Python DSE stylesheet

Group 02

May 30, 2024

1 Introduction

The code shall use styling consistent with PEP8 as outlined in here. The most important points are summarized and the code flow type to be used shall be explained.

2 Code flow

This section details some coding principles to be followed. The larger examples here do not necessarily show good style, for that refer to the Styling section.

Importability

Every file containing a function or a class which can be used in another file should be importable without side-effects. This means that there should be no bare scripting when importing the file. To prevent this use:

```
if __name__=="__main__":
    # You can call functions and do scripting here
    foo(bar=None)
```

Furthermore it is recommended that the code shall use a `def main()` function instead of scripting outside of a function:

```
def main():
    # Even better to call functions and do scripting here
    foo(bar=None)


if __name__=="__main__":
    main()
```

Class style

Every tool/class should have a function that can be called that can give the final answer on it's own. For example when a tool calculates a mass, there should be a function `def get_mass(self, paramater)` or similar which ensures all necessary calculations are done. Two examples of how this can be done are:

```
class Rotor:
    # Helper functions omitted

    def calc_masses(self, T_required_tot, t_flight, E_spec):
        '''Calculate propulsion and battery masses for rotor only flight'''
        T_req_pr = T_required_tot / self.N

        self.required_params_per_rotor(T_req_pr)
        self.calculate_A_blade_total(T_req_pr)
        self.calculate_ct_and_solidity(T_req_pr)
        self.calculate_omega()
        self.calculate_power_per_rotor(T_req_pr)
        self.calculate_power_total()
        self.calculate_total_energy(t_flight)

        ### Calculate masses
        W_sys = {}

        W_sys['prop']    = ENV['g'] * (self.calculate_rotor_group_mass(T_req_pr) + self.
    calculate_motor_mass())
        W_sys['battery'] = ENV['g'] *  self.calculate_battery_mass(t_flight, E_spec)

        return W_sys
```

where one function calls many functions and defines the flow, and

```
class Horizontal:
    #Helper functions omitted

    def Getweight(self, S, W_prop, W_bat_v):
        '''
        Inputs:
```

```
        S         [m^2] Wing surface area              array-like
        W_prop    [N]   Weight of propulsion system    array-like
        W_bat_v   [N]   Weight of battery for vert     array-like
        Outputs:
        W_tot     [N]   Total weight of iteration      array-like
        '''

        cfV = self.configV
        Ws = self.Getweightsys(S)

        Ws['wing'] *= 1+cfV['tiltwingWF']+cfV['empennageWF']

        W = (1+cfV['contingency']) * (
              Ws['wing']
            + Ws['fuselage']
            + W_prop
            + Ws['battery'] + W_bat_v
            + Ws['const']
            )

        self.m = W/ENV['g']

        return W
```

where each function calls the function it depends on by itself. Both have their advantages and disadvantages: the former is easier to test and the latter is easier to call intermediates and still get a sensible answer.

Classes are expected to be used, even if no OOP principles are used. This makes the code much more readable and easier to call, as functions are grouped by the class. Storing certain intermediate values as class attributes instead of putting these in the function call can make the code much more readable and manageable however and should be done if possible.

Function style

Functions are expected to be in S.I. when called or returning a value, unless it is made **very** clear that this is not the case.
If a function uses many variables when called, such as:

```
class ExampleClass:
    def example(self, config,  b, m_init, M_MO, V_stall, t_f=0.0015,  rho_f=2699, CL_max
    =1.5, w_max=4.5, update_b=True, Range=20000, Endurance=1800, Cfc=0.005, LambdaLE=0,
    e_min=0.7, e_max=0.85, eta_prop=0.6, eta_power=0.7, E_spec=9e5, Lf=3, Rf=0.15):
```

the ordering can be unclear when called with only positional arguments:
`object.example(2, 3, 5, 3, 6, 2, 1, 3, 4).`
To remedy this use at most 5 positional arguments (self is not an argument). You can force the use of keyword arguments with a `*`, after which no positional arguments are allowed by python, see:

```
class ExampleClass:
    def example(self, config,  b, m_init, M_MO, V_stall, *, t_f=0.0015,  rho_f=2699,
    CL_max=1.5, w_max=4.5, update_b=True, Range=20000, Endurance=1800, Cfc=0.005,
    LambdaLE=0, e_min=0.7, e_max=0.85, eta_prop=0.6, eta_power=0.7, E_spec=9e5, Lf=3, Rf
    =0.15):
```

which forces to the following function call:
`object.example(2, 3, 5, 3, 6, t_f=2, rho_f=1, CL_max=3, w_max=4).`

3 Styling

Imports

Imports should be in the order: Standard libraries, third party libraries (e.g. numpy) and last local/self made libraries/files. See:

```
# Standard libraries
import math
import random

# Third party libraries
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
import sys

# Local imports
import FolderExample.generic_import as generic_import
import FolderExample.generic_import
```

Whitespace

Indent size should be 4 spaces (not tabs, likely automatically replaced by your editor however).
There should be 2 empty lines after imports.
There should be 2 empty lines before and after a class and top level functions.
There should be 1 empty line between functions in a class.

For an example, see the style example in /Style/style_example.py

Naming

Folders shall use CamelCase
Filenames shall use snake_case
Classes shall use CamelCase
Functions shall use snake_case
Variables shall use snake_case

New variables should be clear, unambiguous, and introduced with a comment and unit if applicable. Fully descriptive words are preferred, unless it is a commonly known acronym which is described by either the function docstring or in a comment. Common acronyms are:

- max: maximum
- num: number
- min: minimum
- opt: optimal
- req: required

Docstrings

There are two styles of docstrings used. The single line function description, and the more elaborate, and function description and input/output description. Examples of both are:

```
def foo(bar="fizzBuzz", foo_bar=None):
    '''This function does something great'''


def bar(foo, bar_bar, foo_bar=True):
    '''
    This function calculates the bar-foo force

    Positional arguments:
```

```
    foo       [-]       Float        The foo coefficient
    bar_bar   [m/s]     Array-like   The bar speed

    Keyword arguments:
    foo_bar   [-]       Bool         Whether or not foo is at bar

    Returns:
    bar_foo   [N]       Array-like   The bar-foo force
    '''
```

Ordering

The code is expected to be ordered as

- Import
- Top-level functions
- Classes
- Scripting (main function)