Computing und Sensorik mit Python im Physikunterricht

Weiterbildungskurs der Deutschschweizerischen Physikkommission

1. Teil: Freitag, 27. Januar 2017

2. Teil: Samstag, 4. März 2017/11. März 2017

Teil 1:
Exemplarische Einführung in Python mit Physik-Simulationen

Gymnasium Bern-Neufeld, Bremgartenstrasse 133, 3012 Bern

V1.03

Kursprogramm 1. Teil

1. Tag: Freitag, 27. Januar 2017

09.00 - 09.15	Kurseröffnung, Christian Stulz (Präsident DPK)
09.15 – 11.30	Einstieg in Python an elementaren Beispielen aus der Mechanik Prinzipien der animierten Grafik für Simulationen Simulation des radioaktiven Zerfalls
11.30 - 12.10	Rechnergestütze Methoden in der Physik (Prof. Chr. Keller, ETHZ)
12.15- 13.30	Mittagessen in der Mensa
13.30 – 14.00	Python Entwickklungssysteme (Geany, PyCharm)
14.00 - 16.00	Random Walk Experimente aus der Akustik Datenspeicherung
16.00- 16.45	Ausgewähle Goodies und Highlights
16.45 -17.00	Q&A, Schlussdiskussion

Dozent: Aegidius Plüss unter Mitarbeit von Jarka Arnold

Website: www.python-exemplarisch.ch/dpkkurs

Physikunterricht im ständigen "Kampf" mit mathematischen Vorkenntnissen

- Numerische Simulationen sind oft einfacher als algebraische Herleitungen
- Simulationen können Experimente gewinnbringend ersetzen, insbesondere im Selbststudium (als Ergänzung eines Lehrmittels) oder bei grossem Aufwand
  z.B. Keplersche Gesetze aus Beobachtungen
- Simulationen sind oft "näher an der Relatität" als theoretische "Verkürzungen" auf Grund fehlender Mathematik-Kenntnisse. Bei Simulationen sind weniger Vereinfachungen ("Modellannahmen") nötig

---

- Exemplarisches Beipiel: Vertikaler Wurf
  - Einstieg: Experiment (nicht trivial, z.B. Highspeed-Kamera)
  - Fragen/Aufgabenstellung:
    - Bewegung beschreiben (Kinematik): v = v(t), y = y(t))
    - Begründung für Bewegung (Gewicht zieht nach unten, F = ma)
  - Mathematisch/Formel ohne Differentialrechnung nur mit einigen didaktischen "Verrenkungen":

$$x = x(t) = v_o t - \frac{g}{2} t^2$$

Arbeitshypothese für die Simulation: Beschleunigung ist konstant (Erdbeschleunigung)

Definition der Beschleunigung: $a = \frac{\Delta v}{\Delta t} = -g$

Differenzenquotient, Anschaulich: Änderung der Geschwindigkeit pro Zeiteinheit

Gesucht: Zeitliche Entwicklung, d.h.
ausgehend von der Geschwindigkeit $v(t)$ die neue Geschwindigkeit $v(t+\Delta t)$

$$\Delta v = v(t+\Delta t) - v(t) = a\Delta t \quad \text{oder:} \quad v(t+\Delta t) = v(t) + a\Delta t$$

in Worten:

neue Geschwindigkeit = alte Geschwindigkeit + Beschleunigung*Zeitintervall

mit $a = -g$

Analog für Zusammenhang zwischen Ort und Geschwindigkeit:

Definition der Geschwindigkeit: $v = \frac{\Delta y}{\Delta t}$ (i.a. nur für "kleine" Δt)

$$\Delta y = y(t+\Delta t) - y(t) = v\Delta t \quad \text{oder:} \quad y(t+\Delta t) = y(t) + v\Delta t$$

nicht Buchstabe s verwenden!

oder in Worten:

neuer Ort = alter Ort + Geschwindigkeit * Zeitintervall

Numerische Simulation: Der ausserphysikalische Aufwand muss sehr klein sein!

Kriterien:

- Das neu Gelernte muss nachhaltig sein->Übertragbarkeit auf andere Probleme
- Wenn immer möglich auf Vorwissen aufbauen->was ist Lehrperson/SuS bekannt?
- Einfachheit/Verfügbarkeit->neue Geräte? neue Software?auch zuHause?Gratis?
- Modern oder "verstaubt" (Schule muss aktuell sein, sollte nicht im Glashaus leben!)

Google:TigerJython->Download->Los geht's!

Kommentar (wichtiges Element der Programms)

```
# mech1.py
# Vertikaler Wurf ohne Luftwiderstand

g = 9.81 # Erdbeschleunigung (m/s^2)
dt = 0.1 # Zeitschritt (s)

# Anfangsbedingungen:
t = 0
y = 0  # Ort (m)
v = 20 # Geschwindigkeit (m/s)

while t < 10:
    print t, v, y
    v = v - g * dt
    y = y + v * dt
    t = t + dt
```

Variable, keine Typendeklaration,
Float (Genauigkeit wie in anderen Sprachen double)

Wiederholstruktur

Laufbedingung (keine Klammer nötig, mit : abschliessen)

Programmblock: Einrücken (keine Blockklammern)

Zuweisung (keine Gleichung, in Pascal :=)
Alter Wert wird genommen und damit gerechnet,
und dann eine neue Variable mit gleichem Namen gemacht.

print-Anweisung (Python V2.x: Schlüsselwort, Python 3.x: Funktion)
Komma-Trennung sehr praktisch, es können auch numerische
Grössen ausgeschrieben werden, in Python 3.x muss es ein String sein)

Übungen:

1. Verändern Sie den Zeitschritt
2. Vergleichen Sie mit theor. Wurfhöhe
3. Schreiben Sie "t = ... y =... v =..." aus

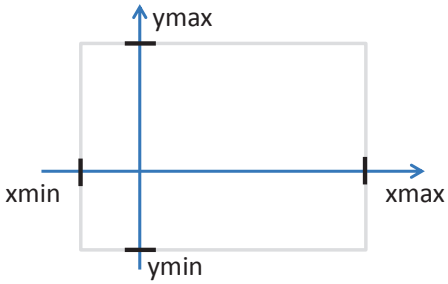
Bibliotheksimport (erweitert Namensraum, daher wichtig, damit es keine Namenskonflikte gibt)
Es gibt zwei Formen: `import ...` oder `from ... import ...`

```
# mech2.py
# Vertikaler Wurf ohne Luftwiderstand mit Grafik

from gpanel import *

g = 9.81 # Erdbeschleunigung (m/s^2)
dt = 0.1 # Zeitschritt (s)

# Anfangsbedingungen:
t = 0
y = 0  # Ort (m)
v = 20 # Geschwindigkeit (m/s)

makeGPanel(0, 10, -200,  50) #xmin, xmax, ymin, ymax
pos(t, y)
while t < 10:
    draw(t, y)
    v = v - g * dt
    y = y + v * dt
    t = t + dt
```

Erzeugt Grafikfenster (mit default Grösse) und gegebenem Koordinatensystem

Setzt Grafikcursor auf gegebenen Punkt, ohne zu zeichnen

Zeichnet Verbindungslinie zum letzten Punkt



Mathematisch-physikalische Orientierung
(In Informatik nicht üblich)

`pos(x1, y1)`
`draw(x2, y2)`
ersetzt
`line(x1, y1, x2, y2)`
(hat "Erinnerungsvermögen")

Übungen:

1. Verändern Sie die Anfangsgeschwindigkeit
2. Verändern Sie das Koordinatensystem
3. Ersetzen Sie pos/draw durch line

Grid und Labels möglichst einfach!

```
# mech2a.py
# Vertikaler Wurf ohne Luftwiderstand mit Grafik

from gpanel import *

g = 9.81 # Erdbeschleunigung (m/s^2)
dt = 0.1 # Zeitschritt (s)

# Anfangsbedingungen:
t = 0
y = 0  # Ort (m)
v = 20 # Geschwindigkeit (m/s)

makeGPanel(-1, 11, -220, 70)
drawGrid(0, 10, -200, 50)
pos(t, y)
while t < 10:
    draw(t, y)
    v = v - g * dt
    y = y + v * dt
    t = t + dt
```

Bereich auf allen Seiten um rund 10% vergrössern
(damit es Platz für die Beschriftung gibt)

Früheres xmin, xmax, ymin, ymax

Übungen:

1. Verändern Sie die Werte bei makeGPanel/drawGrid
2. Verwenden Sie drawGrid(xmin, xmax, ymin, ymax, farbe)
3. Zeichnen Sie den Graph mit anderer Farbe (setColor())
4. Orientieren Sie sich in der API Dokumentation

Ablauf in Echtzeit

```
# mech2b.py
# Vertikaler Wurf ohne Luftwiderstand mit Grafik
# Ablauf in Echtzeit

from gpanel import *
import time

g = 9.81 # Erdbeschleunigung (m/s^2)
dt = 0.1 # Zeitschritt (s)

# Anfangsbedingungen:
t = 0
y = 0  # Ort (m)
v = 20 # Geschwindigkeit (m/s)

makeGPanel(-1, 11, -220, 70)
drawGrid(0, 10, -200, 50, "darkgray")
pos(t, y)
while t < 10:
    draw(t, y)
    v = v - g * dt
    y = y + v * dt
    t = t + dt
    time.sleep(dt)
```

Modul `time` importieren

Delay einbauen . Achtung hier Sekunden (oft ms)
(Frage: gibt dies exakt eine Periode von 0.1 s?)

Übungen:

1. Prüfen Sie die "Echtzeit"
2. Zeichnen Sie den theoretischen Verlauf von y

Echte Simulation: Computeranimation (wie in einem Trickfilm)

Periodisch wiederholen:

Situation zeichnen
Einen Moment beibehalten
Neue Situation berechnen
Alte Situation löschen und

Reihenfolge kann etwas variieren

```
# mech3.py
# Vertikaler Wurf ohne Luftwiderstand mit Grafik
# Echtzeit-Simulation, Flackern

from gpanel import *
import time

g = 9.81 # Erdbeschleunigung (m/s^2)
dt = 0.1 # Zeitschritt (s)

# Anfangsbedingungen:
t = 0
y = 0  # Ort (m)
v = 20 # Geschwindigkeit (m/s)

makeGPanel(-6, 6, -220, 70)
while t < 10:
    clear()
    v = v - g * dt
    y = y + v * dt
    t = t + dt
    drawGrid(-5, 5, -200, 50, "darkgray")
    pos(0, y)
    fillCircle(0.3)
    time.sleep(dt)
```

Alles löschen

Neue Situation rechnen

Neues Bild zeichnen

Warten

```
# mech3a.py
# Vertikaler Wurf ohne Luftwiderstand mit Grafik
# Echtzeit-Simulation, kein Flackern, da enableRepaint(False)

from gpanel import *
import time

g = 9.81 # Erdbeschleunigung (m/s^2)
dt = 0.1 # Zeitschritt (s)

# Anfangsbedingungen:
t = 0
y = 0  # Ort (m)
v = 20 # Geschwindigkeit (m/s)

makeGPanel(-6, 6, -220, 70)
enableRepaint(False)
while t < 10:
    clear()
    v = v - g * dt
    y = y + v * dt
    t = t + dt
    drawGrid(-5, 5, -200, 50, "darkgray")
    pos(0, y)
    fillCircle(0.3)
    repaint()
    time.sleep(dt)
```

Schaltet automatisches Rendern aus

Löscht nur Bildbuffer, aber nicht Bildschirm

Zeichnet in Bildbuffer, aber nicht auf den Bildschirm

von "Hand" rendern (alles auf einen "Dätsch" zeichnen)

Übungen:

1. Verändern Sie den Zeitschritt. Feststellungen?


```
# mech3b.py
# Vertikaler Wurf ohne Luftwiderstand mit Grafik
# Mit Statusbar

from gpanel import *
import time

g = 9.81 # Erdbeschleunigung (m/s^2)
dt = 0.1 # Zeitschritt (s)

# Anfangsbedingungen:
t = 0
y = 0  # Ort (m)
v = 20 # Geschwindigkeit (m/s)

makeGPanel(-6, 6, -220, 70)
enableRepaint(False)
addStatusBar(30)
while t < 10:
    clear()
    v = v - g * dt
    y = y + v * dt
    t = t + dt
    drawGrid(-5, 5, -200, 50, "darkgray")
    pos(0, y)
    fillCircle(0.3)
    repaint()
    setStatusText("t = " + str(t) + ", v = " + str(v) + ", y =" + str(y))
    time.sleep(dt)
```

Statusbar mit 30 pixel Höhe erstellen (Zusatzfenster unten am GPanel)

Übungen:

1. String formatieren:
   `setStatusText("t = %6.2f s, v = %6.2f m/s, y = %6.2f m" %(t, v, y))`

Platzhalter (6 Stellen, auf 2 Nachkommastellen gerundet)

String in Statusbar schreiben
(numerische Werte in String casten)

```
# mech3c.py
# Vertikaler Wurf ohne Luftwiderstand mit Grafik
# Mit bewegtem Image

from gpanel import *
import time

g = 9.81 # Erdbeschleunigung (m/s^2)
dt = 0.1 # Zeitschritt (s)

# Anfangsbedingungen:
t = 0
y = 0  # Ort (m)
v = 20 # Geschwindigkeit (m/s)

makeGPanel(-6, 6, -220, 70)
enableRepaint(False)
img = getImage("sprites/alien.png") # load image
while t < 10:
    clear()
    v = v - g * dt
    y = y + v * dt
    t = t + dt
    image(img, 0, y) # show image
    repaint()
    time.sleep(dt)
```

Bild von Disk laden
(hier von TJ-Distribution, siehe Help)

Dieses Verfahren ist besser als immer
image(datei, x, y) aufrufen

Bild zeigen (im Bildbuffer)

Übungen:

1. Orientieren Sie sich über die TJ-Bilder
2. Nehmen Sie ein anderes TJ-Bild
3. Erstellen Sie ein eigenes Bild

png, transparenter Hintergrund


```
# mech3d.py
# Vertikaler Wurf ohne Luftwiderstand mit Grafik
# Mit bewegtem Image und Hintergrundbild

from gpanel import *
import time

g = 9.81 # Erdbeschleunigung (m/s^2)
dt = 0.1 # Zeitschritt (s)

# Anfangsbedingungen:
t = 0
y = 0  # Ort (m)
v = 20 # Geschwindigkeit (m/s)

makeGPanel(-6, 6, -220, 70)
enableRepaint(False)
img = getImage("sprites/alien.png") # load image
bgImg = getImage("sprites/town.jpg")
while t < 10:
    clear()
    v = v - g * dt
    y = y + v * dt
    t = t + dt
    image(bgImg, -6, -220) # show background
    image(img, 0, y) # show image
    repaint()
    time.sleep(dt)
```

Background-Bild von Disk laden
(hier von TJ-Distribution)

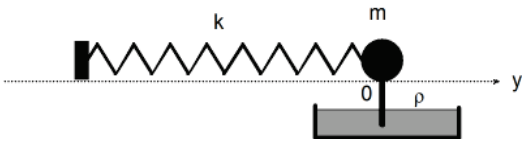
Background zeigen

Übungen:

1. Verändern Sie den Zeitschritt
2. Vergleichen Sie mit theor. Wurfhöhe

Nun vom Wissen und der Anstrengung profitieren!

Harmonische Schwingung (mit Dämpfung)




```
F = -k*y - r*v          # Kraft
a = F/m                 # Beschleunigung
v = v + a*dt            # Neue Geschwindigkeit
y = y + v*dt            # Neue Koordinate
t = t + dt              # Neue Zeit
```



```
# harmon1.py
# Gedaempfte harmonische Schwingung

from gpanel import *
import time
from math import sqrt, pi

dt = 0.01       # Zeitschritt (s)
m = 0.5         # Masse (kg)
k = 4           # Federkonstante (N/kg)
r = 0.1         # Reibungskoeffizient in N/m/s

# Anfangsbedingungen
t = 0
y = 0.8
v = 0

# Theorie (ungedämpft):
T = 2 * pi * sqrt(m / k)
print "T =", T, "s =", s

makeGPanel(-1, 11, -1.2, 1.2)
drawGrid(0, 10.0, -1.0, 1, "darkgray")

pos(t, y)  # Anfangspunkt
while t < 10:
      draw(t, y)                # Zeichne
      F = -k*y - r*v            # Kraft
      a = F/m                   # Beschleunigung
      v = v + a*dt              # Neue Geschwindigkeit
      y = y + v*dt              # Neue Koordinate
      t = t + dt                # Neue Zeit
      time.sleep(dt)
```

Zur Kontrolle/als Test

X11-Farben als String

Führen Sie Computerexperimente aus!

Übungen:

- Masse/Federkonstate ändern, Schwingungsdauer?
- Dämpfung ändern:
  - r = 0, r = 1 (noch nicht kritisch) Schwingungsdauer?
  - Kritische Dämpfung (asymptotischer Grenzfall) ? Verhalten?
- Eigene Ideen


```
# harmon2.py: Simulationsexperiment

from gpanel import *
import time

dt = 0.01     # Zeitschritt (s)
m = 0.5       # Masse (kg)
k = 4         # Federkonstante (N/kg)
r = 0.1       # Reibungskoeffizient in N/m/s

# Anfangsbedingungen
t = 0
y = 0.8
v = 0

makeGPanel(-1, 1, -1, 1)
enableRepaint(False)
img = getImage("sprites/marble.png") # load image
s = toWindowWidth(32)# Pixel radius of ball
while True:
    clear()
    F = -k*y - r*v            # Kraft
    a = F/m                   # Beschleunigung
    v = v + a*dt              # Neue Geschwindigkeit
    y = y + v*dt              # Neue Koordinate
    bgColor("yellow")
    line(-1, -s, 1, -s)
    image(img, y - s, -s)
    repaint()
    t = t + dt
    time.sleep(dt)
```

Von Hand rendern

Bild laden (aus TJ-Distribution)
Grösse 64x64 pixels

Koord bis zur linken unteren Ecke

Hintergrund

Kugelunterlage

Kugel

Rendern

Übungen:

1. Verändern Sie den Zeitschritt. Feststellungen?
2. Nehmen Sie ein anderes Bild
3. "Verschönern" Sie die Simulation nach eigenen Ideen

```
# harmon3.py: Kombination

from gpanel import *
import time

dt = 0.01      # Zeitschritt (s)
m = 0.5        # Masse (kg)
k = 4          # Federkonstante (N/kg)
r = 0.1        # Reibungskoeffizient in N/m/s

t = 0
y = 0.8
v = 0

pTim = GPanel(-1, 11, -1.2, 1.2)
drawPanelGrid(pTim, 0, 10.0, -1.0, 1.0)
pSim = GPanel(-1, 1, -1, 1)
pSim.enableRepaint(False)
img = pSim.getImage("sprites/marble.png")
s = pSim.toWindowWidth(32)
pTim.pos(t, y)
while True:
    pTim.draw(t, y)
    pSim.clear()
    F = -k*y - r*v
    a = F/m
    v = v + a*dt
    y = y + v*dt
    pSim.bgColor("yellow")
    pSim.line(-1, -s, 1, -s)
    pSim.image(img, y - s, -s)
    pSim.repaint()
    t = t + dt
    time.sleep(dt)
```

Perfekte Implementierung:
Displays sind Objekte

"Zeit"-Display "erzeugen (Konstruktor)

"pTim ist ein Objekt"

"Simulations"-Display "erzeugen (Konstruktor)

Alle Funktionen der Displays erhalten einen Präfix: das Objekt mit dem Punktoperator

SIMULA was the first object language (1970).

Radioaktives Zerfallsgesetz:

Gutes Beispiel, warum Simulationen auch Sinn machen: man vermeidet gefährliche bzw. kostspielige Experimente.

Man geht davon aus, dass die Wahrscheinlichkeit dp, dass ein bestimmtes Radionuklid von N gleichartigen Isotopen im nächsten Zeitschritt dt zerfällt, proportional zum Zeitschritt ist, also

$dp = \lambda dt$ beträgt, wo λ die Zerfallskonstante ist.

dp ist also von nichts abhängig (völlig spontan), also insbesondere auch nicht, wie "alt" das Isotop ist (und auch nicht von irgendwelchen Umweltbedingungen. Damit hatte Einstein grosse Mühe "Gott würfelt nicht" Gibt es "hidden variables?"->Grundfragen der Quantentheorie)

Ausgehend von dieser Modellannahme kann das Zerfallsgesetz mit Simulationen auf zwei Arten hergeleitet werden:

1. klassisch (Mittelwerte betrachten, durch "Ausglätten")
2. statistisch (wie es richtig wäre)

Radioaktives Zerfallsgesetz (klassisch):

Einige Grafik-Features zeigen

```
# rad1.py

from gpanel import *
import time

k = 0.05       # Decay constant (/s)
N0 = 100       # Starting number of isotops
N = N0         # Current number of isotopes
dt = 0.01      # Time interval for population check (s)
t = 0          # Current time
dp = k * dt    # Probability for decay in dt

makeGPanel(-10, 110, -10, 110)
drawGrid(0, 100, 0, 100)
title("Radioaktiver Zerfall (N versus t)")
setColor("blue")
lineWidth(2)
pos(0, N)
while t <= 100:
    dN = -N * dp
    N = N + dN
    draw(t, N)
    t += dt
#    time.sleep(dt)
```

Text in Titelzeile

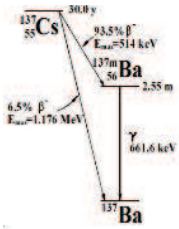
Linienfarbe

Liniendicke

Änderung der Population (Abnahme)

Neuer Wert = Alter Wert + Änderung

Cs-137/Ba-137m
Isotopengenerator



Übungen:

1. GM-Zähler: Was misst man eigentlich?
2. Könnte man auch dieses simulieren? (siehe raddemo.py)

Radioaktives Zerfallsgesetz (statistisch):

Man kann das Problem aber auch ganz anders (viel natürlicher und korrekter) angehen: Man schaut sich zur Zeit t von den N Isotopen ein Isotop nach dem anderen an und einscheidet auf Grund einer der Wahrscheinlichkeit $dp = \lambda dt$, ob es in dt zerfällt oder überlebt.

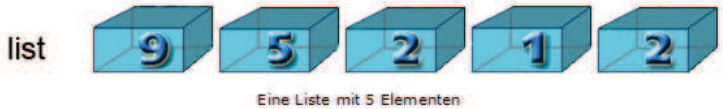
d.h. man würfelt!

Man benötigt eine geeignete Datenstruktur für die Population: **Listen**
In anderen Programmiersprachen **Arrays**
Etwas Einarbeitung nötig, z.B. mit TigerJython-Lehrmittel

3.9 LISTEN

EINFÜHRUNG

Manchmal musst du Werte speichern, die zusammen gehören, aber deren genaue Anzahl du beim Erstellen des Programms nicht kennst. Du brauchst also eine Datenstruktur, in der du mehrere Werte speichern kannst. Die Struktur sollte so flexibel sein, dass sie auch die **Reihenfolge** der hinzugefügten Werte berücksichtigt. Es ist naheliegend, eine Aneinanderreihung von einfachen Behältern dafür einzusetzen, von der du bereits gehört hast, nämlich eine **Liste**. Hier erfährst du nun ausführlich, wie man mit Listen umgeht.



Eine Liste mit 5 Elementen

Eine Liste besteht aus einzelnen Elementen, die nacheinander angeordnet sind. Im

Zwischenübung: Listen definieren und sie durchlaufen:

```
# listenuebung.py

li = [9, 5, 2, 1, 2]
print li[0], li[1], li[2], li[3], li[4]

for element in li:
    print element

print range(5)

for i in range(5):
    print li[i]
```

Eckige Klammern
Elemente von beliebigem Typ
Beliebig viele Elemente

Auf Elemente zugreifen: mit Index in eckigen Klammern
Index beginnt bei 0

Mit for-Schleife Containerstruktur durchlaufen
Gibt ein Element um das andere zurück
(ganz anders als Basic/Pascal/C/Java)

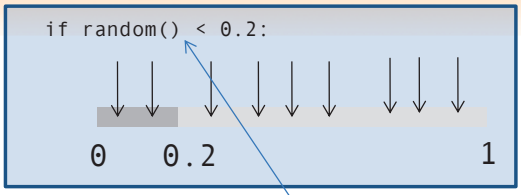
range(5) ist die Liste mit den Elemente [0, 1, 2, 3, 4]

der Index i durchläuft die Werte 0...4
entspricht dem klassischen `for (int i = 0; i < 5; i++)`
(automatischer Schleifenzähler)

Übungen:

1. Eine Liste li ist ein Objekt: es hat Methoden (mit Punktoperator)
   Element am Ende anfügen:li. append()
   Liste sortieren: li.sort()
2. Teillisten bilden mit Slice-Operation li[startIndex:endIndex]
3. Listen erzeugen mit Multiplikationszeichen: li = [0] * 5


```
# rad2.py

from gpanel import *
import time
from random import random

k = 0.05      # Decay constant (/s)
N0 = 100      # Starting number of isotops
N = N0        # Current number of isotopes
dt = 0.01     # Time interval for population check (s)
t = 0         # Current time
dp = k * dt   # Probablity for decay in dt

makeGPanel(-10, 110, -10, 110)
drawGrid(0, 100, 0, 100)
title("Radioaktiver Zerfall (N versus t)")
setColor("blue")
lineWidth(2)
population = [1] * N0  # Population als Liste, alle lebend
pos(0, N0)
while t <= 100:
    for i in range(N0):
        if population[i] == 1 and random() < dp:
            population[i] = 0
    N = population.count(1)
    draw(t, N)
    t += dt
#    time.sleep(dt)
```



Ist mit Wahrscheinlichkeit 0.2 wahr

Population als Liste mit 0 und 1

Jedes Isotop einzeln betrachten

Falls es noch am Leben ist, mit W' von dp zerfallen lassen

zerfallen lassen

Lebende zählen

Übungen:

1. Mehrmals laufen lassen->statistische Schwankungen erleben!
2. Anfangszahl auf 1000 erhöhen Schwankungen kleiner

Python/Jython und Entwicklungsumgebungen

Python ist ein Interpreter, d.h. die Programm-Source wird in einen Zwischencode (analog Java Bytecode) kompiliert, der dann zu Laufzeit interpretiert wird. Damit ergibt sich eine hohe Benutzerinteraktivität, was für kleine Programme und für die Ausbildung günstig ist. Nachteilig ist, dass bestimmte Syntaxfehler erst zu Laufzeit bemerkt werden und dass Python-Programme nicht "standalone" (ohne Python/TigerJython) laufen (kein Maschinencode).

Also:
Python ist geeignet für **Ausbildung**, schnelles **Prototyping** (Programmtests), **Web-Development**
Python nicht geeignet für: Mission Critical Systeme, z.B. Steuerung von Raumschiffen, Waffen, usw.

Python ist schon lange bekannt, hatte aber keine gute IDE (Idle). Hat sich in der letzten Jahren grundlegend geändert:
- PyCharm
- Geany
- TigerJython
- Python(x, y)
- usw.

Für die Schule am besten (persönliche Einschätzung): TigerJython oder wer Jython nicht liebt Geany.

Auch hier ist aber animierte Grafik wie das GPanel ist ein MUSS!

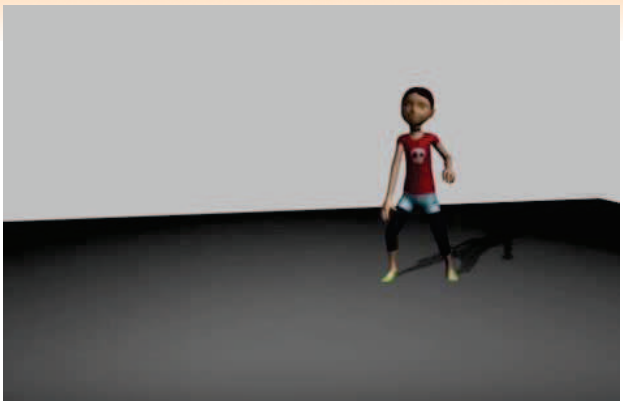
Übungen:

1. Geany + QT4/GPanel gemäss Anleitung auf http://www.python-exemplarisch.ch installieren
2. Irgendein Programm aus dem Kapitel Visualisierung ausführen (z.B. SimEx3)

Random Walk
Drunken Man's Walk

Gleichlange Schritte in zufälliger Richtung

Der Betrunkene hat trotz zufälliger Bewegungsrichtungen sehr wohl eine Chance, einmal nach Hause zu kommen. Resultat: Der Mittelwert der Distanz zum Ursprung geht mit Wurzel aus t.



Diese Gesetzmässigkeit hat kein geringerer als Albert Einstein 1905 in seiner berühmten Abhandlung *"Über die von der molekularkinetischen Theorie der Wärme geforderte Bewegung von in ruhenden Flüssigkeiten suspendierten Teilchen"* hergeleitet und damit eine theoretische Erklärung für die Brownsche Bewegung gegeben (Nobelpreis). Ein Jahr später hat M. Smoluchowski mit einer anderen Überlegung dasselbe Resultat gefunden.

Simulation sehr einfach und anschaulich

Aber hier ist Richtungsgrafik viel besser: **Turtlegrafik**. In TigerJython sehr schöne (altbewährte) Implementierung. So gut, dass viele Lehrpersonen im Informatik-Anfängerunterricht fast nur Turtlegrafik machen. Siehe Site von Tobias Kohn http://jython.tobiaskohn.ch.

Modularer Programmentwurf (allgemeines Problemlösungsverhalten):

Einteilen des komplexen Problems in einfache Teilschritte, die möglichst unabhängig voneinander gelöst werden (Teile und Herrsche)

Teilschritte -> Benannte Programmteile = Funktionen (Methoden)

Prozedurales / funktionales Programmieren = Fundamental wichtiges Programmierkonzept (wenn nicht das Wichtigste überhaupt)

Funktion ohne Parameter:

```
from gpanel import *

def redDot():
    setColor("red")
    fillCircle(0.1)

makeGPanel()
pos(0.5, 0.5)
redDot()
```

Funktionsdefinition/Parameterklammern (leer), Doppelpunkt

Funktionskörper einrücken!

Funktionsaufruf (leere Klammern nötig!)

Funktion mit Parameter:

```
from gpanel import *

def colorDot(x, y, r, color):
    pos(x, y)
    setColor(color)
    fillCircle(r)

makeGPanel()
colorDot(0.5, 0.5, 0.1, "red")
```

Funktionsdefinition/Parameterklammern mit formalen Parametern

Funktionsaufruf mit aktuellen Parameterns

Übungen:

1. Rufen Sie colorDot() mehrmals mit random-Parametern auf

Let's walk: Aber bitte strukturiert (trotz dem Wirtshausbesuch...)

1. Schritt: Turtle macht eine einzige Simulation

ein "Ride" mit vorgegebener Schrittzahl t

```
# rwalk1.py

from gturtle import *

def ride(t):
    step = 10
    repeat t:
       fd(step)
       setRandomHeading()

makeTurtle()
ride(100)
```

Turtle-Modul importieren

Funktionsdefinition
(mit formalem Parameter, Platzhalter)

t mal wiederholen (nur TigerJython, sonst while verwenden)

TigerJython Turtle kennt viele nützliche Funkionen

Erzeugt Turtlefenster

Funktionsaufruf mit aktuellem Parameter
"100ter Ride"

Übungen:

1. Mehrmals laufen lassen
2. Turtle vertecken, um sie schneller zu machen
3. Schrittzahl des Rides erhöhen

2. Schritt: Das Hauptprogramm verwendet ride() mehrmals, um den Mittelwert mit vorgegebenen t zu bestimmen

```
# rwalk2.py
# Mean distance for 100 trials

from gturtle import *

def ride(t):
    step = 10
    repeat t:
       fd(step)
       setRandomHeading()
    return distance(0, 0)

makeTurtle()
hideTurtle()

sum = 0
repeat 100:
    home()
    d = ride(400)
    sum += d
print sum / 100
```

Bei jedem Ride die Distanz zum Wirtshaus zurück geben

Benötigen Summenvariable für den Mittelwert

Machen 100 Mal einen 400ter Ride

Starten immer wieder in Beiz

Führen Ride aus

Distanzen für jeden Ride aufsummieren

Mittelwert für einen 400ter Ride

Übungen:

1. Vergleichen Sie die Distanz für folgende Rides: 100ter, 400ter, 900ter
2. Erkennen Sie das Einsteinsche Wurzelgesetz?
3. Schreiben Sie ein Programm, das auch noch d in Funktion von t in einem GPanel aufzeichnet

Plüss'sches Arbeits- und Lebensprinzip:

Ein Programm muss nicht nur korrekt laufen, sondern auch schön geschrieben sein!

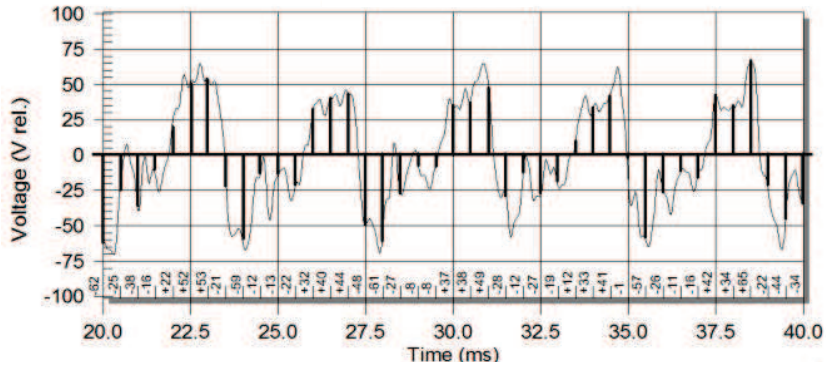
(damit man Freude am Programmieren hat).

Have a look under the hood!

Programmieren hat viel von einer Lebenskunst, es braucht dabei Phantasie, Intelligenz und Ästhetik, aber auch Entscheidungsfreude, Durchhaltewillen und das Einhalten selbstauferlegter Prinzipen und Grenzen (Moral der Geschichte).

Akustik, mein Lieblingsgebiet

- Starke Bezüge zum täglichen Leben,"Betroffenheit" der Schülerinnen und Schüler
- Fächerübergreifend (Musik, Biologie, Physiologie)
- Physikalisch exemplarisch relevant: Wellen, Resonanz, FFT
- Der Comuter als Experimentier-Werkzeug. Typisch Overtone (www.aplu.ch/overtone)
- Grundlage: Digitalisierung von Sound



Zeit	20.0	20.5	21.0	21.5	22.0	22.5	23.0	23.5	24.0	24.5	25.0	25.5	26.0
Wert	-62	-25	-38	-16	+22	+52	+53	-21	-59	-12	-13	-22	...

Datenstruktur: die Funktionswerte zu den Zeiten t, t + dt, t + 2dt, ... in einer Liste

```
samples = [-62, -25, -38, -16, 22, 52, 53, -21, -59, -12, -13, -11, ...]
```


Function Player

```
# akustik1.py
# Erzeugen eines Sinustons

from soundsystem import *
from math import sin, pi

def func(t):
    return A * sin(2 * pi * freq * t)

def createSamples():
    samples = [0] * 200000
    dt = 1 / fs # Sampling interval
    for n in range(len(samples)):
        t = n * dt
        samples[n] = int(func(t))
    return samples

A = 30000
freq = 500 # Hz
fs = 20000 # sampling frequency (Hz)
samples = createSamples()
openMonoPlayer(samples, 20000)
play()
```

Modul sound importieren

Funktion, die man abspielen möchte

Liste mit den digitalen Abtastwerten
(Integers im Bereich -32768...32367, 16 bit)
20'000 sample/s -> 10 Sekunden Spielzeit)

Liste füllen

auf int casten! -> Datentypen sind wichtig!

Globale Werte sind in Funktionen lesend verfügbar

"Platte auflegen" (Tourenzahl)

Abspielen

Übungen:

1. Verändern Sie die Frequenz
2. Verändern Sie die Amplitude

Flexibilität einer höheren Programmiersprache: die Welt ist offen!

```
# akustik2.py
# Schwebung

from soundsystem import *
from math import sin, pi

def func(t):
    y1 = A * sin(2 * pi * freq1 * t)
    y2 = A * sin(2 * pi * freq2 * t)
    return y1 + y2

def createSamples():
    samples = [0] * 200000
    dt = 1 / fs
    for n in range(len(samples)):
        t = n * dt
        samples[n] = int(func(t))
    return samples

A = 15000
freq1 = 500 # Hz
freq2 = 501 # Hz
fs = 20000 # sampling frequency (Hz)
samples = createSamples()
openMonoPlayer(samples, 20000)
play()
```

Funktion anpassen

Teilamplituden etwas verkleinern

Frequenzdifferenz 1Hz

Übungen:

1. Untersuchen Sie die Schwebungsfrequenz in Abhängigkeit der beiden Frequenzen


```
# akustik3.py
# Schwebung mit grafischer Darstellung

from gpanel import *
from soundsystem import *
from math import sin, pi

def showSamples():
    makeGPanel(0, len(samples), -33000, 33000)
    for n in range(len(samples)):
        draw(n, samples[n])

def func(t):
    y1 = A * sin(2 * pi * freq1 * t)
    y2 = A * sin(2 * pi * freq2 * t)
    return y1 + y2

def createSamples():
    samples = [0] * 200000 # length of sample
(5s)
    dt = 1 / fs # Sampling interval
    for n in range(len(samples)):
        t = n * dt
        samples[n] = int(func(t))
    return samples

A = 15000
freq1 = 500 # Hz
freq2 = 501 # Hz
fs = 20000 # sampling frequency (Hz)
samples = createSamples()
openMonoPlayer(samples, 20000)
play()
showSamples()
```

Übungen:

1. Wählen Sie als zweite Frequenz 500.5 Hz, 502 Hz, u.a. und beobachten Sie die Schwebungsfrequenz

```
# akustik4.py
# Fourier-Synthese (Saegezahn)

from gpanel import *
from soundsystem import *
from math import sin, pi

def showSlice():
    makeGPanel(-10, 100, -36000, 36000)
    drawGrid(0, 80, -30000, 30000, "darkgray")
    for n in range(81):
        pos(n, samples[n]) if n == 0 else draw(n, samples[n])

def func(t):
    y = 0
    for n in range(1, 11):
        y += A / n * sin(2 * pi * freq * n * t)
    return y

def createSamples():
    samples = [0] * 200000 # length of sample (10s)
    dt = 1 / fs # Sampling interval
    for n in range(len(samples)):
        t = n * dt
        samples[n] = int(func(t))
    return samples

A = 15000
freq = 500 # Hz
fs = 20000 # sampling frequency (Hz) (40 values / period)
samples = createSamples()
showSlice()
openMonoPlayer(samples, 20000)
play()
```

Nur einen kleinen Ausschnitt (slice) zeigen

statt if...else.
Lustige Python-Syntax

1/n Amplitude

Grundton und 9 Obertöne

(alle in Phase)

Übungen:

1. Nur ungerade Obertöne nehmen->Rechteck
2. Phasen in Obertönen einführen->Hört man sie?

Die Welt ist offen für Ihre eigenen Untersuchungen....


```
# akustik5.py
# Tonleitern

r = 2**(1/12)
a = 440

scale_temp = [a,  a * r**2, a * r**4, a * r**5, a * r**7, a * r**9, a * r**11, 2*a]
scale_pure = [a, a * 9/8, a * 5/4, a * 4/3,  a * 3/2,  a * 5/3, a * 15/8, 2 * a]

playTone(scale_temp, 1000)
playTone(scale_pure, 1000)

playTone(scale_temp, 1000, block = False)
playTone(scale_pure, 1000)
```

Wohltemperierte Stimmung: Oktave in 12 Halbtöne mit gleichem Frequenzverhältnis r einteilen, d.h.

$$r^{12} = 2 -> r = \sqrt[12]{2}$$

Kammerton

Blockierende Funktion
(zentraler Begriff)

Nicht blockierende Funktion
(benannter Parameter versus positional er Parameter)

Werden (quasi) zusammen abgespielt

Tonsequenzen als Listen(weitere Features, siehe API-Doc)

Übungen:

1. Spielen Sie nur einen bestimmten Ton gleichzeitig während 10 s mit der wohltemperierten und natürlichen Tonleiter ab
2. Implementieren Sie die pythagoräische Tonleiter und und vergleichen Sie sie mit der natürlichen

Datenakquisition/Datenspeicherung

Persistenz von Daten: Sie überleben die Programmdauer

```
# writedata.py

from math import exp, cos, pi

def q(t):
    return A * exp(-r*t) * cos(2 * pi * f * t)

A = 5
r = 0.3
f = 0.5
dt = 0.01

with open("mydata", 'w') as fOut:
    t = 0
    while t <= 10:
        y = q(t)
        fOut.write(str(y) + "\n")
        t += dt
print "Done"
```

Datenakquisition (simuliert)

Amplitude

Dämpfung

Frequenz

Zeitschritt

Datei zum Schreiben öffnen (Textdatei!)

File-Variable

Messwert

Als String zeilenweise hineinschreiben

Datei wird am Ende des with-Blocks automatisch von Python geschlossen

Übungen:

1. Datei *mydata* im Verzeichnis des Programms mit irgendeinem Texteditor anschauen

Daten wiederverwenden/Datenauswertung

```
# readdata.py

from gpanel import *

dt = 0.01
makeGPanel(-1, 11, -6, 6)
drawGrid(0, 10, -5.0, 5.0, "darkgray")
lineWidth(2)

with open("mydata") as fIn:
    t = 0
    for line in fIn:
        draw(t, float(line))
        t += dt
```

Datei zum Lesen öffnen

File-Variable

zeilenweise lesen

in Float umwandeln (Zeilenumbruch \n wird ignoriert) und darstellen

Datei wird am Ende des with-Blocks automatisch von Python geschlossen

Übungen:

1. Setzen Sie den Anfangspunkt richtig:
   `pos(t, float(line)) if t == 0 else draw(t, float(line))`

Datenspeicherung in Cloud/auf Webserver

Statt lokale Datei, Speicherung auf Cloud (z.B. Dropbox). Ist kein Problem, falls lokales Verzeichnis syncrhonisiert wird.

Aber sehr elegant:Datenbank eines Webservers (allgemein zugänglich über Web-Browser)

insert.php ist verantwortlich für die Verbindung zwischen HTTP-Request und Datenbank

```
<?php

$con= mysqli_connect('fdb12.awardspace.net', '2188329_raspi', 'mypassword',
'2188329_raspi', '3306');

$table = $_GET["table"];
$x = $_GET["x"];
$y = $_GET["y"];
$sql ="INSERT INTO $table (x, y) VALUES ('$x', '$y')";
mysqli_query($con, $sql);
mysqli_close($con);
?>
```

Authetifizierung

Request-Parameter extrahieren

SQL-Kommando

Gratis Webserver: siehe http://www.python-exemplarisch.ch/rpi

unter Web-Hosts

Zusammenarbeit Physik-Informatik


```
# writeweb.py

import socket
import time
from math import cos, exp

host = "www.aplu.dx.am"
port = 80
dx = 5
table = "dataNN"

def f(x):
    return 0.5 + 0.5 * exp(-0.01 * x) * cos(0.1 * x)

x = 0
while x <= 100:
    y = f(x)
    print x, y
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host , port))
    request = "GET /insert.php?table=" + table + "&x=" + str(x) + "&y=" + str(y) + \
              " HTTP/1.1\r\nHost: " + host + "\r\n\r\n"
    s.send(request)
    s.shutdown(1)
    s.close()
    x += dx
    time.sleep(dx)
print "Done"
```

Übungen:

1. Erzeugen Sie die Werte neu / etwas anders
2. Zeigen Sie die Tabellenwerte als Grafik während der Aufnahme im Browser an
3. Zeigen Sie Tabellenwerte einer anderen Person an

Wählen Sie die Ihnen zugeteilte Tabelle aus!!!

Datensimulation: gedämpfte Schwingung

Datenakquisition alle 5 s

HTTP-Get Request der Datei insert.php mit den Parametern: table, x und y

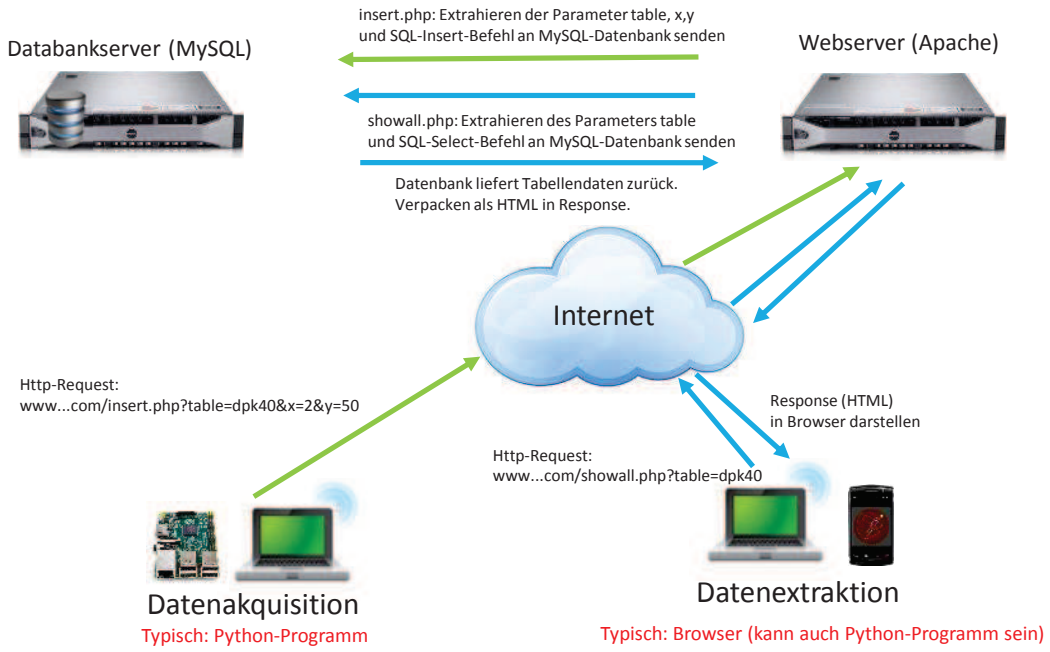
Verbindung nach jedem Request schliessen (stateless)

http://www.aplu.dx.am/showall.php?table=dataNN Tabellenwerte als Text anzeigen

http://www.aplu.dx.am/showgraph.php?table=dataNN Tabellenwerte als Grafik anzeigen (automatischer Refresh)

http://www.aplu.dx.am/deleteall.php?table=dataNN Tabelleninhalt löschen

Prinzip der Datenspeicherung auf einem Webserver



Goodies/Highlights

Keplersche Gesetze: kepler.py "Herausfinden/Nachweisen" mit Computersimulation

Echte 3-Körper-Probleme

Comet-capture	Swing-by	Sun-Earth-Moon	Double-star (instable planet)	Double-star (stable planet)
astro1.py	astro2.py	astro3.py	astro4.py	astro5.py

Kritisch, keine Echtzeitsimulation, sehr sensibel abhängig von Anfangsbedingungen (Quelle: Astronomisches Institut der Uni Bern)

Entropie anschaulich, Mass für die Unordnung: entropy.py

Wieviele Fragen brauchen wir, um den Mikrozustand zu erfahren: S = ld(W)
Wir wissen, wie viele, aber nicht genau welche Teilchen links bzw. rechts sind
Siehe Programmierkonzepte, Kapitel "Information & Ordnung"

Fraktale: Mit wenig Aufwand die Natur nachbilden?

Mandelbrot: mandelbrot.py
Farn: fern.py

Freude an Ästhetik/Kunst!

Smartphone/Tablet im Physikunterricht?

Sensoren verwenden->Tag für Physik im Unterricht (ETH 2016, iMobile Physics)

Android-App: RemoteSensor.apk http://www.aplu.ch/home/apluhomex.jsp?site=137
PC-Datenlogger: remotesensorclient.py

Mit Programmieren geht alles viel einfacher...