## Informatik I

PD Dr. J.-G. Smaus
G. Röger, R. Mattmüller
Wintersemester 2010/2011

Universität Freiburg
Institut für Informatik

# Übungsblatt 6

**Abgabe: 27. Januar 2011**

**Aufgabe 6.1** (Identität, 2 Punkte, auf Papier)
Betrachten Sie den folgenden Code, der eine fiktive Klasse `Foo` verwendet:

```
a = Foo()
b = Foo()
c = a
print(a is b, a is c, b is c) # Ausgabe 1
c = b
print(a is b, a is c, b is c) # Ausgabe 2
b = Foo()
print(a is b, a is c, b is c) # Ausgabe 3
a = c
b = a
print(a is b, a is c, b is c) # Ausgabe 4
```

Was wird bei Ausgabe 1-4 jeweils ausgegeben? Warum?

**Aufgabe 6.2** (Veränderlichkeit von Objekten, 1+1 Punkte, Abg.: `cookies.py`)
Betrachten Sie die Klasse `ChocolateCookie` aus der Vorlesung:

```
class ChocolateCookie:
    def __init__(self, chocolate, cookie):
        self.chocolate = chocolate
        self.cookie = cookie

    def prettyprint(self):
        print("ChocolateCookie", self.chocolate, self.cookie)
```

(a) Schreiben Sie eine Methode `new_scaled(self, factor)` die einen neuen ChocolateCookie zurückgibt, der `factor`-mal so groß ist (also mit der `factor`-fachen Schokoladen- und Keksmenge).

(b) Schreiben Sie eine Methode `scale(self, factor)`, die den Keks selbst `factor`-mal so groß macht.

**Aufgabe 6.3** (Veränderlichkeit und Gleichheit, 1+1+2 Punkte, Abg.: `cars.py`)

(a) Schreiben Sie eine Klasse `Car` mit den Attributen `manufacturer`, `model`, `color`, die alle in der `__init__`-Methode (als Strings) gesetzt werden.

Schreiben Sie zudem eine Methode `prettyprint`, die zum Beispiel für ein Auto `Car("Audi", "RS 6", "dark blue")` die Ausgabe

```
Audi RS 6 in dark blue
```

erzeugt.

(b) Ergänzen Sie Ihre Klasse um eine Methode `repaint(self, color)`, die dem Auto eine neue Farbe gibt (also kein neues Auto erstellt, sondern das bestehende verändert).

(c) Sie erstellen nun zwei Autos `a = Car("Skoda", "Fabia", "black")` und `b = Car("Skoda", "Fabia", "black")` und vergleichen sie mit `a == b`. Sie sind nicht überrascht, dass dieser Vergleich `False` zurückliefert (weshalb?), das ist aber nicht, was Sie eigentlich wollen.

Implementieren Sie daher eine Methode, die dafür sorgt, dass zwei gleichfarbige Autos, die das gleiche Modell vom gleichen Hersteller sind, vom Operator `==` als gleich angesehen werden.

*Hinweis:* Bedenken Sie, dass mit dem Vergleichsoperator beliebige Objekte verglichen werden können müssen. Was passiert zum Beispiel, wenn Sie mit Ihrer Implementierung ein Auto mit einer Zahl vergleichen? (Die Funktion `type` kann hier von Nutzen sein.)

**Aufgabe 6.4** (`math`-Modul, 1+1+1+1+2+2+2+2 Punkte, Abg.: `coords.py`)

(a) Schreiben Sie eine Klasse `CartesianCoordinate`, die eine zweidimensionale Koordinate im kartesischem Koordinatensystem repräsentiert. Das heißt, die Koordinate besteht aus einer $x$- und einer $y$-Komponente. Schreiben Sie die Klasse so, dass zum Beispiel `CartesianCoordinate(3, 4)` eine Koordinate mit $x$-Wert 3 und $y$-Wert 4 erstellt.

Schreiben Sie zudem eine Methode `prettyprint`, die die Koordinate in der Form `(3, 4)` (oder alternativ `( 3 , 4 )`) ausgibt.

(b) Ergänzen Sie Ihre Klasse um eine Methode

`distance(self, cartesian_coordinate)`,

die den Abstand zwischen der Koordinate und einer anderen Koordinate `cartesian_coordinate` berechnet. Sie benötigen hierzu das Modul `math`, das Teil der Python-Standardbibliothek ist und Ihnen verschiedene mathematische Funktionen und Konstanten bereits fertig zur Verfügung stellt. Sie machen das Modul verfügbar, indem Sie `import math` (ganz oben) in Ihrer Datei einfügen. Dann können Sie mit `math.funktionsname` auf die benötigte Funktion zugreiffen. Zum Beispiel berechnet `math.sqrt(4)` die Quadratwurzel aus 4 und `math.pow(3, 4)` den Wert $3^4$.

*Hinweis:* Werfen Sie vielleicht auch einmal einen Blick auf die Dokumentation des `math`-Moduls, um sich mit der Notation der Sprachreferenz vertraut zu machen. Sie finden die Dokumentation der Standardbibliothek unter `http://docs.python.org/py3k/library/`. Alternativ zu `math.pow`

und `math.sqrt` können Sie auch den eingebauten Operator `**` verwenden: `x**y` berechnet $x^y$.

(c) Fügen Sie Ihrer Klasse eine Methode

```
add_cartesian(self, cartesian_coordinate)
```

hinzu, die eine *neue* Koordinate zurückgibt, die der Summe des Objekts selbst und einer anderen kartesischen Koordinate `cartesian_coordinate` entspricht. Verändern Sie dabei nicht die addierten Koordinaten.

*Hinweis*: Die Summe zweier Koordinaten $(x, y)$ und $(x', y')$ ist definiert als $(x, y) + (x', y') = (x + x', y + y')$.

(d) Schreiben Sie eine Klasse `PolarCoordinate`, die eine Polarkoordinate repräsentiert. Das heißt, die Koordinate wird durch den Winkel `phi` (im Gradmaß) und den Abstand `r` beschrieben, die die Position der Koordinate relativ zum Ursprung festlegen. Schreiben Sie die Klasse so, dass man zum Beispiel mit `PolarCoordinate(78, 3)` eine Koordinate mit Winkel 78° und Abstand 3 erstellt.

Schreiben Sie zudem wieder eine Methode `prettyprint`, die die Koordinate in der Form `(79 Grad, 3)` (oder alternativ `( 79 Grad, 3 )`) ausgibt.

(e) Erweitern Sie Klasse `PolarCoordinate` um eine Methode `as_cartesian`, die die Koordinate im kartesischen System zurückgibt. Die $x$- und $y$-Koordinate errechnet sich dabei folgendermaßen:

$$x = r \cdot \cos(\phi)$$
$$y = r \cdot \sin(\phi)$$

Sie können hierzu die Funktionen `cos` und `sin` des `math`-Moduls verwenden. Beachten Sie bitte, dass diese für Winkel im Bogenmaß definiert sind. Sie können mit den Funktionen `radians(winkelingrad)` und `degrees(winkelimbogenmaß)` des selben Moduls Winkel zwischen den beiden Repräsentationen konvertieren.

(f) Erweitern Sie umgekehrt ihre Klasse `CartesianCoordinate` um eine Methode `as_polar`, die die Koordinate im Polarsystem zurückgibt. Den Abstands zum Nullpunkt können Sie mit der `distance`-Methode bestimmen. Die Berechnung des Winkels ist relativ kompliziert, wie Sie im Wikipedia-Artikel zu Polarkoordinaten nachlesen können. Zum Glück gibt es im `math`-Modul die Funktion `atan2`, die die komplexen Berechnungen und Fallunterscheidungen übernimmt.

*Hinweis:* Schlagen Sie die `atan2`-Funktion in der Standardbibliothek nach. Beachten Sie, dass der berechnete Winkel wieder im Bogenmaß angegeben ist.

(g) Erweitern Sie nun die Klasse `PolarCoordinate` um eine Methode

`add_polar(self, polar_coordinate)`,

die eine *neue* Koordinate zurückgibt, die der Summe des Objekts selbst und einer anderen Polarkoordinate `polar_coordinate` entspricht. Modifizieren Sie dabei nicht die addierten Koordinaten.

(h) Erweitern Sie zuletzt beide Klassen um eine Methode `add(self, other)`, die die Summe von `self` und `other` als neues Objekt (der gleichen Klasse wie `self`) zurückgibt. Dabei soll es egal sein, ob `other` als Polarkoordinate oder als kartesische Koordinate repräsentiert ist.

*Hinweis:* Verwenden Sie (ausnahmsweise) `isinstance`, um die Klasse von `other` zu überprüfen.

**Hinweis zum Übungsblatt:** Um für die Programmieraufgaben Punkte zu erhalten, müssen für jede Funktion und Methode (außer `__init__` und `prettyprint`) je mindestens zwei Testfälle angegeben sein. Verwenden Sie hierbei die Konvention aus dem betreuten Programmieren:

- Eine *Methode* `my_method` einer Klasse `MyClass` testen Sie, indem Sie außerhalb der Klasse die Testfunktion `test_MyClass_my_method` definieren. Eventuell benötigte Instanziierungen von Klassen stehen am Anfang des Rumpfes der Testfunktion. Beispiel: angenommen, Sie wollen die Methode `multiply_with` der folgenden Klasse `Multiplier` testen.

```
class Multiplier:
    def __init__(self, factor):
        self.factor = factor
    def multiply_with(self, number):
        return self.factor * number
```

Dann könnte ein Test so aussehen:

```
def test_Multiplier_multiply_with():
    m1 = Multiplier(3)
    m2 = Multiplier(-2)
    assert m1.multiply_with(4) == 12
    assert m2.multiply_with(3) == -6
```

- Alle Tests für eine Funktion `my_function` stehen in einer neuen parameterlosen Funktion mit dem Namen `test_my_function`.

Die Übungsblätter müssen individuell bearbeitet werden. Gruppenabgaben sind nicht zulässig.