

Rekursive Datenstrukturen

Übung 10

Einführung in die Programmierung

Über dieses Übungsblatt

In diesem Übungsblatt möchten wir zunächst auf das objektorientierte Interface von Basic-IO eingehen. Wir nutzen diese Möglichkeit um rekursive Datenstrukturen besser kennenzulernen.

Objektorientiertes Interface von Basic-IO

Basic-IO bietet abgesehen von den bisher verwendenden einfachen "draw" Befehlen die Möglichkeit komplexere Objekte durch Komposition aus einfachen Objekten (Kreisen, Rechtecken) zu erstellen. Wir empfehlen dazu Kapitel 4.2 des Basic-IO Handbuchs zu lesen. Hier ein funktionierendes Minimalbeispiel:

```
import jguvc_eip.basic_io as bio
from jguvc_eip import image_objects as iobj

rect1 = iobj.Rectangle(10,10)
circle = iobj.Circle(20)


obj_stack = iobj.HorizontalStack([rect1, circle])
bio.draw_object(obj_stack)
```

Die einzelnen Objekte oder auch Stapel von Objekten können dann gemeinsam verändert (gedreht, verschoben oder skaliert) werden bevor sie gezeichnet werden, bsp. mit der `translate` Funktion.

Aufgabe Newton'sche Physik 1

Letzte Änderung: 06. July 2023, 13:26 Uhr

20 Punkte — [im Detail](#)

Ansicht:  | 



Das zu simulierende Objekt

Wir werden über die nächsten Übungsblätter eine einfache Physiks simulation mit Schwerkraft programmieren und in (kleinen) Schritten erweitern. Wir möchten das oben abgebildete Objekt simulieren: eine Stahlkugel steckt anfangs oben auf einem vertikalen Draht und fällt herunter; oben und unten befinden sich "Stopper", die die Kugel aufhalten.

- 1) Zeichnen Sie das Objekt wie oben in der Abbildung angegeben zunächst ohne den roten Ball. Erstellen Sie dazu drei `iobj.Rectangle` Objekte, die Sie dann mit `iobj.VerticalStack` übereinander stapeln; die Gerade kann durch ein Rechteck mit Breite 1 substituiert werden.
- 2) Erstellen Sie die rote Kugel als `iobj.Circle` Objekt und fügen Sie dieses zu dem in 1) erstellten Objekt mit `iobj.Overlay` hinzu.
- 3) Animieren die Kugel, so dass Sie mit konstanter Geschwindigkeit (also wie alle bisherigen Animationen) von oben nach unten "fällt"; die Grenzen müssen so gewählt werden, dass sich die Kugel nicht mit den Rechtecken überschneidet.

Hinweis: Verwenden Sie die `iobj.Translate` Funktion an der richtigen Stelle im Code, um die Kugel zu bewegen.

Nun wollen wir das Fallen mit konstanter Beschleunigung, nicht konstanter Geschwindigkeit simulieren; das Ergebniss sollte deutlich realistischer sein. Das einzige Physikalische Hintergrundwissen, dass man benötigt ist auf Schulniveau und lässt sich in dem folgenden Diagramm zusammenfassen:



Um ein einfaches Fallen eines Objektes zu simulieren, reicht es aus eine konstante Beschleunigung anzunehmen. Auf der Erde ist dies $G = 9,81m/s^2$, kann aber in unserer Simulation o.B.d.A. auf jeden beliebigen (sinnvollen) konstanten Wert gesetzt werden.

In jedem diskreten Zeitschritt unserer Simulation passiert also grundlegend folgendes: die Geschwindigkeit wird um die Beschleunigung erhöht und die Position wird um die Geschwindigkeit erhöht. Desweiteren muss man überprüfen, ob man die untere Grenze ("Stopper") erreicht hat; falls ja, kann man diesen selbstverständlich nicht überschreiten. In Falle einer solchen Kollision wird die Beschleunigung und Geschwindigkeit auf null gesetzt und das Objekt bewegt sich nicht mehr.

- 4) Simulieren Sie das Fallen der Kugel mit konstanter Beschleunigung.

Hinweis: Falls Sie das Flackern stört können Sie den folgenden Codeschnipsel an den Anfang ihrer Mal-Schleife verwenden um [double buffering](#) zu verwenden. Dies ist allerdings rein Ästhetisch und beeinflusst die Bewertung nicht.

```
bio.set_active_image(db_active_buffer)
bio.set_visible_image(db_visible_buffer)
bio.copy_image(db_visible_buffer, db_active_buffer)
db_visible_buffer, db_active_buffer = db_active_buffer, db_visible_buffer
```



`db_active_buffer` `db_visible_buffer` sind lediglich integer-Zahlen und können am Anfang des Codes initial auf 0 und 1 gesetzt

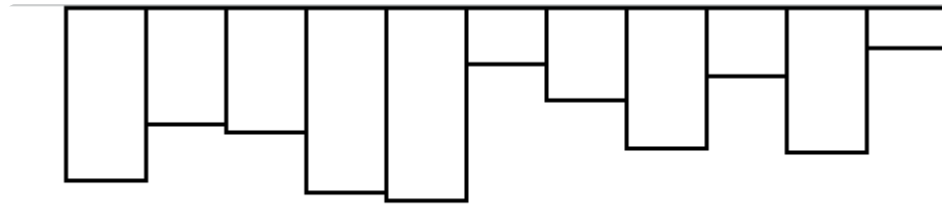
werden.

Aufgabe Rekursive Datenstrukturen

Letzte Änderung: 14. July 2023, 12:00 Uhr

20 Punkte — [im Detail](#)

Ansicht:  | 



So könnte ein Histogramm aussehen

Da wir in den letzten Wochen genug Zeit mit dem Ein/Auslesen von Daten verbracht haben arbeiten wir diese Woche einfach mit zufälligen Daten um Zeit zu sparen. Erstellen Sie eine Liste von 50 Zufallszahlen zwischen 0 und 100 mit :

```
from random import randint
```

```
list = [randint(0,100) for i in range(100)] # Ich bin eine list-comprehension. Praktisch, aber nur in einfachen Fällen zu benutzen!
```

1) Erstellen Sie erneut ein Histogramm (siehe Blatt 7) mit 50 Balken, deren Breite 10 ist und deren Höhe den Zufallswerten entspricht. Verwenden Sie diesmal aber das Objektorientierte Interface von Basic-IO. Erstellen Sie dazu einfach Rechteck-Objekte der richtigen Höhe `iobj.Rectangle`, speichern Sie sie in einer Liste `rects` und stapeln Sie sie mithilfe der `iobj.HorizontalStack([rect1, rect2,...])` Funktion. **Diesmal werden keine Achsenbeschriftungen, Ticks o.ä. verlangt.** Zeigen Sie das Histogramm für fünf Sekunden mit der `bio.draw_object` Methode an und löschen Sie dann die Ausgabe.

Hinweis: Da die Objekte mit `HorizontalStack` standardmäßig oben links aligniert werden, ist unser Balkendiagramm "falsch herum", das ist aber nicht weiter schlimm.

2a) Nun möchten wir die Art, wie wir unser zusammengesetztes Balken-Objekt erstellen rekursiv gestalten. Anstatt mit einem Aufruf alle Objekte zu vereinen, vereinen wir immer nur zwei Objekte auf einmal: `comp_obj = iobj.HorizontalStack([comp_obj, rect])`; diesen Befehl müssen wir selbstverständlich mehrfach ausführen. Zeigen Sie das erstellte Objekt dann wieder für fünf Sekunden an und löschen Sie danach die Ausgabe: so können Sie überprüfen ob das Bild das gleiche ist wie zuvor! Fügen Sie eine kurze Pause zwischen dem Anzeigen beider Histogrammen ein (weißer Bildschirm), so dass man erkennt wann die Anzeige wechselt.

Hinweis: Halten Sie ihre Lösung allgemein, sie sollte für eine beliebig lange Liste funktionieren.

Hinweis 2: Basic-IO ist nativ fähig, rekursiv gestapelte Datenstrukturen in einem Draw-Befehl anzuzeigen.

2b) Die in 1) verwendete Datenstruktur ist grundlegend eine Liste, also eine Sequenz von Objekten. Welche Datenstruktur haben wir in 2a) erstellt? Sie sollen nun einen Graphen (also ein Diagramm aus Kanten und Knoten) erstellen, der die Datenstruktur repräsentiert. Dies können Sie direkt Digital in einem Programm tun, z.B. mit [Dia](#), oder per Hand zeichnen und es Abfotografieren. Jeder Knoten soll einem `HorizontalStack` oder `Rectangle` Objekt entsprechen; Für jedes `HorizontalStack` Objekt verbinden wir seinen Knoten (mit einer Kante) mit den Knoten aller Objekte, die es enthält.

Tipp: Sie können einen Debugger verwenden um die Struktur der geschachtelten `HorizontalStack` Objekte zu verstehen.

3a) Wir möchten nun die rekursive Datenstruktur aus 2a) wieder in eine Liste überführen. Überlegen Sie sich, wie Sie mit einer Schleife (`while` oder `for`) über alle Elemente der rekursiven Datenstruktur iterieren, und diese dann extrahieren können. Erstellen Sie zunächst eine leere Liste `extracted_items = []`, iterieren Sie über alle Elemente der rekursiven Datenstruktur und hängen Sie in jedem Iterationsschritt ein Objekt an ihre Liste an mit `extracted_items.append([item])`. Packen Sie die Elemente der liste alle erneut in ein `HorizontalStack` Objekt und zeigen das Histogramm erneut an und überprüfen Sie so erneut, ob das Bild das gleiche ist. Verwenden Sie erneut eine kurze Pause zwischen den Anzeigen.

Tipp: Sie können mit `comp_obj.objects` auf die Liste aller Objekte zugreifen, die der `HorizontalStack comp_obj` enthält.

3b) Lösen Sie die Aufgabe 3a) mit einer rekursiven Funktion.

Tipp: Sie können `isinstance(obj, iobj.HorizontalStack)` feststellen ob `obj` eine Instanz der Klasse `iobj.HorizontalStack` ist oder nicht. Dies könnte für ihr Rekursionsende nützlich sein!