TP 3 : Chaînes de caractères, print et input

Objectif minimal : section 2 et 3.

1 Rappels sur les chaînes de caractères.

Rappels sur la manipulation. On a déja vu les chaînes de caractères, en les utilisant avec la fonction d'affichage `print`. Revoyons tout ça :

- on crée un objet de type « chaîne de caractères » (`str`) en écrivant un texte entre apostrophes (`'une chaîne'`), guillemets (`"une autre chaîne"`) ou encore entre triple guillemets, cette dernière méthode permettant de créer des chaînes s'étendant sur plusieurs lignes :

```
s=""" une chaine qui s'etend sur deux lignes.
C'est pratique pour commenter une fonction ! """
```

- l'accès à des caractères se fait comme pour les listes ou les tuples : si `s` est une chaîne de caractères, `s[0]` est son premier caractère, `s[len(s)-1]` son dernier. On peut aussi utiliser des indices négatifs (compris entre $-n$ et -1, avec n la taille de la chaîne), par exemple `s[-1]` est similaire à `s[len(s)-1]`. L'accès avec tout autre indice provoque une erreur.
- le *slicing* fonctionne de la même manière que pour les listes ou les tuples : `s[i:j]` est la sous-chaîne contenant les caractères `s[i],...,s[j-1]`. Ce mécanisme est tolérant avec les indices trop grands et trop petits, et on peut utiliser des indices négatifs ou encore un pas : `s[0:len(s):2]` (équivalent à `s[::2]`) est la chaîne constituée d'un caractère sur 2 de `s` à partir du premier, `s[len(s)-1::-1]` (équivalent à `s[::-1]`) est la chaîne constituée des caractères de `s` à l'envers. [1]
- les chaînes de caractères sont très semblables à des tuples de caractères : elles sont *immuables* (on ne peut modifier un caractère, ou utiliser des méthodes comme `append`).
- Comme pour les listes ou les tuples, concaténer deux chaînes de caractères se fait avec `+`. L'équivalent du tuple vide `()` ou de la liste vide `[]` est la chaîne vide `""`.

Conversion en chaînes de caractères. Dans la suite, on va vouloir parfois convertir un entier ou un flottant en une chaîne de caractères, et réciproquement. Comme souvent en Python, la fonction de conversion porte le nom du type que l'on souhaite, ici `str` (*string*). Par exemple :

```
>>> str(5)
'5'
>>> str(3.14)
'3.14'
>>> float('1.41')
1.41
>>> int('8559')
8559
```

Remarquez qu'on peut donner un entier naturel écrit dans une certaine base, en précisant la base Python fait tout seul la conversion :

```
>>> int('A12',16)
2578
```

Une chaîne de caractères : un itérable. Comme beaucoup de structures complexes en Python, les chaînes de caractères sont des itérables : on peut parcourir directement les caractères à l'aide d'une boucle `for`, essayez ceci :

```
for caractere in "une chaîne de caractères":
    print(caractere)
```

1. Ceci devrait être équivalent à `s[len(s)-1:-1:-1]`, d'écriture similaire à `range`. Malheureusement, le premier `-1` est compris comme `len(s)-1` et on obtient la chaîne vide avec `s[len(s)-1:-1:-1]`.

D'une chaîne à une liste et réciproquement. Comme souvent avec les itérables, il est possible « d'éclater » une chaîne en une liste, essayez `list("chaine")`. Inversement, il est possible de réunir les caractères d'une liste de caractères en un mot, essayez `"".join(['m', 'o', 't'])`.

On a utilisé ici la méthode `join` sur la chaîne de caractères vide. Ceci peut-être appliqué avec une chaîne quelconque (essayez par exemple `"-".join(['m', 'o', 't'])`). Les éléments de la chaîne peuvent être des mots plus complexes que de simples caractères, essayez !

Enfin, on peut séparer des mots d'une chaîne autour d'un motif particulier, à l'aide de la méthode `split`. Essayez par exemple `"Ceci est une phrase".split(" ")`.

En résumé, si `L` est une liste de chaînes, et `s` est une chaîne, alors `s.join(L)` renvoie une nouvelle liste où les éléments de `L` ont été réunis avec `s` comme séparateur, et si `t` est une autre chaîne, `t.split(s)` renvoie une liste, obtenue par séparation de `t` autour de `s`.

La table des caractères ASCII. Les 128 caractères de la table ASCII comprennent les lettres minuscules et majuscules non accentuées, les chiffres de 0 à 9, à peu près tous les caractères d'un clavier QWERTY (pas de lettres accentuées !), et des caractères d'espacement [2]. À chaque caractère est associé un code entre 0 et 127 (donc représentable sur 7 bits). En Python, les fonctions `ord` et `chr` permettent de passer d'un caractère à son numéro et réciproquement, essayez les boucles suivantes :

```
for i in range(97,97+26):
    print(chr(i))
```

```
for caractere in "ABCDEFGHIJKLMNOPQRSTUVWXYZ":
    print(ord(caractere))
```

```
for i in range(10): #conversion des chiffres en lettres.
    print(ord(str(i))):
```

On notera que les lettres minuscules (respectivement les lettres majuscules, les chiffres) sont codés par des numéros contigus, avec `'a'` (respectivement `'A'`, `'0'`) ayant le plus petit numéro. En fait, tous les caractères y compris accentués ont un numéro accessible via `ord` : l'ASCII limité aux caractères précédents a été étendu en un ensemble beaucoup plus gros appelé Unicode. Par exemple :

```
>>> chr(339)
'œ'
>>> chr(201)
'É'
>>> chr(1278) #un caractère cyrillique que je ne peux afficher ici !
```

2 Manipulations élémentaires de chaînes

Exercice 1. Écrire une fonction qui affiche à l'écran les préfixes d'un mot (le mot vide est un préfixe), en utilisant le slicing.

```
>>> prefixes('Python')

P
Py
Pyt
Pyth
Pytho
Python
```

Indication : slicing et boucle.

Exercice 2. Écrire une fonction `miroir`, qui renvoie une chaîne de caractères étant l'inverse de celle passée en argument.

2. La table complète se trouve par exemple ici : `http://www.table-ascii.com/`.

```
>>> miroir('Python')
'nohtyP'
```

Indication : transformer la chaîne en liste, retourner la liste, et utiliser `join` ou plus simplement utiliser un sclicing.

Exercice 3. Écrire une fonction `egales` qui teste l'égalité de deux chaînes de caractères, en utilisant le test d'égalité (`==`) ou de différence (`!=`) uniquement sur des caractères et pas sur la chaîne complète (bien que cela fonctionne !). La fonction doit renvoyer un booléen.

```
>>> egales('python','pythoN')
False
```

Indication : tester d'abord l'égalité des longueurs puis faire une boucle.

Exercice 4. Déduire des deux fonctions précédentes une fonction `palindrome`, qui teste si le mot passé en entrée est un palindrome (un mot égal à son miroir). La fonction renvoie un booléen.

```
>>> palindrome('ressasser')
True
```

Exercice 5. *Production automatique.* Écrire une fonction `chaine_plage(a,b)` qui prend en entrée deux entiers a et b (avec $a \leq b$), et qui renvoie une chaîne constituée de la concaténation des caractères de numéro a inclus à b exclus.

```
>>> chaine_plage(97,97+26)
'abcdefghijklmnopqrstuvwxyz'
>>> ord('0')
48
>>> chaine_plage(48,48+10)
'0123456789'
>>> chaine_plage(0,10000) #à tester !
```

Indication : on pourra utiliser `"".join` sur une liste de caractères, ou procéder par concaténations successives à partir d'une chaîne de caractères initialement vide.

Exercice 6. *Somme des chiffres d'un entier.*

1. Pour calculer la somme des chiffres d'un entier, on peut effectuer des divisions euclidiennes successives par 10 pour récupérer les chiffres un à un (on l'a déja vu). Écrire une fonction `sdc(n)` qui renvoie la somme des chiffres de l'entier naturel n.

2. Une autre méthode consiste à convertir l'entier en chaîne, parcourir cette chaîne en convertissant chaque caratère en chiffre, et sommer les chiffres obtenus. Écrire cette version également.

```
>>> sdc(4444**4444)
72601
```

3 print et input

La fonction `print`. Jetons un coup d'œil à la documentation de la fonction `print` :

```
>>> help(print)
Help on built-in function print in module builtins:

print(...)
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)

    Prints the values to a stream, or to sys.stdout by default.
    Optional keyword arguments:
    file:  a file-like object (stream); defaults to the current sys.stdout.
    sep:   string inserted between values, default a space.
    end:   string appended after the last value, default a newline.
    flush: whether to forcibly flush the stream.
```

Remarquons que par défaut, les objets affichés sont séparés par des espaces (`' '`), et l'affichage se termine par un retour chariot (`\n`), mais ceci peut-être modifié :

```
>>> print("mot1","mot2","mot3",sep=" xxxxxx ", end=" fin ! \n")
mot1 xxxxxx mot2 xxxxxx mot3 fin !
```

Le champ `file` sert à spécifier la destination : c'est `sys.stdout` par défaut. `stdout` signifie *standard output*, c'est-à-dire la sortie standard, qui est l'écran. On peut modifier cette sortie standard pour écrire dans un fichier, ce qu'on ne fera pas ici.

Exercice 7. En jouant avec `end`, écrire une fonction `affiche_plage(a,b)` similaire à `chaine_plage(a,b)` mais qui affiche à l'écran les caractères, séparés par des tirets `-` (la fonction ne renvoie rien). Par exemple :

```
>>> affiche_plage(97,97+26)
a-b-c-d-e-f-g-h-i-j-k-l-m-n-o-p-q-r-s-t-u-v-w-x-y-z
```

Contrainte : la fonction ne doit pas utiliser de liste.

La fonction `input`. À l'inverse de `print`, `input` permet de récupérer quelque chose depuis *l'entrée standard*, qui est par défaut, le clavier :

```
>>> help(input)
Help on built-in function input in module builtins:

input(...)
    input([prompt]) -> string

    Read a string from standard input.  The trailing newline is stripped.
    If the user hits EOF (Unix: Ctl-D, Windows: Ctl-Z+Return), raise EOFError.
    On Unix, GNU readline is used if enabled.  The prompt string, if given,
    is printed without a trailing newline before reading.
```

Cette fonction prend en paramètre optionnel une chaîne de caractères (*prompt string* dans la documentation), l'affiche avant de lire une chaîne de caractères depuis l'entrée standard (standard input). En général, on affecte cette chaîne lue à une variable. Dans l'exemple ci-dessous, `une chaîne` a été entrée au clavier. La saisie s'arête lorsqu'on appuie sur la touche Entrée.

```
>>> s=input("Entrez-moi quelque chose : ")
Entrez-moi quelque chose : une chaîne
>>> print(s)
une chaîne
```

Les fonctions de conversions de type permettent de convertir une chaîne de caractères en le type qui nous intéresse. Une conversion de type se fait sous la forme `t(x)` où `t` est le type voulu et `x` l'objet à convertir. Lorsqu'on traite des fichiers, on convertit souvent des chaînes de caractères en d'autres types.

```
>>> n=int(input("Entrez-moi un entier : "))
Entrez-moi un entier : 42
>>> (n+12)//5
10
```

Exercice 8. Écrire une fonction `somme()` (sans argument) qui fait rentrer à l'utilisateur 5 nombres (avec 5 appels à `input`) au clavier, et renvoie leur somme.

```
>>> somme()
987
451
239
10415
7
12099
```

(Dans l'exemple précédent, les 5 premiers entiers ont été rentrés à la main, le dernier et la somme renvoyée par la fonction).

Exercice 9. Lorsque l'utilisateur appuie simplement sur la touche Entrée, `input` renvoie une chaîne de caractères vide. Reprendre l'exercice précédent en une fonction `somme2()`, qui permet à l'utilisateur d'entrer autant de nombres qu'il le souhaite. Lorsqu'il appuie sur la touche Entrée sans avoir écrit de nombre, la fonction renvoie la somme.

```
>>> somme2()
58
74

132
```

Exercice 10. Reprendre l'exercice précédent en une fonction `somme3()`. Là, l'utilisateur entre les nombres en une seule fois, séparés par des espaces (on pourra utiliser `split`, notamment, voir plus haut pour les explications).

```
>>> somme3()
85 78 140 25 7 63 6985 1
7384
```

(Même chose, on a tapé la première ligne au clavier, le nombre sur la seconde est la somme).

4 Juste(s) prix

Exercice 11. *Le juste prix !* On va créer un jeu du juste prix [3], aussi connu sous le nom de « C'est plus, c'est moins ». Le principe est le suivant : l'ordinateur choisi un nombre au hasard entre 1 et 1000. Vous essayez de deviner ce nombre, et pour ce faire, vous proposez des entiers au clavier. À chaque proposition, l'ordinateur vous dit si l'entier qu'il a choisi est plus grand ou plus petit que votre proposition. Le jeu s'arrête lorsque vous donnez l'entier choisi, avec un message de félicitations.

Tout d'abord, votre programme doit choisir un entier au hasard. On va pour cela importer la fonction `randint` du module `random`. Le code suivant affecte à `a` un nombre au hasard entre 1 et 1000.

```
from random import randint
a=randint(1,1000)
```

Programmez le jeu proprement dit, sous la forme d'une fonction `juste_prix()` (ne prenant pas d'argument). Pour demander à l'utilisateur de donner un nombre, vous utiliserez `input()`. Votre fonction fera usage d'une boucle `while`, qui s'arrêtera lorsque l'utilisateur aura trouvé le nombre mystère.

Exercice 12. *Juste prix à l'envers.* À l'inverse, écrivez une fonction `juste_prix_2()`. L'ordinateur vous propose des nombres, et vous répondez par « oui », « plus » ou « moins ». Dégagez une stratégie « optimale » pour le programme, celui-ci devant être capable de détecter si vous mentez.

```
>>> juste_prix_2()
501 ? moins
251 ? plus
376 ? moins
314 ? plus
345 ? moins
330 ? moins
322 ? plus
326 ? moins
324 ? moins
323 ? moins
Tu as triché !
```

5 Plus difficile : la suite de Conway

Vous connaissez peut-être la suite *de Conway* obtenue à partir de « 1 » en lisant à voix haute le nombre de chiffres consécutifs égaux, du terme précédent. Voici les premiers termes :

— 1 (le premier terme !) ;

3. Cet exercice était déja présent dans un TP précédent, passez-le si vous l'avez déja fait !

— 11 (le précédent se lit « un 1 ») ;
— 21 (le précédent se lit « deux 1 ») ;
— 1211 (le précédent se lit « un 2, un 1 ») ;
— 111221 (le précédent se lit « un 1, un 2, deux 1 ») ;
— 312211 (le précédent se lit « trois 1, deux 2, un 1 ») ;
— etc...

Exercice 13. Écrire une fonction `suite(n)` renvoyant la liste formée des n premiers termes de cette suite. On fera un détour par les chaînes de caractères pour passer d'un terme au suivant.

```
>>> print(*suite(10), sep="\n")
1
11
21
1211
111221
312211
13112221
1113213211
31131211131221
13211311123113112211
```

Remarque : le joker `*` permet de transformer une liste en une séquence de paramètres. C'est très pratique pour `print` par exemple, qui est une fonction capable de recevoir un nombre arbitraire d'arguments.

Exercice 14. Vérifier expérimentalement que le rapport du nombre de chiffres entre un terme et le précédent tend vers une constante, appelée la constante de Conway $\lambda \simeq 1.303577269$.