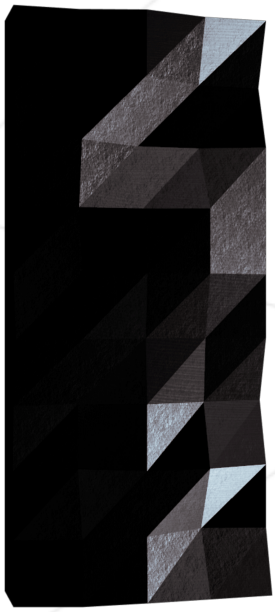

D02 - Formation Python-Django

Bases Python 2

Vincent Montécot vmonteco@student.42.fr
42 Staff pedago@staff.42.fr

Résumé: Aujourd'hui, vous allez partir à la conquête de la Silicon Valley grâce à vos nouvelles compétences en POO avec Python !

Table des matières

Chapitre I

Préambule

Voici les paroles de la “Free Software Song” :

Join us now and share the software ;
You’ll be free, hackers, you’ll be free.
Join us now and share the software ;
You’ll be free, hackers, you’ll be free.

Hoarders can get piles of money,
That is true, hackers, that is true.
But they cannot help their neighbors ;
That’s not good, hackers, that’s not good.

When we have enough free software
At our call, hackers, at our call,
We’ll kick out those dirty licenses
Ever more, hackers, ever more.

Join us now and share the software ;
You’ll be free, hackers, you’ll be free.
Join us now and share the software ;
You’ll be free, hackers, you’ll be free.

Chapitre II

Consignes

- Seule cette page servira de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- Si aucune information contraire n'est explicitement présente, vous devez partir du principe que les versions des langages et framework utilisés sont les suivantes (ou ultérieures) :
  - `Python 3`
  - `HTML5`
  - `CSS 3`
  - `Javascript EM6`
  - `Django 1.9`
  - `psycopg2 2.6`
- Sauf indication contraire dans le sujet, les fichiers en python de chaque exercice sur `Python` seul (d01, d02 et d03) doivent comporter à leur fin un bloc `if __name__ == '__main__'`: afin d'y insérer le point d'entrée dans le cas d'un programme, ou des tests dans le cas d'un module.
- Sauf indication contraire dans le sujet, chaque exercice des journées portant sur `Django` aura sa propre application dans le projet à rendre pour des raisons pédagogiques.
- Les exercices sont très précisément ordonnés du plus simple au plus complexe. En aucun cas nous ne porterons attention ni ne prendrons en compte un exercice complexe si un exercice plus simple n'est pas parfaitement réussi.
- Attention aux droits de vos fichiers et de vos répertoires.
- Vous devez suivre la procédure de rendu pour tous vos exercices : seul le travail présent sur votre dépot GIT sera évalué en soutenance.
- Vos exercices seront évalués par vos camarades de piscine.
- Vous ne devez laisser dans votre répertoire aucun autre fichier que ceux explicitement specifiés par les énoncés des exercices.
- Sauf indication contraire dans le sujet vous ne devez pas inclure dans votre rendu :

  - Les dossiers `__pycache__`.
  - Les éventuelles migrations.
  - Le dossier créé par la commande `collectstatic` de `manage.py` (avec pour chemin la valeur de la variable `STATIC_ROOT`).
  - Les fichier en bytecode Python (Les fichiers avec une extension en `.pyc`).
  - Les fichiers de base de donnée (notamment avec `sqlite`).
  - Tout fichier ou dossier devant ou pouvant être créé par le comportement normal du travail rendu.

  Il vous est recommandé de modifier votre `.gitignore` afin d'éviter les accidents.

- Lorsque vous devez obtenir une sortie précise dans vos programmes, il est bien entendu interdit d'afficher une sortie précalculée au lieu de réaliser l'exercice correctement.
- Vous avez une question ? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Votre manuel de référence s'appelle `Google / man / Internet / ...`.
- Pensez à discuter sur le forum Piscine de votre Intra !
- Lisez attentivement les exemples. Ils pourraient bien requérir des choses qui ne sont pas autrement précisées dans le sujet...
- Par pitié, par Thor et par Odin ! Réfléchissez nom d'une pipe !

Chapitre III

Règles spécifiques de la journée

- Aucun code dans le scope global. Faites des fonctions !
- Sauf indication contraire, tous vos fichiers écrits en Python devront se terminer par un bloc

```
if __name__ == '__main__':
    # Your tests and your error handling
```

- Toute exception non catchée invalidera le travail, même dans un cas d'erreur qu'on vous demande de tester.
- Aucun import autorisé, à l'exception de ceux explicitement mentionnés dans la section 'Fonctions Autorisées' du cartouche de chaque exercice..

Chapitre IV

Exercice 00

	Exercice : 00
Exercice 00 : À la conquête de la Silicon Valley !	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `render.py, myCV.template, settings.py`	
Fonctions Autorisées : `n/a`	
Remarques : `n/a`	

Vous avez fini votre super formation de développeur et une nouvelle vie pleine de perspectives s'offre à vous. Arrivé à la Silicon Valley, vous n'avez qu'une idée en tête : développer votre idée de générateur de CV révolutionnaire à l'aide d'une technologie avant-gardiste, et devenir le nouveau Bill Gates de la recherche d'emploi.

Il ne reste plus qu'à développer la technologie.

Réalisez un programme `render.py` qui prendra un fichier avec une extension `.template` en paramètre. Ce programme devra prendre le contenu du fichier, remplacer certains motifs par des valeurs définies dans un fichier `settings.py` (La présence d'un bloc `if __name__ == '__main__':` n'est pas nécessaire pour ce fichier) et écrire le résultat dans un fichier avec pour extension `.html`.

L'exemple suivant *devra* pouvoir être exactement reproduit avec votre programme.

```
$> cat settings.py :
name = duoquadragintian
$> cat file.template :
<p>''-Who are you?
-A {name}!''</p>
$> python3 render.py file.template
$> cat file.html :
<p>''-Who are you?
-A duoquadragintian!''</p>
```

Les erreurs, notamment une mauvaise extension de fichier, un fichier n'existant pas ou un mauvais nombre d'arguments, devront être gérées.

Vous devez rendre un fichier `myCV.template` qui, une fois converti en fichier HTML, devra au moins contenir la structure complète d'une page (`doctype`, `head` et `body`), le titre de la page, le nom et le prénom du propriétaire du CV, son âge, et sa profession. Bien entendu, ces informations ne devront pas apparaître directement dans le fichier `.template`.

Chapitre V

Exercice 01

	Exercice : 01
Exercice 01 : Startup innovante cherche stagiaire. 10 ans d'expérience requis.	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `intern.py`	
Fonctions Autorisées :	
Remarques : `n/a`	

Vous ne pouvez pas vous lancer seul dans une telle aventure. Vous décidez de recruter quelqu'un pour ~~faire le café~~ vous assister, un stagiaire de préférence (c'est moins cher).

Réalisez la classe `Intern` contenant les fonctionnalités suivantes :

`Un constructeur` prenant une chaine de caractère en paramètre et assignant sa valeur un attribut `Name`. Une valeur par défaut `"My name? I'm nobody, an intern, I have no name."` sera implémentée.

Une méthode `__str__()` qui retournera l'attribut `Name` de l'instance.

Une classe `Coffee` avec une simple méthode `__str__()` qui retournera la chaîne de caractère `"This is the worst coffee you ever tasted."`.

Une méthode `work()` qui lèvera juste une exception (utilisez le type `(Exception)` de base) avec pour texte `"I'm just an intern, I can't do that..."`.

Une méthode `make_coffee()` qui retournera une instance de la classe `Coffee` que vous aurez implémentée dans la classe `Intern`.

Dans vos tests, vous devez instancier deux fois la classe `Intern`, une fois sans nom, et une autre fois avec pour nom "Mark".

Affichez le nom de chacune des instances. Demandez à Mark de vous faire un café et affichez le résultat. Demandez à l'autre stagiaire de travailler. Vous **devez** gérer l'exception dans votre test.

Chapitre VI

Exercice 02

	Exercice : 02
Exercice 02 : 5 classes 1 cup.	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `beverages.py`	
Fonctions Autorisées :	
Remarques : `n/a`	

Le café c'est bien mais à la longue c'est un peu lassant de ne pas avoir plus de choix. Réalisez une classe `HotBeverage` avec les fonctionnalités suivantes :

Un attribut `Price` d'une valeur de `0.30`.

Un attribut `Name` avec pour valeur `"hot beverage"`.

Une méthode `description()` retournant une description de l'instance. La valeur de la description sera `"Just some hot water in a cup."`.

Une méthode `__str__()` retournant une description de l'instance sous cette forme :

```
name : <name attribute>
price : <price attribute limited to two decimal points>
description : <instance's description>
```

par exemple l'affichage d'une instance de HotBeverage donnerait :

```
name : hot beverage
price : 0.30
description : Just some hot water in a cup.
```

Réalisez ensuite les classes dérivées de `HotBeverage` suivantes :

`Coffee` :

`name` : `"coffee"`

`price` : `0.40`

`description` : `"A coffee, to stay awake."`

```
Tea :
  name : "tea"
  price : 0.30
  description : "Just some hot water in a cup."
Chocolate :
  name : "chocolate"
  price : 0.50
  description : "Chocolate, sweet chocolate..."
Cappuccino :
  name : "cappuccino"
  price : 0.45
  description : "Un po' di Italia nella sua tazza!"
```

Vous ne devez redéfinir QUE ce qui est nécessaire. (cf. DRY)

Instanciez dans vos tests chacune des classes parmi : `HotBeverage`, `Coffee`, `Tea`, `Chocolate` et `Cappuccino` et affichez les.

Chapitre VII

Exercice 03

	Exercice : 03
Exercice 03 : Glorious coffee machine !	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `machine.py, beverages.py`	
Fonctions Autorisées : `random`	
Remarques : `n/a`	

Ça y est, votre compagnie est lancée ! Vous avez maintenant un local acquis grâce à votre première levée de fonds, un stagiaire pour faire le café et une plante verte de niveau 10 à l'entrée du bâtiment pour garder le tout.

Cependant il faut bien l'avouer : le café produit par votre stagiaire est infect et un demi SMIC par mois c'est un peu cher pour du jus de chaussette. C'est le moment d'investir dans du nouveau matériel nécessaire à votre réussite professionnelle !

Réalisez la classe `CoffeeMachine` contenant :

- Un constructeur.
- Une classe `EmptyCup` héritant de `HotBeverage`, avec pour nom `"empty cup"`, comme prix `0.90` et comme description `"An empty cup?! Gimme my money back!"`. Copiez le fichier `beverages.py` de l'exercice précédent dans le dossier de cet exercice pour pouvoir utiliser les classes qu'il contient.
- Une classe `BrokenMachineException` héritant de `Exception` avec pour texte `"This coffee machine has to be repaired."`. Ce texte doit être défini dans le constructeur de l'exception.
- Une méthode `repair()` qui répare la machine pour lui permettre de servir à nouveau des boissons chaudes.
- Une méthode `serve()` qui aura les caractéristiques suivantes :

  Paramètres : Un unique paramètre (autre que `self`) qui sera une classe dérivée de `HotBeverage`.

  Retour : Une fois sur deux, la méthode retourne une instance de la classe passée

en paramètre, et une fois sur deux une instance de `EmptyCup`.

Obsolescence : La machine n'est pas de la meilleure qualité et tombe donc en panne après 10 boissons servies.

En cas de panne : l'appel de la méthode `serve()` doit provoquer la levée d'une exception de type `CoffeeMachine.BrockenException` jusqu'à-ce que la méthode `repair()` soit appelée.

Réparation : Après un appel de la méthode `repair()`, la méthode `serve()` peut de nouveau fonctionner sans levée d'exception pendant un cycle de 10 boissons, avant de retomber en panne.

Dans vos tests, instanciez la classe `CoffeeMachine`. Demandez diverses boissons venant du fichier `beverages.py` et affichez la boisson que vous sert la machine jusqu'à-ce qu'elle tombe en panne (vous *devez* gérer l'exception alors levée). Réparez la machine et recommencez jusqu'à-ce que la machine tombe de nouveau en panne (gérez à nouveau l'exception).

Chapitre VIII

Exercice 04

	Exercice : 04
Exercice 04 : Une classe de base ft. RMS.	
Dossier de rendu : $ex04/$	
Fichiers à rendre : `elem.py, text.py`	
Fonctions Autorisées :	
Remarques : `n/a`	

Il est maintenant temps d'améliorer votre présence sur le WEB. Vous aimeriez bien utiliser vos nouvelles connaissance en Python pour modéliser efficacement votre contenu HTML mais vous aimeriez recevoir le conseil d'un être supérieur pour savoir comment faire. Vous décidez d'offrir votre stagiaire en sacrifice aux dieux de la programmation.

Maintenant que vous avez une machine pour faire le café, vous n'avez plus vraiment besoin de lui... Vous l'immolez donc.

Saint IGNUcius vous apparait alors pour vous faire une révélation :

> *"Les éléments HTML partagent peu ou prou la même structure (balise, contenu, attributs). Il serait judicieux de réaliser une classe capable de rassembler tous ces comportements et caractéristiques communes pour ensuite utiliser la puissance de l'héritage en Python pour dériver facilement et simplement cette classe sans avoir à tout réécrire."*

C'est alors que St. IGNUcius aperçoit le Mac sur lequel vous travaillez. Prenant peur, il s'enfuit sans vous donner plus de détails, ne laissant derrière lui qu'un fichier de tests. Sans hésitation, vous réalisez la classe `Elem` avec les caractéristiques suivantes :

- Un constructeur pouvant prendre en paramètre le nom de l'élément, ses attributs `HTML`, son contenu et le type d'élément (balises simples ou doubles).
- Une méthode `__str__()` retournant le code `HTML` de l'élément.
- Une méthode `add_content()` permettant d'ajouter des éléments à la fin du contenu.

Si vous réalisez bien votre travail, il vous sera possible de représenter n'importe quel élément `HTML` et son contenu avec votre classe `Elem`. Dernière ligne droite :

- le fichier `tests.py` fourni dans la tarball en annexe du sujet **doit** fonctionner correctement (pas d'erreur d'assertion, la sortie du test annonçant explicitement son succès). Evidemment, nous ne sommes pas assez cruels pour tester des fonctionnalités qui ne sont pas explicitement réclamées dans cet exercice. Hahaha... Non, nous ne le sommes pas, je vous assure.
- Vous devrez également reproduire et afficher la structure suivante à l'aide de votre classe `Elem` :

```
<html>
  <head>
    <title>
      "Hello ground!"
    </title>
  </head>
  <body>
    <h1>
      "Oh no, not again!"
    </h1>
    <img src="http://i.imgur.com/pfp3T.jpg" />
  </body>
</html>
```

Chapitre IX

Exercice 05

	Exercice : 05
Exercice 05 : Faites vos propres éléments !	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `elem.py, elements.py`	
Fonctions Autorisées :	
Remarques : `n/a`	

Félicitations ! Vous êtes maintenant capable de générer n'importe quel élément HTML et son contenu. Cependant, c'est un peu lourd de générer chaque élément en précisant à chaque fois chaque attribut à chaque instanciation. C'est l'occasion d'utiliser l'héritage pour faire d'autres petites classes plus simples d'utilisation. Réalisez les classes suivantes dérivées de votre classe `Elem` de l'exercice précédent :

- `html`, `head`, `body`
- `title`
- `meta`
- `img`
- `table`, `th`, `tr`, `td`
- `ul`, `ol`, `li`
- `h1`
- `h2`
- `p`
- `div`
- `span`
- `hr`
- `br`

Le constructeur de chaque classe devra pouvoir prendre le contenu en premier argument, ainsi :

```
print( Html( [Head(), Body()] ) )
```

affichera bien :

```
<html>
  <head></head>
  <body></body>
</html>
```

Soyez malin et réutilisez les fonctionnalités que vous avez codées dans la classe Elem. Vous **devez** utiliser l'héritage.

Démontrez le fonctionnement de ces classes par des tests de votre choix en nombre suffisant pour couvrir toutes les fonctionnalités. Après avoir codé ces classes, vous n'aurez plus besoin de préciser le nom ou le type d'un tag, ce qui est très pratique. Vous ne devez donc plus jamais instancier directement votre class `Elem`, c'est desormais **interdit**.

Pour vous permettre de bien comprendre l'avantage des classes dérivées `d'Elem` par rapport à l'utilisation directe `d'Elem`, reprenons la structure du document `HTML` de l'exercice précédent. Vous devez le reproduire en utilisant vos nouvelles classes.

```
<html>
  <head>
    <title>
      "Hello ground!"
    </title>
  </head>
  <body>
    <h1>
      "Oh no, not again!"
    </h1>
    <img src="http://i.imgur.com/pfp3T.jpg" />
  </body>
</html>
```

C'est beaucoup plus simple, non ? :)

Chapitre X

Exercice 06

	Exercice : 06
Exercice 06 : Validation	
Dossier de rendu : $ex06/$	
Fichiers à rendre : `Page.py, elem.py, elements.py`	
Fonctions Autorisées :	
Remarques : `n/a`	

Malgré de réels progrès dans votre travail, vous aimeriez bien que tout soit un peu plus propre. Un peu plus cadré, vous êtes comme ça : vous aimez les contraintes et les défis. Alors pourquoi ne pas imposer une norme à la structure de vos documents `HTML` ? Commencez par copier les classes des deux exercices précédents dans le dossier de cet exercice.

Créez une classe `Page` dont le constructeur doit prendre en paramètre une instance d'une classe héritant de `Elem`. Votre classe `Page` doit implémenter une méthode `isvalid()` qui doit renvoyer `True` si toutes les règles suivantes sont repectées, et `False` sinon :

- Si pendant le parcours de l'arbre, un noeud n'est pas de type `html`, `head`, `body`, `title`, `meta`, `img`, `table`, `th`, `tr`, `td` , `ul`, `ol`, `li`, `h1`, `h2`, `p`, `div`, `span`, `hr`, `br` ou `Text`, l'arbre est invalide.
- `Html` doit contenir exactement un `Head`, **puis** un `Body`.
- `Head` ne doit contenir qu'un unique `Title` et uniquement ce `Title`.
- `Body` et `Div` ne doivent contenir que des éléments des types suivant : `H1`, `H2`, `Div`, `Table`, `Ul`, `Ol`, `Span`, ou `Text`.
- `Title`, `H1`, `H2`, `Li`, `Th`, `Td` ne doivent contenir qu'un unique `Text` et uniquement ce `Text`.
- `P` ne doit contenir que des `Text`.
- `Span` ne doit contenir que des `Text` ou des `P`.
- `Ul` et `Ol`] doivent contenir au moins un `Li` et uniquement des `Li`.

- `Tr` doit contenir au moins un `Th` ou `Td` et uniquement des `Th` ou des `Td`. Les `Th` et les `Td` doivent être mutuellement exclusifs.
- `Table` : ne doit contenir que des `Tr` et uniquement des `Tr`.

Votre classe `Page` doit également être capable de :

- Afficher son code `HTML` lorsqu'on en `print` une instance. Attention : le code `HTML` affiché doit être précédé d'un doctype si et seulement si le type de l'élément racine est `Html`.
- Ecrire son code `HTML` dans un fichier à l'aide d'une méthode `write_to_file` qui prend en paramètre le nom du fichier. Attention : le code `HTML` écrit dans le fichier doit être précédé d'un doctype si et seulement si le type de l'élément racine est `Html`.

Démontrez le fonctionnement de votre classe `Page` par des tests de votre choix en nombre suffisant pour couvrir toutes les fonctionnalités.