Les chaines de caractères

Computing & Society 2.1 International Alphabets and Unicode

The English alphabet is pretty simple: upper- and lowercase *a* to *z*. Other European languages have accent marks and special characters. For example, German has three so-called *umlaut* characters, ä, ö, ü, and a *double-s* character ß. These are not optional frills; you couldn't write a page of German text without using these characters a few times. German keyboards have keys for these characters.

The German Keyboard Layout

Many countries don't use the Roman script at all. Russian, Greek, Hebrew, Arabic, and Thai letters, to name just a few, have completely different shapes. To complicate matters, Hebrew and Arabic are typed from right to left. Each of these alphabets has about as many characters as the English alphabet.

Hebrew, Arabic, and English

The Chinese languages as well as Japanese and Korean use Chinese characters. Each character represents an idea or thing. Words are made up of one or more of these ideographic characters. Over 70,000 ideographs are known.

Starting in 1988, a consortium of hardware and software manufacturers developed a uniform encoding scheme called Unicode that is capable of encoding text in essentially all written languages of the world.

Today Unicode defines over 100,000 characters. There are even plans to add codes for extinct languages, such as Egyptian hieroglyphics.

The Chinese Script

Les chaines de caractères

1. Les données composites

Les chaines de caractères constituent un cas particulier d'un type de données plus général que l'on appelle des ***données composites***. On retrouve dans ces composites :

✓ **Les chaines de caractères**
✓ **Les tuples**
✓ **Les listes**
✓ **Les dictionnaires**

Une donnée composite est une entité qui rassemble dans une seule structure un ensemble d'entités plus simples (ex : les caractères d'une chaine).

2. Les chaines de caractères

2.1. Accès aux caractères individuels d'une chaine

Un langage de programmation tel que Python doit donc être pourvu de mécanismes qui permettent d'accéder séparément à chacun des caractères d'une chaine.

Python considère qu'une chaine de caractères est un objet de la ***catégorie des séquences***, lesquelles sont des ***collections ordonnées d'éléments***. Chaque caractère dans une chaine peut donc être désigné par sa place dans la séquence, ***à l'aide d'un index***.

Pour accéder à un caractère bien déterminé, on utile le nom de la variable qui contient la chaine et on lui accole***, entre 2 crochets***, l'index numérique qui correspond à la position du caractère dans la chaine.

! Attention : numérotation à partir de 0 !

```
ch="Classe préparatoire"
print(ch[0])
print(ch[6])

a=ch[7]
print(a)

print(type(a))
```

Rappel : nous avons vu que c'est sur la base de la norme internationale ***UNICODE*** que python codait ses caractères dans ses espaces mémoire.

2.2. Opérations élémentaires sur les chaines

Python intègre de nombreuses fonctions qui permettent d'effectuer divers traitements sur les chaines de caractères :

✔ Conversion majuscule/minuscule
✔ Découpage en chaines plus petites
✔ Recherche de mots
✔ Concaténation de mots…

Les chaines de caractères

Les opérations élémentaires sur les chaines sont :

✔ ***Assembler plusieurs petites chaines*** pour en construire de plus grandes = CONCATENATION.
On utilise dans python l'opérateur « + ».

```
a="Petit poisson"
b=" deviendra grand"
c=a+b
print(c)
```

✔ Déterminer la ***longueur (nombre de caractères)*** d'une chaine, en faisant appel à la fonction intégrée : `len()`.

```
a="Bonjour à tous"
print(a)
print(len(a))
```

✔ ***Convertir en nombre véritable*** une chaine de caractères qui représente un nombre : il faut passer par la fonction `int()` qui convertit la chaine en entier.

```
ch="f(x)=2x"

print(ch[6
coef1=ch[6]*3
print(coef1)

print(ch[5])
coef2=ch[5]*3
print(coef2)

print(ch[5])
coef3=ch[5]+3
print(coef3)
```

```
ch="f(x)=2x"

print(ch[5])
coef3=int(ch[5])+3
print(coef3)
print(type(coef3))
```

✔ ***Extraction*** ou « ***slicing*** » : extraire une petite chaine d'une chaine plus longue, c'est du slicing dans python. On indique entre crochets les indices correspondants au début et fin de la « tranche ».

```
ch="Juliette"

print(ch[0:3])

print(ch[3:7])

print(ch[3:])

print(ch[:3])

print(ch[-1])
```

Attention : Dans la tranche [n,m], le $n^{ième}$ caractère est inclus, mais pas le $m^{ième}$.

Les chaines de caractères

✔ ***Parcours d'une séquence :*** Pour parcourir une chaine de caractères, il suffit d'utiliser une boucle de type : `for … in ….`

```
nom="Cleopatre"
for car in nom:
    print(car,"*",end="")
```

La variable car contiendra successivement tous les caractères de la chaine.

Le traitement que l'on vient de voir s'appliquera aussi aux ***listes*** (cf plus loin dans ce cours).

```
liste = ['chien', 'chat', 'crocodile', 'éléphant']
for animal in liste:
    print('longueur de la chaîne', animal, '=', len(animal))
```

L'exécution de ce script donne :

```
longueur de la chaîne chien = 5
longueur de la chaîne chat = 4
longueur de la chaîne crocodile = 9
longueur de la chaîne éléphant = 8
```

Le nom qui suit le mot réservé in est celui de la séquence qu'il faut traiter. Le nom qui suit le for est celui que vous choisirez pour la variable destinée à contenir successivement tous les éléments de la séquence. Cette variable est définie automatiquement, pas besoin de la définir avant !!.

Autre exemple d'opération :

```
for i in range ( len(ma_chaine) ):
        caract = ma_chaine[i]
```

```
nom="Cleopatre"
for i in range(len(nom)):
    caract=nom[i]
    print(caract,"*",end="")
```

✔ ***Trouver un caractère dans une chaine :*** Pour trouver à quel index (ou rang) apparait un caractère dans une chaîne, on utilise l'instruction: `<var>.find (« car »).`

```
ch="il fait beau aujourd'hui mais très froid"
print(ch.find("e"))
```

Attention : il n'indique le rang que du premier rencontré…

✔ ***Compter le nombre de fois qu'apparait un caractère dans une chaine :*** Pour compter le nombre de fois qu'apparait un caractère dans une chaîne, on utilise l'instruction: `<var>.count (« car »).`

```
ch="nous allons chercher combien il y a de l"
print(ch.count("l"))
```

2.3. L'encodage Utf-8

La norme Utf-8 est la norme préférentielle pour la plupart des textes courants parce que :

✔ d'une part, elle assure une parfaite compatibilité avec les textes en « pur » ***ASCII*** (beaucoup de codes sources de logiciels)
✔ d'autre part, cette norme est celle qui est la plus économe en ressources, tout au moins pour les textes écrits dans une langue occidentale

En Utf-8, les caractères du code ASCII sont encodés sur ***un seul octet***.

Exercice de SYNTHESE:

On donne les extraits de code Python suivants, en face de chaque bloc d'instructions, noter le résultat affiché.

```
ch="bonjour"
print(ch[2])
print(ch[:3])
print(ch[3:])
print(ch[2:6])
print(ch.find("j"))

for car in ch:
    print(car,end=" ")

print()

for i in range(len(ch)):
    print(i, end=" ")

print()

for i in range(len(ch)):
    print(ch[i], end=" ")
```