Globale Verständnisfragen

Zum Abschluss dieses Lernmoduls wollen wir noch einmal die wichtigsten Erkenntnisse und Informationen, die Sie aus diesem Lernmodul mitnehmen sollen, mit einigen zusammenfassenden Fragen verschiedenen Typs abfragen und wiederholen. Sie können diese Fragen als "inverted classroom" verstehen. Die Idee ist, dass Sie sich hier überlegen können, ob Sie die gestellten Fragen selbst beantworten können. Falls es noch Unklarheiten gibt, können Sie die entsprechenden Antworten nochmals direkt in den passenden Abschnitten des Lernmoduls rekapitulieren. Dementsprechend sind die Antworten auch absichtlich skizzenhaft. Bitte zögern Sie nicht, bei konkreten Fragen einen der Kanäle zur Hilfe zur Selbsthilfe zu nutzen.

Vokabeltest

Was ist eine Klasse?

Klassen sind ein zentrales Prinzip in der objektorientierten Programmierung. Sie können weitestgehend synonym, wenigstens auf dem Level dieses Programmierkurses, verstanden werden als Konstruktionspläne, Vorlagen, abstrakte Definitionen, oder Rahmenbeschreibungen von modellierten Zusammenhängen. Klassen spezifizieren im Wesentlichen zwei Dinge:

"Platzhalter" für konkrete Daten: Hierbei sind die Daten zumindest konzeptionell "ähnlich". In der OOP spricht man von Eigenschaften oder Attributen.

"Operationen" die die Daten manipulieren: Hierbei müssen die Operationen unabhängig von den konkreten Werten der Daten funktionieren; um dies sicherzustellen, fordern wir die schwammige "Ähnlichkeit" der Daten. Beispiele solcher Operationen sind elementare Dinge wie die Definition der Daten, ihr Auslesen, oder auch fortgeschrittene Dinge wie die Kombination von Daten. In der OOP spricht man von Funktionen oder Methoden.

Wichtig ist, dass Python eine objektorientierte Sprache ist, auch wenn wir im Kurs dies nur wenig genutzt haben. Selbst profane Dinge wie der Datentyp float sind Objekte.

Was ist ein Objekt?

Ein Objekt ist eine konkrete Instanz einer Klasse, bei der die Daten mit konkreten Werten gefüllt sind. Beispielsweise ist die Klasse float ein Platzhalter für irgendeine Gleitkommazahl, und das Objekt, das wir mit der Anweisung a = 1.2345 instantiieren, hat einen konkreten Wert. Wichtig ist, dass die Methoden der Klasse float in diesem Objekt zur Verfügung stehen, beispielswiese die Methode zur Generierung einer String-Repräsentation für die Bildschirmausgabe.

Was ist eine Attribut?

Ein Attribut ist in einer Klasse ein Platzhalter für ein konkretes Datum eines festgeschrieben Typs. In einem Objekt, also in einer Instanz der Klasse, ist ein Attribut eine Referenz auf konkrete Daten.

Syntaxfragen

Wie definieren man eine Klasse?

Diese Frage wurde ausführlich hier beantwortet. In Kurzform lautet die Antwort, dass wir mit der Zeile class name: eine Klasse definieren, hierbei ist name der frei wählbare Name und der Doppelpunkt ist obligatorisch. Es folgt ein Block, der über die übliche Einrückungsregel spezifiziert wird. In diesem Block werden Methoden und Attribute festgelegt.

Wie definiert man eine Klasse mit rein privaten Attributen?

Zur Erinnerung: Private Attribute sind nur innerhalb von Objekten einer Klasse sichtbar und manipulierbar, und eben gerade nicht von außerhalb. Dies stellt die Kapselung sicher, die ein zentrales Prinzip der objektorientierten Programmierung bzw. objektorientierten Modellierung.

Um Attribute privat zu machen, müssen die Namen der Attribute mit doppelten oder einfachen Unterstrichen begonnen werden. Den Unterschied erläutern wir weiter unten.

Wie kann man sicherstellen, dass ein Objekt einer selbst definierten Klasse in print() gestopft werden kann?

Hierzu muss eine Klasse die Methode __str__(self) bereitstellen. Sie wird vom Python-Interpreter immer dann aufgerufen, wenn eine String-Repräsentation benötigt wird, bspw. im Anwendungsszenario des print() Befehls.

Einfache Verständnisfragen

Was ist der Unterschied zwischen Klasse und Objekt?

Diese Frage wurde bereits weiter oben beantwortet.

Was ist Polymorphie?

Polymorphie, oder auf Deutsch Vererbung, bezeichnet in der objektorientierten Programmierung, Klassenhierarchien aufzubauen. Die Idee ist hier, Klassen je nach Sichtweise immer weiter zu spezialisieren oder andersherum immer weiter zu abstrahieren. So können Redundanzen im Code minimiert werden, und auch umfangreiche Fallunterscheidungen. Ein Objekt einer abstrakten Klasse kann bspw. eine einheitliche Methodensignatur festlegen, während Objekte von spezialisierteren, geerbten Klassen konkreter auf die Attribute eingehen. Wichtig in diesem Zusammenhang ist das Konzept des Operator Overloading.

Warum muss immer self in Klassenmethoden auftauchen?

Der Aufruf OBJEKT.METHODE(ARGUMENTE) ist in Python "unter der Haube" eine Abkürzung für KLASSE.METHODE(OBJEKT,ARGUMENTE). Beispiele finden Sie in diesem Abschnitt.

Transferfragen

Was ist der Unterschied zwischen _myattribute und __myattribute?

Beide Varianten machen das Attribut privat. Der Unterschied ist nur bei Vererbung von Bedeutung. Doppelte Unterstriche definieren das Attribut in der Klasse als privat, einfache Unterstriche machen das Attribut privat in allen vererbten Klassen. In der Praxis sind fast immer einfache Unterstriche die schlauere Wahl, es sei denn, die abstraktere Klasse erlaubt den Lese- und Schreibzugriff über eine entsprechende Methode.

Geben Sie ein Beispiel jenseits der Lerneinheit an für die Verwendung von __add().

Zur Erinnerung: __add() ist die Methode, die für eine Klasse den Operator + überlädt. Dies nutzen wir seit der ersten Lerneinheit bspw. für die String-Konkatenation.

Als Beispiel können wir uns den Unterschied zwischen reellen und komplexen Zahlen vorstellen: Bei komplexen Zahlen sind sowohl der Realteil als auch der Imaginärteil zu addieren, als reelle Zahlen.

Als weiteres Beispiel können wir uns eine Statistik vorstellen, konkret die Eingabe von Rückmeldungen einer Umfrage zur Gesamtaggregation aller Rückmeldungen. Die "Addition" kann hier in die Berechnung von Mittelwerten und Medianen resultieren.

Erklären Sie mit dem Wissen aus dieser Lerneinheit, warum die Subtraktion von Brüchen mit der Klasse Fraction (import fractions) aus dem Stand mit dem Minuszeichen funktioniert. Ein Blick in help(fractions) mag hierfür hilfreich sein.

Diese Frage wurde im Prinzip bereits in der vorherigen Frage beantwortet: Die Klasse Fraction implementiert eine sinnvolle __sub(self,...) Methode, die wir automagisch nutzen können.

Erläutern Sie mit einem Blick in den Quelltext bzw. des Hilfetexts des Pakets fractions, warum Sie Brüche einfach in den print() Befehl stopfen können.

Das liegt daran, dass diese Klasse eine vernünftige __str(self) Methode bereitstellt.

Erstellen Sie eine geerbte Klasse LaTeX_Fraction auf der Basis der Klasse Fraction, die eine neue Funktion to_latex() zur Verfügung stellt und bspw. für LaTeX_Fraction(3,4).to_latex() den String $3/4$ zurückgibt.

Hier sind wir in der Beantwortung faul, und implementieren die Musterlösung direkt. Der Weg zur Lösung besteht, nach dem Verständnis von Polymorphie, Kapselung und Privatheit, einfach darin, in der Dokumentation der Ausgangsklasse nachzusehen, wie man Zugriff auf die Zahlwerte von Zähler und Nenner erhält. Der Rest ist Stringarithmetik und String-Escapesequenzen für den LaTeX-Syntax.