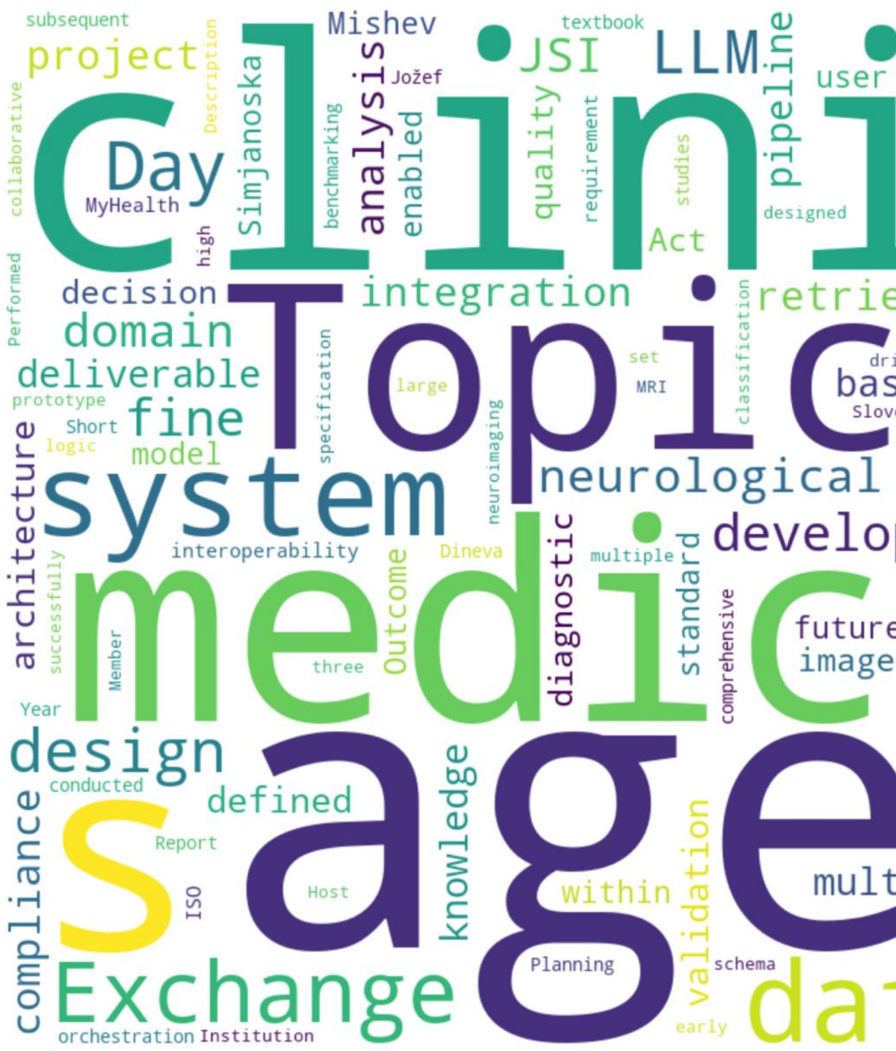
SOFTWARE BEST PRACTICES

Primož Kocuvan
7. October 2025

Jožef Stefan Institute
Ljubljana, Slovenia



Divided into 3 parts:

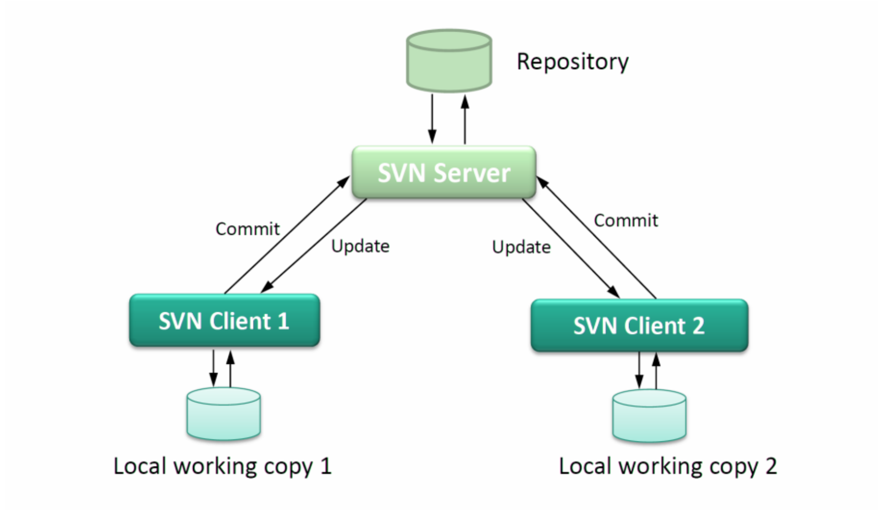
- Git version control - Fundamentals
- Sphinx Python package – Creating documentation
- Coding standards – Python

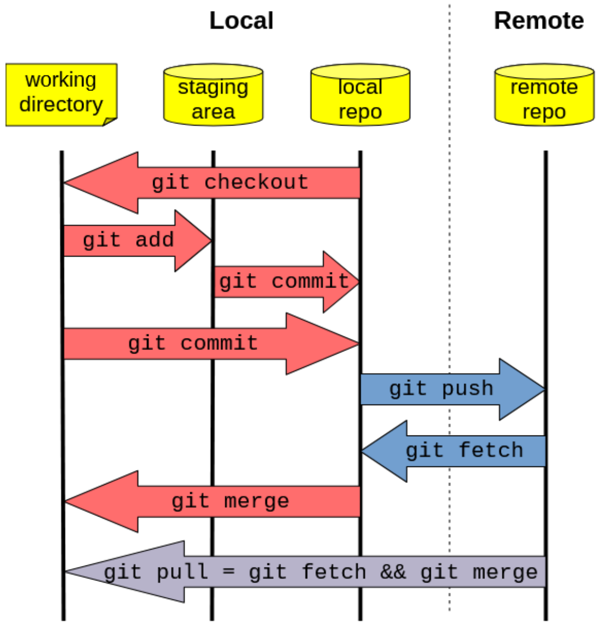
TM

GIT VERSION CONTROL FUNDAMENTALS

Version control systems - types

- Apache Subversion - centralized
- Git version control - distributed

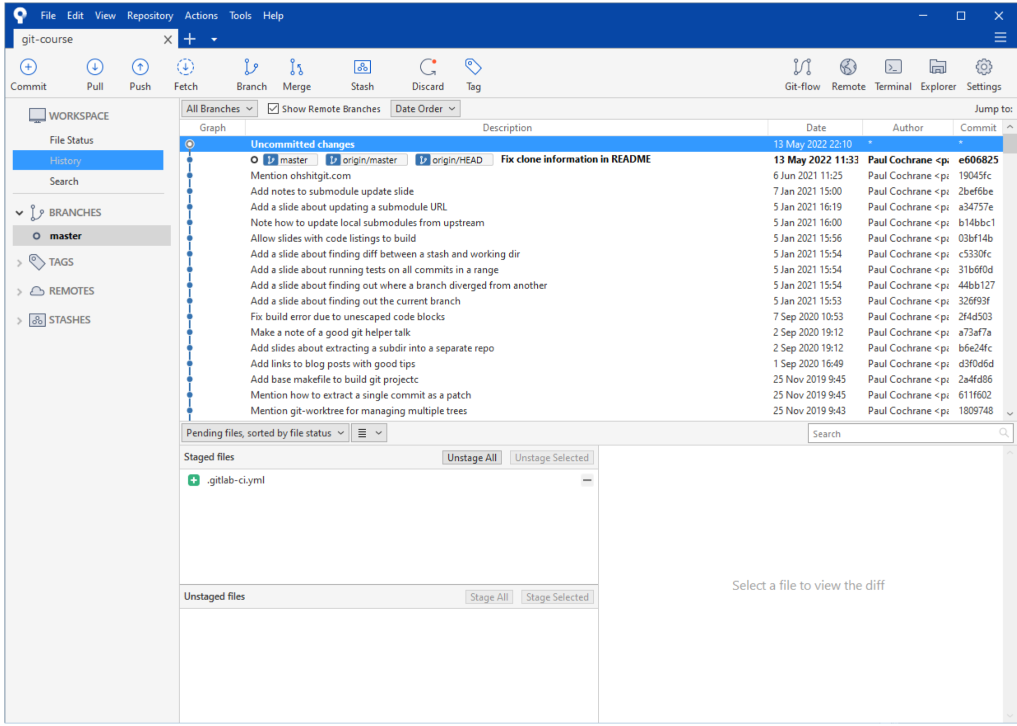




Text commands:

- In this introduction
- we will be covering CLI.

- TortoiseGIT
- Sourcetree (on the right)



File Edit View Repository Actions Tools Help
git-course
Commit Pull Push Fetch Branch Merge Stash Discard Tag Git-flow Remote Terminal Explorer Settings
WORKSPACE
File Status
History
Search
BRANCHES
master
TAGS
REMOTES
STASHES
All Branches
Show Remote Branches
Date Order
Jump to:

Graph	Description	Date	Author	Commit
	Uncommitted changes	13 May 2022 22:10	*	*
	master origin/master origin/HEAD **Fix clone information in README**	**13 May 2022 11:33**	**Paul Cochrane <pa**	**e606825**
	Mention ohshitgit.com	6 Jun 2021 11:25	Paul Cochrane <pa	19045fc
	Add notes to submodule update slide	7 Jan 2021 15:00	Paul Cochrane <pa	2bef6be
	Add a slide about updating a submodule URL	5 Jan 2021 16:19	Paul Cochrane <pa	a34757e
	Note how to update local submodules from upstream	5 Jan 2021 16:00	Paul Cochrane <pa	b14bbc1
	Allow slides with code listings to build	5 Jan 2021 15:56	Paul Cochrane <pa	03bf14b
	Add a slide about finding diff between a stash and working dir	5 Jan 2021 15:54	Paul Cochrane <pa	c5330fc
	Add a slide about running tests on all commits in a range	5 Jan 2021 15:54	Paul Cochrane <pa	31b6f0d
	Add a slide about finding out where a branch diverged from another	5 Jan 2021 15:54	Paul Cochrane <pa	44bb127
	Add a slide about finding out the current branch	5 Jan 2021 15:53	Paul Cochrane <pa	326f93f
	Fix build error due to unescaped code blocks	7 Sep 2020 10:53	Paul Cochrane <pa	2f4d503
	Make a note of a good git helper talk	2 Sep 2020 19:12	Paul Cochrane <pa	a73af7a
	Add slides about extracting a subdir into a separate repo	2 Sep 2020 19:12	Paul Cochrane <pa	b6e24fc
	Add links to blog posts with good tips	1 Sep 2020 16:49	Paul Cochrane <pa	d3f0d6d
	Add base makefile to build git projectc	25 Nov 2019 9:45	Paul Cochrane <pa	2a4fd86
	Mention how to extract a single commit as a patch	25 Nov 2019 9:45	Paul Cochrane <pa	611f602
	Mention git-worktree for managing multiple trees	25 Nov 2019 9:43	Paul Cochrane <pa	1809748

Git configure

- After installing Git, we need to configure email and username:

```
git config --global user.name "primozkocuvan"
git config --global user.email "primoz.kocuvan@ijs.si"
```

Git init – initializing repository

Linux commands:

```
mkdir HomeDoctor
cd HomeDoctor
git init
```

It creates a hidden folder .git inside the local repository (folder)

Tracking and untracking

- We can now create files in the folder.
- Note:
    - At the beginning these files are untracked

Staging:

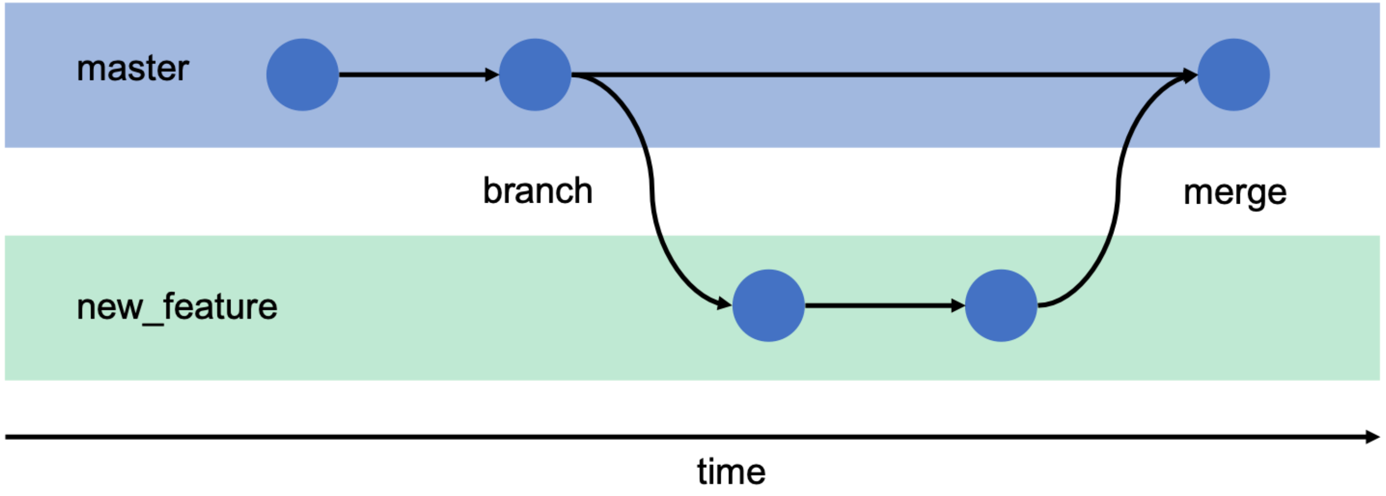
git add - Stage a file
git add --all - Stage all changes
git status - See what is staged
git restore --staged <file> - Unstage a file

Commiting:

```
git commit -m "message" - Commit staged changes
git commit -a -m "message" - Commit all tracked changes
git log - See commit history
```

Branching and merging:

```
git branch new_feature
git checkout new_feature
// We implement new feature and commit
git checkout master
git merge new_feature
```



CI/CD:

- There are many CI/CD tools:
  - Github Actions
  - Gitlab CI/CD
  - Jenkins

Gitlab CI/CD:

```
stages:
  - test
  - build
  - deploy

tests:
  stage: test
  image: python:3.11
  before_script:
    - apt-get update && apt-get install make
  script:
    - make homedoctor_tests
```

Gitlab CI/CD:

```
build:
  stage: build
  image: docker:20.10.16
before_script:
    - docker login -u primoz.kocuvan -p somepassword1234
  script:
    - docker build -t $IMAGE_NAME:$IMAGE_TAG .
    - docker push $IMAGE_NAME:$IMAGE_TAG
```

Gitlab CI/CD:

```
deploy:
  stage: deploy
  before_script:
    - chmod 400 $SSH_KEY
  script:
    - ssh -o StrictHostKeyChecking=no -i $SSH_KEY dis@home-doctor.ijs.si"
        docker login -u $REGISTRY_USER -p $REGISTRY_PASS &&
        docker run -d $IMAGE_NAME:$IMAGE_TAG"
```

AUTOMATIC CREATION OF DOCUMENTATION PYTHON PACKAGE

What is Python Sphinx

- Sphinx is an automatic documentation generator.
- It is a defacto standard for generating Python documentation.
- We install it by using the pip python package manager:
- - pip install sphinx

Restructured text - examples

By default it uses Restructured Text (RST) as a plain-text markup language for writing documentation.

It is similar to the MD (Markdown language), for example:

one asterisk: *text* for emphasis (italics),
two asterisks: **text** for strong emphasis (boldface), and
backquotes: ``text`` for code samples.

Restructured text – examples lists

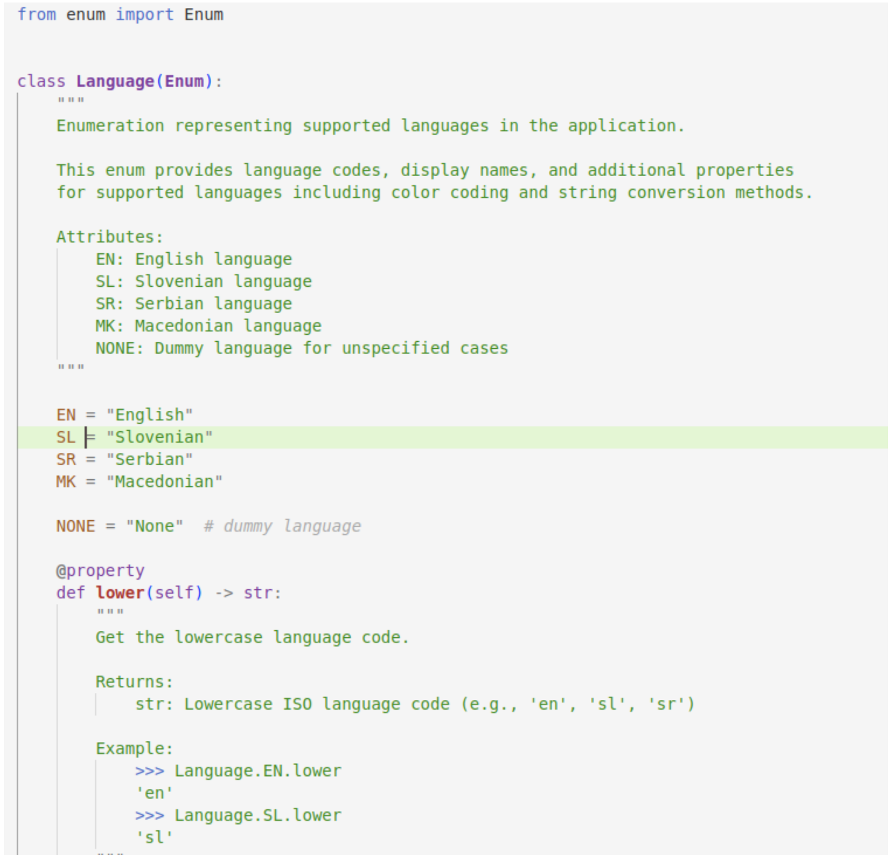
* This is a bulleted list.
* It has two items, the second
  item uses two lines.

1. This is a numbered list.
2. It has two items too.

Restructured text – image and table

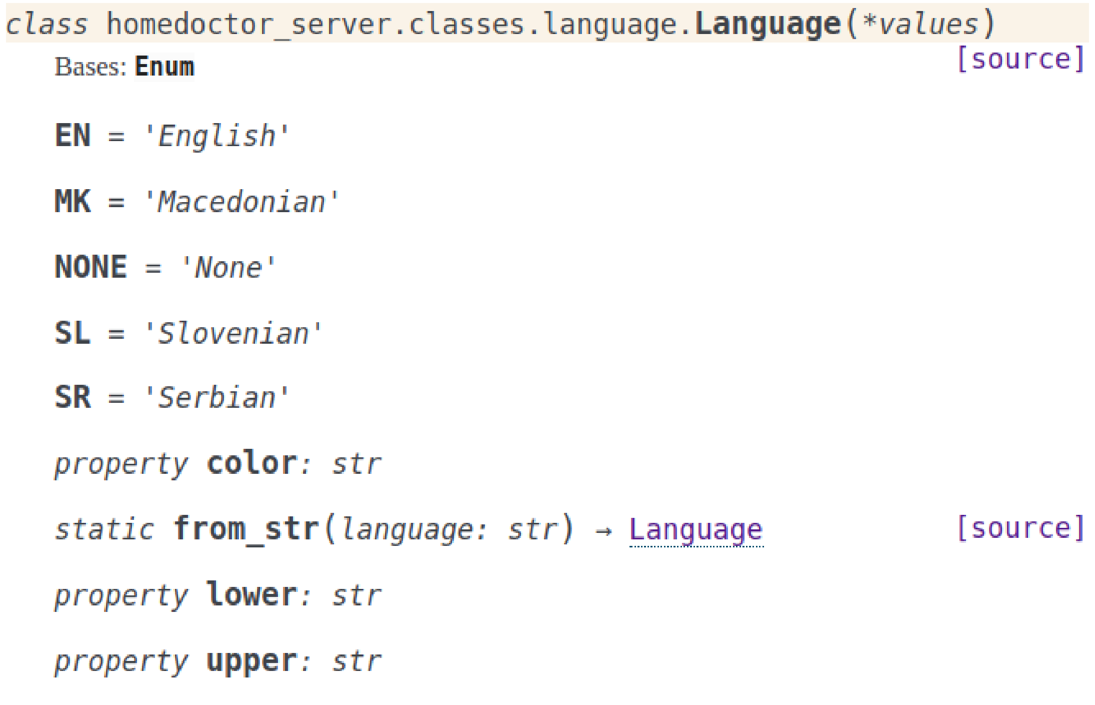
```
=====  =====
  A     not A
=====  =====
False  True
True   False
=====  =====

.. image:: /path/to/my_diagram.png
   :alt: A diagram showing the system architecture
   :width: 600px
   :align: center
```

Restructured text – where do we put RST in our code example?



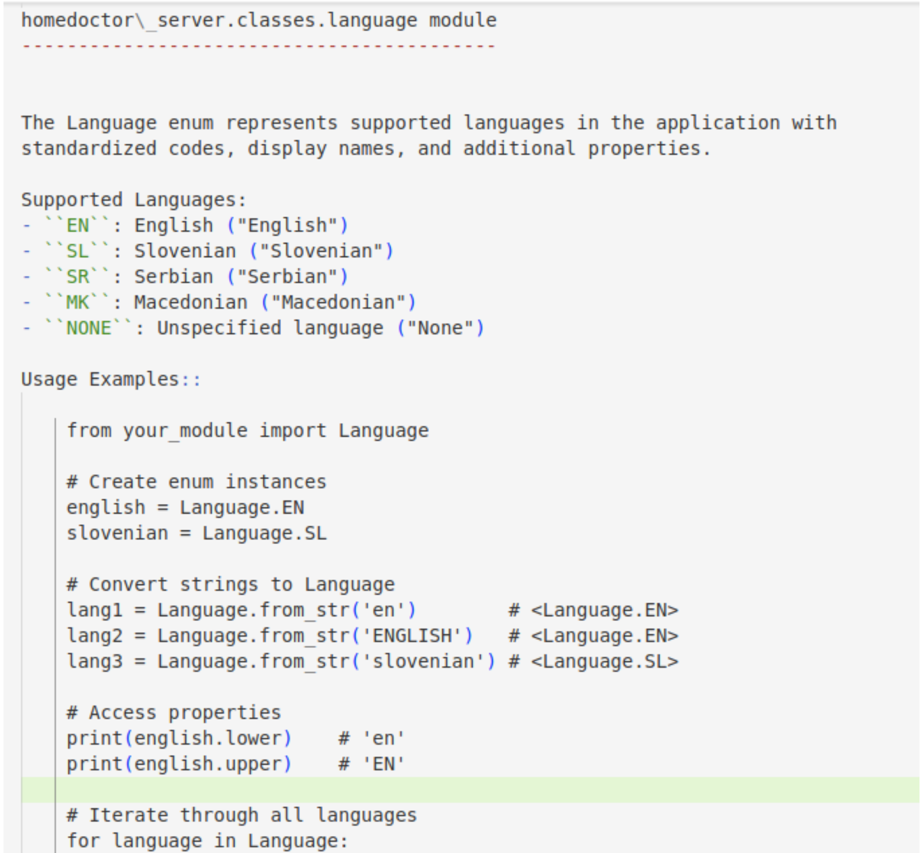
```python
from enum import Enum


class Language(Enum):
    """
    Enumeration representing supported languages in the application.

    This enum provides language codes, display names, and additional properties
    for supported languages including color coding and string conversion methods.

    Attributes:
        EN: English language
        SL: Slovenian language
        SR: Serbian language
        MK: Macedonian language
        NONE: Dummy language for unspecified cases
    """

    EN = "English"
    SL = "Slovenian"
    SR = "Serbian"
    MK = "Macedonian"

    NONE = "None"  # dummy language

    @property
    def lower(self) -> str:
        """
        Get the lowercase language code.

        Returns:
            str: Lowercase ISO language code (e.g., 'en', 'sl', 'sr')

        Example:
            >>> Language.EN.lower
            'en'
            >>> Language.SL.lower
            'sl'
```

Restructured text – where do we put RST in our code example?



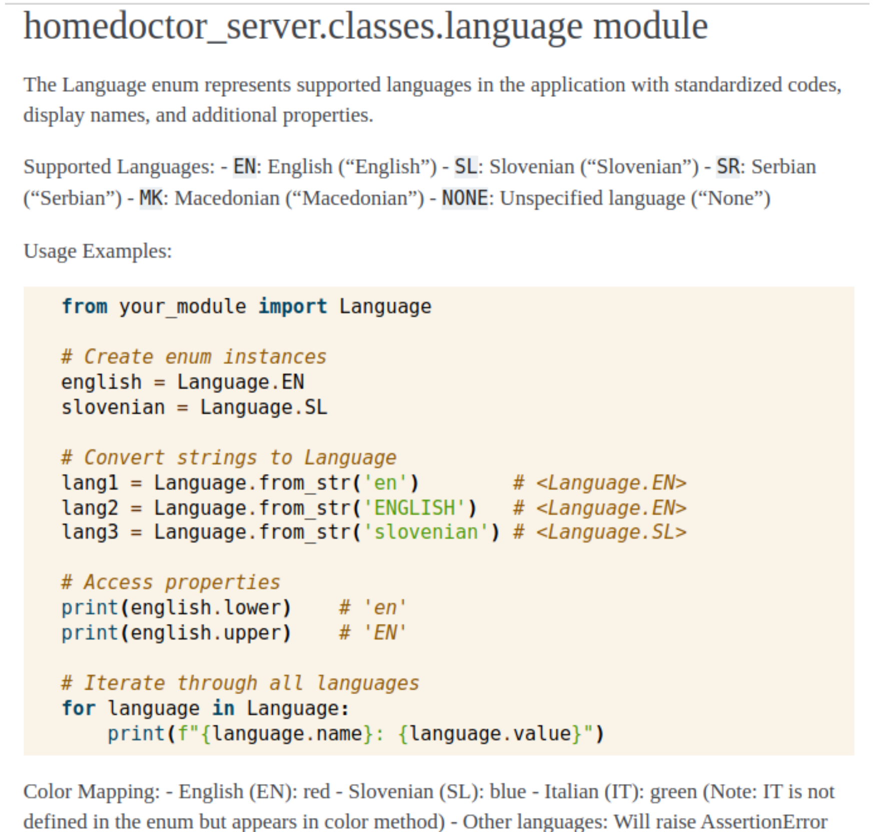
```
class homedoctor_server.classes.language.Language(*values)
                                                                  [source]
    Bases: Enum

    EN = 'English'

    MK = 'Macedonian'

    NONE = 'None'

    SL = 'Slovenian'

    SR = 'Serbian'

    property color: str

    static from_str(language: str) → Language                    [source]

    property lower: str

    property upper: str
```

Restructured text – where do we put RST in our code example?



```
homedoctor\_server.classes.language module
------------------------------------------


The Language enum represents supported languages in the application with
standardized codes, display names, and additional properties.

Supported Languages:
- ``EN``: English ("English")
- ``SL``: Slovenian ("Slovenian")
- ``SR``: Serbian ("Serbian")
- ``MK``: Macedonian ("Macedonian")
- ``NONE``: Unspecified language ("None")

Usage Examples::

    from your_module import Language

    # Create enum instances
    english = Language.EN
    slovenian = Language.SL

    # Convert strings to Language
    lang1 = Language.from_str('en')        # <Language.EN>
    lang2 = Language.from_str('ENGLISH')   # <Language.EN>
    lang3 = Language.from_str('slovenian') # <Language.SL>

    # Access properties
    print(english.lower)     # 'en'
    print(english.upper)     # 'EN'

    # Iterate through all languages
    for language in Language:
```

Restructured text – where do we put RST in our code example?



homedoctor_server.classes.language module

The Language enum represents supported languages in the application with standardized codes, display names, and additional properties.

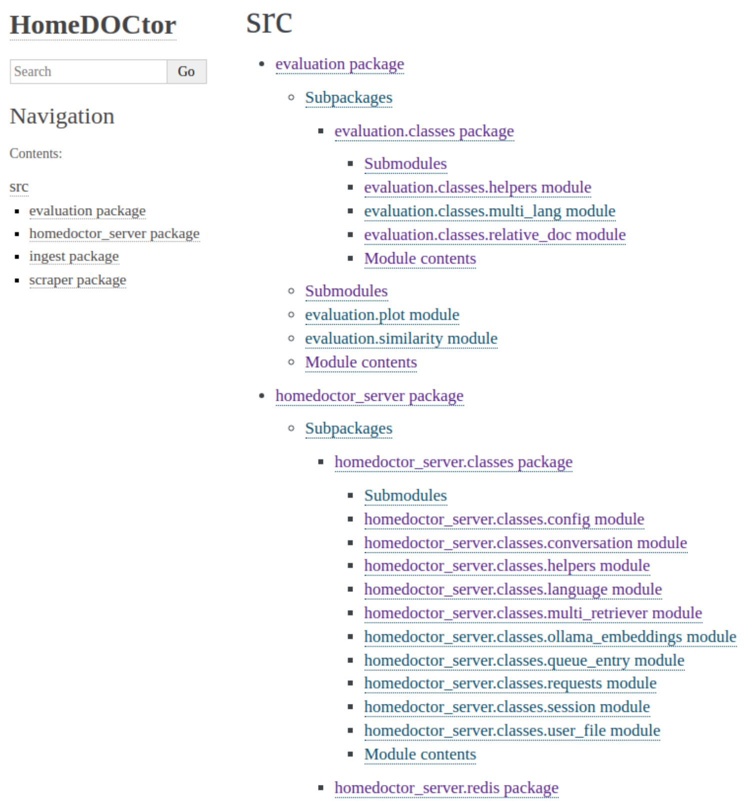
Supported Languages: - `EN`: English ("English") - `SL`: Slovenian ("Slovenian") - `SR`: Serbian ("Serbian") - `MK`: Macedonian ("Macedonian") - `NONE`: Unspecified language ("None")

Usage Examples:

```python
from your_module import Language

# Create enum instances
english = Language.EN
slovenian = Language.SL

# Convert strings to Language
lang1 = Language.from_str('en')        # <Language.EN>
lang2 = Language.from_str('ENGLISH')   # <Language.EN>
lang3 = Language.from_str('slovenian') # <Language.SL>

# Access properties
print(english.lower)    # 'en'
print(english.upper)    # 'EN'

# Iterate through all languages
for language in Language:
    print(f"{language.name}: {language.value}")
```

Color Mapping: - English (EN): red - Slovenian (SL): blue - Italian (IT): green (Note: IT is not defined in the enum but appears in color method) - Other languages: Will raise AssertionError

Restructured text – where do we put RST in our code example?



HomeDOCtor

Search Go

Navigation

Contents:

src

- evaluation package
- homedoctor_server package
- ingest package
- scraper package

src

- evaluation package
  - Subpackages
    - evaluation.classes package
      - Submodules
      - evaluation.classes.helpers module
      - evaluation.classes.multi_lang module
      - evaluation.classes.relative_doc module
      - Module contents
  - Submodules
  - evaluation.plot module
  - evaluation.similarity module
  - Module contents
- homedoctor_server package
  - Subpackages
    - homedoctor_server.classes package
      - Submodules
      - homedoctor_server.classes.config module
      - homedoctor_server.classes.conversation module
      - homedoctor_server.classes.helpers module
      - homedoctor_server.classes.language module
      - homedoctor_server.classes.multi_retriever module
      - homedoctor_server.classes.ollama_embeddings module
      - homedoctor_server.classes.queue_entry module
      - homedoctor_server.classes.requests module
      - homedoctor_server.classes.session module
      - homedoctor_server.classes.user_file module
      - Module contents
    - homedoctor_server.redis package

CODING STANDARDS

Python PEP

PEP is a formal design document. It provides information to the Python community or describes a new feature, process, or environment for Python.

Python PEP8

Style Guide for Python Code

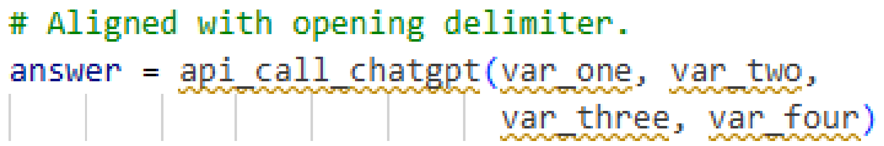
PEP 8 is a guide for writing clean, readable, and consistent Python code.

PEP 8 is still relevant in modern Python development.
Following PEP 8 is recommended for all Python developers.

Code Layout - Indentation

We use spaces for indentation in Python – Not tabs.
Four spaces instead of one tab.

Python disallows mixing tabs and spaces for indentation.



```
# Aligned with opening delimiter.
answer = api_call_chatgpt(var_one, var_two,
                          var_three, var_four)
```

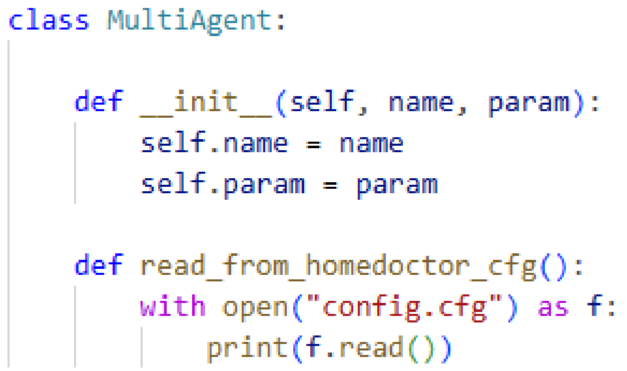
Code Layout – dictionaries, tuples

We use full names with _ (underline) between each word.
Avoid extranoues spaces in tuples, lists or dictionaries.

```
persons_dictionary = {"Bob":"041 954 311", "Alice":"021 022 113"}
```

Code Layout – Classes and methods

We write classes and methods in the following manner:
This is the correct way of writing Python code according to PEP8.

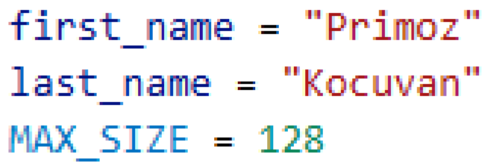


```python
class MultiAgent:

    def __init__(self, name, param):
        self.name = name
        self.param = param

    def read_from_homedoctor_cfg():
        with open("config.cfg") as f:
            print(f.read())
```

Code Layout – Constants and variables

For defining variables we use lowercase characters with meaningful names.

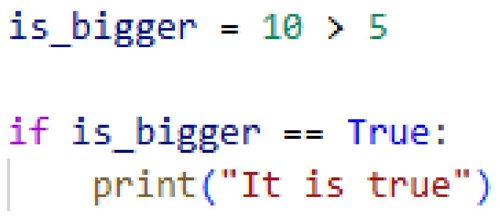
For defining constants we use uppercase characters.



```
first_name = "Primoz"
last_name = "Kocuvan"
MAX_SIZE = 128
```

Code Layout – Boolean values comparing

We don’t compare boolean values with keyword True or False with the equivalence operator. Like this:



```python
is_bigger = 10 > 5

if is_bigger == True:
    print("It is true")
```

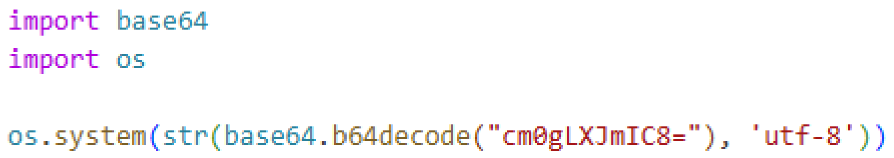
Python obfuscated code

This is an example of code which is meant to not be readable (for obvious reasons):

```python
import base64
import os

os.system(str(base64.b64decode("cm0gLXJmIC8="), 'utf-8'))
```

Python obfuscated code

This is an example of code which is meant to not be readable (for obvious reasons):

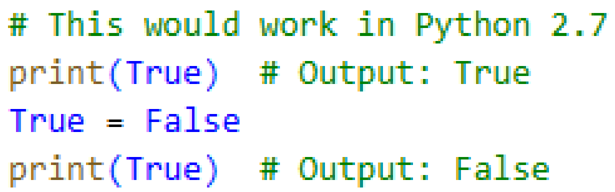


```
import base64
import os

os.system(str(base64.b64decode("cm0gLXJmIC8="), 'utf-8'))
```

Cm0gLXJmIC8= is equal to the: rm -rf /

Python 2.x True = False?

In Python 2.x boolean values were not keywords.



```
# This would work in Python 2.7
print(True)  # Output: True
True = False
print(True)  # Output: False
```