TP d'informatique – Python et bases de données

Jouons un peu avec Musicbrainz

CPP 2A
Septembre – décembre 2015

Rendu du TP Pour ce TP, vous devrez rendre, sous forme électronique uniquement :

1. les fichiers Python (`.py`) correspondant au code que vous avez écrit (et complété) ainsi que ceux nécessaire à leur bonne éxécution (Veillez à bien respecter les noms de fichier et de fonction évoqués dans l'énoncé) ;
2. un fichier texte (`.txt`) ou PDF (`.pdf`) [1] contenant les réponses aux questions qui exigent une explication, un paragraphe concernant les difficultés qui ont nécessité une proportion non négligeable du temps passé globalement sur le projet et tout autre élément que vous jugez pertinent.

Le TP est à faire en équipe de 2 étudiants (binômes). Le rendu du TP se fera en utilisant l'application Teide (`https://intranet.ensimag.fr/teide/`). La date limite de rendu est fixée au :

Lundi 19 octobre 2015, 20h

1 Objectifs du TP, calendrier, conseils

Lors des Travaux Pratiques encadrés, vous avez pu travailler sur la base de données musicale libre *Musicbrainz*, dont un extrait très simplifié vous a été distribué sous forme de fichier SQLite. Dans ce TP, vous allez travailler sur la même base de données, mais cette fois-ci en l'interrogeant non plus directement comme vous avez pu le faire en séance (à l'aide du client sqlite3 ou de la petite interface graphique fournie), mais en passant par un programme Python (langage que vous connaissez bien maintenant...). Ce TP vous permettra d'une part d'appliquer vos connaissances en bases de données et Python, mais aussi d'illustrer le principe des architectures informatiques deux-tiers et trois-tiers. [2]

Certaines questions sont marquées comme facultatives. Elles ne sont pas comptées dans le barème. Vous pouvez les traiter pour satisfaire votre curiosité et/ou pour avoir un retour de votre correcteur sur vos réponses, mais ceux qui traitent ces questions n'auront pas plus de points que les autres.

Les questions 1 à 4 n'utilisent que des notions de Python que vous connaissez depuis la 1A, vous pouvez donc les traiter sans attendre les cours de bases de données. Traitez au moins la question 1 avant la première séance machine (TD numéro 2), et impérativement toutes les questions 1 à 4 avant le cours numéro 4 en guise de révision sur la programmation Python.

Les questions 5 et suivantes utilisent le langage SQL, vous ne pourrez pas les traiter avant le cours numéro 3.

L'énoncé de TP est long car il contient beaucoup d'indications pour vous aider. Si vous bloquez sur une question, vérifiez qu'il n'y a pas une indication qui puisse vous aider un peu plus loin dans l'énoncé.

2 Révisions et compléments de Python

2.1 Construction de chaînes à partir de listes

Le fichier `exercices_listes.py` contient une définition de variable `couleurs` qui est une liste de chaînes de caractères. Une première ébauche de fonction `affiche_couleurs` est proposée, mais pour l'instant elle affiche ceci :

```
>>> affiche_couleurs(couleurs)
Les couleurs sont : bleujaunevert.
```

Ouvrez le fichier `exercices_listes.py` dans Spyder (faites attention à utiliser « Spyder3 » dans le menu quand vous travaillez à l'Ensimag, comme en première année). Exécutez-le dans l'interprète Python, et vérifiez que vous obtenez le résultat ci-dessus.

1. Pas de `.doc`, `.docx` ou autre format exotique !
2. C'est au programme des CPGE, vous ne perdez donc pas votre temps... Cf. la section 10.5 page 271 du livre « Informatique pour tous en classes préparatoires aux grandes écoles » pour plus de détails.

☞ *Par défaut, la fonction* `print` *affiche son paramètre puis un retour à la ligne. Pour éviter d'afficher ce retour à la ligne, on peut utiliser* `print('chaine', end='')` *comme c'est fait dans le squelette.*

Question 1 Modifiez la fonction `affiche_couleurs` pour que l'affichage se fasse correctement, en séparant les couleurs par des virgules comme ceci (attention, il faut $n-1$ virgules s'il y a n mots) :

```
>>> affiche_couleurs(couleurs)
Les couleurs sont : bleu, jaune, vert.
```

☞ *Cette question est vraiment un exercice de révisions, vous avez déjà fait quelque chose qui y ressemblait* beaucoup *en 1A...*

La fonction `affiche_couleurs` utilise directement la fonction `print` de Python pour afficher le résultat. Un problème avec cette approche est que l'affichage est envoyé directement à l'utilisateur, on ne peut pas le récupérer dans le programme Python (souvenez-vous de la 1A, la différence entre `print` et `return` ...). On peut s'en rendre compte avec ces lignes dans du fichier `exercices_listes.py` :

```
1 print("Tentative d'affectation :")
2 chaine = affiche_couleurs(couleurs)
3 print("L'affectation est faite, voici le résultat :")
4 print(chaine)
```

qui affichent :

```
Tentative d'affectation :
Les couleurs sont : bleu, jaune, vert.
L'affectation est faite, voici le résultat :
None
```

La fonction n'utilise pas `return` donc elle ne renvoie rien (`None` en Python).

Question 2 En vous inspirant de la fonction `affiche_couleurs`, écrire une fonction `construit_chaine_couleurs` qui prend le même paramètre que `affiche_couleurs`, mais *renvoie* la chaîne de caractère `'Les couleurs sont : bleu, jaune, vert.'` sans l'afficher.

☞ *La fonction peut construire la chaîne petit à petit dans la variable « `resultat` » en utilisant « `resultat = resultat + ...` » autant de fois que nécessaire, et renvoyer le résultat avec « `return resultat` ». On de doit pas utiliser* `print`.

☞ *Un raccourci pratique : au lieu d'écrire « `resultat = resultat + x` », on peut écrire « `resultat += x` » qui fait exactement la même chose.*

Vérifiez que le morceau de code du squelette

```
1 print("Une affectation qui devrait marcher :")
2 chaine = construit_chaine_couleurs(couleurs)
3 print("La chaine est construite, je vais l'afficher")
4 print(chaine)
```

affiche bien ceci :

```
Une affectation qui devrait marcher :
La chaine est construite, je vais l'afficher
Les couleurs sont : bleu, jaune, vert.
```

2.2 Compléments sur Python : les tuples

Vous avez vu en première année la notion de liste en Python. Par exemple, `[1, 3, 42]` est une liste Python à 3 éléments, qui sont les nombres 1, 3 et 42. Python a un deuxième concept très similaire : les tuples (ou N-uplets en bon français). Un tuple s'écrit comme une liste, mais en remplaçant les crochets (`[]`) par des parenthèses. La principale différence avec les listes est qu'un tuple n'est pas modifiable (on ne peut pas changer le nombre d'éléments, ni les éléments eux-mêmes). Par exemple :

```
>>> x = [1, 2]
>>> x[0] = 42
>>> x
[42, 2]
```

```
>>> y = (1, 2)
>>> y[0]
1
>>> y[0] = 42
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

On peut bien sûr imbriquer les tuples et les listes : `[(1, 2), (3, 4)]` est une liste qui contient deux doublets (tuples à 2 éléments).

Dans le cas particulier des tuples à 1 seul élément, on utilise la syntaxe `(element,)` pour éviter l'ambiguïté avec les parenthèses de groupement. Par exemple, `(42)` est le nombre 42, alors que `(42,)` est un tuple contenant le nombre 42.

Dans `exercices_listes.py`, vérifiez que vos fonctions `affiche_couleurs` et `construit_chaine_couleurs` fonctionnent de la même manière avec un tuple et avec un tableau. Un test vous est fourni en fin de fichier.

2.3 Listes de tuples

Nous avons vu qu'une liste Python peut contenir des données de n'importe quel type, et il est donc possible d'avoir des listes de listes, des listes de tuples, ...

Dans le fichier `trousse.py`, vous trouverez une déclaration de variable `contenu` qui est une liste de tuples :

```
contenu = [
    ('stylo', 'bleu', False),
    ('gomme', 'rouge', True),
    ('feutre', 'vert', False)
]
```

Question 3 Que valent les expressions `contenu[0]` ? `contenu[1][2]` ? `contenu[2][1][0]` ?

☞ *Vous pouvez charger ce fichier dans votre interpréteur Python et évaluer les expressions dans l'interpréteur interactif, mais il faut aussi comprendre pourquoi ces expressions ont ces valeurs.*

La liste `contenu` représente une trousse dans laquelle chaque objet est représenté par un tuple de forme `(`*`nom`*`, `*`couleur`*`, `*`féminin`*`)`. Ici, *nom* et *couleur* sont des chaînes de caractères, et *féminin* est un booléen qui est vrai si l'objet est de genre féminin (auquel cas il faudra utiliser `'une'` et non `'un'` comme article). Le but de cette partie est d'écrire une fonction qui construise à partir de cette liste une chaîne de la forme :

```
Dans ma trousse, il y a :
- un stylo bleu
- une gomme rouge
- un feutre vert
```

Question 4 Écrire et tester la fonction `description_trousse` qui renvoie (sans l'afficher) la chaîne spécifiée ci-dessus.

☞ *Un retour chariot dans une chaîne s'écrit `'\n'` (« n » comme « newline » en anglais).*

☞ *Si écrire la fonction complète vous paraît trop compliqué, commencez par parcourir la liste et concaténer tous les noms d'objets, avec une boucle comme :*

```
for objet in contenu:
    resultat += objet[0]
```

☞ *Pour tester si une variable `v` vaut `True`, on peut écrire simplement « `if v:` » plutôt que « `if v == True:` ».*

2.4 Rappels sur les programmes multi-fichiers en Python

On peut utiliser des fonctions définies dans un fichier `toto.py` depuis un autre fichier en utilisant `import toto` en début de fichier, puis en écrivant `toto.fonction()` pour appeler la fonction `fonction` du module `toto` (c-à-d du fichier `toto.py`).

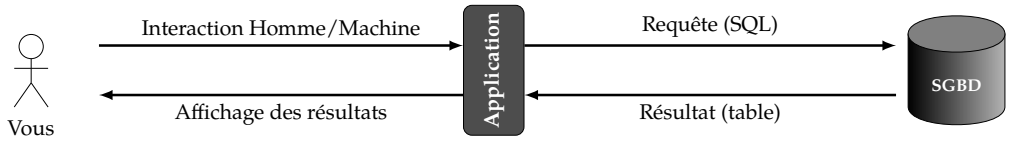
Attention, `import toto` a pour conséquence d'exécuter tout le contenu du fichier `toto.py`. Si le fichier contient des définitions de fonctions (`def`), ces fonctions sont rendues disponibles mais ne sont pas appelées au

moment de l'import. Par contre, si `toto.py` contient du code en dehors des définitions de fonctions, alors tout ce code est exécuté au moment de l'import (en général, ce n'est pas souhaitable).

3 Préliminaires techniques pour les bases de données

3.1 De la dialectique des bases de données

Jusqu'ici donc, vous avez interrogé une base de données en dialoguant avec le Système de Gestion de Bases de Données à l'aide de requêtes SQL. Dans la vraie vie, les choses se passent rarement comme cela. [3] En pratique, l'utilisateur final se trouve devant un programme, qui peut être par exemple une application standard ou une page web, qui sert d'intermédiaire entre l'utilisateur et le système de gestion de bases de données :



L'utilisateur n'interagit donc pas directement avec le SGBD, mais avec l'application intermédiaire. Selon les interactions avec l'utilisateur, l'application se charge de construire des requêtes au SGBD et de renvoyer le résultat à l'utilisateur de manière conviviale.

À titre d'exemple, considérons une application fictive de recherche d'horaires de train. Voici un scénario possible de fonctionnement de cette application :

1. L'application demande à l'utilisateur : « `Quelle est la ville de départ ?` ».
2. L'utilisateur répond « `Caussade` ».
3. L'application demande à l'utilisateur : « `Quelle est la ville d'arrivée ?` ».
4. L'utilisateur répond « `Brive-la-Gaillarde` ».
5. L'application envoie au SGBD la requête SQL :
```
SELECT heureDep, heureArr FROM trains
WHERE villeDep = 'Caussade' AND villeArr = 'Brive-la-Gaillarde';
```
6. Le SGBD répond au programme en envoyant l'ensemble de tuples $\{(15h47, 17h22); (18h21, 19h58)\}$
7. Le programme affiche à l'utilisateur :
```
Résultats pour le trajet Caussade -> Brive-la-Gaillarde
Premier train : 15h47  ->  17h22
Second train :  18h21  ->  19h58
```

3.2 En pratique, avec Python et SQLite3...

Dans notre cas, c'est un programme Python qui jouera le rôle d'application intermédiaire. Le dialogue se fera avec le SGBD SQLite3, et plus particulièrement la base Musicbrainz dont vous avez l'extrait sous forme de fichier `.db`. La plupart des langages de programmation ont des bibliothèques permettant à un programme de dialoguer avec la plupart des SGBD existants. Le couple Python-SQLite3 ne déroge pas à la règle, et c'est le module `sqlite3` que nous utiliserons.

Code fourni Tout le code permettant l'accès à la base de données vous est fourni dans le fichier `database.py`. *Vous n'avez pas à le modifier*, mais vous pouvez le parcourir si vous êtes de nature curieuse. Ce module contient notamment une fonction `execute_query(dbfile, query)`. Cette fonction :

- prend en paramètre un nom de fichier SQLite3 sous forme de chaîne de caractères, et une requête SQL sous forme de chaîne de caractères ;
- renvoie, sous forme de liste, le résultat de la requête (une table). Chaque élément de la liste est un tuple (au sens Python), correspondant à un tuple (au sens SQL) de la table.

C'est la seule fonction du module que vous aurez à appeler explicitement.

Vous avez également dans l'archive :

- le fichier SQLite3 `musicbrainz-FR.db` contenant l'extrait de la base sur lequel vous avez déjà travaillé, et dont vous avez le descriptif dans le sujet de TP SQL effectué en séance ;

3. Que celui qui a déjà recherché l'horaire du train Belfort – Culmont-Chalindrey en tapant une requête SQL me jette la première pierre...

- deux fichiers `HTTPServerMusique.py` et `index.html` qui ne vous serviront que dans les questions de la section 4.4. Gardez-les sous le coude pour la dernière partie.

4 Bases de données : travail demandé

4.1 Premier contact

Ouvrez le fichier `database.py` dans Spyder. Dans l'interpréteur de commandes Python, importez les fonctions du module `database` :

```
>>> import database
```

Puis exécutez, à l'aide de la fonction `execute_query`, la première requête du TP Musicbrainz que vous avez ou que vous allez faire en séance :

```
>>> database.execute_query("musicbrainz-FR.db",
... "SELECT DISTINCT nomDisque FROM Disques WHERE idArtiste = 22164;")
```

4.2 Des artistes, des albums

Créez un nouveau fichier `musique.py` dans Spyder, et importez le module `database` comme toute première instruction (`import database` au tout début du fichier).

Question 5 Écrivez une fonction `get_matching_artists(pattern)` prenant en paramètre une chaîne de caractères et renvoyant la liste des identifiants d'artistes et noms d'artistes dont le nom *contient* la chaîne de caractères passée en paramètre (pensez à l'opérateur `LIKE` de SQL). Le tout doit être classé par ordre alphabétique de noms d'artistes.

☞ *Pendant la mise au point de votre fonction, vous pouvez afficher la requête avant de l'envoyer à SQLite, par exemple :*

```
def get_matching_artists(pattern):
    query = "..."
    print(query) # Seulement pour debugger, à supprimer après.
    return database.execute_query("musicbrainz-FR.db", query)
```

Écrire une fonction qui teste `get_matching_artists` en demandant à l'utilisateur une chaîne de caractères (grâce à la fonction `input`) et en affichant la liste des noms d'artistes concordants. Par exemple, sur la chaîne de caractères `"Python"`, elle doit afficher :

```
[(619093, 'Lhacene 357 python'), (58603, 'Python Lee Jackson'), (331160, 'Spittin Python')]
```

Attention, pour le bon fonctionnement de la suite, la fonction `get_matching_artists(pattern)` ne doit pas interagir directement avec l'utilisateur (elle n'utilise ni `print` ni `input`), c'est le programme principal qui s'en occupe.

Question 6 Écrivez maintenant une fonction `get_releases_by_artist(id_artist)` prenant en paramètre un identifiant d'artiste et renvoyant la liste de tous ses disques (albums et autres).

☞ *Si `x` est un entier et `s` une chaîne, on peut écrire `s + str(x)` pour obtenir la concaténation de `s` et de la représentation textuelle de `x`.*

Question 7 Écrire une nouvelle fonction qui teste `get_releases_by_artist` en effectuant les tâches suivantes.

1. Demander à l'utilisateur une chaîne de caractères.
2. Retourner la liste des artistes concordants, numérotée de 1 à n (ou redemander un autre nom tant qu'il n'y a pas d'artiste concordant).
3. Demander à l'utilisateur de choisir un artiste en tapant un numéro de 1 à n.
4. Afficher la liste des noms d'albums de cet artiste.

☞ *Attention, l'affichage numérote à partir de 1, mais les listes Python sont numérotées à partir de 0.*

Voici un exemple d'interaction :

```
Quel artiste vous intéresse ?
> Noir Désir
7 artiste(s) trouvé(s)...
=========================================================
1    Alain Bashung & Noir Désir
2    Jeanne Balibar, France Cartigny, Femmouzes T., Dadou et Diésel de KDD
     & Noir Désir, Akosh S. Unit, Rodolphe Burger, Theo Hakola, Blankass & Grégoire Simon
3    Noir Désir
4    Noir Désir & Akosh S. Unit
5    Noir Désir & Alain Bashung
6    Noir Désir & Brigitte Fontaine
7    Noir Désir & Têtes Raides
Parmi cette liste d'artistes, lequel vous convient (donnez le numéro entre 1 et 7) ?
> 3
32 disque(s) trouvé(s)...
=============================================================================================================
1       666.667 Club
2       Aux sombres héros de l'amer
3       Ces gens-là
4       Charlie
5       Comme elle vient
(...)
```

Question 8 (facultative, non comptée dans le barème) Que se passe-t-il si vous essayez de lister les albums de l'artiste `Pep's` ? Vous aurez probablement un problème, qui serait inacceptable dans un vrai logiciel [4] mais que nous considérerons comme acceptable dans le cadre de ce TP. Pouvez-vous l'expliquer ? Mieux, le corriger ?

4.3 The Ultimate Musicbrainz Fight...

Nous allons maintenant nous intéresser à la comparaison des artistes selon leur prolificité. Plus précisément, nous aller chercher à classer les artistes selon leur nombre d'enregistrements (« recording » dans la terminologie Musicbrainz). Créez un nouveau fichier `dbfight.py` dans Spyder ou IDLE, et comme avant importez le module `database` en toute première instruction du programme.

Question 9 Écrivez une fonction `rank_artists(id_artist_list)` prenant en paramètre une liste d'identifiants d'artistes et retournant une liste constituée de couples (nom d'artiste, nombre d'enregistrements), contenant les données pour tous les artistes dont l'identifiant apparaît dans la liste passée en paramètre. La liste retournée par la fonction doit être triée par nombre d'enregistrements décroissant.

Pas clair ? Voici un exemple d'exécution de cette fonction pour trois artistes :

```
>>> dbfight.rank_artists([1128, 685411])
[('Jacques Brel', 2100), ('Stromae', 18)]
```

(Visiblement Jacques Brel a interprété environ 100 fois plus de titres que Stromae.)

La fonction ne doit effectuer qu'*une seule* requête SQL (moitié des points à la question sinon). Pensez à l'opérateur SQL `IN` (utilisable dans la clause `WHERE`) qui permet de tester l'appartenance d'une valeur d'attribut à une liste de valeurs possibles.

☞ *Construire le fragment de requête `'IN (..., ...)'` ne devrait pas vous poser de problème si vous avez déjà répondu aux questions de la partie 2.*

Question 10 (facultative, non comptée dans le barème) Testez votre fonction en écrivant un petit programme qui demande à l'utilisateur d'entrer successivement des noms d'artistes, et qui appelle la fonction `rank_artists` pour effectuer la comparaison. Pour ce faire, vous pourrez adapter le petit programme développé à la question 7.

4. C'est ce qu'on appelle une faille de type « SQL injection »

4.4 Le tiers manquant...

Préliminaires techniques : les applications Web pour les nuls Dans ce TP, nous avons interagi jusqu'ici avec la base de données par le biais d'un programme Python que nous avons lancé directement sur notre machine (application *standalone*). C'est le principe de l'architecture deux-tiers : un tiers [5] pour le SGBD, un tiers pour le programme Python. En réalité, ce n'est pas le cas d'utilisation le plus fréquent des SGBD. Une manière beaucoup plus courante d'interagir avec une base de données est d'utiliser une *application Web*, exécutée dans un navigateur : c'est ce qui se passe par exemple lorsque vous commandez un billet de train sur le site `http://www.voyages-sncf.com/` ou que vous effectuez une requête sur un moteur de recherche.

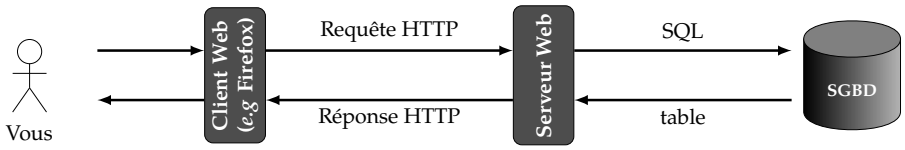
Le principe de fonctionnement des applications Web est très simple. Vous avez :

- d'un côté, un serveur, qui est un petit programme tournant sans discontinuer sur une machine située quelque part et accessible à l'aide d'une URL (`http://www.google.fr/` par exemple) ;
- d'un autre côté, un client (navigateur en général, Firefox, Chrome, Safari par exemple) qui est un petit programme tournant sur la machine personnelle d'un utilisateur.

Lorsque l'utilisateur tape une URL dans la barre d'adresse du navigateur, celui-ci contacte le serveur concerné en utilisant un protocole de communication dédié. [6] Si l'adresse est valide et la requête formulée assez poliment et de manière compréhensible, le serveur envoie une réponse, en général sous forme de texte formaté. C'est ce texte qui est affiché dans le navigateur.

Comme vous l'avez sans doute compris, le serveur est un programme chargé de répondre avec du texte (entre autres) aux requêtes des clients : à peu près comme ce que vous avez fait jusqu'à présent dans ce TP, sauf qu'au lieu d'interagir directement avec l'utilisateur (à l'aide d'affichages avec `print()`, et d'entrées clavier avec `input()`), on utilise un protocole de communication particulier. En particulier, si vous voulez interagir avec le SGBD côté serveur, cela s'effectue de la même manière que précédemment.

Dans ce nouveau contexte, notre architecture deux-tiers du début a donc maintenant un tiers supplémentaire :



Question 11 Ouvrez le fichier `HTTPserver_musique.py` dans votre éditeur, et lancez-le. En principe, le serveur se lance et affiche le message ci-dessous. [7]

```
Serveur démarré sur le port  4242.
```

Ouvrez un navigateur web (Firefox par exemple), et tapez l'URL `http://localhost:4242/` dans la barre d'adresse. [8] Que se passe-t-il ? Entrez un nom d'artiste dans la boîte de dialogue et validez. Que se passe-t-il ?

Nous allons maintenant modifier le code Python du serveur. Pour que nos modifications soient prises en compte, il faut relancer le serveur. Commencez par l'arrêter en plaçant le curseur dans l'interprète Python, et en faisant un Control-C. Le programme devrait vous répondre `^C reçu, je ferme le serveur. Merci.`

☞ *Il peut arriver que `HTTPServerMusic.py` renvoie une erreur venant de Python. Relancer Python voire Spyder devrait résoudre ce problème.*

Question 12 Complétez la fonction `format_matching_artists` dans le fichier `HTTPserver_musique.py`. Cette fonction doit renvoyer une chaîne de caractères représentant la liste des artistes renvoyée par la fonction `get_matching_artists`, formatée d'une manière lisible. À titre d'exemple, cette fonction appelée sur la chaîne de caractères `"brigitte"` doit renvoyer la chaîne :

```
"""
34 artiste(s) trouvé(s)...
================================================================================================================
```

5. Ici, « tiers » est à comprendre au sens d'« entité », pas au sens de la fraction.
6. Il s'agit du protocole HTTP – *HyperText Transfer Protocol*.
7. Si ce n'est pas le cas, vérifiez que votre script `musique.py` importé par `HTTPserver_musique.py` ne demande pas une entrée à l'utilisateur. S'il en demande une, il faut déplacer le code de `musique.py` qui ne fait pas partie d'une définition de fonction dans un fichier `test_musique.py` et commençant lui-même par "`import musique`" (voir section 2.4).
8. Le nom de machine « localhost » désigne la machine sur laquelle est lancée le navigateur. L'URL signifie que votre serveur est hébergé sur la même machine que vous (et écoute les connexions entrantes sur le port 4242).

```
1 Alla Francesca & Brigitte Lesne
2 Areski Belkacem & Brigitte Fontaine
3 Brigitte
4 Brigitte Bardot
5 Brigitte Bardot & Serge Gainsbourg
6 Brigitte Bop
7 Brigitte Engerer
8 Brigitte Engerer, Régis Pasquier
...
"""
```

Testez à nouveau en relançant `HTTPserver_musique.py` puis en tapant un nom d'artiste dans la boîte de dialogue de la page affichée dans le navigateur. Étonnant, non ?

☞ *Attention, la fonction doit bien* renvoyer *une chaîne, et non l'afficher.*

4.5 Fraude

Il est interdit d'utiliser le TP d'une autre équipe, ou de mettre à disposition son TP à une autre équipe. Vous êtes encouragés à aller chercher de l'information sur internet, mais pas autorisé à copier-coller du code que vous n'avez pas écrit vous même. La sanction en cas de fraude est la note 0/20 au TP. Pour plus de détails sur ce qui est autorisé ou non, voir la charte des TPs de l'Ensimag, que nous appliquerons aussi pour ce TP : https://intranet.ensimag.fr/teide/Charte_contre_la_fraude.php

En cas de doute, n'hésitez pas à demander conseil à vos enseignants.