## *Wichtige Hinweise:*

1. Sie müssen die Aufgaben A1, A2 und A3 bearbeiten.

2. In Ihrem Ruby-Prüfungsprojekt finden Sie vier Ordner ***a1, a2,*** und ***a3***. Alle Klassen, Module und Methoden einer Aufgabe gehören in das entsprechende Subverzeichnis.

3. Schreiben Sie ***in jedes Script, das Sie bearbeiten, Ihre Matrikelnummer***.

***4.*** Zu erreichen sind ***95 Pkt.*** Davon sind 5 Punkte optional.

# A1 Einzelaufgaben (Punkte 30 Pkt)

1. Gegeben die Definition der Näherungsformel für $\ln(x)\ f\ddot{u}r\ x > 0$

$$\ln(x) \approx ln(x,n) = \sum_{i=0}^{n} \frac{(x-1)^{(2*i+1)}}{(2*i+1)*(x+1)^{(2*i+1)}} \quad ,\text{für x} > 0$$

   *a.* Berechnen Sie die Formel **iterativ**. Geben Sie ***false*** zurück, wenn die Methode mit ungültigen Werten für x und n aufgerufen wird. (***5 Pkt***)

   *b.* Berechnen Sie die Formel **rekursiv**. Geben Sie ***false*** zurück, wenn die Methode mit ungültigen Werten für x und n aufgerufen wird. (***5 Pkt***)

***Beispielaufrufe entnehmen Sie bitte den RUnit Tests***

**2.** Schreiben Sie eine Methode ***collect_ary_plus_depth*** für die Klasse ***Array***, die für ein beliebig geschachteltes Array, für jedes enthaltene Array die Schachtelungstiefe berechnet. Das Ergebnis der Methode ist eine Array von 2-elementigen Arrays, von denen das erste Element die Schachtelungstiefe und das zweite Element ein Array ist. Die Schachtelungstiefe für das erste Array ist 0. (***10 Pkt***)

***Beispielaufruf mit Ergebnis:***

```
[1,[2,[3]]].collect_ary_plus_depth()

Ergebnis: [[0, [1, [2, [3]]]], [1, [2, [3]]], [2, [3]]]
```

**3.** Schreiben Sie für die Klasse ***Hash***, die Methoden:

   a. ***verknuepfe(other_hash)***: Die Methode sucht zu den Werten des ersten Hashes gleiche Schlüssel im zweiten Hash und bildet dann den Schlüssel des ersten Hashes auf den Wert des zweiten Hashes ab. Die Methode arbeitet nicht destruktiv. (***5 Pkt***)
   ***Beispielaufruf mit Ergebnis:***

```
({1=>4, 2=> 7, 3=>7}).verknuepfe({4=> 9, 7=> 13, 9=>14})

Ergebnis: {1=>9, 2=>13, 3=>13}
```

   b. ***injektiv?()***: Die Methode prüft, ob der Hash eine injektive Abbildung definiert. Eine Abbildung ist injektiv, wenn jedes Element der rechten Seite nur einen Partner auf der linken Seite hat. (***5 Pkt***)
   ***Beispielaufruf mit Ergebnis:***

```
({1=>4, 2=> 7, 3=>7}.injektiv?())  Ergebnis: false
({4=> 9, 7=> 13, 9=>14}.injektiv?()) Ergebnis: true
```

**Hinweise:**

- Schreiben Sie Ihre Lösung in das Skript ***A1.rb*** und verwenden Sie nur die in der Aufgabe genannten Methodennamen.
- ***A1Test.rb*** enthält die Unit-Tests zu den Aufgaben.

## A2 Objekte vergleichen und sortieren (30 Punkte)

1. Implementieren Sie die Klasse ***Wuerfel***. Ein Würfel hat genau eine Eigenschaft, die Kantenlänge. Schreiben Sie eine Methode für die Initialisierung und eine Methode, die einen Würfel als lesbare Zeichenkette zurückgibt. (***5 Pkt***)
2. Sie sollen gleiche Würfel in Arrays wiederfinden können. Ergänzen Sie die Klasse Würfel um die Methode, die das Auffinden zweier inhaltsgleicher Würfel in einem Array ermöglicht. Zwei Würfel sind inhaltsgleich, wenn sie gleiche Kantenlänge haben.
   Denken Sie daran, dass Sie aus Effizienzgründen, die einfachen Prüfungen vor dem eigentlichen Inhaltsvergleich implementieren. (***6 Pkt***)
3. Schreiben Sie die Methoden für die Klasse Würfel, die sicherstellen, dass nicht zwei gleiche Würfel in eine Menge (***Set***) eingefügt werden können, bzw. die sicherstellen, dass über inhaltsgleiche Würfel, die als Schlüssel, in einem ***Hash*** verwendet werden, Einträge sicher wiedergefunden werden.
   Denken Sie daran, dass Sie aus Effizienzgründen, die einfachen Prüfungen vor dem eigentlichen Inhaltsvergleich implementieren. (***9 Pkt***)
4. Erweitern Sie die Klasse Würfel um eine Methode, so dass ein Array von Würfeln sortiert werden kann und Würfel mit den Methoden (***<,<=,>,>=***) verglichen werden können. Die Ordnung für Würfel ist über die Kantenlänge definiert. (***6 Pkt***)

   (***4Pkt*** für eine vollständig korrekte Lösung)

Hinweise:

- Implementieren Sie die Klasse ***Wuerfel*** im Verzeichnis ***a2.***
- Die zugehörigen RUnit-Tests finden Sie in ***WuerfelTest.rb***.

## A3 Klassen und Objektrekursion (35 Punkte / 5 korrekte Gesamtlösung)

Gegeben ein nicht vollständiges Klassenmodell für den Aufbau eines Zuges. Ein Zug besteht aus Zugkomponenten. Die erste Komponente ist immer eine Lok, dann folgen 0-n Personenwagen. Es gibt Personenwagen 1'ter und 2'ter Klasse. Sie sollen das Klassenmodell so erweitern / ergänzen, dass das Ankoppeln von einzelnen Zugkomponenten an einen Zug möglich ist. Dabei müssen gewisse Regeln eingehalten werden. Im Einzelnen:

1. Für die Klasse ***Zug***:
    a. Schreiben Sie die Methode ***ankoppeln(zugkomponente)***, die an das Ende eines Zuges die übergebene Zugkomponente anhängt. Verwenden Sie die Methode ***letzte_komponente()*** der Klasse Zug, die eine Referenz auf die letzte Zugkomponente des Zuges liefert. Wenn der Parameter der Methode ***ankoppeln*** nicht vom Typ ***ZugKomponente*** ist, soll die Methode ***false*** zurückliefern. Führen Sie das Ankoppeln einer Komponente an einen Zug auf das Ankoppeln einer Zugkomponente an eine Zugkomponente zurück. (***3 Pkt***)
    b. Die Methode ***letzte_komponente()*** liefert eine Referenz auf die letzte Zugkomponente des Zuges. Lösen Sie das Ermitteln der letzen Komponente mit einer geeigneten Methode von Enumerable. (***3 Pkt***)
    c. Um b. lösen zu können, müssen Sie den Zug zunächst **enumerierbar** machen. Führen Sie den Iterator auf einen Iterator für Zugkomponenten zurück. (***3 Pkt***)
2. Das Modul ***ZugKomponente***: Jede Zugkomponente hat einen Nummer und einen Nachfolger, der ebenfalls eine Zugkomponente ist. Bei der Initialisierung einer Zugkomponente gibt es noch keinen Nachfolger und die Nummer ist standardmäßig 0. Durch Ankoppeln von Zugkomponenten wird der Nachfolger auf die nachfolgende Zugkomponente gesetzt. Dies kann aber nur in den konkreten Subklassen erfolgen, da nur diese die Regeln für das Ankoppeln kennen. ***ZugKomponente*** führt in der Methode ***ankoppeln*** Basisprüfungen durch, die für jedes Ankoppeln in den Subklassen durchlaufen werden müssen. Zur Unterstützung der Methoden aus ***Zug*** und der konkreten Subklassen müssen in ***ZugKomponente*** folgende Erweiterungen vorgenommen werden:
    a. Machen Sie Zugkomponenten **enumerierbar**. Reichen Sie den Aufruf des Iterators an den Nachfolger weiter und achten Sie darauf, dass Blöcke für jede Zugkomponente ausgeführt werden müssen. (***5 Pkt***)
    b. Schreiben Sie die Methode ***nummerieren(nr)***, die der Zugkomponente als Nummer den Parameter zuweist und die Nummerierung mit inkrementierter Nummer rekursiv auf dem Nachfolger aufruft. (***3 Pkt***)
3. Für die Klasse ***Lok***:
    a. ***Lok*** ist eine ***ZugKomponente***: Stellen Sie diese Vererbungsbeziehung her. (***1 Pkt***)
    b. Schreiben Sie die Methode ***ankoppeln(zugkomponente)***: (***3.5 Pkt***)
        i. Die Methode prüft die Bedingungen der Superklasse.
        ii. Die Methode stellt sicher, dass an eine Lok nur Personenwagen angehängt werden dürfen.
        iii. Wenn alle Bedingungen erfüllt sind, dann wird die übergebene Zugkomponente zum Nachfolger der Lok und die Nummerierung des Nachfolgers angestoßen.
        iv. Die Methode gibt ***true*** zurück wenn das Ankoppeln erfolgreich war, sonst ***false***.
4. Für die Klasse ***PersonenWagen***:
    a. ***PersonenWagen*** ist eine ***ZugKomponente***: Stellen Sie diese Vererbungsbeziehung her. (***1 Pkt***)
    b. Bei der Initialisierung wird die Klasse des Wagens übergeben. Stellen Sie auch sicher, dass die Anteile der Superklasse initialisiert werden. (***1.5 Pkt***)
    c. Schreiben Sie die Methode ***ankoppeln(zugkomponente)***: (***6 Pkt***)

i. Die Methode prüft die Bedingungen der Superklasse.
ii. Die Methode stellt sicher, dass an einen Wagen zweiter Klasse nur Wagen zweiter Klasse angehängt werden dürfen. An Wagen erster Klasse dürfen erste und zweite Klasse Wagen aber keine Loks angehängt werden.
iii. Wenn alle Bedingungen erfüllt sind, dann wird die übergebene Zugkomponente zum Nachfolger und die Nummerierung des Nachfolgers angestoßen.
iv. Die Methode gibt ***true*** zurück wenn das Ankoppeln erfolgreich war, sonst ***false***.

**Hinweise:**

- Ergänzen Sie die Klassen, Module und Methoden im Verzeichnis a3.
- Die meisten Methodenrahmen sind vorgegeben.
- Die zugehörigen RUnit-Tests finden Sie in ***ZugTest.rb.***

**Beispiele mit Graphik:**

```
@lok = Lok.new()
@zug = Zug.new(@lok)
@pw1 = PersonenWagen.new(2)
@pw2 = PersonenWagen.new(2)
@pw3 = PersonenWagen.new(1)
```

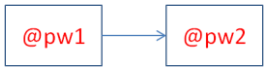

**Beispiel 1: einzelne Komponenten ankoppeln**

```
@zug.ankoppeln(@pw3)
@zug.ankoppeln(@pw1)
@zug.ankoppeln(@pw2)
```


**Beispiel 2: Kette von Komponenten ankoppeln**

```
@zug.ankoppeln(@pw3)
```


```
@pw1.ankoppeln(@pw2)
```




```
#Kette von Wagen
@zug.ankoppeln(@pw1)
```