Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

FS24 CAS PML - Python

Niklaus Johner

niklausbernhard.johner@bfh.ch

FS24 CAS PML - Python

3. Arrays: strings, lists and tuples

Arrays Basics

- *str*-Objekte sind Zeichenketten

```
klass = str()
literal = "123"
```

- *list*-Objekte sind veränderliche (mutable) Sequenzen von Objekten

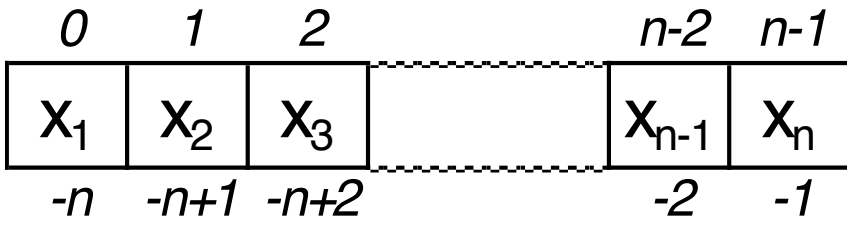
```
klass = list()
literal = [1, 2, 3]
```

- *tuple*-Objekte sind unveränderliche Sequenzen von Objekten

```
klass = tuple()
literal = (1, 2, 3)
```

Indexierung und Slicing

seq mit *n* Elementen



- Mit Indizien kann man Elemente ergreifen

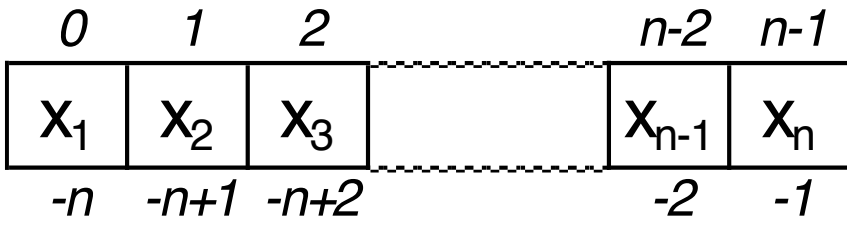
```
seq[0] # Erstes Element
seq[n-1] # Letztes Element
```

- Mit negativen Indizien beginnt man am Ende

```
seq[-1] # Letztes Element
seq[-n] # Erstes Element
```

Indexierung und Slicing

seq mit *n* Elementen



- Slices sind Subsets einer Sequenz
- Man erhält sie mit der *[von:bis]* Syntax

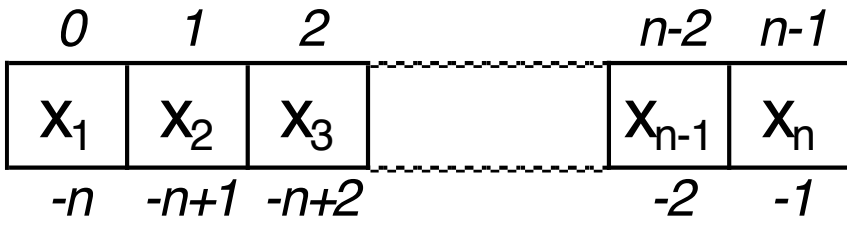
```
seq[0:3]  # first to third
seq[3:]   # fourth to the end
seq[:3]   # first to third
```

- Da kann man auch negative Indizien verwenden

```
seq[-2:]   # last two
seq[:-2]   # all except last two
```

Indexierung und Slicing

seq mit *n* Elementen



- Man kann noch einen *Schritt angeben [von:bis:schritt]*

```
seq[::3]      # every third element
seq[1:10:2]   # every other from 2nd to 10th
```

- Wenn man den Schritt nicht angibt, nimmt mann jedes Element.

```
seq[::]   # all elements in sequence
```

Zuweisungen

- Man kann auch slices einer Variablen zuweisen

```
In [1]: my_list = [1, 2, 3]
   ...: first_element = my_list[0]
   ...: pair = my_list[:2]
   ...: print(first_element, pair)
   ...:
1 [1, 2]
```

- Oder jedes Element einer verschiedenen Variable zuweisen

```
In [2]: first, second = my_list[:2]
   ...: print(first, second)
   ...:
1 2
```

Addition Operation auf Arrays

Addition entspricht Verkettung (concatenation)

```
In [1]: s1 = "This is"
   ...: s2 = " nice!"
   ...: s3 = s1 + s2
   ...: print(s3)
   ...:
This is nice!
```

```
In [2]: l1 = [0, 0, 0]
   ...: l2 = ["a", "b"]
   ...: l3 = l1 + l2
   ...: print(l3)
   ...:
[0, 0, 0, 'a', 'b']
```

Multiplikation Operation auf Arrays

Ganzzahl Multiplikation entspricht Wiederholung und Verkettung

```
In [5]: s1 = "nice! "
   ...: s2 = 3 * s1
   ...: print(s2)
   ...:
nice! nice! nice!
```

```
In [6]: l1 = [1, 2]
   ...: l2 = l1 * 2
   ...: print(l2)
   ...:
[1, 2, 1, 2]
```

Andere mathematische Operatoren

- Andere mathematische Operatoren sind für arrays nicht definiert:
  - Subtraktion
  - Division
  - Multiplikation bei *float*
  - Potenz
  - Modulo

Nützliche Funktionen für arrays

- *len(array)* : Anzahl Elemente in *array*

```
In [1]: st = "abcd"
   ...: lt = [1, 2, 3]
   ...: ns = len(st)
   ...: nl = len(lt)
   ...: print(ns, nl)
   ...:
4 3
```

- *min(array)* : minimales Element von *array*

```
In [2]: print(min(lt))
1
```

- *max(array)* : maximales Element von *array*

```
In [3]: print(max(lt))
3
```

Nützliche array Methoden

- *array.index* : gibt den Index eines Elements in array

```
In [1]: lt = [1, "a", 2, 1]
   ...: print(lt.index("a"))
   ...: print(lt.index(1))
   ...:
1
0
```

- *array.count* : zählt wie oft ein bestimmtes Objekt in *array* ist

```
In [2]: print(lt.count("a"))
   ...: print(lt.count(1))
   ...:
1
2
```

Der *in* Operator

- Der *in* Operator tested ob ein Objekt in einer Sequenz vorkommt

```
In [1]: 1 in [0, 1, 2]
Out[1]: True

In [2]: 3 in [0, 1, 2]
Out[2]: False
```

Reminder

- slicing: *seq[von:bis:schritt]*
- Get the position of an element in a sequence: *seq.index*
- Get the number of times an element is in a sequence: *seq.count*
- Length of a sequence: *len(seq)*
- min and max values: *min(seq)*; *max(seq)*
- Check whether in sequence: *obj in seq*
- Operations:
  - addition = concatenation
  - integer multiplication = concatenation