FH CAMPUS WIEN
UNIVERSITY OF APPLIED SCIENCES

DIE HOCHSCHULE, DIE MEHR KANN.

ILV Einführung in das Programmieren

Exceptions

Mohamed Goha, BSc. | WS 2024/25

APPLIED LIFE SCIENCES | MASTERSTUDIENGANG BIOINFORMATIK

Motivation

- Beim Programmieren kommt es manchmal zu Problemen/Fehlern, die das weitere Ausführen unseres Programms stoppen:
  - falscher Datentyp, Anwendungsfall, der nicht beachtet wurde, etc.
- Richtiger Umgang mit einem solchen Problem (error handling) kann:
  - Dem User klare Informationen dazu geben, was schiefgegangen ist
  - Wichtige Operationen durchführen, bevor Programm beendet wird (z.B. offene files schließen; Fehler dokumentieren (“loggen”); Modell, das gerade trainiert wird, speichern
  - Fehler beheben und Programm weiter ausführen (Wenn es Sinn macht)

Exceptions

- Im Falle eines Errors wirft Python (und viele andere Sprachen) eine **exception**
- Eine exception enthält Informationen darüber, was schiefgegangen ist.
- Es gibt verschiedene Arten von exceptions
- Es ist möglich, eine eigene exception zu erstellen (mehr dazu in VO “Objektorientiertes Programmieren”)

Exceptions

> Exceptions können gefangen (“caught”) werden, um damit umzugehen
> Wird eine exception geworfen (“raised”) springt die Programmausführung zu der Stelle, an der die exception gefangen wird.
> Es ist auch möglich, manuell exceptions zu werfen (“raise an exception”)

Exceptions: Beispiel

```
def func1():
    try:
        some_dict = {}
        print(some_dict["a_key"]) # raises KeyError!
    except KeyError as ex:
        # do something
        print("KeyError was caught!")
        # possibly reraise error
        raise KeyError
func1()
```

Predefined Python Exceptions

- **TypeError** (inkompatible Datentypen)
- **ValueError** (richtiger Datentyp aber inkorrekter Wert)
- **IndexError** (Sequenz-Index out of range)
- **KeyError** (key nicht im dictionary)
- **ZeroDivisionError**
- **FileNotFoundError**
- **ModuleNotFoundError**

Vollständige Sammlung:

https://docs.python.org/3/library/exceptions.html#bltin-exceptions

Exceptions: Hierarchie

- Nehmen wir an, zwei Funktionen (func1, func2) zu haben, wobei func2 die Funktion func1 aufruft.
- Ein KeyError wird in func1 geworfen
  - Zuerst wird geprüft, ob dieser Error in func1 gefangen wird
  - Falls nicht, wird geprüft, ob er in func2 gefangen wird
  - Falls nicht, wird geprüft, ob er in der Stelle gefangen wird, an der func1 aufgerufen wird
  - Falls nicht, wird das Programm mit der Exception terminiert.

Hierarchie: Beispiel

```
def func1():
    some_dict = {}
    print(some_dict["a_key"]) # raises KeyError!
    # KeyError not caught in func1 - will it be raised?
def func2():
    try:
        func1()
    except KeyError:
        print("Error from func1 was caught in func2!")
# KeyError is not raised, because it is caught in func2!
func2()
```

✓ 0.0s

```
Error from func1 was caught in func2!
```

Conditional Code Execution

> Mit dem “else” keyword können Operationen definiert werden, die ausgeführt werden sollen, falls keine der Exceptions geworfen wird.

```
def func1():
    try:
        some_dict = {"a_key" : 0}
        print(some_dict["a_key"]) # does not raise KeyError
    except KeyError as ex:
        # do something
        print("KeyError was caught!")
        # possibly reraise error
        raise ex
    else:
        print("No exception was raised - everything is okay!")
func1()
```

✓ 0.0s

```
0
No exception was raised - everything is okay!
```

Unconditional Code Execution

- Mit dem “finally” keyword können Operationen definiert werden, die immer ausgeführt werden, egal ob eine exception geworfen wird oder nicht.

```
def func1():
    try:
        some_dict = {}
        print(some_dict["a_key"]) # raises KeyError
    except KeyError as ex:
        # do something
        print("KeyError was caught!")
        # possibly reraise error
        raise ex
    finally:
        print("This is always executed!")
func1()
```

0.0s

```
KeyError was caught!
This is always executed!

---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
```