## Aufgabe 1: Anagramme

Ein Anagramm ist eine Buchstabenfolge mit der durch Permutation der Buchstaben verschiedene Wörter gebildet werden können. Bsp.: Tor, Ort.

**Aufgabenbeschreibung**: Gegeben ist eine Datei (anagram.txt), in der je Zeile ein Wort (ohne Leerzeichen, keine Sonderzeichen, nur Kleinbuchstaben) geschrieben steht. Schreiben Sie ein Programm, das die Datei mit den Wörtern einliest, die Anzahl der Anagramme eines jeden Worts ermittelt und das Ergebnis in eine neue Datei (anagram_output.txt) schreibt. Es soll für die Ausgabe in der Datei jeweils nur das erste gefundene Wort eines Anagramms verwendet und dessen Häufigkeit mit angegeben werden.

*Die Eingabedatei enthält bspw. folgenden Wörter:*

```
atlas
otto
bier
brei
salat
brie
```

*Die dazugehörige Ausgabedatei muss somit wie folgt aussehen:*

```
Wort;Anzahl
atlas;2
otto;1
bier;3
```

**Konzeptskizze:**
Machen Sie sich vor der Implementierung Gedanken zu Ihrem Konzept, wie Sie die Aufgabe lösen möchten! Überlegen Sie sich insbesondere wie Sie die Überprüfung von zwei Wörtern auf die Eigenschaft eines Anagramms algorithmisch beschreiben wollen.

## Aufgabe 2: Exception Handling

In dieser Aufgabe schreiben Sie ein kleines Programm, das Benutzereingaben auswertet und bei einer fehlerhaften Eingabe eine eigens definierte Exception wirft. Entwickeln Sie ein Programm, das in einer Endlosschleife Benutzereingaben (`input()`) abfragt. Zur Auswahl stehen die folgenden Zeichen: `?, q, a, s`. Dabei soll folgendes Verhalten implementiert werden:

- `?`: Ausgabe eines Hilfetextes
- `q`: Programm beendet sich
- `a`: Aufruf der Addieren Funktion
- `s`: Aufruf der Subtrahieren Funktion

Wenn der Benutzer ein `q` eingibt, dann muss die selbst definierte Exception `ProgramQuit` ausgelöst und das Programm anschließend beendet werden. Bei der Eingabe eines nicht unterstützten Zeichens soll die Aufforderung zur Eingabe wieder erscheinen. Wenn `a` eingegeben wird, müssen anschließend zwei weiteren Zahlen abgefragt, addiert und das Ergebnis auf der Konsole ausgegeben werden. Bei der Eingabe `s` ist analog zu verfahren, jedoch muss eine Subtraktion angewendet werden.
Sollte bei der Eingabe der Zahlen für die Rechnung (Addition, Subtraktion) kein gültiger Integer eingegeben werden, so ist eine weitere Exception (wählen Sie eine geeignete, bereits vorhandene Exception) auszulösen.

Schreiben Sie das Programm so, dass die Exceptions geworfen und auch abgefangen werden. Die eigens zu definierende Exception kann über die folgende Zeile realisiert werden:

```
class ProgramQuit( Exception ): pass
```

*Hinweis*: Wenn das Einlesen der Benutzerdaten über die `input()` Funktion fehlschlägt kann eine `EOFError` Exception durch das System geworfen werden. Fangen Sie diese Exception ebenfalls ab und lösen Sie darin die eigens definierte Exception `ProgramQuit`.

*Hinweis*: Berücksichtigen Sie, dass die Kommandos in Groß- oder Kleinbuchstaben eingegeben werden können.

Die folgende Sequenz zeigt einen beispielhaften Ablauf des Programms (Benutzereingaben sind in roter Schriftfarbe dargestellt):

```
Eingabe [?,q,a,s]: ?
HILFE TEXT

Eingabe [?,q,a,s]: r

Eingabe [?,q,a,s]: a

Wert 1: 5

Wert 2: 8
Ergebnis: 13

Eingabe [?,q,a,s]: s

Wert 1: 5

Wert 2: asdf
invalid literal for int() with base 10: 'asdf'

Eingabe [?,q,a,s]: q
Programm beendet sich...
```

**Konzeptskizze:**
Machen Sie sich vor der Implementierung Gedanken zu Ihrem Konzept, wie Sie die Aufgabe lösen möchten!