Les chaînes de caractères

Capacités attendues

- ✓ Identifier l'intérêt des différents systèmes d'encodage.
- ✓ Convertir un fichier texte dans différents formats d'encodage.

1 Représentation des caractères

Les caractères sont les représentations informatiques des lettres (minuscules, majuscules, accentuées ou non, etc.), des chiffres et de tous les symboles utilisés pour communiquer par écrit (espace, ponctuation, etc.).

Depuis les années 1960, la norme (American Standard Code for Information Interchange) définit 95 caractères imprimables codés sur 7 bits, à savoir le caractère **« espace »** et :

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~
```

La norme ISO 8859-1 (aussi appelée Latin-1), conçue en 1986, a étendu la norme ASCII à 191 caractères, en utilisant un octet. Cette norme était celle utilisée dans les pays occidentaux par le système d'exploitation Windows, et tend aujourd'hui à disparaître au profit du standard né en 1991, qui permet aujourd'hui de représenter plus de 130 000 caractères dans une centaine d'écritures.

Concrètement, la norme Unicode est une table associant à chaque caractère un nombre entier naturel, appelé souvent écrit en hexadécimal. Par exemple :

- le point de code correspondant à la lettre `'A'` est U+0041 soit, en décimal :
- le point de code correspondant à la lettre `'a'` est U+0061 soit, en décimal :
- le point de code correspondant au signe € est U+20AC soit 8364 en décimal.

La représentation en machine d'un caractère Unicode s'effectue en général selon le format Ce dernier présente en effet l'avantage d'être compatible avec la norme ASCII, au sens où les 128 premiers caractères sont représentés par les mêmes bits qu'en ASCII, selon le schéma $0\square\square\square\ \square\square\square\square$.

La lettre `'a'` est par exemple codé par l'octet $0011\ 0001$.

Pour les points de code compris entre 128 et 2047, le format UTF-8 utilise 2 octets, selon le schéma

$$110\square\ \square\square\square\square\ \ 10\square\square\ \square\square\square\square$$

Par exemple, le point de code du caractère `'é'` vaut 233, soit en binaire sur 11 bits : $000\ 1110\ 1001$.

En UTF-8, le caractère `'é'` est donc représenté par les octets $110\mathbf{0\ 0011}\ \ 10\mathbf{10\ 1001}$.

Les points de code supérieurs à 2047 sont codés en UTF-8 sur 3 ou 4 octets.

1.1 Problèmes d'affichage d'un texte en français

Pourquoi des caractère étranges tels que Ã© sont parfois affichés à la place des lettres accentués ?

Cela se produit lorsqu'un fichier texte est encodé selon la norme UTF-8, mais lu en Latin-1.
Prenons en effet l'exemple de la lettre `'é'` codée par les octets $1100\ 0011\ \ 1010\ 1001$ en UTF-8 : si l'éditeur de texte (ou le navigateur Internet, etc.) traite ces octets en Latin-1, il interprète chacun d'entre eux comme la représentation d'un caractère sur 8 bits, et produit l'affichage des deux caractères Ã et ©.

Il faut donc veiller à préciser le format d'encodage pour que l'affichage soit correct.

Par précaution, il est ainsi recommandé de ne pas utiliser de caractères accentués dans les noms de variables ou fonctions d'un programme informatique, ni dans les noms de fichiers et répertoires.

---

2 Les chaînes de caractères

Définitions

Une **chaîne de caractères** est un type de variable (`str` en Python, de l'anglais *string*) composé d'un nombre quelconque de caractères. Une chaîne de caractères se note en général entre doubles guillemets droits : `"abcd"`.

La **longueur** d'une chaîne de caractères est le nombre de caractères qu'elle contient. En Python, c'est la fonction `len` (de l'anglais *length*) qui renvoie la longueur de la chaîne de caractères passée en argument.

Une chaîne peut ne contenir aucun caractère : c'est la **chaîne vide** `""`, de longueur nulle.

Les opérateurs

- La **concaténation** "+" : la concaténation `s1+s2` de deux chaînes de caractères `s1` et `s2` est la chaîne obtenue en ajoutant à la fin de `s1` tous les caractères de `s2` (renvoie une **chaîne de caractères**).
  (La concaténation d'une chaîne `s` avec elle même peut s'obtenir avec la syntaxe `s*2`.) (au lieu de `s+s`).
- La **répétition** "`*`" : elle permet de répéter autant de fois qu'on le souhaite une chaîne (renvoie une **chaîne de caractères**).
- L'**inclusion** `in` : permet de déterminer si un chaîne est dans une autre chaîne (renvoie un **booléen**).

Exemple :

`"abc" + "def"` vaut `"abcdef"`

`"abc"*3` vaut `"abcabcabc"`

`"bc"` in `"abc"` vaut `True`

Caractères spéciaux

L'antislash `"\"` est utilisé pour définir ou pour échapper certains caractères spéciaux. On donne quelques exemples ci-dessous.

- `\n` : saut de ligne
- `\"` : double guillemet droit "
- `\\` : antislash \

Exercice 1

Quel est l'affichage produit par le script ci-dessous ?

```
s = "a b"*2 + "ba" + "b"*3
print(s, len(s))
```

> **Les fonctions Python `chr` et `ord`**
>
> En Python, deux fonctions permettent de convertir un point de code en caractère, et inversement :
>
> - `chr(ptc)` renvoie le caractère dont le point de code est `ptc` : l'appel 97 renvoie `"a"` ;
> - `ord(char)` renvoie le point de code du caractère `char` : l'appel `ord("A")` renvoie 65.

Exercice 2

Que renvoie la fonction ci-dessous lorsque l'argument passé en paramètre est une lettre minuscule ?

```
def convert(c):
    return chr(ord(c)-32)
```

Exercice 3

On associe à chaque lettre minuscule son rang selon le tableau suivant.

Lettre	a	b	c	d	...	z
Rang	0	1	2	3	...	25

1. Écrire une fonction `rang(l)` qui renvoie le rang d'une lettre `l`.
2. Écrire une fonction `lettre(r)` qui renvoie la lettre de rang `r`.

Exercice 4

Quel est l'affichage produit par le script ci-dessous ?

```
s = "abn\nn\\\""
print(len(s))
```

> **Accès à un caractère d'indice donné**
>
> Dans une chaîne de caractères, chacun des caractères est repéré par son : il s'agit de sa position, en partant de 0 pour le premier (le plus à gauche), comme illustré ci-dessous pour la chaîne `"abcd"`.
>
caractères	a	b	c	d
indices	0	1	2	3
>
> Étant donnée une chaîne de caractères `ch`, le caractère d'indice `idx` est fourni par la syntaxe

Exercice 5

Quels sont les affichages produits par le script ci-dessous ?

```
s = "abcdefghij"
n = len(s)
print(s[1], s[5])
s = 2*s
print(n)
print(s[12])
```

Exercice 6

Quelle instruction permet d'affecter à la variable `x` la valeur du dernier caractère d'une chaîne de caractères `ch` ?

Exercice 7

Sachant que les chiffres ont un point de code compris entre 48 et 57, écrire une fonction `est_numerique` qui prend en argument une chaîne de caractères de longueur 1, et renvoie `True` si le caractère est un chiffre, `False` sinon.

Exercice 8

Écrire une fonction `est_une_date_valide` qui prend en argument une chaîne de caractères et renvoie `True` si elle représente une date au format jj/mm/aaaa, et `False` sinon. (Penser à ce qui a été fait dans l'exercice précédent !)

Formatage d'une chaîne de caractères : l'opérateur %

Il est souvent utile d'afficher sur une même ligne la valeur de plusieurs variables. En Python, il existe au moins trois solutions.

- Passer plusieurs valeurs en argument de la fonction `print` :

```
jour = 25
mois = 10
annee = 2022
print("La prochaine éclipse aura lieu le", jour, "/", mois, "/", annee, ".")
# La prochaine éclipse aura lieu le 25 / 10 / 2022 .
```

- Concaténer plusieurs chaînes (nécessite la conversion des variables en chaînes de caractères) :

```
print("La prochaine éclipse aura lieu le " + str(jour)
      + "/" + str(mois) + "/" + str(annee) + ".")
```

- Utiliser le marqueur %s, qui signale la présence d'un élément à insérer, et l'opérateur % :

```
print("La prochaine éclipse aura lieu le %s/%s/%s."%(jour, mois, annee))
```

- Utiliser les expressions formatées (en n'oubliant pas le `f` avant d'ouvrir les guillemets) :

```
print(f"La prochaine éclipse aura lieu le {jour}/{mois}/{annee}.")
```

Exercice 9

Écrire une fonction `affiche_coordonnees(x, y)` qui affiche `"Ce point a pour coordonnées (x; y)."` avec les valeurs de `x` et `y` passées en argument.

Par exemple, l'appel `affiche_coordonnees(4, 5)` devra afficher `"Ce point a pour coordonnées (4; 5)."`

2.1 Parcours d'une chaîne de caractères

Une chaîne de caractères est itérable : cela signifie qu'une boucle bornée (boucle `for`) permet de parcourir chacun des caractères d'une chaîne `s` selon la syntaxe ci-dessous.

```
for char in s:
    # à chaque tour de boucle, char prend successivement la
    # valeur des caractères de s
    print(char) # par exemple
```

Exercice 10

Quels sont les affichages réalisés par le script suivant ?

```
s = "abcde"
for idx in range(len(s)):
    print(idx, s[idx])
```

Exercice 11

Écrire une fonction `nb_e(s)` qui renvoie le nombre d'occurrences de la lettre `"e"` dans une chaîne de caractères `s`.

Exercice 12

Écrire une fonction `est_binaire_valide` qui prend en argument une chaîne de caractères `s` et renvoie :

- `True` si `s` ne contient que les caractères `0` et `1` ;
- `False` sinon.

Exercice 13

1. Écrire une fonction `valeur_decimale` qui prend en argument une chaîne de caractères représentant un nombre écrit en binaire, et renvoie sa valeur décimale.
2. Même question pour une chaîne de caractères représentant un nombre écrit en hexadécimal.

Exercice 14

Écrire une fonction `est_palindrome` qui prend en argument une chaîne de caractères et renvoie `True` si elle est un palindrome, `False` sinon. (Il s'agit d'utiliser ce qu'on vient de voir sur les propriétés des chaînes de caractères et de ne pas recopier le DM précédent bien sûr.)

Exercice 15

1. Concevoir une fonction `somme_chiffres(n)` qui renvoie la somme des chiffres d'un entier positif `n`.
   → *Utiliser les fonctions de conversions* `str` *et* `int`.
2. Utiliser la fonction précédente pour écrire une fonction `somme_de_tous_les_chiffres(n)` qui renvoie la somme de tous les chiffres de tous les entiers inférieurs à `n`.

   On doit avoir `somme_de_tous_les_chiffres(14)==55` et `somme_de_tous_les_chiffres(100)==900`.

2.2 Reconstruction d'une chaîne de caractères

Une chaîne peut être construite à l'aide d'une variable cumulative de la manière suivante.

```
chaine = "exemple"
s = ""
for char in chaine~:
    s = s+char
```

Exercice 16

Quelle est la chaîne de caractères affichée par le script suivant ?

```
chaine = "abc"
s = ""
for char in chaine~:
    s = s+char+s
print(s)
```

Exercice 17

Écrire une fonction `str_filter(s, mask)` qui renvoie la chaîne composée des caractères de `s` à l'exception de ceux compris dans la chaîne `mask`. Exemple : l'appel `str_filter("un exemple", " e")` doit renvoyer `"unxmpl"`.

→ L'expression `c in m` est un booléen qui vaut `True` si la chaîne `c` est une sous-chaîne de `m`, et `False` sinon.

Exercice 18

Écrire une fonction `a_l_envers(s)` qui prend en argument une chaîne de caractères `s` et renvoie la chaîne composée des caractères de `s` lus de droite à gauche. Exemple : l'appel `a_l_envers("abcd")` doit renvoyer `"dcba"`.

Exercice 19

Le chiffrement de Vigenère consiste à décaler les lettres contenues dans un message selon une clef déterminée.

Prenons l'exemple du message `"lyceelangevinwallon"`, que l'on souhaite chiffrer avec la clef `"nsi"`.

Les décalages fournis par la clef ont pour valeur 13, 18 et 8. Le chiffrement s'effectue alors de la manière suivante :

- la lettre `"l"` est décalée de 13 rangs (`"y"`) ;
- la lettre `"y"` est décalée de 18 rangs (`"q"`) ;
- la lettre `"c"` est décalée de 8 rangs (`"k"`) ;
- la lettre `"e"` est décalée de 13 rangs (`"r"`) ;
- la lettre `"e"` est décalée de 18 rangs (`"w"`) ;
- et ainsi de suite...

Le message codé est alors `"yqkrwtnfornqaoiydwa"`.

1. Concevoir une fonction `chiffre_vigenere(msg, key)` qui chiffre le message `msg` avec la clef `key`.
2. Concevoir une fonction `dechiffre_vigenere(msg, key)` qui déchiffre le message `msg`, connaissant la clef de chiffrement `key`.

2.3 Les fichiers texte

La lecture ou l'écriture de fichiers texte se fait en Python grâce à la fonction `open`.
Les paramètres de la fonction `open` sont :

- le nom d'un fichier situé dans le même répertoire que le fichier `.py` (ou son chemin relatif/absolu);
- la chaîne `"r"`, `"w"` ou `"a"`, selon que l'on souhaite un accès en lecture ou écriture ;
- le format d'encodage (par le mot-clef `encoding`).

La fonction `open` renvoie un objet de type `_io.TextIOWrapper` pour lequel sont définies un certain nombre de méthodes (fonctions) dont : `write`, `read` et `close`.

La méthode `close` permet de libérer la mémoire accordée à la lecture/écriture du fichier : il est recommandé de l'appeler de manière systématique lorsque la lecture ou l'écriture d'un fichier est terminée.

- Écriture d'un fichier texte au format UTF-8

  Le script ci-dessous crée un fichier texte nommé `data.txt` contenant la chaîne de caractère `s`.
  Si le fichier `data.txt` existe déjà, son contenu est irréversiblement écrasé.

```
s = "Une chaîne de caractères\nsur deux lignes"
writer = open("data.txt", "w", encoding="utf-8")
writer.write(s)
writer.close()
```

- Ajout d'un texte à la fin d'un fichier

  Il est possible d'ajouter du texte à la fin d'un fichier grâce au caractère `"a"` (*append* = ajouter à la fin).

```
s = "\nEncore du texte"
writer = open("data.txt", "a", encoding="utf-8")
writer.write(s)
writer.close()
```

- Lecture d'un fichier texte encodé en Latin-1

  Après exécution du script ci-dessous, le contenu du fichier `data.txt` est copié dans la chaîne de caractère `s`. Une fois que la méthode `read` a été appelée, l'objet `reader` n'est plus utilisable : tout autre appel de `read` renvoie une chaîne vide.

```
reader = open("data.txt", "r", encoding="latin-1")
s = reader.read()
reader.close()
print(s)
```

- Lecture ligne par ligne d'un fichier texte

  En mode lecture, l'objet `_io.TextIOWrapper` est itérable : à chaque tour de la boucle `for` dans le script suivant, la variable `line` prend successivement la valeur des lignes du fichier `data.txt`.

```
reader = open("data.txt", "r", encoding="utf-8")
for line in reader:
    print(line)
reader.close()
```

Exercice 20

Télécharger le fichier `exemple.txt`, qui a été encodé au format Latin-1, et écrire un script qui enregistre le contenu de ce fichier au format UTF-8 dans un fichier `exemple-utf8.txt`.