LU10d - Debugger anwenden

Was ist Debugging?

Im Großen und Ganzen ist Debugging der Prozess der Erkennung und Korrektur von Fehlern in einem Programm.

Es gibt verschiedene Arten von Fehlern, mit denen Sie zu tun haben werden. Einige von ihnen sind leicht zu finden, wie Syntaxfehler, da sie von der IDE angezeigt werden. Ein weiterer einfacher Fall ist, wenn der Fehler schnell identifiziert werden kann, indem man sich den Stack-Trace ansieht, der einem hilft herauszufinden, wo der Fehler aufgetreten ist.

Es gibt jedoch Fehler, die sehr knifflig sein können und bei denen es sehr lange dauert, sie zu finden und zu beheben. Ein subtiler Logikfehler, der früh im Programm aufgetreten ist, kann sich zum Beispiel erst sehr spät bemerkbar machen, und manchmal ist es eine echte Herausforderung, die Dinge zu klären.

An dieser Stelle ist der Debugger sehr nützlich. Der Debugger ist ein leistungsfähiges Werkzeug, mit dem sich Fehler viel schneller finden lassen, da er einen Einblick in die internen Abläufe eines Programms gewährt. Dies ist möglich, indem die Ausführung angehalten und der Zustand des Programms durch eine gründliche Untersuchung der Variablen und ihrer zeilenweisen Änderung analysiert wird. Während des Debuggens haben Sie die volle Kontrolle über die Dinge.

Untersuchen Sie den Code

Versuchen wir es mit einem einfachen Debugging-Fall. Stellen Sie sich vor, wir haben die folgende Anwendung:

averagefinder.py

```
def main():
    print('Average finder v0.1');
    numbers = ['1', '2', '3'];

    avg = find_average(numbers);
    print('The average is ' + str(avg));


def find_average(numbers):
    result = 0
    for number in numbers:
        result += int(number)

    return result
```

```
if __name__ == '__main__':
    main()
```

Das Programm soll den Durchschnitt aller Strings in der Arraylist berechnen.

Es lässt sich problemlos kompilieren und ausführen, aber das Ergebnis ist nicht das, was man erwarten würde. Wenn wir zum Beispiel `1 2 3` als Eingabe übergeben, ist das Ergebnis 6.

Zunächst einmal müssen Sie sich überlegen, woher der vermutete Fehler kommen könnte. Wir können davon ausgehen, dass das Problem nicht in den Druckanweisungen liegt. Höchstwahrscheinlich kommen die unerwarteten Ergebnisse von unserer `findAverage`-Methode. Um die Ursache zu finden, sollten wir ihr Verhalten zur Laufzeit untersuchen.

Breakpoints setzen

Um zu untersuchen, wie das Programm zur Laufzeit arbeitet, müssen wir seine Ausführung vor dem verdächtigen Codestück anhalten. Dies geschieht durch das Setzen von Breakpoints. Breakpoints geben die Codezeilen an, an denen das Programm angehalten wird, damit Sie seinen Zustand untersuchen können.

Klicken Sie auf die Zeilennummerierung um einen Breakpoint zu setzen.

Starten Sie das Programm im Debug-Modus

Starten wir nun das Programm im Debug-Modus.

Rechtsklicken Sie dazu auf den Play-Button und wählen Sie **Debug <Applikationsname>**

Analysieren des Programmzustands

Nachdem die Debugger-Sitzung gestartet wurde, läuft das Programm normal, bis ein Haltepunkt

erreicht wird. In diesem Fall wird die Zeile, in der das Programm angehalten wurde, hervorgehoben und das Fenster des Debuggers erscheint.

Die markierte Zeile ist noch nicht ausgeführt worden. Das Programm wartet jetzt auf weitere Anweisungen von Ihnen. Im Schwebezustand können Sie Variablen untersuchen, die den Zustand des Programms enthalten.

Da die findAverage-Methode noch nicht aufgerufen wurde, sind alle ihre lokalen Variablen wie result noch nicht im Gültigkeitsbereich, aber wir können den Inhalt der ArrayList untersuchen. Sie können auch Informationen über alle Variablen abrufen, die sich derzeit im Gültigkeitsbereich befinden, und zwar im Bereich Variablen.

Schritt für Schritt durch das Programm

Jetzt, wo wir mit dem Debug-Tool-Fenster vertraut sind, ist es an der Zeit, in die findAverage-Methode einzusteigen und herauszufinden, was in ihr passiert.

Schritt 1

Um in eine Methode einzusteigen, klicken Sie auf die Schaltfläche `Step Into` oder drücken Sie **F7**.

Schaltfläche „Step into“ im oberen Teil des Debug-Tool-Fensters Eine weitere Zeile wird im Editor hervorgehoben, da wir den Ausführungspunkt um einen Schritt vorverlegt haben.

Schritt 2

Setzen Sie den Schritt mit `Step Over` **F8** fort. Beachten Sie den Unterschied zu Step Into - auch hier wird die Ausführung um einen Schritt nach vorne verlagert, aber es werden die Schritte von aufgerufenen Funktionen wie wie `int()` auf dem Weg dorthin angezeigt.

Gehen wir weiter und sehen wir uns an, wie die lokale Variable `result` deklariert ist und wie sie bei jeder Iteration der Schleife geändert wird.

Das Inline-Debugging hilft uns, Informationen über die Variablenwerte zu erhalten Im Moment enthält die Variable `s` den Wert 3. Sie wird in einen Integer-Wert umgewandelt und zum Ergebnis addiert, das derzeit den Wert `3.0` hat. Bisher keine Fehler. Die Summe wird korrekt berechnet.

Schritt 3

Zwei weitere Schritte führen uns zur Return-Anweisung und wir sehen, wo die Fehlerquelle liegt. Wir haben vergessen, die Summe durch die Anzahl der Werte zu dividieren. Dies war die Ursache für die fehlerhafte Rückgabe der Methode.

Der Wert des Ergebnisses wird so zurückgegeben, wie er ist, ohne ihn durch die Anzahl der Argumente zu dividieren.

Schritt 4

Lassen Sie uns den Fehler korrigieren.

```
return result / len(numbers)
```

Beenden Sie die Debugger-Sitzung und führen Sie das Programm erneut aus

Um zu prüfen, ob das Programm einwandfrei funktioniert, beenden wir die Debugger-Sitzung und führen das Programm erneut aus. Nun sollte das korrekte Ergebnis `The average is 2.0` angezeigt werden.

---

m319-E3G,, m319-E3F,, m319-E3E

© Kevin Maurizi, Marcel Suter

From:
https://wiki.bzz.ch/ - **BZZ - Modulwiki**

Permanent link:
https://wiki.bzz.ch/modul/archiv/m319python/learningunits/lu10/debugger?rev=1711631267

Last update: **2024/03/28 14:07**