Sequences

- lists and tuples are containers for more than one element: sequences
  - example: `employee = [Markus', Dickinson', 'assistant prof', 'MM317']`
  - another example for a sequence is a string

Indexing

- accessing elements in a list is called indexing
  - example: `greeting = 'hi there' greeting[3]`

Sequences

- lists and tuples are containers for more than one element: sequences
  - example: `employee = [Markus', Dickinson', 'assistant prof', 'MM317']`
  - another example for a sequence is a string

Sequences

- lists and tuples are containers for more than one element: sequences
  - example: `employee = [Markus', Dickinson', 'assistant prof', 'MM317']`
  - another example for a sequence is a string
    - each element in the sequence is assigned a position number, an index (starting from 0)
      - example: `employee[1]`
Indexing

- accessing elements in a list is called indexing
  - example:
    ```python
greeting = 'hi there'
greeting[3]
'hi there'[3]
```
- indexing from the end: `greeting[-2]`
- adding sequences:
  ```python
  long_greeting = greeting + ' how are you'
  ```
- add multiple copies to a sequence: multiply
  ```python
  comment = 'this is ' + 3 * 'very ' + 'good'
  ```

Slicing

- accessing parts of segments is called slicing
  - example:
    ```python
    long_greeting[3:6]
    ```
    - the slice starts at the first index and goes up to the second (non-inclusive)!
  - count from the end:
    ```python
    long_greeting[-5:-1]
    ```
  - going all the way to the end:
    ```python
    long_greeting[4:]
    ```
  - starting at the beginning:
    ```python
    long_greeting[:6]
    ```
  - steps are given as optional third number:
    ```python
    long_greeting[1:6:2]
    ```

Caution!

- Initialization
  Always initialize your variables! Otherwise you may end up with a random value.

- Lists are Mutable
  If you perform an operation on a list, it changes the list. In contrast, tuples and strings are immutable.
Operations on Sequences

- membership:
  \[ \text{employee} = ['Markus, Dickinson', 'assistant prof', 'MM317'] \]
  'MM317' in employee

- check length:
  \[ \text{len} \text{(employee)} \]

- minimum / 'smallest' element:
  \[ \text{nums} = [5, 102, 13, 2, 99, 154, 7] \]
  \[ \text{min(nums)} \]
  \[ \text{min(employee)} \]

- maximum:
  \[ \text{max(nums)} \]

Queues and Stacks

- FIFO and LIFO
  - LIFO Last in, first out (stack)
  - FIFO First in, first out (queue)

Queue and Stack Operations

- add at the end: append
  \[ \text{employee}.append('Computational Linguistics') \]

- retrieve from the end: pop
  \[ \text{employee}.pop() \]
  This returns a value!

- add at the beginning:
  \[ \text{employee}.insert(0, 'Linguistics') \]

- retrieve from the beginning:
  \[ \text{employee}.pop(0) \]

Sorting

- sort destructively
  \[ \text{nums}.sort() \]
  Caution: this does not return a value but modifies the list itself!
  \[ \text{wrong: nums_sorted = nums.sort()} \]

- non-destructive version:
  \[ \text{nums_sorted = sorted(nums)} \]
More List Methods

- Count how often the same element is in a list: count
  `['a', 'rose', 'is', 'a', 'rose', 'is', 'a', 'rose'].count('rose')`

- Add a list destructively: extend
  `employee.extend(['a', 'rose', 'is', 'a', 'rose', 'is', 'a', 'rose'])`

- Insert revisited, you can insert anywhere ...
  `employee.insert(6, 'Linguistics')`

- Find the first occurrence of an element in the list: index
  `['a', 'rose', 'is', 'a', 'rose', 'is', 'a', 'rose'].index('rose')`

Tuples

**Definition**

Tuples are very similar to lists but are immutable. So once you create them, that’s it!

- Indexing and slicing work with tuples just as with lists.
- Tuples do not support methods such as sorting.
- You can create them with parentheses:
  `mytuple=(10, 50, 'foo')`
- Why bother? Later more ;)

Tuples