Allgemeines

- Da dieser Versuch aus mehreren Quelldateien und XML-Dateien besteht. Speichern Sie diese in ein Verzeichnis mit dem Namen:
  `p05NameVorname`

  Name ist durch Ihren Nachnamen und Vorname durch Ihren Vornamen zu ersetzen. Der Verzeichnisname sollte weiterhin keine Leerzeichen enthalten.

- Bevor Sie Ihre Lösung den Betreuern vorstellen, können Sie mit dem in moodle bereitgestellten Unit-Test `p05wt_unit.py` überprüfen, ob Ihre Lösung die richtigen Ergebnisse liefert.
  - Im Anaconda-Prompt können Sie den Unit-Test in Ihrem Projektverzeichnis über folgende Anweisung ausführen:
    `python p05wt_unit.py`
  - Achten Sie **haargenau** auf die in der Aufgabenstellung dargestellte Ausgabe: Ein Leerzeichen zu viel, Groß-/Kleinschreibung nicht beachtet oder falsche Zeilenvorschübe gibt der Unittest als Fehler aus!
- Schreiben Sie Ihre Funktionstests so, dass diese nicht bei Aufruf des Unit-Tests aufgerufen werden.
  Die geforderten Funktionstests finden Sie in den Teilaufgaben angegeben.

- Sollten in der Aufgabenstellung Fragen zu beantworten sein, dann beantworten Sie diese bitte in einem Dokumentationskommentar am Anfang Ihre Python-Datei, z. B.:

```
"""
      Antworten zu den in der Aufgabenstellung gestellten
      Fragen:
      Frage 1: Ihre Antwort
      Frage 2: Ihre Antwort
      ...........
"""
```

Breadboard-Replay

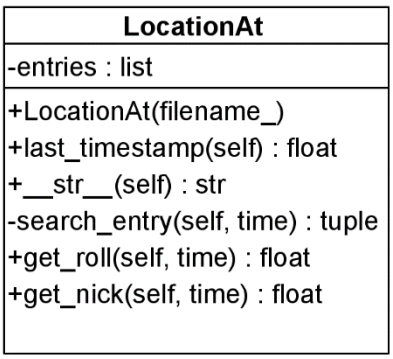
Im Versuch 2 von TI3-Embedded Systems wurde die Lage des Breadboards über die Programmlaufzeit persistent in einer CSV-Datei aufgezeichnet. Ein Datensatz der Datei gibt den Zeitpunkt der Messung in Millisekunden seit Programmstart an, sowie die gemessenen X-, Y- und Z-Beschleunigungswerte. Ausschnitt der Datei `yRotFirst.csv`:

```
63;-1.003222;0.014366;9.862219
166;-1.008010;0.146054;9.867008
269;-1.051108;-0.040704;9.929260
    ...
```

Die Bewegungen sollen dann grafisch mithilfe einer GUI-Anwendung abgespielt werden.

1. Die Klasse `LocationAt`
   Ein Objekt der Klasse übernimmt die Aufgabe die Datensätze der CSV-Datei geeignet einzulesen. Für einen bestimmten Zeitpunkt aus den Beschleunigungswerten (x, y, z) die beiden Rotationswerte (Y-Rotation und X-Rotation) zu ermitteln. Diese stellen die Neigungswerte des Breadboards zu dem angegebenen Zeitpunkt dar.
   Die Klasse soll in der Datei `location.py` implementiert werden.

1.1. Implementieren Sie die Klasse `LocationAt`, deren Struktur in folgendem Klassendiagramm abgebildet ist.



- Der Konstruktor liest die angegebene CSV-Datei ein und speichert alle Einträge in einer Liste von Tupel ab (Attribut: `entries`).
  Der Dateiname wird dabei als Funktionsargument übergeben.
  Achten Sie dabei darauf, dass alle Werte der Tupel aus `float`-Werten bestehen.
- Die Methode `last_timestamp` gibt den Zeitstempel des letzten Datensatzes zurück, sollten keine Datensätze vorhanden sein, dann wird `-1.0` zurückgegeben.
- Die Methode `__str__` soll die Ausgabe des Dateiinhalts gemäß dem in 1.2 dargestellten Beispielaufruf ermöglichen. (Bis zur Zeile vor dem Text `"Letzter Zeitstempel: …"`)
- Die **private** Methode `search_entry` liefert für einen bestimmten Zeitpunkt ein Tupel mit den drei Beschleunigungswerten. Gibt es für den Zeitpunkt keinen Datensatz, so sind die davor zuletzt gemessenen Werte zu verwenden. Sollte der Zeitpunkt vor dem ersten Datensatz liegen so ist das Tupel `(0.0, 0.0, 0.0)` zurückzugeben. Falls der Wert des Funktionsarguments zeitlich nach dem letzten Zeitstempel liegen so ist None zurückzugeben.
- Die Methoden `get_roll` (Rollwinkel) und `get_nick` (Nickwinkel) geben unter Zuhilfenahme der privaten Methode `search_entry` die Neigungswinkel des Breadboards für einen bestimmten Zeitpunkt zurück.

  Dabei gilt:

  Länge des Vektors: $l = \sqrt{x^2 + y^2 + z^2}$
  Falls ein Eintrag (Vektor) für den Zeitpunkt gefunden wurde und die Länge größer als 0 ist, nutzen Sie folgende Formeln:[1]

  Nickwinkel (Neigungswinkel der langen Kante des Breadboards):

  $$\theta = arcsin\left(\frac{-y}{l}\right)$$

  Rollwinkel (Neigungswinkel der kurzen Kante des Breadboards):

  $$\varphi = arcsin\left(\frac{x}{l \cdot \cos(\theta)}\right)$$

  andernfalls wird 0 für den entsprechenden Winkel zurückgegeben.

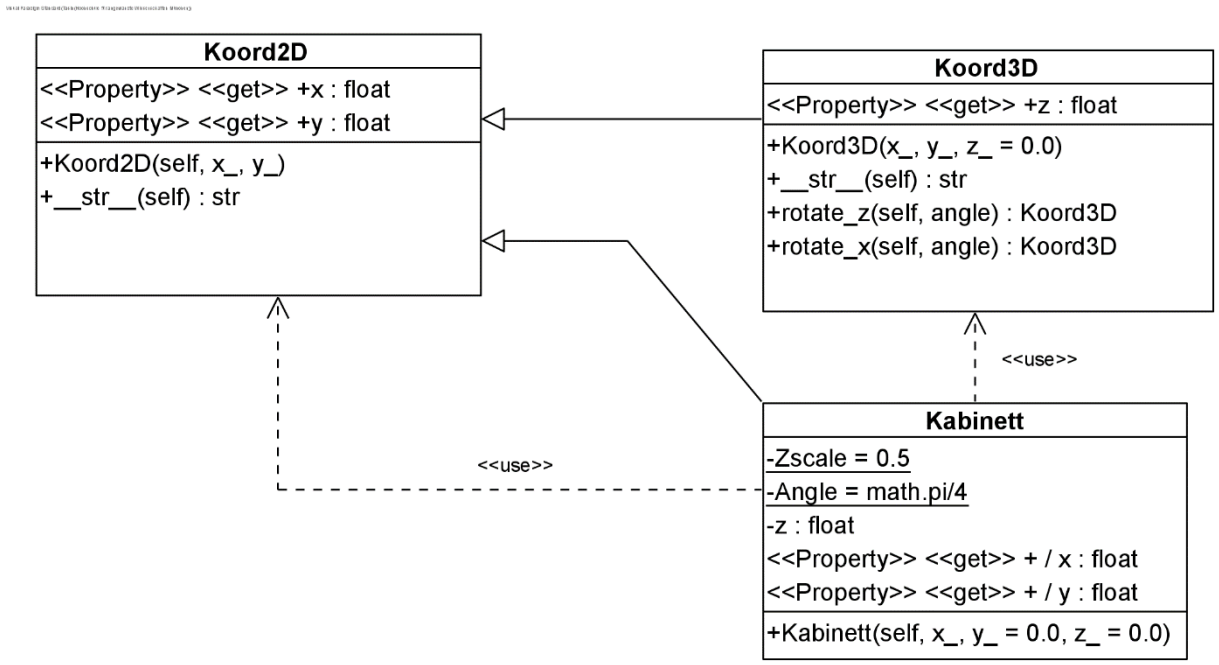
[1] s. a. https://de.wikipedia.org/wiki/Eulersche_Winkel

1.2. Testen Sie die Klasse mittels der zur Verfügung gestellten Dateien `location_test.py` und `xRotFirst.csv`. Folgende Ausgabe soll sich dabei ergeben:

```
Eintraege
    63: x: -1.003 y:  0.014 z:  9.862
   166: x: -1.008 y:  0.146 z:  9.867
   269: x: -1.051 y: -0.041 z:  9.929
        ... weitere Einträge ...
 15797: x: -0.951 y: -0.043 z:  9.903
 15901: x: -0.946 y:  0.000 z:  9.915
 16006: x: -1.030 y:  0.010 z:  9.980

Letzter Zeitstempel: 16006.0
     0: nick:  0.00 roll:  0.00
  1000: nick:  0.29 roll: -5.88
  2000: nick: -0.11 roll: -5.92
  3000: nick: 13.02 roll: -9.92
  4000: nick: 27.62 roll:-11.53
  5000: nick: 45.90 roll: -9.69
  6000: nick: 63.27 roll: -6.49
  7000: nick: 79.50 roll:-14.93
  8000: nick: 86.15 roll:-65.66
  9000: nick: 83.46 roll:-11.67
 10000: nick: 54.67 roll:-14.13
```

2. Die Koordinatenklassen `Koord2D`, `Koord3D` und `Kabinett`
Implementieren Sie die Klassen, gemäß der nachfolgenden Implementierungsvorgaben, so dass deren Struktur und Beziehungen zueinander dem folgenden Klassendiagramm entsprechen.
Die Klassen sollen in der Datei `koordinate.py` implementiert werden.



2.1 Die Klasse `Koord2D` kapselt eine zweidimensionale Koordinate.
Für die nur lesbaren Eigenschaften `x` und `y` sind geeignete private Attribute zu wählen.

Folgende Anweisungen

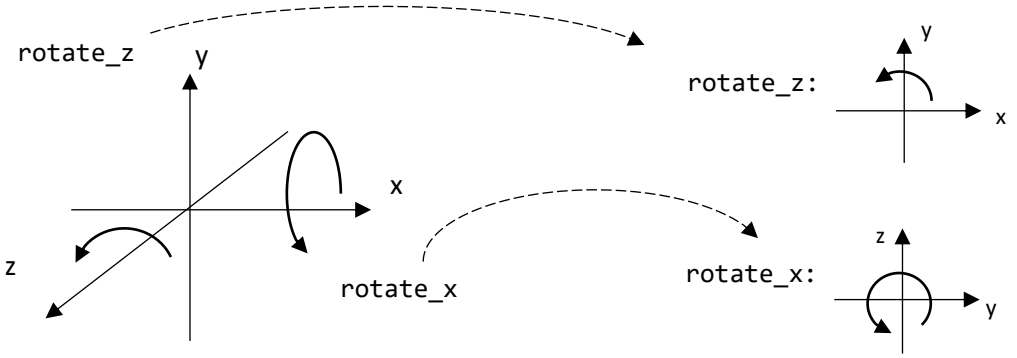
```
print(Koord2D(3.5,-4))
print(Koord2D(-3,0))
```

sollen folgende Ausgabe ergeben

```
(3.50;-4.00)
(-3.00;0.00)
```

2.2 Die Klasse `Koord3D` spezialisiert die Klasse `Koord2D` und kapselt eine dreidimensionale Koordinate.
Für die nur lesbare Eigenschaft `z` ist ein geeignetes privates Attribut zu wählen.
Beachten Sie das optionale Funktionsargument im Konstruktor.

Die Methoden `rotate_z` und `rotate_x` liefern jeweils ein neues `Koord3D`-Objekt, welche die um den Winkel `angle` gedrehte 3D-Koordinate darstellt.
Dabei wird die Koordinate mithilfe von `rotate_z` um die Z-Achse gedreht, während `rotate_x` die Koordinate um die X-Achse dreht.



(Darstellung des Koordinatensystems der GUI-Zeichenfläche)

(Drehen des Koordinatensystems)

Tipps:

- Verwenden Sie die komplexe Ebene zum Rotieren der Koordinate.
  Nutzen Sie für `rotate_z` den x-Wert als Realteil und den y-Wert als Imaginärteil, für `rotate_x` den y-Wert als Realteil und den z-Wert als Imaginärteil.
- Ein Rotieren der Koordinate kann auch als Drehung des (dadurch repräsentierenden) komplexen Vektors betrachtet werden (z. B. über Multiplikation mit einem geeigneten Einheitsvektor).

Folgende Anweisungen

```
import math

k1=Koord3D(3, 4, -1.5)
print(k1)
k2=k1.rotate_z(math.pi/2)
print(k2)
k3=k1.rotate_x(math.pi/2)
print(k3)
```

sollen folgende Ausgabe ergeben

```
(3.00;4.00;-1.50)
(-4.00;3.00;-1.50)
(3.00;1.50;4.00)
```

2.3 Die Klasse `Kabinett` spezialisiert die Klasse `Koord2D` und kapselt eine dreidimensionale Koordinate. Der z-Wert dient dabei zur Festlegung eines Vektors, um die zweidimensionale Koordinate (x- und y-Wert) zu verschieben und damit die Kabinettperspektive zu realisieren.
Ein `Kabinett`-Objekt soll als erstes Funktionsargument (`x_`) neben einer Zahl, auch ein `Koord2D`- und ein `Koord3D`-Objekt akzeptieren.

Achtung: Ist das Argument `x_` eine Zahl, dann werden die beiden anderen Funktionsargumente zur Erzeugung einer `Kabinett`-Instanz verwendet.
Wird ein `Koord2D`-Objekt zur Konstruktion eines `Kabinett`-Objekts verwendet, so ist auch das Funktionsargument `z_` zu beachten.

Überschreiben Sie die Eigenschaftsmethoden (Property-Getter) für x und y um die transformiere Koordinate zu erhalten. Anstelle des entsprechenden Attributwerts wird der transformierte Wert zurückgegeben.
Für die transformierten Werte gelten:

$$x_{trans} = x - {}^{1}/_{2} \cdot z \cdot \cos\left(\frac{\pi}{4}\right)$$
$$y_{trans} = y - {}^{1}/_{2} \cdot z \cdot \sin\left(\frac{\pi}{4}\right)$$

Verwenden Sie zur Berechnung auch die statischen Datenkomponenten der Klasse `Kabinett`.

2.4 Testen Sie Ihre Klassen mit der bereitgestellten Python-Datei `koordinaten_test.py`. Folgende Ausgabe soll sich dabei ergeben:

```
k1_2d: (3.00;4.00)
k1_2d: x=3.0, y=4.0
k2_3d: (3.00;5.00;1.00)
k1r_3d: (5.00;-3.00;1.00)
k2r_3d: (3.00;1.00;-5.00)
k2r_3d: x=3.00, y=1.00, z=-5.00
k1_kabinett: (3.00;4.00)
k2_kabinett: (2.65;4.65)
k3_kabinett: (2.65;4.65)
k4_kabinett: (2.47;3.47)
```

3. Verwendung der Klassen zur grafischen Visualisierung mithilfe von `breadboard_GUI.py`.

3.1 Starten Sie das bereitgestellte Python-Programm `breadboard_GUI.py`, um Ihre Klassen zu testen. Als Programmparameter kann auch eine andere CSV-Dateien angeben, die dann die zeitliche Position des Breadboards entsprechend grafisch darstellt.
Achtung: Beim Starten der GUI über Spyder kann es zu Fehlerausgaben kommen, obwohl das Programm korrekt läuft. Testen Sie das Programm unbedingt über die Eingabeaufforderung!

3.2 Zeichnen Sie ein einfaches Klassendiagramm, das die Beziehungen der Klassen `BBoardWidget`, `MyTimer` und `LocationAt` zueinander und zu evtl. Elternklassen darstellt.
Auf die Angabe von Attributen, Properties und Methoden kann verzichtet werden.

3.3 *Für Profis (optional):* Integrieren Sie die `QTimer`-Funktionalität in die `BBoardWidget`-Klasse, so dass auf die Klasse `MyTimer` verzichtet werden kann.
Fügen Sie im linken oberen Eck einen `QPushButton` mit dem Buttontext "`Start`" ein, um das Breadboard-Replay erst nach Drücken auf diesen Knopf zu starten. Beachten Sie dabei, dass nach dem Starten der Animation der Knopf so lange deaktiviert sein soll, bis die Animation beendet ist.