Exceptions & API documentation

Bálint Aradi

Course: Scientific Programming / Wissenchaftliches Programmieren (Python)

https://www.bccms.uni-bremen.de/people/b-aradi/wissen-progr/python/2023

Outline

- Exceptions
- Extracting API documentation via Sphinx

You might need to install some Conda packages to try the examples in this lecture:

```
conda install sphinx
```

On Linux make sure, that the “make” tool is installed on your system (probably it is already)

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)
```

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

Where did the error occur?

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 exceptions

- A robust program should handle exceptions which can be expected

```
fname = "missing_file"
with open(fname, "r") as fp:
    txt = fp.read()
```

```
Traceback (most recent call last):
  File "...", line 2, in <module>
    with open(fname, "r") as fp:
FileNotFoundError: [Errno 2] No such file or directory: 'missing_file'
```

```
try:
    with open(fname, "r") as fp:
        txt = fp.read()
except FileNotFoundError:
    print(f"Could not open {fname}")
    # Recover here or exit
```

```
Could not open missing_file
```

Handling exceptions

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

```
try:
    fp = open(fname, "r")
except FileNotFoundError:
    print(f"Could not open file {fname}, using default content")
    txt = "default text"
else:
    txt = fp.read()
    fp.close()
    print("File {fname} succesfully read".format(fname))
```

- If exception is raised by any statement in the try block, it is compared with the exceptions in the except clauses
- Block of first matching exception will be executed
- If no exception matches, program stops due to unhandled exception
- 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(f"Input file {fname} not found")
    print(f"Exception as string: {exc}")
    print("Exception arguments:", exc.args)
else:
    print("File {} read".format(fname))
```

Exception arguments

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

- Number and type of exception arguments are exception dependent

Handling exceptions

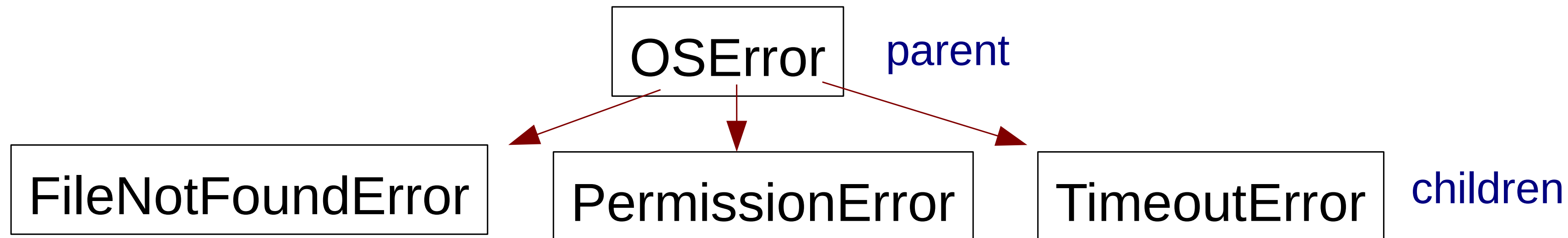
- A **try … except …** construct may contain several except claues
- If an exception is raised, the **first** matching except clause will be executed

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

- 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 an 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 not present or present but not readable")
```

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)
```

- Only the highest level main program/scriopt should call exit, never functions in a module

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):
    """Signalizes linear algebra problems (e.g. linear dependence)"""
```

User exceptions should be derived from the Exception class

Testing exception in pytest

- Pytest can test, whether an exception had been 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 had been 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

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 independent from source code (e.g. HTML-pages)
- Modules can be reused without knowing the internal code details

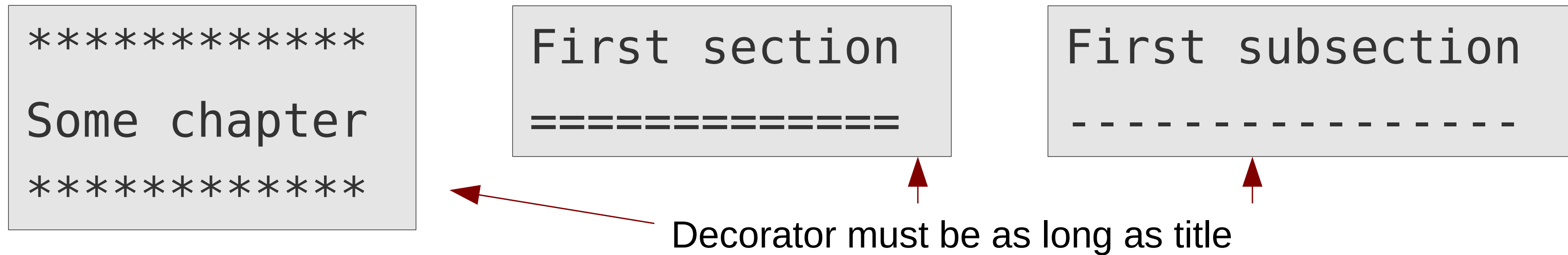
Extracting API documentation

Sphinx documentation system

- Suitable for simple code related documents (e.g. user manual, reference manual, 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




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

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

```
Bulleted list:

* 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

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

   api
```

Watch out for correct indentation!

```
We include a code example::

    print("Hello, World!")

Snippet above will be rendered as code
```

Special environment for specifying table of content (toc)

Includes **api.rst** into the document and lists its 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
```

Take default value wherever possible

- Edit generated **conf.py** file

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

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

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

Automated API-extraction

Doc-strings in Google/Numpy-format

Render TeX in HTML with MathJax

Activate some extensions

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:
```

- Extract documentation and convert to HTML-format

Linux

```
make html
```

Windows

```
./make.bat html
```

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

Visualizing API documentation

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

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

Linux

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 not too complicated
- Sphinx has several output format beside html (LaTeX, PDF, etc.)
- 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

.gitignore

```
__pycache__
docs/_build
```

Have fun!