**CODING STANDARDS:**

The project adheres to the PEP8 styling guide for python code.

* **File naming convention:**
* Python files should have short, intuitive, all- camel case names.
* Underscores can be used in the file name if it improves readability.

Yes: actionsTraining.py

mvDerivation\_Common.py

* **Organizing imports:**
* Imports are always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.
* Imports should usually be on separate lines:

Yes: import os

import sys

No: import sys, os

* It's okay to say this though:

from subprocess import Popen, PIPE

* Imports should be grouped in the following order:

1. Standard library imports.
2. Related third party imports.
3. Local application/library specific imports.

A blank line should be put between each group of imports.

* Absolute imports are recommended, as they are usually more readable and tend to be better behaved (or at least give better error messages) if the import system is incorrectly configured (such as when a directory inside a package ends up on sys.path):

import mypkg.sibling

from mypkg import sibling

from mypkg.sibling import example

* When importing a class from a class-containing module, it's usually okay to spell this

from myclass import MyClass

from foo.bar.yourclass import YourClass

If this spelling causes local name clashes, then spell them explicitly:

import myclass

import foo.bar.yourclass

and use "myclass.MyClass" and "foo.bar.yourclass.YourClass".

* Wildcard imports (from <module> import \*) should be avoided, as they make it unclear which names are present in the namespace.
* **Indentations:**
* Use 4 spaces per indentation level.
* Avoid using more than five levels of indention. In other case, try breaking the logic and modularise it.
* Spaces are the preferred indentation method. Tabs should be used solely to remain consistent with code that is already indented with tabs.
* **String Quotes:**
* In Python, single-quoted strings and double-quoted strings are the same.
* PEP does not make a recommendation for this. Pick a rule and stick to it.
* When a string contains single or double quote characters, however, use the other one to avoid backslashes in the string. It improves readability.
* **Naming convention for instance variables and method names:**
* All names should be short and descriptive of their use.
* The “Style Guide for Python Code” recommends using lowercase with words separated by underscores (example: my\_variable). However, mixedCase can also be used where it supports readability (example: myVariable ).
* Use one leading underscore only for internal methods and instance variables (i.e. protected). Example: \_myProtectedVar
* Use two leading underscores to denote class-private names Example: \_\_myPrivateVar
* Don’t use leading or trailing underscores for public attributes unless they conflict with reserved words, in which case, a single trailing underscore is preferable (example: class \_)
* All identifiers and method names must be ASCII compatible.
* **Whitespace in expressions and statements:**
* Avoid extraneous whitespace in the following situations:
* Immediately inside parentheses, brackets or braces.

Yes: spam(ham[1], {eggs: 2})

No: spam( ham[ 1 ], { eggs: 2 } )

* Between a trailing comma and a following close parenthesis.

Yes: foo = (0,)

No: bar = (0, )

* Immediately before a comma, semicolon, or colon:

Yes: if x == 4: print x, y; x, y = y, x

No: if x == 4 : print x , y ; x , y = y , x

* However, in a slice the colon acts like a binary operator, and should have equal amounts on either side (treating it as the operator with the lowest priority). In an extended slice, both colons must have the same amount of spacing applied. Exception: when a slice parameter is omitted, the space is omitted.

Yes:

ham[1:9], ham[1:9:3], ham[:9:3], ham[1::3], ham[1:9:]

ham[lower:upper], ham[lower:upper:], ham[lower::step]

ham[lower+offset : upper+offset]

ham[: upper\_fn(x) : step\_fn(x)], ham[:: step\_fn(x)]

ham[lower + offset : upper + offset]

No:

ham[lower + offset:upper + offset]

ham[1: 9], ham[1 :9], ham[1:9 :3]

ham[lower : : upper]

ham[ : upper]

* Immediately before the open parenthesis that starts the argument list of a function call:

Yes: spam(1)

No: spam (1)

* Immediately before the open parenthesis that starts an indexing or slicing:

Yes: dct['key'] = lst[index]

No: dct ['key'] = lst [index]

* Avoid trailing whitespace anywhere. Because it's usually invisible, it can be confusing: e.g. a backslash followed by a space and a newline does not count as a line continuation marker.
* Always surround these binary operators with a single space on either side: assignment (=), augmented assignment (+=, -= etc.), comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not), Booleans (and, or, not).
* Don't use spaces around the = sign when used to indicate a keyword argument, or when used to indicate a default value for an unannotated function parameter.

Yes:

def complex(real, imag=0.0):

return magic(r=real, i=imag)

No:

def complex(real, imag = 0.0):

return magic(r = real, i = imag)

* Compound statements (multiple statements on the same line) are generally discouraged.

Yes:

if foo == 'blah':

do\_blah\_thing()

do\_one()

do\_two()

do\_three()

Rather not:

if foo == 'blah': do\_blah\_thing()

do\_one(); do\_two(); do\_three()

* While sometimes it's okay to put an if/for/while with a small body on the same line, never do this for multi-clause statements. Also avoid folding long lines!
* **Blank Lines:**
* Surround top-level function and class definitions with two blank lines.
* Method definitions inside a class are surrounded by a single blank line.
* Extra blank lines may be used (sparingly) to separate groups of related functions.
* Use blank lines in functions, sparingly, to indicate logical sections.
* **Comments:**
* Comments should describe the underlying purpose of the piece of code they apply to, rather than stating the obvious.
* Always make a priority of keeping the comments up to date when the code changes!
* Block comments generally apply to some (or all) code that follows them and are indented to the same level as that code. Each line of a block comment starts with a # and a single space.
* An inline comment is a comment on the same line as a statement. Inline comments should be separated by at least two spaces from the statement. They should start with a # and a single space.
* Use inline comments sparingly.