Objekt-orientierte Programmierung 2

Proseminar Python

Wladislaw Tarassov

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

05.07.2022

Agenda

1. Spezielle Methoden
   - Was sind dunder methods?
   - Warum werden dunder methods nicht direkt aufgerufen?
   - Wofür werden sie gebraucht?
   - Beispiel dunder method
   - dunder methods Überblick
2. Operator Overloading
   - Operator Overloading und wofür wird es benötigt?
   - Operator Overloading Beispiel
3. Context Manager
   - Was sind Context Manager?
   - Context Manager Anwendungsbeispiel
4. Zusammenfassung

Namen

- Spezielle Methoden/special methods
- Magische Methoden/magic methods
- double underscore methods
- dunder methods

Syntax

- __name__
- Zwei Unterstriche vor und nach Methodenname
- Ausprache einer dunder Method (__init__):
- Falsch: Unterstrich Unterstrich init Unterstrich Unterstrich
- Richtig: dunder init

Was sind dunder methods?

- Methoden, die Python kennt
- Magisch, da sie implizit im Hintergrund von Python aufgerufen werden

```
1 a + b
2 a.__add__(b)
```

Listing 1: Was passiert im Hintergrund?

Warum werden dunder methods nicht direkt aufgerufen?

- Direkter Aufruf von dunder methods ist nicht gedacht, sondern Aufruf von high-level Operationen
- Beispiel für high-level Operationen: +, -, == usw.

```
1 a = 4
2 b = 4.0
3
4 print(a.__eq__(b))
```

Listing 2: Input

```
1 NotImplemented
```

Listing 3: Output

Warum werden sie nicht direkt aufgerufen?

```
1 a = 4
2 b = 4.0
3
4 print(b.__eq__(a))
```

Listing 4: Input

```
1 True
```

Listing 5: Output

Warum werden sie nicht direkt aufgerufen?

```
1 a = 4
2 b = 4.0
3
4 print(a == b)
```

Listing 6: Input

```
1 True
```

Listing 7: Output

Wofür werden sie gebraucht?

- Damit eingebaute Funktionen/built-in functions funktionieren
- Zum Anpassen von eingebauten Funktionen

```
1 z = [3, 4, 2, 5, 2]
2
3 print(z.__len__())
4 print(len(z))
```

Listing 8: Input

```
1 5
2 5
```

Listing 9: Output

- Operatoren überladen
- Dienen als Vertrag zwischen Python Interpreter und Nutzer

Beispiel Code für eine dunder method

```
1  class Person:
2      def __init__(self, name):
3          self.name = name
4
5      def hallo(self):
6          print('Hallo, mein Name ist', self.name)
7
8
9  p = Person('Marvin')
10 p.hallo()
```

Listing 10: Input

```
1  Hallo, mein Name ist Marvin
```

Listing 11: Output

Beispiel Code für eine dunder method in Java

```
1  public class Person
2  {
3  String sname;
4  public Person(String name)
5  {
6      sname = name;
7  }
8  public void hallo()
9  {
10     System.out.println("Hallo, mein Name ist " +
           ↪ sname);
11 }
12 }
```

Listing 12: Input 1

Beispiel Code für eine dunder method in Java

```
1 public static void main(String []args)
2 {
3     Person p = new Person("Marvin");
4     p.hallo();
5 }
```

Listing 13: Input 2

```
1 Hallo, mein Name ist Marvin
```

Listing 14: Output

Dunder methods: Grundlegende Anpassungen

- ___new___(self)
  - Erstellt ein neues Objekt (eine Instanz der Klasse)
- ___init___(self)
  - Initialisierung eines Objektes
  - Konstruktor
- ___del___(self)
  - del()
  - Wird aufgerufen, wenn ein Objekt gelöscht werden soll
- ___repr___(self)
  - repr()
  - Gibt einen String zurück
  - Für Entwickler gedacht

Dunder methods: Grundlegende Anpassungen 2

- __str__(self)
  - str()
  - Gibt einen String zurück
  - Für Nutzer gedacht
- __bytes__(self)
  - bytes()
  - Gibt ein Byte Objekt (Byte String) zurück
- __format__(self)
  - format()
  - Wertet formatierte Zeichenfolgenliterale aus
- __lt__(self, anotherObj)
  - Für $<$ Operator
- __le__(self, anotherObj)
  - Für $<=$ Operator

Dunder methods: Grundlegende Anpassungen 3

- __eq__(self, anotherObj)
  - Für == Operator
- __ne__(self, anotherObj)
  - Für != Operator
- __gt__(self, anotherObj)
  - Für > Operator
- __ge__(self, anotherObj)
  - Für >= Operator

Dunder methods: Arithmetische Operatoren

- __add__(self, anotherObj)
  - Für + Operator
- __sub__(self, anotherObj)
  - Für - Operator
- __mul__(self, anotherObj)
  - Für * Operator
- __matmul__(self, anotherObj)
  - Für @ Operator (Numpy-Matrix-Multiplikation)
- __truediv__(self, anotherObj)
  - Einfache / Division.
- __floordiv__(self, anotherObj)
  - // floor division

Operator Overloading und wofür wird es benötigt?

- Überladen von Operatoren
- Gleiche Methodennamen, unterschiedliches Verhalten
- Verhalten von Operatoren nach Wunsch anpassen
- Man kann alle Operatoren überladen aber keine neuen kreieren
- Chaining

```
1 a + b + c
```

Listing 15: Chaining

Operator Overloading Beispiel

```
1  class Point:
2      def __init__(self, x=0, y=0):
3          self.x = x
4          self.y = y
5      def __str__(self):
6          return "({0},{1})".format(self.x, self.y)
7      def __add__(self, other):
8          x = self.x + other.x
9          y = self.y + other.y
10         return Point(x, y)
11 p1 = Point(2, 3)
12 p2 = Point(4, 5)
13 print(p1+p2)
```

Listing 16: Input

Operator Overloading Beispiel

```
1 (6,8)
```

Listing 17: Output

Wofür werden Context Manager benötigt?

```
1 file = open('./text.txt', 'w')
2 file.write('hallo')
3 file.close()
```

Listing 18: leak

Problem gelöst

```
1 try:
2     file = open('./text.txt', 'w')
3     file.write('hallo')
4 except IOError:
5     print('Exception')
6 finally:
7     file.close()
```

Listing 19: Exception Handling

Context Manager

- Paarweise Ausführung von verwandten Operationen mit Code dazwischen (__enter__ und __exit__)
- Codereduzierung
- Exception Handling
- Datenverlust verhindern
- Zuweisung und Freigabe von Ressourcen und Schließung von Dateien

Context Manager Beispiel Klasse

```
1  class File:
2
3      def __init__(self, file_name, method):
4          self.file_obj = open(file_name, method)
5
6      def __enter__(self):
7          return self.file_obj
8
9      def __exit__(self, type, value, traceback):
10         self.file_obj.close()
```

Listing 20: Klasse

Context Manager Beispiel

- with speichert __exit__ Methode in der file Klasse
- Die __enter__ Methode wird gerufen
- Die __enter__ Methode öffnet die Datei und gibt sie zurück
- Die offene Datei wird an file übertragen
- Die Datei wird mit write beschrieben
- Die __exit__ Methode wird gerufen
- Die __exit__ Methode schließt die Datei

```
1 with open("./text.txt", "w") as file:
2     file.write("hallo")
```

Listing 21: with Statement

Zusammenfassung

- Methoden die Python bekannt sind, werden dunder methods genannt und werden im Hintergrund aufgerufen
- ohne dunder methods funktionieren eingebaute Funktionen nicht
- Operatoren können angepasst werden aber nicht neu erstellt werden
- Um Datenverlust oder Exceptions zu behandeln kann man Context Manager benutzen

Literatur

1. https://docs.python.org/3/reference/datamodel.html#special-method-names
2. https://www.python-kurs.eu/python3_magische_methoden.php
3. https://www.geeksforgeeks.org/___init___-in-python/
4. https://www.programiz.com/python-programming/operator-overloading
5. https://docs.python.org/3/reference/datamodel.html#special-method-names
6. https://rszalski.github.io/magicmethods/#conclusion

Literatur 2

7 https://www.pythonmorsels.com/what-are-dunder-methods/

8 https://www.educative.io/answers/how-to-overload-an-operator-in-python

9 https://lernenpython.com/operator-overloading/Python_Operator_Overloading

10 https://www.pythontutorial.net/python-oop/python-operator-overloading/

11 https://realpython.com/python-with-statement/managing-resources-in-python

12 https://book.pythontips.com/en/latest/context_managers.html