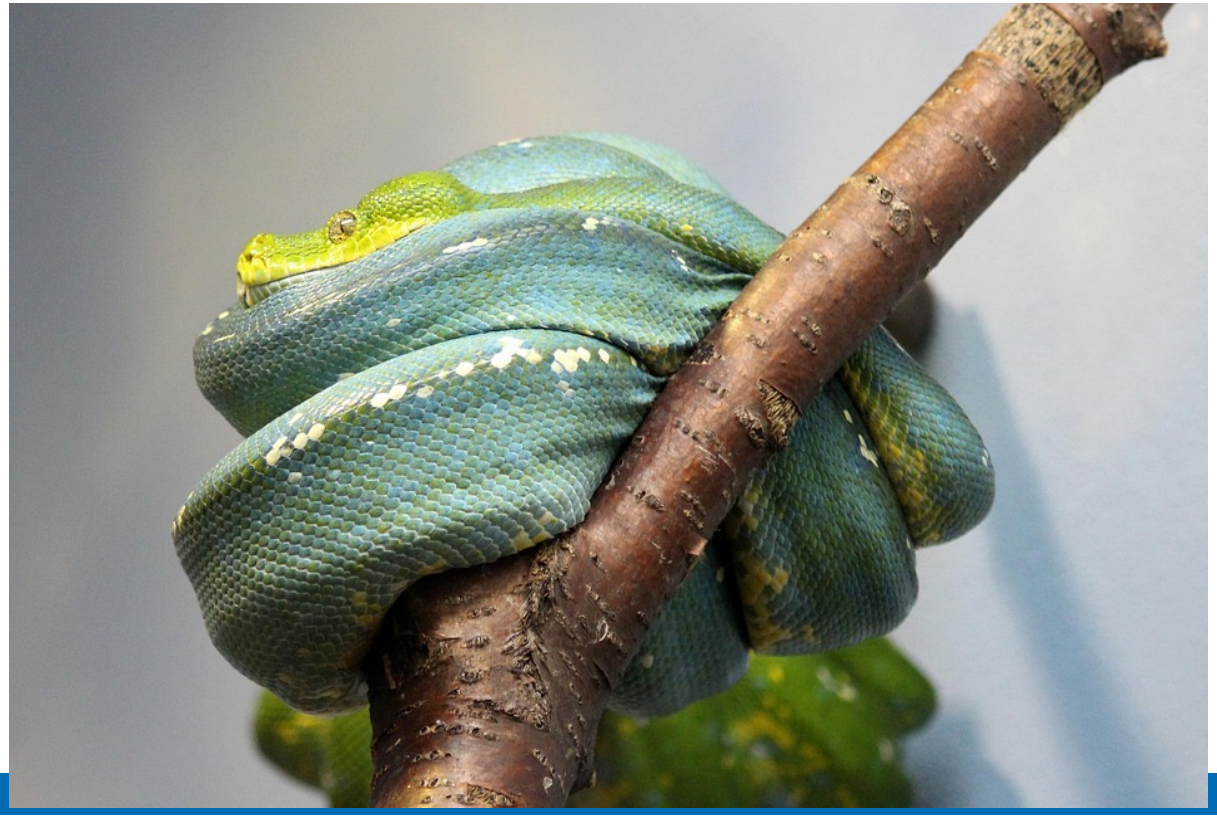
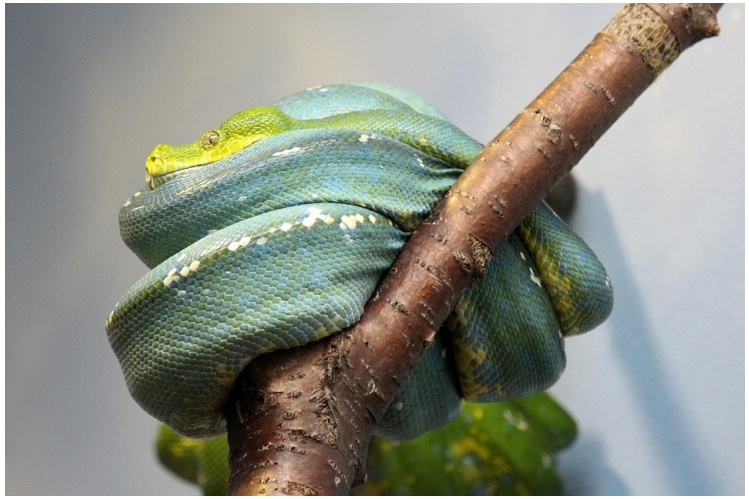
Programming in Python



Topics

1. Running system commands and programs
2. Character string formatting (repetition)
3. List comprehensions (repetition)
4. Set comprehensions
5. Dictionary comprehensions



Running system commands and programs

Heinrich Heine
Universität Düsseldorf

The function `os.system()`

- It is possible, to run shell commands and external programs within the run-time of a Python program.
- This requires the function `os.system(…)` from module `os`.
- The function will wait until the process terminates and returns its exit status (Linux).

```
[2]: import os

[3]: os.system("makeblastdb -in alle.fasta -dbtype prot")

[3]: 0

[4]: os.system("blastp -query arabidopsis.fasta -db alle.fasta -out ara.blast")

[4]: 0
```

Running system commands and programs

Heinrich Heine
Universität Düsseldorf

The function `os.system()`

[2]:
```
import os
```

[3]:
```
os.system("makeblastdb -in alle.fasta -dbtype prot")
```

[3]: 0

[4]:
```
os.system("blastp -query arabidopsis.fasta -db alle.fasta -out ara.blast")
```

[4]: 0

[5]:
```
with open("ara.blast", "r") as fin:
    for zeile in fin:
        zeile = zeile.rstrip()
        print(zeile)
```

```
BLASTP 2.11.0+
```

Running system commands and programs

Heinrich Heine
Universität Düsseldorf

The function `os.system()`

[5]:
```
with open("ara.blast", "r") as fin:
    for zeile in fin:
        zeile = zeile.rstrip()
        print(zeile)
```

```
BLASTP 2.11.0+


Reference: Stephen F. Altschul, Thomas L. Madden, Alejandro A.
Schaffer, Jinghui Zhang, Zheng Zhang, Webb Miller, and David J.
Lipman (1997), "Gapped BLAST and PSI-BLAST: a new generation of
protein database search programs", Nucleic Acids Res. 25:3389-3402.


Reference for composition-based statistics: Alejandro A. Schaffer,
L. Aravind, Thomas L. Madden, Sergei Shavirin, John L. Spouge, Yuri
I. Wolf, Eugene V. Koonin, and Stephen F. Altschul (2001)
```

Running system commands and programs

Heinrich Heine
Universität Düsseldorf

The function `os.popen()`

- The function `os.popen(…)` can be used to create a sequence object containing all output lines returned by a started process as items.

```
[2]: import os
```

```
[3]: for zeile in os.popen("ls -l /usr/sbin |head -4"):
         print(zeile.rstrip())
```

```
total 7212
lrwxrwxrwx 1 root root         7 Apr 16  2020 addgroup -> adduser
-rwxr-xr-x 1 root root       860 Dec  7  2019 add-shell
-rwxr-xr-x 1 root root     37785 Apr 16  2020 adduser
```

Running system commands and programs

Heinrich Heine
Universität Düsseldorf

Example for calling an external program

- In the following example, we want to create a pairwise alignment of two sequence files.
- The external program should be `needle` from the EMBOSS-package (pairwise global alignment according to Needleman & Wunsch).
- We use command line arguments to create an output file containing the alignment.

Running system commands and programs

Heinrich Heine Universität Düsseldorf

Example for calling an external program

[19]:
```
import os
```

[20]:
```
seq1 = "homo.fasta"
seq2 = "arabidopsis.fasta"
alignment = seq1 + "_vs_" + seq2 + ".needle"
```

[21]:
```
print("Wir erstellen nun ein Alignment mit", seq1, "und", seq2, "...")
aufruf = "needle -asequence " + seq1 + " -bsequence " + seq2 + " -outfile " +␣
 ↪alignment + " -auto"
print("Aufruf: " + aufruf)
os.system(aufruf)
print("Fertig.")
```

```
Wir erstellen nun ein Alignment mit homo.fasta und arabidopsis.fasta …
Aufruf: needle -asequence homo.fasta -bsequence arabidopsis.fasta -outfile
homo.fasta_vs_arabidopsis.fasta.needle -auto
Fertig.
```

Running system commands and programs

Heinrich Heine
Universität Düsseldorf

Example for calling an external program

[22]:
```
with open(alignment, "r") as fin:
    for zeile in fin:
        zeile = zeile.rstrip()
        print(zeile)
```

```
########################################
# Program: needle
# Rundate: Fri 26 Mar 2021 09:37:48
# Commandline: needle
#    -asequence homo.fasta
#    -bsequence arabidopsis.fasta
#    -outfile homo.fasta_vs_arabidopsis.fasta.needle
#    -auto
# Align_format: srspair
# Report_file: homo.fasta_vs_arabidopsis.fasta.needle
########################################
```

Character string formatting

Heinrich Heine
Universität Düsseldorf

- There are several ways to format a character string.
- One variant uses the `format(…)` method of a string object.
- Within the character string, the placeholders `{` and `}` can be included together with optional format codes. The arguments of `format()` will then be inserted into the returned character string in place of the respective placeholders.

Character string formatting

Heinrich Heine
Universität Düsseldorf

Insertion without formatting

- In the simplest case, curly brackets are used within a character string. They will be replaced by given arguments to the `format(…)` method in the order of appearance.

```
[1]: s = "Gegeben seien die Werte {},{} und {}.".format(3,8,6)
     print(s)
```

```
Gegeben seien die Werte 3,8 und 6.
```

Character string formatting

Heinrich Heine
Universität Düsseldorf

Insertion by number

- If a number is given within the curly brackets, the placeholder will be replaced by the respective argument according to the number.

[1]:
```
s1 = "Hallo"
s2 = "Welt"
s = "{1} {0}".format(s2,s1)
print(s)
```

```
Hallo Welt
```

Character string formatting

Heinrich Heine
Universität Düsseldorf

Insertion by keyword

- In contrast to the standard enumeration of arguments, it is also possible to use keyword arguments.

```
[1]: s = "Gegeben sei die Menge {{{Wert1},{Wert2},{Wert3}}}".
      ↪format(Wert1=3,Wert2=8,Wert3=6)
     print(s)
```

```
Gegeben sei die Menge {3,8,6}
```

Character string formatting

Heinrich Heine
Universität Düsseldorf

Curly backets in formatted strings

- To be able to contain the literal characters `{` and `}` in a formatted string, they have to be written twice.

[1]:
```
s = "Gegeben sei die Menge {{{Wert1},{Wert2},{Wert3}}}".
 ↪format(Wert1=3,Wert2=8,Wert3=6)
print(s)
```

```
Gegeben sei die Menge {3,8,6}
```

Character string formatting

Heinrich Heine
Universität Düsseldorf

Formatting codes

- Following the keyword or position identifiers, a formatting code can be included.
- The formatting code can also stand alone.
- Formatting codes are similar to the formatting codes of the function `printf(…)`, which is known from other higher programming languages.

[1]:
```
s = "Gegeben seien die Werte {:.2f},{:07.3f}. {:9s}.".format(3.14159265,80.
↪6666666,"Test")
print(s)
```

```
Gegeben seien die Werte 3.14,080.667. Test     .
```

Character string formatting

Heinrich Heine
Universität Düsseldorf

Combining keywords and formatting codes

- Keywords and formatting codes can also be combined.

[1]:
```
n = 5
s = "Blätter: {leaves:5d}   Aeste: {branches:5d}".
 ↪format(leaves=n,branches=2*n-3)
print(s)
```

```
Blätter:     5   Aeste:     7
```

Character string formatting

Heinrich Heine
Universität Düsseldorf

Left - and right-justification

- Starting the formatting code with the characters < and > is useful to justify the inserted text accordingly.

[1]:
```
s = "{s:>10s}: {leaves:>5d}".format(s="Blaetter",leaves=5)
print(s)
```

```
  Blaetter:     5
```

[2]:
```
s = "{s:<10s}: {leaves: 5d}".format(s="Blaetter",leaves=5)
print(s)
```

```
Blaetter  :     5
```

Character string formatting

Heinrich Heine
Universität Düsseldorf

Formatting code: Flags

- Shape of the formatting code:
  [Flags][Width][.Precision]Type

Flag	Meaning
#	Prefix for octal or hexadecimal numbers
0	Padding with 0
-	left-justified
	Leading space character, if numerical value is unsigned
+	Always print sign of numerical value (+ or -)

Character string formatting

Heinrich Heine
Universität Düsseldorf

Formatting code: Width

- Shape of the formatting code:

  [Flags][Width][.Precision]Type
- The width defines how many positions are reserved for the value.
- The value is fully inserted, if it is larger than the reserved space.
- The value is padded by space characters or 0, depending on the flags, if it is smaller than the reserved space.

Character string formatting

Formatting code: Precision

- Shape of the formatting code:
  [Flags][Width][.Precision]Type
- The precision defines the number of significant post decimal positions to be shown.
- The value will be rounded to the last significant post decimal position displayed.

```
[2]: import math
     print("pi =",math.pi)

     pi = 3.141592653589793
```

```
[3]: s = "pi = {:.3f}".format(math.pi)
     print(s)

     pi = 3.142
```

Character string formatting

Heinrich Heine
Universität Düsseldorf

Formatting code: Type

- Shape of the formatting code:
  [Flags][Width][.Precision]Type

Typ	Meaning
d	Signed integer
u	Unsigned integer (decimal)
o	Unsigned integer (octal)
x	Unsigned integer (hexadecimal)
e	Floating point number in exponential format
f	Floating point number
s	character string

List comprehensions

Heinrich Heine
Universität Düsseldorf

Compact list definition

- List comprehensions are a compact way to define a list.
- They consist of:
  - An input sequence
  - A variable, representing the elements in the input sequence.
  - An optional dependency
  - A destination expression, which is applied to the elements in the input sequence, which fulfill the optional dependency, and transfers the results to the output list.

List comprehensions

Heinrich Heine
Universität Düsseldorf

Compact list definition

- In most cases, we could also use the function `map(…)`.

-
```
[i**2 for i in range(10) if i % 2 == 0]
```

Output expression

Variable

Input sequence

Optional dependency

List comprehensions

Heinrich Heine
Universität Düsseldorf

Compact list definition

[ ]:
```
import random
```

[1]:
```
l = []
for i in range(10):
    if i % 2 ==0:
        l.append(i**2)
print(l)
```

```
[0, 4, 16, 36, 64]
```

Die Liste kann auch mit Hilfe einer List-Comprehension erzeugt werden.

[3]:
```
l = [i**2 for i in range(10) if i % 2 == 0]
print(l)
```

```
[0, 4, 16, 36, 64]
```

List comprehensions

Heinrich Heine
Universität Düsseldorf

Compact list definition with nested loops

- List comprehensions may contain nested loops

```
[1]: l = [i+j for i in range(2) for j in range(3)]
     print(l)
```

```
[0, 1, 2, 1, 2, 3]
```

List comprehensions

Heinrich Heine
Universität Düsseldorf

Various object types in the destination expression

- The output expression may be expression, also a tuple for example, leading to a list of tuples.

```
[1]: liste = [(i,j) for i in range(6) for j in range(4) if i != j and i < j]
     print(liste)
```

```
[(0, 1), (0, 2), (0, 3), (1, 2), (1, 3), (2, 3)]
```

List comprehensions

Heinrich Heine
Universität Düsseldorf

Using a list comprehension to create a matrix

- Nested list comprehensions are an easy way to define a matrix as a list of lists.

[1]:
```
import random
```

Eine 3 x 2 Matrix mit Zufallszahlen generieren.

[2]:
```
l = [[random.randrange(1,11) for spalte in range(2)] for zeile in range(3)]
print(l)
```

```
[[8, 1], [1, 4], [9, 10]]
```

List comprehensions

Heinrich Heine
Universität Düsseldorf

Examples

- Create a list comprehension that returns a list with 10 zeros.

List comprehensions

Heinrich Heine
Universität Düsseldorf

Examples

- Create a list comprehension that returns a list of lists containing 10 rows and 5 columns that represents the matrix in the example:

```
[[0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0],
 [0, 0, 0, 0, 0]]
```

List comprehensions

Heinrich Heine
Universität Düsseldorf

Examples

- Create a list comprehension that returns a list of lists representing the matrix in the example:

  [[0, 1, 2, 3, 4],

  [5, 6, 7, 8, 9]]

Set comprehensions

Heinrich Heine
Universität Düsseldorf

Compact set definition

- Analogous to list comprehensions, we can also define sets in a compact manner.
- In contrast to list comprehensions, every element in the output set will be unique.

[1]:
```
set1 = {s**2 for s in [-4, -3, -2, -1, 0, 1, 2, 3, 4]}
print(set1)
```

```
{0, 1, 4, 9, 16}
```

Set comprehensions

Heinrich Heine
Universität Düsseldorf

Examples

- Create a set comprehension that returns a set of the following tuples:
  {(1, 2), (1, 3), (1, 4), (2, 3), (2, 4), (3, 4)}

Set comprehensions

Heinrich Heine
Universität Düsseldorf

Sieve of Eratosthenes

[1]:
```
import math
```

[2]:
```
n = 100
sqrtn = int(math.sqrt(n))
nonprime = set()
```

[3]:
```
# Mark all multiples of a prime number as non-prime,
# starting with 2.
for i in range(2, sqrtn + 1):
    if i not in nonprime:
        # Multiples of i are non-prime:
        # We can begin with i * i, because k * i with k < i
        # had been stroke out by multiples of k.
        for j in range(i * i, n + 1, i):
            nonprime.add(j)
```

Set comprehensions

Heinrich Heine
Universität Düsseldorf

Sieve of Eratosthenes

[4]:
```
# Print out prime numbers:
prime = set(range(2, n + 1)) - nonprime
print(prime)
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97}
```

Set comprehensions

Heinrich Heine
Universität Düsseldorf

Sieve of Eratosthenes using set comprehension

[1]:
```
import math
```

[2]:
```
n = 100
sqrtn = int(math.sqrt(n))
```

[3]:
```
# Mark all multiples of a prime number as non-prime,
# starting with 2.
nonprime = {j for i in range(2, sqrtn + 1) for j in range(i * i, n + 1, i)}
```

[4]:
```
# Print out prime numbers:
prime = set(range(2, n + 1)) - nonprime
print(prime)
```

```
{2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73,
79, 83, 89, 97}
```

Dictionary comprehensions

Heinrich Heine
Universität Düsseldorf

We can use the same construction for a compact dictionary definition

[1]:
```
dic = {i: 0 for i in range(10)}
```

[2]:
```
print(dic)
```

```
{0: 0, 1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 0, 7: 0, 8: 0, 9: 0}
```