Danny Rehl, Programmieren I - Python

Übungsblatt 09

Abgabefrist: Montag, den 13.01.2025, um 23:59 Uhr

Sie finden für dieses Übungsblatt eine Vorlage für Ihre Abgabe zum Herunterladen im Moodle.

Bitte achten Sie darauf, dass Ihr Code PEP 8 konform geschrieben ist. Das ist Voraussetzung für die abschließende Bewertung. Nutzen Sie die Hinweise des Pylint-Scorers, um die verbleibenden Style-Fehler zu finden.

Aufgabe 1 (3 Punkte)

Folgender Code beschreibt zwei Beispiele `pizza_margherita` und `pizza_hawaii` einer Pizza-Datenstruktur, basierend auf einem `dict` Dictionary:

```python
if __name__ == "__main__":
    pizza_margherita = {
        "name": "Margherita",
        "dimensions": 28,
    }
    pizza_hawaii = {
        "name": "Hawaii",
        "dimensions": (30, 40),
    }
    pizza_orders = [pizza_margherita, pizza_hawaii]
    for i, pizza in enumerate(pizza_orders, start=1):
        print("Order", i, "is a Pizza", pizza["name"])
        if isinstance(pizza["dimensions"], int):
            pizza_shape = "circle"
        else:
            pizza_shape = "rectangle"
        print("with a", pizza_shape, "shape")
        if pizza_shape == "circle":
            pi = 3.1415
            pizza_area = pi * (pizza["dimensions"] / 2) ** 2
        if pizza_shape == "rectangle":
            pizza_area = pizza["dimensions"][0] * pizza["dimensions"][1]
        print("and an area of", pizza_area, "cm²")
```

Die Datenstruktur enthält zwei Keys: `"name"` mit der Bezeichnung der Pizza, sowie `"dimensions"`, einer Angabe zur Pizza-Größe in Zentimeter (cm). Anhand `"dimensions"` lässt sich außerdem direkt erkennen, ob es sich um eine kreisförmige Pizza (nur ein `int` Wert, der Durchmesser) oder um eine Pizza in Form eines Rechtecks (Tupel mit zwei `int` Werten für die beiden Seitenlängen) handeln soll.

Die erwartete Ausgabe ist:

```
Order 1 is a Pizza Margherita
with a circle shape
and an area of 615.734 cm²
Order 2 is a Pizza Hawaii
with a rectangle shape
and an area of 1200 cm²
```

Das funktioniert so also zwar alles, aber langfristig soll der Code noch viel mehr können, wird also noch um viele weitere Zeilen anwachsen. Damit der Code am Ende immer noch gut lesbar ist, soll er jetzt schon besser strukturiert werden.

Wir wollen zur besseren Strukturierung des Code bestimmte Logikschritte in den "Hintergrund" verschieben, sodass sie nicht mehr den Code auf Modullevel "verstopfen": Sowohl die Unterscheidung bezüglich der Pizza-Form (also ob Kreis oder Rechteck), als auch die Entscheidung über die entsprechend zu verwendende Formel zur Berechnung der Fläche müssen nicht zwingend auf Modullevel stattfinden. Eine übersichtlichere Lösung ist die Einführung zweier Funktionen `get_pizza_shape` und `get_pizza_area`:

```
def get_pizza_shape(pizza):
    if isinstance(pizza["dimensions"], int):
        pizza_shape = "circle"
    else:
        pizza_shape = "rectangle"
    return pizza_shape

def get_pizza_area(pizza):
    pizza_shape = get_pizza_shape(pizza)
    if pizza_shape == "circle":
        pi = 3.1415
        pizza_area = pi * (pizza["dimensions"] / 2) ** 2
    if pizza_shape == "rectangle":
        pizza_area = pizza["dimensions"][0] * pizza["dimensions"][1]
    return pizza_area
```

Der Code auf Modullevel sieht damit direkt übersichtlicher aus:

```
if __name__ == "__main__":
    pizza_margherita = {
        "name": "Margherita",
        "dimensions": 28,
    }
    pizza_hawaii = {
        "name": "Hawaii",
        "dimensions": (30, 40),
    }
    pizza_orders = [pizza_margherita, pizza_hawaii]
    for i, pizza in enumerate(pizza_orders, start=1):
        print("Order", i, "is a Pizza", pizza["name"])
        print("with a", get_pizza_shape(pizza), "shape")
        print("and an area of", get_pizza_area(pizza), "cm²")
```

Unsere gewählten Funktionsnamen `get_pizza_shape` und `get_pizza_area` sind intuitiv deskriptiv. Der Code liest sich fast wie Prosa. Wer sich beim Lesen des Code für die genaue Logik innerhalb der Funktionen interessiert, der kann immer noch einfach nach oben scrollen und dort die jeweilige Implementation einsehen.

Die von uns hinzugefügten Funktionen heben den Code auf Modullevel also auf ein höheres Abstraktionsniveau. Nach demselben Prinzip haben Sie in der Vorlesung eine weitere Form der Codeabstraktion kennengelernt: Object-oriented programming (OOP).

Wir möchten für diese Aufgabe also die beiden Funktionen in Methoden einer Klasse `Pizza` überführen. Auf dem Modullevel soll die Klasse dann mit folgendem Code funktionieren:

```
if __name__ == "__main__":
    pizza_margherita = Pizza()
    pizza_margherita.name = "Margherita"
    pizza_margherita.dimensions = 28
    pizza_hawaii = Pizza()
    pizza_hawaii.name = "Hawaii"
    pizza_hawaii.dimensions = (30, 40)
    pizza_orders = [pizza_margherita, pizza_hawaii]
    for i, pizza in enumerate(pizza_orders, start=1):
        print("Order", i, "is a Pizza", pizza.name)
        print("with a", pizza.shape(), "shape")
        print("and an area of", pizza.area(), "cm²")
```

Wir können für die Klasse viel Code aus `get_pizza_shape` kopieren und für die Methode `Pizza.shape` wiederverwerten. Analog wird die Funktion `get_pizza_area` zur Methode `Pizza.area`. Sie finden den "alten" Code der zu ersetzenden Funktionen auskommentiert in der Vorlage zu diesem Übungsblatt. Außerdem finden Sie eine erste Version der `Pizza`-Klasse. Die Methode `Pizza.shape` wurde bereits erfolgreich implementiert:

```
class Pizza:
    def shape(self):
        if isinstance(self.dimensions, int):
            pizza_shape = "circle"
        else:
            pizza_shape = "rectangle"
        return pizza_shape
    def area(self):
        # TODO
        pass
```

Ihre Aufgabe ist es, die Klasse zu vervollständigen und die noch fehlende Methode `Pizza.area` zu implementieren. Machen Sie sich klar, dass sich bestimmte Referenzen ändern:

- In den beiden Funktion wurde die Datenstruktur jeweils als Argument `pizza` übergeben - in den Methoden der Klasse ist die Instanz jedoch unter dem Namen `self` referenziert.
- Die Größenangabe zur Pizza ist kein Dictionary-Key mehr, wird also nicht mehr über `pizza["dimensions"]` indiziert, sondern ist ein Attribut der Instanz `self.dimensions`.
- Da die Funktion nicht mehr existiert, ergibt der Aufruf `get_pizza_shape(pizza)` keinen Sinn mehr, sondern stattdessen muss die eigene Methode mit `self.shape()` aufgerufen werden.

Aufgabe 2 (2 Punkte)

Fügen Sie eine Methode `Pizza.__init__` zur Klasse hinzu, sodass die Attribute `Pizza.name` und `Pizza.dimensions` direkt initialisiert werden können:

```
>>> pizza_margherita = Pizza("Margherita", 28)
>>> pizza_margherita.name
'Margherita'
>>> pizza_margherita.dimensions
28
>>>
```

Unser Beispiel-Code zur vorherigen Aufgabe könnte nun also so aussehen:

```
if __name__ == "__main__":
    pizza_margherita = Pizza("Margherita", 28)
    pizza_hawaii = Pizza("Hawaii", (30, 40))
    pizza_orders = [pizza_margherita, pizza_hawaii]
    for i, pizza in enumerate(pizza_orders, start=1):
        print("Order", i, "is a Pizza", pizza.name)
        print("with a", pizza.shape(), "shape")
        print("and an area of", pizza.area(), "cm²")
```

Aufgabe 3 (3 Punkte)

Eine Pizza soll mit verschiedenen Toppings belegt werden können. Definieren Sie dafür eine neue Klasse `Topping`. Die Klasse soll eine neue Basisklasse (base class, top-level class) sein - sie selbst erbt also von keiner anderen Klasse (ausgenommen der expliziten oder impliziten fundamentalen Vererbung durch `object`, wovon generell automatisch *alle* Python new-style classes erben). Instanzen von `Topping` sollen bei der Initialisierung zwei Argumente annehmen: `name` und `price`, welche dann als Instanzvariablen verfügbar sind:

```
>>> extra_cheese = Topping("Extra Cheese", 0.5)
>>> extra_cheese.name
'Extra Cheese'
>>> extra_cheese.price
0.5
>>>
```

Initialisieren Sie anschließend in `Pizza.__init__` eine versteckte ("private") Instanzvariable `Pizza.__toppings` mit einer leeren Liste:

```
self.__toppings = []
```

Implementieren Sie zudem eine neue Methode `Pizza.add_topping`, welche Toppings an die `__toppings` Liste anhängt:

```
def add_topping(self, topping):
    self.__toppings.append(topping)
```

Damit nicht jede beliebige Datenstruktur als `topping` übergeben werden kann, sollen Sie zudem die `Pizza.add_topping`-Methode derart modifizieren, sodass inkompatible Werte für das `topping`-Argument nicht an `__toppings` angehängt werden, sondern zu einem `TypeError` führen.

Sie können dafür die Built-in Function `isinstance` verwenden, um zu prüfen, ob das Argument `topping` eine Instanz Ihrer benutzerdefinierten Klasse `Topping` ist.

Auf der nächsten Seite finden Sie Beispiele für mögliche Interaktionen zwischen `Topping` und `Pizza.add_topping`.

Toppings werden zunächst erstellt, und dann zu einer Pizza hinzugefügt:

```
>>> pizza_hawaii = Pizza("Hawaii", 28)
>>> extra_cheese = Topping("Extra Cheese", 0.5)
>>> pineapple = Topping("Pineapple", 1)
>>> pizza_hawaii.add_topping(extra_cheese)
>>> pizza_hawaii.add_topping(pineapple)
>>>
```

Es können jedoch nur tatsächliche Toppings, also Instanzen von `Topping` hinzugefügt werden. Beispielsweise sind reine `int` oder `str` Datenstrukturen nicht erlaubt und führen zu einem `TypeError`:

```
>>> pizza_margherita = Pizza("Margherita", 10)
>>> not_a_topping = 1234
>>> pizza_margherita.add_topping(not_a_topping)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 22, in add_topping
TypeError: Bad topping, must be type 'Topping', not 'int'
>>> also_not_a_topping = "something"
>>> pizza_margherita.add_topping(also_not_a_topping)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "<stdin>", line 22, in add_topping
TypeError: Bad topping, must be type 'Topping', not 'str'
>>> tasty_topping = Topping("Random tasty topping", 2)
>>> pizza_margherita.add_topping(tasty_topping)
>>>
```

Aufgabe 4 (2 Punkte)

Der Preis einer Pizza aus unserer virtuellen Pizza-Bäckerei wird über deren Fläche sowie der Summe der absoluten Preise der gewählten Toppings in `Pizza.__toppings` berechnet.

Der Grundpreis soll 0,5 Cent pro 1 cm² Fläche betragen. (Erinnerung: `Pizza.area` bestimmt die Fläche der Pizza in genau dieser Einheit cm².) Der Preis der einzelnen Toppings ist in Euro im Attribut `Topping.price` hinterlegt.

Implementieren Sie eine Methode `Pizza.price`, welche den Preis in Euro als `float` zurückgibt.

Hier eine Beispielrechnung:

```
>>> custom_pizza = Pizza("Custom", (10, 20))
>>> custom_pizza.price()
1.0
>>> a_few_olives = Topping("Olives", 0.75)
>>> some_artichokes = Topping("Artichokes", 1.5)
>>> custom_pizza.add_topping(a_few_olives)
>>> custom_pizza.add_topping(some_artichokes)
>>> custom_pizza.price()
3.25
>>>
```

Die Pizza hat eine Fläche von 10 cm × 20 cm = 200 cm². Das entspricht also einem Grundpreis von 200 × 0,5 Cent = 100 Cent = 1 Euro. Dazu kommen noch die Preise für die beiden Toppings, also insgesamt 1 Euro + 0,75 Euro + 1,5 Euro = 3,25 Euro.