Python objet – TP 3

1. Manipuler du texte

Attention : les chaînes de caractères sont un type non modifiable. Si vous voulez modifier une chaine, il faut la stocker dans une nouvelle variable. Voici quelques fonctions très utiles (vous pouvez en trouver d'autres – et de la doc) sur https://docs.python.org/fr/2.7/library/stdtypes.html#string-methods .

len() : Retourne la longueur de la chaîne de caractères.

lower() - upper() – capitalize() : Passe la chaîne de caractères en minuscules / majuscules / capitalisé (la première lettre en majuscule, les autres en minuscule)

split() : Transforme la chaîne de caractères en liste, chaque item de la liste étant un mot. On peut spécifier le séparateur si besoin : split('-') pour séparer entre des tirets.

find(substring) : donne la première position où on trouve la sous-chaine. Note : si on veut juste savoir si la sous-chaine existe, plutôt utiliser l'opérateur in.

replace(old,new) : Remplace la sous-chaîne old par la sous-chaîne new.

count(substring) : Compte le nombre d'occurrences de la sous-chaîne dans la chaîne.

isnumeric() – isalpha() – isalnum() : Teste si une chaîne de caractères est numérique (chiffres), alphabétique (majuscules et/ou minuscules), ou un mélange des deux. Retourne un booléen.

join() : Transformer une liste de chaines de caractères en une seule chaîne :

```
>>> seq = ["A", "T", "G", "A", "T"]
>>> seq
['A', 'T', 'G', 'A', 'T']
>>> "-".join(seq)
'A-T-G-A-T'
>>> " ".join(seq)
'A T G A T'
>>> "".join(seq)
'ATGAT'
```

2. Expressions régulières

Pour travailler avec les expressions régulières, il faut importer le module **re** (regular expressions). Les expressions régulières permettent de faire de la reconnaissance de motifs dans des chaînes de caractères, avec une syntaxe quasi-universelle (vous pourrez les retrouver dans d'autres langages presque sans modification).

En Python : on prend l'habitude de préfixer la chaine de caractères par « r » pour avoir une chaîne « brute » (sans que les \ soient interprétés comme des commandes).

Référez-vous au tableau ci-dessous pour trouver la syntaxe générale des expressions régulières. Voici les fonctions les plus utiles pour traiter des expressions régulières :

search(regex, chaine) : recherche si on trouve le motif quelque part dans la chaine.

```
>>> re.search("c", "abcdef")    # Match
<_sre.SRE_Match object; span=(2, 3), match='c'>
```

match(regex, chaine) : renvoie None si on ne trouve pas le motif au début de la chaine. Sinon, retourne un « match » avec la longueur du motif trouvé. (voir aussi fullmatch() pour tester si la chaine est égale au motif).

```
>>> re.match('[ul]r', 'truc') ## ne renvoie rien !
>>> re.match('[tu]r', 'truc')
<_sre.SRE_Match object; span=(0, 1), match='tr'>
```

findall(regex, chaine) : trouve toutes les occurrences de la regex dans la chaine, et les retourne dans une liste.

```
>>> re.findall('gr*e', 'girafe, tigre, panthere, singe')
['gre', 'ge']
```

compile() : permet de « sauvegarder » une expression régulière qui peut être réutilisée par la suite pour un match, un search ou un findall.

```
>>> regex = re.compile('gr*e')
>>> re.findall(regex, 'girafe, tigre, panthere, singe')
['gre', 'ge']
```

^	début de chaîne de caractères ou de ligne
	Exemple : l'expression ^ATG correspond à la chaîne de caractères ATGCGT, mais pas à la chaîne CCATGTT.
$	fin de chaîne de caractères ou de ligne
	Exemple : l'expression ATG$ correspond à la chaîne de caractères TGCATG, mais pas avec la chaîne CCATGTT.
.	n'importe quel caractère (mais un caractère quand même)
	Exemple : l'expression A.G correspond à ATG, AtG, A4G, mais aussi à A-G ou à A G.
[ABC]	le caractère A ou B ou C (un seul caractère)
	Exemple : l'expression T[ABC]G correspond à TAG, TBG ou TCG, mais pas à TG.
[A-Z]	n'importe quelle lettre majuscule
	Exemple : l'expression C[A-Z]T correspond à CAT, CBT, CCT...
[a-z]	n'importe quelle lettre minuscule
[0-9]	n'importe quel chiffre
[A-Za-z0-9]	n'importe quel caractère alphanumérique
[^AB]	n'importe quel caractère sauf A et B
	Exemple : l'expression CG[^AB]T correspond à CG9T, CGCT... mais pas à CGAT ni à CGBT.
\	caractère d'échappement (pour protéger certains caractères)
	Exemple : l'expression \+ désigne le caractère + sans autre signification particulière. L'expression A\.G corres
*	0 à n fois le caractère précédent ou l'expression entre parenthèses précédente
	Exemple : l'expression A(CG)*T correspond à AT, ACGT, ACGCGT...
+	1 à n fois le caractère précédent ou l'expression entre parenthèses précédente
	Exemple : l'expression A(CG)+T correspond à ACGT, ACGCGT... mais pas à AT.
?	0 à 1 fois le caractère précédent ou l'expression entre parenthèses précédente
	Exemple : l'expression A(CG)?T correspond à At ou ACGT.
{n}	n fois le caractère précédent ou l'expression entre parenthèses précédente
{n,m}	n à m fois le caractère précédent ou l'expression entre parenthèses précédente
{n,}	au moins n fois le caractère précédent ou l'expression entre parenthèses précédente
{,m}	au plus m fois le caractère précédent ou l'expression entre parenthèses précédente
(CG|TT)	chaînes de caractères CG ou TT
	Exemple : l'expression A(CG|TT)C correspond à ACGC ou ATTC.

3. Manipuler des fichiers avec la bibliothèque standard

Pour manipuler des fichiers, il faut commencer par les ouvrir !
open(nom_fichier, [mode]) : ouvrir le fichier avec le nom_fichier ; on peut spécifier le mode d'ouverture : 'r' (read) pour de la lecture, 'w' (write) pour de l'écriture (qui efface le contenu existant) et 'a' (append) pour de l'écriture qui se rajoute au contenu existant.

On oublie pas de refermer le fichier avec close() à la fin.

Lecture/écriture

f.read([nbcar]) : lire le contenu d'un fichier f, en entier. L'option nbcar permet de spécifier le nombre de caractères que l'on veut lire.

```
>>> fichier = open('monfichier.txt' ,'r')
>>> s = fichier.read()
>>> print(s)
>>> fichier.close()
```

f.readline() : lire une ligne par une ligne dans le fichier f ; retourne le contenu d'une seule ligne
f.readlines() : lire toutes les lignes du fichier f ; retourne une liste des lignes du fichier

```
>>> f = open('fichier2.txt','r')
>>> t = f.readline()
>>> print(t)
Ceci est la ligne un
>>> print(f.readline())
Voici la ligne deux
>>> t = f.readlines()
>>> print(t)
['Voici la ligne trois\n', 'Voici la ligne quatre\n']
>>> f.close()
```

f.write(chaine) : écrire la chaine en paramètre dans le fichier f

Le module os

Le module os permet de faire de la manipulation de fichiers un peu plus poussée (déplacement, suppression, changement de répertoire…). Il est recommandé d'importer le module de la manière suivante : *import os*, pour éviter d'écraser les noms d'autres fonctions utiles (notamment la fonction *open()*). Utilisez bien ces fonctions depuis le module os !

os.getcwd() : renvoie l'adresse du répertoire de travail courant

os.chdir(adresse) : fixe le répertoire de travail à l'adresse donnée (relative ou absolue)

os.listdir() : renvoie une liste des noms des dossiers situés à l'adresse courante.

os.rename(adresse1, adresse2) : renomme le fichier (ça peut permettre de le déplacer d'un dossier à un autre !

os.remove(nom_fichier) : supprime le fichier passé en paramètre

4. Gestion des exceptions et des erreurs

Il y a de nombreuses manières de gérer les exceptions en Python. N'hésitez pas à aller voir https://docs.python.org/fr/3.7/tutorial/errors.html pour en apprendre plus (notamment sur les différents types d'exceptions).

Les exceptions servent surtout à déclencher une erreur si un comportement négatif arrive. On peut prévoir de tels comportements (par exemple, l'ouverture d'un fichier est un comportement qui peut générer de nombreux problèmes !). Quand on sait qu'une commande risque de déclencher une erreur, on l'encapsule dans un opérateur **try**, et on met dans un opérateur **except** la commande à exécuter si une erreur survient :

```
filename = input("Veuillez entrer un nom de fichier : ")
try:
      f = open(filename, "r")
except:
      print("Le fichier", filename, "est introuvable")
```

Si on a souvent besoin de tester cette exception, on peut en faire une fonction :

```
def existe(fname):
      try:
            f = open(fname,1r1)
            f.close()
            return 1
      except:
            return 0

filename = input("Veuillez entrer le nom du fichier : ")
if existe(filename):
      print("Ce fichier existe bel et bien.")
else:
      print("Le fichier", filename, "est introuvable.")
```

On peut spécifier dans le except le type d'exception (*ValueError, TypeError, NameError*…).

On peut aussi utiliser les exceptions dans des boucles, en faisant un bon usage de l'opérateur **break** :

```
while True:
     try:
          x = int(input("Please enter a number: "))
          break
     except ValueError:
          print("Oops!  That was no valid number.  Try again...")
```