BTS SIO – 1
ALGORITHMIQUE APPLIQUÉE – Appendice

Les listes/chaînes de caractères

Voici quelques outils intéressant pour les listes.

Savoir si un élément est présent dans une liste

Pour savoir si un élément est présent dans une liste, on peut utiliser le mot-clé `in` :

```
>>> ma_liste = [1, 2, 3]
>>> 2 in ma_liste
True
>>> 12 in ma_liste
False
>>> 12 not in ma_liste
True
```

Trouver l'indice d'un élément d'une liste

On utilise la méthode .index()

```
>>> ma_liste = ['a', 'b', 'c']
>>> ma_liste.index('b')
1
```

Balayer une liste

Pour balayer les éléments d'une liste, on utilise une boucle **for** de ce type :

```
for ele in lst:
   instructions
```

Trouver l'indice d'un élément d'une chaîne de caractères

On utilise la méthode .find()

```
>>> ch = 'abcdefgh'
>>> ch.find('d')
3
```

Connaître le plus grand élément d'une liste

Pour connaître le plus grand élément d'une liste, il faut utiliser la fonction `max()` :

```
ma_liste = [10, 20, 30, 40]
element = max(ma_liste)
print(element)
# affiche 40
```

Remarques :

Cela suppose néanmoins que tous les éléments d'une liste sont comparables et donc de type similaire sinon on obtient une erreur de type **`TypeError`.**

Connaître le plus petit élément d'une liste

Pour connaître le plus petit élément d'une liste, il faut utiliser la fonction `min()` :

```
ma_liste = ["a", "b", "c"]
element = min(ma_liste)
print(element)
# affiche a
```

Ajouter un élément à une liste

```
ma_liste = [1]
ma_liste.append(2)
ma_liste.append(3)
print(ma_liste)
# affiche [1, 2, 3]
```

Ajouter une séquence à une liste

```
ma_liste = [1]
ma_liste.extend([2, 3, 4])
print(ma_liste)
# affiche [1, 2, 3, 4]
```

Insérer un nouvel élément à un index

```
ma_liste = ['a', 'c', 'd']
ma_liste.insert(1, 'b')
print(ma_liste)
# affiche ['a', 'b', 'c', 'd']
```

Supprimer un élément de la liste

```
ma_liste = ['a', 'b', 'c', 'd']
ma_liste.remove('b')
print(ma_liste)
# affiche ['a', 'c', 'd']
```

Supprimer tous les éléments d'une liste

```
ma_liste = [10, 20, 30]
ma_liste.clear()
print(ma_liste)
# affiche []
```

Compter le nombre d'occurrences d'un élément

```
ma_liste = ["John", "Eric", "Michael", "John", "Eric", "John"]
occurrence = ma_liste.count("John")
print(occurrence)
# affiche 3
```

Trier une liste par ordre croissant

```
ma_liste = [3, 2, 6, 4]
ma_liste.sort()
print(ma_liste)
# affiche [2, 3, 4, 6]
```

Trier par ordre décroissant : on utilise le paramètre nommé reverse

```
ma_liste = [3, 2, 6, 4]
ma_liste.sort(reverse=True)
print(ma_liste)
# affiche [6, 4, 3, 2]
```

Pour inverser l'ordre des éléments dans la liste.

```
ma_liste = [4, 2, 3]
ma_liste.reverse()
print(ma_liste)
# affiche [3, 2, 4]
```

Récupération d'éléments ou de plusieurs éléments d'une liste

```
# Initialisation de la liste
List = [1, 2, 3, 4, 5, 6, 7, 8, 9]

# Montre la liste originale
print("\nOriginal List:\n", List)

print("\nSliced Lists: ")

# Donne la liste slicée de 3 à 9 avec un pas de 2
print(List[3:9:2])

# Donner la liste slicée avec un pas de 2
print(List[::2])
```

Vous pouvez laisser vide un des paramètres, je vous laisse voir ce que cela donne…

Mise en page de chaîne de caractères

Si l'on désire faire apparaître certains caractères spéciaux, il faut utiliser un caractère échappé, c'est-à-dire une séquence commençant par \ afin de la différencier des caractères normaux ou des délimiteurs :

retour à la ligne	\n
tabulation	\t
apostrophe	\'
guillemet	\"
antislash	\\

Formatage de chaîne de caractères

Pour afficher une information à l'écran, il est souvent nécessaire de prendre des informations telles que des nombres et les insérer suivant un certain format dans une chaîne de caractères.

On peut indiquer dans une chaîne de caractères l'emplacement où la valeur d'une variable doit être insérée grâce au caractère % suivi d'une lettre indiquant le type de la variable :

%d	Nombre entier en base 10
%i	Nombre entier en base 10
%o	Nombre entier en base 8
%x	Nombre entier en base 16 (lettres en minuscule)
%X	Nombre entier en base 16 (lettres en majuscule)
%f	Nombre réel
%s	Chaîne de caractères

On utilise l'opérateur % après la chaîne de caractères pour spécifier la ou les variables à utiliser. S'il y en a plusieurs, il faut obligatoirement les signaler dans le bon ordre entre parenthèses et séparées par une virgule.

```
>>> total = 122
>>> "Le montant de la facture est de %d euros avec une TVA de 20.0%%" %
total
'Le montant de la facture est de 122 euros avec une TVA de 20.0%'
>>> "%s le %s" % ("hello", "monde")
'hello le monde'
```

Depuis Python 3.6, il est possible d'utiliser le formatage de chaîne de caractères en préfixant la chaîne par `f`. Le motif défini par la chaîne peut contenir entre accolade `{}` le nom d'une variable dont la valeur sera insérée à cet emplacement.

```
>>> code_facture = "XF32"
>>> montant = 122.55
>>> f"La facture n°{code_facture} a un montant de {montant} euros"
'La facture n°XF32 a un montant de 122.55 euros'
```

TD Appendice

Exercice 1 :

Demandez à l'utilisateur de saisir une phrase sans ponctuation. Découper la phrase afin d'obtenir la liste des mots.

1. Affichez le nombre de mots.
2. Affichez le nombre d'occurrences du mot « python ».
3. Supprimez le dernier mot de la liste.
4. Ajoutez à la fin de la liste ["n'est-ce", "pas"]
5. Insérez "euh" en deuxième position de la liste
6. Puis, recréez une phrase à partir de ces mots et affichez là.

Aide :

Pour découper la phrase, utilisez la méthode **split()** d'une chaîne de caractères qui attend en paramètre le séparateur et qui retourne la liste des mots :

```
>>> "Longtemps je me suis couché de bonne heure".split(" ")
['Longtemps', 'je', 'me', 'suis', 'couché', 'de', 'bonne', 'heure']
```

On peut créer une chaîne de caractères à partir d'une liste de chaînes de caractères grâce à la méthode **join()** d'une chaîne de caractères. Cette méthode utilise la chaîne comme séparateur pour concaténer la liste passée en paramètre.

```
>>> l = ['Longtemps', 'je', 'me', 'suis', 'couché', 'de', 'bonne',
'heure']
>>> " voyez-vous ".join(l)
'Longtemps voyez-vous je voyez-vous me voyez-vous suis voyez-vous couché
voyez-vous de vo
```

Exercice 2 :

Réalisez les opérations demandées en utilisant le *slicing* (découpage de liste).

Soit la liste :

jours = ['lundi', 'mardi', 'mercredi', 'jeudi', 'vendredi', 'samedi', 'dimanche']

1. Créez une nouvelle liste que ne contient pas les jours du week-end.
2. Créez une nouvelle liste que ne contient que les jours du week-end.
3. Créez une nouvelle liste qui commence par les jours du week-end.
4. Créez une nouvelle liste qui commence par dimanche.

Exercice 3 :

Réalisez les opérations demandées en utilisant le *slicing* (découpage de liste).

Soit la liste :

```
nombres = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

1. Créez une nouvelle liste qui ne contient que les nombres impairs.
2. Créez une nouvelle liste qui ne contient que les nombres pairs.
3. Créez une nouvelle liste qui contient d'abord les nombres pairs puis les nombres impairs.

Exercice 4 :

Le code de César est un algorithme de chiffrement classique, utilisé par Jules César dans ses correspondances secrètes. Il se base sur le principe du décalage. Une lettre est remplacée par son équivalent dans l'alphabet en effectuant un décalage. Le programme ci-dessous utilise un décalage de 23. Cela donne comme équivalence :

- a → x
- b → y
- c → z
- d → a
- e → b
- f → c
- g → d
- h → e
- i → f
- j → g
- k → h
- l → i
- m → j
- n → k
- o → l
- p → m
- q → n
- r → o
- s → p
- t → q
- u → r
- v → s
- w → t
- x → u
- y → v
- z → w

Ainsi la phrase :
In cryptography, a Caesar cipher is one of the simplest and most widely known encryption techniques.
sera encodée avec un décalage de 23 comme ceci :
Fk zovmqldoxmev, x Zxbpxo zfmebo fp lkb lc qeb pfjmibpq xka jlpq tfabiv hkltk bkzovmqflk qbzekfnrbp.

Écrivez un programme qui demande à l'utilisateur d'entrer une phrase. Puis le programme doit afficher la phrase chiffrée avec un décalage de 23 lettres.

Aide :
Vous pourrez créer une chaîne de caractères avec l'alphabet, ou utiliser les fonctions `ord()` *et* `chr()`*.*

La première retourne le code UNICODE d'un caractère et la seconde retourne un caractère à partir de son code UNICODE.

Exercice 5 :

Avec les variables suivantes :
```
produit = "Aspi Deluxe"
code = "AS0003"
prix_ht = 253.99
garantie = 2
tva = prix_ht * .20
```
Formater les chaînes de caractères pour produire les résultats suivants :
```
'Produit du mois : Aspi Deluxe'
'AS0003 - Aspi Deluxe garanti 2ans'
'AS0003 - Aspi Deluxe'
```

```
'AS0003 - Aspi Deluxe : 253.99€ HT'
'AS0003 - Aspi Deluxe : 304.79€ TTC'
'AS0003 - Aspi Deluxe : 0000304.79€ TTC'
```
Utilisez le formatage *old-school* et le nouveau formatage de chaînes.

Exercice 5 : Chiffre de César, le Retour !

On considère un texte composé uniquement de caractère minuscules et majuscules et d'espace comme « bonjour tout le monde » par exemple.
On peut crypter ce message en remplaçant chaque lettre par la lettre obtenue par un décalage circulaire de +3 dans l'alphabet.
Par exemple 'a' devient 'd', 'b' devient 'e', …, 'y' devient 'b' et 'z' devient 'c'. Ainsi le message « bonjour tout le monde » devient « erqmrxu wrxw oh prqgh ». On parle de codage par la méthode de César avec un décalage de 3.

Le but de cet exercice est de décrypter le message 'eph hxbeat st strwxuugtg at rwxuugt st rthpg', codé par la méthode de César, sans connaitre le décalage appliqué.

1. Écrire une fonction **codage(message,d)** qui à un message (sous forme de chaîne de caractères) et à un entier d compris entre 0 et 25, renvoie le message codé par la méthode de César, avec le dacalage de d.
2. Écrire une fonction **freq_lettres(messagecode)** qui à un message **messagecode** (sous forme de chaîne de caractères) retourne la liste des fréquences des lettres de l'alphabet de ce message.
Par exemple, freq_lettres(*texte*)[0] devra être la fréquence du caractère 'a' dans le message *texte* et freq_lettres(*texte*)[3] celle du 'd'.
3. Écrire une fonction **calcul_auto_decal(messagecode)** qui à un message **messagecode** (un message codé), retourne le décalage probable, basé sur le fait que, statistiquement, dans un texte suffisamment long, le caractère le plus présent est 'e'.
4. Écrire un algorithme permettant de décrypter le message codé **'eph hxbeat st strwxuugtg at rwxuugt st rthpg'**.