Recursive-Descent-Parser

Sie sollen einen Parser für die Analyse und Auswertung arithmetischer Ausdrücke mit Hilfe der Methode des rekursiven Abstiegs implementieren. Der Parser soll arithmetische Ausdrücke analysieren, die von der folgenden Grammatik generiert werden:

start	→ addexpr
addexpr	→ mulexpr (('+' | '–') mulexpr)*
mulexpr	→ bracketexpr (('*' | '/') bracketexpr)*
bracketexpr	→ '(' addexpr ')' | '−' bracketexpr | number
number	→ integer ('.' integer)?
integer	→ ('0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | '9')+

Die Anführungszeichen markieren hier die Terminalsymbole der Grammatik. Der Operator | steht für die Disjunktion. Die regulären Operatoren ?, *, + stehen für 0 oder 1 (?) bzw. beliebig viele (*) bzw. mindestens 1 (+) Wiederholung des vorhergehenden Ausdruckes. Leerzeichen in den arithmetischen Ausdrücken sollen ignoriert werden.

Der Parser soll aus der Eingabedatei arthmetische Ausdrücke (einer pro Zeile) lesen, parsen und gleichzeitig den Wert berechnen, und dann das Ergebnis im Format *Ausdruck = Wert* ausgeben. Hier ein Beispielaufruf:

```
> cat test.txt
2.7 * -3.5 + 1.9
3*(9-7)/4.0
> python eval.py test.txt
```

Ausgabe (Die Zahl der Nachkommastellen ist egal):

```
2.7 * -3.5 + 1.9 = -7.55
3*(9-7)/4.0 = 1.5
```

Schreiben Sie eine Klasse Parser, welche für jedes Nichtterminal der obigen Grammatik eine Methode besitzt, welche einen Ausdruck dieses Typs erkennt, den Ergebniswert berechnet und zurückgibt. Die Methode für das Nichtterminal auf der linken Seite einer Regel ruft die Methoden für die Nichtterminale auf der rechten Seite der Regel auf. Schreiben Sie außerdem eine Methode **current_char**, welche das aktuelle Zeichen zurückgibt und eine Methode **move_forward**, welche zur nächsten Zeichenposition wechselt. Die Methode **start** erhält den zu analysierenden Ausdruck als Argument.

Wenn ein Ausdruck ungrammatisch ist, geben Sie eine Fehlermeldung in der folgenden Form aus:

```
Fehler: schließende Klammer erwartet: 7 * (3.75 - 1
                                                   ^
```

Das Symbol ^ in der zweiten Zeile zeigt die Fehlerposition an.

Dann machen Sie mit der nächsten Eingabezeile weiter. Für die Fehlerbehandlung verwenden Sie den Exception-Mechanismus. Definieren Sie dazu eine eigene Exception-Klasse mit `def ExprError(Exception):` und fangen Sie die Exceptions im Hauptprogramm ab.

Im Fall einer Division durch 0 sollten Sie ebenfalls eine Exception auslösen.

Verarbeiten Sie die Eingabe beim Parsen Zeichen für Zeichen von links nach rechts. Verwenden Sie ein Dictionary, um die Ziffern 0-9 auf Integerzahlen abzubilden (und nicht Funktionen wie `int` oder `float`).

Vorüberlegungen

- Wie generiert die obige Grammatik den Ausdruck 2.75 * (-3 + 4)?
- Spielen Sie am gleichen Beispiel durch, welche Funktionsaufrufe bei der Verarbeitung des Ausdruckes erfolgen und welcher Teil der Eingabe jeweils verarbeitet wird.
- Schreiben Sie Pseudocode für die Funktion **number**.
- Wie können Sie in der Funktion **bracketexpr** entscheiden, welche der drei Alternativen vorliegt?
- Woran merken Sie bei der Verarbeitung eines Ausdrucks, dass er nicht wohlgeformt ist?
- Wie gehen Sie mit Leerzeichen um?
- Wie berechnen Sie in der Funktion **number** den Wert von Fließkommazahlen?

Implementieren Sie das Programm und testen Sie es systematisch mit verschiedenen Eingaben.