NTNU

Norwegian University of Science and Technology

Documenting code for your research

Towards reproducibility I

Xavier Sánchez Díaz

February 17, 2022

Outline

The reproducibility crisis
Motivation

"Recent studies imply that research presented at top AI conferences is not documented well enough for the research to be reproduced…" (O. E. Gundersen)

Standing on the Feet of Giants – Reproducibility in AI, AIMag, vol. 40, no. 4, pp. 9-23, Dec. 2019.

The reproducibility crisis

Motivation

For a research piece to be trustworthy it needs to be:

- **Open**
  - Available to everyone
  - Accessible to everyone
- **Explainable**
  - Via extrinsic explanations (testing)
  - Via intrinsic explanations (documentation)
- **Reproducible**
  - Clear methodology
  - Dataset available

The reproducibility crisis

Motivation

For a research piece to be trustworthy it needs to be:

- **Open**
  - Available to everyone
  - Accessible to everyone
- **Explainable**
  - Via extrinsic explanations (testing)
  - Via intrinsic explanations (documentation)
- **Reproducible**
  - Clear methodology
  - Dataset available

What is documentation?

Documentation in Computer Science usually refers to the collection of technical and detailed information and specification about a piece of software. It usually includes (but is not limited to):

- Comments in the code
- Docstrings
- Diagrams
- Use cases
- Manuals
- Guides and Tutos

Levels of documentation

1. Following style guides/programming principles for your code

Levels of documentation

1. Following style guides/programming principles for your code
2. Comments in your code

Levels of documentation

1. Following style guides/programming principles for your code
2. Comments in your code
3. Type-hinting

Levels of documentation

1. Following style guides/programming principles for your code
2. Comments in your code
3. Type-hinting
4. Docstrings in your code

Levels of documentation

1. Following style guides/programming principles for your code
2. Comments in your code
3. Type-hinting
4. Docstrings in your code
5. Following style guides for your docstrings

Levels of documentation

1. Following style guides/programming principles for your code
2. Comments in your code
3. Type-hinting
4. Docstrings in your code
5. Following style guides for your docstrings
6. Generate a manual for your software

Levels of documentation

1. Following style guides/programming principles for your code
2. Comments in your code
3. Type-hinting
4. Docstrings in your code
5. Following style guides for your docstrings
6. Generate a manual for your software
7. Unit tests

Style guides

Python tools

Python purists have these *Python Enhancements Proposals* (**PEP**) that you are encouraged to follow. The most important are PEP 0, PEP 8 and PEP 257.

PEP0
: Index of PEPs

PEP8
: Naming conventions, max. line width, spaces between methods, spaces before comments, spaces between operators, order of arguments, etc.

PEP257
: Docstrings conventions and recommendations

Type-hinting

Python tools

Type-hinting is an incredibly useful addition to Python 3 in which you annotate your code with the datatypes that you expect for functions and variables. So this:

```python
def somefunc(x,y):
    # do some stuff
    return x + y
```

Becomes this:

```python
def somefunc(x: int, y: int) -> int:
    # do some stuff
    return x + y
```

Type-hinting

Python tools

You can also do the same with more complex datatypes, too:

```python
def somefunc(x: np.ndarray, y: np.ndarray) -> tuple:
    # do some stuff
    return 2 * x, 3 * y
```

Docstrings

Python tools

Docstrings (short for *documentation strings*) are extended summaries of what a piece of code is supposed to do. They can span multiple lines (see PEP257) and are located below the **signature** of function and methods:

```python
def feasible(ind: ind_type) -> bool:
    """Define feasibility region for individuals of
    ind_type. Returns a boolean.
    """
    w = weight(ind)
    feas = False

    if w <= C:
        feas = True
    return feas
```

Stylised docstrings

Python tools

Of course there are **style guides** for docstrings:

```python
def cxOnePoint(ind1: ndarray, ind2, R: Random = None) -> tuple:
    """Performs crossover between two numpy array individuals using
    one crossover point. The crossover is performed in place.

    Parameters
    ----------
    ind1 : ndarray
        First individual to participate in the crossover
    ind2 : ndarray
        Second individual to participate in the crossover
    R : Random, optional
        Random number generator to set deterministic seed, by default None

    Returns
    -------
    tuple of ndarray
        Tuple of modified individuals
    """
    if R is None:
        R = Random()
    ind_size = ind1.shape[0]
    p = R.randint(1, ind_size)
    ind1[p:], ind2[p:] = ind2[p:].copy(), ind1[p:].copy()
    return ind1, ind2
```

Is it really worth it?

Is it really worth it?

Yes.

Your editor recognises the hints

Is it really worth it?

Readability is not the only benefit you get from **properly documenting your code**:

1. **Linters** recognise type hinting and include it the signatures when looking for help or code definitions.
2. **Docstrings** are shown when using the `help()` function (e.g. in Jupyter)
3. There are **automatic tools** for documentation generation via docstring extraction
   - See *Sphinx*, *Javadoc* or *Doxygen*
   - You can extend these descriptions manually with examples or math
   - Examples can be used for unit testing!

What can I do then?

Recap

1. Following style guides/programming principles for your code (PEPs)

What can I do then?

Recap

1. Following style guides/programming principles for your code (PEPs)
2. Comments in your code (Mandatory)

What can I do then?

Recap

1. Following style guides/programming principles for your code (PEPs)
2. Comments in your code (Mandatory)
3. Type-hinting (Highly suggested)

What can I do then?

Recap

1. Following style guides/programming principles for your code (PEPs)
2. Comments in your code (Mandatory)
3. Type-hinting (Highly suggested)
4. Docstrings in your code (Highly suggested)

What can I do then?

Recap

1. Following style guides/programming principles for your code (PEPs)
2. Comments in your code (Mandatory)
3. Type-hinting (Highly suggested)
4. Docstrings in your code (Highly suggested)
5. Following style guides for your docstrings (Encouraged)

What can I do then?

Recap

1. Following style guides/programming principles for your code (PEPs)
2. Comments in your code (Mandatory)
3. Type-hinting (Highly suggested)
4. Docstrings in your code (Highly suggested)
5. Following style guides for your docstrings (Encouraged)
6. Generate a manual for your software (Encouraged)

What can I do then?

Recap

1. Following style guides/programming principles for your code (PEPs)
2. Comments in your code (Mandatory)
3. Type-hinting (Highly suggested)
4. Docstrings in your code (Highly suggested)
5. Following style guides for your docstrings (Encouraged)
6. Generate a manual for your software (Encouraged)
7. Unit tests (Encouraged but complicated)

What can I do then?

Recap

1. Following style guides/programming principles for your code (PEPs)
2. Comments in your code (Mandatory)
3. Type-hinting (Highly suggested)
4. Docstrings in your code (Highly suggested)
5. Following style guides for your docstrings (Encouraged)
6. Generate a manual for your software (Encouraged)
7. Unit tests (Encouraged but complicated)

What can I do then?

Recap

1. Following style guides/programming principles for your code (PEPs)
2. Comments in your code (Mandatory)
3. Type-hinting (Highly suggested)
4. Docstrings in your code (Highly suggested)
5. Following style guides for your docstrings (Encouraged)
6. Generate a manual for your software (Encouraged)
7. Unit tests (Encouraged but complicated)

Additional notes

Recap

- Julia uses docstrings in Markdown
- R has markdown docs
- Jupyter has markdown for you to make executable and explainable code
- Sphinx can be used for languages other than Python via extensions
- Javadoc (Java) and Doxygen (Java, C++, PHP...) are good alternatives to Sphinx
- Git is crucial. GitHub/GitLab wikis are another alternative for code documentation.

Thank you!

Slides available at

`https://saxarona.github.io/project/python-docs/`