TP : Mise en œuvre du langage `Python`

Vincent Berry, Christophe Fiorio, Tanmoy Modal

1 Dictionnaires

* Exercice 1 création d'un dictionnaire

Écrire une fonction qui renvoie un dictionnaire associant des noms d'aéroport à leurs indicatifs. Voici quelques exemples :

MPL Montpellier

DUB Dublin

CDG Paris Charles-de-Gaule

ORY Paris Orly

YYZ Toronto

LHR London

LAX Los Angeles

On a fait une erreur dans la liste précédente : `LHR` est l'indicatif de l'aéroport de *London Heathrow* et pas de Londres. En outre il faudrait rajouter l'autre aéroport de Londres, l'aéroport *London City* qui a pour indicatif `LCY`.

Écrire la séquence de code Python qui corrige le dictionnaire.

* Exercice 2 création d'un dictionnaire à partir de 2 listes

Écrire une fonction qui prend 2 listes en paramètres et qui renvoie un dictionnaire dont les clefs sont données par la première liste et les valeurs par la seconde liste telles que la $n^{\text{ième}}$ valeur de la première liste est la clef de la $n^{\text{ième}}$ valeur de la seconde liste

* Exercice 3 Recherche d'indicatif

Écrire une fonction qui retourne l'indicatif d'un aéroport.

* Exercice 4 Recherche d'indicatif

Écrire la séquence de code qui imprime la liste des aéroports avec leur indicatifs dans l'ordre alphabétique des indicatifs.

* Exercice 5 Recherche d'indicatif

Même chose que précédemment mais dans l'ordre alphabétique des noms d'aéroport.

2 Expressions régulières

Les expressions régulières sont des outils très puissants que l'on retrouve dans plusieurs langages, mais aussi éditeurs de texte comme *Emacs*. Elles permettent de vérifier si une chaîne de caractères, ou une partie d'une chaîne de caractères, a la forme voulue. Cela permet donc de vérifier des formats.

Une expression régulières utilise un certain nombre de symboles pour exprimer le motif à reconnaître. On retrouve à peu près les mêmes dans tous les langages. Savoir maîtriser les expressions régulières fait donc parti du bagage de base de tout informaticien.

Les caractères utilisés pour définir des motifs sont :

. correspond à n'importe quel caractère

∧ indique le début de la chaîne ou *not* si placé devant un groupe

$ indique la fin d'une chaîne de caractère

[abc] indique une liste de caractères possibles, ici les caractères *'a'*, *'b'* ou *'c'*.

(...) indique un groupe

(a|b) indique une alternative : ici le caractère *'a'* ou le caractère *'b'*.

\d équivaut à $[0-9]$, c'est à dire à un chiffre compris entre 0 et 9.

\D n'est pas un chiffre

\s un espace, une tabulation, un retour chariot, ou tout autre caractère spécial qui fait une séparation.

\S pas un espace

\w un caractère alphanumérique ; équivalent à $[a-zA-Z0-9\_]$.

\W pas un caractère alphanumérique ; équivalent à $[\wedge a-zA-Z0-9\_]$.

+ placé après un groupe ou une liste – `[ ]` ou `( )` – indique au moins un élément du groupe

? placé après un groupe ou une liste – `[ ]` ou `( )` – indique au plus un élément du groupe

***** placé après un groupe ou une liste – `[ ]` ou `( )` – indique 0 ou plus éléments du groupe

{n} placés après un groupe ou une liste indique n occurences du groupe ou de la liste

\n dans la fonction `sub` permet d'indiquer le $n^{\text{ième}}$ groupe de l'expression régulière

Attention, en Python le caractère \ dans une chaîne de caractères désigne un caractère spécial. Ainsi par exemple, \n désigne un retour chariot. On utilisera donc les chaînes brutes (*raw string*) de python qui n'interprètent pas les caractères spéciaux. Une chaîne brute est une chaîne précédé du caractère `'r'`. Ainsi la chaîne `r'\n'` est une chaîne de 2 caractères, les caractères \ et `n`, alors que la chaîne `' \n'` est une chaîne à un seul caractère, le caractère fin de ligne.

En python, grâce au module `re`, on dispose d'un certains nombre de fonctions qui permettent de manipuler les expressions régulières. Les deux plus courantes qu'on utilisera dans ces exercices sont les fonctions *match*, *search*, *findall* et *sub*.

match(regexp, string) vérifie si le début d'une chaîne de caractères correspond à une expression régulière ; `match` renvoie `None` si ce n'est pas le cas.

search(regexp, string) fait la même chose que `match` mais cherche l'expression régulière dans toute la chaîne de caractères.

findall(regexp, string) renvoie un tableau de toutes les sous-chaînes de caractères correspondant à l'expression régulière.

sub(regexp,replace,string) retourne la chaîne de caractères obtenue en remplaçant dans `replace` les motifs définis par `regexp` et trouvés dans `string`

Pour les exercices suivants, il faut com par importer le module *re*. N'oubliez pas non plus de préfixer les fonctions *match*, *search*, *findall* et *sub* par `re.` puisque ces fonctions sont définies dans le module *re*.

* Exercice 6 Vérification simple d'adresse mel

Écrire une fonction qui vérifie si une chaîne de caractère est bien une adresse email. dans un premier temps, on se contentera de vérifier que la chaîne comprend un `@`.

* Exercice 7 Vérification formal nom de mel

Cette fois-ci on vérifiera que la chaîne avant le `@` est bien formatée, c'est à dire ne contient que des lettres, des chiffres, des `.`, `-` ou `_`.

* Exercice 8 Vérification d'adresse mel

Écrire une fonction qui vérifie si une chaîne de caractère est bien une adresse email (on se limitera aux domaines `.fr`, `.com` et `.org`).

* Exercice 9 Vérification format numéro de téléphone

Écrire une fonction qui vérifie si une chaîne de caractère est bien un numéro de téléphone. Le numéro peut, ou pas, comporter une indicatif international sous la forme de 2 chiffres après `+` ou 2 chiffres entre parenthèses. Si l'indicatif international est présent, le numéro de téléphone est composé de 9 chiffres (le 0 est omis) qui peuvent comporter entre chaque chiffre un espace, un `.` ou un tiret. Si l'indicatif international n'est pas présent, le 0 doit être présent.

* Exercice 10 Simplification numéro de téléphone

Écrire une fonction qui prend une chaîne de caractères en paramètre et renvoie `None` si elle ne correspond pas à un numéro de téléphone, ou renvoie deux chaînes de caractères : l'indicatif et le numéro de téléphone simplifié à 9 chiffres.

* Exercice 11 Expression formatée

Écrire une fonction qui vérifie si une chaîne de caractère contient une séquence de 1 à 3 lettres, suivie d'une séquence de 1 à 2 chiffres, suivie de la même séquence de chiffres suivie de la première séquence de lettres sans tenir compte de la casse des lettres. Les chaînes *ab55ab* et *abC4141aBc* sont deux exemples de chaînes vérifiant le format demandé.

3 Fichiers

* Exercice 12 Lecture csv

Écrire une fonction qui lie le fichier csv de données et range ces données dans un dictionnaire. La fonction devra vérifier si les lignes de données sont au bon format, c'est à dire :

1. un code de 3 lettres en majuscule
2. un code de 4 lettres en majuscule
3. une chaîne de caractères
4. une chaîne de caractères
5. une chaîne de caractères

Le dictionnaire résultat a pour clef le premier code de 3 lettres. Chaque valeur du dictionnaire correspond à la ligne codée par un dictionnaire dont les clefs sont les valeurs de la première ligne du CSV.

* Exercice 13 Lecture csv

Écrire à nouveau la fonction de l'exercice 3 mais avec le dictionnaire obtenu à l'exercice 12.

* Exercice 14 Écriture csv csv

Écrire une fonction qui à partir du dictionnaire obtenu à l'exercice 12 écrit un nouveau fichier csv, classé par ordre alphabétique des noms d'aéroport, dont la structure correspond à celle du dictionnaire de l'exercice 1.

* Exercice 15 Écriture csv csv

Écrire une fonction qui lit le fichier obtenu à l'exercice 14 pour le stocker dans un dictionnaire utilisable pour les exercices 3, 4 et 5.

Tester à nouveau vos fonctions de ces exercices avec le dictionnaire ainsi obtenu ici.