



# On Python and Tensors

## *Initial Considerations*

Alexander G. Ororbia II

COGS-621: Foundations of Scientific Computing

9/2/2025

# High performance computing (HPC)

- Use of supercomputers or computer clusters (grids) to tackle complex computation problems
- Will have guest lecture on basics of RIT research computing (RC)
  - RIT RC:  
<https://www.rit.edu/researchcomputing/>
  - (We will have a guest lecture / walkthrough on Sept 16 -- Viet -- on using RIT RC)
- Python is the “glue” (part of multilanguage model of HPC)
  - Ecosystem = NumPy, SciPy, and Matplotlib



**The Oak Ridge Laboratory  
Frontier Supercomputer**

## Environments

IPython console, Jupyter notebook, Spyder, ...

## Python interpreter

Python 3, ...

## Python packages

numpy, scipy, matplotlib, ...

## System and system libraries

OS, BLAS, LAPACK, ...

# Your toolset

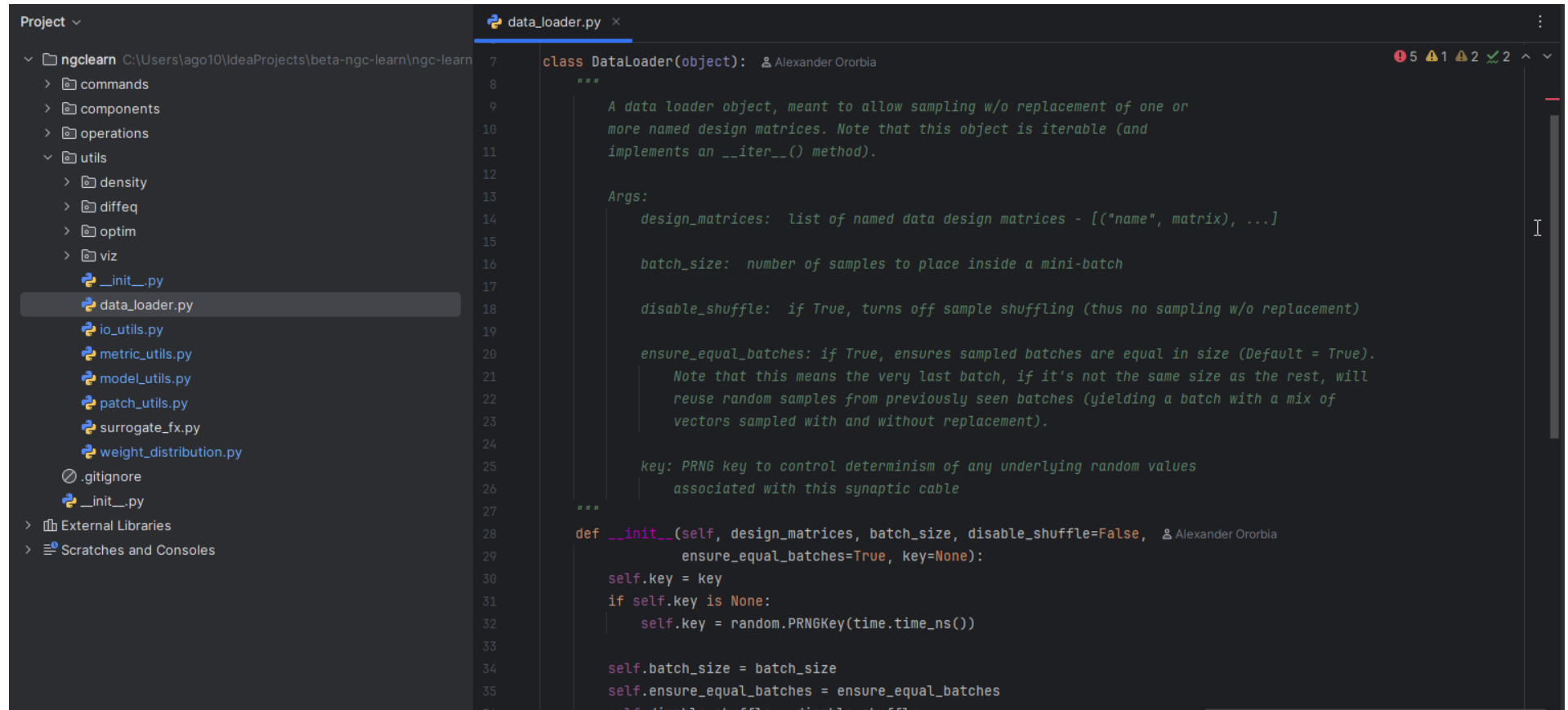
Some of the tools to engage with scientific computing

# Python and development environments

- Vi / Vim, Emacs (simpler / bare-bones, classical)
- Atom, Gedit (GNOME Editor)
- Pycharm (an integrated development environment; IDE)

**A JetBrains  
PyCharm view of  
some ngclearn  
source code!**

**(PyCharm  
community edition  
is free and available  
for multiple OS)**



The screenshot displays the PyCharm IDE interface. On the left, the 'Project' sidebar shows the file structure of the 'ngclearn' project, with 'data\_loader.py' selected. The main editor window shows the code for the 'DataLoader' class. The code includes a docstring, arguments, and an initialization method.

```
class DataLoader(object):  
    """  
    A data loader object, meant to allow sampling w/o replacement of one or  
    more named design matrices. Note that this object is iterable (and  
    implements an __iter__() method).  
    """  
    Args:  
        design_matrices: list of named data design matrices - [("name", matrix), ...]  
        batch_size: number of samples to place inside a mini-batch  
        disable_shuffle: if True, turns off sample shuffling (thus no sampling w/o replacement)  
        ensure_equal_batches: if True, ensures sampled batches are equal in size (Default = True).  
            Note that this means the very last batch, if it's not the same size as the rest, will  
            reuse random samples from previously seen batches (yielding a batch with a mix of  
            vectors sampled with and without replacement).  
        key: PRNG key to control determinism of any underlying random values  
            associated with this synaptic cable  
    """  
    def __init__(self, design_matrices, batch_size, disable_shuffle=False, key=None):  
        self.key = key  
        if self.key is None:  
            self.key = random.PRNGKey(time.time_ns())  
        self.batch_size = batch_size  
        self.ensure_equal_batches = ensure_equal_batches  
        self.disable_shuffle = disable_shuffle
```

# Executing Python scripting code

- There are many ways to “run” code (execute Python interpreter)
  - Write / dynamically execute code “snippets” w/ Python interpreter directly
  - Use IPython interactive environment
  - Set up and iteratively build Python (Spyder) notebooks
  - Write Bash (terminal-level) scripts to call Python files (Linux-oriented)
- You will want to develop your own “workflow” as well as consider what your target audience will be
  - Notebooks can be useful for tutorials/demo(s), executable via web-browser
  - Interacting w/ Python interpreter can be helpful for quick sanity checks and personal self-study (good to do in this class)
  - Bash-scripting is old-fashioned but robust and reliable (it’s what I do my research, because I’m old)

# Version control

- Source code management (also known as “revision control”)
  - Key software engineering practice of controlling, organizing, & tracking different versions of your software/scripts
  - *Backbone*: a history of computer files (e.g., Python text, supporting file types like images, etc.)
  - Useful for publicly “releasing” your code (to engender *scientific reproducibility*)
- Quality (free) tools:
  - Bitbucket
  - Github
    - Good to have, and very useful for team / multi-person projects
    - You should sign up for the “student developer pack” (gets you lots of important features like private repos)



# Github: One possible simple (work)flow



- 1) Git clone / set up your (local) repository (repo)
- 2) Write your code / folders (do things locally)
- 3) Git add / commit your code (differences) to the repo itself
  - `git add <source_code_file/folder>` // this points to what code is to be added
  - `git commit -m "<insert message>"` // this gets your code ready to upload
  - `git push` // this physically places your code (or changes) onto repo
  - `git status`

# **Developing some of the basics**



# Vectorized operations / objects

```
import numpy as np
```

- Vectors, matrices, tensors ( $> 2^{\text{nd}}$  order arrays)
  - Essential tool for numerical computation
  - Useful data representation, especially for operations (ops) that are repeated for set of values
- Vectorization – eliminates the need for explicit loops (batch operations applied to data)
  - In Python, ops often built on lower-level libraries (BLAS, LAPACK, XLA< etc.)
- Essentially what NumPy, i.e., *numpy* (and *numpy.linalg*), is for
  - Built on C (for processing/manipulating arrays)
  - All elements in a numpy (fixed-size) array are of the same type (homogeneous)
  - These arrays have in-built operations to be applied to them (along with functions/modules that work with these data structures)

# Questions?

