Good Programming Practices, Computer Tools and QOL Tricks

...

Florentin Guth

CSD PhD Seminar - December 6, 2021

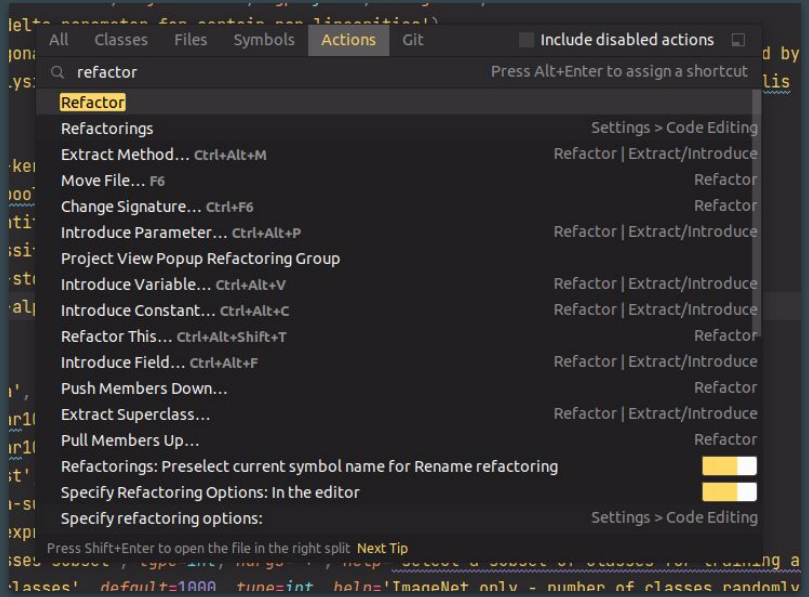
Outline

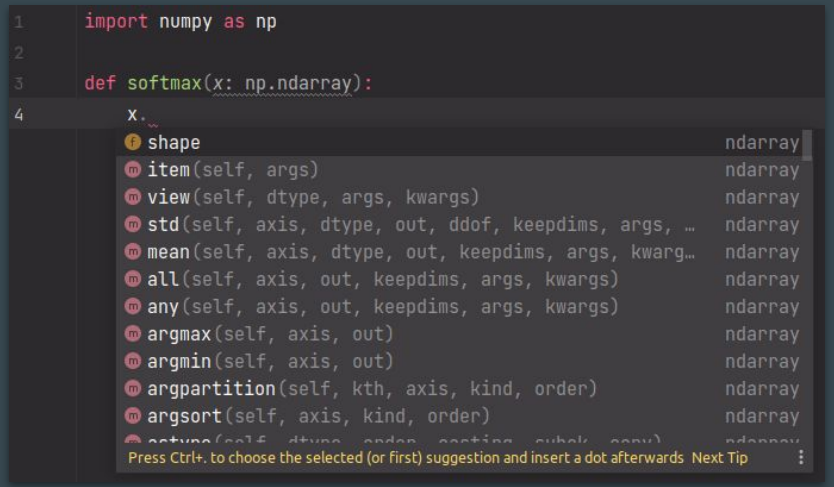
- Editing
- Programming
- Python Tricks
- Debugging
- Git and GitHub
- Visualization
- SSH
- Writing

Editing

Editor - PyCharm

- Powerful and easy to use
- Ctrl-Shift-A: universal key binding
- Refactoring tools
- Code-completion and inspection
  - Help it with type annotations





Get paid version for free with student email address!

Editor - Vim

- Command-line editor (useful for ssh or quick edits)
- Lots of commands to edit text quickly
- Steep learning curve but worth it
- Possible to use vim keybindings in PyCharm, your browser, Overleaf...


```
florentin@lappy:~/.../Thèse/Random lunches$ vim main.py
```

Usually pre-installed!

```python
def get_cost(configuration, participants, history):
    """ Measure the cost of a configuration according to past history of random lunches.
    :param configuration: list of sorted lists of integers, corresponding to participant indices.
    :param participants: DataFrame sorted by email addresses.
    :param history: dictionary of sorted email address pairs to number of past lunches.
    :return: the cost of the configuration (an integer), the smaller the better.
    """
    cost = 0
    for pair in enumerate_pairs(configuration):
        p0, p1 = participants.iloc[pair[0]], participants.iloc[pair[1]]
        emails = str(tuple(sorted((p0[email_field], p1[email_field]))))
        num_times = history.get(emails, 0)
        if p0[team_field] == p1[team_field]:
            num_times += 1000
        cost += num_times ** 2
    return cost


def update_history(configuration, participants, cost, payer_team, history):
    """ Updates in-place the history to account for the chosen configuration. """
    for pair in enumerate_pairs(configuration):
        # Record the pair in the history.
        emails = str(tuple(sorted((participants.loc[participants.index[pair[0]], email_field],
:'<,'>s/cost/loss/g
```

Programming

Commenting and Documenting

```python
def get_group_sizes(num_participants):
    """ Break the number of participants into group sizes.
    We try to make groups that are of size ideal_group_size or ideal_group_size +/- 1.
    :return: a dictionary whose keys are group sizes and whose values are the number of such groups.
    """
    # Start with k groups of size group_size and a smaller rest.
    num_full_groups = num_participants // ideal_group_size
    rest = num_participants % ideal_group_size

    # If there are not enough participants, then we can only make one group.
    if num_full_groups == 0:
        groups = {rest: 1}

    # If the rest is empty, we are done, there are only groups of the ideal size.
    elif rest == 0:
        groups = {ideal_group_size: num_full_groups}

    # Otherwise, we try to make groups of size ideal_group_size - 1 or ideal_group_size + 1, if possible.
    # This is done by moving participants from full groups to the rest groups, or by moving participants from the rest
    # group to full groups.
    else:
```

Start by explaining what your function, object, code does, and how it does it.

```python
def complex_conv2d(x, w):
    """ (B, C, M, N) complex and (K, C, H, W) complex to (B, K, M', N') complex """
    x = complex_to_real_channels(x, separate_real_imag=True)  # (B, 2C, M, N)
    w = torch.cat([torch.cat([w.real, -w.imag], dim=1), torch.cat([w.imag, w.real], dim=1)], dim=0)  # (2K, 2C, H, W)
    y = F.conv2d(x, w)  # (B, 2K, M', N')
    return real_to_complex_channels(y, separate_real_imag=True)  # (B, K, M', N') complex
```

Write explicitly all necessary information to understand or check your code (assumptions, shapes of tensors, ...).

Give explicit names to variables and functions, even if they are long.

```python
def get_random_minimal_cost_configuration(participants, group_sizes, history):
    """ Returns a random configuration among those of minimal cost. """
```

Factorization

- Copy-pasting code is usually harmful in the long run
- Factorize (compress) away redundant code
- Introduce pertinent variables, use (local) functions, loops...
- Dictionaries provide a nice switch statement

```python
def dataset(train: bool):  # Returns the train or validation dataset.
    dataset_class = dict(CIFAR10=datasets.CIFAR10, CIFAR100=datasets.CIFAR100,
                         MNIST=datasets.MNIST, ImageNet=datasets.ImageFolder)[dataset_name]
    kwargs = dict(
        root=os.path.join(args.data, "train" if train else "val") if dataset_name == "ImageNet" else "./data",
        transform=transforms.Compose((train_transforms if train else test_transforms) + common_transforms)
    )
    if dataset_name != "ImageNet":
        kwargs.update(train=train, download=True)
    return dataset_class(**kwargs)

print_and_write(f"Working on {dataset_name}", logfile, summaryfile)
train_dataset = dataset(train=True)
val_dataset = dataset(train=False)
```

```python
if args.cifar10:
    print_and_write("Working on CIFAR10", logfile, summaryfile)

    if args.grayscale:
        train_dataset = datasets.CIFAR10(
            root='./data', train=True, download=True, transform=transforms.Compose([
                transforms.RandomHorizontalFlip(),
                transforms.RandomCrop(32, padding=4),
                transforms.Grayscale(),
                transforms.ToTensor(),
                normalize,
            ]))

        val_dataset = datasets.CIFAR10(
            root='./data', train=False, download=True, transform=transforms.Compose(
                transforms.Grayscale(), transforms.ToTensor(), normalize]))

    else:
        train_dataset = datasets.CIFAR10(
            root='./data', train=True, download=True, transform=transforms.Compose([
                transforms.RandomHorizontalFlip(),
                transforms.RandomCrop(32, padding=4),
                transforms.ToTensor(),
                normalize,
            ]))

        val_dataset = datasets.CIFAR10(
            root='./data', train=False, download=True, transform=transforms.Compose(
                transforms.ToTensor(), normalize]))

elif args.cifar100:
    print_and_write("Working on CIFAR100", logfile, summaryfile)

    train_dataset = datasets.CIFAR100(root='./data', train=True,
            download=True, transform=transforms.Compose([
            transforms.RandomHorizontalFlip(),
            transforms.RandomCrop(32, padding=4),
            transforms.ToTensor(),
            normalize,
        ]))

    val_dataset = datasets.CIFAR100(root='./data', train=False, download=True,
                                    transform=transforms.Compose([transforms.ToTensor(), normalize, ]))
```

Architecture Design

```python
def build_layers(input_type: SplitTensorType, modules, i, args):
    """ Builds a list of layers from a list of modules.
    :param input_type:
    :param modules: list of strings describing the layers
    :param i: index of the current block
    :param args: command-line arguments
    :return: Sequential module
    """
    builder = Builder(input_type)

    def branching_kwargs(**submodules):...

    for module in modules:
        if module == "Fw":...

        elif module == "R":
            builder.add_batched(Realifier)

        elif module == "C":
            builder.add_batched(Complexifier)

        elif module == "B":...

        elif module in ["mod", "rho"]:...

        elif module == "Std":
            builder.add_batched(Standardization)

        elif module in ["P", "Pr", "Pc"]:...
```

- It can be useful to sit and think about how to implement a new feature (consider several options)
- Think from the point of view of the user: what would I like to be able to write?
- Plan for the future: more general and flexible is usually better
- Saves a lot of time if everything is customizable from command-line arguments
- Monolithic functions or objects are harder to modify

```
python main.py --arch "Fw rho Std P"
python main.py --arch "Fw rho P"
python main.py --arch "Fw mod Std R C"
python main.py --arch "Fw rho P mod"
```

Python Tricks

String Manipulation

```
In [3]: x = np.random.random(10)

In [4]: f"{x.shape} {x.min()=} {x.max()=:.2f}"
Out[4]: '(10,) x.min()=0.06936024503854088 x.max()=0.87'
```

f-strings are the ultimate logging tool.

eval can be used to allow powerful command-line customization.

```
if args.data_subset is not None:
    train_dataset = torch.utils.data.Subset(train_dataset, eval(args.data_subset))
```

```
class Branching(nn.Module):
    """ Implements a branching path. Expects a dict of split tensors, applies a submodule to each one
    and merges the result. """
    def __init__(self, input_type: Dict[str, SplitTensorType], **kwargs):
        """
        :param input_type: dictionary of SplitTensorTypes, for each branch of the path
        :param kwargs: of the form `key`_module_class and `key`_module_kwargs
        """
        super().__init__()

        self.input_type = input_type

        submodules = {}
        for key, input_type in self.input_type.items():
            module_class = kwargs.pop(f"{key}_module_class", Identity)
            module_kwargs = kwargs.pop(f"{key}_module_kwargs", {})
            submodules[key] = module_class(input_type, **module_kwargs)
        self.submodules = ModuleDict(submodules)
        assert len(kwargs) == 0
```

Dictionaries provide a way to manipulate variables by name (keyword arguments, locals). Use sparingly!

```
def compute_many_things():
    some_var = ...
    some_other_var = ...
    another_one = ...
    do_you_get_the_point = ...
    return locals()
```

```
In [5]: for i in range(10):
   ...:     locals()[f"var{i}"] = i
   ...:

In [6]: var5
Out[6]: 5
```

Capture Sub-Expressions with the Walrus Operator

The walrus operator (:=) allows simplifying the code in many cases.

```
scores = [score for item in iterable if (score := get_score(item)) is not None]
```

```
def argmax(iterable, score):
    """ Finds the first item in iterable with the highest score. """
    best_item = None
    best_score = float("-inf")
    for item in iterable:
        if (item_score := score(item)) > best_score:
            best_item = item
            best_score = item_score
    return best_item
```

```
if any(is_problematic(witness := item) for item in iterable):
    raise ValueError(f"Problem with {witness}")
```

Dealing with Long Computations

```python
def get_random_minimal_cost_configuration(participants, group_sizes, history):
    """ Returns a random configuration among those of minimal cost. """
    num_participants = len(participants)

    # Compute all configuration of with minimal cost.
    best_cost = float("inf")
    best_configurations = []
    try:
        iterator = tqdm.tqdm(enumerate_random_configurations(num_participants, group_sizes),
                             total=num_configurations(num_participants, group_sizes))
        for configuration in iterator:
            # Translate participants from numbers to email addresses.
            cost = get_cost(configuration, participants, history)

            if cost < best_cost:
                best_cost = cost
                best_configurations = [configuration]
            elif cost == best_cost:
                best_configurations.append(configuration)
            iterator.desc = f"Found {len(best_configurations)} configurations of cost {best_cost}"
    except KeyboardInterrupt:
        # Impatient user, just return one of the best configurations found so far
        pass

    # Return a random configuration among all best ones.
    return best_configurations[random.randrange(0, len(best_configurations))]
```

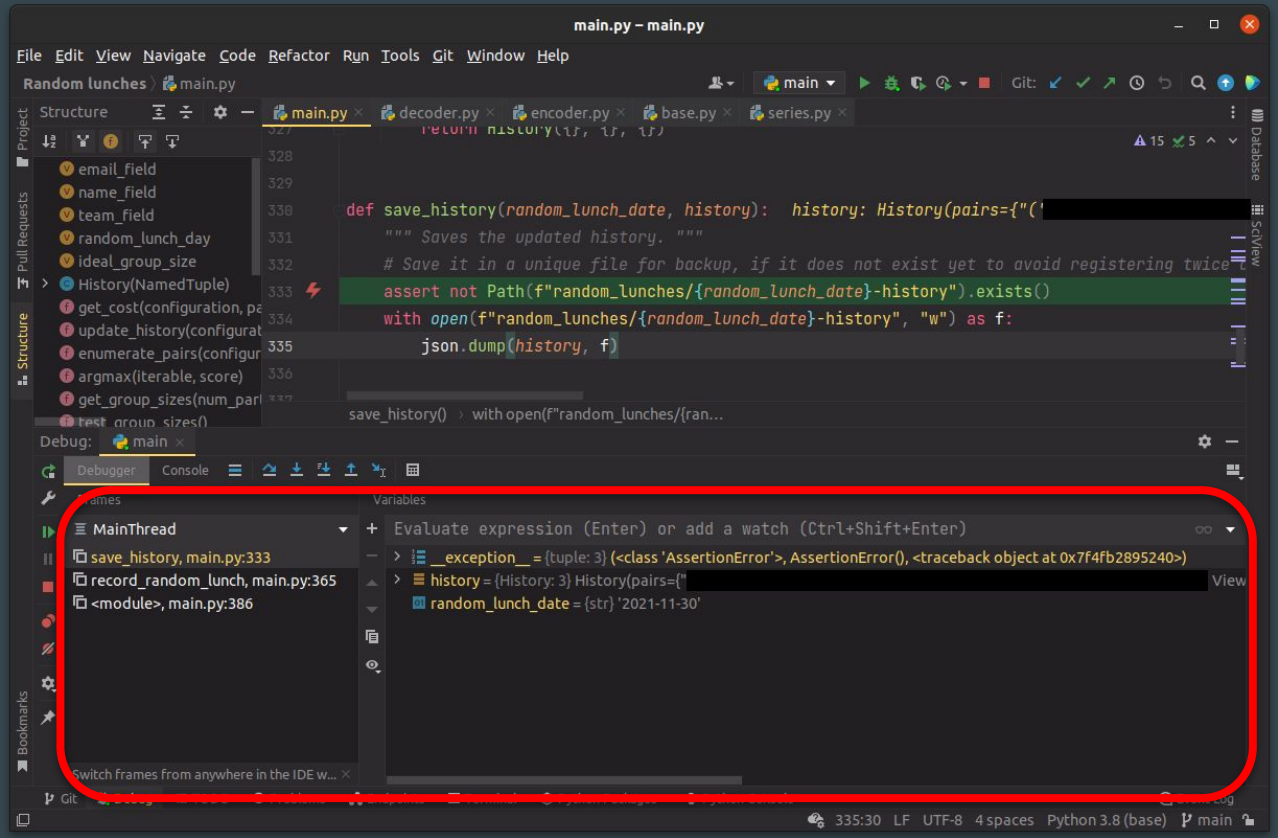
- Use tqdm to easily have a running time estimate
- Catch KeyboardInterrupt to perform final operations even when interrupted
- Cache results to disk using json or pickle to avoid redoing slow computations

```python
filename = self.full_path(name, suffix=".pt", level=level)
if Path(filename).exists():
    return torch.load(filename)
ret = closure(*args, **kwargs)
torch.save(ret, filename)
```

```
(base) florentin@lappy:~/.../Thèse/Random lunches$ python main.py
Found 1 configurations of cost 4000003:   0%|          | 5647/4509264634875.0 [00:19<4340128:20:29, 288.60it/s]
```

Debugging

Visual Debugging with PyCharm



- Interrupt on error (or manually)
- Trace execution
- Evaluate expressions
- Run step by step

Preventive Debugging

Do sanity checks with assertions or explicit raises (much better than silent errors).

```
if args.remove_L[i]:
    pass
elif i < len(args.separation_sizes):
    assert False  # TODO: this is deprecated
    total_phases, linear_phases = scattering_non_lin.phases()
    proj = FwStructuredProj(
        prev_out_channels=groups, L=args.scat_angles[i], A=4
```

```
def test_enumerate_groups():
    """ Small function to test enumeration of all possible groups. """
    from scipy.special import binom
    num_participants = 4
    for group_size in range(num_participants + 2):
        groups = list(enumerate_groups(num_participants, group_size))
        assert all(len(group) == group_size for group in groups)
        assert len(groups) == binom(num_participants, group_size)
        print(num_participants, group_size, groups)
```

Test early and independently if feasible.

Log relevant variables for post-execution checks.

```
logger.info(f"Command line: {' '.join(sys.argv)}")
```

```
def tensor_summary_stats(x):
    """ Returns summary statistics about x: shape, mean, std, min, max. """
    return f"shape {x.shape}, values in [{x.min():.3f}, {x.max():.3f}] and around {x.mean():.3f} +- {x.std():.3f}"
```

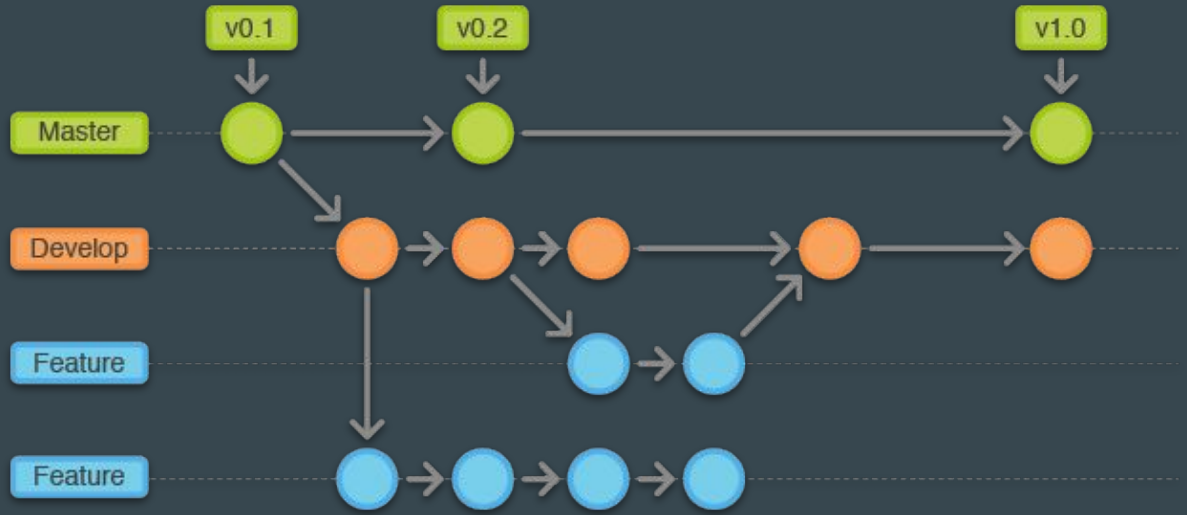
Rubber Duck Debugging

If all hope is lost: explain your problem in detail to your rubber duck (or your exasperated officemate). Explain how your program actually behaves and why it behaves this way. This usually helps :)

Git and GitHub

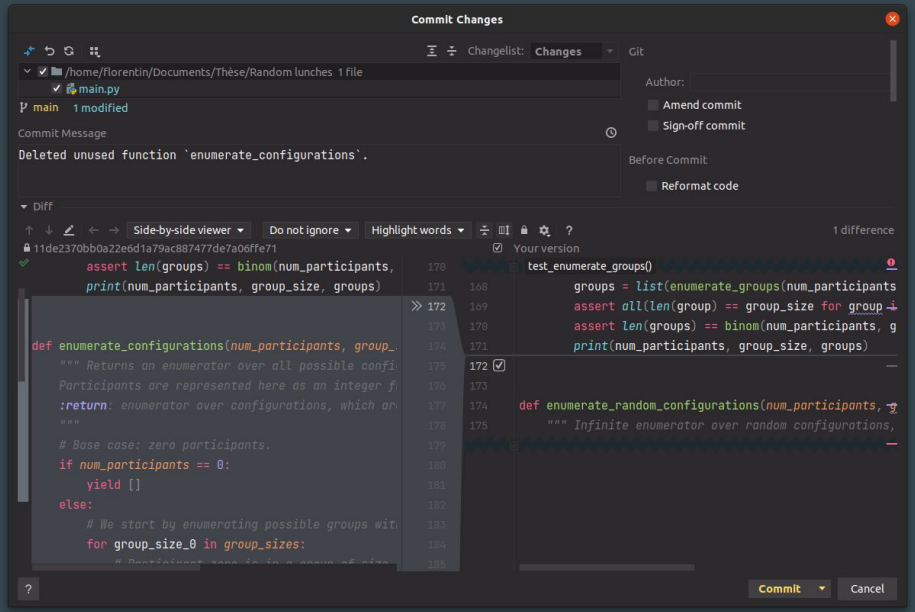
Git

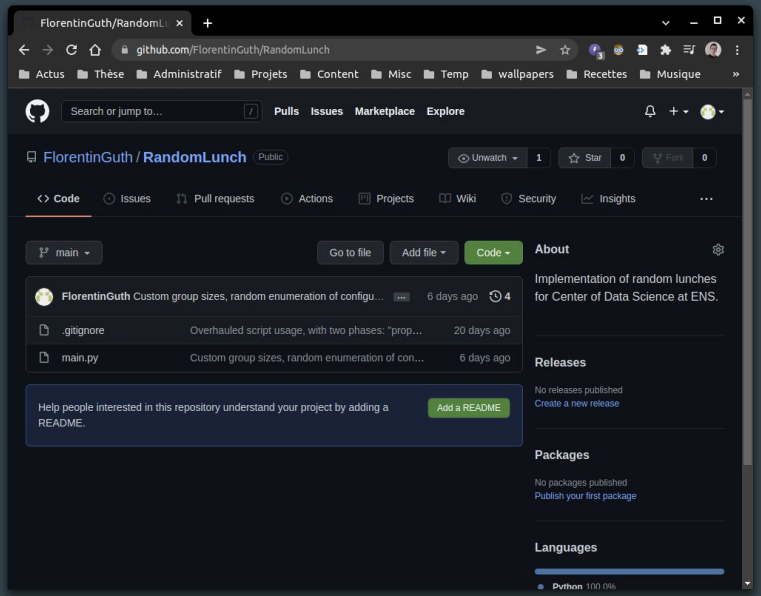
- Track your code changes
- Rollback to a previous version of your code
- Have multiple current versions with branches
- Know when a bug was introduced
- Easier to use with PyCharm integration



GitHub

- Collaborate with other people on code
- Sync your code across computers
- Easy online backup
- Free pro account with student email address
- Easier to use with PyCharm integration





Visualization

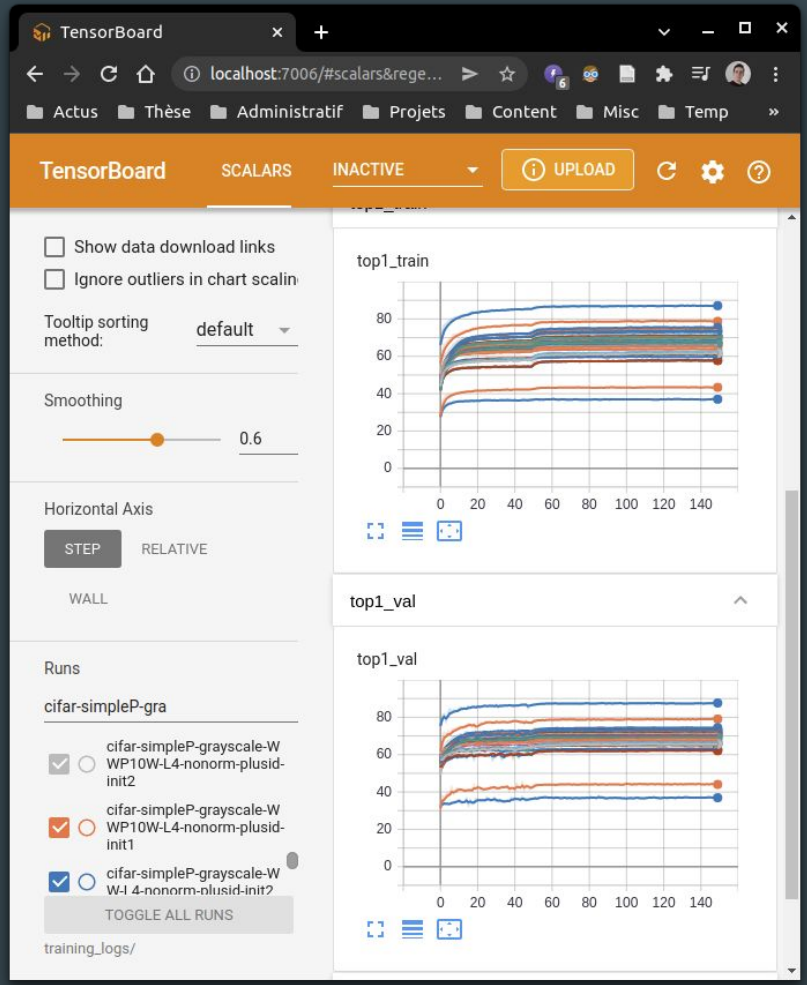
Tensorboard

- Monitor running experiments
- Compare them to previous ones
- Visualize the evolution of scalars, images...
- Easy-to-use web interface

```
from torch.utils.tensorboard import SummaryWriter
writer = SummaryWriter(logs_dir)
```

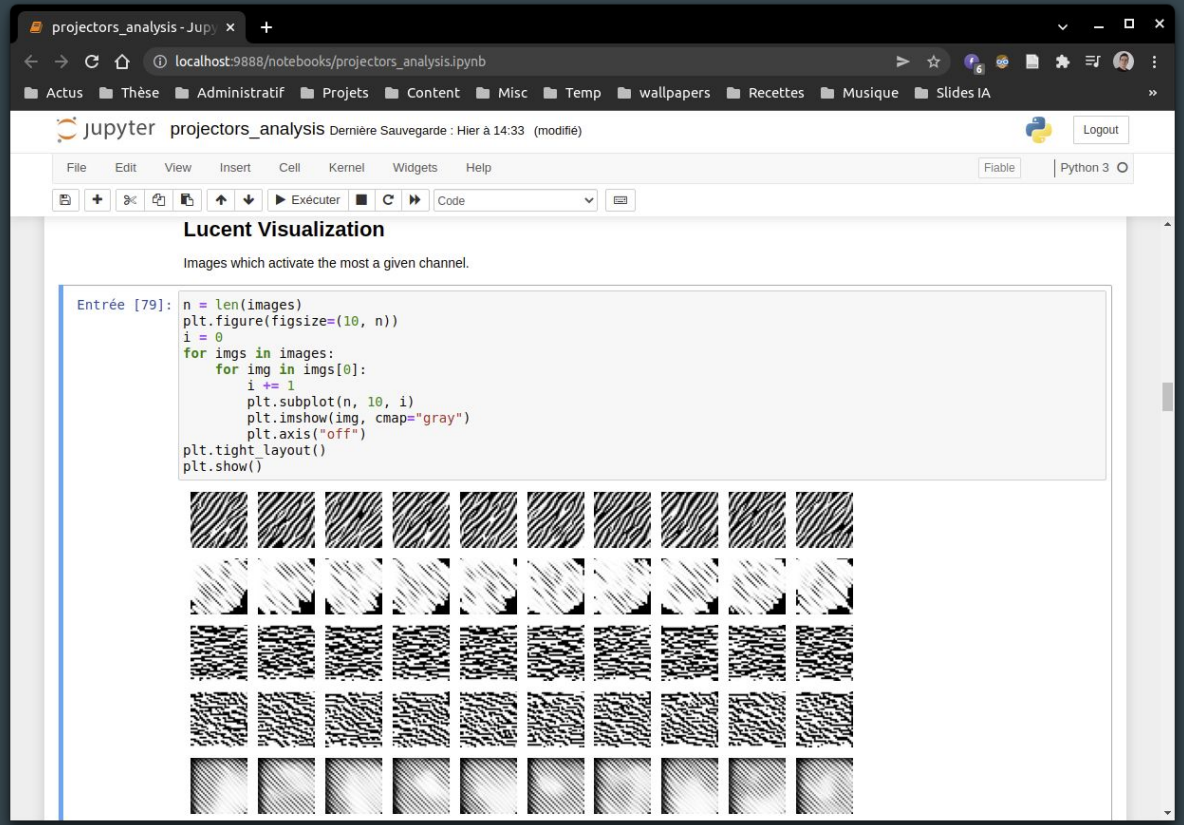
```
suffix = "train" if is_training else "val"
writer.add_scalar(f'top5_{suffix}', top5.avg, global_step=epoch)
writer.add_scalar(f'top1_{suffix}', top1.avg, global_step=epoch)
```

```
(torchjz) fguth@rusty1:~/sparse_scat_net$ tensorboard --logdir training_logs/

TensorFlow installation not found - running with reduced feature set.
Serving TensorBoard on localhost; to expose to the network, use a proxy or pass --bind_all
TensorBoard 2.3.0 at http://localhost:6006/ (Press CTRL+C to quit)
```

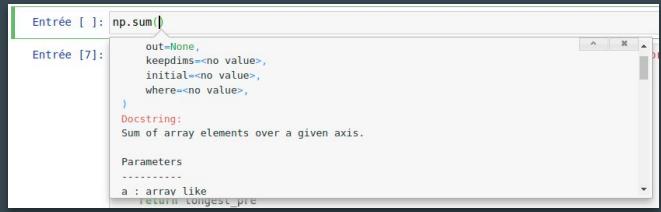


Jupyter Notebooks

```
(base) florentin@lappy:~/.../Classification/code$ jupyter-notebook
[I 2021-12-04 20:12:00.717 LabApp] JupyterLab extension loaded from /home/florentin/anaconda3/lib/python3.8/site-packages/jupyterlab
[I 2021-12-04 20:12:00.718 LabApp] JupyterLab application directory is /home/florentin/anaconda3/share/jupyter/lab
[I 20:12:00.721 NotebookApp] Serving notebooks from local directory: /home/florentin/Documents/Thèse/Classification/code
[I 20:12:00.721 NotebookApp] Jupyter Notebook 6.4.3 is running at:
[I 20:12:00.721 NotebookApp] http://localhost:8888/?token=cd37947d7d88bf263bbb0cc683ac7463813012e77e5a2d08
```



- Cells can contain python code or markdown/LaTeX
- Show outputs (e.g. plots)
- Shift-Tab for doc
- Ctrl-click for multiple cursors
- Best used with other imported files and autoreload




```
%load_ext autoreload
%autoreload 2
```

SSH

SSH Configuration

- Set .ssh/config for proxy jumps and port forwardings (run tensorboard and notebooks remotely!)
- Generate a private key for no-password logins

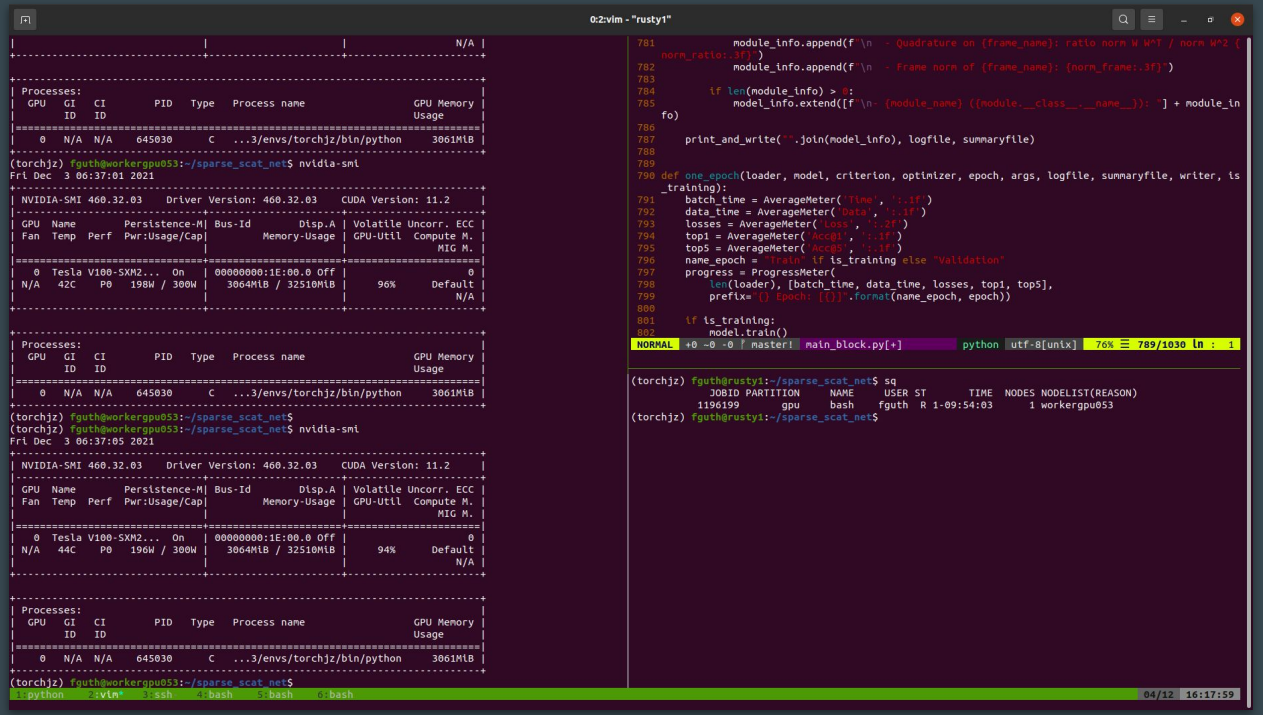
```
Host di
    Hostname di.ens.fr
    User guth
    LocalForward 8888 localhost:8888
    LocalForward 8889 localhost:8889
    LocalForward 6006 localhost:6006
    LocalForward 6007 localhost:6007
    LocalForward 6008 localhost:6008
    LocalForward 6009 localhost:6009

Host genievre
    HostName genievre
    ProxyJump di
    User guth
    LocalForward 8888 localhost:8888
    LocalForward 8889 localhost:8889
    LocalForward 6006 localhost:6006
    LocalForward 6007 localhost:6007
    LocalForward 6008 localhost:6008
    LocalForward 6009 localhost:6009
```

```
(base) florentin@lappy:~$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/florentin/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
```

```
(base) florentin@lappy:~$ ssh-copy-id genievre
```

Terminal Multiplexer

tmux



- Persistent processes across logouts
- Several terminals in one session

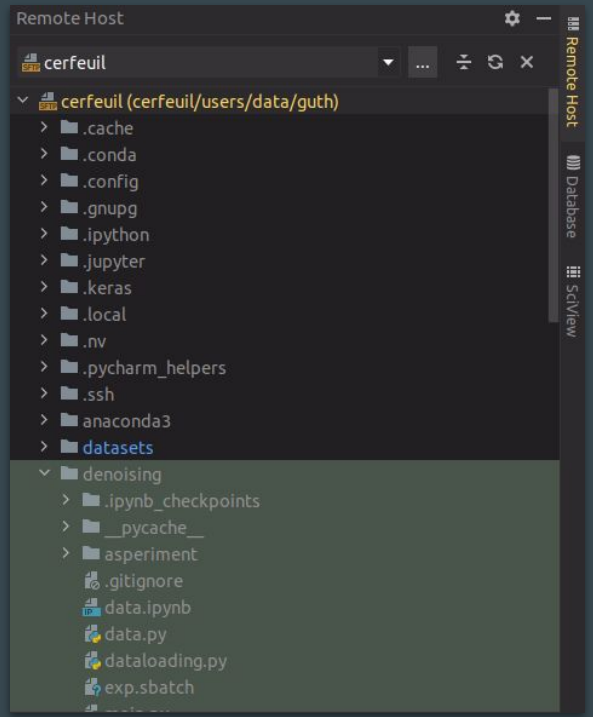
Other Bash Tools

- Aliases in .bashrc
- Search bash history with Ctrl-R
- Type man + command to access its help
- Browse files with less (or vim)
- Lost of useful commands: grep, pipes, redirections...
- Script in python rather than bash!

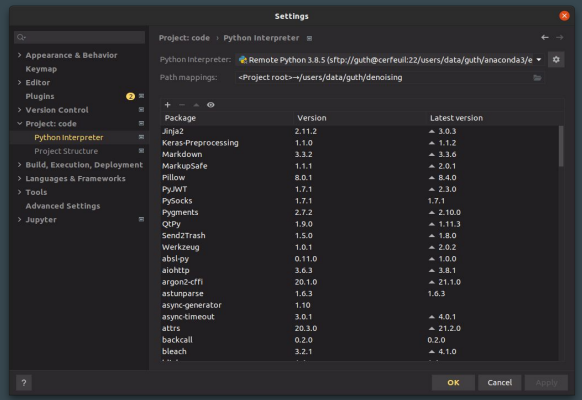

```
(reverse-i-search)`tens': tensorboard --logdir training_logs/
```

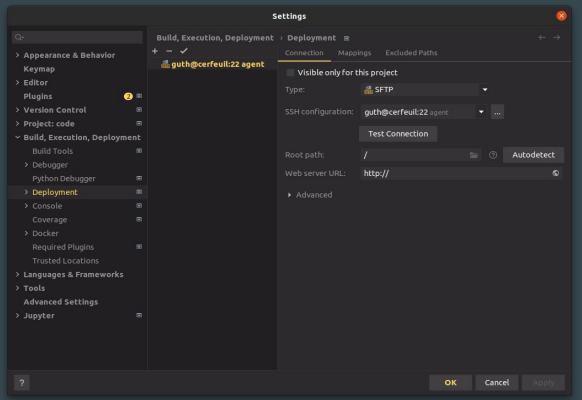
```
126 #Aliases
127 alias reload='source ~/.bashrc'
128 alias ll='ls -lrth'
129 alias ca='conda activate'
130 alias cda='conda deactivate'
131 alias tb='tensorboard --logdir'
132 alias sq='squeue -u fguth'
133 alias sra100='srun -N 1 -c 10 --mem 100G -p gpu --gpus=a100-40gb:1 --constraint=a100 --constraint
    =a100-40gb --pty bash -i'
134 alias srv100='srun -N 1 -c 10 --mem 100G -p gpu --gpus=v100-32gb:1 --constraint=v100 --constraint
    =v100-32gb --pty bash -i'
135 alias port='ssh -L 9888:localhost:9888'  # usage: port workergpuN, after srun and in another pane
```

Remote Editing and Uploading/Downloading with PyCharm



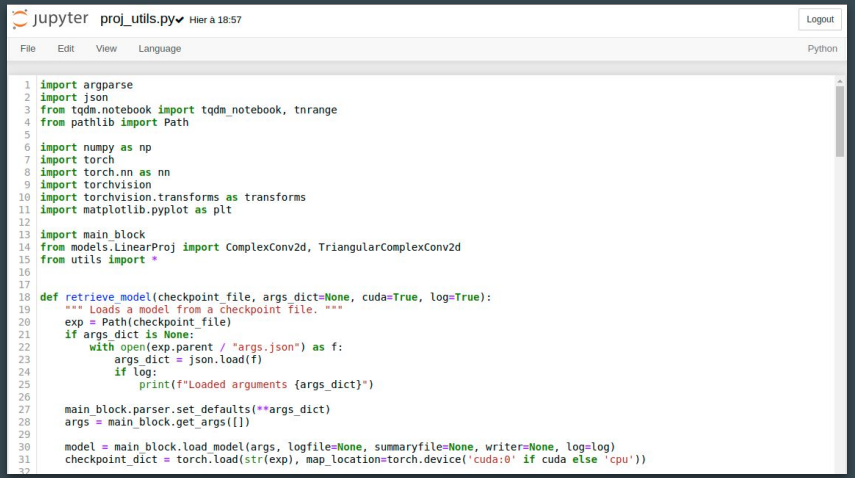
- Automatically upload files
- Remote execution and debugging
- Show remote files in the editor, download them easily





Remote Editing and Uploading/Downloading with Other Tools

- Jupyter can edit python files and produced images can be directly saved locally
- GitHub can be used to sync files
- Vim for remote editing,
  rsync for upload/download



GitHub

Vim

>_ RSYNC

Writing

LaTeX Tips

Consider creating your own preamble file with general packages and useful commands. Commands can be shorthands for common sub-expressions or a way to remember the syntax for something.

```
\usepackage{subdepth}
\let\originalleft\left
\let\originalright\right
\renewcommand{\left}{\mathopen{}\mathclose\bgroup\originalleft}
\renewcommand{\right}{\aftergroup\egroup\originalright}
```

Use this to correct maths typesetting (fix a bug in left/right, handles indices and exponents properly).

The Detexify website/app gives the command corresponding to the hand-drawn symbol.

```
\newcommand\RR{\mathbb{R}}
\newcommand\CC{\mathbb{C}}
\newcommand\NN{\mathbb{N}}
\newcommand\ZZ{\mathbb{Z}}
\newcommand\QQ{\mathbb{Q}}
\newcommand\KK{\mathbb{K}}
\newcommand\EE{\mathbb{E}}
\newcommand\PP{\mathbb{P}}

\newcommand\paren[1]{\left( #1 \right)}
\newcommand\parenn[1]{( #1 )}
\newcommand\bracket[1]{\left[ #1 \right]}
\newcommand\brackett[1]{[ #1 ]}
\newcommand\curly[1]{\left\{ #1 \right\}}
\newcommand\curlyy[1]{\{ #1 \}}
\newcommand\intint[1]{\left\llbracket #1 \right\rrbracket}
\newcommand\intintt[1]{\llbracket #1 \rrbracket}
\newcommand\norm[1]{{\left\lVert #1 \right\rVert}}
\newcommand\normm[1]{{\lVert #1 \rVert}}
\newcommand\abs[1]{{\left\lvert #1 \right\rvert}}
\newcommand\abss[1]{{\lvert #1 \rvert}}
\newcommand\inner[1]{\left\langle #1 \right\rangle}
\newcommand\innerr[1]{\langle #1 \rangle}
\newcommand\floor[1]{\left\lfloor #1 \right\rfloor}
\newcommand\floorr[1]{\lfloor #1 \rfloor}
\newcommand\expect[1]{\EE\bracket{#1}}
\newcommand\expectt[1]{\EE\brackett{#1}}
\newcommand\outerprod[2]{\left( #1 \middle\vert #2 \right)}
\newcommand\st{\,\middle\vert\,}
\newcommand\stt{\,\vert\,}

\newcommand\trans{^{\mathrm T}}
\newcommand\inv[1]{{#1}^{-1}}
\newcommand\ind[1]{\mathds{1}_{#1}}
\newcommand\diff{\mathrm{d}}
\newcommand\summ[2]{\sum_{#1 = 1}^{#2}}
\newcommand\sums{+\cdots+}
\newcommand\prodd[2]{\prod_{#1 = 1}^{#2}}
\newcommand\prods{\times\cdots\times}
\newcommand\extt[2]{\bigwedge_{#1=1}^{#2}}
\newcommand\exts{\wedge\cdots\wedge}
\DeclareMathOperator*{\argmax}{arg\,max}
\DeclareMathOperator*{\argmin}{arg\,min}
\DeclareMathOperator{\sgn}{sgn}
\DeclareMathOperator{\Tr}{Tr}
\DeclareMathOperator{\tr}{tr}
\DeclareMathOperator*{\Vect}{Vect}
\DeclareMathOperator*{\vect}{vect}
\DeclareMathOperator*{\Span}{Span}
\DeclareMathOperator*{\spann}{span}
\DeclareMathOperator{\Ker}{Ker}
\DeclareMathOperator{\Imm}{Im}
```

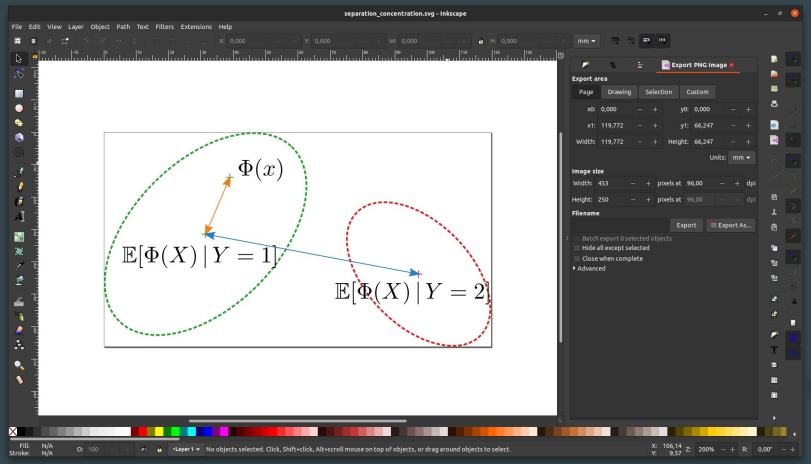
Generating High-Quality Tables and Figures

The booktabs package produce pretty tables (using only horizontal rules is recommended).

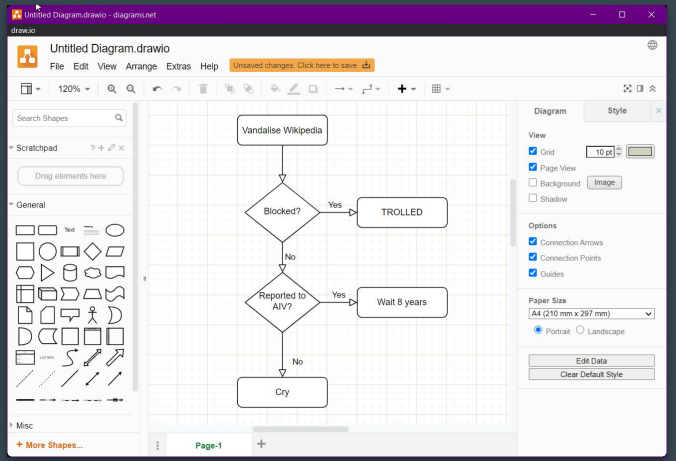
	$\Phi(x)$	x	$F^T \rho F x$			$S_T(x)$
			$\rho = \rho_a$	$\rho = \rho_t$	$\rho = \rho_{rt}$	
MNIST	Error (%)	7.4	1.3	1.4	1.3	0.8
	Fisher	19	68	69	67	130
CIFAR	Error (%)	60.5	28.1	34.8	26.5	27.7
	Fisher	6.7	15	13	16	12

```
matplotlib.rcParams.update({'text.usetex': True, 'font.family': 'serif'})  # 'sans-serif' for Beamer
plt.savefig("figure.pdf", transparent=True, bbox_inches="tight", pad_inches=0)
```

Generate pdf output with matplotlib, use LaTeX fonts for axe labels and legends.

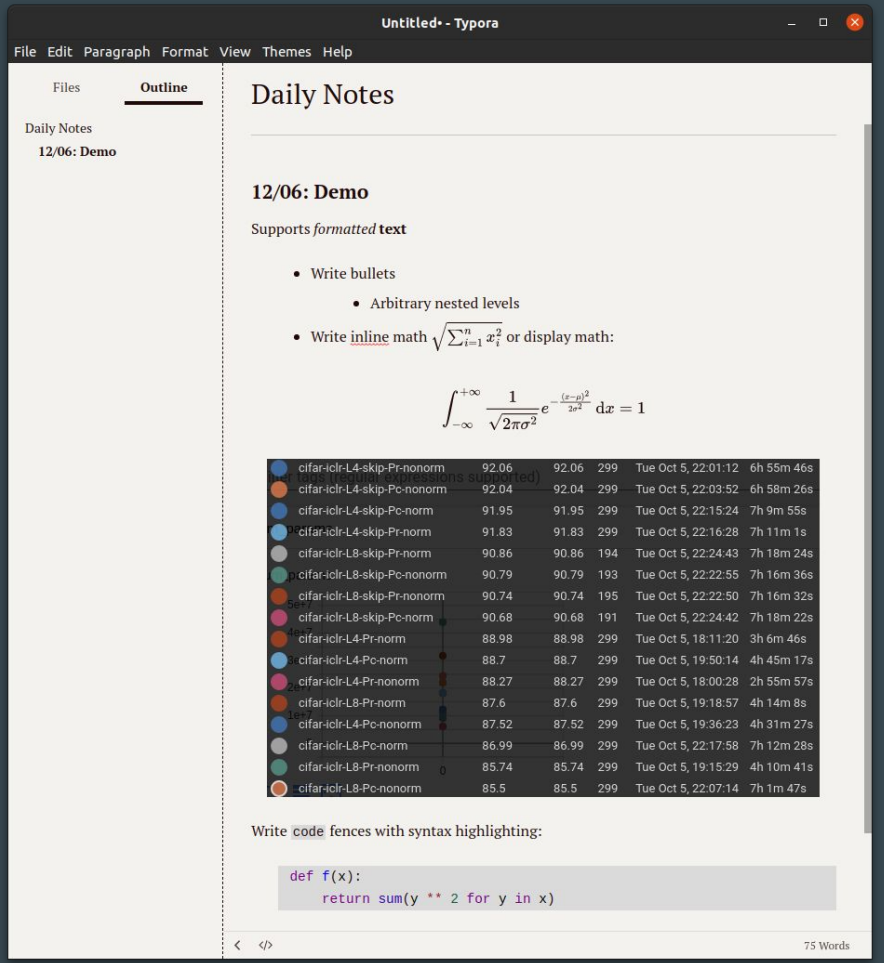


Create pdf figures with diagrams.net or Inkscape.



Taking Notes with Typora

Markdown editors like Typora are useful for daily notes. They support rich text: table of contents, images, LaTeX, tables, code...



Summary

- Invest time in your tools and skills, it’s worth it
- Explain things in natural language before coding or debugging
- Strive for generality and simplicity

Thank you!