# Enumerate(iterable, start=0)

Das ist eine in Python eingebaute Funktion.

* Iterable: muss eine Sequenz sein. Wie eine List, tuple oder String
* Start: ist der Start der Aufzählung
* Rückgabe-Wert: Die Funktion gibt ein Tupel von (index, Wert) zurück.

Beispiel:

seasons = ['Spring', 'Summer', 'Fall', 'Winter']

**>>>** list(enumerate(seasons))  
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]

words= "Manger"

for index, iterator in enumerate (words,start=0):

    print("iterator: "+ iterator + " und index: " +str(index) )

🡪 iterator: M und index: 0

iterator: a und index: 1

Iterator: n und index: 2

Iterator: g und index: 3

iterator: e und index: 4

iterator: r und index: 5

# Funktion Definition

Es ist möglich bei der Definition anzugeben, welche Datentype die Parameter haben sowie der Rückgabewert:

Beispiel:

**def** twoSum(self, nums: List[int], target: int) -> List[int]:

# String-Methoden

word = "string"

* word.lower() 🡪 Alles kleinschreiben
* word.upper() 🡪 Alles Großschreiben
* word.title() 🡪 Erste Buchstabe Groß und der Rest klein
* word.isupper() 🡪 True, wenn alles Groß ist
* word.islower() 🡪 True, wenn alles klein ist
* word.capitalize() 🡪 Erste Buchstabe Groß und der Rest klein

# Regex-Methode (re)

Diese Methode wird verwendet, um Zeichenkette mit einem rationalen Ausdruck zu vergleichen. Die Nutzung dieser Methode erfordert die Bibliothek „re“. Verglichen können Uni-code( string) als auch Bytes. Mit dieser Bibliothek kann man überprüfen, ob eine

In einem Regex sind spezielle Zeichen nicht interpretiert. Das heißt, "\n" wird als eine Zeichenkette mit "\" und "n" .

Regeln beim Vergleich:

* Sei A und B zwei rationale Ausdrücke. A und B können verkettet werden, um einen neuen Ausdruck AB zu erstellen. In dem Fall qp stimmt mit AB überein, wenn q A überreinstimmt und p B auch.
* | und ( sind spezielle Zeichen

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Opérateurs** | **Description** | **Exemples** | | |
| **Expression régulière** | **Chaînes décrites** | **Chaînes non décrites** |
| *expr1 expr2* | Opérateur de concaténation de deux expressions (implicite). | ab | « ab » | « a », « b », chaîne vide |
| . | Un caractère et un seul | . | « a », « b », etc. | chaîne vide, « ab » |
| *expr*? | Ce quantificateur correspond à ce qui le précède, présent **zéro ou une fois**. Si de multiples correspondances existent dans un texte, il trouve d’abord ceux placés en tête du texte et retourne alors la plus grande longueur possible à partir de cette position initiale. | a? | chaîne vide, « a » | « aa », « b » |
| *expr*+ | Ce quantificateur correspond à ce qui le précède, répété **une ou plusieurs fois**. Si de multiples correspondances existent dans un texte, il trouve d’abord ceux placés en tête du texte et retourne alors la plus grande longueur possible à partir de cette position initiale. | a+ | « a », « aa », « aaaaa », etc. | chaîne vide, « b », « aaab » |
| *expr*\* | Ce quantificateur correspond à ce qui le précède, répété **zéro ou plusieurs fois**. Si de multiples correspondances existent dans un texte, il trouve d’abord ceux placés en tête du texte et retourne alors la plus grande longueur possible à partir de cette position initiale. | a\* | chaîne vide, « a », « aaa », etc. | « b », « aaab » |
| *expr1*|*expr2* | C’est l’opérateur de choix entre plusieurs alternatives, c’est-à-dire l’union ensembliste. Il peut être combiné autant de fois que nécessaire pour chacune des alternatives possibles. Il fait correspondre **l’une des expressions placées avant ou après l’opérateur**. Ces expressions peuvent éventuellement être vides, et donc (x|) équivaut à x?. | a|b | « a », « b » | chaîne vide, « ab », « c » |
| [*liste*] | Un des caractères entre crochets (« classe de caractères ») | [aeiou] | « a », « e », « i », etc. | chaîne vide, « b », « ae » |
| [^*liste*] | Un caractère n’étant pas entre crochets (« classe de caractères ») | [^aeiou] | « b », etc. | chaîne vide, « a », « bc » |
| (*expr*) | Groupement de l’expression entre parenthèses | (détecté) | « détecté » | « détect », « détecta », « détectés » |
| *expr*{*n*} | Exactement *n* occurrences de l’expression précédant les accolades | a{3} | « aaa » | « aa », « aaaa » |
| *expr*{*n*,*m*} | Entre *n* et *m* occurrences de l’expression précédant les accolades | a{2,4} | « aa », « aaa », « aaaa » | « a », « aaaaa » |
| *expr*{*n*,} | Au moins *n* occurrences de l’expression précédant les accolades | a{3,} | « aaa », « aaaa », « aaaaa », etc. | « aa » |
| ^ | Ce prédicat ne correspond à aucun caractère mais fixe une condition nécessaire permettant de trouver un accord sur ce qui le suit en indiquant que ce doit être au **début d’une ligne** (donc être au début du texte d’entrée ou après un saut de ligne). Il ne peut être considéré ainsi qu’au début de l’expression régulière, ailleurs il est considéré littéralement. Il s’applique comme condition à la totalité du reste de l’expression régulière (et concerne donc toutes les alternatives représentées). | ^a trouve « a » en début de ligne mais pas dans « ba ». | | |
| $ | Ce prédicat ne correspond à aucun caractère mais fixe une condition nécessaire permettant de trouver un accord sur ce qui le précède en indiquant que ce doit être à **la fin d’une ligne** (donc être à la fin du texte d’entrée ou juste avant un saut de ligne). Il ne peut être considéré ainsi qu’à la fin de l’expression régulière, ailleurs il est considéré littéralement. Il s’applique comme condition à la totalité du reste de l’expression régulière (et concerne donc toutes les alternatives représentées). | a$ trouve « a » en fin de ligne mais pas dans « ab ». | | |

Die Methode können hier gelesen werden: <https://docs.python.org/fr/3/library/re.html>

Über Wikipedia kann man mehr über die Geschichte der Methode lernen: <https://de.wikipedia.org/wiki/Regul%C3%A4rer_Ausdruck>

Beispiel: Die Funktion match() sucht hier

import re

#result= re.fullmatch(r"[A]\*|.[a-z]+", "A")

result= re.match(r"(?P<First>\w+)", "Ami Python ici")

print(result.group('First'))