1) Coding Challenge

Guidelines:

- *Language: Python 3.8*
- *You may use any package from the PyPI*

An important task in mass photometry is the interpretation of the measured mass distributions. "eventsFitted.csv" (attached) is an output file of our data analysis program DiscoverMP. Each line in that file contains information about one particle that was detected during the measurement.

1) Write a class that can be initialised with an existing "eventsFitted.csv" file and implement the following methods
    a. `calc_average_mass()`: returns the average mass of all particles
    b. `calc_median_mass()`: returns the median mass of all particles
    c. `plot_histogram(png_filename)`: generates an image that displays the mass histogram with the average and median mass as two vertical lines. Writes this image to the provided filename.

2) Write unit tests for the `calc_median_mass()` and the `calc_average_mass()` method.

3) What does your histogram tell you about the chemical sample represented by the data?

4) What changes would you make to the requirements of the class to make it more useful for interpreting the data?

2) Code Review Challenge

The following is a utility class to provide temporary storage for recent measurements in a data streaming context. It behaves as though it were an infinitely indexable array, but internally it overwrites old data as it runs out of space.

Your task is to provide a review of this code, as if a junior colleague had asked you to assess whether it is suitable to become part of the main codebase.
You may highlight any concerns that you find, including any concerns about coding style, interface usability, or any bugs. You may also draw attention to anything you particularly like.

This code has been deliberately written for this exercise, so please do not worry about offending the author through any criticism you identify.

Code:

```
from typing import Generic, Optional, overload, TypeVar, Union
import numpy as np

ValType = TypeVar("ValType")


class RecentHistoryBuffer(Generic[ValType]):
    def __init__(
        self,
        length: int,
        *,
        dtype=np.float64,
        fill: Optional[ValType] = None,
    ) -> None:
        self._l = length
        self._last_written_index = 0
        self._data: np.ndarray = np.empty(shape=(length,), dtype=dtype)
        self.fill(fill)

    def min(self) -> ValType:
        # Get the lowest value in the buffer.
        # Useful for minimum recently seen problems.
        return np.min(self._data)

    def max(self) -> ValType:
        # Get the lowest value in the buffer.
        # Useful for maximum recently seen problems.
        return np.max(self._data)

    def getArr(self) -> np.ndarray:
        return self._data

    def fill(self, fill: Optional[ValType] = None) -> None:
        """
        Fill the buffer with the provided value for initialisation
        """
        if fill is None:
            self._data[:] = np.nan
        else:
            self._data[:] = fill

    def __len__(self) -> int:
        self._l

    def __setitem__(self, key: int, value: ValType) -> None:
        self._last_written_index = max(key, self._last_written_index)
        self._data[key % self._l] = value

    @overload
    def __getitem__(self, key: int) -> ValType:
        ...

    @overload
    def __getitem__(self, key: slice) -> np.ndarray:
        ...
```

```
56
57      def __getitem__(self, key: Union[int, slice]) -> Union[ValType, np.ndarray]:
58          if isinstance(key, int):
59              assert (
60                  key <= self._last_written_index
61              ), "Reading past the last written value"
62              return self._data[key % self._l]  # type: ignore
63          elif isinstance(key, slice):
64              assert (
65                  key.stop <= self._last_written_index
66              ), "Reading slice past last written value"
67              assert key.stop > key.start, "RingBuffer slice stops before it starts"
68              assert (
69                  key.stop - key.start <= self._l
70              ), "RingBuffer slice is longer than buffer length"
71              s = slice(
72                  key.start % self._l,
73                  key.stop % self._l,
74                  key.step,
75              )
76              return self._data[s]
77
```

Usage example:

```
# Initialisation
laser_power_history = RecentHistoryBuffer(length=50)
index = 527 # This would be provided by an external coordination mechanism.


...
# Some worker thread reads and records values into the buffer
laser_power_history[index] = read_laser_power()

...
# Some other thread confirms that the recent values are high enough, e.g.
if laser_power_history.max() < LASER_POWER_THRESHOLD:
    # Raise an exception to say the system hasn't turned on yet
if laser_power_history[index-10: index].mean() > laser_power_history[index]:
    # Alert the user that the value is falling
```