Dokumentation zum Arbeitsblatt 5 EWR

Tom Lambert

03. Juni 2017

Inhaltsverzeichnis

1 Problemstellung und Theoretischer Hintergrund

Bei der Berechnung von Reihen müssten unendlich viele Summanden zusammenaddiert werden. Dies ist praktisch nicht möglich, jedoch lässt sich ein Wert näherungsweise bestimmen, indem man möglichst viele Summanden berechnet und addiert.
Durch jeden weiteren Summanden wird der gesamte Wert genauer.

Außerdem spielt bei der Berechnung von `Decimal`-Werten am Computer die Genauigkeit eine Rolle. Besonders bei irrationalen und periodischen Zahlen kommt es oft zu problemen beim abspeichern der Werte (siehe VL für nähere Informationen). Der `Decimal`-Typ von Python unterstützt verschiedene Genauigkeiten die hier ebenfalls verglichen werden sollen.

2 Beschreibung des Programms

Das Programm untergliedert sich in mehrere Dateien und Klassen.

2.1 `Sum`-Klasse

Eine Reihe ist nichts weiter als eine Summe mit unendlich vielen Summanden. Die Unendlichkeit spielt in der Python-Implementierung keine Rolle, da sowieso nur endlich viele Summanden genutzt werden können. Daher wurde die `Sum`-Klasse implementiert, die für eine Lauf-Variable zwischen 2 Werten einen Summanden berechnet und diese summiert.

2.1.1 Konstruktor

Der Konstruktor nimmt die 3 wichtigen Dinge für die Berechnung entgegen:

- `delegate`
  Eine Funktion zum Berechnen eines Summanden. Diese muss einen ganzzahligen Parameter (`int`) entgegen nehmen der die Laufvariable i darstellt.

- `start_value`
  Dies ist ein optionaler Parameter mit dem Standardwert 1.
  Dies ist der Start-Wert der Summen-Berechnung. Der Laufvariablen i wird dieser wert in der ersten Iteration zugewiesen.

- `end_value`
  Dies ist ein optionaler Parameter mit dem Standardwert 10.
  Dies ist der End-Wert der Summen-Berechnung. Der Laufvariablen i wird dieser wert in der letzten Iteration zugewiesen.

An die Mathematische Notation angelehnt, würde die Berechnung wie folgt aussehen:

$$\sum_{i=start\_value}^{end\_value} \text{delegate}(i) \tag{1}$$

2.1.2 Berechnung (`calculate`-Methode)

Die eigentliche Berechnung findet innerhalb der `calculate`-Methode statt, die keine weiteren Parameter entgegen nimmt. Die Berechnung geschieht über eine `for`-Schleife, die in jedem Durchlauf den im Konstruktor übergebenen Delegaten aufruft. Der so berechnete Summand wird auf eine gemeinsamme „Summen-Variable“ aufgerechnet. Nach allen Schleifendurchläufen wird die so berechnete Summe zurück gegeben.

2.1.3 Caching

Die `Sum`-Klasse hat einen integrierten Caching-Mechanismus. Dadurch werden bereits berechnete Summen in einem Zwischenspeicher abgelegt und können so in der Calculate-Methode wieder verwendet werden.

Hierfür wird nach jeder Berechnung die Summe in einem Wörterbuch abgelegt. Dieses Caching-Wörterbuch benutzt den Start-Wert der Summe als Schlüssel. Jeder Wert ist nun ein weiteres Wörterbuch mit dem End-Wert als Schlüssel und der berechneten Summe als Wert.

Vor der Berechnung einer Summe wird nun geprüft, ob eine andere Summen-Berechnung den selben Start-Wert hatte. Falls ja, wird geprüft ob eine andere Berechnung einen End-Wert nutzte, der kleiner oder gleich dem aktuellen ist. Falls ja, wird der bereits berechnete Wert als Start-Wert genutzt und die Schleife beginnt beim End-Wert der längst durchgeführten Berechnung.
Falls kein passendes, bereits berechnetes Summen-Ergebnis vorliegt, wird einfach vom Start-Wert los gerechnet.

Das Caching ist auf die aktuelle Klassen-Instanz begrenzt, sodass verschiedene Instanzen auch unterschiedliche Caching-Wörterbücher haben können. Das hat den Vorteil, dass verschiedene Klassen-Instanzen auch mit unterschiedlichen Genauigkeiten des `Decimal`-Typs arbeiten können.

2.1.4 Unit-Tests

Für die `Sum`-Klasse sind Unit-Tests ist der Datei *Sum_Tests.py* implementiert. Diese testen alle Fälle mit unterschiedlichen Start- und End-Werten sowie das verwendete Caching.

2.2 `Decimal`-Werte vergleichen

Für das Berechnen mehrerer `Decimal`-Werte und dem Vergleich mit anderen wurde eine eigene Klasse `DecimalComparer` implementiert.

2.2.1 Konstruktor

Dem Konstruktor wird eine Funktion übergeben, welche einen `Decimal`-Wert zurück gibt. Außerdem wird die `precisions`-Klassenvariable initialisiert. Diese Variable enthält eine Liste von Werten die die Genauigkeiten darstellen mit denen der Decimal-Typ nacheinander arbeiten soll.

2.2.2 `run`-Methode

Die `run`-Methode ist das eigentliche Herzstück der Klasse. Diese durchläuft in einer Schleife sämtliche Werte der `precisions`-Variablen und setzt zunächst `Decimal.prec` auf diesen Wert. Im Anschluss wird der dem Konstruktor übergebene Delegat aufgerufen.
Die durch den Delegaten berechneten Werte werden am Ende des Schleifendurchlaufs einem Wörterbuch hinzugefügt. Dieses benutzt die Präzision als Schlüssel und das Ergebnis als Wert. Das Wörterbuch wird am Ende der Methode dem Aufrufer zurück gegeben.

2.3 Hauptprogramm

Das Hauptprogramm ist in *ab4.py* implementiert. Die Datei ist in der Funktionen unterteilt.

2.3.1 `test_k`

Diese Funktion enthält Mechanismen um die Berechnung der Summe mit unterschiedlich hohen End-Werten zu realisieren. Die End-Werte sind dabei stets 10er-Potenzen deren Exponent mit k bezeichnet wird.

Für den Vergleich wird mehrfach die `Sum`-Klasse verwendet, sodass das dort implementierte Caching zum Einsatz kommt.

Dem Benutzer werden die maximale Differenz sowie die Durchschnitte der Ergebnisse angezeigt, sodass dieser die Unterschiede besser einschätzen kann.

2.3.2 `test_prec`

Diese Funktion enthält Mechanismen um die Summe mit dem `Decimal`-Typ mit unterschiedlichen Genauigkeiten zu berechnen. Hierfür wird die `DecimalComparer`-Klasse eingesetzt, die eine Funktion übergeben bekommt die die `Sum`-Klasse zur Berechnung nutzt.

Dem Benutzer werden die maximale Differenz sowie die Durchschnitte der Ergebnisse angezeigt, sodass dieser die Unterschiede besser einschätzen kann.

2.3.3 `main`

Die `main`-Funktion enthält die Haupt-Logik des Programs. Das heißt es wird zunächst ausgegeben was genau geschieht. Anschließend wird der Benutzer gefragt ob er weitere k- oder Präzisions-Werte für die Auswertung hinzufügen möchte. Außerdem, ob er eine benutzerdefinierte Präzision oder einen benutzerdefinierten k-Wert für die jeweils anderen Tests bestimmen möchte. Falls ja, kann er dies jeweils tun. Die Nachfrage an den Benutzer geschieht jedoch nur, wenn `__name__ == "__main__"` ist.

Anschließend werden die `test_k`- und `test_prec`-Funktion aufgerufen.

Die `main`-Funktion wird am Ende der Datei aufgerufen.

2.4 Weitere Dateien

2.4.1 *utils.py*

utils.py enthält allgemeine Hilfsfunktionen die nichts näheres mit der eigentlichen Problemstellung zu tun haben. Die hier enthaltenen Funktionen wurden ausgelagert, da sie so in anderen Programmen leichter wieder verwendet werden können.

Die Funktion `average` wurde implementiert um kein vielfach größeres Package installieren zu müssen nur um einen Durchschnitt berechnen zu können.

2.4.2 *ui.py*

ui.py enthält mehrere Funktionen zum Interagieren mit dem Benutzer. *ui* steht dabei für *User Interface*. Es sind verschiedene Funktionen für das Abfragen von Werten und für die formatierte Ausgabe enthalten. Diese wurden in eine eigene Datei ausgelagert um sie später wiederverwenden zu können.

3 Experimente, Beobachtungen und Auswertung

3.1 Verschiedene Anzahl Summanden

Bei der Berechnung mehrerer Summen mit unterschiedlich vielen Werten hat sich gezeigt, dass die Summe immer größer wird, wenn man mehr Summanden addiert. Die Programm-Ausgabe zeigt das recht deutlich.

```
k = 1  | result = 2.928968253968253968253968254
               + 2.258409263671366292551149422 = ...
k = 2  | result = 5.187377517639620260805117676
               + 2.298093342910724651851400526 = ...
k = 3  | result = 7.485470860550344912656518202
               + 2.302135175494037351521959644 = ...
k = 4  | result = 9.787606036044382264178477846
```

```
Maximum difference between the sums is 6.858637782076128295924509592
The average difference between the results is 2.286212594025376098641503197
```

Verwendet wurden $k = 1, 2, 3, 4$ mit dem End-Wert 10^k sowie einer Präzision von 28.

3.2 Verschiedene Präzisionen

Der Test mit mehreren Präzisionen hat gezeigt, dass sich der Wert auch in weit größeren Stellen ändert, als der letzten. Auch hier zeigt dies die Programmausgabe recht gut:

```
The following values were calculated:
 precision =   1 | result = 2
 precision =   2 | result = 3.8
 precision =   3 | result = 6.16
 precision =   4 | result = 8.446
The average of these results is 5.102.
The maximum difference of these results is 6.446.
```

Hier wurden die Präzisionen $1, 2, 3$ und 4 sowie ein End-Wert von 10^5 verwendet.

3.3 Fazit

Wie zu erwarten war wird der Wert deutlich genauer um so mehr Summanden Addiert werden. Aus den Vorlesungen wissen wir bereits, dass die Summe nicht konvergiert, daher ist es nicht verwunderlich dass der Wert immer größer wird.
Der Vergleich mehrerer Präzisionen zeigt zusätzlich wie wichtig für exakte numerische Lösungen eine Hohe Präzision des verwendeten `Decimal`-Typs ist.