

Coding best practices

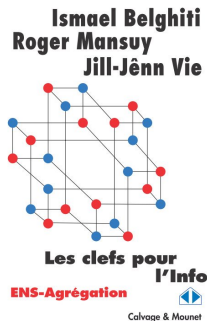
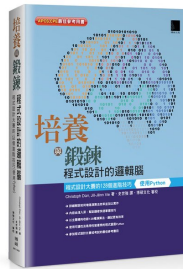
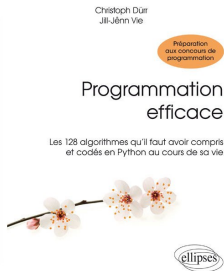
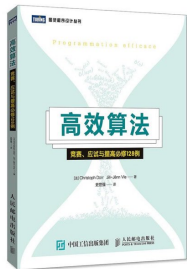
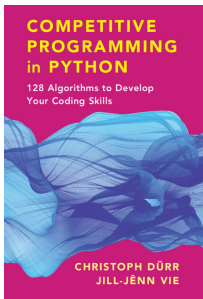
Jill-Jênn Vie



New in ML workshop

Hi, I'm JJ

- ▶ Researcher at Inria Saclay (south of Paris) in AI for education
- ▶ I wrote some books about algorithms and a Python package



```
pip install tryalgo
```

Writing code to oneself



Writing code for someone else



A cliché

Quote:

Unlike pro developers, researchers code for themselves only.

But at minimum you code for **your future self**.

(e.g., the version of you who is going to write your PhD thesis)

Versioning is a way to keep a backup of your work

(in case your house burns, for example)

A cliché

Quote:

Unlike pro developers, researchers code for themselves only.

But at minimum you code for **your future self**.

(e.g., the version of you who is going to write your PhD thesis)

Versioning is a way to keep a backup of your work

(in case your house burns, for example)

In case of fire



1. git commit



2. git push



3. leave building

Minimal requirements

- ▶ Committed code should run
- ▶ README with “how to use”
- ▶ Requirements should be listed **with versions**

(some times I had to guess the version using binary search)

If possible

- ▶ Examples
- ▶ Documentation

Coding worst practices

- ▶ Copy-pasting
- ▶ “I don’t have time for documentation right now”
- ▶ Trusting too much someone else’s package

Writing pretty code

`pycodestyle` follows PEP8 (Style Guide for Python code)

(also an `--aggressive` mode that will directly fix your code, use with caution)

`pylint` does static analysis: warns you about useless variables, etc.

Existing equivalents in other languages

Automatically writing beautiful docs:

```
def dijkstra(graph, weight, source=0, target=None):
    """single source shortest paths by Dijkstra
    :param graph: directed graph in listlist or listdict format
    :param weight: in matrix format or same listdict graph
    :assumes: weights are non-negative
    :param source: source vertex
    :type source: int
    :param target: if given, stops once distance to target found
    :type target: int
    :returns: distance table, precedence table
    :complexity:  $O(|V| + |E|\log|V|)$ 
    """
    n = len(graph)
    assert all(weight[u][v] >= 0 for u in range(n) for v in graph[u])
    prec = [None] * n
    dist = [float('inf')] * n
    dist[source] = 0
    heap = OurHeap([(dist[node], node) for node in range(n)])
    while heap:
        dist_node, node = heap.pop()          # Closest node from source
        if node == target:
            break
        for neighbor in graph[node]:
            old = dist[neighbor]
            new = dist_node + weight[node][neighbor]
            if new < old:
                dist[neighbor] = new
                prec[neighbor] = node
                heap.update((old, neighbor), (new, neighbor))
    return dist, prec
```

Sphinx's autodoc

tryalgo.dijkstra module

Shortest paths by Dijkstra jill-jënn vie et christoph dürr - 2015-2018

tryalgo.dijkstra.dijkstra(graph, weight, source=0, target=None) [\[source\]](#)

single source shortest paths by Dijkstra

Parameters

- **graph** – directed graph in listlist or listdict format
- **weight** – in matrix format or same listdict graph
- **source** (*int*) – source vertex
- **target** (*int*) – if given, stops once distance to target found

Assumes

weights are non-negative

Returns

distance table, precedence table

Complexity

$O(|V| + |E|\log|V|)$

Writing tests

No need to do it for everything, but some parts of your code.

```
jj@altaria:~/code/tryalgo$ python -m unittest
```

```
.....  
-----
```

```
Ran 102 tests in 1.185s
```

```
OK
```

What a test looks like

```
class OurQueue:
    """
    A queue for counting efficiently the number of events
    within time windows.
    Complexity:
        All operators in amortized  $O(W)$  time
        where  $W$  is the number of windows.
    From JJ's KTM repository: https://github.com/jilljenn/ktm.
    """
    def __init__(self):
        self.queue = []
        self.window_lengths = [
            3600 * 24 * 30, 3600 * 24 * 7, 3600 * 24, 3600]
        self.cursors = [0] * len(self.window_lengths)

    def __len__(self):
        return len(self.queue)

    def get_counters(self, t):
        self.update_cursors(t)
        return [len(self.queue)] + [len(self.queue) - cursor
                                     for cursor in self.cursors]

    def push(self, time):
        self.queue.append(time)

    def update_cursors(self, t):
        for pos, length in enumerate(self.window_lengths):
            while (self.cursors[pos] < len(self.queue) and
                   t - self.queue[self.cursors[pos]] >= length):
                self.cursors[pos] += 1
```

```
import unittest
from utils.this_queue import OurQueue

class TestOurQueue(unittest.TestCase):
    def test_simple(self):
        q = OurQueue()
        q.push(0)
        q.push(0.8 * 3600 * 24)
        q.push(5 * 3600 * 24)
        q.push(40 * 3600 * 24)
        self.assertEqual(
            q.get_counters(40 * 3600 * 24),
            [4, 1, 1, 1, 1])

    def test_complex(self):
        q = OurQueue()
        q.push(0)
        q.push(10)
        q.push(3599)
        q.push(3600)
        q.push(3601)
        q.push(3600 * 24)
        q.push(3600 * 24 + 1)
        q.push(3600 * 24 * 7)
        q.push(3600 * 24 * 7 + 1)
        q.push(3600 * 24 * 7 * 30)
        q.push(3600 * 24 * 7 * 30 + 1)
        self.assertEqual(
            q.get_counters(3600 * 24 * 7 * 30 + 1),
            [11, 2, 2, 2, 2])
```

Continuous integration



build passing

Make sure you don't break the existing version,
so that other people software, which relies on yours, won't break.

Ex. Travis, CircleCI, GitHub actions

argparse

Sometimes on servers you can only run jobs as bash scripts.
It's good to provide hyper-parameters in the command line.

```
if __name__ == '__main__':  
    parser = argparse.ArgumentParser(description='Generates tokens.')  
    parser.add_argument('filename', type=str, nargs='?', default='text',  
                        help='Try files in demo/ e.g. "demo/text.txt"')  
    parser.add_argument('--n', type=int, nargs='?', default=1,  
                        help='How many sequences should be printed')  
    parser.add_argument('--l', type=int, nargs='?', default=42,  
                        help='Length of these sequences')  
    args = parser.parse_args()
```

```
(venv) jj@altaria:~/code/markov.py$ python markov.py -h
```

```
usage: markov.py [-h] [--n [N]] [--l [L]] [filename]
```

```
Generates tokens.
```

```
positional arguments:
```

```
  filename      Try files in demo/ e.g. "demo/text.txt"
```

```
optional arguments:
```

```
  -h, --help    show this help message and exit
```

```
  --n [N]       How many sequences should be printed
```

```
  --l [L]       Length of these sequences
```

Embed hyper-parameters (argparse) in logs

```
saved_results = {
    'options': vars(args),
    'predictions': predictions,
    'results': results,
    'folds': folds
}
with open(os.path.join(folder, 'results-{}.json'.format(iso_date)), 'w') as f:
    json.dump(saved_results, f)
```

Creates assistments09/results-2022-07-06T23:04:22.097694.json which contains:

```
{
  "options": {"data": "assist09", "n_iter": 20, "d": 5},
  "predictions": [
    {"fold": 0, "pred": [0.6, 0.4], "truth": [1, 0]},
    {"fold": 1, "pred": [0.7, 0.2], "truth": [1, 1]},
  ],
  "results": [
    {"name": "auc", "value": 0.76},
    {"name": "accuracy", "value": 0.82}
  ],
  "folds": "<path-to-fold-file>"
}
```

A personal suggestion

Using Makefile

It defines a dependency graph, for example:

LaTeX figures $\rightarrow$ figures PDF $\rightarrow$ article / slides

A personal suggestion

Using Makefile

It defines a dependency graph, for example:

LaTeX figures $\rightarrow$ figures PDF $\rightarrow$ article / slides

Here is the Makefile of my CV:



Another example

data → preprocessed data → logs (incl. hyper-parameters) → plots

Logging metrics as much as possible

(it's OK to log several times at different locations: files, standard output)

Then your plots will only be from your logs; you can only recompute what has changed

Another example

data → preprocessed data → logs (incl. hyper-parameters) → plots

Logging metrics as much as possible

(it's OK to log several times at different locations: files, standard output)

Then your plots will only be from your logs; you can only recompute what has changed

Another example

data → preprocessed data → logs (incl. hyper-parameters) → **plots**

Logging metrics as much as possible

(it's OK to log several times at different locations: files, standard output)

Then your plots will only be from your logs; you can only recompute what has changed

Machine-learning specific

Start small

- ▶ Just a few lines of data are enough to test your pipeline
- ▶ Try to overfit a single batch

Feel free to ask questions

You can open issues; sometimes researchers are happy to know their code is useful

Do not forget to cite other people's software, e.g.



Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830

[SW] John MacFarlane, *pandoc* version 2.17, Jan. 2022. University of Berkeley. LIC: GPLv2.
URL: <https://pandoc.org/>, VCS: <https://github.com/jgm/pandoc>,

Feel free to ask questions

You can open issues; **sometimes** researchers are happy to know their code is useful

Do not forget to cite other people's software, e.g.



Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830

[SW] John MacFarlane, *pandoc* version 2.17, Jan. 2022. University of Berkeley. LIC: GPLv2.
URL: <https://pandoc.org/>, VCS: <https://github.com/jgm/pandoc>,

Feel free to ask questions

You can open issues; **sometimes** researchers are happy to know their code is useful



Do not forget to cite other people's software, e.g.

Fabian Pedregosa et al. "Scikit-learn: Machine learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830

[SW] John MacFarlane, *pandoc* version 2.17, Jan. 2022. University of Berkeley. LIC: GPLv2.
URL: <https://pandoc.org/>, VCS: <https://github.com/jgm/pandoc>,

↑ `\fullcite` made using `\usepackage{biblatex-software}`, thanks  Software Heritage

Stuff I didn't cover

screen for keeping an interactive session open even while you leave the server.

tmux is similar and better: great for pair-programming.

python -m pdb, the Python debugger.

magic-wormhole for sending (non-sensitive) data over the Internet

 one-click banner that opens a temporary Jupyter notebook or Lab in the browser with your repo (amazing, JupyterHub Core Team! 🙌)

ansible-playbook for automating a remote install

Take home message

- ▶ Versioning like git
- ▶ Help others help you

(README, requirements.txt, doc, examples)

- ▶ Start small
- ▶ There's nothing worse than making an irreproducible plot

Thanks! Questions?

jill-jenn.vie@inria.fr

- [1] [SW] John MacFarlane, *pandoc* version 2.17, Jan. 2022. University of Berkeley. LIC: GPLv2. URL: <https://pandoc.org/>, VCS: <https://github.com/jgm/pandoc>,
- [2] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.