

Applying Best Practices for Sustainable Code

Cambridge Digital Humanities

Digital Humanities & Research Software Engineering Summer School

July 3, 2024



Schedule



- | | |
|----------------------|---------------------------------------|
| 13:30 – 14:00 | Introduction to the workshop exercise |
| 14:00 – 15:00 | Group work – Part 1 |
| 15:00 – 15:30 | Break |
| 15:30 – 16:30 | Group work – Part 2 |
| 16:30 – 17:00 | Wrap-up discussion and feedback |

Applying Best Practices

Goal: Apply best practices in sustainable code to a provided example project; Ensure code is easy to *use, understand, extend,* and *attribute*

Best Practices – Broad Categories:

- Documentation
- Formatting & Refactoring
- Publishing

Documentation

- *Good* documentation:
 - provides insights into the algorithm being used,
 - explains why things that may seem surprising are correct, and
 - allows the reader to understand how and why the code works the way it does
- Audience:
 - Future users and developers
 - Yourself and others
- Documentation appears as:
 - Comments in the code
 - Header comments in functions and classes - **Docstrings** in Python
 - A README file
 - A requirements.txt file

Documentation: Comments

```
# File path to the data
file_path = "data.csv"
```

```
# Load the data
data = load_data(file_path)
```

```
# Clean the data
data = clean_data(data)
```

```
# Analyze the data
results = analyze_data(data)
```

```
# TODO: Add more detailed analysis (e.g., correlation analysis)
# TODO: Implement data visualization (e.g., plots, charts)
```

```
simple_preprocess(str(sentence), deacc=True) # deacc=True removes punctuations
```

```
bigram = Phrases(tokens, min_count=minCount, threshold=thresh) # higher threshold leads to fewer phrases
```

- *Structural comments*
- *Explanatory comments*
- *Comments indicating future tasks: TODO*

Documentation: Docstrings

```
# Function to deposit money
def deposit(account_name, amount):
    """
    Deposit money into a user's account.

    Parameters:
    - account_name (str): Name of the account holder.
    - amount (int): Amount to be deposited.

    Returns:
    - str: Deposit status message.
    """
    if amount > 0:
        user_accounts[account_name]['balance'] += amount
        user_accounts[account_name]['transactions'].append(f"Deposit: +{amount}")
        return f"Deposit successful. New balance: {user_accounts[account_name]['balance']}"
    else:
        return "Invalid deposit amount"
```

The **docstring** describes the purpose, parameters, and return values of the function.

Documentation: README

Natural Language Toolkit (NLTK)

NLTK -- the Natural Language Toolkit -- is a suite of open source Python modules, data sets, and tutorials supporting research and development in Natural Language Processing. NLTK requires Python version 3.8, 3.9, 3.10, 3.11 or 3.12.

For documentation, please visit nltk.org.

Contributing

Do you want to contribute to NLTK development? Great! Please read [CONTRIBUTING.md](#) for more details.

See also [how to contribute to NLTK](#).

Donate

Have you found the toolkit helpful? Please support NLTK development by donating to the project via PayPal, using the link on the NLTK homepage.

Citing

If you publish work that uses NLTK, please cite the NLTK book, as follows:

Bird, Steven, Edward Loper and Ewan Klein (2009).
Natural Language Processing with Python. O'Reilly Media Inc.

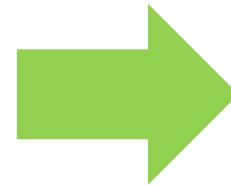


Documentation: Requirements

Generate a requirements.txt file by looking at the *import* statements in the code

Import statements within the .ipynb or .py file:

```
import ipywidgets as ipyw
import matplotlib.pyplot as plt
import pandas as pd
import torch
import torchvision
import torchvision.transforms as transforms
import tqdm
```



requirements.txt file:

```
≡ requirements.txt
1  ipywidgets==8.1.2
2  matplotlib
3  pandas
4  torch==2.3.0
5  torchvision==0.18.0
6  tqdm==4.66.4
7
```


Formatting

Whitespace

- Whitespace or blank spaces can effectively clarify the function and structure of code
- Consistency is key

Naming

- Names should succinctly indicate what a function does or a variable means
- Rule of thumb: length of the variable name \propto scope and complexity
- Consistency of convention: snake_case or camelCase

Formatting

Poorly formatted:

```
def p(l):  
    r =[]  
    for i in l:  
        if i % 2==0:r.append(i)  
    return r
```

```
def s(l):  
    t = 1  
    for i in l:  
        t *=i  
    return t
```

```
a = [1, 2, 3, 4, 5]  
b = p(a)  
c = s(b)  
print(b, c)
```

Well formatted:

```
def get_even_numbers(numbers):  
    even_numbers = []  
    for number in numbers:  
        if number % 2 == 0:  
            even_numbers.append(number)  
    return even_numbers
```

```
def multiply_elements(elements):  
    product = 1  
    for element in elements:  
        product *= element  
    return product
```

```
numbers = [1, 2, 3, 4, 5]
```

```
even_numbers = get_even_numbers(numbers)  
product_of_even_numbers = multiply_elements(even_numbers)  
print(even_numbers, product_of_even_numbers)
```

Formatting

Poorly formatted:

```
def calc_sum(numsList):
    result = 0
    for n in numsList:
        result += n
    return result

def CalcProduct(nums_list):
    product = 1
    for num in nums_list:
        product *= num
    return product

numbers = [1, 2, 3, 4, 5]
sumResult = calc_sum(numbers)
print(sumResult)
product_result = CalcProduct(numbers)
print(product_result)
```

Well formatted:

```
def calculate_sum(numbers_list):
    sum_result = 0
    for number in numbers_list:
        sum_result += number
    return sum_result

def calculate_product(numbers_list):
    product_result = 1
    for number in numbers_list:
        product_result *= number
    return product_result

numbers = [1, 2, 3, 4, 5]

sum_result = calculate_sum(numbers)
print(sum_result)

product_result = calculate_product(numbers)
print(product_result)
```

Refactoring – Functions

Functions: a predefined set of variables and expressions

- Typically describe *an action*
- Functions have **scope** – a variable is only available inside the region it is created
- Allow for flexible input and output – **code reuse**
- Outline the main tasks
- Guidelines:
 - Smaller functions are easier to read
 - Each function should accomplish **one logical task**

```
def calculate_circle_area(radius):  
    '''Calculate the area of a circle  
    Parameters: radius (double) - the radius of the circle  
    Returns: double - the area of the circle  
    '''  
  
    pi = 3.14  
    return pi * radius ** 2
```

```
data = load_data(data_location)  
pp_data = preprocess_data(data)  
top_n = get_top_n_words(pp_data)  
visualise_top_n_words(top_n)
```

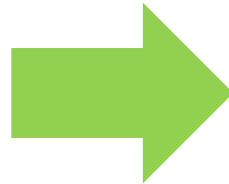
Refactoring

- General Guideline: Don't Repeat Yourself (DRY)
 - Can you minimise the amount of code by using existing lines more effectively?
 - Are repeated lines more suited to a loop or a function?

```
names1 = ["Alice", "Bob", "Charlie"]
names2 = ["David", "Eve", "Frank"]
names3 = ["Grace", "Heidi", "Ivan"]

uppercase_names1 = []
for name in names1:
    uppercase_names1.append(name.upper())
print(f"Uppercase names1: {uppercase_names1}")

uppercase_names2 = []
for name in names2:
    uppercase_names2.append(name.upper())
print(f"Uppercase names2: {uppercase_names2}")
...
```



Refactored:

```
def convert_to_uppercase(names):
    return [name.upper() for name in names]

names1 = ["Alice", "Bob", "Charlie"]
names2 = ["David", "Eve", "Frank"]
names3 = ["Grace", "Heidi", "Ivan"]

all_names = names1 + names2 + names3
uppercase_names = convert_to_uppercase(all_names)

print(f"Uppercase names: {uppercase_names}")
```

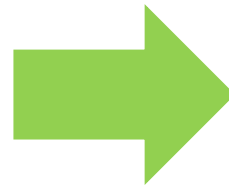
Publication





- Publication best practices ensure code is citable and easily accessible and outlines proper use
- Code publication practices include:
 - Minting a digital object identifier (DOI)
 - Adding a CITATION.cff file
 - Adding a LICENSE file and copyright statement
 - Preparing a GitHub release
 - Package the project

Publication - Citation

Example CITATION.cff file:

```
1  cff-version: 1.2.0
2  message: "If you use this software, please cite it as below."
3  authors:
4    - family-names: "Lovelace"
5      given-names: "Ada"
6      orcid: "https://orcid.org/0000-0000-0000-0000"
7    - family-names: "Babbage"
8      given-names: "Charles"
9      orcid: "https://orcid.org/0000-0000-0000-0000"
10 title: "My Research Software"
11 version: 2.0.4
12 doi: 10.5281/zenodo.1234
13 date-released: 2017-12-18
14 url: "https://github.com/github-linguist/linguist"
```



-  Readme
-  Apache-2.0 license
-  Security policy
-  Cite this repository ▼

Cite this repository

If you use this software in your work, please cite it using the following metadata. [Learn more about CITATION files.](#)

APA

BibTeX

Bird, S., Klein, E., & Loper, E. (2009). Natural L



View citation file

Publication – Packaging and Releases

- Packaging a python project enables others to easily access it using the Python Package Index (PyPI)
- Packaged projects can be installed using the command: `pip install mypackage` where `mypackage` is the name of the python project
- Releases indicate a particular version of the project, where features are bundled together

Natural Language Toolkit (NLTK)

pypi v3.8.1 ci-workflow passing

Workshop Exercise

- Data-centric task: Text analysis applied to Shakespeare's *Hamlet*

- Basic Functionality:

- **Load** the data
- **Preprocess** the data
- **Analyse** the data
- **Visualise** the data

Load the data:

Open file and extract each line of text

Preprocessing:

Parsing – remove punctuation and parse into words

Remove Stop-words – words that are not important for the analysis

Lemmatisation – break down a word into it's root

Analysis:

Count the frequency of each word

Visualisation:

Plot a histogram of the top 10 most-frequency words

Workshop Exercise

- You've inherited code – your task is to prepare the code for publication!
 - `text_analysis_notebook.ipynb`
- We've provided you with a checklist of best-practices
 - You **do not** have to complete all tasks in the time available
 - Tasks labelled with “*Essential*” should be prioritized
- Divide the tasks amongst your group
 - Pair up on tasks that interest you, or work solo
 - Avoid conflicts: Be sure that only one person is working on each document at a time