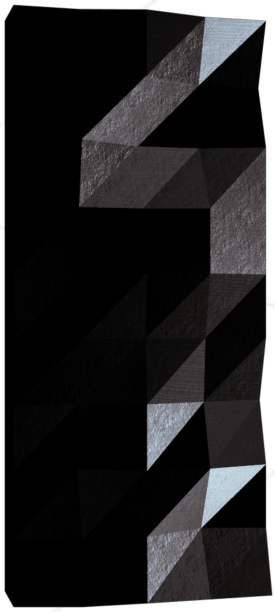

Formation Python-Django - 0

Starting

Résumé: Aujourd'hui, nous allons découvrir ensemble une première partie des bases syntaxiques et sémantiques de **`Python`**.

Version: 1

Table des matières

Chapitre I

Préambule

The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one– and preferably only one –obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea – let's do more of those!

```
import this
```

Chapitre II

Règles communes

- Votre projet doit être réalisé dans une machine virtuelle.
- Votre machine virtuelle doit avoir tout les logiciels necessaire pour réaliser votre projet. Ces logiciels doivent être configurés et installés.
- Vous êtes libre sur le choix du systmème d'exploitation à utiliser pour votre machine virtuelle.
- Vous devez pouvoir utiliser votre machine virtuelle depuis un ordinateur en cluster.
- Vous devez utiliser un dossier partagé entre votre machine virtuelle et votre machine hote.
- Lors de vos évaluations vous allez utiliser ce dossier partager avec votre dépot de rendu.
- Vos fonctions de doivent pas s'arrêter de manière inattendue (segmentation fault, bus error, double free, etc) mis à part dans le cas d'un comportement indéfini. Si cela arrive, votre projet sera considéré non fonctionnel et vous aurez 0 au projet.
- Nous vous recommandons de créer des programmes de test pour votre projet, bien que ce travail **ne sera pas rendu ni noté**. Cela vous donnera une chance de tester facilement votre travail ainsi que celui de vos pairs.
- Vous devez rendre votre travail sur le git qui vous est assigné. Seul le travail déposé sur git sera évalué. Si Deepthought doit corriger votre travail, cela sera fait à la fin des peer-evaluations. Si une erreur se produit pendant l'évaluation Deepthought, celle-ci s'arrête.

Chapitre III

Règles spécifiques de la journée

- Aucun code dans le scope global. Faites des fonctions !
- Chaque fichier rendu doit être terminé par un appel de fonction dans une condition identique à :

```
if __name__ == '__main__':
   your_function( whatever, parameter, is, required )
```

- Il est toléré de placer une gestion d'erreur dans cette même condition.
- Aucun import autorisé, à l'exception de ceux explicitement mentionnés dans la section 'Fonctions Autorisées' du cartouche de chaque exercice..
- Les exceptions soulevées par la fonction open ne sont pas à gérer.
- L'interprète à utiliser est `python3`.

Chapitre IV

Exercice 00 : mes premières variables

	Exercice : 00
Exercice 00 : mes premières variables	
Dossier de rendu : $ex00/$	
Fichiers à rendre : `var.py`	
Fonctions Autorisées : `n/a`	

Créez un script nommé `var.py` dans lequel vous devez définir une fonction `my_var`. Dans celle-ci, déclarez neuf variables de types differents et imprimez les sur la sortie standard. Vous devez reproduire exactement la sortie suivante :

```
$> python3 var.py
42 est de type <class 'int'>
42 est de type <class 'str'>
quarante-deux est de type <class 'str'>
42.0 est de type <class 'float'>
True est de type <class 'bool'>
[42] est de type <class 'list'>
{42: 42} est de type <class 'dict'>
(42,) est de type <class 'tuple'>
set() est de type <class 'set'>
$>
```

Il est bien entendu **interdit** d'écrire explicitement les types de vos variables dans les prints votre code. N'oubliez pas d'appeler votre fonction à la fin de votre script comme expliqué dans les consignes :

```
if __name__ == '__main__':
    my_var()
```

Chapitre V

Exercice 01 : Nombres

	Exercice : 01
Exercice 01 : Nombres	
Dossier de rendu : $ex01/$	
Fichiers à rendre : `numbers.py`	
Fonctions Autorisées : `n/a`	

Pour cet exercice, vous êtes libres de définir autant de fonctions que vous le souhaitez et de les nommer comme bon vous semble.

La tarball `d01.tar.gz` en annexe de ce sujet contient un sous-dossier `ex01/` dans lequel se trouve un fichier `numbers.txt` contenant les nombres de 1 a 100 séparés par une virgule.

Concevez un script `Python` nommé `numbers.py` dont le rôle est d'ouvrir le fichier `numbers.txt`, de lire les nombres qu'il contient, et de les afficher sur la sortie standard, un par ligne et sans virgule.

Chapitre VI

Exercice 02 : Mon premier dictionnaire

	Exercice : 02
Exercice 02 : Mon premier dictionnaire	
Dossier de rendu : $ex02/$	
Fichiers à rendre : `var_to_dict.py`	
Fonctions Autorisées : `N.A.`	

Vous êtes à nouveau libres de définir autant de fonctions que vous le souhaitez et de les nommer comme bon vous semble. Cette consigne ne sera plus mentionnée, sauf en cas de contradiction explicite.

Créez un script nommé `var_to_dict.py` dans lequel vous devez copier la liste de couples `d` suivante telle quelle dans l'une ou l'autre de vos fonctions :

```
d = [
  ('Hendrix'  , '1942'),
  ('Allman'   , '1946'),
  ('King'     , '1925'),
  ('Clapton'  , '1945'),
  ('Johnson'  , '1911'),
  ('Berry'    , '1926'),
  ('Vaughan'  , '1954'),
  ('Cooder'   , '1947'),
  ('Page'     , '1944'),
  ('Richards' , '1943'),
  ('Hammett'  , '1962'),
  ('Cobain'   , '1967'),
  ('Garcia'   , '1942'),
  ('Beck'     , '1944'),
  ('Santana'  , '1947'),
  ('Ramone'   , '1948'),
  ('White'    , '1975'),
  ('Frusciante', '1970'),
  ('Thompson' , '1949'),
  ('Burton'   , '1939')
]
```

Votre script doit transformer cette variable en un dictionnaire avec la date pour clé, et le nom du musicien pour valeur. Il doit ensuite afficher ce dictionnaire sur la sortie standard selon ce formatage precis :

```
1970 : Frusciante
1954 : Vaughan
1948 : Ramone
1944 : Page Beck
1911 : Johnson
...
```

l'ordre final ne sera pas necessairement identique a l'exemple, et c'est un comportement normal. Savez-vous pourquoi ?

Chapitre VII

Exercice 03 : Recherche par clé

	Exercice : 03
Exercice 03 : Recherche par clé	
Dossier de rendu : $ex03/$	
Fichiers à rendre : `capital_city.py`	
Fonctions Autorisées : `import sys`	

Vous disposez des dictionnaires suivants à recopier tels quels dans l'une ou l'autre des fonctions de votre script :

```
states = {
  "Oregon"   : "OR",
  "Alabama"  : "AL",
  "New Jersey": "NJ",
  "Colorado" : "CO"
}

capital_cities = {
  "OR": "Salem",
  "AL": "Montgomery",
  "NJ": "Trenton",
  "CO": "Denver"
}
```

Ecrivez un programme qui prend en argument un état (ex : Oregon) et qui affiche sur la sortie standand sa capitale (ex : Salem). Si l'argument ne donne aucun résultat, votre script doit afficher : `Unknown state`. S'il n'y a pas ou bien s'il y a trop d'arguments, votre script ne doit rien faire et quitter.

```
$> python3 capital_city.py Oregon
Salem
$> python3 capital_city.py Ile-De-France
Unknown state
$> python3 capital_city.py
$> python3 capital_city.py Oregon Alabama
$> python3 capital_city.py Oregon Alabama Ile-De-France
$>
```

Chapitre VIII

Exercice 04 : Recherche par valeur

	Exercice : 04
Exercice 04 : Recherche par valeur	
Dossier de rendu : $ex04/$	
Fichiers à rendre : `state.py`	
Fonctions Autorisées : `import sys`	

Vous disposez à nouveau des deux mêmes dictionnaires qu'à l'exercice précédent. Vous devez à nouveau les recopier tels quels dans l'une ou l'autre des fonctions de votre script.

Créez un programme qui prend une capitale en argument et affiche cette fois çi l'état correspondant. Le reste des comportements de votre programme doivent être identiques à ceux de l'exercice précédent.

```
$> python3 state.py Salem
Oregon
$> python3 state.py Paris
Unknown capital city
$> python3 state.py
$>
```

Chapitre IX

Exercice 05 : Recherche par clé ou par valeur

	Exercice : 05
Exercice 05 : Recherche par clé ou par valeur	
Dossier de rendu : $ex05/$	
Fichiers à rendre : `all_in.py`	
Fonctions Autorisées : `import sys`	

En partant toujours des mêmes deux dictionnaires, que vous devez toujours recopier tels quels dans l'une ou l'autre des fonctions de votre script, écrivez un programme ayant les comportements suivants :

- Le programme doit prendre en argument une string contenant autant d'expressions à chercher que l'on veut, séparées par une virgule.
- Pour chaque expression de cette string, le programme doit détecter si c'est une capitale, un état, ou aucun des deux.
- Le programme ne doit pas être sensible à la casse, ni aux espaces blancs en trop.
- S'il n'y a pas de paramètre ou bien trop de paramètres, le programme n'affiche rien.
- Lorsqu'il y a deux virgules d'affilées dans la string, le programme n'affiche rien.
- Le programme doit afficher les résultats séparés par un retour à la ligne et utiliser le formatage précis suivant :

```
$> python3 all_in.py "New jersey, Tren ton, NewJersey, Trenton, toto,   ,     sAlem"
Trenton is the capital of New Jersey
Tren ton is neither a capital city nor a state
NewJersey is neither a capital city nor a state
Trenton is the capital of New Jersey
toto is neither a capital city nor a state
Salem is the capital of Oregon
$>
```

Chapitre X

Exercice 06 : Tri d'un dictionnaire

	Exercice : 06
Exercice 06 : Tri d'un dictionnaire	
Dossier de rendu : $ex06/$	
Fichiers à rendre : `my_sort.py`	
Fonctions Autorisées : `N.A.`	

Intégrez ce dictionnaire à l'une ou l'autre de vos fonctions tel quel :

```
d = {
    'Hendrix'  :  '1942',
    'Allman'   :  '1946',
    'King'     :  '1925',
    'Clapton'  :  '1945',
    'Johnson'  :  '1911',
    'Berry'    :  '1926',
    'Vaughan'  :  '1954',
    'Cooder'   :  '1947',
    'Page'     :  '1944',
    'Richards' :  '1943',
    'Hammett'  :  '1962',
    'Cobain'   :  '1967',
    'Garcia'   :  '1942',
    'Beck'     :  '1944',
    'Santana'  :  '1947',
    'Ramone'   :  '1948',
    'White'    :  '1975',
    'Frusciante': '1970',
    'Thompson' :  '1949',
    'Burton'   :  '1939',
}
```

Ecrivez un programme qui affiche les noms des musiciens triés par ordre d'année croissant puis par ordre alphabétique de noms lorsque les dates sont identiques, un par ligne, sans afficher les dates.

Chapitre XI

Exercice 07 : Tableau périodique des éléments

	Exercice : 07
Exercice 07 : Tableau périodique des éléments	
Dossier de rendu : $ex07/$	
Fichiers à rendre : `periodic_table.py`	
Fonctions Autorisées : `import sys`	

La tarball `d01.tar.gz` en annexe de ce sujet contient un sous-dossier `ex07/` dans lequel se trouve un fichier `periodic_table.txt` qui décrit le tableau périodique des éléments dans un format pratique pour les informaticiens.

Créez un programme qui utilise ce fichier pour écrire une page `HTML` représentant le tableau périodique des éléments correctement formaté.

- Chaque élement doit être dans une 'case' d'un tableau `HTML`.
- Le nom d'un élément doit être dans une balise titre de niveau 4.
- Les attributs d'un élément doivent être sous forme de liste. Doivent être listés au minimum le numéro atomique, le symbole, et la masse atomique.
- Vous devez respecter à minima le layout d'une table de Mendeleiev telle qu'on peut la trouver sur google, à savoir qu'il doit y avoir des cases vides là où il doit y en avoir, ainsi que des retours a la ligne là où il en faut.

Votre programme doit créer le fichier résultat `periodic_table.html`. Ce fichier `HTML` doit bien entendu être immédiatement lisible par n'importe quel navigateur et doit être valide `W3C`.

Vous êtes libre de concevoir votre programme comme vous l'entendez. N'hésitez pas à fragmenter votre code en fonctionnalités distinctes et eventuellement réutilisables. Vous pouvez customiser les balises avec un style CSS "inline" pour rendre votre rendu plus

joli (ne serait-ce que le contour des cases du tableau), voire même générer un fichier `periodic_table.css` si vous préférez.

Voici un extrait d'un exemple de sortie pour vous donner une idée :

```
[...]
  <table>
    <tr>
      <td style="border: 1px solid black; padding:10px">
        <h4>Hydrogen</h4>
            <ul>
                <li>No 42</li>
                <li>H</li>
                <li>1.00794</li>
                <li>1 electron</li>
            <ul>
      </td>
      [...]
```

Chapitre XII

Rendu et peer-évaluation

Rendez votre travail dans votre dépôt `Git` comme d'habitude. Seul le travail présent dans votre dépôt sera évalué en soutenance. Vérifiez bien les noms de vos dossiers et de vos fichiers afin que ces derniers soient conformes aux demandes du sujet.

`L'évaluation se déroulera sur l'ordinateur du groupe évalué.`