Q1.2

1. Since passing a structured object makes the inside attributes all wrapped up, it is more intuitive and modular and avoids confusion about parameter order, the readability and maintainability is maintained.
2. Since the all attributes are wrapped in an object, adding new attributes only requires updating the data class without modifying function parameters (signatures), so the extensibility is enhanced.

Q1.3

I have delete the import in `training.py` and used the string literal for type hint: `def update_stats(self, stats:"PokemanTrainingStats")`

Q1.4

We can extract the modules (in this case the PokemanTrainingStats class) that multiple files needs to use into a separate, independent file, and different dependent files can import it separately.

Q2.1

To accept arbitrary attributes, can pass the attributes wrapped up in a dictionary. And use setattr() and getattr() to set and get these attributes.

Q2.2

Can make use of the dictionary in Q2.1 to query a specific attribute with its key.

Q2.3

If generalize the getting attributes to a single function:

1. Firstly the safety of the code is reduced, because using a same function the caller can get any attributes the caller want to call (by directly using the `getattr`) and can set any attribute the caller want to set (by directly using `setattr`), this less safe than using explicit calling because if each attribute has their corrsponding methods, called the method for one attribute won’t affect other attributes.

2. Even if the caller don’t act maliciously, mis-using these methods can increase the probability of runtime error (Because the function calling is not as intuitive as the original version so user might make mistakes, and also it is harder for python to do type checking, etc).
3. Also, wrapping all the attributes together will decrease the clarity and maintainability of the whole code.