommes, le site web est prêt à recevoir des demandes d'ajout de cours. Cependant, rien n'est prévu pour qu'un·e visiteur·euse puisse créer la requête *POST* d'ajout[4]. Nous allons donc modifier la page de `cours` (c.f. Section 2.2), de manière à avoir un formulaire et émettre la requête (c.f. Figure 4, ainsi que les pages des salles et des classes, qui contiennent aussi des formulaires similaires).

Nous allons donc modifier le template `cours.html` pour rajouter la partie du code qui correspond à un formulaire. Dans un premier temps, il faut déclarer le formulaire à l'aide des balises `<form>` et `</form>`[5]. Nous devons ajouter deux attributs :

- l'`action`, i.e. à quelle page envoyer les données du formulaire (dans notre cas, il s'agit de la page `ajouter_page` que nous venons de créer); et
- la méthode (`method`), qui est la nature de la requête à faire (POST dans notre cas).

Remarque : Nous n'écrivons pas du HTML directement, mais un template de HTML. Ainsi, plutôt que d'entrer l'URL de la page, ce qui serait pénible à maintenir; nous pouvons simplement indiquer « l'URL de la page "`ajouter_page`" ». Il suffit d'indiquer : `{% url 'edt:nom_de_page' %}`.

Question 9 *Ajouter un formulaire vide dans le template `cours.html`.*

Remarque : Django fourni quelques outils basiques contre d'éventuelles attaques. Même si ce n'est pas magique et que ça ne fait pas tout, c'est une bonne pratique de les utiliser.

Il est donc recommandé d'ajouter `{% csrf_token %}` en début de formulaire, qui permet de limiter les attaques de type *cross site*[6].

Il nous reste maintenant à remplir le formulaire. Nous voulons ajouter deux champs de texte (un pour l'intitulé du cours, et l'autre pour le quota horaire), ainsi qu'un bouton de validation.

Concernant les champs de texte, nous allons utiliser les balises `<input>` et `</input>`[7]. Le contenu des champs de texte seront les valeurs contenues dans la requête. Pour indiquer la clef associée, il faut spécifier le *nom* de chaque champ, à l'aide de l'attribut `name`. Il est possible d'intégrer les `inputs` dans des `labels`[8] pour ajouter un texte qui précède le champ de texte.

Concernant le bouton, il suffit d'utiliser les balises `<button>` et `</button>`[9].

Question 10 *En vous inspirant des autres pages, ajouter le formulaire d'ajout de cours.*

Testez le bon fonctionnement de votre code en ajoutant des cours, et en vérifiant qu'ils apparaissent bien sur la page et/ou dans l'interface administrateur·ice.

2.4 Utiliser la base de données pour générer un emploi du temps

⚠ Cette section est plus dure et moins dirigée que les précédentes. Pas de panique ! Procédez méthodiquement, étape par étape, et si vous avez des questions, n'hésitez pas à demander.

Nous avons maintenant un site qui permet d'ajouter des classes, des salles et des cours. Il nous reste maintenant à générer des emplois du temps en fonction des cours auxquels est inscrite chaque classe. Pour se simplifier la vie, on ne va pas enregistrer les inscriptions dans la base de données. À la place, on va afficher une page où l'utilisateur·ice aura à cocher les cases pour faire les inscriptions (c.f. Figure 5), puis un emploi du temps sera généré et affiché à la volée (c.f. Figure 6).

Remarque : Dans la vraie vie, on aimerait sans doute sauvegarder les inscriptions et les emplois du temps, etc.; on ne va pas se préoccuper de ces problèmes dans notre cas.

Comme pour les sections précédentes, il nous faut donc :

1. créer une page avec un formulaire (c.f. Figure 5), nous appellerons cette page `curriculum`;

[4] Bien sûr, cette requête peut être créée à la main, mais c'est pas très pratique...
[5] `https://developer.mozilla.org/fr/docs/Learn/Forms/Your_first_form`
[6] `https://docs.djangoproject.com/fr/5.1/ref/csrf/`
[7] `https://developer.mozilla.org/fr/docs/Web/HTML/Element/input`
[8] `https://developer.mozilla.org/fr/docs/Web/HTML/Element/label`
[9] `https://developer.mozilla.org/fr/docs/Web/HTML/Element/button`

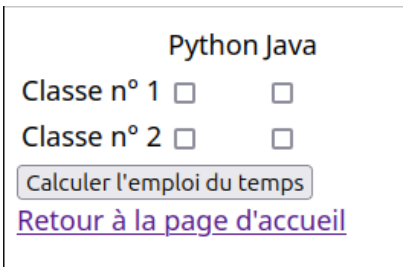
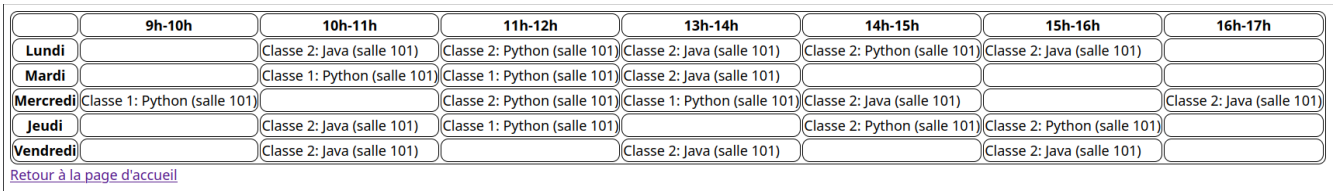


Figure 5: Page d'inscription des classes aux cours.



	9h-10h	10h-11h	11h-12h	13h-14h	14h-15h	15h-16h	16h-17h
Lundi		Classe 2: Java (salle 101)	Classe 2: Python (salle 101)	Classe 2: Java (salle 101)	Classe 2: Python (salle 101)	Classe 2: Java (salle 101)	
Mardi		Classe 1: Python (salle 101)	Classe 1: Python (salle 101)	Classe 2: Java (salle 101)			
Mercredi	Classe 1: Python (salle 101)		Classe 2: Python (salle 101)	Classe 1: Python (salle 101)	Classe 2: Java (salle 101)		Classe 2: Java (salle 101)
Jeudi		Classe 2: Java (salle 101)	Classe 1: Python (salle 101)		Classe 2: Python (salle 101)	Classe 2: Python (salle 101)	
Vendredi		Classe 2: Java (salle 101)		Classe 2: Java (salle 101)		Classe 2: Java (salle 101)	

Retour à la page d'accueil

Figure 6: Un emploi du temps généré à la volée.

2. faire en sorte que le formulaire de `curriculum` envoie une requête POST à une seconde page (que nous appellerons `calculer_edt`);
3. dans la fonction correspondant à cette seconde page, il faut récupérer le contenu de la requête POST, la donner au calculateur d'emplois du temps (c.f. Semaine 7);
4. demander au solveur de trouver une solution, créer un template d'emploi du temps, et le remplir à l'aide de la solution trouvée.

Les étapes 1, 2 et 4 sont déjà faites pour vous. Il ne vous reste plus qu'à modifier la fonction `calculer_edt` dans `views.py` pour créer une instance du solveur, lui donner toutes les informations nécessaires et obtenir une solution.

Solveur d'emploi du temps fourni.

Instance de solveur. Lors de la préparation du TP (c.f. Section 1.2), nous avons installé un paquet `EdT_Solver`. Ce paquet est essentiellement le corrigé du TP 7, que vous connaissez donc normalement très bien. Il y a toutefois quelques différences mineures : plutôt que d'avoir l'ajout des cours, salles, classes, etc. dans un script, le paquet vous fourni une classe `Instance`, qui représente une instance de solveur. En l'utilisant, vous pouvez ajouter les cours, classes, salles, etc. un par un, puis demander à résoudre le problème :

- Pour ajouter une salle, utilisez la méthode `solver.ajouter_salle(numéro_de_salle)`;
- Pour ajouter un cours, utilisez la méthode `solver.ajouter_cours(intitulé, quota_horaire)`;
- Pour ajouter une classe, utilisez la méthode `solver.ajouter_classe(liste_de_cours)`. Attention, il faut ajouter les classes dans l'ordre (i.e. commencer par la classe 1, puis la 2, etc.).

Solution. Par ailleurs, dans le TP 7, je vous fournissais une fonction `afficher_solution` qui permettait d'afficher une chaîne de caractère qui dessinait l'emploi du temps généré. Ici, je vous fourni la fonction `build_timetable`, qui vous renvoie une structure directement utilisable dans le contexte utilisé par le template de `calculer_edt`. Ainsi, le code fourni se charge déjà de calculer la solution et de compléter le template :

```
def calculer_edt(request):
⟶solver = edt_solver.Instance(5, 7) # 5 jours, 7h par jour
⟶
```

```
    # Ajouter tous les cours et salles à l'instance,
    # puis utiliser la requête POST pour inscrire les classes aux bons cours

    solution = solver.resoudre()
    timetable = edt_solver.affichage.build_timetable(solution, solver, ensemble_cours, ensem
    context = {
        "solution": timetable,
    }
    return render(request, "edt/edt.html", context)
```

Requête POST. La page `curriculum`, qui contient le formulaire d'inscription, envoie une requête POST que vous aurez à manipuler. Chaque bouton à cocher est nommé « `classe-cours` », où `classe` est le nom de la classe et `cours` est le nom du cours. Si le bouton est coché, la requête contiendra `on` pour la clef correspondante. Par exemple, si la classe « 1 » est inscrite au cours de python, alors `request.POST["1-Python"] == "on"`; et si la classe n'est pas inscrite, alors la clef n'est pas présente.

⚠ Je vous conseille d'utiliser la méthode `get` de la requête, qui renvoie `None` si on demande une clef absente :

```
if request.POST.get("1-Python") == "on":
    # la classe est inscrite
else:
    # la classe n'est pas inscrite
```

Question 11 *Complétez le code de la fonction `calculer_edt` et décrivez brièvement votre approche. Testez le bon fonctionnement de code.*

Solution:

3 Évaluation

Afin d'améliorer ce TP pour les années suivantes, merci de répondre à ces quelques questions.

1. Combien de temps avez-vous passé sur cette série d'exercices ?

   1. _____

2. Sur une échelle de 1 à 10, avez-vous trouvé que c'était trop court (1), trop long (10), ou adéquat (5) ?

   2. _____

3. Est-ce que l'intérêt pédagogique de cet série d'exercice est clair ?

   Solution:

4. Avez-vous été bloqué·e à un moment ? Si oui, précisez à quelle question.

   Solution:

5. Y a-t-il des questions/phrases qui n'étaient pas claires ? Pour lesquelles vous ne compreniez pas ce qui était attendu ?

   Solution:

6. À titre personnel, trouvez-vous le sujet intéressant ? Avez-vous des suggestions pour l'année prochaine ?

   Solution: