Approach

While writing the program which finds the anagrams I was trying to write the code which is based on maintainability, reusability, performance and scalability. My approach is based on separating logic for the functionalities which allows to test the program easier and develop it further in the future. Each method has its responsibility. So they don't intersect with each other and are independent logically.

```python
def read_file() -> List[str]:
    with open("./sample.txt", "r") as f:
        return [line.strip() for line in f]
```

- I used the *"with"* keyword to handle auto-closing the file.
- In *"open()"* i passed 2 arguments, first one is the file path i want to be read, second is the action which i am going to do (*"r"* for reading)
- For the return statement I used list comprehension to improve the readability of the code and also to short it.

```python
def get_anagrams(words: List[str]) -> Dict[tuple, List[str]]:
    results = defaultdict(list)

    for word in words:
        key_word = tuple(sorted(Counter(word).items()))
        results[key_word].append(word)

    return results
```

- The parameter of functions retrieves a list of words where the anagrams should be detected from
- As *"results"* I decided to use *"defaultdict"* which stores the letters frequency as the keys, and the words as the values in a list. I preferred *"defaultdict"* over the *"dict"*, because I am gonna make grouping, so I do not need key errors, for new keys I just need the empty list to be set as default.
- Going through the words list:
    - Casting the word to counter to calculate the frequency of appearance for each letter.
    - Sorting the list before setting it as a key in dict. So for any words which have the same letters, the key will be the same.
    - Adding the actual word to the value list for the key.
- Returning the dict as a response.

```
if __name__ == "__main__":
    anagrams = get_anagrams(read_file())
    for list_value in anagrams.values():
        print(' '.join(list_value))
```

- Using if statement in front to be sure for the safety and that the code is not gonna be imported in other files.
- Getting the anagrams calling the respective method passing as argument the list gotten from file
- Print the value for each key using

Maintainability & Reusability

- *read_file()* handles reading data from file (I/O)
- *get_anagrams()* handles the logic of words validation and contains the main algorithm which determines the anagrams

Due to the methods being independent they may be used in other places.

Scalability, Performance

Scalability:
For larger datasets for improving the performance and avoid over usage of memory would be good to use generator

Complexity:
"Counter" runs on O(k) complexity per word. Due to sorting list, complexity evolves to O(k * logk)
The final algorithm complexity is O(n * k * log k) where *"n"* is the number of words and the *"k"* is the length of word

Data structures:

- defaultdict - is highly optimized by default and removes overhead for manual handling the key existence
- Counter - implemented by C, used to speed up code interpreting, so to speed up my code