Les opérations sur les chaînes de caractères

1. Le standard Unicode

Le standard Unicode maintient une table de codes contenant plus de 100 000 caractères et autres symboles, ceci pour environ une centaine de systèmes d'écritures. Les valeurs comprises entre 0 et 128 sont similaires aux valeurs de la table de caractères ASCII apparue dans les années 1960 et encore largement utilisée dans les ordinateurs jusque dans les années 2000. Dans ces tables, la valeur 32 correspond au caractère espace, la plage de valeurs 48 à 57 correspond aux chiffres de 0 à 9, la plage 65 à 90 donne les lettres majuscules de A à Z et la plage 97 à 122 correspond aux lettres minuscules de a à z.

Il existe deux fonctions `chr()` et `ord()` qui permettent de convertir un code en caractère, et inversement, en suivant le standard Unicode. Ainsi, si vous faites une recherche rapide sur Internet, vous trouverez que le symbole smiley correspond au code 128512, et que l'écriture de ce code en hexadécimal est 0x1F600. En langage Python, il n'y a pas de type caractère comme dans d'autres langages. Les fonctions retournant un caractère retournent ainsi une chaîne avec ce seul caractère à l'intérieur. Voici un exemple :

```
A = chr(128512)
B = chr(0x1F600)
print(A)
print(B)
print(type(A))
print(ord("A"))
print(ord("0"))

>> ☺
>> ☺
>> <class 'str'>
>> 65
>> 48
```

2. Concaténer deux chaînes

Le type chaîne de caractères permet d'utiliser l'opération + pour accoler deux chaînes en une. Pratique, non ?

```
a = 'Bonjour '
b = 'toi'
c = a + ' ' + b
print(c)

>> Bonjour toi
```

Vous pouvez tenter d'ajouter un nombre à une chaîne de caractères. Bien malin qui pourra prédire le résultat… car plusieurs options sont possibles. Soit le langage est plutôt souple et permissif sur ce point et il va convertir automatiquement le nombre en chaîne et l'accoler à l'autre chaîne. Soit le langage est strict et il va donner un message d'erreur.

```
a = ’Bonjour ’
b = 1
c = a + ’ ’ + b

>> c = a + ’ ’ + b
>> TypeError: must be str, not int
```

Ce message d’erreur n’est pas des plus clair. Une partie de l’activité du développeur consiste à apprendre à décoder les messages de l’environnement. Mais rassurez-vous, avec les quelques repères que vous avez déjà, cela devient possible. Le message parle de "TypeError" ce qui sous-entend qu’il y a un problème sur les types de variables rencontrées. La suite du message "must be str, not int" se traduit par : "Après l’opérateur +, j’attendais une chaîne de caractères (string) et vous m’avez donné un nombre entier (int), j’arrête là et je renvoie un message d’erreur". Le langage Python a donc un comportement strict sur ce point. Pas de souci, il suffit d’utiliser une fonction qui transforme un entier en chaîne de caractères à savoir la fonction `str()` :

```
a = ’Bonjour ’
b = 1
c = a + ’ ’ + str(b)
print(c)

>> Bonjour 1
```

3. Connaître la longueur d’une chaîne

On peut connaître la longueur d’une chaîne de caractères en utilisant la fonction `len()` :

```
A = "Bonjour"
print(len(A))

>> 7
```

4. Accéder à un caractère et à une sous-chaîne

On peut accéder à chaque caractère en écrivant son indice entouré d’une paire de crochets. L’indice 0 est associé au premier caractère en partant de la gauche. Ainsi le dernier caractère à droite porte l’indice 6 dans l’exemple précédent. La notation `A[1]` désigne le deuxième caractère en partant de la gauche, soit la lettre "o". Pour désigner le dernier caractère, on peut écrire l’indice -1, qui correspond à la première lettre en partant de la droite, soit "r" dans cet exemple. L’indice -2 correspond à la deuxième lettre en partant de la droite, et ainsi de suite :

```
A = "Bonjour"
print(A[0])
print(A[1])
print(A[-1])
print(A[-2])

>> B
>> o
```

```
>> r
>> u
```

Le langage Python permet d'utiliser des plages d'indices pour extraire une sous-chaîne. L'écriture `0:2` signifie que l'on prend tous les caractères de la position 0 jusqu'à la position 2 non incluse, ce qui donne la chaîne "Bo". Lorsque l'on écrit la plage d'indices `2:-2`, on extrait les lettres à partir de la troisième position en partant de la gauche jusqu'à la deuxième position non comprise en partant de la droite, on obtient ainsi la chaîne "njo".

Pour afficher les caractères à partir de la lettre "j" jusqu'à la fin, il faut écrire la plage `3:` sans préciser d'indice après le symbole. Et pour sélectionner les premiers caractères jusqu'à la lettre "n" d'indice 2, on utilise la plage `:3` sans préciser d'indice de départ.

```
A = "Bonjour"
print(A[0:2])
print(A[2:-2])
print(A[3:])
print(A[:3])
```

Ce qui donne comme résultat :

```
>> Bo
>> njo
>> jour
>> Bon
```

5. Transformer une chaîne en majuscules/mininuscules

Il est possible d'effectuer des opérations de transformation sur les chaînes. On peut citer par exemple les fonctions `upper()` et `lower()` qui retournent une chaîne composée uniquement de majuscules ou de minuscules respectivement. On accède à ces fonctions à travers une variable de type chaîne en utilisant la syntaxe `LaChaine.LaFonction()`

Contrairement à la fonction `len()` qui s'applique sur une variable : `len(chaîne)`, ces fonctions sont appelées depuis une variable avec l'utilisation du symbole . placé à droite du nom de la variable. Voici quelques exemples :

```
A = "Salut !!"
print(A.upper())
print(A.lower())

>> SALUT !!
>> salut !!
```

Notez que les fonctions `upper()` et `lower()` ne modifient pas la chaîne en cours, elles retournent une nouvelle chaîne, résultat du traitement.

6. Rechercher et remplacer une chaîne

La fonction `find()` permet de rechercher une chaîne à l'intérieur d'une autre chaîne et de retourner sa position. Dans la chaîne "Bonjour !", si nous recherchons la chaîne "Bon", nous la trouvons en tout début, c'est-à-dire à l'indice 0. De la même manière, la chaîne "jour" est détectée à la quatrième lettre, donc à l'indice 3. Si aucune correspondance n'est trouvée, la valeur -1 est retournée.

```
A = "Bonjour !"
print(A.find("Bon"))
print(A.find("jour"))
print(A.find("toto"))

>> 0
>> 3
>> -1
```

On peut remplacer une certaine partie de la chaîne par une autre grâce à la fonction `replace()` :

```
A = "Bonjour !"
B = A.replace("Bon","Abat-")
print(A)
print(B)

>> Bonjour !
>> Abat-jour !
```

La fonction `replace()` retourne une nouvelle chaîne sans modifier la chaîne originale, comme les fonctions `upper()` et `lower()`.

7. Afficher avec la fonction print()

La fonction `print()` permet d'afficher le contenu d'une ou de plusieurs variables. La syntaxe de la fonction `print()` est très pratique, car elle permet de passer autant de variables que l'on souhaite et de types quelconques. L'affichage liste les différentes valeurs en les séparant par un espace :

```
a = 2
b = "UN"
c = 4.8
print(a)
print(a,b)
print(a,b,c)

>> 2
>> 2 UN
>> 2 UN 4.8
```

Il est possible de changer ce séparateur en utilisant le paramètre optionnel `sep` :

```
print(a,b,c,sep='-')
print(a,b,c,sep='')       # absence de séparateur
print(a,b,c,sep='___')

>> 2-UN-4.8
>> 2UN4.8
>> 2___UN___4.8
```

8. Formater la sortie

Il existe un moyen pour organiser les affichages à l'écran. Nous allons donner une chaîne de caractères comportant le texte de sortie et les positions où seront insérées les variables à afficher. Ainsi, dans cette chaîne, nous écrivons `{i}`, où i désigne la position de la variable dans la série de variables passées en arguments. Attention, la première variable est positionnée à l'indice 0 :

```
print('{0} et {1}'.format('12', '3.222'))
print('{0} et {1}'.format('5', '3.1'))

>> 12 et 3.222
>> 5 et 3.1
```

Pour améliorer l'alignement de l'affichage en sortie, on peut imposer des longueurs et des formats aux valeurs affichées. Ainsi, en écrivant `{0,2d}`, on demande à la première variable de type entier de s'afficher sur deux colonnes. Si la valeur ne fait qu'une colonne de largeur, elle sera automatiquement calée sur la droite.

```
print('0123456789')
print('--{0:2d}--'.format(12))
print('--{0:2d}--'.format(5))

>> 0123456789
>> --12--
>> -- 5--
```

Pour un nombre à virgule, on peut demander un nombre maximal de colonnes pour son affichage ainsi qu'un nombre strict de colonnes pour la partie après la virgule. S'il y a trop de chiffres après la virgule, ils seront tronqués. S'il n'y en a pas assez, des zéros seront rajoutés :

```
print('01234567890123456789')
print('--{0:8.3f}--'.format(3.123456))
print('--{0:8.3f}--'.format(3.1))

>> 01234567890123456789
>> --   3.123--
>> --   3.100--
```

Nous avons demandé un affichage sur huit colonnes, dont trois pour les chiffres après la virgule. Ainsi, l'affichage occupe les colonnes 2 à 9, soit huit colonnes. Les chiffres après la virgule occupent toujours trois colonnes.

Pour les chaînes de caractères, il est possible de choisir le nombre de colonnes pour l'affichage, ainsi que l'alignement : gauche, droit ou centré.

```
print("--12345678--")
texte = 'abcd'
# alignement à gauche
print("--{:8}--".format(texte))
# alignement à droite
print("--{:>8}--".format(texte))
# texte centré
print("--{:^8}--".format(texte))

>> --12345678--
>> --abcd    --
>> --    abcd--
>> --  abcd  --
```

Ne vous focalisez pas sur les syntaxes concernant le formatage des affichages. Vous devez connaître leurs existences et savoir ce que le formatage permet de faire. En cas de besoin, revenez chercher l'information dans ce chapitre.