Exercices de Programmation & Algorithmique 1

Série 4 – Itérations et Chaînes de caractères
(15 octobre 2020)

Département d'Informatique – Faculté des Sciences – UMONS

Pré-requis : Boucles while; boucles for; invocation de méthodes sur les chaînes de caractères et lecture de fichier texte (cours jusqu'au **Chapitre 6**).

Objectifs : Etre capable de concevoir des fonctions itératives; de manipuler les chaînes de caractères et d'interagir avec l'utilisateur.

---

1 Le contrat

1.1 À réaliser sur papier

> **Important** : avant de passer à la suite du contrat (sur machine); vous devez faire valider vos solutions par un assistant ou élève-assistant.

Le jeu du *pendu* se joue à deux joueurs. Le premier essaye de deviner, lettre par lettre, le mot choisi par le second en un nombre d'essais limité. Dans le cadre de cette séance, veuillez considérer les points suivants :

- Le joueur 2 doit choisir un mot. Ce mot doit être présent dans le fichier `words.txt`.
- Le nombre de tentatives infructueuses pour deviner le mot est fixé à 10.
- À chaque étape, avant que le joueur 1 ne propose une lettre, il doit être affiché le nombre de tours restants, les lettres déjà jouées, ainsi que le mot à deviner dans lequel les lettres non-trouvées sont représentées par "*".
- À chaque étape, le joueur 1 propose une lettre. Si cette lettre n'a pas encore été jouée, le jeu vérifie si elle est bonne (présente dans le mot à deviner), ou mauvaise.
- À la fin de la partie, si le mot a été trouvé, le score est de

$$\text{nbr. de lettres différentes dans le mot} + \text{nbr. d'essais restants} - \text{nbr. de lettres essayées}$$

1 Sur papier et en français, identifiez les grandes étapes nécessaires à une solution algorithmique pour le jeu du pendu. Ensuite, vous pouvez détailler ces grandes étapes pour obtenir un niveau d'analyse facilement traduisible en Python. Cette démarche s'appelle une *analyse top-down*. Dans le cadre de cette analyse, vous n'êtes pas obligé d'aller trop "bas" dans le niveau de détails. Par exemple, il est inutile de préciser le comportement des tâches telles que :

- Entrer un mot ;
- Vérifier que le mot fait partie du fichier `words.txt` ;
- Entrer une lettre.

1.2 À réaliser sur machine

2 Créez un module `userInput`. Implémentez-y les fonctions `belongs_to_dictionary(word)`, `ask_word_in_dictionary()`, et `ask_letter(tried_letters)` comme définies ci-après. La fonction `belongs_to_dictionary` retourne vrai ou faux, suivant que le mot passé en argument appartient au dictionnaire défini par le fichier `words.txt` (sans prendre en compte la casse) ou non. La fonction `ask_word_in_dictionary` demande à l'utilisateur de rentrer un mot. Tant que ce dernier n'est pas un mot du dictionnaire, la fonction demande à nouveau à l'utilisateur d'entrer un mot. Autrement, la fonction retourne ce mot. La fonction `ask_letter(tried_letters)`

demande à l'utilisateur d'entrer **une** lettre qui ne fait pas partie de la liste de lettres contenues dans `tried_letters`. Tant que l'entrée n'est pas valide, la fonction demande à l'utilisateur une nouvelle entrée. Quand l'entrée est correcte, la lettre est retournée.

3 Implémentez, dans un module `hangman` (pendu en anglais), le jeu du pendu en respectant votre analyse. Découpez votre implémentation en fonctions (qui peuvent correspondre à certaines des étapes identifiées dans votre analyse) et pensez à utiliser votre fonction `ask_word_in_dictionary`. Vous pouvez utiliser le module `hangmantui` (Text-based User Interface pour le pendu) afin d'afficher, sur la console, les étapes du pendu. Vous pouvez les fonctions suivantes de ce module :

`hangman(tries)` : affiche un pendu à 10 étapes, pour le lequel il reste `tries` essais restants avant de perdre ;

`clear()` : efface l'écran, entre deux affichages.

2 Exercices complémentaires

Rappel. Légende concernant les exercices complémentaires :

Exam	Date d'une question demandée lors d'un examen ou d'un test (disponible sur moodle avec ou sans corrigé).
★☆☆	Exercice complémentaire – relativement simple ou direct – dont le but est d'aider l'étudiant ayant des difficultés à remplir le contrat de cette série. Il permet par exemple, si nécessaire, de revoir certaines notions de manière plus progressive, avant de s'attaquer au contrat en lui-même.
★★☆	Exercice complémentaire dont le niveau est proche de celui du contrat.
★★★	Exercice complémentaire constituant un challenge plus important ou incluant des subtilités.

Implémentez les fonctions suivantes. Par convention, nous considérons que les positions sont comprises entre 1 et la taille du mot, contrairement à Python. Lors de votre implémentation, n'oubliez pas de tenir compte de ce choix. Pensez à écrire des tests unités pour chacune de ces fonctions. Il est préférable d'écrire ces tests avant d'entamer l'implémentation des fonctions.

★☆☆ 4 Une fonction `caractere(s, i)` qui retourne le i-ème caractère de la chaine de caractères `s`. Si ce caractère n'existe pas, retourne `None`.

Tests :

```
caractere('happy', 1)  →  'h'
caractere('happy', 0)  →  None
caractere('happy', 8)  →  None
```

★☆☆ 5 Une fonction `caracteres(s, i, j)` qui retourne les caractères compris entre la position `i` et la position `j` (bornes inclues). Si ces bornes sont invalides, retourne `None`.

Tests :

```
caracteres('happy', 2, 4)  →  'pp'
caracteres('happy', 0, 2)  →  'ha'
caracteres('happy', 2, 8)  →  None
```

★☆☆ 6 Une fonction `change_caractere(s, i, a)` qui retourne une chaîne de caractères identique à `s` dans laquelle le i-ème caractère a été remplacé par le caractère stocké dans `a`. Si la position n'est pas valide, retourne `None`.

Tests :

```
change_caractere('happy', 1, 'p')  →  'pappy'
change_caractere('happy', 0, 'p')  →  None
```

★☆☆ 7 Une fonction `change_caracteres(s, i, j, t)` qui retourne une chaîne de caractères identique à `s` dans laquelle les caractères situés entre la position `i` et `j` (bornes inclues) ont été remplacés par la chaîne de caractères stockée dans `t`. Si les bornes ne sont pas correctes, retourne `None`.

Tests :

```
change_caracteres('happy', 1, 3, 'you')  →  'youpy'
change_caracteres('happy', 2, 2, 'oi')  →  'hoippy'
change_caracteres('happy',-1, 2, 'oi')  →  None
```

★☆☆ 8 Une fonction `decouvre(s1, s2, x)` qui prend en entrée un mot `s1`, le même mot mais partiellement masqué (par des symboles `*`) dans le paramètre `s2` et un caractère `x`. Cette fonction retourne le mot `s2` dans lequel les caractères cachés qui correspondent à la lettre `x` ont été découverts.

Tests :
```
decouvre('abcba', 'a***a', 'a')  →  'a***a'
decouvre('abcba', 'a***a', 'b')  →  'ab*ba'
decouvre('abcba', 'a***a', 'c')  →  'a*c*a'
decouvre('abcba', 'a***a', 'd')  →  'a***a'
```

Implémentez les fonctions suivantes :

★★☆ 9 Une fonction `plus_grand_bord(w)` qui, étant donné un mot `w`, retourne le plus grand bord de ce mot. On dit qu'un mot u est un bord de w (avec $u \neq w$) si u est à la fois un préfixe (non-vide) de w et un suffixe (non-vide) de w. Si w n'a pas de bord, la fonction retourne `None`.

Tests :
```
plus_grand_bord('abdabda')  →  'abda'
plus_grand_bord('souris')  →  's'
plus_grand_bord('happy')  →  None
```

★★★ 10 Une fonction `intersection(v, w)` qui calcule l'intersection entre `v` et `w`. On définit l'intersection de deux mots comme étant la plus grande partie commune à ces deux mots.

Tests :
```
intersection('programme', 'grammaire')  →  'gramm'
intersection('cardinalite', 'ordinateur')  →  'rdina'
```

★★★ 11 Une fonction `anagrammes(v, w)` qui retourne vrai si et seulement si les mots `v` et `w` sont anagrammes.

Tests :
```
anagrammes('marion', 'romina')  →  true
anagrammes('happy', 'papy')  →  false
```

★★☆ 12 Ajoutez au module `userInput` une série de fonctions utiles pour dialoguer avec l'utilisateur. Toutes les fonctions ci-dessous prennent une chaîne de caractères en paramètre.

(a) `convert_to_int`
Si la chaîne de caractères représente un nombre entier, la fonction retourne l'entier.
Sinon, la fonction retourne `None`.
Conseil : méthode `isdigit()` sur chaîne de caractères.

Tests :
```
convert_to_int('7')  →  7
convert_to_int('a')  →  None
```

(b) `convert_to_float`
Si la chaîne de caractères représente un nombre réel, la fonction retourne le float.
Sinon, la fonction retourne `None`.
Conseil : Pour rappel, un nombre entier est aussi un nombre réel. Les autres nombres réels (qui ne sont pas entiers) peuvent s'écrire sous la forme *partie_entière.partie_décimale*.

Tests :
```
convert_to_float('2.4')  →  2.4
convert_to_float('2.a')  →  None
```

(c) `is_one_word`
Si la chaîne de caractères représente un seul mot (ie. une succession contigue de caractères minuscules ou majuscules), la fonction retourne ce mot.
Sinon, la fonction retourne `None`.

Tests :
```
is_one_word('happy')  →  'happy'
is_one_word('happy birthday')  →  None
```

(d) `is_one_letter`
Si la chaîne de caractères représente une seule lettre, la fonction retourne la lettre.
Sinon, la fonction retourne `None`.

Tests :
```
is_one_letter('h')  →  'h'
is_one_letter('happy')  →  None
is_one_letter('5')  →  None
```

★★★ 13 Vous allez devoir suivre trois étapes afin d'adapter la fonction `belongs_to_dictionary` pour qu'elle possède le même comportement que les fonctions implémentées dans l'exercice précédent. Le processus permettant de demander à l'utilisateur d'entrer un mot jusqu'à ce que celui appartienne au fichier"word.txt" sera généralisé et dédié à une autre fonction appelée `prompt`. Voici les trois étapes à réaliser :

(a) Modifiez la fonction `belongs_to_dictionary` afin qu'elle prenne également une chaine de caractères en paramètre. Si la chaîne de caractères est un mot qui figure dans la liste des mots qui sont dans le fichier `words.txt`, la fonction retourne le mot.
Sinon, la fonction retourne `None`.

Tests :
```
belongs_to_dictionary('happy')  →  'happy'
belongs_to_dictionary('api')  →  None
```

(b) Ajoutez la fonction `prompt(message, func)` au module `userInput`. Cette fonction prend deux paramètres : une chaîne de caractères `message` et une fonction de validation `func`. Cette fonction va permettre de dialoguer avec l'utilisateur en affichant à l'écran le message `message` tant que l'utilisateur fournit une entrée qui n'est pas validée par la fonction `func`.

Autrement dit, cette fonction va attendre que l'utilisateur entre une information au clavier. Appelons cette information `entree`. Si on fournit `entree` en paramètre à la fonction `func` et qu'elle nous retourne `None`, cela signifie que l'information n'est pas validée et il faut recommencer (réafficher le message, laisser l'utilisateur entrer une information, vérifier l'information). Dès que la fonction `func` retourne autre chose que `None`, l'information est validée et la fonction `prompt` retourne ce que la fonction `func` a retourné.

Voici un exemple d'utilisation de `prompt` :
`unentier = prompt('Veuillez entrer un entier', convert_to_int)`
demandera à l'utilisateur d'entrer un entier (tant que celui-ci entre quelque chose qui n'est pas un entier), et affectera cet entier dans `unentier`.

(c) modifiez votre implémentation du pendu afin qu'il utilise la fonction `prompt` au lieu de l'ancienne fonction `belongs_to_dictionary`.

★★★ 14 **Méthode de la fausse position** La méthode de la fausse position est une méthode de recherche d'une racine d'une fonction continue. Cette méthode commence par deux points a et b représentant l'intervalle de recherche. Ces points sont choisis de telle sorte que les signes des images de a et b soient différents et donc, par extension, que la fonction étudiée coupe l'axe des x en un point situé entre a et b.
À chaque étape de cet algorithme, la droite passant par $(a, f(a))$ et $(b, f(b))$ est calculée. Cette droite, tout comme la fonction, coupe l'axe des x en un point c situé entre a et b. Une fois ce point obtenu, on calcule l'image de c par la fonction f et :
— Soit cette image est "fort proche de 0", dans ce cas on peut estimer qu'il s'agit d'une racine de la fonction,
— Soit cette image est du même signe que a. Dans ce cas, on passe à l'étape suivante en considérant l'intervalle formé de c et de b,
— Soit cette image est du même signe que b. Dans ce cas, on passe à l'étape suivante en considérant l'intervalle formé de a et de c.

Il vous est demandé d'implémenter une fonction `fausse_position(f, a, b, epsilon)` qui, étant donné une fonction `f` continue, un intervalle `a,b` et une précision `epsilon` calcule une

racine de `f` située dans l'intervalle donné. L'algorithme s'arrête quand il a trouvé une image de `f` à une distance maximale `epsilon` de zéro. Si la fonction `f` ne possède pas de racine dans l'intervalle donné, ou que les images de `a` et `b` ne respectent pas les hypothèses, `fausse_position` retourne `None`.

Dans le cadre de cet exercice, vous **devez** utiliser les fonctions `intersectionAbscisses(d)` et `droite(p1, p2)` du module `Droite` implémenté dans la série 2.

Exam 15 (20 novembre 2009) Problème 2 : Approximation de l'intégrale d'une fonction.

Exam 16 (19 janvier 2010) Problème 3 : Approximation de π.

Exam 17 (11 juin 2010) Problème 1 : Approximation de π.

Exam 18 (10 juin 2011) Problème 3 : Compression de chaînes.

Exam 19 (26 octobre 2011) Problème 2 : Anagrammes.