Introduction à la programmation en Python

Projet : reconnaissance de la langue d'un texte

Département Réseaux et Télécommunications, IUT de Villetaneuse, Université Paris 13

Écrit par Luca Saiu — http://ageinghacker.net

Énoncé version 2, décembre 2018

1 Introduction

Ce projet consiste à *écrire un programme Python qui, étant donné un texte, détermine dans quelle langue il est écrit.*

Par exemple le programme pourra accepter en entrée le texte (tiré de *La Maison du chat-qui-pelote*, Balzac, 1830) :

> Le mal est fait, ma femme, dit Joseph Lebas, il faut chercher à donner de bons conseils à notre sœur. Puis, l'habile négociant analysa lourdement les ressources que les lois et les mœurs pouvaient offrir à Augustine pour sortir de cette crise ; il en numérota pour ainsi dire les considérations, les rangea par leur force dans des espèces de catégories, comme s'il se fût agi de marchandises de diverses qualités ; puis il les mit en balance, les pesa, et conclut en développant la nécessité où était sa belle-sœur de prendre un parti violent qui ne satisfit point l'amour qu'elle ressentait encore pour son mari. Aussi ce sentiment se réveilla-t-il dans toute sa force quand elle entendit Joseph Lebas parlant de voies judiciaires.

Étant donné ce texte le programme répondra `'français'`, ou `'french'`. Étant donné un autre texte écrit, par exemple, en chinois ou en allemand, le programme répondra avec le nom de la langue.

Étant donné un texte de la taille de cet exemple, la réponse de votre programme réalisé selon la technique présentée ici devrait être typiquement fiable. Votre programme peut répondre avec une langue incorrecte si le texte en entrée est de taille très faible.

1.1 À propos de ce projet

Ce projet est un exercice de compréhension de l'énoncé, particulièrement par rapport au modèle mathématique. Les mathématiques nécessaires sont *extrêmement simples*, limitées essentiellement au théorème de Pythagore. La formule dans §2.4.3 est le vrai noyau de la formalisation à comprendre, et elle est déjà fournie. La partie précédente sert à présenter cette seule formule importante d'une façon claire et intuitive.

Le code a écrire est de taille faible, basé sur les dictionnaires, les chaînes de caractères et les boucles `for`.

2 Théorie

Ici nous présentons l'idée à la base d'un système automatique pour la reconnaissance de la langue d'un texte.

2.1 Tables d'occurrence

Étant donnée une chaîne de caractères nous pouvons compter le nombre d'occurrences de chaque caractère, en obtenant une *table d'occurrence*.

Par exemple dans la chaîne `'Les ordinateurs sont intéressants'` on aura :

`' '`	3
`'L'`	1
`'a'`	2
`'d'`	1
`'e'`	3
`'i'`	2
`'n'`	4
`'o'`	2
`'r'`	3
`'s'`	6
`'t'`	4
`'u'`	1
`'é'`	1

On voit bien que le caractère `'L'` est présent une seule fois, et le caractère 'a' deux fois. Ici on a aussi considéré le caractère espace (`' '`), présent trois fois.

Tout caractère non présent dans notre texte `'Les ordinateurs sont intéressants'`, est (implicitement) associé à un nombre d'occurrences 0 ; par exemple la lettre `'z'` n'est pas présente dans le texte, qu'on peut donc raisonnablement affirmer contenir *zéro occurrences* du caractère `'z'`. De nombreux autres caractères sont également absents : les idéogrammes chinois, par exemple, ont tous un nombre d'occurrences 0 dans `'Les ordinateurs sont intéressants'`.

2.2 Tables de fréquence

En divisant chaque nombre d'occurrences dans une table d'occurrence par la taille de la chaîne (ici 33, car la longueur de `'Les ordinateurs sont intéressants'` est 33 caractères) on obtient une *table de fréquence* :

`' '`	$\frac{3}{33}$
`'L'`	$\frac{1}{33}$
`'a'`	$\frac{2}{33}$
`'d'`	$\frac{1}{33}$
`'e'`	$\frac{3}{33}$
`'i'`	$\frac{2}{33}$
`'n'`	$\frac{4}{33}$
`'o'`	$\frac{2}{33}$
`'r'`	$\frac{3}{33}$
`'s'`	$\frac{6}{33}$
`'t'`	$\frac{4}{33}$
`'u'`	$\frac{1}{33}$
`'é'`	$\frac{1}{33}$

Cette table de fréquence nous dit que, dans notre texte, exactement trois trente-troisièmes des caractères sont faits par des caractères espace `' '`, deux trente-troisièmes par `'a'`, etc. Chaque fréquence est un nombre rationnel dans $[0, 1]$, et la somme de toute fréquence dans la table donnera toujours exactement 1. On peut en fait vérifier que

$$\frac{3}{33}+\frac{1}{33}+\frac{2}{33}+\frac{1}{33}+\frac{3}{33}+\frac{2}{33}+\frac{4}{33}+\frac{2}{33}+\frac{3}{33}+\frac{6}{33}+\frac{4}{33}+\frac{1}{33}+\frac{1}{33} \quad = \quad \frac{33}{33} \quad = \quad 1$$

2.3 Tables de fréquence et langues

En analysant des textes de taille importante on remarque que chaque langue est associée à une table de fréquence caractéristique, différente par rapport aux tables de fréquence des autres langues même si les langues utilisent le même alphabet. Par exemple en français le caractère `'e'` est le plus commun : sans compter les espaces et les signes de ponctuation, en moyenne environs 15% d'un texte est fait par des lettres `'e'` (minuscules, sans accents). Ces percentages ne sont pas les mêmes dans d'autres langues.

Notre phrase d'exemple `'Les ordinateurs sont intéressants'` n'est pas un bon échantillon statistique de la langue française. Mais un texte de taille suffisamment grande, par exemple un roman, le sera.

En utilisant des textes-modèles écrits en chaque langue, supposés être des bons exemples typiques de la langue en question et de taille suffisante, on peut construire une table de fréquence « idéale » pour chaque langue. Nous allons dorénavant *identifier chaque langue avec sa table de fréquence.*

Un texte écrit dans une langue à déterminer, encore inconnue, aura *sa* table de fréquence. Nous faisons l'hypothèse, réaliste pour des textes de taille suffisante, que la langue ayant la table de fréquence idéale *la plus proche* de cela du texte sera la langue du texte.

Cette idée intuitive de *distance* entre deux tables de fréquence sera développée dans la section suivante.

2.4 Distance entre deux tables de fréquence

Deux tables de fréquence peuvent être plus ou moins similaires entre elles.

Imaginons un cas simplifié où trois langues nommées $\mathcal{X}$, $\mathcal{Y}$ et $\mathcal{Z}$ n'utilisent que les deux lettres `'a'` et `'b'` et où les tables de fréquence de $\mathcal{X}$, $\mathcal{Y}$ et $\mathcal{Z}$ soient :

$\mathcal{X}$		$\mathcal{Y}$		$\mathcal{Z}$	
`'a'`	0,5	`'a'`	0,47	`'a'`	0,01
`'b'`	0,5	`'b'`	0,53	`'b'`	0,99

C'est intuitivement clair que la « distance » entre les langues $\mathcal{X}$ et $\mathcal{Y}$, ayant une fréquence similaire entre les caractères `'a'` et `'b'`, devrait être mineure par rapport à la distance entre $\mathcal{X}$ et $\mathcal{Z}$ ou par rapport à la distance entre $\mathcal{Y}$ et $\mathcal{Z}$. La langue $\mathcal{Z}$ est beaucoup moins « balancée » : dans $\mathcal{Z}$ la lettre `'b'` est beaucoup plus commune que la lettre `'a'`, ce qui n'est pas le cas en $\mathcal{X}$ et $\mathcal{Y}$.

Pour formaliser la notion de distance entre deux tables de fréquence nous pouvons utiliser la notion géométrique de *distance euclidienne*[1], normalement exprimée par le théorème de Pythagore.

2.4.1 Distance entre deux tables de fréquence, simplifiée : deux lettres

Soient $\mathcal{L}$ et $\mathcal{M}$ deux langues utilisant, au lieu d'un alphabet de taille ordinaire, juste les deux caractères `'a'` et `'b'`, avec fréquences $f_{\mathtt{a}}^{\mathcal{L}}$, $f_{\mathtt{b}}^{\mathcal{L}}$, $f_{\mathtt{a}}^{\mathcal{M}}$ et $f_{\mathtt{b}}^{\mathcal{M}}$. La distance entre $\mathcal{L}$ et $\mathcal{M}$ sera

$$d_2(\mathcal{L},\mathcal{M}) = \sqrt{(f_{\mathtt{a}}^{\mathcal{L}} - f_{\mathtt{a}}^{\mathcal{M}})^2 + (f_{\mathtt{b}}^{\mathcal{L}} - f_{\mathtt{b}}^{\mathcal{M}})^2}$$

En utilisant cette formule et les tables dans §2.4 on peut calculer que $d_2(\mathcal{X},\mathcal{Y}) = d_2(\mathcal{Y},\mathcal{X}) \sim 0{,}042$, que $d_2(\mathcal{X},\mathcal{Z}) = d_2(\mathcal{Z},\mathcal{X}) \sim 0{,}693$, et que $d_2(\mathcal{Y},\mathcal{Z}) = d_2(\mathcal{Z},\mathcal{Y}) \sim 0{,}651$; on a donc déterminé *formellement* que la distance entre les langues $\mathcal{X}$ et $\mathcal{Y}$ est plus faible que la distance entre les langues $\mathcal{X}$ et $\mathcal{Z}$ ou la distance entre les langues $\mathcal{Y}$ et $\mathcal{Z}$.

On remarque aussi que la distance entre une langue et elle-même est zéro : $d_2(\mathcal{X},\mathcal{X}) = d_2(\mathcal{Y},\mathcal{Y}) = d_2(\mathcal{Z},\mathcal{Z}) = 0$.

2.4.2 Distance entre deux tables de fréquence, simplifiée : trois lettres

En ajoutant une troisième lettre `'c'` aux langues $\mathcal{L}$ et $\mathcal{M}$ la distance euclidienne peut encore s'exprimer par le théorème de Pythagore, cette fois *en trois dimensions* :

$$d_3(\mathcal{L},\mathcal{M}) = \sqrt{(f_{\mathtt{a}}^{\mathcal{L}} - f_{\mathtt{a}}^{\mathcal{M}})^2 + (f_{\mathtt{b}}^{\mathcal{L}} - f_{\mathtt{b}}^{\mathcal{M}})^2 + (f_{\mathtt{c}}^{\mathcal{L}} - f_{\mathtt{c}}^{\mathcal{M}})^2}$$

À ce point on peut voir facilement comment généraliser à des alphabets de taille plus grande : par exemple en utilisant quatre lettres on aurait une distance d_4, définie en utilisant quatre termes à sommer à l'intérieur de la racine carrée.

2.4.3 Distance entre deux tables de fréquence : définition générale

Les langues réelles utilisent, bien sûr, un nombre de caractères plus grand que deux ou trois. On peut généraliser les formules dans §2.4.1 et §2.4.2 à un alphabet générique A (contenant au moins tout caractère dans *les deux* langues $\mathcal{L}$ et $\mathcal{M}$), en obtenant cette *définition de distance entre deux tables de fréquence* :

$$d(\mathcal{L},\mathcal{M}) = \sqrt{\sum_{l\in A} (f_l^{\mathcal{L}} - f_l^{\mathcal{M}})^2}$$

Le signe $\sum$ est simplement une façon compacte d'indiquer une somme : ici à l'intérieur de la racine carrée on aura un terme à sommer $(f_l^{\mathcal{L}} - f_l^{\mathcal{M}})^2$ pour chaque caractère l contenu dans l'alphabet A.

On remarque que $d(\mathcal{L},\mathcal{M})$ est en accord avec $d_2(\mathcal{L},\mathcal{M})$ ou $d_3(\mathcal{L},\mathcal{M})$ pour des langues $\mathcal{L}$ et $\mathcal{M}$ utilisant deux ou, respectivement, trois caractères.

On pourrait même vérifier que toute propriété mathématique d'une distance ou *métrique*[2] est satisfaite par notre définition de d, et également par ses cas particuliers d_2 et d_3.

3 Implémentation

La fonction principale à implémenter s'appellera `langue`.

`langue` accepte un paramètre de type `str`, le texte dont il faut reconnaître la langue ; elle renvoie le nom de la langue du paramètre, déterminée en cherchant la langue connue dont la table de fréquence a la plus petite distance avec la table de fréquence du paramètre.

Par exemple, si `s` est une chaîne de caractères contenant la citation de Balzac de page 1, `langue (s)` renverra comme résultat `'français'`, ou `'french'`.

1. Voir aussi https://en.wikipedia.org/wiki/Euclidean_distance et le début de https://en.wikipedia.org/wiki/Pythagorean_theorem#Euclidean_distance_in_various_coordinate_systems. Le fait que les formules soient aussi similaires à ces données ici n'est pas une coïncidence.

2. Voir, par exemple, https://fr.wikipedia.org/wiki/M%C3%A9trique_(math%C3%A9matiques)#D%C3%A9finitions.

Encore une fois on remarque que la distance entre une langue et elle-même sera zéro : $d(\mathcal{L},\mathcal{L}) = 0$ pour toute $\mathcal{L}$; on voit aussi que pour deux langues quelconques $\mathcal{L}$ et $\mathcal{M}$ on aura toujours $d(\mathcal{L},\mathcal{M}) = d(\mathcal{M},\mathcal{L})$. Ce signifie, intuitivement, que chaque langue est identique à elle-même et que, par exemple, la distance entre l'allemand et le russe est égale à la distance entre le russe et l'allemand : d est *symétrique*.

L'*identité des indiscernables* et l'*inégalité triangulaire* (https://fr.wikipedia.org/wiki/In%C3%A9galit%C3%A9_triangulaire) sont également satisfaites.

Le fait que notre notion de distance entre langues ait un comportement mathématique aussi propre ne devrait pas être surprenant : nous avons simplement appliqué aux langues une notion géométrique élémentaire, *en identifiant une langue avec un point* en 2, 3 ou n dimensions — où n est le nombre de caractères de l'alphabet A.

3.1 Fichiers fournis

Le fichier disponible à l'adresse http://ageinghacker.net/teaching/programming-python/corpora/corpora-version-1.tar.gz est un archive contenant un script de shell `generate-python-source.sh` et plusieurs fichiers de texte de taille importante, un par langue. Le script sert à générer (sur sa sortie standard) un fichier Python contenant des définitions de variables globales. L'archive et le script de shell sont particulièrement utiles pour étendre l'ensemble des langues supportées, en ajoutant des nouveaux textes fournis par vous.

Le fichier généré par `generate-python-source.sh` en utilisant les textes fournis est également disponible, déjà prêt, à l'adresse http://ageinghacker.net/teaching/programming-python/corpora/langues.py, et peut être téléchargé directement et importé comme module à partir de vos sources.

3.1.1 Contenu du fichier (généré) `langues.py`

Pour toute langue supportée $\mathcal{L}$ le fichier contient une définition de variable globale `text_`$\mathcal{L}$; la valeur de chaque variable `text_`$\mathcal{L}$ est une longue chaîne de caractères contenant le texte « canonique » dans la langue.

Les noms des langues $\mathcal{L}$ sont *en anglais*, et tous en minuscules ; quand le nom de la langue est composé par plusieurs mots les mots sont séparés par des tirets-bas (`'_'`). Par exemple le module `langues` contient des variables globales nommées `text_english`, `text_russian` et `text_ancient_greek`.

Les textes contiennent de la ponctuation, des espaces et des caractères spéciaux comme les retours à la ligne `'\n'`. Si vous souhaitez ignorer certains caractères dans vos tables de fréquence, c'est à vous de filtrer les caractères que vous considérez comme inutiles.

Le module `langues` contient aussi un dictionnaire dans une variable globale nommée `language_name_to_text`, ayant par clés les noms de toutes langues $\mathcal{L}$ (en tant que chaînes de caractères) et par valeurs les textes dans chaque langue.

Les clés du dictionnaire sont nommées comme chaque « $\mathcal{L}$ », c'est à dire comme les variables « `text_`$\mathcal{L}$ » sans le prefixe « `text_` ».

4 Phases du travail

Cet ordre d'implémentation est *fortement* recommandé pour pouvoir bien tester votre code quand il n'est pas encore fini. Ces conseils sont particulièrement orientés aux débutants.

4.1 Début

Téléchargez ou régénérez `langues.py` à partir des URLs donnés dans §3.1. Créez un nouveau répertoire `projet-langues` ; rentrez dans le répertoire avec la commande `cd`. Copiez `langues.py` dans le répertoire courant. Ouvrez votre éditeur de texte préféré. Dans votre nouveau fichier ajoutez les deux lignes initiales (shebang et encodage). Importez le module `langues`.

4.2 Calculer une table d'occurrence : `de_str_a_table_doccurrence`

Définissez une fonction `de_str_a_table_doccurrence` qui, étant donnée une chaîne de caractères, renvoie la table d'occurrence correspondante comme spécifié dans §2.1.

Une table d'occurrence est à implémenter par un dictionnaire ayant des caractères par clés, et les nombres de leurs occurrences comme données associées.

Testez votre fonction interactivement, en utilisant `ipython`, sur des chaînes de petite taille où vous pouvez vérifier la résultat à la main. Par exemple la chaîne `'aabq'` contient `'a'` deux fois, et `'b'` et `'q'` une fois chacun.

4.3 Calculer une table de fréquence : `de_str_a_table_de_frequence`

Définissez une fonction `de_str_a_table_de_frequence` qui, étant donnée une chaîne de caractères, renvoie la table de fréquence correspondante comme spécifié dans §2.2. Votre fonction doit utiliser internement `de_str_a_table_doccurrence` pour calculer son résultat.

La table de fréquence sera encore à implémenter comme un dictionnaire, de façon très similaire aux tables d'occurrence ; mais les données associées ne seront plus entières ici.

Étant donné une table d'occurrence et la taille de la chaîne (ou le nombre de caractères « intéressants » selon votre critère : voir la tâche optionnelle dans §5.1) c'est très facile de calculer une table de fréquence : pour chaque nombre associé à une clé du dictionnaire, il faudra faire une division, et soit modifier le dictionnaire utilisé pour la table d'occurrence, soit en remplir un autre. Le nouveau dictionnaire sera à renvoyer à la fin.

Testez interactivement avec `ipython`, sur des petites chaînes. Par exemple `de_str_a_table_de_frequence ('aabc')` doit donner comme résultat `{'a': 0.5, 'b': 0.25, 'c': 0.25}`.

4.4 Calculer les tables de fréquence des langues données

Construisez un nouveau dictionnaire `tables_de_frequences` ayant les noms des langues comme clés (comme `language_name_to_text`), et des tables de fréquence (donc des autres dictionnaires) comme les données associées.

Vous devez itérer avec une boucle `for` sur `language_name_to_text` et appeler `de_str_a_table_de_frequence` dans la boucle.

4.5 Distance entre deux tables de fréquence : `distance`

Définissez une fonction `distance` à deux paramètres. Étant données deux tables de fréquence, la fonction renverra leur distance en utilisant la formule dans §2.4.3.

Remarque : c'est possible que un caractère soit dans une des tables de fréquence, mais pas dans l'autre ; par exemple un idéogramme chinois ne va probablement pas apparaître du tout dans la table de fréquence du français. Un caractère qui n'est pas contenu comme clé dans une table de fréquence a une fréquence 0, à utiliser pour le calcul de la distance ; un caractère contenu dans une table mais pas dans l'autre *ne constitue pas une erreur*.

Il faut itérer sur l'*union* des caractères qui sont les clés des deux dictionnaires. (Suggestion : cette union peut être réalisée dans une fonction auxiliaire.)

La fonction racine carrée est disponible en Python dans le module `math`, que vous devrez importer. Elle s'appelle `math.sqrt` (en anglais, *square root*).

Testez la fonction interactivement dans `ipython`, en utilisant des tables de fréquence simples obtenues en utilisant des chaînes de taille faible données par vous, et aussi en utilisant les tables de fréquence pour $\mathcal{X}$, $\mathcal{Y}$ et $\mathcal{Z}$ données dans §2.4, où vous pouvez vérifier les valeurs correctes des distances.

Testez aussi en utilisant des petits exemples où les ensembles de caractères dans les deux langues ne sont pas identiques.

Vérifiez au moins que :

— la distance entre une langue et elle-même soit zéro ;

— la distance entre $\mathcal{L}$ et $\mathcal{M}$ soit égale à la distance entre $\mathcal{M}$ et $\mathcal{L}$ pour deux langues différentes $\mathcal{L}$ et $\mathcal{M}$.

4.6 La fonction principale : `langue`

Définissez la fonction principale `langue`, d'un paramètre. La fonction calculera la table de fréquence de son paramètre (un texte, de type `str`), et cherchera la langue de distance minimale en itérant sur le dictionnaire `tables_de_frequences`. Il s'agit de calculer la distance minimale entre la table de fréquence du texte donné, et les tables de fréquence des langues connues. Renvoyez le nom de la langue de distance minimale.

Cette fonction est similaire à la fonction minimum, que vous avez implémenté au TP.

Testez.

4.7 Programme principale

Appelez la fonction `langue`, peut-être dans un boucle, en lui donnant le résultat d'un appel de `raw_input` (Python 2) ou `input` (Python 3) et affichez son résultat.

(Si vous avez déjà utilisé `raw_input` ou `input` avant cette tâche, ou `print` hors debugging, vous avez mal travaillé. Les fonctions doivent travailler *sur leur paramètres* et *donner des résultats*, pas polluer l'entrée et la sortie.)

Testez.

Fêtez.

5 Tâches optionnelles

Les tâches spécifiées ici ne sont pas obligatoires, mais vous pouvez les réaliser pour améliorer votre note, après avoir complété la partie principale.

La description de ces tâches est, intentionnellement, laissée plus vague pour vous donner la possibilité d'explorer le système en lisant la documentation, et plus d'espace pour expérimenter de façon créative.

5.1 Caractères « inutiles »

Vous pouvez modifier votre programme pour ignorer certains caractères que vous considérez comme non intéressants pour reconnaître une langue, comme par exemple les retours à la ligne (mais nous ne conseillerions pas d'ignorer les espaces ; pourquoi ?).

5.2 Ligne de commande

Votre programme Python peut devenir, plutôt facilement, un vrai outil Unix réutilisable.

Votre programme peut lire son texte d'entrée à partir de son *entrée standard* ou à partir d'un fichier dont le nom est donnée dans la ligne de commande ; il affichera sa sortie (le nom de la langue) sur sa sortie standard.

Pour connaître le nom du fichier d'entrée ou n'importe quelle autre variation de comportement du programme que vous avez devisé vous pouvez utiliser des *options à ligne de commande* (par exemple "`-l`" dans "`ls -l /tmp`" est une option à ligne de commande) et des autres paramètres non-options (par exemple "`/tmp`" dans "`ls -l /tmp`" est un paramètre non-option).

Vous aurez besoin de `sys.argv` et du module `getopt`.

5.3 Tables pré-calculées

Au lieu de calculer les tables de fréquence de toute langue au démarrage de votre programme, vous pouvez les calculer juste une fois, et les sauvegarder dans un fichier qui sera lu à l'initialisation. Cette technique devrait rendre le demarrage de votre programme plus rapide.

Utilisez *un seul* fichier, pour toute langue. Le module `pickle` sera utile.

Soumettez aussi le code (Python) avec lequel vous aurez généré votre fichier, même si ce code n'est plus utilisé à temps d'exécution.

5.4 Tables de fréquence *par mot*

Dans les langues utilisant des espaces pour séparer les mots (ces qui sont la majorité : les exceptions principales étant le chinois et le japonais) vous pouvez calculer des tables de fréquence *par mot* au lieu que par lettre. Par exemple le mot plus commun en anglais est *the*, an allemand *der*. Même quand un mot existe en plusieurs langues (par exemple *a*, qui est un mot correct au moins en français, en anglais et en italien : respectivement une forme verbale, un article indéterminé, une préposition), sa fréquence sera différente en chaque langue.

L'idée des tables d'occurrence et de fréquence par mots est *identique* à cela des tables d'occurrence et de fréquence par caractères, mais couper les textes en mots est plus laborieux par rapport à une simple boucle `for` sur une chaîne.

Quelle technique, entre les tables par lettre et par mot, donne un résultat plus fiable pour des textes de petite taille ? Et pour des textes plus longs ? Pouvez-vous implémenter les deux, et choisir la meilleure technique selon l'entrée ?

6 Suggestions et remarques

6.1 Caractères non ASCII

En Python la taille d'une chaîne de caractères non ASCII, par exemple des voyelles avec des accents en français, peut ne pas être égale au nombre des caractères.

Par exemple `len ('abc')` donne comme résultat `3`, mais `len ('àbc')` peut donner `4`. En fait la « taille » d'une chaîne, dans certaines versions de Python, est définie comme le nombre d'*octets*, et non de caractères, contenus dans la chaîne : en particulier dans l'encodage UTF-8 la taille en mémoire de chaque caractère est variable, et le caractère `à` est représenté par *deux* octets même si `a` et `b` occupent juste un octet chacun.

Vous pouvez complètement ignorer cette complication et considérer, par exemple, chacun des deux octets dans `à` comme un « caractère » séparé. Cette variante semble quand même reconnaître les langues de façon fiable, selon nos expériences.

Le comportement est différent entre la version 2 et la version 3 de Python. Faites attention à la version que vous utilisez et utilisez toujours la même version de façon cohérente, ou bien vérifiez la compatibilité avec plusieurs versions.