Numérique et Sciences Informatiques
Chapitre III - Les chaînes de caractères

Classe de terminale

I. Les jeux de caractères

Pour coder un caractère en binaire, on utilise un jeu de caractères codés qui associe des caractères d'un ou plusieurs systèmes d'écriture avec une représentation numérique.

Exemples de jeu de caractères

Ascii , ISO 8859-1, ISO-8859-1, Unicode, UTF-8...

I.1. Le code ASCII

L'ASCII (American Standard Code for Information Interchange) est composé est une norme de codage de caractères apparue dans les années 1960. Elle définit 128 caractères codés sur 7 bits. Or les ordinateurs travaillent souvent en octets et donc le bit fort d'un octet codant un caractère ASCII est toujours 0.

Les caractères de numéro 0 à 31 et le 127 ne sont pas imprimables. Ils correspondent à des commandes de contrôle de terminal informatique. Le caractère numéro 32 (20_{16}) est l'espace. Les autres caractères sont les chiffres arabes, les lettres latines majuscules et minuscules sans accent et quelques symboles de ponctuation.

Voici la table ASCII :

Décimal	Binaire	Hexadécimal	Caractère
0	0000 0000	0	
1	0000 0001	01	
2	0000 0010	02	
3	0000 0011	03	
4	0000 0100	04	
5	0000 0101	05	
6	0000 0110	06	
7	0000 0111	07	
8	0000 1000	08	
9	0000 1001	09	
10	0000 1010	0A	
11	0000 1011	0B	
12	0000 1100	0C	
13	0000 1101	0D	
14	0000 1110	0E	
15	0000 1111	0F	
16	0001 0000	10	
17	0001 0001	11	
18	0001 0010	12	
19	0001 0011	13	
20	0001 0100	14	
21	0001 0101	15	
22	0001 0110	16	
23	0001 0111	17	
24	0001 1000	18	
25	0001 1001	19	
26	0001 1010	1A	
27	0001 1011	1B	
28	0001 1100	1C	
29	0001 1101	1D	
30	0001 1110	1E	
31	0001 1111	1F	

Décimal	Binaire	Hexadécimal	Caractère
32	0010 0000	20	
33	0010 0001	21	!
34	0010 0010	22	”
35	0010 0011	23	#
36	0010 0100	24	$
37	0010 0101	25	%
38	0010 0110	26	&
39	0010 0111	27	’
40	0010 1000	28	(
41	0010 1001	29	)
42	0010 1010	2A	*
43	0010 1011	2B	+
44	0010 1100	2C	,
45	0010 1101	2D	-
46	0010 1110	2E	.
47	0010 1111	2F	/
18	0011 0000	30	0
49	0011 0001	31	1
50	0011 0010	32	2
51	0011 0011	33	3
52	0011 0100	34	4
53	0011 0101	35	5
54	0011 0110	36	6
55	0011 0111	37	7
56	0011 1000	38	8
57	0011 1001	39	9
58	0011 1010	3A	:
59	0011 1011	3B	;
60	0011 1100	3C	<
61	0011 1101	3D	=
62	0011 1110	3E	>
63	0011 1111	3F	?

Décimal	Binaire	Hexadécimal	Caractère
64	0100 0000	40	@
65	0100 0001	41	A
66	0100 0010	42	B
67	0100 0011	43	C
68	0100 0100	44	D
69	0100 0101	45	E
70	0100 0110	46	F
71	0100 0111	47	G
72	0100 1000	48	H
73	0100 1001	49	I
74	0100 1010	4A	J
75	0100 1011	4B	K
76	0100 1100	4C	L
77	0100 1101	4D	M
78	0100 1110	4E	N
79	0100 1111	4F	O
80	0101 0000	50	P
81	0101 0001	51	Q
82	0101 0010	52	R
83	0101 0011	53	S
84	0101 0100	54	T
85	0101 0101	55	U
86	0101 0110	56	V
87	0101 0111	57	W
88	0101 1000	58	X
89	0101 1001	59	Y
90	0101 1010	5A	Z
91	0101 1011	5B	[
92	0101 1100	5C	\
93	0101 1101	5D	]
94	0101 1110	5E	^
95	0101 1111	5F	_

Décimal	Binaire	Hexadécimal	Caractère
96	0110 0000	60	‘
97	0110 0001	61	a
98	0110 0010	62	b
99	0110 0011	63	c
100	0110 0100	64	d
101	0110 0101	65	e
102	0110 0110	66	f
103	0110 0111	67	g
104	0110 1000	68	h
105	0110 1001	69	i
106	0110 1010	6A	j
107	0110 1011	6B	k
108	0110 1100	6C	l
109	0110 1101	6D	m
110	0110 1110	6E	n
111	0110 1111	6F	o
112	0111 0000	70	p
113	0111 0001	71	q
114	0111 0010	72	r
115	0111 0011	73	s
116	0111 0100	74	t
117	0111 0101	75	u
118	0111 0110	76	v
119	0111 0111	77	w
120	0111 1000	78	x
121	0111 1001	79	y
122	0111 1010	7A	z
123	0111 1011	7B	{
124	0111 1100	7C	|
125	0111 1101	7D	}
126	0111 1110	7E	~
127	0111 1111	7F	

I.2. Le format ISO-8859-1

Par la suite d'autres encodages ont vu le jour afin de pallier les limites de l'ASCII.

Le format ISO-8859-1 (ou latin-1) étend donc l'ASCII avec des caractères accentués utiles aux langues originaires d'Europe occidentale. Il se code donc sur 8 bits (1 octet). Les 128 premiers caractères sont ceux de l'ASCII, ce qui assure la compatibilité avec ce dernier. Il sert principalement pour l'Europe occidentale.
Vous pouvez trouver le format ISO-8859-1 à cette adresse.

Pour le français il manque cependant le œ, le Œ et le Ÿ et, bien entendu, le symbole €.

I.3. Le format UTF-8

Pour pallier les limites de l'ISO-8859-1, le projet Unicode de codage unifié de tous les alphabets est né dans les années 1990. Différents codages sont utilisés pour représenter des caractères Unicode (UTF-8, UTF-16, UTF-32…).

Le codage UTF-8 est un codage de longueur variable. Certains caractères sont codés sur un seul octet, ce sont les 128 caractères du codage ASCII. Les autres caractères peuvent être codés sur 2, 3 ou 4 octets. Ainsi l'UTF-8 permet en théorie de représenter $2^{21} = 2097152$ caractères différents, en réalité un peu moins. Il y a actuellement environ une centaine de milliers de caractères Unicode (incluant les caractères des langues vivantes ou mortes et également de nombreux emojis 🌹)

Les caractères en UTF-8 doivent avoir une forme particulière décrite dans la table ci-dessous :

Nombre d'octets codant	Format de la représentation binaire
1	0xxxxxxx
2	110xxxxx 10xxxxxx
3	1110xxxx 10xxxxxx 10xxxxxx
4	111110xxx 10xxxxxx 10xxxxxx 10xxxxxx

L'encodage UTF-8 est lui aussi compatible avec l'ASCII. En revanche ISO-8859-1 et UTF-8 sont incompatibles entre eux pouvant conduire à ce genre de problèmes :

```
1 J'Ã©crit mal.
```

En effet, à priori, le 'é' est mal codé. Il prend certainement 2 octets en UTF-8 Mais est affiché en ISO-8859-1.

II. Notions de Python

II.1. Vocabulaire

Définition

Une **collection** est composé de plusieurs éléments, souvent de même type.

Définition

Une collection est dite **ordonnée** si les éléments qui la composent sont dans un certain ordre.
On peut ainsi accéder à un élément en connaissant son emplacement, appelé **indice**. On dit que l'élément de la collection est **indexé** par son indice.
Une collection qui n'est pas ordonnée est **non ordonnée**.

Remarque

Les indices commencent toujours à 0.

Exemple

```
1 >>> ma_collection = (5, 4, 6, 3, 2)
2 >>> ma_collection[2]
3 6
4 >>> ma_collection[0]
5 5
```

Définition

Un objet que l'on peut modifier est dit **mutable**.
Un objet que l'on ne peut pas modifier est dit **immuable**.

Exemple

```
>>> ma_collection = (5, 4, 6, 3, 2)
>>> ma_collection[2] = 3
Traceback (most recent call last):
...
TypeError: 'tuple' object does not support item assignment
>>> ma_deuxieme_collection = [5, 4, 6, 3, 2]
>>> ma_deuxieme_collection[2] = 3
>>> ma_deuxieme_collection
[5, 4, 3, 3, 2]
```

`ma_collection` est immuable alors que `ma_deuxiemme_collection` est mutable.

II.2. Les chaînes de caractères

Définition

Une chaîne de caractères est une collection immuable ordonnée de caractères. Elle peut être vide ou non vide.
En Python, on écrit les chaines de caractères entre `''` ou `""`.
Une chaîne de caractères est de type `str`.

Remarque

Soit l le nombre de caractères d'une chaîne de caractères.
Les indices peuvent être positifs ou négatifs, mais ne peuvent en aucun cas être supérieur ou égaux à l ou inférieurs à $-l$.

indice	0	1	2	...	$l-2$	$l-1$
autre indice	$-l$	$-(l-1)$	$-(l-2)$	...	-2	-1

Exemple

```
>>> ma_chaine = "abcdef"
>>> ma_chaine[2]
'c'
>>> ma_chaine[7]
Traceback (most recent call last):
...
IndexError: string index out of range
>>> ma_chaine[-2]
'e'
>>> ma_chaine[2] = 'e'
Traceback (most recent call last):
...
TypeError: 'str' object does not support item assignment
>>> type(ma_chaine)
<class 'str'>
>>> type(ma_chaine) == str
True
>>> ma_deuxieme_chaine = ""
```

II.3. Opérations sur les chaînes de caractères

II.3.a. La concaténation

Définition

Concaténer deux chaînes de caractères, c'est les mettre bout à bout.
On utilise le symbole `+` pour concaténer deux chaînes de caractères.

Remarque

L'ordre des opérandes autour du symbole `+` est important pour concaténer deux chaînes de caractères.

Exemple

```
>>> 'abc' + 'def'
 'abcdef'
>>> 'def' + 'abc'
 'defabc'
```

II.3.b. Opérations de comparaisons

Remarque

Une chaîne de caractères étant ordonnée, on peut comparer deux chaînes de caractères en partant des caractères d'indice 0, puis en cas d'égalité en comparant les caractères en incrémentant l'indice.

Exemple

```
>>> 'abc' < 'def'
 True
>>> 'abc' < 'abe'
 True
>>> 'abc' == 'abcde'
 False
>>> 'abc' > 'abcde'
 False
```

II.3.c. Le slicing

Définition

Le **slicing** d'une chaîne de caractère consiste à prendre une tranche de la chaîne de caractères.
En Python, on utilisera des indices entre crochets et séparés par `:`.

Remarque

Si on omet le premier indice (avant les `:`), alors on effectue un slicing depuis le début de la chaîne de caractères.
Si on omet le dernier indice (après les `:`), alors on effectue un slicing jusqu'à la fin de la chaîne de caractères.

Exemple

```
>>> chaine = 'abcdefghi'
>>> chaine[3:5]
'de'
>>> chaine[:5]
'abcde'
>>> chaine[3:]
'defghi'
```

II.4. La fonction len()

Définition

La fonction `len(chaine)` renvoie la **longueur** de la chaîne de caractère mise en paramètre, c'est à dire le nombre de caractères qui la composent.

Exemple

```
>>> chaine = 'abcdefghi'
>>> len(chaine)
9
>>> len('')
0
```

II.5. Méthodes sur les chaînes de caractères

Définition

Une **méthode** est une fonction qui s'applique uniquement sur un type d'objet.
Pour appliquer une méthode à un objet, on écrit en Python :

```
mon_objet.ma_methode(parametres)
```

II.5.a. La méthode find()

Définition

La méthode `chaine.find(sous_chaine)` renvoie l'indice de la première occurence de la sous-chaine `sous_chaine` dans la chaîne de caractères `chaine` si elle existe, -1 sinon.
En option, elle accepte deux autres paramètres qui correspondent aux indices de début (inclus) et de fin (exclu) d'une sous-chaîne de `chaine` dans laquelle on recherche la sous-chaine `sous_chaine`.

Exemple

```
>>> chaine = 'abcdefghi'
>>> chaine.find('cde')
2
>>> chaine.find('cdf')
-1
>>> chaine.find('cde', 2, 6)
2
```

II.5.b. La méthode index()

Définition

La méthode `chaine.index(sous_chaine)` renvoie l'indice d'une occurence de la sous-chaine `sous_chaine` dans la chaîne de caractères `chaine` si elle existe, sinon elle renvoie un message d'erreur.
En option, elle accepte deux autres paramètres qui correspondent aux indices de début (inclus) et de fin (exclu) d'une sous-chaîne de `chaine` dans laquelle on recherche la sous-chaine `sous_chaine`.

Exemple

```
>>> chaine = 'abcdefghi'
>>> chaine.index('cde')
2
>>> chaine.index('cdf')
Traceback (most recent call last):
...
ValueError: substring not found
>>> chaine.index('cde', 2, 6)
2
```

II.5.c. La méthode split()

Définition

La méthode `chaine.split(sep)` casse la chaîne de caractère `chaine` dès qu'elle rencontre le séparateur `sep` et renvoie les sous-chaînes ainsi trouvées dans un tableau (voir chapitre V).

Exemple

```
>>> chaine = 'abcdefghi'
>>> chaine.split('cde')
['ab', 'fghi']
>>> 'Bonjour à tous'.split(' ')
['Bonjour', 'à', 'tous']
```

II.6. Encodage d'une chaîne de caractères dans une certaine norme sous forme d'une suite d'octets

En Python, pour encoder une chaîne de caractères dans une certaine norme, il faut faire comme suit :

```
ma_chaine.encode(norme_encodage)
```

Exemple

```
>>> "bonjour".encode('cp1026')
b'\x82\x96\x95\x91\x96\xa4\x99'
```

Le « b » devant la réponse signifie « byte »(octet).

II.7. Décodage d'une suite d'octets correspondant à une chaîne de caractères dans une certaine norme

En Python, pour décoder une suite d'octets dans une certaine norme, il faut faire comme suit :

```
suite_octets.decode(norme_encodage)
```

Exemple

```
>>> b'\x82\x96\x95\x91\x96\xa4\x99'.decode('cp1026')
'bonjour'
```

II.8. La fonction str()

Définition

La fonction `str(chaine)` « transforme »un objet en une chaîne de caractères. Cette dernière est renvoyée par la fonction.

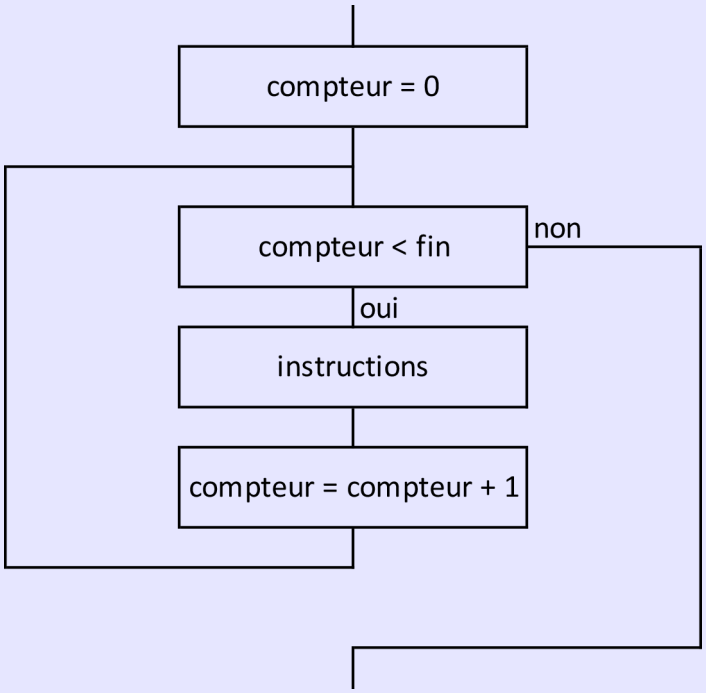
Exemple

```
>>> nb = 3
>>> str(nb)
'3'
```

II.9. Les boucles bornées

Définition

Lorsqu'on veut répéter un certain nombre de fois une séquence d'instructions, on utilise une boucle bornée (ou boucle Pour).
On utilise pour cela un **compteur** qui s'incrémente à chaque passage dans la boucle. Dès que le compteur dépasse une valeur de fin donnée, il n'y a plus de passage dans la boucle.



Remarque

Le compteur peut prendre la forme d'un nombre qui s'incrémente mais aussi de n'importe quel type d'élément appartenant à une collection ordonnée.

En Python, on écrit :

```
1 for element in collection:
2     instructions
```

Exemple

```
1 >>> res = ''
2 >>> for caractere in 'abcdef':
3         res = caractere + res
4 >>> res
5  'fedcba'
```

Ici, le compteur sous-entendu est l'indice du caractère `caractere` qui se trouve dans la chaîne `'abcdef'`.

II.10. la collection range

Définition

La collection `range` est une collection de nombres entiers.
Il y a trois façons de l'utiliser :

- `range(fin)` : collection des nombres entiers de `0` (inclus) à `fin` exclu.
- `range(debut, fin)` : collection des nombres entiers de `debut` (inclus) à `fin` exclu.
- `range(debut, fin, pas)` : collection des nombres entiers de `debut` (inclus) à `fin` exclu incrémentés de `pas`.

Exemple

`range(7)` correspond aux nombres 0, 1, 2, 3, 4, 5 et 6.

`range(2, 7)` correspond aux nombres 2, 3, 4, 5 et 6.

`range(2, 7, 3)` correspond aux nombres 2 et 5.

On peut utiliser la collection range dans les boucles `for` :

```
>>> chaine = 'abcdef'
>>> res = ''
>>> for indice in range(len(chaine)):
        res = chaine[indice] + res
>>> res
 'fedcba'
```

II.11. La fonction print()

Définition

La fonction `print(objet)` affiche une représentation de l'objet.

Exemple

```
>>> chaine = 'abcdef'
>>> print(chaine)
  abcdef
>>> print(3)
  3
```

III. Exercices

III.1. Exercice 1

Créer une fonction `affiche_verticalement(mot: str)` qui affiche verticalement la chaîne de caractère `mot`, c'est-à-dire que chaque caractère est affiché sur une ligne.

III.2. Exercice 2

1. Créer une fonction `caracteres_ascii() -> str` qui renvoie l'ensemble des caractères de la table ASCII, les uns à la suite des autres, sous forme d'une chaîne de caractères.

2. Créer une procedure `table_ascii_valeurs()` qui affiche l'ensemble des caractères de la table ASCII, sous forme d'un tableau de 128 lignes et 4 colonnes. Dans la première colonne, on trouve la valeur décimale du caractère, dans la deuxième colonne sa valeur binaire, dans la troisième colonne sa valeur hexadécimale et dans la dernière le caractère.

3. Créer une procedure `table_ascii()` qui affiche l'ensemble des caractères de la table ASCII, sous forme d'un tableau de 16 colonnes et 8 lignes.