Coding standards

- Coding standards are guidelines for code style and documentation
- They may be formal PEP 8(Style Guide for Python Code) standards, or company specific standards

Why bother with a coding standard?

- Consistency between developers
- Ease of maintenance and development
- Readability, usability, security, performance
- If you deviate from the standard for any reason, document it

“Code is read much more often than it is written"

------ Guido van Rossum

PEP 8 Recommendations Highlight

- Avoid mixing tabs and spaces, recommend using spaces
- Use 4 spaces per indentation level
- Line length <=72 chars for block comments; <= 79 chars for all other lines;

- Naming conventions:
  - Modules should be all lowercase. Underscores can be used if it improves readability
  - Class names should be in PascalCase (class MyClass:)
  - Function names and instance variables should be lowercase with underscores as necessary
  - Constants should be ALL_CAPITAL_WITH_UNDERSCORES

- Order of import statements:
  1. Standard library imports.
  2. Related third party imports.
  3. Local application/library specific imports.
  4. put a blank line between each group of imports.

Not recommended	Best Practice
from qcore import geo import os, sys from shared_workflow import shared from subprocess import Popen, PIPE import vtk	import os import sys from subprocess import Popen, PIPE import vtk from qcore import geo from shared_workflow import shared
from math import *	import math or from math import ceil

Not recommended	Best Practice
if number != None:	if number is not None:
if flag == True:	if flag:
# Bad: use global statement: global WIDTH	# Avoid or Possibly use class instead self.width

# Bad: use Magic number 	
fraction = available_nodes / 264
	# define as a constant instead:
	MAX_NODES = 264

person = {
'first': 'Tobin',
'last': 'Brown',
'age':20
}
 	
# Bad: we have to change the replacement fields within 	
our string, once we add new values
print('{} {} is {} years old'.format(
person['first'],
person['last'],
person['age'])
)
	print('{first} {last} is {age} years old'.format(**person))
	
	# Output
	Tobin Brown is 20 years old

Use comprehension for simple cases

For simple transformations that can be expressed as a list comprehension, use list comprehensions over map() or filter(). Use map() or filter() for expressions that are too long or complicated to express with a list comprehension.
Although a map() or filter() expression may be functionally equivalent to a list comprehension, the list comprehension is generally more concise and easier to read.

Not recommended	Best Practice
values = [1, 2, 3] doubles = map(**lambda** x: x * 2, values)	values = [1, 2, 3] doubles = [x * 2 **for** x **in** values]

exec

The exec statement enables you to dynamically execute arbitrary Python code which is stored in literal strings. Building a complex string of Python code and then passing that code to exec results in code that is hard to read and hard to test. Anytime the use of exec error is encountered, you should go back to the code and check if there is a clearer, more direct way to accomplish the task.

Not recommended	Best Practice
s = "print(\"Hello, World!\")" exec(s) # Output Hello, World!	def print_hello_world(): print("Hello, World!") print_hello_world()

Eg. In slurm_gm_workflow/install.py, this statement needs fixing, as it's hard to find what params it imports, would they have naming conflicts with the variables inside the current script etc.

```
with open(params_vel_path, "r") as f:
    exec(f.read(), globals())
```

Comments

- Decide a docstring style
- Comment clearly in the right place
- Update comments when codes changes

Good examples:

- Comments at top to describe the purpose of a script

```
#!/usr/bin/env python3
"""
Functions for easy estimation of WC, uses pre-trained models.
"""
import os
import glob
```

- Comments to describe what does this line of code do, so it's easier to track back and for others to understand

```
# Make a numpy array of the input data in the right shape.
# The order of the features has to the same as for training!!
data = np.array(
    [float(nx), float(ny), float(nz), float(nt), float(ncores)]
).reshape(1, 5)
```

- Comments to describe the purpose and params of function

```
def estimate_BB_chours(
    data: np.ndarray,
    model_dir: str = BB_MODEL_DIR,
    model_prefix: str = MODEL_PREFIX,
    scaler_prefix: str = SCALER_PREFIX,
):
    """Make bulk BB estimations, requires data to be
    in the correct order (see above)

    Params
    ------
    data: np.ndarray of int, float
        Input data for the model in order fd_count, hf_nt, n_cores
        Has to have shape [-1, 3]

    Returns
    -------
    core_hours: np.ndarray of floats
        Estimated number of core hours
    run_time: np.ndarray of float
        Estimated run time (hours)
    """
    if data.shape[1] != 3:
        raise Exception(
            "Invalid input data, has to 3 columns. " "One for each feature."
        )
```

Documentation

Codes are easy to forget

Use UC wiki to document necessary projects: what it is, how to use etc. Include examples!

Auto Coding format – Black

https://github.com/ambv/black

- Coding style is a strict subset of PEP 8
  Note the line length in Black is default to 88.
  To reset, use option *--line-length 79*

- Really useful in a team, save time and money
  You will have fewer discussions on where spaces should be,
  where a new line should be inserted etc and faster pull request processes.

Summary

Comply with PEP8
Find a standard that suits our projects and stick to it

Further readings

https://www.python.org/dev/peps/pep-0008/

http://google.github.io/styleguide/pyguide.html

http://docs.quantifiedcode.com/python-anti-patterns/