PROGRAMMATION PYTHON

Chapitre 8: Les chaînes de caractères

Sommaire

Définition

Les chaines de caractères vont nous permettre de stocker et traiter des valeurs alphanumériques (chiffres et lettres) comme « Chat, Chien, Toto1,… ». C'est un type d'objet incontournable en programmation.

Contrairement aux autres types vus jusqu'à présent, les chaines de caractères sont des types spéciaux appelées des objets.

Un objet est une structure de données, comme les variables qui peut contenir elle-même d'autres variables et fonctions.

Les chaines de caractères sont considérées comme des objets car elles contiennent des fonctions qui permettent de faire des opérations comme mettre en minuscule, majuscule…

Python traite les chaines de caractères comme de type **str** pour **String**.

```
msg = "dfd"
type(msg)
<class 'str'>
```

Définition

Certains langages disposent du type chaines de caractères et type caractère. Dans ces langages une variable qui contient un caractère est considérée comme un type caractère et pour plus de 1 caractère une chaine de caractères.

Python ne fait pas cette distinction, tout contenu d'une variable entourée par des guillemets simples ou double est considérée comme chaine de caractères.

```
msg = "toto"
type(msg)
<class 'str'>
msg1="t"
type(msg1)
<class 'str'>
msg = ""
type(msg)
<class 'str'>
msg1=''
type(msg1)
<class 'str'>
```

Un objet comme str est issu d'une classe. La classe est une forme de type de données qui permet de définir des fonctions et variables propres au type. Dans le cas des chaines de caractères on a les fonctions comme mettre en minuscule, majuscule… fournies par les développeurs de Python.

Les méthodes de bases de la classe str

La classe **str** de Python contient une multitude de fonctions disponible ici Opérations usuelles sur des chaînes ou Méthodes de chaînes de caractères.

Les plus utilisées sont décrites ci-dessous:

- Pour mettre une chaine de caractères en majuscule on utilise la fonction **upper** avec le format suivant: $chaine.upper()$. Elle retourne le résultat de la chaine de caractère en majuscule.

```
msg = "hello"
msg.upper()
'HELLO'
msg
'hello'
```

- Pour mettre une chaine de caractères en minuscule on utilise la fonction **lower** avec le format suivant: $chaine.lower()$. Elle retourne le résultat de la chaine de caractères en minuscule.

```
msg = "HELLO"
msg.lower()
'hello'
msg
'HELLO'
```

Les méthodes de bases de la classe str

- Pour mettre la première lettre en majuscule on utilise la fonction **capitalize**.

```
msg = "bonjour tout le monde!"
msg.capitalize()
'Bonjour tout le monde!'
```

- Pour mettre chaque mot de la chaine en majuscule on utilise la fonction **title.**

```
msg = "bonjour tout le monde!"
msg.title()
'Bonjour Tout Le Monde!'
```

- Pour supprimer les espaces au début et à la fin de la chaine, on utilise la fonction **strip.**

```
msg = "    bonjour tout le monde!    "
msg
'    bonjour tout le monde!    '
msg.strip()
'bonjour tout le monde!'
```

- Pour avoir le nombre de caractères d'une chaine on utilise la fonction **len.**

```
msg = "hello"
len(msg)
5
msg = ""
len(msg)
0
```

Formater et afficher une chaîne

Pour afficher le contenu d'une variable ainsi qu'une chaine de caractères on utilise la fonction **print.**

Cependant pour afficher une chaine avec le contenu d'une ou plusieurs variables, on utilisera les méthodes de formatage de chaînes de caractères.

Python propose une méthode **format** pour formatter une chaine.

```
nom = "John"
prenom = "Doe"
age = 14
print("Je suis {0} {1} j'ai {2} ans.".format(nom, prenom, age))
Je suis John Doe j'ai 14 ans.
```

De gauche à droite on a:

- Une chaîne de caractères qui ne présente rien de particulier que des nombres qui indiquent l'ordre d'insertion des valeurs des variables 0, puis 1..
- La méthode **format** qui va passer les paramètres des variables dont les valeurs seront insérées dans la chaîne.
- Quand Python exécute cette méthode, il remplace la chaine {0} par la première variable passée à la méthode $format$ ensuite la deuxième variable ainsi de suite.

Formater et afficher une chaîne

La méthode **format** peut être utilisée pour créer une nouvelle chaîne de caractères.

```
nom = "John"
prenom = "Doe"
age = 14
personne = "{0} {1} {2}".format(nom, prenom, age)
personne
'John Doe 14'
type(personne)
<class 'str'>
print(personne)
John Doe 14
```

L'ordre entre les accolades {} est importante le premier paramètre de *format* correspond à l'élément {0} dans la chaine.

```
personne = "{1} {0} {2}".format(nom, prenom, age)
personne
'Doe John 14'
```

Si on ne spécifie pas des valeurs entre les accolades dans ce cas c'est l'ordre de format qui sera pris en compte.

```
personne = "{} {} {}".format(nom, prenom, age)
print(personne)
John Doe 14
```

Formater et afficher une chaîne

On peut également nommer les variables que l'on va afficher ce qui est plus intuitif que leur indice.

```
print("Je suis {nom} {prenom} j'ai {age} ans.".format(nom="John", prenom="Doe", age=14))
Je suis John Doe j'ai 14 ans.
```

Python propose aussi une autre méthode la **f-string** qui permet de faire les mêmes opérations que **format.** De plus avec la f-string on peut afficher le nom de la variable.

```
# Formater avec f-string
nom = "John"
prenom = "Doe"
age = 14
print(f"{nom} {prenom} {age}")
John Doe 14
# Formater en affichant le nom des variables
print(f"{nom=} {prenom=} {age=}")
nom='John' prenom='Doe' age=14
# Création de variable
nomComplet = f"{nom} {prenom}"
print(nomComplet)
John Doe
```

Contrairement à **format** avec la **f-string** on n'a pas besoin de spécifier ici l'ordre d'insertion des valeurs des variables lors du formatage de la chaine de caractères.

Concaténation de chaînes

La concaténation de chaînes est une opération qui consiste à regrouper deux chaines en une en mettant la seconde à la suite de la première. Pour faire de la concaténation on utilisera le signe plus +.

```
# Concaténation simple
nom = "John"
prenom = "Doe"
nom + prenom
'JohnDoe'
# Concaténation en mettant un espace
nom + " " + prenom
'John Doe'
# Concaténation de plusieurs chaînes
"Persone " + nom + " " + prenom
'Persone John Doe'
```

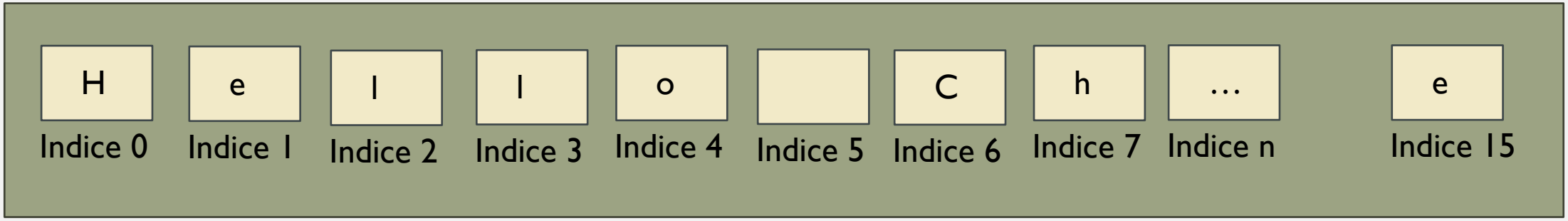
La concaténation ne marche que sur les types chaine de caractères si on essaie de faire une concaténation entre un type chaine de caractères et un nombre, Python lève une exception car ambiguë Python ne sait pas s'il doit faire une opération de nombre ou une concaténation de chaines. On utilisera la fonction **str** pour convertir un nombre en chaine si on veut éviter une erreur.

```
# Essaie concaténation entre chaine et nombre
nom = "John"
age = 14
nom + age
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    nom + age
TypeError: can only concatenate str (not "int") to str
# Concaténation avec convertion
nom = "John"
age = 14
nom + str(age)
'John14'
```

Parcours de chaîne par indice

Une chaîne de caractères par définition est une séquence de caractères. Chaque caractère dans une chaîne est identifié par une position appelée indice dont le premier est 0.

Par exemple la représentation de la chaine de caractère « Hello Christophe » donne:



L'espace est considéré comme un caractère (indice 5). Pour afficher le caractère situé à l'indice n en python on utilise la syntaxe suivante $variable\_chaine[indice]$.

```
>>> # Définition d'une chaine
>>> message = "Hello Christophe"
>>> # Affichage des caractères à des indices spécifiques
>>> message[0]
'H'
>>> message[5]
' '
>>> message[6]
'C'
>>> message[7]
'h'
```

Parcours de chaîne par indice

Python permet aussi d'avoir le caractère situé à la fin de la chaine en spécifiant un indice négatif. Par exemple:

- Le caractère situé à l'indice 1 correspond au deuxième caractère de la chaine.
- Le caractère situé à l'indice -1 correspond au dernier caractère de la chaine.
- Le caractère situé à l'indice -2 correspond à l'avant dernier caractère la chaine.

```
>>> # Initialisation de la chaine
>>> msg = "Hello"
>>> msg[0]
'H'
>>> msg[1]
'e'
>>> msg[-1]
'o'
>>> msg[-2]
'l'
```

Python lève l'exception **IndexError** quand on spécifie un indice qui n'existe pas dans la chaine:

```
>>> # Initialisation de la chaine
>>> msg = "Hello"
>>> msg[100]
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    msg[100]
IndexError: string index out of range
```

Parcours de chaîne par indice

On peut utiliser les boucles *for* ou *while* pour parcourir tous les caractères d'une chaine à partir des indices.

Par exemple pour afficher la liste des caractères dans la chaine « Christophe » on aura:

```
>>> # Initialisation de la chaîne
>>> nom = "Christophe"
>>> # Initialisation de la variable indice
>>> indice = 0
>>> print("Les caractères de la chaine '" + nom + "' sont:")
Les caractères de la chaine 'Christophe' sont:
>>> while indice < len(nom):
...     print(nom[indice])
...     indice += 1
...
...

C
h
r
i
s
t
o
p
h
e
```


```
>>> # Initialisation de la chaîne
>>> nom = "Christophe"
>>> # Initialisation de la variable total de caractères
>>> total = len(nom)
>>> for indice in range(total):
...     print(nom[indice])
...
...

C
h
r
i
s
t
o
p
h
e
```

Le dernier caractère d'une chaine a pour indice le nombre total des caractères moins 1.
Par exemple si on a une variable chaine *msg* qui est composé de 4 caractères le dernier caractère a pour indice 3 car on commence à compter à partir de 0.

Parcours de chaîne avec for

On l'a vu précédemment qu'on peut parcourir une chaine à partir de l'indice avec les boucles $for\ et\ while$.

L'une des méthodes les plus couramment utilisée est la boucle for pour traiter ou lire les caractères d'une chaine sans spécifier les indices.

```
>>> # Définition de la chaine
>>> print("Les caractères de la chaîne sont:")
Les caractères de la chaîne sont:
>>> for caractere in nom:
...     print(caractere)
...
...

C
h
r
i
s
t
o
p
h
e
```

Si on veut les indices

```
>>> # Code avec enumerate pour avoir les indices des caractères
>>> nom = "Christophe"
>>> print("Les caractères avec indices de la chaîne sont:")
Les caractères avec indices de la chaîne sont:
>>> for indice, caractere in enumerate(nom):
...     print("Indice", indice, ":", caractere)
...
...

Indice 0 : C
Indice 1 : h
Indice 2 : r
Indice 3 : i
Indice 4 : s
Indice 5 : t
Indice 6 : o
Indice 7 : p
Indice 8 : h
Indice 9 : e
```

Sélection de chaînes

Python permet aussi d'extraire une partie d'une chaine en utilisant la syntaxe suivante:

- $\boldsymbol{chaine}[\boldsymbol{indicedepart}:\boldsymbol{indicefin}]$: renvoie la chaîne composée des caractères situés de l'indice de départ jusqu'à l'indice de fin exclu.
- $\boldsymbol{chaine}[\boldsymbol{indicedepart}:]$: renvoie la chaine composée des caractères de l'indice de départ jusqu'à l'indice de fin soit le dernier caractère.
- $\boldsymbol{chaine}[:\boldsymbol{indicedefin}]$: renvoie la chaine composée des caractères de l'indice de départ jusqu'à l'indice de fin spécifié exclu.

```
>>> # Définition de la chaine
>>> nom = "Christophe"
>>> # La chaine "toph" peut être extraite avec
>>> nom[5:9]
'toph'
>>> # La chaine "ristophe" peut être extraite avec
>>> nom[2:]
'ristophe'
>>> # La chaine "Chris" peut être extraite avec
>>> nom[:5]
'Chris'
```

Dans l'extraction d'une chaine quand on spécifie un indice de fin celui-ci est toujours exclu soit l'intervalle [début, fin[.

La fonction *id*

Comme vu dans le **Chapitre 2,** les variables sont de façon imagée des cases dans lesquelles Python va stocker des valeurs identifiées par une étiquette appelée nom de variable.

Python dispose d'une fonction interne **id** qui permet d'avoir l'identifiant de l'adresse mémoire ou la variable a été stockée. Cette valeur change à chaque exécution du programme.

Si la mémoire de l'ordinateur était représentée comme une étagère avec plusieurs cases, l'id permettra d'avoir le numéro de la case où est stockée la valeur de la variable.

```
>>> # Déclaration d'une variable entière
>>> a = 1
>>> id(a)
140722933130024
>>> # Déclaration d'une variable entière b avec la même valeur
>>> b = 1
>>> id(b)
140722933130024
>>>
>>> # Déclaration d'une variable chaine
>>> nom1 = "Christophe"
>>> id(nom1)
2328340450032
>>> # Déclaration d'une autre variable chaine avec la même valeur
>>> nom2 = "Christophe"
>>> id(nom2)
2328340450032
```

La fonction *id*

La fonction **id** ne renvoie pas tout le temps le même numéro de case. Par exemple quand on utilise la fonction **input,** la saisie de l'utilisateur est stockée dans une nouvelle case.

```
>>> # Création d'une variable avec une valeur
>>> a = 1
>>> id(a)
140722933130024
>>> # Demander une valeur à l'utilisateur qui va être 1
>>> b = input()
1
>>> id(b)
140722933173376
>>> # Si on convertit en entier on aura le même id
>>> b = int(b)
>>> b
1
>>> id(b)
140722933130024
```

Précédemment la fonction **id** renvoyait la même valeur parce les variables contiennent les mêmes valeurs or dans le cas d'input on ne sait pas ce que va fournir l'utilisateur et si la valeur qu'il va saisir existe ou non. Après saisie de l'utilisateur si on convertit et qu'on a la même valeur qu'une déjà stockée, l'identifiant devient celui de la valeur actuellement stockée.

Ce changement d'identifiant ne fonctionne pas sur les objets comme les chaines, les listes… car ces types sont des objets pas des types simples comme les entiers.

La fonction *id*

Les chaines de caractères sont des objets immutables ce qui veut dire que lorsqu'elles sont créées leur *id* ne mute pas à un nouveau id comme c'est le cas pour les entiers.

```
>>> # Définition d'une variable chaine
>>> nom1 = "John"
>>> id(nom1)
1760796801968
>>> # Demander à l'utilisateur de saisir une chaine qui sera la même valeur John
>>> nom2 = input()
John
>>> nom2
'John'
>>> id(nom2)
1760790153904
>>> nom2 = str(nom2)
>>> id(nom2)
1760790153904
```

Les deux variables malgré qu'elles contiennent des valeurs identiques sont stockées dans des cases mémoires différentes. En un mot les valeurs sont dupliquées.

La méthode d'allocation des cases mémoires n'est pas la même pour tous les types de données la méthode est différente pour les types primitifs et les types complexes comme les strings…

Subtilité des comparaisons avec *is not* et *is*

Dans le chapitre sur les structures conditionnelles on a vu que les mots clés **is not** et **is** permettaient de vérifier si une valeur d'une variable est égale ou différente à la valeur d'une autre variable.

Cependant cette méthode n'est pas du tout la même chose quand on parle des types complexes comme les strings.

En réalité les mots clés **is** et **is not** font une comparaison des cases mémoire pas des valeurs des variables. Ces méthodes vérifient que deux variables sont stockées dans la même case mémoire. Pour comparer les valeurs on utilisera == et != quand on parle des chaines.

Pour résumer:

- "***is***" ***et*** "***is not***" donne le même résultat que $!= et ==$ pour les variables de types simples car ces valeurs sont mutables les identifiants de case mémoire change quand les valeurs sont identiques plusieurs variables avec la même valeur auront la même case mémoire.
- "***is***" ***et*** "***is not***" est différent de $!= et ==$ pour les variables de types complexes comme les chaines. Deux chaines de mêmes valeurs n'ont pas forcément les mêmes identifiants de case mémoire. Pour ces types on utilisera == pour vérifier l'égalité des valeurs des variables de type chaine et != pour vérifier la différence entre les valeurs des variables de type chaine.

Subtilité des comparaisons avec *is not* et *is*

```
>>> # Cas des types simple entiers
>>> a = 1
>>> id(a)
140722933130024
>>> id(b)
140722933130024
>>> a is b
True
>>> # Demandons un entiers à l'utilisateur c
>>> c = input()
1
>>> c
'1'
>>> # Après saisie la valeur est stockée dans une case différente
>>> id(c)
140722933173376
>>> c is a
False
>>> # L'id devient identique à celui de a et b si on convertit en entier
>>> c = int(c)
>>> c
1
>>> id(c)
140722933130024
>>> c is a
True
```

```
>>> # Cas des types complexes comme les strings
>>> nom1 = "John"
>>> id(nom1)
2328340470704
>>> nom2 = "John"
>>> id(nom2)
2328340470704
>>> nom1 is nom2
True
>>> # Demander un nom à l'utilisateur qui sera la même valeur
>>> nom3 = input()
John
>>> nom3
'John'
>>> id(nom3)
2328340447920
>>> nom3 is nom1
False
>>> nom3 is nom2
False
>>> nom3 == nom1
True
>>> nom3 == nom2
True
```

Même si == et != permettent de vérifier l'égalité ou la différence des valeurs de deux variables, une condition à respecter est que les variables soit du même type. Ainsi 1 == "1" sera Faux car même s'ils ont la même valeur les types sont différent.

Exercices

Exercice 1

Soit un utilisateur dont le nom d'utilisateur est « Christophe » et son mot de passe « chris1234 ».
Ecrire un programme qui demande à l'utilisateur son nom d'utilisateur et son mot de passe, le programme doit afficher « Nom d'utilisateur et/ou mot de passe incorrect(s) ». Si l'une des deux valeurs est incorrecte sinon afficher « Bienvenue Christophe ! ».
Une condition à respecter est d'appliquer l'égalité stricte sur le mot de passe pas sur le nom d'utilisateur c'est-à-dire si l'utilisateur écrit « christophe », « cHristophe »… ou le nom d'utilisateur avec des espaces en début ou fin tant que le mot de passe est correct il doit être accepté. Ne pas appliquer la différence entre majuscule et minuscule sur le nom d'utilisateur.

Exercice 2

Ecrire un programme qui demande un mot à l'utilisateur, ensuite le programme doit afficher le pluriel du mot. Par exemple ajouter « s » si le mot est du type « chat », pour les mots se terminant par « s » comme « souris » ne pas mettre au pluriel afficher le mot tel quel, si le mot se termine par « al » le pluriel deviendra « aux » comme « Cheval » qui donnera « Chevaux ». On ne tiendra pas compte des exceptions des pluriels de la langue française. Mettre le code qui va renvoyer le pluriel dans une fonction.

Exercices

Exercice 3

La distance de Hamming entre deux mots de même longueur est le nombre d'endroits où les lettres sont différentes.
Par exemple:

$$\begin{matrix} JAPON \\ SAVON \end{matrix} => a\ pour\ distance\ de\ Hamming\ 2$$

Car la première lettre de Japon est différente de la première lettre de Savon, les troisièmes lettres aussi sont différentes.
Ecrire un programme qui demande à l'utilisateur deux mots de mêmes longueurs et ensuite affiche la distance de Hamming entre les deux mots.
Attention on ne doit pas tenir compte des majuscules et des minuscules Japon, SAVON doit donner comme distance de Hamming 2.
Afficher un message d'erreur si l'utilisateur fournir une chaine vide ou des mots de longueurs différentes.
Le calcul de la valeur de Hamming doit être effectué dans une fonction qui prend en paramètre deux mots et renvoie un entier qui correspond à la distance de Hamming.

Exercices

Exercice 4

Ecrire une fonction *verlan()*, qui permet de renvoyer un mot à l'envers. Par exemple $verlan("mot")$ doit renvoyer « tom ».
Dans la fonction main écrire un programme qui va demander à l'utilisateur un mot et afficher le verlan du mot.

Exercice 5

Ecrire une fonction qui prend en paramètre un mot et renvoie True si le mot est un palindrome sinon False. Un palindrome est un mot qui s'écrit indifféremment de gauche à droite ou de droite à gauche. Peu importe le sens de lecture ou d'écriture on obtient le même mot. Par exemple « SOS » ou « RADAR » est un palindrome cependant « TOTO » n'est pas un palindrome car l'inverse donne « OTOT ».
Ecrire un programme qui va demander un mot à l'utilisateur. Ensuite le programme doit appeler la fonction de palindrome pour vérifier si le mot est un palindrome ou non et afficher le résultat par exemple si l'utilisateur entre « TOTO » on doit afficher « Le mot 'TOTO' n'est pas un palindrome. »
On doit gérer les cas d'erreur.

Exercices

Mini projet 1

Créer un programme qui lorsqu'il démarre affiche le menu suivant: pluriel de mot, verlan de mot, vérification palindrome, calcul de la distance de Hamming, nombre de mots d'une phrase, nombre de caractères dans une phrase (différent de l'espace), nombre de consonnes d'une phrase, nombre de voyelle d'une phrase.
L'utilisateur devra faire le choix d'une option.
En fonction du choix demander les paramètres à l'utilisateur par exemple si le choix est palindrome on demandera un mot à l'utilisateur, si le choix est le calcul de la distance de Hamming on demandera deux mots à l'utilisateur.
Après saisie des paramètres si aucune erreur, afficher le résultat de l'opération.
Certaines fonctions ont déjà été créées dans les exercices précédents importer les modules déjà créés et si nécessaire créer de nouveaux modules pour les autres opérations.

Exercices

Mini projet 2

Une molécule d'ADN est formée d'environ six milliards de nucléotides. Dans un brin d'ADN il y'a seulement quatre types de nucléotides qui sont notés A, C, T ou G. Une séquence de d'ADN est donc un long mot de la forme: TAAATTACGA….

Ecrire une fonction *presence_de_sequence()* qui teste si une séquence de base contient une séquence de nucléotides donnée composée uniquement de A, C, T ou G. La fonction doit prendre en paramètres la séquence de base et la séquence à chercher.

Un crime a été commis dans un château. On a récupéré deux brins d'ADN, provenant de deux positions éloignées de l'ADN du coupable. Il y'a quatre suspects, dont on a la séquence d'ADN. L'objectif est d'écrire un programme pour trouver le suspect.

Premier code du coupable: CATA
Deuxième code du couple: ATGC

Exercices

Mini projet 2 suite

La liste des coupables:

ADN du Mr Léo
CCTGGAGGGTGGCCCCACCGGCCGAGACAGCGAGCATATGCAGGAAGCGGCAGGAATAAGGA
AAAGCAGC

ADN du Mlle Rose
CTCCTGATGCTCCTCGCTTGGTGGTTTGAGTGGACCTCCCAGGCCAGTGCCGGGCCCCTCATA
GGAGAGG

ADN du Mr Bob
AAGCTCGGGAGGTGGCCAGGCGGCAGGAAGGCGCACCCCCCCAGTACTCCGCGCGCCGGGA
CAGAATGCC

ADN du Mme Prouesse
CTGCAGGAACTTCTTCTGGAAGTACTTCTCCTCCTGCAAATAAAACCTCACCCATGAATGCTCA
CGCAAG

FIN CHAPITRE 8