Python : Chaînes de caractères

1ère NSI

Plan du cours

Objet de type `str` - Notion d'objet, de méthode

- Un **objet** est une sorte de **super-variable** possédant des caractéristiques (**attributs**) et des fonctions qui lui sont propres (**méthodes**)

Objet de type `str` - Notion d'objet, de méthode

- Un **objet** est une sorte de **super-variable** possédant des caractéristiques (**attributs**) et des fonctions qui lui sont propres (**méthodes**)
- En Python, **tout est objet** !

Objet de type `str` - Notion d'objet, de méthode

- Un **objet** est une sorte de **super-variable** possédant des caractéristiques (**attributs**) et des fonctions qui lui sont propres (**méthodes**)
- En Python, **tout est objet** !
- Un exemple simple d'objet : les **chaînes de caractères**

```
1 >>> chaine = "Salut ! Je suis une chaîne de caractères."
```

Contrairement à une variable simple comme un `int`, on peut « manipuler » une chaîne de caractères de façon très spécifique...

Majuscules et minuscules avec `.upper()` et `.lower()`

- Pour mettre une chaîne de caractères en minuscules ou en majuscules, on utilise les méthodes `upper` et `lower` de la façon suivante :

```
1  >>> chaine = "Taisez-vous !"
2  >>> chaine.upper()
3  'TAISEZ-VOUS !'
4  >>> chaine.lower()
5  'taisez-vous !'
```

Majuscules et minuscules avec `.upper()` et `.lower()`

- Pour mettre une chaîne de caractères en minuscules ou en majuscules, on utilise les méthodes `upper` et `lower` de la façon suivante :

```
1 >>> chaine = "Taisez-vous !"
2 >>> chaine.upper()
3 'TAISEZ-VOUS !'
4 >>> chaine.lower()
5 'taisez-vous !'
```

- Ces méthodes renvoient une **copie de la chaîne modifiée** : la chaîne de départ reste inchangée !

```
1 >>> chaine = "salut"
2 >>> chaineMaj = chaine.upper()
3 >>> chaine
4 'salut'
5 >>> chaineMaj
6 'SALUT'
```

Compter les occurences avec `.count()`

- La méthode `count` permet de compter le nombre d'occurences d'une chaîne dans une autre :

```
1 >>> chaine = "Bonjour"
2 >>> chaine.count("o")
3 2
4 >>> chaine.count("jour")
5 1
```

Compter les occurences avec `.count()`

- La méthode `count` permet de compter le nombre d'occurences d'une chaîne dans une autre :

```
1  >>> chaine = "Bonjour"
2  >>> chaine.count("o")
3  2
4  >>> chaine.count("jour")
5  1
```

- La méthode retourne le nombre de fois où apparaît la chaîne passée en paramètres dans la chaîne sur laquelle s'applique le `count`.

Compter les occurences avec `.count()`

- La méthode `count` permet de compter le nombre d'occurences d'une chaîne dans une autre :

```
1  >>> chaine = "Bonjour"
2  >>> chaine.count("o")
3  2
4  >>> chaine.count("jour")
5  1
```

- La méthode retourne le nombre de fois où apparaît la chaîne passée en paramètres dans la chaîne sur laquelle s'applique le `count`.
- Comme toujours, Python est **sensible à la casse** : il fait la différence entre majuscules et minuscules.

Modifier une chaîne avec `.replace`

- Python permet de remplacer un morceau de chaîne par un autre, avec la méthode `replace`.

Modifier une chaîne avec `.replace`

- Python permet de remplacer un morceau de chaîne par un autre, avec la méthode `replace`.
- Dans l'exemple suivant, on remplace tous les occurences du mot `Python` par `Pokémon` :

```
1  >>> chaine = "J'adore Python. Python, c'est trop bien !"
2  >>> chaine.replace("Python", "Pokémon")
3  "J'adore Pokémon. Pokémon, c'est trop bien !"
```

Modifier une chaîne avec `.replace`

- Python permet de remplacer un morceau de chaîne par un autre, avec la méthode `replace`.
- Dans l'exemple suivant, on remplace tous les occurences du mot `Python` par `Pokémon` :

```
1 >>> chaine = "J'adore Python. Python, c'est trop bien !"
2 >>> chaine.replace("Python", "Pokémon")
3 "J'adore Pokémon. Pokémon, c'est trop bien !"
```

- On peut limiter le nombre de remplacements avec un troisième argument facultatif :

```
1 # On remplace seulement la 1ère occurence
2 >>> chaine.replace("Python", "Pokémon", 1)
3 "J'adore Pokémon. Python, c'est trop bien !"
```

Rechercher avec `.find`

- La méthode `find` permet de rechercher un caractère ou une chaîne de caractères dans la chaîne considérée.

Rechercher avec `.find`

- La méthode `find` permet de rechercher un caractère ou une chaîne de caractères dans la chaîne considérée.
- Elle retourne l'**indice** de la première occurence trouvée, ou -1 si la chaîne passée en paramètres n'est pas dans la chaîne sur laquelle on applique le `find` :

```
1 >>> "Bonjour".find("j")
2 3
3 >>> "Bonjour".find("i") # Pas de i dans 'Bonjour'
4 -1
```

Indices et chaînes de caractères- Indice simple

- Dans une chaîne de caractères, chaque caractère est repéré par son **indice**, qui correspond à son emplacement dans la chaîne

Indices et chaînes de caractères- Indice simple

- Dans une chaîne de caractères, chaque caractère est repéré par son **indice**, qui correspond à son emplacement dans la chaîne
- Comme toujours en informatique, on aime bien compter à partir de 0 :

Caractères	B	o	n	j	o	u	r
Indice	0	1	2	3	4	5	6

Indices et chaînes de caractères- Indice simple

- Dans une chaîne de caractères, chaque caractère est repéré par son **indice**, qui correspond à son emplacement dans la chaîne
- Comme toujours en informatique, on aime bien compter à partir de 0 :

Caractères	B	o	n	j	o	u	r
Indice	0	1	2	3	4	5	6

- Pour accéder au caractère associé à un indice donné dans une chaîne, on utilise la syntaxe suivante :

```
1  >>> chaine = "Bonjour"
2  >>> chaine[3]
3  'j'
4  >>> chaine[5]
5  'u'
```

Indices et chaînes de caractères- Indice simple

- Attention à ne pas dépasser la taille de la chaîne !

```
1  >>> chaine = "Bonjour"
2  >>> chaine[7]
3  Traceback (most recent call last):
4    File "<stdin>", line 1, in <module>
5  IndexError: string index out of range
```

Indices et chaînes de caractères- Indice simple

- Attention à ne pas dépasser la taille de la chaîne !

```
>>> chaine = "Bonjour"
>>> chaine[7]
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IndexError: string index out of range
```

- Les **indices négatifs** sont autorisés, on compte alors depuis la fin de la chaîne :

```
>>> chaine = "Bonjour"
>>> chaine[-1]
'r'
>>> chaine[-3]
'o'
```

Indices et chaînes de caractères - *Slice* (plage d'indices)

- Il est possible d'extraire une **sous-chaîne** d'une chaîne donnée avec la syntaxe suivante :

```
1 | chaine[debut:fin:pas]
```

Indices et chaînes de caractères - *Slice* (plage d'indices)

- Il est possible d'extraire une **sous-chaîne** d'une chaîne donnée avec la syntaxe suivante :

```
chaine[debut:fin:pas]
```

- Comme dans une boucle `for`, on extrait alors tous les caractères dont l'indice varie de `debut` jusqu'à `fin-1` inclus.

```
>>> phrase = "J'aime bien les fraises"
>>> phrase[:8] # Du début à l'indice 7
"J'aime b"
>>> phrase[7:11] # De l'indice 7 à l'indice 10
'bien'
>>> phrase[16:] # De l'indice 16 à la fin
'fraises'
```

- Par défaut, l'indice de départ est 0, et l'indice de fin correspond au dernier caractère.