EINFÜHRUNG IN DIE PROGRAMMIERUNG

STRUCTURED DATA TYPES

DHBW MANNHEIM
WIRTSCHAFTSINFORMATIK (DATA SCIENCE)

Markus Menth
Martin Gropp

STRUCTURED/COMPOUND DATA TYPES

Python knows a number of compound data types

- Used to group values together

Built-in Types

- `list`
- `tuple`
- `set`
- `dict` (dictionary)

LISTS

LISTS

Store a list of values

- Typically values of the same type (not necessary)
- Written as a list of comma-separated values between square brackets

EXAMPLES

- `list_of_numbers = [1, 4, 9, 16, 25]`
- `mixed_contents = [1, "Hallo", 1-1, 2**3]`

Number of entries via len

- `len(list_of_numbers)`

LISTS: INDEXING AND SLICING

Lists are very similar to strings

- Like strings (and all other built-in sequence type), lists can be indexed and sliced
- Slicing returns a new list containing the requested elements

EXAMPLES

```
list_of_numbers = [1, 4, 9, 16, 25]

list_of_numbers[0]

list_of_numbers[-1]

list_of_numbers[-3:]

list_of_numbers[:]
```

LIST MODIFICATION

Lists support assignment to slices

- This can change the size of the list or clear its contents

EXAMPLE

```
letters = ['a', 'b', 'c', 'd', 'e', 'f', 'g']

letters[2:5] = ['C', 'D', 'E']

letters[2:5] = []

letters[:] = []
```

LIST CONCATENATION AND MODIFICATION

Lists support concatenation, just like strings

```
list_of_numbers + [36, 49, 64, 81, 100]
```

Lists are not immutable (unlike strings): it is possible to change their contents

```
list_of_numbers[4] = 555
```

Items can be added to the end of the list using **append()**

```
list_of_numbers.append(216)
```

LIST NESTING

It is possible to nest lists

- Create lists of lists

Example

```
y = [
    [1, 2, 3],
    [4, 5, 6]
]

print(y)
print(y[1])
print(y[1][2])
```

ITERATING LISTS

Lists can be iterated using `for`

```
for x in [1, 2, 3, 4, 5]:
        print(x)
```

LIST FUNCTIONS

Built-in methods to use on lists

- `copy`
- `count`
- `index`
- ...

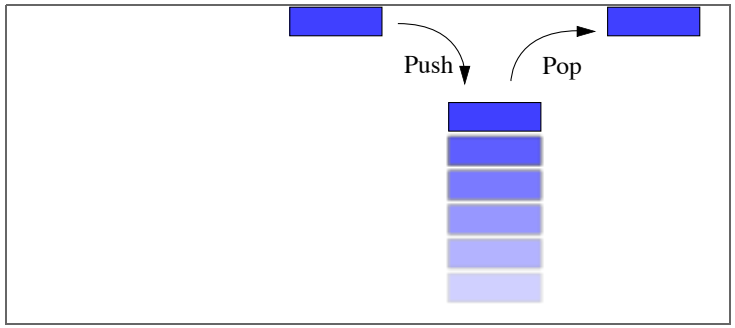
EXAMPLE

```
x = ['A', 'B', 'C']

print(x.count('A'))     # 1
print(x.index('C'))     # 2
```

LISTS AS STACKS

Stacks implements the LIFO principle: last-in, first-out



Lists can be used as stacks

- Add an item: `list.append()`
- Remove an item: `list.pop()`

Additional names: Keller, Stapel

Image Source: User:Boivie [Public domain], **via Wikimedia Commons**

STACKS: EXERCISE

Implement a simple calculator that uses **Reverse Polish / Postfix** notation

- It should support the following operations: +, −, ∗, /
- Test expression `10 20 + 10 ∗` $((10 + 20) * 10)$
- Output: `10 20 + 10 ∗ = 300`

Hints

- Use input to read a line of text from the user
- Parse it to a list of tokens using `string.split(" ")`
- Use a for loop to parse token by token
- Use a stack to store the values

Solution

- **stack-calculator.py**

LISTS AS QUEUES

Lists can be used as **queues**

- Implements the First-In, First-Out (FIFO) principle

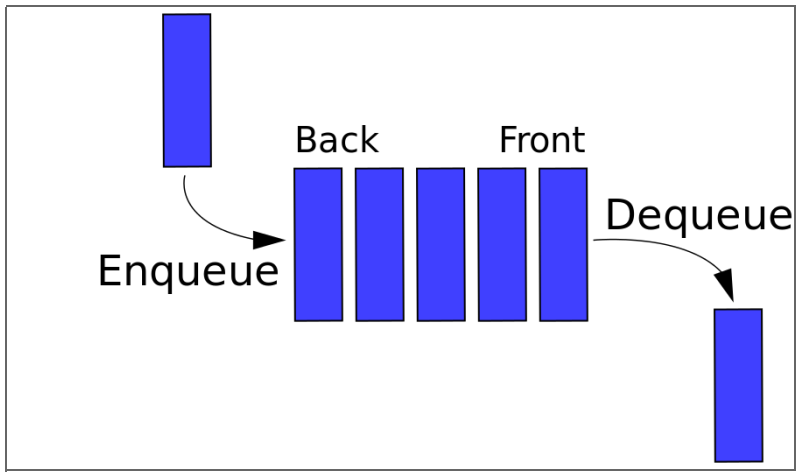


Image Source: User:Vegpuff/Wikipedia. **CC BY-SA 3.0**, **from Wikimedia Commons**

LISTS AS QUEUES

EXAMPLE

```
# Create a list
queue = ['B', 'C']

# Enqueue: Append an item at the end (left)
queue.insert(0, 'D')
print(queue)

# Dequeue: Remove an item from the beginning (right)
queue.pop()
print(queue)
```

EXERCISE

Join every greeting with every name and save it in a list `greetings`. Print out the list.

```
begruessungen = ["Moin", "Guten Tag", "Tach", "Hallöchen"]
namen = ["Peter", "Marie", "Lea"]
```

Solution

- **greetings.py**

TUPLES

TUPLES

Python tuples: comprised of a number of values separated by commas

- Example: `t = (1, 2, 3)`
- Parentheses can be omitted: `t = 1, 2, 3`

Very similar to lists

- Often used in different contexts
- Tuples are immutable and (often) contain heterogeneous data (i.e., different data types)
- Lists are mutable and (often) used with homogeneous contents

UNPACKING OF SEQUENCES

EXAMPLE

```
t = (12345, 54321, 'hello!')
x, y, z = t
```

Frequent use case: Swapping variable contents

```
x, y = y, x
```

ACCESS TUPLES

Tuples can be accessed by referring to the *index* number.

```
x = ('A', 'B', 'C')

print(x[0])
```

You can also loop through tuples by using a *for* loop.

```
x = ('A', 'B', 'C')

for y in x:
    print(y)
```

UPDATE TUPLES

Tuples are immutable!
→ you cannot *change, add or remove* items once the tuple is created.

- **workaround.py**

SETS

SETS

A set is a collection which is unordered and has no duplicate elements. Its elements can not be accessed (or modified) by index.

EXAMPLE

```
basket = {
  'apple', 'orange', 'apple',
  'pear', 'orange', 'banana'
}
```

SET INITIALIZATION

EXAMPLE

Using `set(...)` to convert a list to a set:

```
numbers = set(range(10))

letters = ['a', 'b', 'c', 'c']
set_letters = set(letters)
```

ACCESS SET ITEMS

It is not possible to access a set's items by index or key.

EXAMPLE ☠

```
a = {1, 2, 3}
print(a[0])
```

```
TypeError: 'set' object is not subscriptable
```

ACCESS SET ITEMS

```
numbers = {1, 1, 1, 1, 1, 2, 3, 4, 5}
```

Access a set's items through *for* loop:

```
for number in numbers:
    print(number)
```

Check for a specific value in a set:

```
if 3 in numbers:
    print(True)
```

ADD/REMOVE SET ITEMS

```
numbers = {1, 2, 3}
letters = {'a', 'b', 'c'}
```

Add set items:

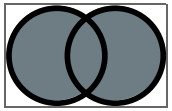
```
numbers.add(4)
numbers.update(letters)
```

Remove set items:

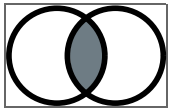
```
numbers.remove(4)
numbers.discard(3)
numbers.pop()
numbers.clear()
```

SET OPERATIONS

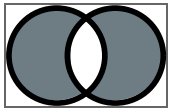
- `x = a.union(b)` *or* `a.update(b)`



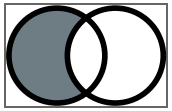
- `x = a.intersection(b)`



- `x = a.symmetric_difference(b)`



- `x = a - b`



EXERCISE

Go to the website **Lorem Ipsum** and copy 100 characters in a variable. Remove through smart joining all letters 'd' and 's' completely.
How many characters are left?
set_100_characters.py

DICTIONARIES / DICTS

Python offers a data structure for key→value mappings called **dict**.
The keys behave like a set.

EXAMPLE

Key	Value
'key1'	'Hallo'
'key2'	1
'key3'	1
'key4'	(1,2,3)

Initialization

```
d = {'key1': 'Hallo', 'key2': 1, 'key3': 1, 'key4': (1, 2, 3)}
```

ACCESSING VALUES BY KEY

- `d[key]`

```
print(d["key1"])
```

  This raises an exception when the key does not exist.

- `d.get(key[, default])`

```
x = d.get("key1", -1)
```

  This does not raise an exception but returns a default value if the key does not exist.

MODIFYING THE DICTIONARY

Change Values and Add Items

```
d["key3"] = 3
d.update({"key6": 3})
```

Remove Items

```
d.pop("key6")
d.popitem()     # removes the last inserted item
del d["key3"]
```

LOOPING THROUGH DICTIONARIES

VIEW OF THE DICTIONARY

- Keys: `d.keys()`
- Values: `d.values()`
- Keys and values: `d.items()`

Example

```
for k, v in d.items():
    print(f"{k}: {v}")
```

DICT: FREQUENTLY USED FUNCTIONS

INFORMATION ABOUT THE DICT

- Number of items:

```
len(d)
```

- Membership testing: `key in d`

```
if 'key1' in d: ...
```

- Membership testing: `key not in d`

```
if 'key1' not in d: ...
```

ADDITIONAL READING

- Google for Education - Python Dict and File

EXERCISES

LISTS

- Create a list `a` with 100 random numbers (e.g., using **randrange**)
- Sort the list in descending order

DICTS

- Count and display occurrences of each word in the US constitution
- Bonus: Sort the dictionary by descending word frequency and display the top 10 words and their frequency (hint: use **sorted** and provide a key)

LIST COMPREHENSIONS

LIST COMPREHENSIONS

Frequent task: create new lists from existing ones, for example by applying an operation to each source element.

```
squares = []
for x in range(10):
        squares.append(x**2)
```

List comprehensions provide a concise way to create lists:

```
squares = [
    x**2
    for x in range(10)
]
```

LIST COMPREHENSIONS

Definition

- A list comprehension consists of square brackets [ ]
- They contain an expression followed by a for clause
- The for clause is followed by zero or more for or if clauses
- Result: new list

LIST COMPREHENSIONS

EXAMPLE (SINGLE FOR AND NO IF)

```
squares = [
    x**2
    for x in range(10)
]
```

Example (single for and single if)

```
squares = [
    x**2
    for x in range(10)
    if x % 2 == 0
]
```

LIST COMPREHENSIONS

Multiple for statements can be used

```
begruessungen = ("Moin", "Guten Tag", "Tach", "Hallöchen")
namen = ("Peter", "Marie", "Lea")

[
    begruessung + " " + name
    for name in namen
    for begruessung in begruessungen
]
```

LIST COMPREHENSIONS

Multiple for and if statements can be used

```
[
    begruessung + " " + name
    for name in namen
    if name != "Peter"
    for begruessung in begruessungen
    if begruessung.endswith('n')
]
```

- E.g., create new lists with a subset of entries that fulfill a certain criterion

NESTED LIST COMPREHENSIONS

List comprehensions can be **nested**
Example: transposing a matrix

```
matrix = [
        [1, 2, 3, 4],
        [5, 6, 7, 8],
        [9, 10, 11, 12],
]
```

SOLUTION

```
[
    [row[i] for row in matrix]
    for i in range(
        max([
            len(r)
            for r in matrix
        ])
    )
]
```

NONE

NONE

None defines a null value, or no value at all.

```
x = None

if x:
  print("Do you think None is True?")
elif x is False:
  print("Do you think None is False?")
else:
  print("None is not True, or False, None is just None...")
```

NONE

None is not the same as

- 0
- True / False
- empty string.

None is a data type of its own (NoneType) and only None can be None.

```
x = None

print(type(x))    # <class 'NoneType'>
```

Source: **w3schools**

EXERCISE

LIST COMPREHENSIONS: EXERCISES

Exercise

- Create a list `a` with 100 random positive and negative numbers
    - `from random import randrange`
    - `randrange(-100, 100)`
- Use a list comprehension to keep only the negative values

Exercise

- Use list `a` from the first exercise
- Apply the abs function on each element

LIST COMPREHENSIONS: EXERCISES

Exercise

- Strip leading and trailing whitespace from this list of words: `freshfruit = [' banana', ' loganberry ', 'passion fruit ']`
- Use **string.strip**

Exercise

- Use list `a` with numbers from a previous exercise
- Create a list of 2-tuples like (number, square) for each entry

LIST COMPREHENSIONS: EXERCISES

Rewrite the following program to use list comprehensions

```
sentence = "the quick brown fox jumps over the lazy dog"
words = sentence.split()
word_lengths = []
for word in words:
      if word != "the":
          word_lengths.append(len(word))
print(word_lengths)
```

LIST COMPREHENSIONS: EXERCISES

Find all of the numbers from 1-1000 that are divisible by 7

SOLUTION

```
results = [
    num
    for num in range(1000)
    if num % 7 == 0
]
```

LIST COMPREHENSIONS: EXERCISES

Find all of the numbers from 1-1000 that have a 3 in them

SOLUTION

```
results = [
    num
    for num in range(1000)
    if '3' in str(num)
]
```

LIST COMPREHENSIONS: EXERCISES

Count the number of spaces in a string

```
teststring = '''Find all of the words in a
               string that are less than 4 letters'''
```

SOLUTION

```
teststring = '''Find all of the words in a
                string that are less than 4 letters'''
results = [
    character
    for character in teststring
    if character == ' '
]
```

LIST COMPREHENSIONS: EXERCISES

Remove all of the vowels in a string (make a list of the non-vowels)

```
teststring = '''Find all of the words in a
                string that are less than 4 letters'''
vowels = ['a','e','i','o','u',' ']
```

SOLUTION

```
teststring = '''Find all of the words in a
                string that are less than 4 letters'''
vowels = ['a','e','i','o','u',' ']

results = [
    letter
    for letter in teststring
    if letter.lower() not in vowels
]
```

LIST COMPREHENSIONS: EXERCISES

Find all of the words in a string that are less than 4 letters

```
teststring = '''Find all of the words in a
                string that are less than 4 letters'''
```

SOLUTION

```
teststring = '''Find all of the words in a
                string that are less than 4 letters'''
results = [
    word
    for word in teststring.split()
    if len(word) < 4
]
```

LIST COMPREHENSIONS: EXERCISES

Challenge: use a **dictionary comprehension** to determine the length of each word in a sentence

```
sentence = '''Use a dictionary comprehension to determine the
            length of each word in a sentence'''
```

SOLUTION

```
sentence = '''Use a dictionary comprehension to determine the
             length of each word in a sentence'''
results = {
    word: len(word)
    for word in sentence.split()
}
```

EXERCISE

FIBONACCI-FOLGE

SOLUTION

```
n = 100
a, b = 0, 1
while a < n:
  print(a, end=' ')
  a, b = b, a+b
  print()
```