KTI

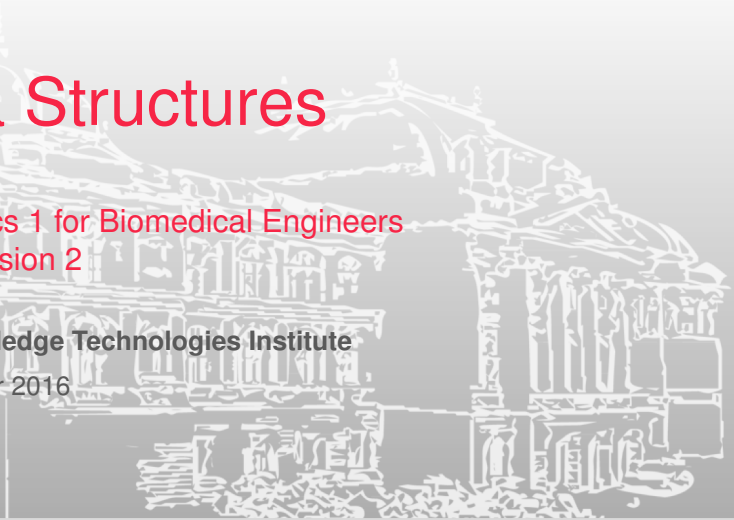
TU Graz

Data Structures

Informatics 1 for Biomedical Engineers
Tutor Session 2

KTI, Knowledge Technologies Institute

17. Oktober 2016



Today's Topics

1. What are data structures?
2. Python lists
    - Indexing, slicing, manipulating
3. Python tuples
4. Python sets
5. Python dictionaries
6. Combining data structures

Student Goals

- Know the differences between Python data structures
- Get a feeling for when to use which structure
- Know the difference between mutable and immutable structures
- Use the Python data structures in your own programs

What are data structures?

- Containers for data
- Store information in certain forms
- Access and manipulate stored data

Python lists

- List aka array, vector, etc.
- Python: more than just array
  - Sequence type - iteration (remember loops, etc.)
  - In-place manipulation
  - Mutable
  - Usability
- Lists start at position 0, not 1! (index out of range)
- Indexing: offset from the beginning (0 + index)

Lists: examples and methods 0

- Implicit declaration: square brackets
- Explicit declaration: list()
- A list can have elements of mixed type

```
1  # Initializing a list
2  some_list = [60, -2, 89, 10]
3
4  some_list[0]
5  # output: 60
6
7  mixed_list = ['foo', 18, 1.3, 'elem']
8  mixed_list[3]
```

Lists: examples and methods 1

- Length of a list
- Changing list elements

```
1  # How many elements are in a list?
2  len(some_list)
3  # output: 4
4
5  # Changing an element of a list (item assignment)
6  mixed_list = ['foo', 18, 1.3, 'elem']
7  mixed_list[3] = 'end␣of␣list'
8  mixed_list
9  # output: ['foo', 18, 1.3, 'end of list']
```

Lists: examples and methods 1.1

- List slicing
- Use indices
- Half-open interval: [ )

```
1  # List slicing
2  index_list = [0, 1, 2, 3, 4, 5]
3  index_list[0:3]
4  # What elements will be displayed?
5  # Has the original list changed?
```

Lists: examples and methods 1.2

- Copying lists

```
1  first_list = [1, 2, 3, 4, 5]
2  second_list = ['Good', 'morning']
3  # Two 'names' for one list
4  second_list = first_list
5  # Getting a copy of the list
6  third_list = second_list[:]
```

Lists: examples and methods 2

- Sorting a list (in place): list.sort()

```
1  # List method: in-place sorting
2  some_list.sort()
3  some_list
4  # output: [-2, 10, 60, 89]
```

- How does sorting our mixed_list work?

Lists: examples and methods 2.1

- In-place methods vs returning a new object
  - list.sort() vs. sorted(list)

```
1  # Old list vs. new list
2  some_list.sort()
3  some_list
4  # output: [-2, 10, 60, 89]
5  another_list = [4, 95, 33, 6, 8]
6  sorted(another_list) # returns a new object
7  another_list
```

Lists: examples and methods 3

- Adding an element to a list
- list.append(<element>)

```
1  # List method: appending an element
2  new_list = [45, 3, 99]
3  new_list.append(101)
4  new_list
5  # output: [45, 3, 99, 101]
```

Lists: examples and methods 4

- Reversing the elements of a list (in place)
- list.reverse()

```
1  # List method: reversing list elements
2  another_list = ['a', 'b', 'c', 'd']
3  another_list.reverse()
4  another_list
5  # output: ['d', 'c', 'b', 'a']
```

Lists: examples and methods 4.1

- Iterating over a list in reversed order
- reversed()

```
1  # List method: reversing list elements
2  another_list = ['a', 'b', 'c', 'd']
3  for elem in reversed(another_list):
4          print(elem)
5  another_list
```

Lists: examples and methods 5

- Iterating over lists

```
1   # Print all elements of a list
2   another_list = ['zero', 'one', 'two', 'three', 'four']
3   for element in another_list:
4       print(element)
5
6   # output:
7   # zero
8   # one
9   # two
10  # three
11  # four
```

Lists: examples and methods 5.1

- Iterating over list indices

```
1  # Print all elements and indices of a list
2  #another_list = ['zero', 'one', 'two', 'three', 'four']
3  for index in range(len(another_list)):
4      print(index, another_list[index])
```

- Also see: enumerate() [1]

[1] https://docs.python.org/3/library/functions.html#enumerate

More list methods

- extend
- insert
- pop
- clear
- count
- ... [2]

[2] For more list methods see Python documentation:
`https://docs.python.org/3/tutorial/datastructures.html`

Python tuples

- Immutable sequence type (iterable, but you cannot change single elements) - item assignment not supported
- ... but they can contain mutable objects (like lists)

Tuples: examples 1

```
1   # Declaring a tuple
2   newspapers = 'Die Presse', 'Der Standard', 'FAZ'
3   some_words_and_numbers = ('hi', 'bye', 15) # Preferred!
4
5   # Special tuples
6   one_element_tuple = ('singleton', )
7   empty_tuple = ()
8
9   # Accessing tuple elements via index:
10  newspapers[2]
11  # output: 'FAZ'
```

Tuples: examples 2

- Value unpacking and multiple assignment
- Practical for functions (return values)

```
1  # Value unpacking / multiple assingment
2  # newspapers = ('Die Presse', 'Der Standard', 'FAZ')
3  np1, np2, np3 = newspapers
4  print(np1, np2, np3)
5
6  # output: Die Presse Der Standard FAZ
```

Python sets

- Mathematical sets
- Unordered collection with no duplicate elements
- Support mathematical operations, e.g. union, intersect, difference
- Sets with complex elements (lists, etc.): modification/extension necessary (not easy!)
- Elements have to be unique and easily comparable (hashable)

Sets: examples 1

```
1   # Declaring a set implicitly: curly brackets {}
2   backpack = {'notebook', 'phone', 'key', 'gum', 'pen'}
3   backpack
4   # output: {'gum', 'key', 'notebook', 'pen', 'phone'}
5
6   # Other way: function set()
7   other_bag = set(['key', 'ipad', 'bottle'])
8   some_letters = set('mimimifoofoo')
9
10  # Empty set - explicit declaration: set(), not {}!
11  empty_set = set()
```

Sets: examples 2

- Sets with mixed elements: not recommended

```
1  # Set with mixed elements
2  my_set = {1,2,3,3,5,6,4,3,3,3,'hallo'}
3
4  # What will happen here?
5  sorted(my_set)
6
7  # And here?
8  test_set = {1, 2, 3, 4, 1, 2, 3, 4, [1,2,1,3,2,1]}
```

Sets: examples 3

- Set operations

```
1  # Sets support mathematical operations
2  backpack = {'bottle', 'phone', 'keys', 'notebook'}
3  bag = {'phone', 'keys', 'snack', 'tablet', 'gum'}
4
5  # Items in both bags - intersection
6  backpack & bag
7
8  # Items in one or the other but not both - difference
9  backpack ^ bag
```

Python dictionaries

- Aka hash map, associative array
- Collection of key: value pairs
- Dictionary is indexed by keys, not numbers
- Keys
  - ...must be unique and of an immutable data type
  - ...can be strings or numbers (must be hashable)
  - ...are a set: sorted by hash values, do not stick to user-defined item order

Dictionaries: examples 1

```
1   # Declaring a dictionary
2   contact_info = {'name': 'someone', 'phone': 12345,
3   'city': 'Graz'}
4
5   # Accessing dictionary entries
6   contact_info.keys()
7   # output: dict_keys(['phone', 'name', 'city'])
8   contact_info.values()
9   # output: dict_values([12345, 'someone', 'Graz'])
10  contact_info.items()
11  # output: dict_items([('phone', 12345), ('name', 'someone'),
12  ('city', 'Graz')])
```

Dictionaries: examples 2

```
1   # Working with dictionaries
2   backpack = {'notebook': 1, 'phone': 1, 'key': 1,
3                'gum': 4, 'pen': 2, 'bottle': 1}
4
5   # Changing values using a key
6   backpack['pen'] = 3
7
8   # Emptying our backpack - iterating over dictionary
9   for key in backpack.keys():
10      backpack[key] = 0
11
12  # Checking our bag:
13  for key, value in backpack.items():
14          print(key, value)
```

Combining data structures

- List of lists aka matrix
- Tuple of lists
- List of dictionaries
- Dictionary with list as values
- and so on

Combining data structures

- List of lists aka matrix
- Tuple of lists
- List of dictionaries
- Dictionary with list as values
- and so on

```
1   # Data structure:
2   sensor_data =
3   {0: {"name": "i", "data": []},
4    1: {"name": "ii", "data": []},
5    2: {"name": "iii", "data": []},
6    3: {"name": "avr", "data": []},
7    4: {"name": "avl", "data": []},
8    5: {"name": "avf", "data": []},
9    6: {"name": "v1", "data": []},
10   7: {"name": "v2", "data": []},
11   8: {"name": "v3", "data": []},
12   9: {"name": "v4", "data": []},
13   10: {"name": "v5", "data": []},
14   11: {"name": "v6", "data": []}
15  }
```

Combining data structures: examples 1

- List of lists (of lists...) = matrix

```
1  # Simple matrix = list of lists
2  alphabet = [['a', 'b', 'c'],
3              ['d', 'e', 'f'], 'u',
4              ['x', 'y', 'z']]
5
6  # Accessing matrix elements
7  alphabet[0]
8  alphabet[1][1]
```

Combining data structures: examples 2

- List of dictionaries

```
1   # Declaring a list with dictionaries as elements
2   list_of_grades = [
3       {'name': 'hubert', 'grade': 3},
4       {'name': 'mauke', 'grade': 1},
5       {'name': 'student3', 'grade': 5}
6   ]
7
8   # Iterating over the entries
9   for each_entry in list_of_grades:
10      for key, value in each_entry.items()
11          print(key, '.....', value)
```

Complex example

Task: Playing tic tac toe

- Create a matrix with 3 columns and rows
- Fill matrix with '_'
- Fill the matrix with 3 'X' or 'O' (horizontally, vertically or diagonally) to win

Complex example: sample solution

```
1   # Complex example - tic tac toe
2   game_board = []
3   for some_index in range(3):
4       game_board.append(['O'] * 3)
5
6   # Win using the diagonal:
7   for row in range(len(game_board)):
8       for index in range(len(game_board)):
9           if row == index:
10              game_board[row][index] = 'X'
11
12  for row in game_board:
13      print(row)
```

Student Task

Task: Compare DNA sequences

- Input: 2 lists with simplified DNA information (A, G, C, T)
- Compare the lists and return a list with the differences
  - Compare single elements of each list
  - If elements are the same, save a ’0’ in the result list
  - If elements differ, save the element from the first and second list at that place in the result list
  - Print the list with the saved differences

Student task: sample solution

```
1  # Student Task: sample solution
2  index = 0
3  for index in range(len(main_seq)):
4      if main_seq[index] == compare_seq[index]:
5          differences.append(0)
6      elif main_seq[index] != compare_seq[index]:
7          differences.append(main_seq[index]
8                             + compare_seq[index])
```