```
>>> r.index(10)
5
>>> r[5]
10
>>> r[:5]
range(0, 10, 2)
>>> r[-1]
18
```

Comparer des *range* avec == et != les compare comme des séquences. Soit deux objets *range* sont considérées comme égaux si ils représentent la même séquence de valeurs. (Notez que deux objets *range* dits égaux pourraient avoir leurs attributs `start`, `stop` et `step` différents, par exemple `range(0) == range(2, 1, 3)` ou `range(0, 3, 2) == range(0, 4, 2)`.)

Modifié dans la version 3.2: Implémente la classe de base abstraite *Sequence*. Supporte le *slicing* et les indices négatifs. Tester l'appartenance d'un `int` en temps constant au lieu d'itérer tous les éléments.

Modifié dans la version 3.3: == et != comparent des *range* en fonction de la séquence de valeurs qu'ils définissent (au lieu d'une comparaison fondée sur l'identité de l'objet).

Nouveau dans la version 3.3: Les attributs `start`, `stop` et `step`.

Voir aussi:

- The linspace recipe shows how to implement a lazy version of range suitable for floating point applications.

4.7. Type Séquence de Texte — `str`

Les données textuelles en Python est manipulé avec des objets `str` ou *strings*. Les chaînes sont des séquences immuables de points de code Unicode. Les chaînes littérales peuvent être écrites de différentes manières :

- Les guillemets simples : `'autorisent les "guillemets"'`
- Les guillemets : `"autorisent les guillemets 'simples'"`.
- Guillemets triples : `'''Trois guillemets simples'''`, `"""Trois guillemets"""`

Les chaînes entre triple guillemets peuvent couvrir plusieurs lignes, tous les espaces associés seront inclus dans la chaîne littérale.

Les chaînes littérales qui font partie d'une seule expression et ont seulement des espaces entre elles sont implicitement converties en une seule chaîne littérale. Autrement dit, `("spam " "eggs") == "spam eggs"`.

Voir Littéraux de chaînes de caractères et de suites d'octets pour plus d'informations sur les différentes formes de chaînes littérales, y compris des séquences d'échappement prises en charge, et le préfixe `r` (*raw* (brut)) qui désactive la plupart des traitements de séquence d'échappement.

Les chaînes peuvent également être créés à partir d'autres objets à l'aide du constructeur `str`.

Comme il n'y a pas de type « caractère » distinct, l'indexation d'une chaîne produit des chaînes de longueur 1. Autrement dit, pour une chaîne non vide *s*, `s[0] == s[0:1]`.

Il n'y a aucun type de chaîne muable, mais `str.join()` ou `io.StringIO` peuvent être utilisées pour construire efficacement des chaînes à partir de plusieurs fragments.

Modifié dans la version 3.3: Pour une compatibilité ascendante avec la série Python 2, le préfixe `u` est à nouveau autorisé sur les chaînes littérales. Elle n'a aucun effet sur le sens des chaînes littérales et ne peut être combiné avec le préfixe `r`.

class **str**(*object=''*)
class **str**(*object=b''*, *encoding='utf-8'*, *errors='strict'*)

Renvoie une représentation string de *object*. Si *object* n'est pas fourni, renvoie une chaîne vide. Sinon, le comportement de `str()` dépend de si *encoding* ou *errors* sont donnés, voir l'exemple.

Si ni *encoding* ni *errors* ne sont donnés, `str(object)` renvoie `object.__str__()`, qui est la représentation de chaîne « informelle » ou bien affichable de *object*. Pour les chaînes, c'est la chaîne elle-même. Si *object* n'a pas de méthode `__str__()`, `str()` utilise `repr(object)`.

Si au moins un des deux arguments *encoding* ou *errors* est donné, *object* doit être un bytes-like object (par exemple `bytes` ou `bytearray`). Dans ce cas, si *object* est un objet `bytes` (ou `bytearray`), alors `str(bytes, encoding, errors)` est équivalent à `bytes.decode(encoding, errors)`. Sinon, l'objet *bytes* du *buffer* est obtenu avant d'appeler `bytes.decode()`. Voir Séquences Binaires — bytes, bytearray, memoryview et Protocole tampon pour plus d'informations sur les *buffers*.

Donner un objet `bytes` à `str()` sans ni l'argument *encoding* ni l'argument *errors* relève du premier cas, où la représentation informelle de la chaîne est renvoyé (voir aussi l'option `-b` de Python). Par exemple :

```
>>> str(b'Zoot!')
"b'Zoot!'"
```

Pour plus d'informations sur la classe `str` et ses méthodes, voir les sections Type Séquence de Texte — str et Méthodes de chaînes de caractères. Pour formater des chaînes de caractères, voir les sections Chaînes de caractères formatées littérales et Syntaxe de formatage de chaîne. La section Services de Manipulation de Texte contient aussi des informations.

4.7.1. Méthodes de chaînes de caractères

Les chaînes implémentent toutes les opérations communes des séquences, ainsi que les autres méthodes décrites ci-dessous.

Les chaînes gèrent aussi deux styles de mise en forme, l'un fournissant une grande flexibilité et de personnalisation (voir `str.format()`, Syntaxe de formatage de chaîne et Formatage personnalisé de chaîne) et l'autre basée sur `printf` du C qui gère une gamme plus étroite des

types et est légèrement plus difficile à utiliser correctement, mais il est souvent plus rapide pour les cas, il peut gérer (Formatage de chaines à la printf).

La section Services de Manipulation de Texte de la bibliothèque standard couvre un certain nombre d'autres modules qui fournissent différents services relatifs au texte (y compris les expressions régulières dans le module `re`).

`str.`**`capitalize`**()

Renvoie une copie de la chaîne avec son premier caractère en majuscule et le reste en minuscule.

`str.`**`casefold`**()

Renvoie une copie *casefolded* de la chaîne. Les chaînes *casefolded* peuvent être utilisées dans des comparaison insensibles à la casse.

Le *casefolding* est une technique agressive de mise en minuscule, car il vise à éliminer toutes les distinctions de casse dans une chaîne. Par exemple, la lettre minuscule `'ß'` de l'allemand équivaut à `"ss"`. Comme il est déjà minuscule, `lower()` ferait rien à `'ß'`; `casefold()` le convertit en `"ss"`.

L'algorithme de *casefolding* est décrit dans la section 3.13 de la norme Unicode.

Nouveau dans la version 3.3.

`str.`**`center`**(*width*[, *fillchar*])

Donne la chaîne au centre d'une chaîne de longueur *width*. Le remplissage est fait en utilisant l'argument *fillchar* (qui par défaut est un espace ASCII). La chaîne d'origine est renvoyée si *width* est inférieur ou égale à `len(s)`.

`str.`**`count`**(*sub*[, *start*[, *end*]])

Donne le nombre d'occurrences de *sub* ne se chevauchant pas dans le *range* [*start*, *end*]. Les arguments facultatifs *start* et *end* sont interprétés comme pour des *slices*.

`str.`**`encode`**(*encoding="utf-8"*, *errors="strict"*)

Donne une version encodée de la chaîne sous la forme d'un objet *bytes*. L'encodage par défaut est `'utf-8'`. *errors* peut être donné pour choisir un autre système de gestion d'erreur. La valeur par défaut pour *errors* est `'strict'`, ce qui signifie que les erreurs d'encodage lèvent une `UnicodeError`. Les autres valeurs possibles sont `'ignore'`, `'replace'`, `'xmlcharrefreplace'`, `'backslashreplace'` et tout autre nom enregistré *via* `codecs.register_error()`, voir la section Error Handlers. Pour une liste des encodages possibles, voir la section Standard Encodings.

Modifié dans la version 3.1: Gestion des arguments par mot clef.

`str.`**`endswith`**(*suffix*[, *start*[, *end*]])

Donne `True` si la chaîne se termine par *suffix*, sinon `False`. *suffix* peut aussi être un tuple de suffixes à rechercher. Si l'argument optionnel *start* est donné, le test se fait à partir de cette position. Si l'argument optionnel *end* est fourni, la comparaison s'arrête à cette position.

`str.`**`expandtabs`**(*tabsize=8*)

Donne une copie de la chaîne où toutes les tabulations sont remplacées par un ou plusieurs espaces, en fonction de la colonne courante et de la taille de tabulation donnée. Les positions des tabulations se trouvent tous les *tabsize* caractères (8 par défaut, ce qui donne les positions de tabulations aux colonnes 0, 8, 16 et ainsi de suite). Pour travailler sur la chaîne, la colonne en cours est mise à zéro et la chaîne est examinée caractère par caractère. Si le caractère est une tabulation (`\t`), un ou plusieurs caractères d'espacement sont insérés dans le résultat jusqu'à ce que la colonne courante soit égale à la position de tabulation suivante. (Le caractère tabulation lui-même n'est pas copié.) Si le caractère est un saut de ligne (`\n`) ou un retour chariot (`\r`), il est copié et la colonne en cours est remise à zéro. Tout autre caractère est copié inchangé et la colonne en cours est incrémentée de un indépendamment de la façon dont le caractère est représenté lors de l'affichage.

```
>>> '01\t012\t0123\t01234'.expandtabs()
'01      012     0123    01234'
>>> '01\t012\t0123\t01234'.expandtabs(4)
'01  012 0123    01234'
```

`str.`**`find`**(*sub*[, *start*[, *end*]])

Donne la première la position dans la chaîne où *sub* est trouvé dans le *slice* `s[start:end]`. Les arguments facultatifs *start* et *end* sont interprétés comme dans la notation des *slice*. Donne `-1` si *sub* n'est pas trouvé.

Note: La méthode `find()` ne doit être utilisée que si vous avez besoin de connaître la position de *sub*. Pour vérifier si *sub* est une sous chaine ou non, utilisez l'opérateur `in` :

```
>>> 'Py' in 'Python'
True
```

`str.`**`format`**(**args*, ***kwargs*)

Formate une chaîne. La chaîne sur laquelle cette méthode est appelée peut contenir du texte littéral ou des emplacements de remplacement délimités par des accolades `{}`. Chaque champ de remplacement contient soit l'indice numérique d'un argument positionnel, ou le nom d'un argument donné par mot-clé. Renvoie une copie de la chaîne où chaque champ de remplacement est remplacé par la valeur de chaîne de l'argument correspondant.

```
>>> "The sum of 1 + 2 is {0}".format(1+2)
'The sum of 1 + 2 is 3'
```

Voir Syntaxe de formatage de chaîne pour une description des options de formatage qui peuvent être spécifiées dans les chaînes de format.

Note: When formatting a number (`int`, `float`, `complex`, `decimal.Decimal` and subclasses) with the `n` type (ex: `'{:n}'.format(1234)`), the function temporarily sets the `LC_CTYPE` locale to the `LC_NUMERIC` locale to decode `decimal_point` and `thousands_sep` fields of `localeconv()` if they are non-ASCII or longer than 1 byte, and the `LC_NUMERIC` locale is different than the `LC_CTYPE` locale. This temporary change affects other threads.

Modifié dans la version 3.6.5: Lors du formatage d'un nombre avec le format `n`, la fonction change temporairement `LC_CTYPE` par la valeur de `LC_NUMERIC` dans certains cas.

`str.`**`format_map`**(*mapping*)

Semblable à `str.format(**mapping)`, sauf que `mapping` est utilisé directement et non copié dans un `dict`. C'est utile si, par exemple `mapping` est une sous-classe de `dict` :

```
>>> class Default(dict):
...     def __missing__(self, key):
...         return key
...
>>> '{name} was born in {country}'.format_map(Default(name='Guido'))
'Guido was born in country'
```

Nouveau dans la version 3.2.

`str.`**`index`**(*sub*[, *start*[, *end*]])

Comme `find()`, mais lève une `ValueError` lorsque la chaîne est introuvable.

`str.`**`isalnum`**()

Donne `True` si tous les caractères de la chaîne sont alphanumériques et qu'il y a au moins un caractère, sinon `False`. Un caractère `c` est alphanumérique si l'un des tests suivants est vrais : `c.isalpha()`, `c.isdecimal()`, `c.isdigit()` ou `c.isnumeric()`.

`str.`**`isalpha`**()

Donne `True` si tous les caractères de la chaîne sont alphabétiques et qu'elle contient au moins un caractère, sinon `False`. Les caractères alphabétiques sont les caractères définis dans la base de données de caractères Unicode comme *Letter*, à savoir, ceux qui ont « Lm », « Lt », « Lu », « Ll », ou « Lo » comme catégorie générale. Notez que ceci est différent de la propriété *Alphabetic* définie dans la norme Unicode.

`str.`**`isdecimal`**()

Renvoie vrai si tous les caractères de la chaîne sont des caractères décimaux et qu'elle contient au moins un caractère, sinon faux. Les caractères décimaux sont ceux pouvant être utilisés pour former des nombres en base 10, tels que U+0660, ARABIC-INDIC DIGIT ZERO. Spécifiquement, un caractère décimal est un caractère dans la catégorie Unicode générale « Nd ».

`str.`**`isdigit`**()

Renvoie vrai si tous les caractères de la chaîne sont des chiffres et qu'elle contient au moins un caractère, faux sinon. Les chiffres incluent des caractères décimaux et des chiffres qui nécessitent une manipulation particulière, tels que les *compatibility superscript digits*. Ça couvre les chiffres qui ne peuvent pas être utilisés pour construire des nombres en base 10, tel que les nombres de Kharosthi. Spécifiquement, un chiffre est un caractère dont la valeur de la propriété *Numeric_Type* est *Digit* ou *Decimal*.

`str.`**`isidentifier`**()

Donne `True` si la chaîne est un identifiant valide selon la définition du langage, section Identifiants et mots-clés.

Utilisez `keyword.iskeyword()` pour savoir si un identifiant est réservé, tels que `def` et `class`.

`str.`**`islower`**()

Donne `True` si tous les caractères capitalisables [4] de la chaîne sont en minuscules et qu'elle contient au moins un caractère capitalisable. Donne `False` dans le cas contraire.

`str.`**`isnumeric`**()

Donne `True` si tous les caractères de la chaîne sont des caractères numériques, et qu'elle contient au moins un caractère, sinon `False`. Les caractères numériques comprennent les chiffres, et tous les caractères qui ont la propriété Unicode *numeric value*, par exemple U+2155, *VULGAR FRACTION OF FIFTH*. Formellement, les caractères numériques sont ceux avec les priorités *Numeric_Type=Digit*, *Numeric_Type=Decimal*, ou *Numeric_Type=Numeric*.

`str.`**`isprintable`**()

Donne `True` si tous les caractères de la chaîne sont affichables ou qu'elle est vide sinon, `False`. Les caractères non affichables sont les caractères définis dans la base de données de caractères Unicode comme *« Other »* ou *« Separator »*, à l'exception de l'espace ASCII (`0x20`) qui est considéré comme affichable. (Notez que les caractères imprimables dans ce contexte sont ceux qui ne devraient pas être protégés quand `repr()` est invoquée sur une chaîne. Ça n'a aucune incidence sur le traitement des chaînes écrites sur `sys.stdout` ou `sys.stderr`.)

`str.`**`isspace`**()

Donne `True` s'il n'y a que des caractères blancs dans la chaîne et il y a au moins un caractère, sinon `False`. Les caractères blancs sont les caractères définis dans la base de données de caractères Unicode comme *« Other »* ou *« Separator »* ainsi que ceux ayant la propriété bidirectionnelle valant « WS », « B » ou « S ».

`str.`**`istitle`**()

Donne `True` si la chaîne est une chaîne *titlecased* et qu'elle contient au moins un caractère, par exemple les caractères majuscules ne peuvent suivre les caractères non capitalisables et les caractères minuscules ne peuvent suivre que des caractères capitalisables. Donne `False` dans le cas contraire.

`str.`**`isupper`**()

Donne `True` si tous les caractères différentiables sur la casse [4] de la chaîne sont en majuscules et il y a au moins un caractère différentiable sur la casse, sinon `False`.

`str.`**`join`**(*iterable*)

Donne une chaîne qui est la concaténation des chaînes contenues dans *iterable*. Une `TypeError` sera levée si une valeur d'"*iterable* n'est pas une chaîne, y compris pour les objets `bytes`. Le séparateur entre les éléments est la chaîne fournissant cette méthode.

`str.`**`ljust`**(*width*[, *fillchar*])

Renvoie la chaîne justifiée à gauche dans une chaîne de longueur *width*. Le rembourrage est fait en utilisant *fillchar* (qui par défaut est un espace ASCII). La chaîne d'origine est renvoyée si *width* est inférieur ou égale à `len(s)`.

`str.`**`lower`**()

Renvoie une copie de la chaîne avec tous les caractères capitalisables [4] convertis en minuscules.

L'algorithme de mise en minuscules utilisé est décrit dans la section 3.13 de la norme Unicode.

`str.`**`lstrip`**([*chars*])

Renvoie une copie de la chaîne des caractères supprimés au début. L'argument *chars* est une chaîne spécifiant le jeu de caractères à supprimer. En cas d'omission ou `None`, la valeur par défaut de *chars* permet de supprimer des espaces. L'argument *chars* n'est pas un préfixe, toutes les combinaisons de ses valeurs sont supprimées :

```
>>> '   spacious   '.lstrip()
'spacious   '
>>> 'www.example.com'.lstrip('cmowz.')
'example.com'
```

static `str.`**`maketrans`**(*x*[, *y*[, *z*]])

Cette méthode statique renvoie une table de traduction utilisable pour `str.translate()`.

Si un seul argument est fourni, ce soit être un dictionnaire faisant correspondre des points de code Unicode (nombres entiers) ou des caractères (chaînes de longueur 1) à des points de code Unicode.

Si deux arguments sont fournis, ce doit être deux chaînes de caractères de même longueur. Le dictionnaire renvoyé fera correspondre pour chaque caractère de x un caractère de y pris à la même place. Si un troisième argument est fourni, ce doit être une chaîne dont chaque caractère correspondra à `None` dans le résultat.

`str.`**`partition`**(*sep*)

Divise la chaîne à la première occurrence de *sep*, et donne un *tuple* de trois éléments contenant la partie avant le séparateur, le séparateur lui-même, et la partie après le séparateur. Si le séparateur n'est pas trouvé, le *tuple* contiendra la chaîne elle-même, suivie de deux chaînes vides.

`str.`**`replace`**(*old*, *new*[, *count*])

Renvoie une copie de la chaîne dont toutes les occurrences de la sous-chaîne *old* sont remplacés par *new*. Si l'argument optionnel *count* est donné, seules les *count* premières occurrences sont remplacées.

`str.`**`rfind`**(*sub*[, *start*[, *end*]])

Donne l'indice le plus élevé dans la chaîne où la sous-chaîne *sub* se trouve, de telle sorte que *sub* soit contenue dans `s[start:end]`. Les arguments facultatifs *start* et *end* sont interprétés comme dans la notation des *slices*. Donne `-1` en cas d'échec.

`str.`**`rindex`**(*sub*[, *start*[, *end*]])

Comme `rfind()` mais lève une exception `ValueError` lorsque la sous-chaîne *sub* est introuvable.

`str.`**`rjust`**(*width*[, *fillchar*])

Renvoie la chaîne justifié à droite dans une chaîne de longueur *width*. Le rembourrage est fait en utilisant le caractère spécifié par *fillchar* (par défaut est un espace ASCII). La chaîne d'origine est renvoyée si *width* est inférieure ou égale à `len(s)`.

`str.`**`rpartition`**(*sep*)

Divise la chaîne à la dernière occurrence de *sep*, et donne un tuple de trois éléments contenant la partie avant le séparateur, le séparateur lui-même, et la partie après le séparateur. Si le séparateur n'est pas trouvé, le *tuple* contiendra deux chaînes vides, puis par la chaîne elle-même.

`str.`**`rsplit`**(*sep=None*, *maxsplit=-1*)

Renvoie une liste des mots de la chaîne, en utilisant *sep* comme séparateur. Si *maxsplit* est donné, c'est le nombre maximum de divisions qui pourront être faites, celles « à droite ». Si *sep* est pas spécifié ou est `None`, tout espace est un séparateur. En dehors du fait qu'il découpe par la droite, `rsplit()` se comporte comme `split()` qui est décrit en détail ci-dessous.

`str.`**`rstrip`**([*chars*])

Renvoie une copie de la chaîne avec des caractères finaux supprimés. L'argument *chars* est une chaîne spécifiant le jeu de caractères à supprimer. En cas d'omission ou `None`, les espaces sont supprimés. L'argument *chars* n'est pas un suffixe : toutes les combinaisons de ses valeurs sont retirées :

```
>>> '   spacious   '.rstrip()
'   spacious'
>>> 'mississippi'.rstrip('ipz')
'mississ'
```

`str.`**`split`**(*sep=None*, *maxsplit=-1*)

Renvoie une liste des mots de la chaîne, en utilisant *sep* comme séparateur de mots. Si *maxsplit* est donné, c'est le nombre maximum de divisions qui pourront être effectuées, (donnant ainsi une liste de longueur `maxsplit+1`). Si *maxsplit* n'est pas fourni, ou vaut `-1`, le nombre de découpes n'est pas limité (Toutes les découpes possibles sont faites).

Si *sep* est donné, les délimiteurs consécutifs ne sont pas regroupés et ainsi délimitent des chaînes vides (par exemple, `'1,,2'.split(',')` donne `['1', '', '2']`). L'argument *sep* peut contenir plusieurs caractères (par exemple, `'1<>2<>3'.split('<>')` renvoie `['1', '2', '3']`). Découper une chaîne vide en spécifiant *sep* donne `['']`.

Par exemple :

```
>>> '1,2,3'.split(',')
['1', '2', '3']
>>> '1,2,3'.split(',', maxsplit=1)
['1', '2,3']
>>> '1,2,,3,'.split(',')
['1', '2', '', '3', '']
```

Si *sep* n'est pas spécifié ou est `None`, un autre algorithme de découpage est appliqué : les espaces consécutifs sont considérés comme un seul séparateur, et le résultat ne contiendra

pas les chaînes vides de début ou de la fin si la chaîne est préfixée ou suffixé d'espaces. Par conséquent, diviser une chaîne vide ou une chaîne composée d'espaces avec un séparateur `None` renvoie `[]`.

Par exemple :

```
>>> '1 2 3'.split()
['1', '2', '3']
>>> '1 2 3'.split(maxsplit=1)
['1', '2 3']
>>> '   1   2   3   '.split()
['1', '2', '3']
```

`str.`**`splitlines`**([*keepends*])

Renvoie les lignes de la chaîne sous forme de liste, la découpe se fait au niveau des limites des lignes. Les sauts de ligne ne sont pas inclus dans la liste des résultats, sauf si *keepends* est donné, et est vrai.

Cette méthode découpe sur les limites de ligne suivantes. Ces limites sont un sur ensemble de universal newlines.

Représentation	Description
`\n`	Saut de ligne
`\r`	Retour chariot
`\r\n`	Retour chariot + saut de ligne
`\v` or `\x0b`	Tabulation verticale
`\f` or `\x0c`	Saut de page
`\x1c`	Séparateur de fichiers
`\x1d`	Séparateur de groupes
`\x1e`	Séparateur d'enregistrements
`\x85`	Ligne suivante (code de contrôle *C1*)
`\u2028`	Séparateur de ligne
`\u2029`	Séparateur de paragraphe

Modifié dans la version 3.2: `\v` et `\f` ajoutés à la liste des limites de lignes.

Par exemple :

```
>>> 'ab c\n\nde fg\rkl\r\n'.splitlines()
['ab c', '', 'de fg', 'kl']
>>> 'ab c\n\nde fg\rkl\r\n'.splitlines(keepends=True)
['ab c\n', '\n', 'de fg\r', 'kl\r\n']
```

Contrairement à `split()` lorsque *sep* est fourni, cette méthode renvoie une liste vide pour la chaîne vide, et un saut de ligne à la fin ne se traduit pas par une ligne supplémentaire :

```
>>> "".splitlines()
[]
>>> "One line\n".splitlines()
['One line']
```

À titre de comparaison, `split('\n')` donne :

```
>>> ''.split('\n')
['']
>>> 'Two lines\n'.split('\n')
['Two lines', '']
```

`str.`**`startswith`**(*prefix*[, *start*[, *end*]])

Donne `True` si la chaîne commence par *prefix*, sinon `False`. *prefix* peut aussi être un tuple de préfixes à rechercher. Lorsque *start* est donné, la comparaison commence à cette position, et lorsque *end* est donné, la comparaison s'arrête à celle ci.

`str.`**`strip`**([*chars*])

Donne une copie de la chaîne dont des caractères initiaux et finaux sont supprimés. L'argument *chars* est une chaîne spécifiant le jeu de caractères à supprimer. En cas d'omission ou `None`, les espaces sont supprimés. L'argument *chars* est pas un préfixe ni un suffixe, toutes les combinaisons de ses valeurs sont supprimées :

```
>>> '   spacious   '.strip()
'spacious'
>>> 'www.example.com'.strip('cmowz.')
'example'
```

Les caractères de *char* sont retirés du début et de la fin de la chaîne. Les caractères sont retirés de la gauche jusqu'à atteindre un caractère ne figurant pas dans le jeu de caractères dans *chars*. La même opération à lieu par la droite. Par exemple :

```
>>> comment_string = '#....... Section 3.2.1 Issue #32 .......'
>>> comment_string.strip('.#! ')
'Section 3.2.1 Issue #32'
```

`str.`**`swapcase`**()

Renvoie une copie de la chaîne dont les caractères majuscules sont convertis en minuscules et vice versa. Notez qu'il est pas nécessairement vrai que `s.swapcase().swapcase() == s`.

`str.`**`title`**()

Renvoie une version en initiales majuscules de la chaîne où les mots commencent par une capitale et les caractères restants sont en minuscules.

Par exemple :

```
>>> 'Hello world'.title()
'Hello World'
```

Pour l'algorithme, la notion de mot est définie simplement et indépendamment de la langue comme un groupe de lettres consécutives. La définition fonctionne dans de nombreux contextes, mais cela signifie que les apostrophes (typiquement de la forme possessive en Anglais) forment les limites de mot, ce qui n'est pas toujours le résultat souhaité :

```
>>> "they're bill's friends from the UK".title()
"They'Re Bill'S Friends From The Uk"
```

Une solution pour contourner le problème des apostrophes peut être obtenue en utilisant des expressions rationnelles :

```
>>> import re
>>> def titlecase(s):
...     return re.sub(r"[A-Za-z]+('[A-Za-z]+)?",
...                   lambda mo: mo.group(0)[0].upper() +
...                              mo.group(0)[1:].lower(),
...                   s)
...
>>> titlecase("they're bill's friends.")
"They're Bill's Friends."
```

`str.`**`translate`**(*table*)

Renvoie une copie de la chaîne dans laquelle chaque caractère a été changé selon la table de traduction donnée. La table doit être un objet qui implémente l'indexation via `__getitem__()`, typiquement un mapping ou une sequence. Pour un ordinal Unicode (un entier), la table peut donner soit un ordinal Unicode ou une chaîne pour faire correspondre un ou plusieurs caractère au caractère donné, soit `None` pour supprimer le caractère de la chaîne de renvoyée soit lever une exception `LookupError` pour ne pas changer le caractère.

Vous pouvez utiliser `str.maketrans()` pour créer une table de correspondances de caractères dans différents formats.

Voir aussi le module `codecs` pour une approche plus souple de changements de caractères par correspondance.

`str.`**`upper`**()

Return a copy of the string with all the cased characters [4] converted to uppercase. Note that `s.upper().isupper()` might be `False` if `s` contains uncased characters or if the Unicode category of the resulting character(s) is not « Lu » (Letter, uppercase), but e.g. « Lt » (Letter, titlecase).

L'algorithme de capitalisation utilisé est décrit dans la section 3.13 de la norme Unicode.

`str.`**`zfill`**(*width*)

Renvoie une copie de la chaîne remplie par la gauche du chiffre (le caractère ASCII) `'0'` pour faire une chaîne de longueur *width*. Un préfixe (`'+'` / `'-'`) est permis par l'insertion du caractère de rembourrage *après* le caractère désigne plutôt qu'avant. La chaîne d'origine est renvoyée si *width* est inférieur ou égale à `len(s)`.

Par exemple :