

MODUL 122

ABLÄUFE MIT SCRIPTS AUTOMATISIEREN – TEIL 5 PYTHON

Urs Dummermuth
Version 2022

Inhalt

Python.....	3
› Entwicklungsgeschichte	4
› Arbeitsumgebung.....	4
Interaktiver Modus	5
› Strukturierung durch Einrückung.....	7
Aufgabe - Sprachvergleich	7
Beispiel Zahlen und Generatoren	8
Auftrag - Beispiel integer.py ausführen	8
Grundrechenoperationen	10
Fibonacci-Folge	10
Beispiel - Fibonacci.py	12
Beispiel - Strings.py (Zeichenketten)	13
Programm ggt_python.py	14
Programm fib_rek.py	14
› Python-Module.....	15
Einstieg	15
Ausgangslage - Betrachten wir ein Beispiel:	15
› Funktionen	16
Die Funktion open()	16
Import-Befehl	18
Aufgabe - Erstellen Sie ein Modul listmemoryinfo	19
Die dir()-Funktion	20
› Python und Argumente	21
Programm Systeminfo.py	21
› Einlesen von Daten unter Python	22
› Objektorientierte Programmierung	23
Aufgabe - Programmieren Sie eine Klasse	24
› Tkinter	25
Vorbereitung.....	25
Beispiel 1	25
› Weiterführende Aufgaben.....	26
Lesen - Stöbern im Begleitbuch «Programmierung mit Python»	26
Video - Object Oriented Programming (OOP) - For Beginners.....	26
Video - pytsx3 and PyPDF2	26
Aufgabe - Was können Sie für Ihre Praxisarbeit verwenden?	26

Python

Einstieg

Ausgangslage

Python wurde mit dem Ziel entworfen, möglichst einfach und übersichtlich zu sein. Dies soll durch zwei Maßnahmen erreicht werden:

- › Zum einen kommt die Sprache mit relativ wenigen Schlüsselwörtern aus,
- › zum anderen ist die Syntax reduziert und auf Übersichtlichkeit optimiert.

Python-Programme sind systemunabhängig und können sowohl auf einer Linux- wie auch Windows-Entwicklungsumgebung erstellt werden. Mit `where .exe python` finden wir heraus, ob und wo Python unter Windows installiert ist. Wir werden jedoch in der Folge auf der Linux-Arbeitsumgebung der bmLP1 arbeiten.

Als zusätzliche Ressourcen stehen jeweils im gleichnamigen Unterverzeichnis der Arbeitsblätter Bücher und Links zur Verfügung.

- › Herdt-Verlag - Campus
- › [WiseOwl - Programming in Python - YouTube](#)

Handlungsziele

1. Zu automatisierende Funktion oder zu automatisierenden Ablauf mit den dazugehörigen Benutzerinteraktionen als Ablaufstruktur (z.B. Programmablaufplan) grafisch darstellen.
2. Ablaufstruktur mit Hilfe einer Scriptsprache umsetzen.
3. Script in eine Systemumgebung integrieren.
4. Script auf eine vollständige und korrekte Ausführung der erforderlichen Funktionalität bzw. des Ablaufs testen.
5. Dokumentation für den Einsatz des Scripts erstellen.

Unterrichtsziele

- › Sie können Python im interaktiven Modus ausführen und Befehle und Funktionen testen
- › Sie können die Python-Programme ausführen und verstehen
- › Sie können diese Python-Programme für ihre eigenen Bedürfnisse anpassen
- › Sie können ein Praxisprojekt in Python gemäss LBV-122-6 realisieren und dokumentieren
- › Sie können das Programm demonstrieren und den Nutzen erklären

Entwicklungsgeschichte

Die Sprache wurde Anfang der 1990er Jahre von Guido van Rossum am CENTRUM WISKUNDE & INFORMATICA In Amsterdam als Nachfolger für die Programmier-Lehrsprache ABC entwickelt und war ursprünglich für das verteilte Betriebssystem Amoeba (verteiltes Betriebssystem von Andrew Tannenbaum, <http://www.cs.vu.nl/~ast/>) gedacht. Ein Pythonskript wird transparent in einen Zwischencode übersetzt, der dann vom Interpreter ausgeführt wird.

Arbeitsumgebung

Wir arbeiten mit den virtuellen Maschinen bmLP1.

› Öffnen Sie auf bmLP1 ein Terminal (Ctrl+Alt+T):

```
vmadmin@bmLP1:/home/vmadmin# sudo su <Enter>
root@bmLP1:/home/vmadmin#
```

Geben Sie folgende Befehle ein:

```
python3.10 --version
```

oder

```
type -a python3.10
python3.10 ist /usr/bin/python3.10
python3.10 ist /bin/python3.10
```

› Sollte keine Version angezeigt werden, installieren Sie das Python Paket wie folgt:

Pakete aktualisieren und installieren

```
sudo apt-get upgrade
sudo add-apt-repository ppa:deadsnakes/ppa
sudo apt-get update
sudo apt-get install python3.10
```

Interaktiver Modus starten zu Testzwecken:

Python3.10

```
root@bmLP1:/home/vmadmin# python3.10
Python 3.10.2 (main, Jan 15 2022, 18:02:07) [GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Mit Ctrl+d wird der Modus beendet.

Auf den Geschmack bringen

- › **Abstract data types** Python ist einfach, aber mächtig. Mächtig, da effiziente und abstrakte aber vielseitig einsetzbare Datenstrukturen zur Verfügung stehen.
- › **Easy object oriented programming concept** Python ist leistungsstark im objektorientierten Paradigma und bleibt dabei verständlich und einfach.
- › **Rapid application development** Python unterstützt die schnelle Softwareerstellung.

Es gibt viele weitere einleitende Eigenschaften zu Python; Python muss jeder selbst entdecken und erleben. Mastering Python the Right Way kann Ihnen beim Einsteigen helfen:

https://dev.to/brayan_kai/mastering-python-the-right-way-2gi

Interaktiver Modus

Mit der Eingabe von python3.10 im Terminal starten Sie den interaktiven python-Modus:

```
root@bmlp1:/home/vmadmin# python3.10
Python 3.10.2 (main, Jan 15 2022, 18:02:07) [GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Der interaktive python-Modus kann mit `ctrl+d` (steuerung + d) verlassen werden.

Beispiel: Zuerst werden die benötigten Module importiert, dann die Funktion verwendet:

```
>>>import os, sys
```

Die Module stehen nun bis zum Verlassen des interaktiven Modus' zur Verfügung.

```
>>>print(sys.version)
```

Anzeige:

```
Python 3.10.2 (main, Jan 15 2022, 18:02:07) [GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

```
>>>print(os.getcwd())
```

Anzeige:

```
/home/vmadmin/M122_Script/Python
```

Wechseln des Arbeitsverzeichnisses:

```
>>> os.chdir('/home/vmadmin/M122_Script/Python')
>>> print(os.getcwd())
```

Anzeige:

```
/home/vmadmin/M122_Script/Python
```

Home-Verzeichnis:

```
>>> os.path.expanduser('~')
```

Anzeige:



Beispiel mit Zeichenketten:

```
>>> FirstAndLastname='Hans'+ ' '+'Muster'  
>>> print(FirstAndLastname)
```


Anzeige:



Arithmetisches Beispiel:

```
>>> Einnahmen=50+20+120  
>>> Kosten=120+55  
>>> Einnahmen=Einnahmen+60  
>>> print(Einnahmen-Kosten)
```

Anzeige:



Beispiel mit einer Bedingung:

```
python3.10  
free_your_mind = True  
if free_your_mind:  
    print("... your brain will follow!")  
... your brain will follow!
```

```
root@bmLP1:/home/vmadmin# python3.10  
Python 3.10.2 (main, Jan 15 2022, 18:02:07) [GCC 9.3.0] on linux  
Type "help", "copyright", "credits" or "license" for more information.  
>>> free_your_mind = True  
>>> if free_your_mind:  
...     print("... your brain will follow!")  
...  
... your brain will follow!  
>>> 
```

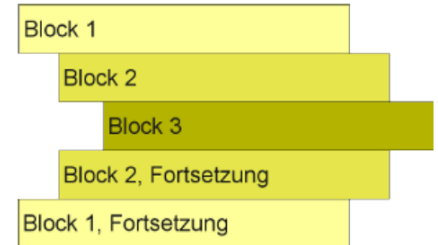
Siehe dazu auch: <https://docs.python.org/3/tutorial/interpreter.html>

Strukturierung durch Einrückung

Das Strukturierungsprinzip von Python unterscheidet sich deutlich von anderen Programmiersprachen. Andere Sprachen strukturieren (klammern) Programmblöcke durch Schlüsselwörter, wie beispielsweise "begin", "end", "do", "done" oder geschweiften Klammern.

In Python ist dies nun gänzlich anders. Hier haben Leerzeichen eine Bedeutung. Die Einrückung von Zeilen dient hier als Strukturierungselement, so dass Programmierer "gezwungen" werden übersichtlichen Code zu schreiben.

Wesentlich sind hierbei der Doppelpunkt „:“ am Ende des Anweisungskopfes und die gleichmässige Einrückung der zugehörigen Anweisungen:



Beispiel: Führen Sie im interaktiven Python-Modus folgendes Listing aus:

```
>>> age = -5
>>> if age < 0: # Der Anweisungskopf wird mit ":" abgeschlossen
    print("Das stimmt wohl kaum!") #Mit Tabulator einrücken!
```

Das stimmt wohl kaum!"

```
>>> languages = ["Bash", "C", "C++", "Eiffel", "Java", "Perl",
"Python"]
>>> for x in languages:
    print(x) #Mit Tabulator einrücken!
```

Beispiel: Führen Sie im interaktiven Python-Modus folgendes Listing aus:

```
>>> # Die Längen einiger Zeichenketten ermitteln:
>>> kittycat = ['Katze', 'Fenster', 'rauswerfen']
>>> for x in kittycat:
    print(x, len(x))
```



Aufgabe - Sprachvergleich

Was unterscheidet die Listen languages von kittycat in den Beispielen eins und zwei?

Beispiel Zahlen und Generatoren

Untersuchen wir den Umgang mit Zahlen und lernen gleich Generatoren kennen.



Auftrag - Beispiel integer.py ausführen

- › Das Script muss ausführbar sein.
- › Wir können die Berechtigungen zuerst anzeigen oder gleich neu setzen:
 `chmod +x integer.py`
- › Dann starten Sie den Editor und führen das Script aus.

Code:

```
#!/usr/bin/python3.10
#-----
# integer.py
#-----
print("Integer und Float in Anwendung")
import math
from math import sqrt

# String ausgeben
print("-----")
print("Hello Python")
print("-----")

# Integer: Pythagoras
a = 5
b = 6
print("Integer: " + str(a) + ", " + str(b))
print("Pythagoras  $a^2+b^2+c^2$ ")

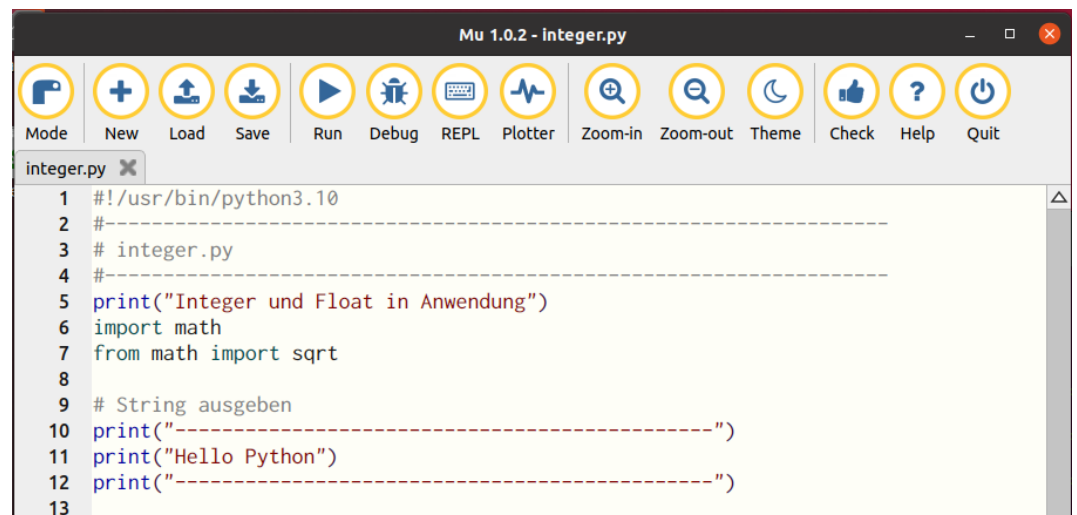
# Potenzierung geschieht mit dem ** Operator
quadratsumme = a**2 + b**2

# Float erlaubt Gleitkommazahlen, int nur ganze Zahlen
c = float(sqrt(quadratsumme))
# Ausgabe
print("a=5, b=6, c="+str(c))
print(a,b,c)

# Division
print("Division mit / Operator")
print(100 - 36 / 10) # Division mit /
print("Division mit // Operator")
print(100 - 36 // 10) # Division mit //
```


Starten Sie ein Terminal und starten Sie den Editor mit

```
vmadmin@bmLP1:~$ mu-editor
```



Das Script wird mit Load geöffnet und mit Run gestartet:

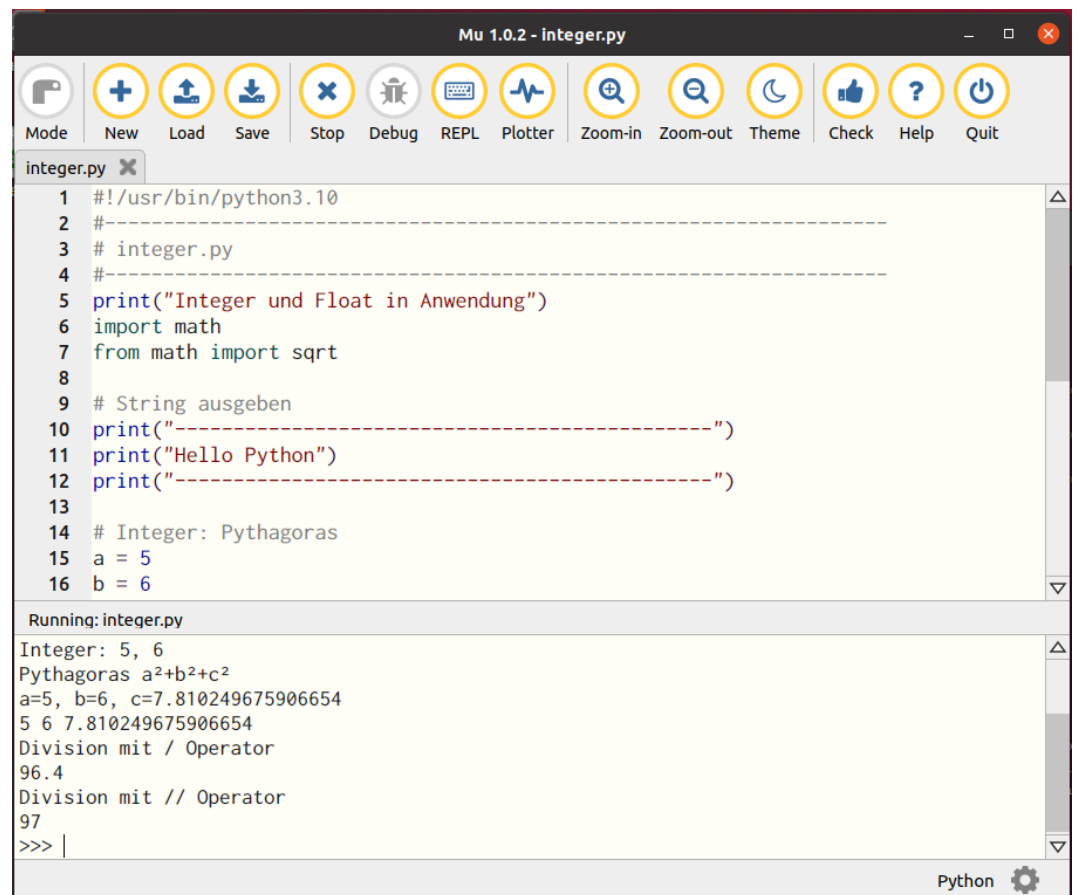


Abb. 1 - Script im MU-Editor geladen und ausgeführt



Grundrechenoperationen


Ergänzen Sie das Skript mit folgenden Berechnungen und geben Sie das Resultat aus:

```
(100 - 36 / 10)  
(100 - 36 // 10)
```

Verwenden Sie folgenden Kopfteil für das neue Codesegment.

```
# Division  
print("Division mit / und // Operator")
```

Was stellen Sie bei den zwei Divisionen fest?



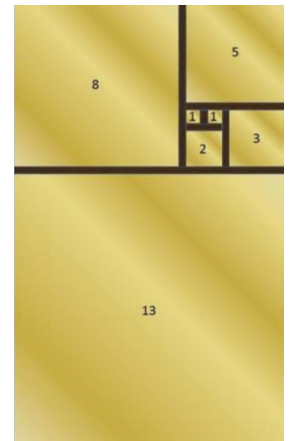
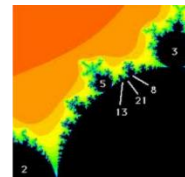
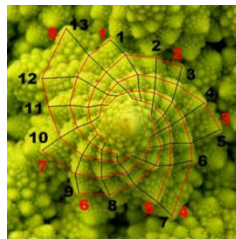
Fibonacci-Folge

Die Fibonacci-Folge ist eine unendliche Folge von Zahlen (den Fibonacci-Zahlen), bei der sich die jeweils folgende Zahl durch Addition ihrer beiden vorherigen Zahlen ergibt:
0, 1, 1, 2, 3, 5, 8, 13, ...

Benannt ist sie nach Leonardo Fibonacci, der damit 1202 das Wachstum einer Kaninchenpopulation beschrieb.

Die Fibonacci-Folge wird oftmals von Hollywood-Esoterikern und anderen Mystikern zur kommerziellen Vermarktung genutzt. Die Anziehungskraft kommt wohl von den vielen Fibonacci-basierten Annahmen

Fibonacci-Folgen in Natur und Wissenschaft:



Wie lassen sich Fibonacci Zahlen berechnen?

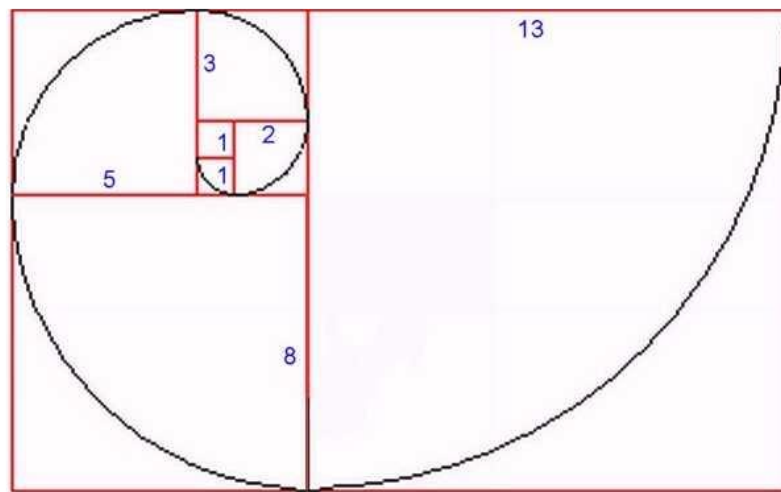
$\text{fib}(n) = \text{fib}(n-1) + \text{fib}(n-2)$ mit $\text{fib}(1) = \text{fib}(2) = 1$

$\text{fib}(3) = \text{fib}(2) + \text{fib}(1) \rightarrow 2 = 1 + 1$
 $\text{fib}(4) = \text{fib}(3) + \text{fib}(2) \rightarrow 3 = 2 + 1$
 $\text{fib}(5) = \text{fib}(4) + \text{fib}(3) \rightarrow 5 = 3 + 2$
 $\text{fib}(6) = \text{fib}(5) + \text{fib}(4) \rightarrow 8 = 5 + 3$

Sie sollten nach den Arbeitsblättern AB122-0401 bis AB122-0407 in der Lage sein, die Fibonacci-Folge rekursiv zu programmieren.

Wir machen es mit einem **Python-Generator**. Generatoren sind eine einfache und mächtige Möglichkeit, Iteratoren zu kreieren. Äußerlich gleichen sie Funktionen.

Syntaktisch betrachtet gibt es nur einen Unterschied: Statt der `return`-Anweisung findet man in einem Generator eine oder mehrere `yield`-Anweisungen.



Merkmale von Generatoren:

- › Ein Generator wird wie eine Funktion aufgerufen.
- › Er liefert als Rückgabewert ein Iterator-Objekt zurück.
- › `yield` wirkt dann wie ein `return` einer Funktion, d.h. der Wert des Ausdrucks oder Objektes hinter `yield` wird zurückgegeben. Allerdings wird der Generator dabei nicht wie eine Funktion beendet, sondern er wird nur unterbrochen wartet nun auf den nächsten Aufruf, um hinter dem `yield` weiterzumachen. Sein gesamter Zustand wird bis zum nächsten Aufruf zwischengespeichert.
- › Der Generator wird erst beendet, wenn entweder der Funktionskörper vollständig abgearbeitet wurde oder der Programmablauf auf eine `return`-Anweisung (ohne Wert) stößt.

Wird in einer normalen Funktion ein `return` erreicht, wird der Kontrollfluss und Funktionsablauf beendet. Außerdem werden alle lokalen Variablen der Funktion wieder freigegeben. Bei einem erneuten Aufruf der Funktion würde Python wieder ganz am Anfang der Funktion beginnen und die komplette Funktion erneut ausführen. `yield` speichert den gesamten Funktionszustand ab und reaktiviert diesen beim nächsten Aufruf.

Betrachten wir als Beispiel den Fibonacci-Generator, ein Generator, der die Fibonacci-Zahlen rasant schnell erzeugt.

Erstellen Sie ein Skript `fibonacci.py` und übernehmen Sie folgendes Listing.



Beispiel - Strings.py (Zeichenketten)

Zeichenketten können indiziert werden, wobei das erste Zeichen einer Zeichenkette den Index 0 hat ("nullbasierte Zählung").

Es gibt keinen speziellen Zeichentyp (wie char in C) - ein Zeichen ist einfach eine Zeichenkette der Länge eins. Implementieren Sie folgendes Listing und beschreiben Sie den Ablauf der while-Schleife

Listing:

```
#!/usr/bin/python3.10
# -----
# Skript: strings.py
# -----
print("Zeichenketten")
print("-----")
# String ausgeben
ausgabe = "Python lernen!"
print(ausgabe)
print("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX")

# Strings sind in Python indexiert
print(ausgabe[0] + " " + ausgabe[7])
print(ausgabe[1] + " " + ausgabe[8])
print(ausgabe[2] + " " + ausgabe[9])
print(ausgabe[3] + " " + ausgabe[10])
print(ausgabe[4] + " " + ausgabe[11])
print(ausgabe[5] + " " + ausgabe[12])

# Letztes Zeichen eines Strings bestimmen mit len()
letztes_zeichen = len(ausgabe) - 1
print(ausgabe[letztes_zeichen])
print("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX")

# String mit while durchlaufen
i = 0
s = ""
while i <= int(letztes_zeichen):
    s = s + ausgabe[i]
    print(s)
    i = i + 1
print("XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX")
```



Programm ggt_python.py

Erstellen Sie ein python-Script `ggt_pythonIhrName.py`, das den grössten gemeinsamen Teiler zweier Zahlen `a` und `b` kurz `ggt(a,b)` mit Hilfe eines Generators berechnet.

Lösung:



Programm fib_rek.py

Erstellen Sie ein python-Skript `fib_rekIhrName.py`, das die ersten 25 Zahlen der Fibonacci-Folge mit Hilfe einer Rekursion (keine Iteration) berechnet.

Lösung:



Python-Module

Einstieg

Wenn man den Python-Interpreter beendet und neu startet, gehen alle vorgenommenen Definitionen (Funktionen und Variablen) verloren. Deshalb bevorzugt man - gegenüber dem interaktiven Modus - den Python-Quelltext in Textdateien zu speichern. Diese können dann in anderen Programmen als Module importiert werden und wiederverwendet werden.

Ein Modul ist eine Datei, die Python-Definitionen und -Anweisungen beinhaltet, der Dateiname mit dem .py-Suffix entspricht dem Namen des Moduls.

Ausgangslage – Betrachten wir ein Beispiel:

Hierzu müssen wir die Möglichkeiten der Funktion `open()` untersuchen: `open()` gibt ein Dateiojekt (file object) zurück und wird meistens mit zwei Argumenten aufgerufen:

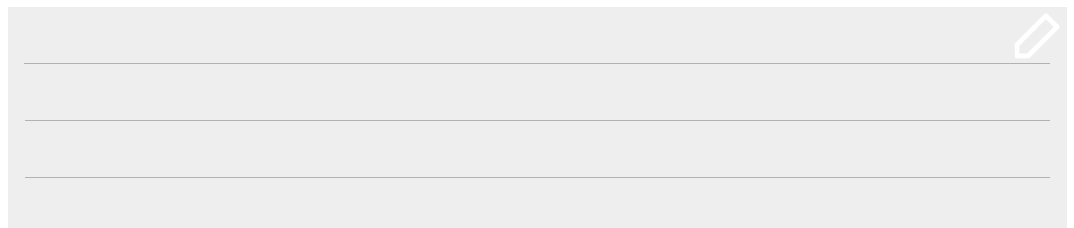
```
open(filename, mode).
```

Das erste Argument ist eine Zeichenkette, die den Dateinamen enthält. Das zweite Argument ist eine andere Zeichenkette mit ein paar Zeichen, die die Art der Benutzung der Datei beschreibt. `mode` kann 'r' sein, wenn die Datei nur gelesen wird, 'w', wenn sie nur geschrieben wird (eine existierende Datei mit demselben Namen wird gelöscht) und 'a' öffnet die Datei zum Anhängen; alle Daten, die in die Datei geschrieben werden, werden automatisch ans Ende angehängt. 'r+' öffnet die Datei zum Lesen und Schreiben. Das `mode`-Argument ist optional, fehlt es, so wird 'r' angenommen.

Die Datei `/proc/cpuinfo` beinhaltet alle relevanten Informationen über die CPU eines Computers.

```
vmadmin@bmLP1:~$ cat /proc/cpuinfo
processor : 0
vendor_id : GenuineIntel
cpu family : 6
model : 69
model name : Intel(R)Core(TM) i7 CPU@ 2.80GHz
stepping : 1
microcode : 0x17
cpu MHz : 2799.503
cache size : 4096 KB
```

Notizen



Die Funktion open()

Anwendung der Funktion `open()` und diverse Ausgabevarianten:

```
#!/usr/bin/python3.10
#-----
# cpufreq1.py bis cpufreq3.py
#-----
# Datei oeffnen und Variable erstellen
cpufreq = open('/proc/cpufreq', 'r')
# Dateiverzeichnispfad
print(cpufreq)
# Ausgabe des Dateiinhaltes
print(cpufreq.read())

... # Ergänzen Sie ...
```

Alternativ zu `cpuinfo.readline()` kann unter Python3.5 auch `next(cpuinfo)` angewandt werden.

Notieren Sie die ersten zwei Elemente der resultierenden Liste `cpuinfo.readlines()` in Listenform `["", ""]`:

```
#!/usr/bin/python3.10
#-----
#  cpuinfo_4.py
#-----
# Datei oeffnen und Variable erstellen
cpuinfo = open('/proc/cpuinfo', 'r')
# Dateiverzeichnispfad
liste = cpuinfo.readlines()
print(liste[4] + liste[7])
```

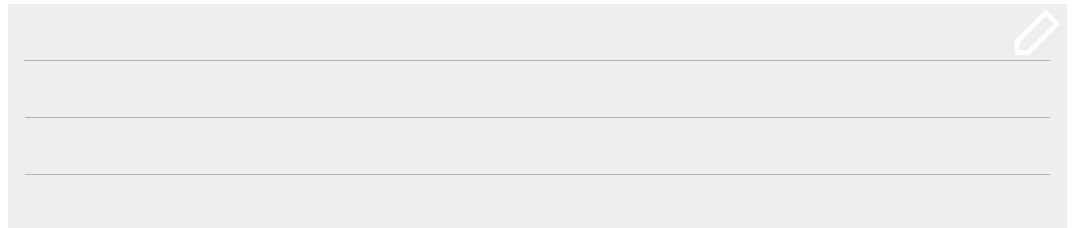
Was gibt das Skript aus?

Antwort / Notiz

Nun können wir eine reduzierte CPU-Ausgabe bauen. Beginnen wir mit einer Funktionsdefinition in einem Skript namens `listcpuinfo.py`:

```
#!/usr/bin/python3.10
#-----
# listcpuinfo.py
#-----
def getcpuinfo():
    cpuinfo = open('/proc/cpuinfo','r')
    liste = cpuinfo.readlines()
    return liste
```

Notizen



Import-Befehl

Mit dem Befehl `import` kann nun das Skript `listcpuinfo.py` in einem anderen Pythonskript wiederverwendet oder in anderen Worten einfach importiert werden. Hierzu muss man dem `import` Befehl lediglich den Skriptnamen als Parameter übergeben:

```
import listcpuinfo
```

Die Funktionalität des Skripts `listcpuinfo.py` steht nun als wiederverwendbares Modul zur Verfügung!

```
#!/usr/bin/python3.10
#-----
# listcpuinfo.py
#-----
def getcpuinfo():
    cpuinfo = open('/proc/cpuinfo','r')
    liste = cpuinfo.readlines()
    return liste
```

Testen wir dies gleich aus und erstellen ein Skript `systeminfo.py`, dass das Modul `listcpuinfo` wiederverwendet:

```
#!/usr/bin/python3.10
#-----
# systeminfo.py
#-----
import listcpuinfo

cpu = listcpuinfo.getcpuinfo()

vendor = cpu[1]
model = cpu[4]
mhz = cpu[6]
cache = cpu[7]

print('-----')
print('CPU-Information')
print('-----')
print(vendor+model+mhz+cache)
```

Die Datei `/proc/meminfo` gibt Aufschluss über den Zustand des Arbeitsspeichers (RAM):

```
cat /proc/meminfo

MemTotal:      6040920 kB
MemFree:       798348 kB
MemAvailable:  4614492 kB
Buffers:       159316 kB
Cached:        3717928 kB
SwapCached:    0 kB
Active:        947144 kB

...

HugePages_Surp: 0
Hugepagesize:   2048 kB
Hugetlb:        0 kB
DirectMap4k:   259968 kB
DirectMap2M:   6031360 kB
DirectMap1G:   2097152 kB
```



Aufgabe - Erstellen Sie ein Modul `listmemoryinfo`

Erstellen Sie ein Modul `listmemoryinfo` mit einer Funktion `getmemoryinfo()` und implementieren Sie das Modul im Skript `systeminfo.py`.

Systeminformationen sollen um folgende Angaben erweitert werden:

- › Total Arbeitsspeicher
- › Freier Arbeitsspeicher
- › Buffers
- › und cached Speicher aus.

Notizen zum Lösungsansatz:



Die `dir()`-Funktion

Die eingebaute Funktion `dir()` wird benutzt, um herauszufinden, welche Namen in einem Modul definiert sind. Es wird eine sortierte Liste von Strings zurückgegeben:

```
>>> import listcpuinfo
>>> dir(listcpuinfo)
['__builtins__', '__cached__', '__doc__', '__file__', '__loader__',
 '__name__', '__package__', '__spec__', 'getcpuinfo']
>>>
```


Zu beachten ist, dass alle Typen von Namen ausgegeben werden: Variablen, Module, Funktionen, etc. Überprüfen Sie diese Aussage, indem Sie das Skript `systeminfo.py` im interaktiven Modus als Modul importieren und die Funktion `dir(systeminfo)` anwenden.

Stimmt diese Aussage? ☐ Wahr ☐ Falsch

`dir()` listet allerdings nicht die Namen der Standardfunktionen von Python und Variablen auf. Falls man diese auflisten will, muss man das Standardmodul `builtins` verwenden:

```
root@bmLP1:/home/vmadmin/M122_Scripts# python3.10
Python 3.10.2 (main, Jan 15 2022, 18:02:07) [GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more
information.
>>> import builtins
>>> dir(builtins)
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException',
 'BlockingIOError', 'BrokenPipeError', 'BufferError', 'BytesWarning',
 'ChildProcessError', 'ConnectionAbortedError', 'ConnectionError',
 'ConnectionRefusedError', 'ConnectionResetError', 'DeprecationWarning',
 'EOFError', 'Ellipsis', 'EncodingWarning', 'EnvironmentError', 'Exception',
 'False', 'FileExistsError', 'FileNotFoundError', 'FloatingPointError',
 'FutureWarning', 'GeneratorExit', 'IOError', 'ImportError', 'ImportWarning',
 'IndentationError', 'IndexError', 'InterruptedError', 'IsADirectoryError',
 'KeyError', 'KeyboardInterrupt', 'LookupError', 'MemoryError',
 'ModuleNotFoundError', 'NameError', 'None', 'NotADirectoryError',
 'NotImplemented', 'NotImplementedError', 'OSError', 'OverflowError',
 'PendingDeprecationWarning', 'PermissionError', 'ProcessLookupError',
 'RecursionError', 'ReferenceError', 'ResourceWarning', 'RuntimeError',
 'RuntimeWarning', 'StopAsyncIteration', 'StopIteration', 'SyntaxError',
 'SyntaxWarning', 'SystemError', 'SystemExit', 'TabError', 'TimeoutError', 'True',
 'TypeError', 'UnboundLocalError', 'UnicodeDecodeError', 'UnicodeEncodeError',
 'UnicodeError', 'UnicodeTranslateError', 'UnicodeWarning', 'UserWarning',
 'ValueError', 'Warning', 'ZeroDivisionError', '__build_class__', '__debug__',
 '__doc__', '__import__', '__loader__', '__name__', '__package__', '__spec__',
 'abs', 'aiter', 'all', 'anext', 'any', 'ascii', 'bin', 'bool', 'breakpoint',
 'bytearray', 'bytes', 'callable', 'chr', 'classmethod', 'compile', 'complex',
 'copyright', 'credits', 'delattr', 'dict', 'dir', 'divmod', 'enumerate', 'eval',
 'exec', 'exit', 'filter', 'float', 'format', 'frozenset', 'getattr', 'globals',
 'hasattr', 'hash', 'help', 'hex', 'id', 'input', 'int', 'isinstance',
 'issubclass', 'iter', 'len', 'license', 'list', 'locals', 'map', 'max',
 'memoryview', 'min', 'next', 'object', 'oct', 'open', 'ord', 'pow', 'print',
 'property', 'quit', 'range', 'repr', 'reversed', 'round', 'set', 'setattr',
 'slice', 'sorted', 'staticmethod', 'str', 'sum', 'super', 'tuple', 'type',
 'vars', 'zip']
>>>
```

Notizen



Python und Argumente

Nicht nur Funktionen, sondern auch einem Python-Skript kann man Argumente (Parameter) übergeben. Ruft man ein Python-Skript auf, werden die Argumente jeweils durch ein Leerzeichen voneinander getrennt hinter dem Skriptnamen aufgeführt. Genau wie es Bash handhabt. Im Python-Skript selbst sind diese Argumente unter der `Listenable` `sys.argv` abrufbar. Der Name des Skriptes (Dateiname) ist unter `sys.argv[0]` erreichbar. Voraussetzung ist, dass Sie das Modul `sys` importieren.

Das Skript (`argumente.py`) listet sämtliche mitgegebenen Argumente auf:

```
#!/usr/bin/python3.10
import sys
for eachArg in sys.argv:
    print(eachArg)
```



Programm Systeminfo.py

Erweitern Sie das Skript `systeminfo.py`, damit es parametrisiert aufgerufen werden kann und die Systeminformationen, die ausgegeben werden, sollen den angegebenen Argumente entsprechen:

Beispiel:

```
>> ./systeminfo.py vendor model mhz cache totalmemory freememory  
buffers cached
```

Notizen

Einlesen von Daten unter Python

Wie in den Aufgaben unter Bash, können auch unter Python sogenannte CSV Dateien eingelesen werden. Dazu stellt Python das Modul csv zur Verfügung. Dieses übernimmt das Einlesen sowie die Auftrennung in Zeilen und Spalten, das ist sehr praktisch!

Beispielcode:

```
#!/usr/bin/python3.10
#-----
# readcsv.py
#-----
import csv

with open ('Data.csv', newline='') as f:
    reader = csv.reader(f, delimiter=";")
    for row in reader:
        print('----- Neue Zeile -----')
        for col in row:
            print('Zelle: ' + str(col))
```

Nutzen Sie zum Testen dieses Skripts die Beispieldatei "Data.csv".

Die offizielle Python Referenz zu csv: <http://docs.python.org/py3k/library/csv>

Notizen:



Objektorientierte Programmierung

Auch unter Python können objektorientierte Konstrukte wie Klassen und Methoden eingesetzt werden. Dadurch können Programme viel eleganter gestaltet werden.

Konstrukt	Beschreibung	Beispiel
class Klassenname	Klassendefinition	class MeineKlasse:
def Methodenname	Methodendefinition (mit 2 Parametern)	def Addition(self, a, b):
Variablenname = Klassenname()	Instanziierung einer neuen Klasse in einer Variablen	mk = MeineKlasse()
Variablenname.Methode	Aufruf einer Methode	mk.Addition(x,y)

```
#!/usr/bin/python3.10
#-----
# MeineKlasse.py
#-----
class MeineKlasse:
    def Addition(Self, a, b):
        return a + b

    def Subtraktion(self, a, b):
        return a - b

def main():
    mk = MeineKlasse()
    x = 20
    y = 10
    print('Addition('+str(x)+'+'+
str(y)+'='+str(mk.Addition(x,y)))
    print('Subtraktion('+str(x)+'-'+
str(y)+'='+str(mk.Subtraktion(x,y)))

if __name__ == '__main__':
    main()
```

Wie nennt man das Konstrukt Addition?

Wozu dient der Parameter self?



Aufgabe – Programmieren Sie eine Klasse

Programmieren Sie eine Klasse, welche diverse Validierungen vornehmen kann und bauen Sie diese in die Skripts der vorherigen Aufgaben ein. Bauen Sie den bisherigen Code so um, dass die neue Klasse zur Validierung verwendet werden kann. Der Code sollte dadurch lesbarer und besser wartbar werden! Speichern Sie die Lösungen unter "EingabeValidierungOO.py" und "DatenmigrationOO.py"

- › Überlegen Sie zuerst, welche Klasse(n) Sie am besten definieren und welche Methoden darin vorkommen sollen.
- › Das Ziel soll sein, den Code möglichst geschickt zu gestalten und zu dokumentieren.

Notizen:

A large rectangular area with horizontal lines for taking notes. A small icon of a paperclip is in the top right corner.

Tkinter

Vorbereitung

```
sudo apt-get install python3.10-tk
```

Beispiel 1

```
#!/usr/bin/python3.10
# -----
# Script: pythongui.py
# Autorin: Susanne Annen
# -----
import tkinter as tk
from math import sqrt

class Application(tk.Frame):
    # Konstruktor
    def __init__(self, title, master=None):
        tk.Frame.__init__(self, master=master)
        self.grid()
        self.master.title(title)
        self.lblAusgabe=tk.Label(self, text="Resultat")
        self.lblAusgabe.grid(row=0, column=0)
        self.btnBeenden=tk.Button(self, text="Quit", fg="red",
command=self.btnBeendenClick)

        self.btnBeenden.grid(row=2, column=0, pady=1)

    def btnBeendenClick(self):
        self.quit()

app=Application('Integer App')

app.mainloop()
```

Programm ausführen:

```
./pythongui_1.py
```

Anzeige:



Weiterführende Aufgaben

Im Begleitbuch (Herdt-Verlag) finden Sie ergänzende und weiterführende Informationen und Übungen zum Thema



Lesen – Stöbern im Begleitbuch «Programmierung mit Python»

- › Nehmen Sie das Begleitbuch HERDT-Programmierung-Python.pdf zur Hand und stöbern Sie darin herum. Was interessiert Sie?
- › Lesen Sie z.B. das Kapitel «Konkrete GUI-Konzepte in Python und das Modul tkinter»
- › Führen Sie Beispiele wie GUI . py aus.
- › Tauschen Sie sich in der Lerngruppe oder Lernpartnerschaft aus.
- › Halten Sie die Erkenntnisse im Arbeitsjournal fest.



Video – Object Oriented Programming (OOP) – For Beginners

- › Schauen Sie sich das Video Python Object Oriented Programming (OOP) - For Beginners
- › https://youtu.be/JeznW_7DIB0



Video – pytsx3 and PyPDF2

- › Schauen Sie sich das Video von Programming Hero an.
- › https://youtu.be/kyZ_5cvrXJI
- › Achten Sie darauf, dass im VM-Player die Soundkarte installiert ist. Mit Ctrl+D kommen Sie zum Menu «Virtual Machine Settings. Ist keine Soundkarte vorhanden, können Sie mit Add... eine hinzufügen. Evtl. müssen Sie de VM neu starten.



Aufgabe – Was können Sie für Ihre Praxisarbeit verwenden?

- › Haben Sie genügend Ideen, ihr Projekt zu erweitern?
- › Tauschen Sie sich in der Stammgruppe aus.

Notizen