

Fortgeschrittene Programmierung



Kapitel 3: Python Module und Dateien

Prof. Dr. Thomas Wieland
WS 2022/2023



Module

- Fassen Funktionen (und ggf. Klassen) in einer separaten .py-Datei zusammen
- Lassen sich importieren (eigene und Bibliothek)
- Beispiel (meinebib.py)

```
konstante = 42
```

```
def printKonst():  
    "Gibt die Konstante aus"  
    print (konstante)
```

```
def tunix():  
    "Macht nix"  
    pass    # Ein Befehl, der gar nichts tut
```

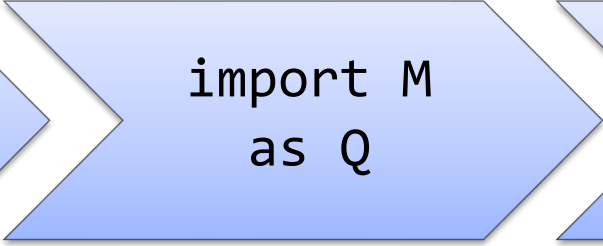
Arten des Imports

Möglichkeiten für Import von Modul M (Datei M.py)



```
import M
```

- Alle Namen aus M in eigenem Namensraum verfügbar, muss mit M.foo() qualifiziert werden



```
import M  
as Q
```

- Modul M wird unter dem Namen Q verfügbar



```
from M  
import *
```

- Alle Namen aus M lokal verfügbar und ohne Verweis auf M zu benutzen



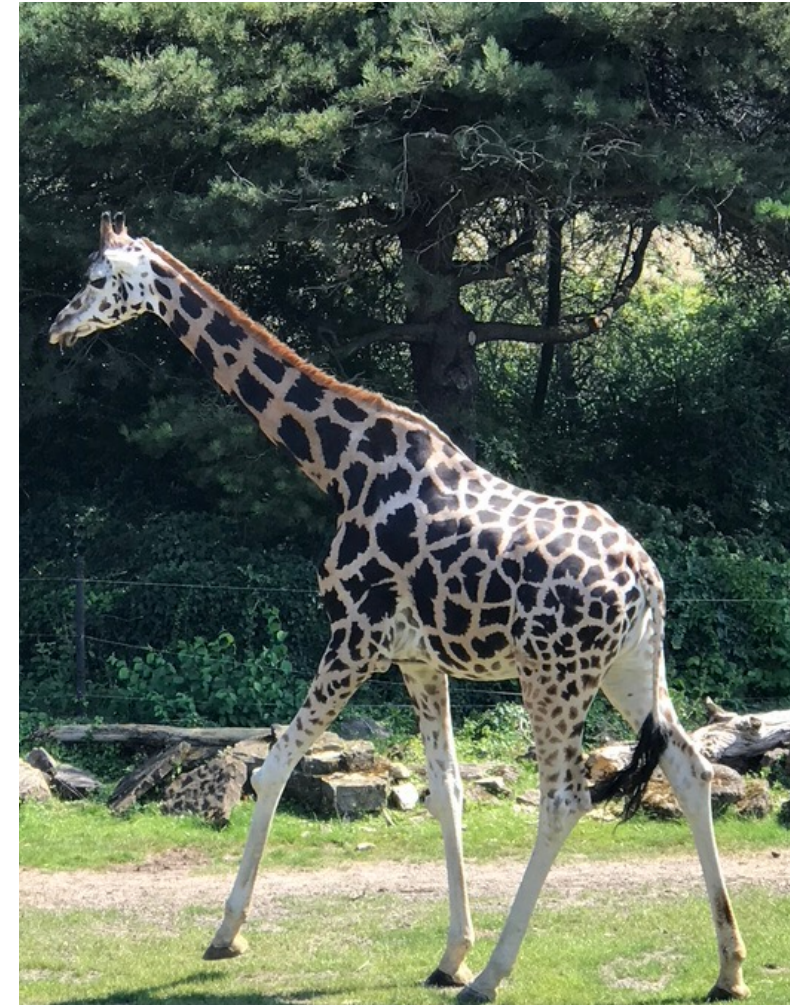
```
from M  
import x
```

- Nur die angegebenen Namen x (und ggf. weitere) sind lokal verfügbar

Mehr über Module

- **Auflistung** der Inhalte
`dir(bib)`
- **Hilfe** abfragen, für Module und einzelne Funktionen
`import(meinebib)`
`help(meinebib)`
`help(meinebib.printKonst)`
- Bestimmung, ob ein Skript gerade **ausgeführt wird oder als Modul** genutzt wird:

```
if __name__ == '__main__':  
    print ("Wird ausgeführt")  
else:  
    print ("als Modul importiert")
```
- Python sucht im **Suchpfad** und im aktuellen Verzeichnis
 - Der Suchpfad kann abgefragt und erweitert werden
`import sys`
`print(sys.path)`
`sys.path.append("folder/to/module")`



Dateien

- Öffnen mit open:

```
datei = open ("testfile.txt")      # read
```

```
datei = open ("testfile.txt", 'r') # read
```

```
datei = open ("testfile.txt", 'w') # write
```

```
datei = open ("testfile.txt", 'a') # append
```

- Für Binärdateien mit einem 'b' nach dem Qualifizierer, also bspw. 'rb'

- Lesen

```
datei.read ()      # komplett einlesen
```

```
datei.read (n)     # n Byte einlesen
```

```
datei.readline ()  # eine Zeile lesen
```

```
datei.readlines () # Liste von Zeilen
```

Weitere Datei-Operationen

- Schreiben

```
datei.write("Neuer Text \n")
```

```
datei.writelines(["1. Zeile\n", "2. Zeile\n"])
```

- Datei schließen

```
datei.close()
```

- Aktuelle Position

```
datei.tell()
```

- Zu einer bestimmten Position gehen

```
datei.seek(pos)
```

- Ausgabepuffer leeren

```
datei.flush()
```

Durch Dateien iterieren

- Dateien sind **sequentielle** Objekte

```
datei = open('meineDatei.txt', 'r')
zahl = 0
for _ in datei:
    zahl += 1
    print(zahl)
datei.close()
```

- Ganze Lexika können mit **pickle** gespeichert werden

```
import pickle
dd = {1: 'q', 2: [1, 2, 3], 'ww': 'Eine Zeichenkette'}
datei = open('meinedatei.bin', 'wb')
pickle.dump(dd, datei)
datei.close()
```

Standard Ein- und Ausgabekanäle

- Modul sys kann auf Standard- (sys.stdin) und Fehlerausgabe (sys.stderr) schreiben, von Standardeingabe (sys.stdin) lesen

```
import sys
sys.stdout.write("Text eingeben:") # = print "...
line = sys.stdin.readline ()
if line == "" :
    sys.stderr.write("Error!\n")
```

- Kann auch vom Betriebssystem abhängig sein

Interaktion mit dem Betriebssystem

- Modul `os` Schnittstelle zum Betriebssystem, z.B.

<code>os.chdir(path)</code>	# Wechselt ins Verzeichnis <code>path</code>
<code>os.link(src, dest)</code>	# Erzeugt Link <code>src</code> auf <code>dest</code>
<code>os.mkdir(path)</code>	# Legt Verzeichnis <code>path</code> an
<code>os.remove(filename)</code>	# Löscht Datei <code>filename</code>
<code>os.system(cmd)</code>	# Führt das Systemkommando <code>cmd</code> aus
<code>os.chmod(path, mode)</code>	# Ändert Zugriffsrechte für <code>path</code>
<code>os.path.exists(path)</code>	# Prüft, ob <code>path</code> existiert

- Und viele andere (siehe Doku)

Zusammenfassung

- Funktionen lassen sich zu **Modulen** zusammenfassen
- Aus Modulen kann man in den **lokalen Namensraum** oder einen eigenen importieren
- Dateien lassen sich **binär oder als Text** öffnen und schreiben
- Beim Öffnen den **Modus** (Lesen, Schreiben, Anhängen) mit angeben
- Man kann durch **Dateiinhalte iterieren**
- Die **Standard-Kanäle** stehen für Bildschirm und Tastatur
- Das Modul `os` erlaubt, **Betriebssystembefehle** auszuführen



Ach Mann, ... ich hätte die
Funktionen auch als
Modul kapseln können?
Das dritte Semester nervt
echt. Nie lernen wir
irgendwas Nützliches ...

Quelle: K. Sierra/B. Bates: Java von Kopf bis Fuß. O'Reilly, 2006