8 – Exceptions & API-documentation

Bálint Aradi

Course: Scientific Programming / Wissenchaftliches Programmieren (Python)

Universität Bremen

Bremen Center for Computational Materials Science BCCMS

https://www.bccms.uni-bremen.de/cms/people/b-aradi/wissen-progr/python/2022

Outline

- Exceptions
- API-Documentation via Sphinx

Exceptions

Exceptions

- Exceptions signalize errors during code executiong
- If an unexpected error happens which Python can not (or does not want to) handle, an exception is raised

```
mystr = "ab"
int(mystr)
```

Where did the error occur?

```
Traceback (most recent call last):
  File "test.py", line 2, in <module>
    int(mystr)
ValueError: invalid literal for int() with base 10: 'ab'
```

Exception class

Error message

- Exceptions are part of a class hierarchy
- Exception class indicates the kind of error occurred.

Call stack trace

- If the exception is raised within a function, the exception contains the entire call stack trace information (how this point of code execution has been reached)

```
def convert_to_int(string):
    return int(string)

convert_to_int("a")
```

```
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    convert_to_int("a")
  File "test.py", line 2, in convert_to_int
    return int(string)
ValueError: invalid literal for int() with base 10: 'a'
```

- The most recent call is shown last

Handling exception

- A robust program should handle exceptions which can be expected

```
fname = "missing"
fp = open(fname, "r")
```

```
Traceback (most recent call last):
  File "test.py", line 2 ...
    fp = open(fname, "r")
FileNotFoundError: [Errno 2] ...
```

```
try:
    fp = open(fname, "r")
except FileNotFoundError:
    print("Can not open file {}".format(fname))
    print("I will use defaults instead")
    txt = "default text"
else:
    txt = fp.read()
    fp.close()
    print("File {} succesfully read".format(fname))
```

Handling exception

- Exception can be caught and processed with the **try … except …** clause

```
try:
    fp = open(fname, "r")
except FileNotFoundError:
    print("Can not open file {}".format(fname))
    print("I will use defaults instead")
    txt = "default text"
else:
    txt = fp.read()
    fp.close()
    print("File {} succesfully read".format(fname))
```

- If any of the statements in the try block raises an exception, it will be compared against the exceptions in the except clauses
- The code in the first matching except block will be executed and then code execution contains after the try except clause
- The optional else block is executed, if no exception occured

Handling exceptions

- The except clause can obtain the exception instance as variable for further inspection

Instance variable

Exception as string (error message)

```
try:
    fp = open(fname, "r")
except FileNotFoundError as exc:
    print("Input file {} not found".format(fname))
    print("Exception as string: {}".format(exc))
    print("Exception arguments:", exc.args)
else:
    print("File {} read".format(fname))
```

Exception arguments

```
Input file failing not found
Exception as string: [Errno 2] No such file or
directory: 'failing'
Exception arguments: (2, 'No such file or directory')
```

- The number and type of the exception arguments are exception dependent

Handling exceptions

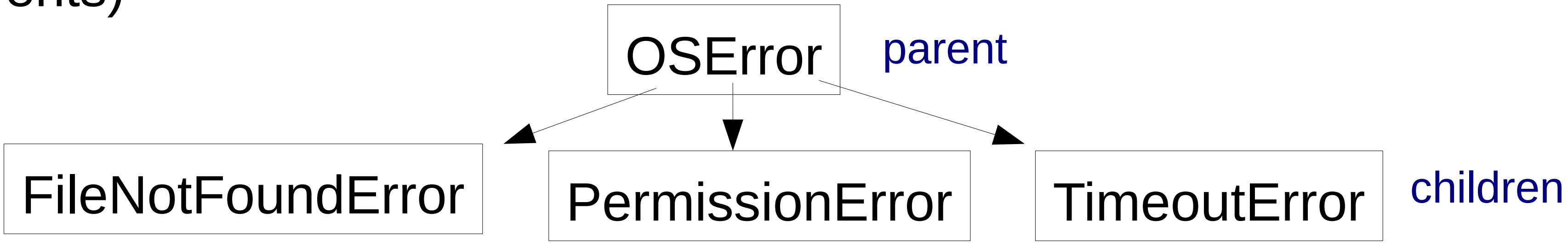
- A **try … except …** construct may contain several except claues
- If an exception is raised, the first **except** clause with a matching exception will be executed

```
try:
    fp = open(fname, "r")
except FileNotFoundError:
    print("Input file {} not found".format(fname))
except PermissionError:
    print("No read permission for input file
{}".format(fname))a
```

- There will be maximally one **except** clause executed.

Exception class hierarchy

- Exceptions are organized in a **class hierarchy**
- More specific exeptions (children) inherit from more general exceptions (parents)



- If a more generic exeption appears in an except clause, it handles the exception itself or any of its descendants lower in the class hierarchy

```
try:
    fp = open(fname, "r")
except OSError:
    print("Could not open file")
    print("File is either not present or it exists
but has wrong permissions")
```

Exiting gracefully via sys.exit()

- A script can be exited via **sys.exit()**
- The argument of exit is given to the operating system and can be used there to take action depending on the exit code

```
import sys

try:
    with open('input.dat', 'r') as fp:
        content = fp.read()
except OSError:
    print("Could not read input file")
    print("Exiting...")
    sys.exit(1)
```

Raising an exception

- Your library can signalize irrecoverable errors by raising exceptions
- You have to pass an initialized exception to the raise command
- You can raise Pythons built-in exceptions, if appropriate.
- Most exceptions in Python accept the error message as only argument.

```
if abs(diagelem) < TOL:
    msg = "Singular matrix"
    raise ValueException(msg)
```

- It is also possible to define your own exceptions via inheritance:

```
class LinAlgError(Exception):
    pass
```

User exceptions should be typically derived from the Exception class

Testing on exceptions in py.test

- Pytest can test, whether an exepction is raised.
- Code which is supposed to raise an exception must be embedded in a context manager (with construct)
- The context is created by the **pytest.raises()** function, which takes the exception type it should look for

```
def test_passes_if_exception_is_raised():
    with pytest.raises(ValueError):
        gaussian_eliminate(aa_singular, bb)
```

- The test passes, if the specified exception is raised during the execution of the context, otherwise it fails.

Be sure to test only for the single **specific exception**, you **expect** to be raised in a given unit test!

API-documentation

```
sudo apt install python3-sphinx make
```

Additionally, if you wish to use LaTeX:

```
sudo apt install texlive-latex-recommended
texlive-latex-extra latexmk
```

If you use conda:

```
conda install sphinx make
```

API-documentation

Application Programming Interface (API)

- All public routines of your project
- They could be called by other projects / scripts by importing modules from this project (**reusability!**)

API-documentation

- Description of the purpose and input/output arguments of the API
- In Python the module/function doc-strings should be used to contain the API-documentation

Extracting API-documentation

- Documentation is extracted from the source code
- Generated documentation can be inspected without looking into the code (e.g. HTML-pages, PDF-document, etc.)
- Modules can be reused without knowing the internal code details

Extracting API documentation

Sphinx documentation system

- Can be used to create simple code related documents (e.g. user manual, reference manuel, etc.)
- Can be used to extract API-documentation from doc-strings
- De-facto standard tool in the Python-world (all documentation on python.org is written using Sphinx)
- It uses the **reStructured Text** (RST) format

ReStructured Text in a nutshell (1)

- HTML/TeX-like formatting language using mostly picturesque notation instead of commands

```
************
Some chapter
************
```

```
First section
=============
```

```
First subsection
----------------
```

Decorator must be as long as title

```
This is *emphasized (italic)* and **bold**.

Here we use a TeX equation: :math:`E = mc^2`
```

```
Bulleted list is easy:

* First bullet item

* Second bullet item
```

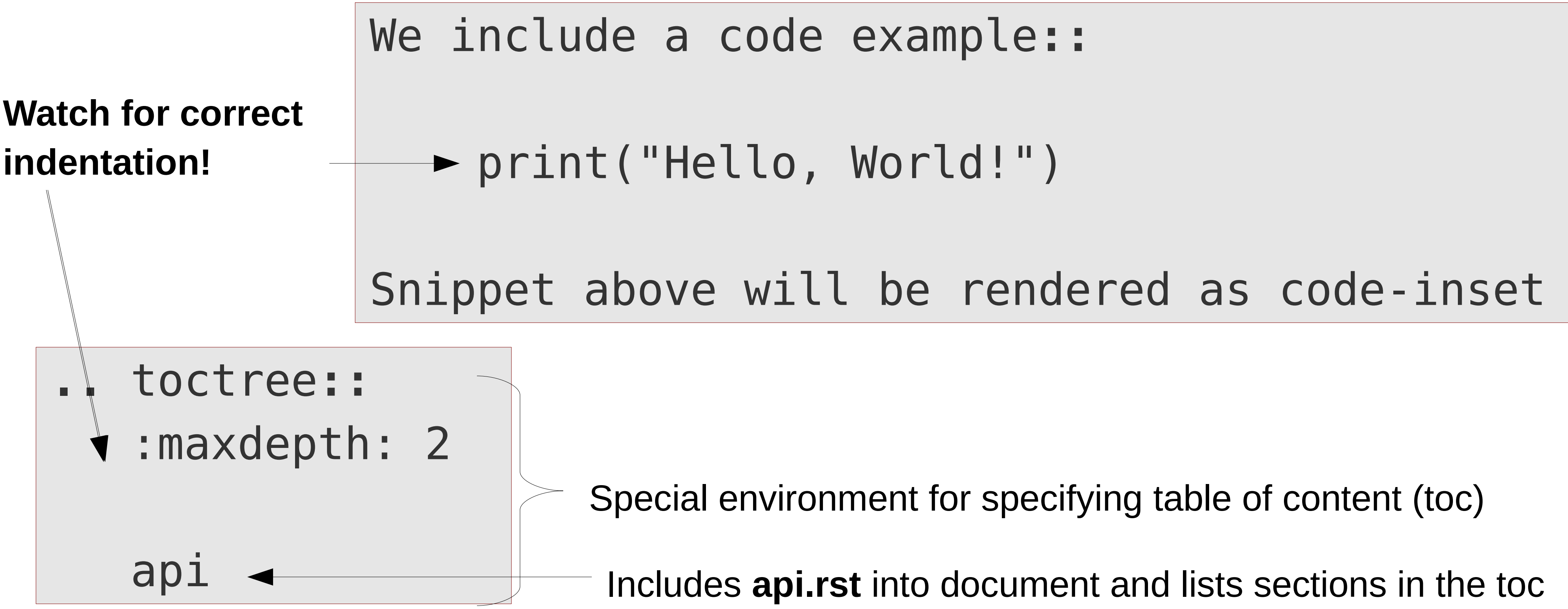
```
Enumerated list:

1. First enumerated item

2. Second enumerated item
```

ReStructured Text in a nutshell (2)

- Similar to Python, indentation is part of the ReST-language semantics



```
We include a code example::

    print("Hello, World!")

Snippet above will be rendered as code-inset
```

Watch for correct indentation!

```
.. toctree::
   :maxdepth: 2

   api
```

Special environment for specifying table of content (toc)

Includes **api.rst** into document and lists sections in the toc

- Read the documentation for all available feature of ReST (quite powerful)

See also

- Quick reStructuredText
- The reStructuredText Cheat Sheet
- A ReStructuredText Primer

Extracting API documentation

- Create a subfolder **docs/** in the project directory
- Set up a sphinx documentation project in it

```
mkdir docs
cd docs
sphinx-quickstart
```

Sets up sphinx-project with default settings

Take default value everywhere

Fill out project details (no default)

```
Project name: Linsolver
Author name(s): YOUR NAME
Project version: 0.1
```

- Edit generated **conf.py** file

~ line 22

```
sys.path.insert(0, os.path.abspath('..'))
```

Ensures that sphinx finds Python module files in parent folder when extracting API-documentation

~ line 32

```
extensions = [
    'sphinx.ext.autodoc',
    'sphinx.ext.napoleon',
    'sphinx.ext.mathjax',
]
```

Use necessary Sphinx-extensions

Automated API-extraction
Doc-strings in Google/Numpy-format
Render TeX in HTML with MathJax

Extracting API documentation

- Edit generated file index.rst and create new file api.rst in the **docs** folder:

index.rst

```
#########
Linsolver
#########

.. toctree::
   :maxdepth: 2

   api
```

api.rst

```
*************
Linsolver API
*************

.. automodule:: solvers
   :members:
```

Generate automatic documentation for all members of the solvers module

- Compile documentation into HTML-format

```
make html
```

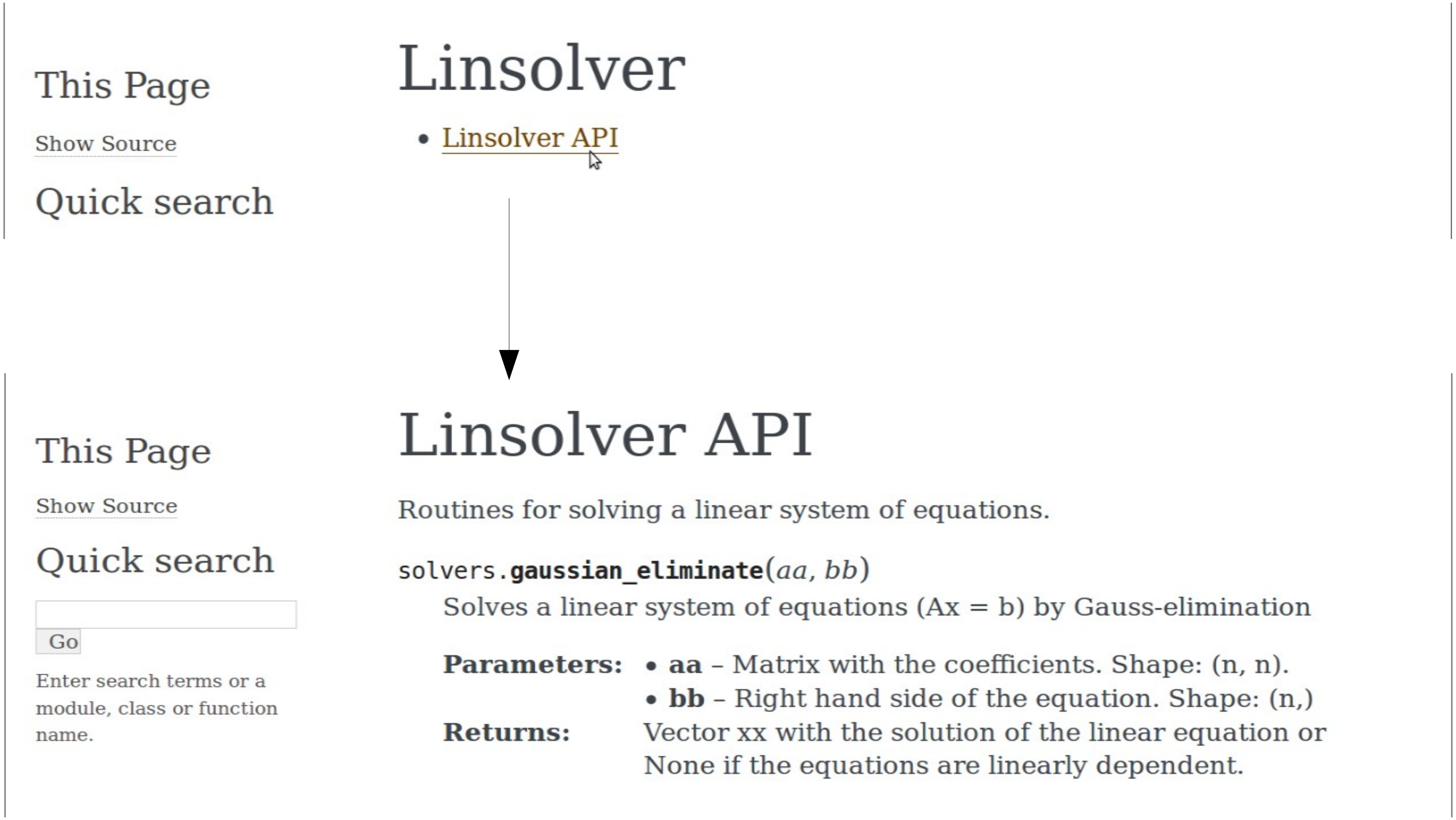
```
Build finished. The HTML pages are in _build/html.
```

Make is a standard Unix tool for coordinating compilation order of interdependent components in a project (e.g. parts of the Sphinx-document)

Visualizing API documentation

- Open the **_build/index.html** file in a web-browser

```
firefox _build/index.html
```



This Page

Show Source

Quick search

Linsolver

- Linsolver API

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

Linsolver API

Routines for solving a linear system of equations.

`solvers.`**`gaussian_eliminate`**(*aa*, *bb*)

Solves a linear system of equations (Ax = b) by Gauss-elimination

Parameters:
- **aa** – Matrix with the coefficients. Shape: (n, n).
- **bb** – Right hand side of the equation. Shape: (n,)

Returns: Vector xx with the solution of the linear equation or None if the equations are linearly dependent.

Some Sphinx-notes

- Sphinx is optimal for small and middle size documents, where type setting is only moderately complicated
- Sphinx has several output format beside html (LaTeX, PDF, etc.)

```
make latexpdf
```

Only works if LaTeX is installed!

- Put the Sphinx source and configuration files of your project under **version control**, but not the **_build** folder

```
cd docs
git add api.rst conf.py index.rst make.bat Makefile
_static/ _templates/
```

- Add the Sphinx build folder to the projects **.gitignore** file

```
.spyderproject
__pycache__
docs/_build
```