

# D02 - Formation Python-Django Bases Python 2

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

I	Préambule	2
II	Consignes	3
III	Règles spécifiques de la journée	4
IV	Exercice 00	5
V	Exercice 01	7
VI	Exercice 02	8
VII	Exercice 03	10
VIII	Exercice 04	12
IX	Exercice 05	14
$\mathbf{X}$	Exercice 06	16

## 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

Sauf contradiction explicite, les consignes suivantes seront valables pour tous les jours de cette Piscine.

- Seul ce sujet sert de référence : ne vous fiez pas aux bruits de couloir.
- Le sujet peut changer jusqu'à une heure avant le rendu.
- 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 <u>la procédure de rendu</u> pour tous vos exercices. L'url de votre dépot GIT pour cette journée est disponible sur votre intranet.
- Vos exercices seront évalués par vos camarades de Piscine.
- En plus de vos camarades, vous pouvez être évalués par un programme appelé la Moulinette. La Moulinette est très stricte dans sa notation car elle est totalement automatisée. Il est donc impossible de discuter de sa note avec elle. Soyez d'une rigueur irréprochable pour éviter les mauvaises surprises.
- Les exercices shell doivent s'éxcuter avec /bin/sh.
- Vous <u>ne devez</u> laisser <u>aucun</u> autre fichier que ceux explicitement specifiés par les énoncés des exercices dans votre dépot de rendu.
- Vous avez une question? Demandez à votre voisin de droite. Sinon, essayez avec votre voisin de gauche.
- Toutes les réponses à vos questions techniques se trouvent dans les man ou sur Internet.
- Pensez à discuter sur le forum Piscine de votre Intra et sur Slack!
- Lisez attentivement les exemples car ils peuvent vous permettre d'identifier un travail à réaliser qui n'est pas précisé dans le sujet à première vue.
- Réfléchissez. Par pitié, par Thor, par Odin!

## Chapitre III

## Règles spécifiques de la journée

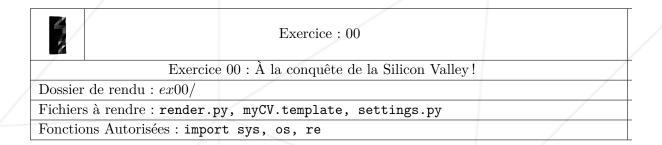
- Aucun code dans le scope global. Faites des fonctions!
- Sauf indication contraire, tous vos fichiers écrits en Python devront se terminer par un bloc

```
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



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 lire 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
"-Who are you?
-A {name}!"
$> python3 render.py file.template
$> cat file.html
"-Who are you?
-A duoquadragintian!"
```

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

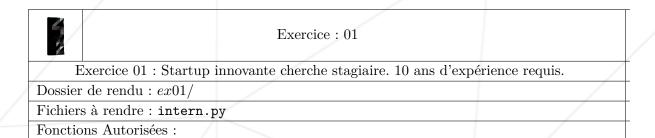
Vous devez rendre un fichier myCV.template qui, une fois converti en fichier HTML, devra au moins contenir la structure complète d'une page (doctype, head et body), le titre de la page, le nom et le prénom du propriétaire du CV, son âge, et sa profession. Bien entendu, ces informations ne devront pas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



help(globals), keyword expansion...

## Chapitre V

#### Exercice 01



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 à 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...".

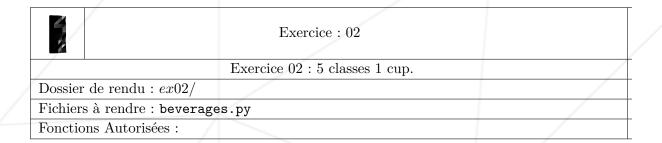
Une méthode make\_coffee() qui retournera une instance de la classe Coffee que vous aurez implémentée dans la classe Intern.

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



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!"

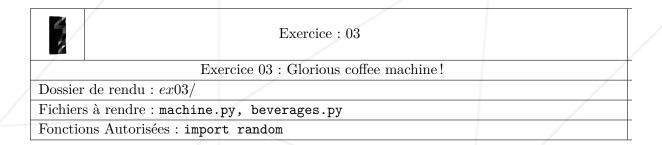


Vous ne devez redéfinir QUE ce qui est nécessaire, ce que vous avez besoin de changer, de redéfinir... (cf. DRY).

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

## Chapitre VII

#### Exercice 03



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 (aléatoirement), 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.

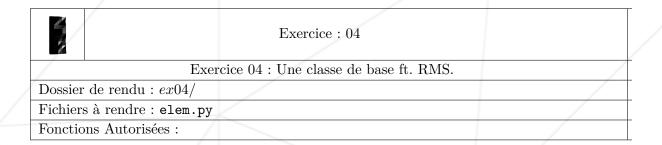
En cas de panne : l'appel de la méthode serve() doit provoquer la levée d'une exception de type CoffeeMachine.BrokenMachineException jusqu'à-ce que la méthode repair() soit appelée.

**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



Il est maintenant temps d'améliorer votre présence sur le WEB. Vous aimeriez bien utiliser vos nouvelles connaissances 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 apparaît 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, ainsi qu'une classe incomplète. Sans hésitation, vous complétez la classe Elem (les trous à remplir étant indiqués par des [...]) qui aura les caractéristiques suivantes :

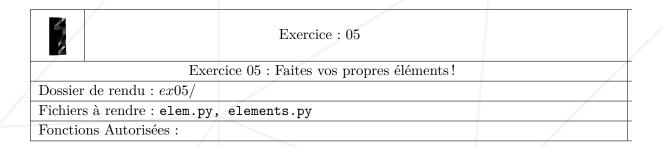
- Un constructeur pouvant prendre en paramètre le nom de l'élément, ses attributs HTML, son contenu et le type d'élément (balises simples ou doubles).
- Une méthode \_\_str\_\_() retournant le code HTML de l'élément.
- Une méthode add\_content() permettant d'ajouter des éléments à la fin du contenu.
- Une sous-classe d'*Exception* en son sein.

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 :

## Chapitre IX

#### Exercice 05



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.

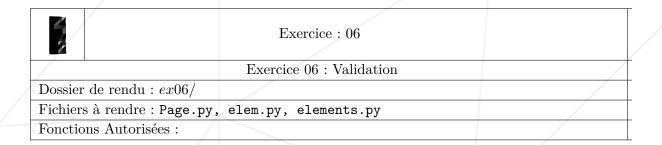
Démontrez le fonctionnement de ces classes par des tests de votre choix en nombre suffisant pour couvrir toutes les fonctionnalités. Après avoir codé ces classes, vous n'aurez plus besoin de préciser le nom ou le type d'un tag, ce qui est très pratique. Vous ne devez donc plus jamais instancier directement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

C'est beaucoup plus simple, non?:)

### Chapitre X

#### Exercice 06



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 is\_valid() 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, U1, O1, 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.