# 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() hier gibt das erste Wort

import re

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

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

print(result.group('First'))

Beispiel 2: Die Funktion findall() sucht hier alle Wörter, die mit einem Großbuchstaben anfangen und dann kleine Buchstaben.

import re

word= re.findall(r"[A-Z].[a-z]+", " j'apprends Python maintenant a Emerson.")

print(word)

# Analyse der Code “Testslogs” von Strobl Ulrich und Fragen

Der Code ist ein Auszug eines Hauptprogramm, das die Daten von der alten Datenbank in die neue Datenbank schreibt.

1. # logfile <- loglocation, id, version, mac, computer, singletestresults
2. logfile = ''
3. rl = 126  # row length definition
4. # loglocation = r\\\\aug-tsw2\\tester1\\pc\\GENTSW-V2\\log\\ + board  (logfile location, not used yet)
5. if singletest == 0:
6. run = 'Complete Test Run'  # Testtype
7. else:
8. run = 'Single Test Run'  # Testtype
9. tswversion = str(trans[10])  # GenTSW version
10. testrunid = str(trans[0])  # GenTSW V2 ptr2 database id
11. macaddr = trans[4]  # First MAC address
12. logfile = logfile + '{:\*^{}}'.format('', rl) + '\n'
13. logfile = logfile + '\* ' + '{:<{}}' \
14. .format(board + ' Test Software: ' + testtyp, int(rl / 3 - 2)) + '{:^{}}'.format(run, int(rl / 3)) + '{:>{}}' \
15. .format('GenTSW V2 v' + tswversion, int(rl / 3 - 2)) + ' \*\n'
16. logfile = logfile + '\* ' + '{:^{}}'.format('(C)2019 by ICC Intelligent Platforms GmbH, Augsburg', rl - 4) + ' \*\n'
17. logfile = logfile + '\* ' + '{:<{}}'.format('Instance: ' + instance, int(rl / 4 - 6)) + '{:^{}}'.format('Tester: ' + userid, int(rl / 4 + 4)) + '{:^{}}' \
18. .format('Computer: ' + computer + '/' + gesite, int(rl / 4 + 4)) + '{:>{}}'.format('Id: ' + testrunid, int(rl / 4 - 4)) + ' \*\n'
19. logfile = logfile + '{:\*^{}}'.format('', rl) + '\n\n'
20. logfile = logfile + '\*\* Board ID: ' + str(boardid) + '\n\*\* Serial No: ' + str(serno) + '\n\*\* Board Version: ' + str(boardver) + '\n\*\* MAC Address: ' + str(macaddr) + '\n'

Die ersten 20 Zeilen sind Textformatierten Ausgabe. Die Länge einer Zeile für Logfile auf rl=126 definiert.

Zeile 21: Instanziierung der der klasse cursor()

cursor = db\_list[b]['connect'].cursor()

Was ist diese Variable db\_list? Wie ist sie definiert?

Zeile 22-29:

try:

    sql\_log("INSERT INTO tswlog (serno,handleref,testtyp,req,timestamp,board,boardver,boardid,wproject,userid,instance,computer,gesite,singletest,logfile,testresult,testcomment,loops,aborted,rma,tablever) VALUES ('{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}','{}')"

.format(serno, handle, testtyp, req, timestamp, board, boardver, boardid, wproject, userid, instance, computer, gesite, singletest,logfile, testresult, testcomment, loops, aborted, rma, tablever), cursor)  # MySQL

except mariadb.Error as error:

    database\_close\_error(lineno(), error)

Hier wird Daten in der Tabelle „tswlog“ hinzugefügt. Der String „logfile“ wird auch hinzugefügt.

Zeile 31:

Was macht die Funktion debug\_log() und sql\_log. Sowie die Variablen trans?

Zeile 34 – 39:

debug\_log("\* results Eintraege und logfile erzeugen ...")

cursor = db\_list[a]['connect'].cursor()

try:

    sql\_log("SELECT \* FROM {} WHERE testrunid={}".format(db\_table[7], trans[0]), cursor)  # MySQL

except mariadb.Error as error:

    database\_close\_error(lineno(), error)

singleresult = cursor.fetchall()

Welcher Wert hat db\_table[7] und trans[0]

Eine Abfrage wird gemacht in der Variable singleresult gespeichert

Für den Rest gibt es sehr viele unbekannte Variablen. Dies erschwert das Verständnis. Außerdem muss die Logik hinter der Code erklärt werden.

# Die Bibliothek mysql.connector kennenlernen

### Abfragen von Daten mit mysql.connector

<https://dev.mysql.com/doc/connector-python/en/connector-python-example-cursor-select.html>

1. Verbindung aufbauen und das Verbindungsobjekt in einer Variable speichern
2. Die Methode cursor() von dem Verbindungsobjekt aufrufen und das Ergebnis in einer neuen Variablen speichern.

Was ist cursor()?

<https://dev.mysql.com/doc/connector-python/en/connector-python-api-mysqlcursor.html>

Die klasse MYSQLCursor ist eine Klasse, mit der SQL-Anweisung ausgeführt werden können. Cursor-Objekte interagieren über ein MySQL-Connection-Objekt mit dem MySQL-Server.

import mysql.connector

cnx = mysql.connector.connect(database='world')

cursor = cnx.cursor()

Vorher muss das Paket mysql.connector installiert werden: *python3 -m pip install mysql-connector*

### Methoden der Klasse MYSQLCursor

* result\_args = cursor.callproc(proc\_name, args=()) Ruft eine gespeicherte Prozedur auf.

Die args Parametersequenz muss einen Eintrag für jedes Argument enthalten, das die Prozedur erwartet. callproc()gibt eine modifizierte Kopie der Eingabesequenz zurück.

Beispiel:

CREATE PROCEDURE multiply(IN pFac1 INT, IN pFac2 INT, OUT pProd INT)

BEGIN

SET pProd := pFac1 \* pFac2;

END;

args = (5, 6, 0) # 0 is to hold value of the OUT parameter pProd

cursor.callproc('multiply', args)

🡪 ('5', '6', 30L)

* cursor.close() Wird verwendet, wenn man mit der Klasse fertig ist.
* iterator = cursor.execute(operation, params=None, multi=True)

Dieser Befehl führt die Operation ( Anweisung oder Befehl) in der Datenbank aus. Dabei können Parameter übergeben werden. Is der Parameter „multi“ auf „True“ gesetzt, können mehrere Abfrage ausgeführt werden.

Beispiel:

insert\_stmt = (

"INSERT INTO employees (emp\_no, first\_name, last\_name, hire\_date) "

"VALUES (%s, %s, %s, %s)"

)

data = (2, 'Jane', 'Doe', datetime.date(2012, 3, 23))

cursor.execute(insert\_stmt, data)

select\_stmt = "SELECT \* FROM employees WHERE emp\_no = %(emp\_no)s"

cursor.execute(select\_stmt, { 'emp\_no': 2 })

**Mit „multi“ auf True**

operation = 'SELECT 1; INSERT INTO t1 VALUES (); SELECT 2'

for result in cursor.execute(operation, multi=True):

if result.with\_rows:

print("Rows produced by statement '{}':".format(

result.statement))

print(result.fetchall())

else:

print("Number of rows affected by statement '{}': {}".format(

result.statement, result.rowcount))

# Die Funktion zip

strs= ["flower","flow","flight"]

for c in zip(\*strs):

    print(c)

🡪 ('f', 'f', 'f')

('l', 'l', 'l')

('o', 'o', 'i')

('w', 'w', 'g')

# Die Funktion sort()

strs= ["flower","flow","flight"]

strs.sort()

print(strs) 🡪['flight', 'flow', 'flower']

Hier vergleicht die Funktion sort() die Elemente miteinander. Somit unterscheidet sich das erste Element von dem letzten Element stark.