4. Ausnahmen
und Module in Python

Inhalt

- Fehlerbehandlung
  - Spezifikationen
- Modulare Programmierung
- Wie schreibt man Module in Python
  - und warum

Fehlerhafte Programme

- Ein Programm kann aus vielen Gründen unerwünschtes Verhalten zeigen
- Fehler beim Entwurf
- Fehler bei der Programmierung des Entwurfs
  - Algorithmen falsch implementiert
- Umgang mit außergewöhnlichen Situationen
  - Abbruch der Netzwerkverbindung
  - Dateien können nicht gefunden werden
  - fehlerhafte Benutzereingaben

Umgang mit außergewöhnlichen Situationen

- Ausnahmesituationen unterscheiden sich von Programmierfehlern darin, dass man sie nicht (zumindest prinzipiell) von vornherein ausschließen kann

- Immer möglich sind zum Beispiel:
  - unerwartete oder ungültige Eingaben
  - Ein- und Ausgabe-Fehler beim Zugriff auf Dateien oder Netzwerk

Umgang mit außergewöhnlichen Situationen

- Die Erkennung und die Behandlung eines Fehlers muss oft in ganz verschiedenen Teilen des Programms stattfinden.

- Beispiel: Daten sind nicht konsistent (ein ID ist falsch)
  - Erkennung: Save Funktion
  - Behandlung: GUI

Ausnahmen

Eine **Ausnahme** (Exception) ist eine Ausnahmesituation, die sich während der Ausführung eines Programmes einstellt.

- Lässt man diese zu, so stürzt das Programm ab!
- Fängt man diese ab (Ausnahmebehandlung), läuft das Programm weiter!
- Die Auslösung einer Ausnahme bedeutet nicht automatisch, dass der Code einen Fehler enthält

Ausnahmen

Die meisten Programmiersprachen, die Ausnahmen unterstützen, verwenden eine gemeinsame Terminologie und Syntax

- Ausnahmen auslösen
- Ausnahmen abfangen oder behandeln
- Verbreitung
- `try` / `raise` (`throw`) und `except` (`catch`) Keywords

Ausnahmebehandlung

- Ausnahmebehandlung (Exception handling)
  - der Prozess, bei dem Fehlerzustände in einem Programm systematisch behandelt werden

```
try:
   #Code der Ausnahemen auslösen kann
except <ErrorType>:
   #Code der die Situation beherrscht
```

- was hinter try kommt, wird ausgeführt, bis ein Fehler auftritt
- was hinter except kommt, wird nur ausgeführt, wenn im try-statement eine Exception der angegebenen Art aufgetreten ist
- Ja! Man muss die verschiedenen ErrorTypes wissen

Beispiel

- wir haben den folgenden Code

```
def div(a,b):
    return a / b

print(div(1,2))
print(div(0,1))
print(div(1,0))
```

```
0.5
0.0
Traceback (most recent call last):
  File "main.py", line 6, in <module>
    print(div(1,0))
  File "main.py", line 2, in div
    return a / b
ZeroDivisionError: division by zero
```

- wir wollen diese Situation vermiden
- wir konnen if verwenden
  - aber in dem Fall muss man extra Logik in der Funktion reinstecken

Beispiel

- Eleangeter geht es mit Exceptions

```
def div(a,b):
    try:
        r = a / b
    except ZeroDivisionError:
        r = None
    return r

print(div(1,2))
print(div(0,1))
print(div(1,0))
```

None, NoneType, Pass

- `None` ist ein Objekt ohne Wert
- `NoneType` ist der Typ dieses Objektes
- None als Ruckgabewert zeigt das
  - die Funktion retourniet nichts
  - zB eine Suchfunktion hat nichts gefunden
- `pass` ist eine Anweisung, die kein Ergebnis hat
  - nützlich, um Code zu strukturieren
  - kann verwendet werden, um zu zeigen
  - das code wird dort irgendwann geschrieben

```
def add(a,b):
    pass
```

Python Syntax

- Wenn man Ausnahmen abfangen will, muss der Code in einem `try-except` Block enthalten sein
- Ausnahmen werden anhand ihres Typs abgefangen
- Ein try-Block kann einen, mehrere oder alle Ausnahmetypen abfangen
- Das Erstellen von Ausnahmen in unserem Code erfolgt mit dem Schlüsselwort `raise`
- Man kann zusätzliche Argumente (zB eine Fehlermeldung) für jede Ausnahme, die ausgelöst wird, bereitstellen

Ausnahmebehandlung

- Eine Ausnahme kann behandelt werden durch:
  - Die Funktion, bei der die Ausnahme ausgelöst wurde
  - Jede Funktion, die dieser Fuktion aufruft
  - der Python-runtime - dies wird zu einem Abbruch des Programmes führen

- der Satz ”unhandled exception has occurred in your application...” muss uns bekannt sein

- jetzt können wir verstehen, was dort passiert wurde!

Ausnahmebehandlung. Wann?

- Um eine Ausnahmesituation zu signalisieren - die Funktion kann den Vertrag nicht erfüllen
  - z. B. Vorbedingungen sind nicht erfüllt
  - oder eine Situation aufgetreten wurde, in der die Funktion nicht weitergehen kann
    - eine erforderliche Datei wurde nicht gefunden
    - ist nicht zugreifbar
    - usw.

- Um Vorbedingungen durchzusetzen

- Generell sollte man keine Ausnahmen verwenden, um den Programmfluss zu steuern!

Ausnahmebehandlung

- Wie kann man überprüfen, ob ein Parameter einer Funktion von einem bestimmten Typ ist?
- mit `type()/isinstance()`

```
def add(a,b):
    if type(a) == int and typ(b) == int:
    # isinstance(a,int)
    ...
```

- mit Exceptions
  - TypeError/AttributeError

```
1  def my_max(arg):
2      try:
3          return max(arg)
4      except TypeError:
5          return 0
6
7
8  print(my_max([1,2,3]))
9  print(my_max(3))
10
```

- Type Annotations

```
def do_something (n:int)
```

- Hinweis: kann man aber muss man nicht
  - kann ein Indikator für schlechtes Design sein
  - die Spezifikation einer Funktion ist wichtiger

Types

- **SyntaxError** – es ist ein syntaktischer Fehler im Quelltext
- **IOError** – eine Datei existiert nicht, man darf nicht schreiben, die Platte ist voll
- **IndexError** – in einer Sequenz gibt es das angeforderte Element nicht
- **KeyError** – ein Mapping hat den angeforderten Schlüssel nicht
- **ValueError** – eine Operation kann mit diesem Wert nicht durchgeführt werden

Modulare Programmierung

- eine Softwaretechnik, die das Ausmaß erhöht, in dem Software aus unabhängigen, austauschbaren Komponenten besteht
  - Jede solche Komponente erfüllt einen Aspekt innerhalb des Programms und enthält alles, was dazu erforderlich ist.
  - Module in Python

- Module sind daher
  - Unabhängig
  - Austauschbar

- Ermöglichen die Gruppierung von Funktionen

- Ermöglichen die einfachere Bereitstellung von Funktionen

- Helfen bei der Lösung von Namenskonflikten

Modulare Programmierung in Python

- ein Python Modul ist eine Datei die aus Anweisungen und Definitionen besteht
- Name: der Dateiname ist der Modulname mit .py ergaenzt
- Docstring
  - Drei öffnende Anführungszeichen
  - ein kurze Beschreibung in einem Satz
  - eine Leerzeile
  - weitere Bemerkungen
  - abschließend drei Anführungszeichen in einer eigenen Zeile
- Anweisungen
  - ein Modul kann ausführbare Anweisungen wie auch Funktionsdefinitionen enthalten
  - diese Anweisungen dienen der Initialisierung des Moduls
    - keine “globale” Variabilen/Namen
  - sie werden nur dann ausgeführt, wenn das Modul das erste mal importiert wird

die import-Anweisung

- Um ein Modul verwenden zu können, muss es zuerst importiert werden.
- Die Importanweisung:
  a. Durchsucht den globalen Namespace nach dem Modul. Wenn das Modul existiert, ist es bereits importiert und man muss nichts weiter machen
  b. Sucht nach dem Modul.
  c. Im Modul definierte Variablen und Funktionen werden in eine neue Symboltabelle (einen neuen Namespace) eingefügt. Nur der Modulname wird zur aktuellen Symboltabelle hinzugefügt

```
Python 3.8.2 (default, Feb 26 2020, 02:56:10)
> sqrt(2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'sqrt' is not defined
> from math import sqrt
> sqrt(2)
1.4142135623730951
>
```

die import-Anweisung


```
from math import sqrt
sqrt(2)
```

```
import math
math.sqrt(2)
```

```
from math import *
sqrt(2)
```

Beispiel

Modul
`useful_functions.py`

mit folgenden Funktionen

- `add(a,b)`
- `mul(a,b)`
- `sub(a,b)`

der Modul-Suchpfad

```
import spam
```

Wenn das Modul *spam* importiert wird, sucht der Interpreter nach einer Datei mit Namen *spam.py*

- im aktuellen Verzeichnis
  - wo das Skript gespeichrt ist
- in der Liste der Verzeichnisse, die durch die Umgebungsvariable **PYTHONPATH** spezifiziert wird
- in der Liste der Verzeichnisse, die durch die Umgebungsvariable **PYTHONHOME** spezifiziert wird.
  - Abhangig von der Installation
  - /usr/local/lib/python (Unix)
- wenn das Modul nicht gefunden wurde, wird die ImportError Ausnahme ausgelöst

Packages

- Packages sind eine Möglichkeit, um Modulen zu strukturieren
- A.B zeigt das B ein Modul in Package A ist
  - Auf dem Laufwerk stellen Folders Packages dar
    - B.py befindet sich in einem Folder A.
- Für den Import von Paketen gelten die gleichen Regeln wie für Module
- Jeder Folder, der ein Package darstellt, enthält eine __init__.py Datei
- __init__.py kann leer sein oder Itialisierungscode enthalten.

Labor 3+

Man muss Module für folgende erstellen:

- Benutzeroberfläche
  - Funktionen für die Benutzerinteraktion.
  - Enthält Eingabe- und Ausgabefunktionen sowie Funktionen für Datenüberprüfung

- Businesslogik
  - Enthält Funktionen, die zum Implementieren von Programmfunktionen erforderlich sind

Beispiel

Lass un Python-Programm schreiben, welches eine Liste von Studenten verwaltet. Jeder Student hat als Attribute Name, Universität.

Funktionalität

- Studenten anlegen und finden
- Studentinfo ausgeben

Übung

Neue Funktionalität

Sort: Studenten nach Name sortieren