

Python Sequenzen (str, tuple, list)

(C) 2019-2021 T.Birnthaler OSTC GmbH

=====

Doku --> <http://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range>

Sequenzen sind eine "geordnete" Folge von Elementen, jedem Element wird ein INDEX zugeordnet (Ganzzahl beginnend mit 0), über den es angesprochen werden kann. Der Index läuft AUFSTIEGEND von 0 bis Sequenz-Länge-1 bzw. ABSTIEGEND von -1 bis -Sequenz-Länge:

```
s1 = "hallo"          s2 = ""          # String + leerer String
t1 = ( True, "hallo", 3.14 )  t2 = ()          # Tupel + leerer Tupel
l1 = [ 123, 456, 789 ]      l2 = []          # Liste + leere Liste
```

```
01234 # Positiver Index (0 .. Länge-1)
"hallo" # Sequenz (str)
  -1 # Negativer Index (-1 .. -Länge) = Abkürzung für len(s) - 1
  -2 #                               len(s) - 2
  -3 #                               len(s) - 3
  -4 #                               len(s) - 4
  -5 #                               len(s) - 5
```

Die Zugriffsgeschwindigkeit auf ein bestimmtes ELEMENT über seinen INDEX ist unabhängig von der Sequenz-Länge immer gleich schnell (RANDOM ACCESS).

Die Zugriffsgeschwindigkeit auf eine bestimmtes ELEMENT über seinen WERT (z.B. welchen Index hat ein bestimmter Wert oder wie häufig kommt ein WERT in der Sequenz vor) ist abhängig von der Sequenz-Länge und immer langsam (erfordert das Durchsuchen der gesamten Sequenz)

Es gelten folgende Beschränkungen:

Datentyp		Elemente	Elementanzahl
str	immutable	UTF-8 codiertes Zeichen	fest
tuple	immutable	Python-Objekt	fest
list	mutable	Python-Objekt	variable
bytes	immutable	Byte 00-FF	fest
bytearray	mutable	Byte 00-FF	variabel
array	mutable	Zahlen gleichen Typs	fest

Folgende Operationen werden von (fast) allen Sequenz-Typen (str, tuple, list, bytes, bytearray) unterstützt (sowohl mutable als auch immutable). S und T sind Sequenzen des gleichen Typs, I, J, K, N sind Ganzzahlen und E ist ein beliebiges Objekt, das die Beschränkungen der Sequenz S erfüllt.

ACHTUNG: Die Sequenz S bleibt unverändert!

Operation	Bedeutung
E in S	Element E in S vorhanden?
E not in S	Element E in S nicht vorhanden?
S + T	Verkettung von S und T
S * N    N * S	Sequenz N-mal verketteten
S[I]	Element mit Index I (Start bei 0)
S[I:J]	Slice von I bis J-1 (mit Schrittweite 1)
S[I:J:K]	Slice von I bis J-1 mit Schrittweite K
len(S)	Länge von S (Anzahl Elemente)
min(S)	Kleinstes Element in S (gemäß Vergleich "<")
max(S)	Größtes Element in S (gemäß Vergleich "<")
sum(S)	Summe der Elemente in S (müssen Zahlen sein)
all(S)	True falls alle Elemente True (oder S leer), sonst False
any(S)	False falls alle Elemente False (oder S leer), s. True
S.index(E[,I[,J]])	Index des 1. Elements gleich E in S (von Index I bis J-1; Standard: 0 bis len(S)-1)
S.count(E)	Anzahl Vorkommen von Element E in S
reversed(S)	Iterator mit Elementen in umgekehrter Reihenfolge
sorted(S, key, reverse)	Sortierte Liste der Elemente (gemäß Vergleich "<") (key=MAPFUNC, reverse=True --> absteigend)

HINWEIS: Indexwerte I und J sowie Schrittweite K dürfen auch negativ sein (auch gemischt).

HINWEIS: Die Formulierung "gemäss ^\_ Vergleich <" bedingt, dass ALLE Elemente per Operator "<" miteinander vergleichbar sind. D.h. entweder alle Elemente sind Zahlen (bool, int, float, complex, Decimal, Fraction) oder alle Elemente haben den gleichen Datentyp str, tuple, list, ...

HINWEIS: Der Indexzugriff wird überwacht, d.h. bei Zugriff über den Rand hinaus erfolgt eine Exception "IndexError". Indexzugriffe und Slicing erzeugen eine Kopie der Sequenz und dürfen aus einer beliebigen Kombination von positiven und negativen Indices bestehen. Der Startindex 0, der Endindex len(S)+1 und die Schrittweite 1 dürfen weggelassen werden.

```

S[:]           = 1:1-Kopie der Sequenz
S[:]          = 1:1-Kopie der Sequenz
S[::1]        = 1:1-Kopie der Sequenz
S[0:]         = 1:1-Kopie der Sequenz
S[:len(S)+1] = 1:1-Kopie der Sequenz
S[0:len(S)+1] = 1:1-Kopie der Sequenz
S[0:len(S)+1:] = 1:1-Kopie der Sequenz
S[:2]         = Jedes 2. Element ab 1. Element (Index 0,2,4,...)
S[1::2]       = Jedes 2. Element ab 2. Element (Index 1,3,5,...)
S[5:]         = Ohne erste 5 Elemente (Index 5..*)
S[:5]         = Erste 5 Elemente (Index 0..4)
S[3:5]        = Element 4 und 5 (Index 3+4)
S[1:]         = Erstes Element weglassen (Index 0)
S[:-1]        = Letztes Element weglassen (Index -1)
S[1:-1]       = Erstes + letztes Element weglassen (Index 0,-1)
S[::-1]       = Alle Elemente in umgekehrter Reihenfolge
S[0:5:-1]     = Erste 5 Elemente in umgek. Reihenfolge (Index 0..4)
S[5:-1]       = Erste 5 Elemente in umgek. Reihenfolge (Index 0..4)
S[1:-1:-1]    = Ohne Erstes + letztes Element in umgek. Reihenfolge
S[-5:]        = Letzte 5 Elemente (Index -5..-1)
S[-5::-1]     = Letzte 5 Elemente in umgek. Reihenfolge (Index -5..-1)

```

#### Immutable Sequenzen

Folgende Operation wird von Immutable Sequenzen (str, tuple, bytes) zusätzlich zu obigen unterstützt (S ist eine immutable Sequenz):

ACHTUNG: Die Sequenz S bleibt unverändert!

Operation	Bedeutung
hash(S)	Hashwert erstellen (eindeutig)

#### Mutable Sequenzen

Folgende Operationen werden von mutable Sequenzen (list, bytearray) zusätzlich zu den obigen Sequenz-Operationen unterstützt (S ist eine mutable Sequenz, T ist eine Sequenz oder ein Iterator des gleichen Typs, I, J, K, N sind Ganzzahlen und E ist ein beliebiges Objekt, das die Beschränkungen der Sequenz S erfüllt):

ACHTUNG: Sequenz S wird verändert (bleibt aber das gleiche Objekt)!

ACHTUNG: Operationen += \*= sind auf String + Tupel anwendbar, erzeugen aber NEUEN String / Tupel!

Operation	Bedeutung
S[I] = E S[I:J] = T del s[I:J] S[I:J:K] = T del s[I:J:K]	Element an Position I durch E ersetzen Elemente von I..J-1 durch Inhalt von Iterable T ersetzt. Elemente von I..J-1 aus Liste entfernen Elemente s[I:J:K] durch Inhalt von Iterable T ersetzen Elemente s[I:J:K] aus Liste entfernen
S.append(E) S.extend(T) s += T S.insert(I, E)	E am Sequenz-Ende anhängen: S[len(S):len(S)] = [E] Elemente von T anhängen: S[len(S):len(S)] = T Element E an Position I einfügen: s[I:I] = [E]
E = S.pop([I]) S.remove(E)	Element an Position I (STD: 0) entfernen und zurück Erstes Element mit Wert E entfernen (Suche!)
S *= N S.reverse() S.clear() S2 = S.copy()	Sequenz N-mal vervielfachen + verketteten Reihenfolge der Elemente umdrehen Sequenz leeren (alle Elemente entfernen) del S[:] Shallow/Flache Kopie erzeugen: S[:]

```
+-----+
HINWEIS: Indexwerte I und J sowie Schrittweite K dürfen auch negativ sein
(auch gemischt).
```

```
list-Sequenz
```

```
-----
Folgende Operation wird von Listen zusätzlich zu den allgemeinen
Sequenz-Operationen und den Operationen für mutable Sequenzen unterstützt:
```

```
ACHTUNG: Die Liste L wird verändert (bleibt aber das gleiche Objekt)!
```

Operation	Bedeutung
L.sort(key,reverse)	Elemente aufsteigend sortieren (gemäM-^_ Vergleich "<") (key=MAPFUNC, reverse=True --> absteigend)

```
HINWEIS: Die Formulierung "gemäM-^_ Vergleich <" bedingt, dass ALLE Elemente per
Operator "<" miteinander vergleichbar sind. D.h. entweder alle Elemente sind
Zahlen (bool, int, float, complex, Decimal, Fraction) oder alle Elemente haben
den gleichen Datentyp str, tuple, list, ...
```