Informatik 1 - Biomedical Engineering

Tutor Session 3 - Data Structures

Overview

- What are data structures?
- Lists
- Tuples
- Sets
- Dictionaries
- Combination of data structures

1) What are data sturctures?

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

2) 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)

2.1) List: examples

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

In [ ]:

```
# Initializing a list
some_list = [-1, 0, 1, 2, 3, 4, 5]
```

In [ ]:

```
print(some_list[0])
print(some_list[0:3])
print(some_list[3:])
print(some_list[::2])

print(some_list[3:1:-1]) #What happens?

# print(some_list[7])   # <-- off by one error!
```

In [ ]:

```
mixed_list = [12, "twelve", 12.0]
print(mixed_list)
```

2.2) List: methonds

Most useful methods for lists:

- append
- sort
- reverse
- len

In [ ]:

```
print(some_list, "length:", len(some_list))
some_list.append(2.5)
print(some_list, "length:", len(some_list))
```

In [ ]:

```
some_list.sort()
print("Sorted:", some_list)
some_list.reverse()
print("Reversed:", some_list)
```

.sort() and .reverse() work in-place!

In [ ]:

```
another_list = [2, 6, 4, 3, 5]
sorted_list = sorted(another_list) # creates a copy
print(another_list)
print(sorted_list)
```

More list methods:

- extend
- insert
- pop
- clear
- count

In [ ]:

```
list_1 = [1,2,3,4]
list_2 = [5,6,7]
list_1.append(list_2)
print(list_1)
```

In [ ]:

```
list_1 = [1,2,3,4]
list_2 = [5,6,7]
list_1.extend(list_2)
print(list_1)
```

In [ ]:

```
list_3 = list_1 + [8, 9]
print(list_3)
```

In [ ]:

```
list_3.insert(4,4.5)    # list.insert(position, value)
print(list_3)
```

In [ ]:

```
list_3.pop()
print(list_3)
```

In [ ]:

```
list_3.count(6)
```

In [ ]:

```
list_3.clear()
print(list_3)
```

2.3) Working with Lists

In [ ]:

```
some_list = ["Graz", "München", "Wien", "Salzburg", "Linz"]
```

In [ ]:

```
for index in range(len(some_list)):   #Iterating over list indices
    print(index, ":", some_list[index])
```

In [ ]:

```
for element in some_list:   #Iterating over list
    print(element)
```

In [ ]:

```
for index, value in enumerate(some_list):
    print(index, ":", value)
```

In [ ]:

```
new_list = some_list            #Two variables for one list
another_new_list = some_list[:] #Getting a copy of the list
some_list.clear()
print(new_list)
print(another_new_list)
```

3) Python tuples

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

Examples:

In [ ]:

```
#Declaring tuples
newspapers = 'Die_Presse', 'Der_Standard', 'FAZ'
some_words_and_numbers = ('hi', 'bye', 15)  # Preferred

#Special tuples
one_element_tuple = ('singleton', )
empty_tuple = ()

#newspapers[2] = 'Die_Zeit'    <- Not a valid action for tuples!
```

In [ ]:

```
print(newspapers[2])
```

In [ ]:

```
#Value unpacking / multiple assingment
np1, np2, np3 = newspapers
print(np1, np2, np3)
```

4) 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 uniqe and easily comparable (hashable)

In [ ]:

```
#Declaring a set implicitly: curly brackets {}
backpack = {'notebook', 'phone', 'key', 'gum', 'pen'}

print(backpack)
```

In [ ]:

```
#Declaring with funktion set()
other_bag = set(['key', 'sandwitch', 'bottle'])
one_element_set = set('singleton')

#Empty set - explicit declaration: set(), not{}!
empty_set = set()

#Set with mixed elements
my_set = {1,2,3,3,5,6,4,3,3,3,'hallo'}
print(my_set)

#test_set = {1,2,3,[1,2,3]} <- can't compare int and list (unhashable)
```

4.1) Set: methods

In [ ]:

```
backpack = {'notebook', 'phone', 'key', 'gum'}
other_bag = {'key', 'sandwitch', 'bottle', 'phone'}
bag = {'phone', 'gum'}

print("Intersection: ", backpack & other_bag) #Items in both bags
print("Difference: ",backpack ^ other_bag)    #Items in only one of the bags
print("Union: ", backpack | other_bag)        #Combine both bags
```

5) 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

In [ ]:

```
# Declaring a dictionary
contact_info = {'name': 'someone', 'phone': 12345, 'city': 'Graz'}
```

In [ ]:

```
print("Dictionary keys:", contact_info.keys())
print("Dictionary values:", contact_info.values())
print("Dictionary Items:", contact_info.items())
```

In [ ]:

```
#Working with dictionaries
contact_info['phone'] = 98765  #Changing values using a key

for key, value in contact_info.items():
    print(key, value)
```

In [ ]:

```
contact_info['mail'] = "something@tugraz.at"
print(contact_info)
```

In [ ]:

```
additional_data = {"country": "Austria", "street": "Inffeldgasse"}
contact_info.update(additional_data)
print(contact_info)
```

In [ ]:

```
print("name" in contact_info)
print("test" in contact_info)
```

In [ ]:

```
contact_info["wrong_key"]
```

In [ ]:

```
del contact_info["mail"]
print(contact_info)
```

6) Combination of data structures

- List of lists (matrix)
- Tuple of lists
- Dictionary with list as values
- A dictionary within a dictionary
- and so on

In [ ]:

```
#Simple matrix = list of lists
my_map = []
for x in range(3):
    for y in range(3):
        coordinates = [x,y]
        my_map.append(coordinates)

for coordinates in my_map:
    print(coordinates)
```

In [ ]:

```
#Declaring a list with dictionaries as elements
list_of_grades = [
    {'name': 'Musterman', 'grade': 4},
    {'name': 'Musterfrau', 'grade': 1}
]

print(list_of_grades[0]['name'])
```

In [ ]:

```
#Declaring two combined dictionaries
children = {'son': 15, 'daughter': 18}
cars = {'label': 'volvo', 'power': '120ps'}
parents = {'mother': children, 'father': cars}

print(parents['mother']['son'])
print(parents['father']['power'])
```

Example Program

Create a tic tac toe game.

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

In [ ]:

```
game_board = []
for index in range(3):
    game_board.append(['_']*3)

player = 'X'

for rounds in range(9):
    x = int(input("Enter the x-coordinate: "))
    y = int(input("Enter the y-coordinate: "))
    game_board[y][x] = player
    for row in game_board:
        print(row)
    if player == 'X':
        player = 'Y'
    else:
        player = 'X'
```