Visualisierung rekursiver Datenstrukturen mit der Turtle

- Inhalte des Lehrplans in Informatik der Q11:
  11.1 Rekursive Datenstrukturen
  - 11.1.1 Listen (29h)
  - 11.1.2 Bäume als spezielle Graphen (29h)

    Im Rahmen praktischer Fragestellungen, z. B. zur Planung von Verkehrsrouten, wenden die Schüler auch die Datenstruktur Graph als Erweiterung der Struktur Baum an.

    - ...
    - die Datenstruktur Graph als Verallgemeinerung des Baums; Eigenschaften (gerichtet/ungerichtet, bewertet/unbewertet); Adjazenzmatrix

ubuntu

Warum die *Turtle* einsetzen?

- Lern-/Lehrende verlieren schnell die Motivation, wenn nach der *Liste* auch beim *Baum* wieder „nur“ textuelle Ausgaben erfolgen
- übliche Grafikausgaben (mit GUIs wie Tkinter, Swing, wxwindows, …) erfordern VIEL ZU VIEL

  Implementierungsaufwand - nicht gerechtfertigt
- Turtle ist schnell programmierbar, einfach integrierbar, liefert native Animationen und … macht Spaß!

Besonderheiten der Python-IDLE

- Falls direkt auf der Konsole gestartet: mainloop() schützt Turtle-Fenster vor Schluss
- Modifikation/Erstellung einer turtle.cfg im Arbeitsverzeichnis erlaubt Voreinstellungen: z.B. using_IDLE = True
- `turtle.exitonclick()`

  Bind bye() method to mouse clicks on the Screen.

  If the value "using_IDLE" in the configuration dictionary is `False` (default value), also enter mainloop. Remark: If IDLE with the `-n` switch (no subprocess) is used, this value should be set to `True` in `turtle.cfg`. In this case IDLE's own mainloop is active also for the client script.

Kurzreferenz 1v3

Turtle-Aktionen		
forward(px)	fd()	bewegt T. um *px* Pixel nach vorne
backward(px)	bk(), back()	bewegt T. um *px* Pixel nach hinten
right(winkel)	rt()	dreht T. um *winkel* im Uhrzeigersinn
left(winkel)	lt()	dreht T. um *winkel* gegen den Uhrzeigersinn
setposition(x,y)	setpos(), goto()	positioniert T. auf x- und y-Koordinaten
setx(x), sety(y)		positioniert T. auf x- oder y-Koordinate
setheading(winkel)	seth()	setzt absolute Blickrichtung der T. auf *winkel*: 0=Osten, 90=Norden
home()		positioniert T. auf Startkoordinaten
circle(radius, winkel, eckenzahl)		zeichnet Kreis gemäß *radius*; falls *winkel* entspr. Kreisbogen; falls *eckenzahl* entspr. Polygon
dot(größe, farbe)		zeichnet Punkt ggf. mit entspr. *größe* und *farbe*
stamp()		zeichnet T.-Abbild an dieser Position
clearstamps(n=None)		löscht (die letzten *n*) T.-Abbilder
speed(zahl)		steuert Animationsgeschwindigkeit gemäß zahl: 1-10 od. 0 (keine A.)
delay(millisec)		verzögert die Animation in *millisec* Wartezeit zw. Fensterupdates
undo()		macht letzte T.anweisung rückgängig
position()	pos()	gibt Position als Koordinatentupel zurück
towards(x,y)		berechnet Winkel zwischen Turtle-Orientierung und Punkt (x|y)
xcor(), ycor()		gibt x- bzw. y-Koordinate zurück
heading()		gibt T.orientierung zurück; 0=Osten, 90=Norden, 180=Westen
distance(x,y)		berechnet Abstand der T. zum Punkt P(x|y)

ubuntu

Kurzreferenz 2v3

Kontrolle des Stifts		
pendown()	down(), pd()	bringt S. aufs Papier
penup()	up(),pu()	nimmt S. vom Papier
pensize(breite)	width()	setzt Stiftbreite auf Pixelbreite
pen()		gibt Stiftstatus als Dictionary zurück
isdown()		gibt True zurück, falls Stift schreibbereit
write(text, move, align, font)		gibt *text* entspr. *align* ='left' mittels *font* = ('Arial',8,'normal') aus; *move*=False: T. unbewegt
pencolor(), fillcolor()		gibt Strich-/Füllfarbe als Namen oder RGB-Tupel zurück
pen-/fillcolor(farbname)		setzt Strich-/Füllfarbe gemäß *farbname*
pen-/fillcolor((r,g,b))		setzt Strich-/Füllfarbe gemäß RGB-Komponenten (0-1 bzw. 0-255)
color(pencolor,fillcolor)		setzt Strich- und Füllfarbe gleichzeitig
filling()		gibt True zurück, falls Füllmodus aktiviert
begin_fill(), end_fill()		Füllmodus (de-)aktivieren
Status der Turtle		
showturtle()	st()	zeige Turtle
hideturtle()	ht()	verstecke Turtle
isvisible()		gibt True oder False zurück
shape(name=None)		*name*-Optionen: “arrow”, “turtle”, “circle”, “square”, “triangle”, “classic”
getshapes()		gibt mögliche T.formen als Zeichenketten zurück (s.o.)
onclick(), onrelease(), ondrag()		Mouse-Events
resizemode(), shape-/turtlesize(), shearfactor(), settiltangle(), tiltangle(), tilt(), shapetransform(), get_shapepoly()		ergänzende Methoden für die 'individuelle' Turtle-Gestaltung
Spezielle Turtle-Methoden		
begin_poly(); end_poly(), get_poly(), clone(), getturtle()/-pen(), getscreen(), setundobuffer(), undobufferentries()		

Kurzreferenz 3v3

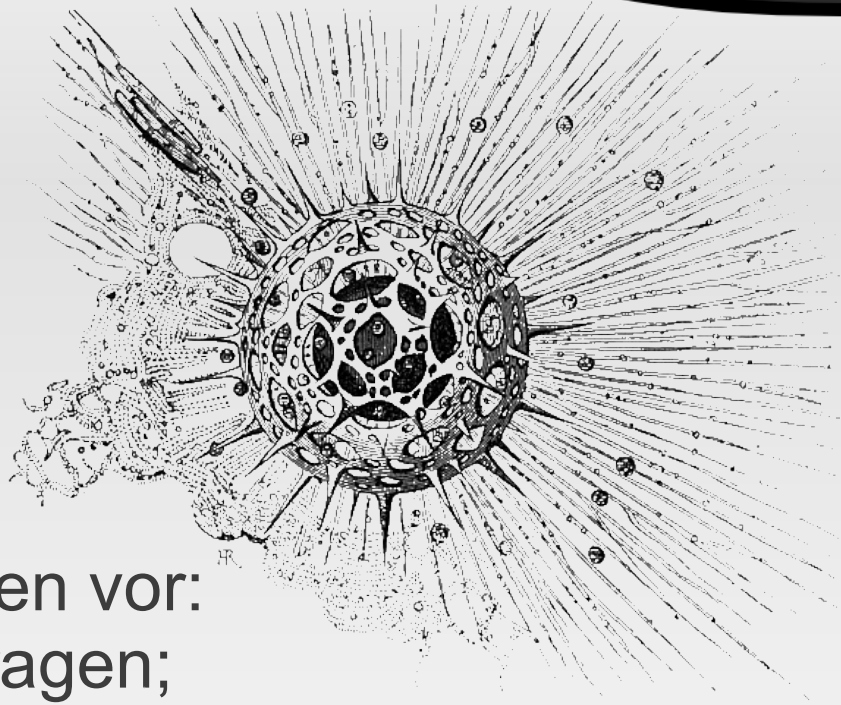
Turtle-Fenster		
bgcolor()		setzt Hintergrundfarbe; Standard = "white"
bgpic(picname)		setzt/liest Hintergrundbild
clearscreen()	clear()	löscht Gezeichnetes
resetscreen()	reset()	löscht Gezeichnetes und versetzt Turtle in Ursprungszustand
screensize(x,y,bgcolor)		setzt Fensterbreite x und -höhe y; auch Angabe von bgcolor möglich
setworldcoordinates(lux,luy,rox,roy)		setzt Koordinatensystem neu: lu=links unten; ro=rechts oben
title(„text“)		setzt *text* als Fenstertitel
window_width(), window_height()		gibt Fensterbreite bzw. -höhe zurück
textinput(„Fenstertitel“, Texttitel“)		öffnet Dialogfenster mit *Fenstertitel* und Eingabefeld *Texttitel*; gibt eingegebene Zeichenkette zurück
numinput(„Fenstertitel“, „Texttitel“, default=None, minval=None, maxval=None)		wie oben, nimmt jedoch Zahlenwerte entgegen und kann das Interval (*minval – maxval*) kontrollieren sowie einen *default*-Wert vorbelegen

ubuntu

Turtle ausprobieren... ...im Interpreter-Modus

- Starten Sie die IDE/den Editor IDLE auf der Konsole:
  `python3.2 /usr/lib/python3.2/idlelib/idle.py -n`
- Interpreter-Modus: importieren Sie alles aus dem turtle-Modul:
  `>>> from turtle import *`
- Spielen Sie ein wenig mit den Methoden der Turtle herum: mit der ersten ausgeführten Anweisung öffnet sich ein Turtle-Fenster

Turtle ausprobieren … im Editor-Modus

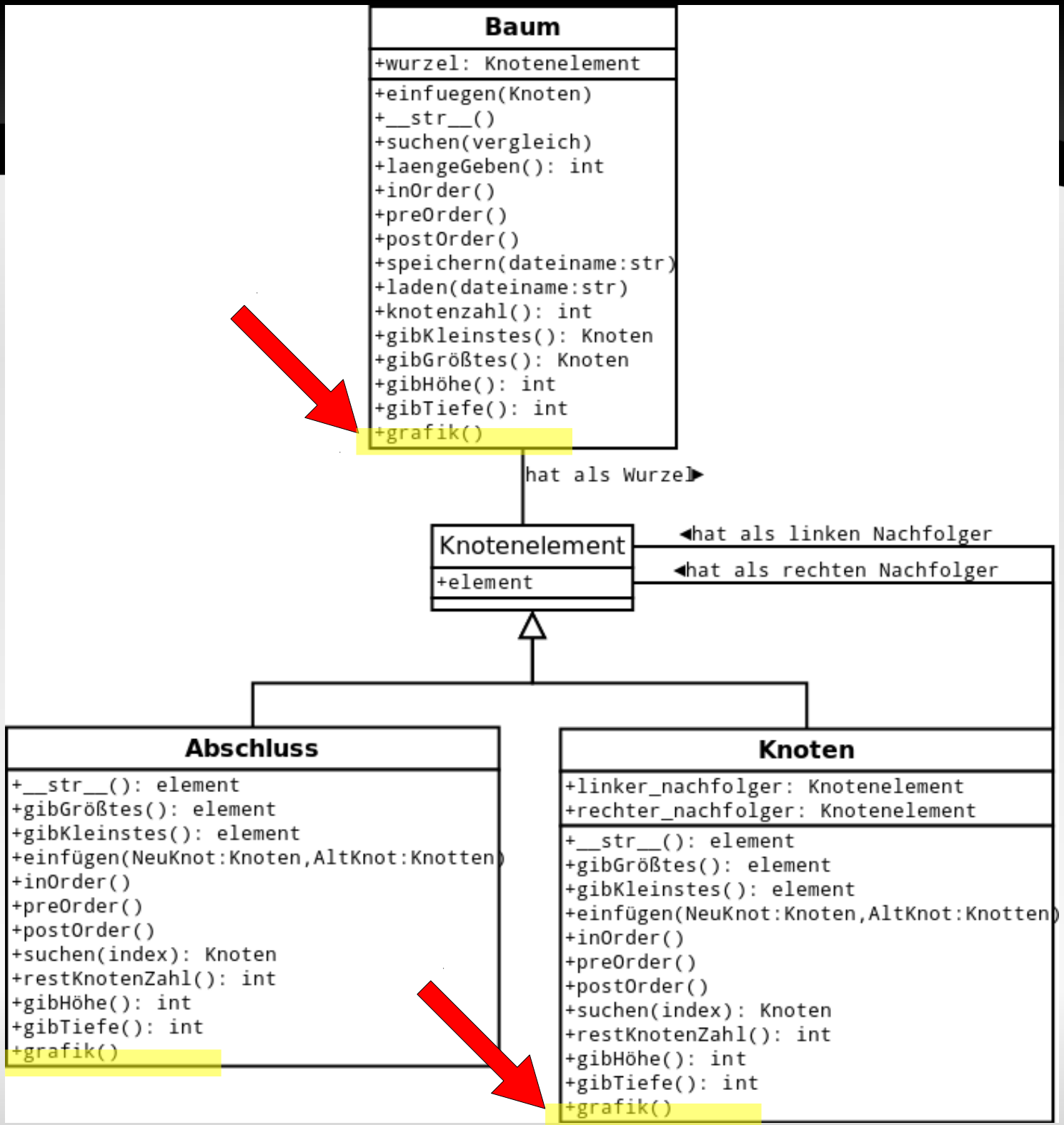


Bitte öffnen Sie das Skript
„turtle_test.py“ in IDLE
und starten es mit „F5“;

Nehmen Sie bitte folgende Manipulationen vor:
- Strich- und Füllfarbe vom Benutzer erfragen;
- Stiftbreite gegen Ende verdoppeln;
- Introspektion: am Ende der Animation die Turtle-Attribute in der rechten oberen Ecke ausgeben

Das Skript farn.py zeigt sehr schön den rekursiven Ansatz bei Abbildung einer farnähnliche Pflanze – spielen Sie mit „Weg“!

Der BinBaum: Composite-Pattern



ubuntu

BinBaum Quellcode 1v3 „Baum“

```
def grafik(self,farbe="black"):
    "Baum mittels Turtle-Grafik visualisieren"
    reset()                              # Zeichenfenster + Turtle auf Anfang
    title("Binärbaum 'Max&Moritz'")      # Benenne Turtle-Fenster
    shape("classic")                     # klassisches Turtle-Symbol
    pencolor(farbe)                      # Stiftfarbe
    pensize(2)                           # Stiftdicke
    speed(4)                             # Geschwindigkeit auf "mittelschnell"
    delay(20)                            # Verzögerung festlegen
    right(90)                            # Orientierung Turtle nach unten
    self.wurzel.grafik(200)              # Grafik-Funktion auf Wurzelknoten aufrufen
    ht()                                 # verstecken
```

ubuntu

BinBaum Quellcode 2v3 „Knoten“

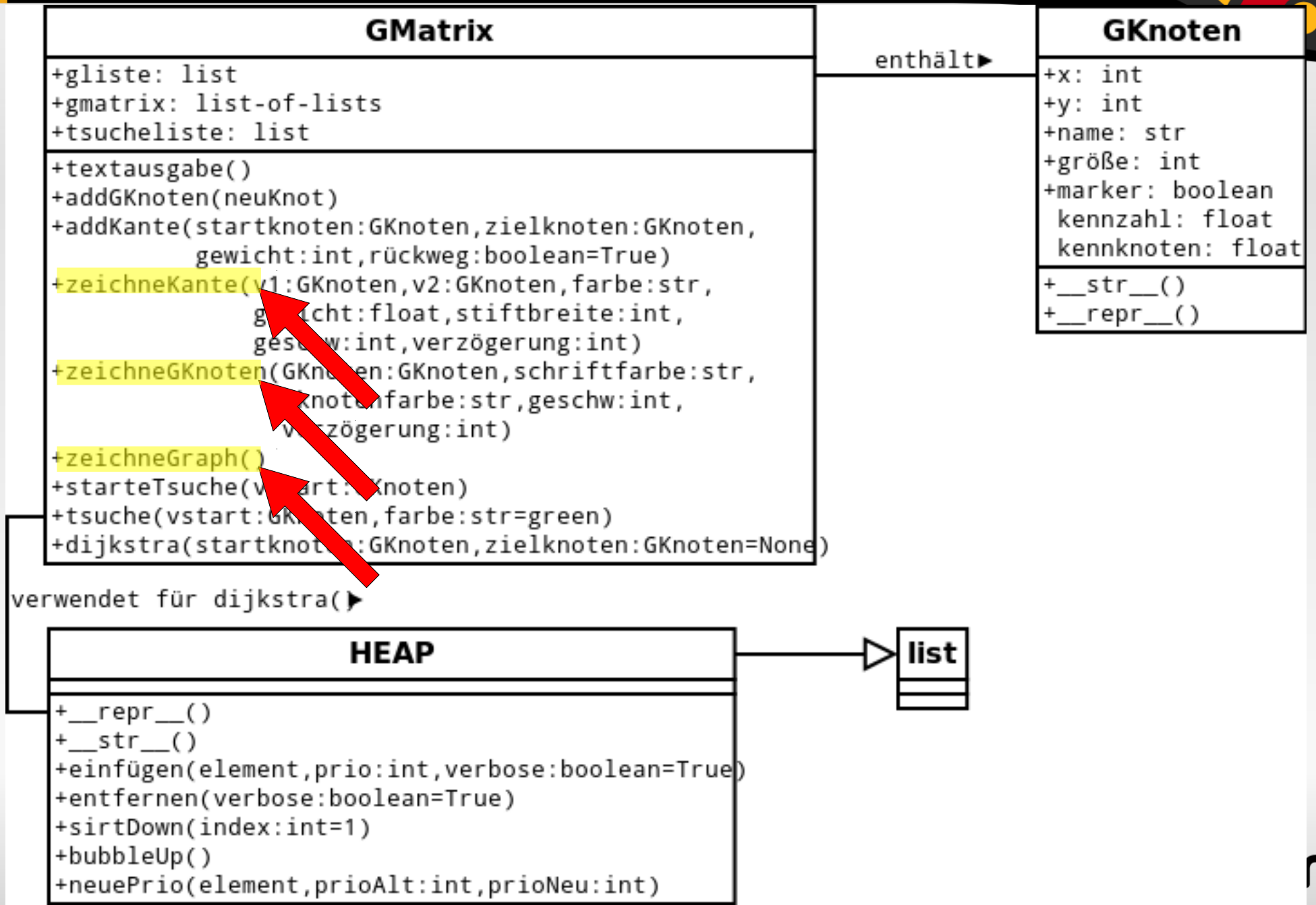
```
def grafik(self,schrittweite,farbeLinks="red", farbeRechts="blue"):
    "gibt Knoten als Turtle-Grafik aus"
    pencolor("black")                          # Stiftfarbe
    write(self.element,align='left',
          font=('Arial',12,'bold'))            # Ausgabe Element + Schriftformatierung
    pencolor(farbeLinks)                       # Stiftfarbe 'rot' für linken Ast
    ort = pos()                                # Ortskoordinaten zwischenspeichern
    right(45)                                  # Rechtsschwenk für l. Ast
    fd(schrittweite)                           # l. Ast auslaufen
    left(45)                                   # Linksschwenk in Senkrechte
    self.lnach.grafik(schrittweite*0.5)        # lnach zeichnen...
    penup()                                    # Stift hoch
    setpos(ort)                                # Ursprungsort aufsuchen
    pendown()                                  # Stift runter
    pencolor(farbeRechts)                      # Stiftfarbe 'blau' für r. Ast
    left(45)                                   # Turtle Linksschwenk f. r. Ast
    fd(schrittweite)                           # r. Ast auslaufen
    right(45)                                  # Rechtsschwenk in Senkrechte
    self.rnach.grafik(schrittweite*0.5)        # rnach zeichnen...
    setpos(ort)                                # Ursprungsort aufsuchen
```

BinBaum Quellcode 3v3 „Abschluss“

```
def grafik(self,schrittweite):
    "löscht den Ast (die 3 letzten turtle-Befehle) zu diesem Abschluss"
    undo();undo();undo()
```

ubuntu

UML: Graph mit Adjazenzmatrix



Adjazenzmatrix: Liste-in-Liste

```
class GKnoten(object):
    "Modelliert Graph-Knoten"
    def __init__(self,x, y, name, größe, marker):
        "Konstruktor"
        self.x = x                          # x-Koordinate
        self.y = y                          # y-Koordinate
        self.name = name                    # Knotenname
        self.größe = größe                  # Knotengröße
        self.marker = marker                # Markierungsattribut

    def __str__(self):
        return self.name                    # Ausgabe Konsole

    def __repr__(self):
        return self.name                    # Ausgabe via 'print()'

class GMatrix(object):
    "Adjazenzmatrix"

    def __init__(self, x,y,name, größe=1, marker=None):
        "Konstruktor"
        self.gliste = []                                # Liste der GKnoten
        self.gliste.append(GKnoten(x,y,name,größe,marker))# erster GKnoten
        self.gmatrix = [[0]]                            # vorbefüllt
        self.tsucheliste = []                           # Hilfsliste für Baumdarstellung
```

Tiefensuche

```
def starteTsuche(self,vstart):                          # bereitet Graph auf Tiefensuche vor:
    "Tiefensuche mit Wurzel GKnoten"
    print("\nTiefensuche startet am GKnoten:", vstart)   # Kommentar
    namensliste=[]                                      # Liste der Gknotennamen
    for i in self.gliste:                               # iteriere über gliste...
        i.marker = False                                # demarkiere alle Knoten
        namensliste.append(i.name)                      # und füge den Namen in Liste ein
    self.tsuche(self.gliste[namensliste.index(vstart)]) # starte Tiefensuche


def tsuche(self,vstart,farbe="green"):
    "eigentliche Tiefensuche"
    vstart.marker=True                                  # GKnoten markieren
    self.zeichneGKnoten(vstart,"black","blue",1,10)     # färbe GKnoten blau

    for i in self.gliste:
        if self.gmatrix[self.gliste.index(vstart)][self.gliste.index(i)] and not i.marker:
            print(repr(vstart).ljust(10),"--->",i)      # Textausgabe
            self.zeichneKante(vstart,i,
                              vstart in self.tsucheliste and "orange" or "green"
                              ,"",2,1,10)               # Kanten zeichnen
            self.tsucheliste.append(vstart)             # fügt akt. GKnoten in Hilfsliste ein
            self.tsuche(i)                              # rekursiver Aufruf
```

ubuntu

Job für die Turtle: zeichneKante(), zeichneGKnoten()

```
def zeichneKante(self, v1,v2, farbe, gewicht, stiftbreite=1, geschw=10, verzögerung=0):
    "zeichnet Kante zwischen 'v1' und 'v2' sowie 'gewicht' mit 'farbe'"
    speed(geschw)                       # Geschwindigkeit
    delay(verzögerung)                  # Verzögerung
    st()                                # Turtle verstecken
    up()                                # Stift hoch
    setpos(v1.x,v1.y)                   # Position v1
    color(farbe)                        # Farbe setzen
    pensize(stiftbreite)                # Stiftbreite setzen
    pd()                                # Stift runter
    setpos(v2.x,v2.y)                   # Zeichnet Verbindung zu
    up()                                # Stift hoch
    ht()
    setpos(v1.x + (v2.x-v1.x)/2,
           v1.y + (v2.y-v1.y)/2)        # Streckenmitte anvisieren
    write(gewicht)                      # Kantengewicht notieren


def zeichneGKnoten(self,GKnoten, schriftfarbe, gknotenfarbe, geschw=10, verzögerung=0):
    "zeichnet GKnoten in 'gknotenfarbe' und beschriftet ihn mit 'schriftfarbe'"
    speed(geschw)                       # Geschwindigkeit
    delay(verzögerung)                  # Verzögerung
    ht()                                # verstecke Turtle
    setpos(GKnoten.x,GKnoten.y)         # steuere GKnoten-Koordinaten an
    pd()                                # Turtle runter
    dot(GKnoten.größe, gknotenfarbe)# zeichnet Punkt entspr. d. Größe
    color(schriftfarbe)                 # wechselt Farbe
    write(GKnoten.name)                 # Ortsnamen schreiben
    up()                                # Turtle wieder hoch
```

Dijkstra: Algorithmus

0. Markiere die **Startstadt** rot, weise ihr die **Kennzahl** 0 zu. Bezeichne diese als **aktuelle Stadt**.

1. Gehe aus von der **aktuellen Stadt** zu allen direkt erreichbaren **Nachbarstädten**.

... und führe das Folgende für jede **Nachbarstadt** durch:

Errechne die **Summe** aus der Kennzahl an der **aktuellen Stadt** und der Länge der Strecke dorthin.

- Ist die **Nachbarstadt** bereits rot markiert, mache nichts.
- Hat die **Nachbarstadt** keine **Kennzahl**, weise ihr die **Summe** als **Kennzahl** zu. markiere die Strecke zur **aktuellen Stadt**.
- Hat die **Nachbarstadt** eine **Kennzahl** kleiner der **Summe**, mache nichts.
- Hat die **Nachbarstadt** eine **Kennzahl** größer der **Summe**, streiche die dortige **Kennzahl** sowie die Markierung. Weise ihr danach die **Summe** als neue **Kennzahl** zu. Markiere die Strecke zur **aktuellen Stadt**.

Errechne die **Summe** aus der Kennzahl an der **aktuellen Stadt** und der Länge der Strecke dorthin.

- Ist die **Nachbarstadt** bereits rot markiert, mache nichts.
- Hat die **Nachbarstadt** keine **Kennzahl**, weise ihr die **Summe** als **Kennzahl** zu. markiere die Strecke zur **aktuellen Stadt**.
- Hat die **Nachbarstadt** eine **Kennzahl** kleiner der **Summe**, mache nichts.
- Hat die **Nachbarstadt** eine **Kennzahl** größer der **Summe**, streiche die dortige **Kennzahl** sowie die Markierung. Weise ihr danach die **Summe** als neue **Kennzahl** zu. Markiere die Strecke zur **aktuellen Stadt**.

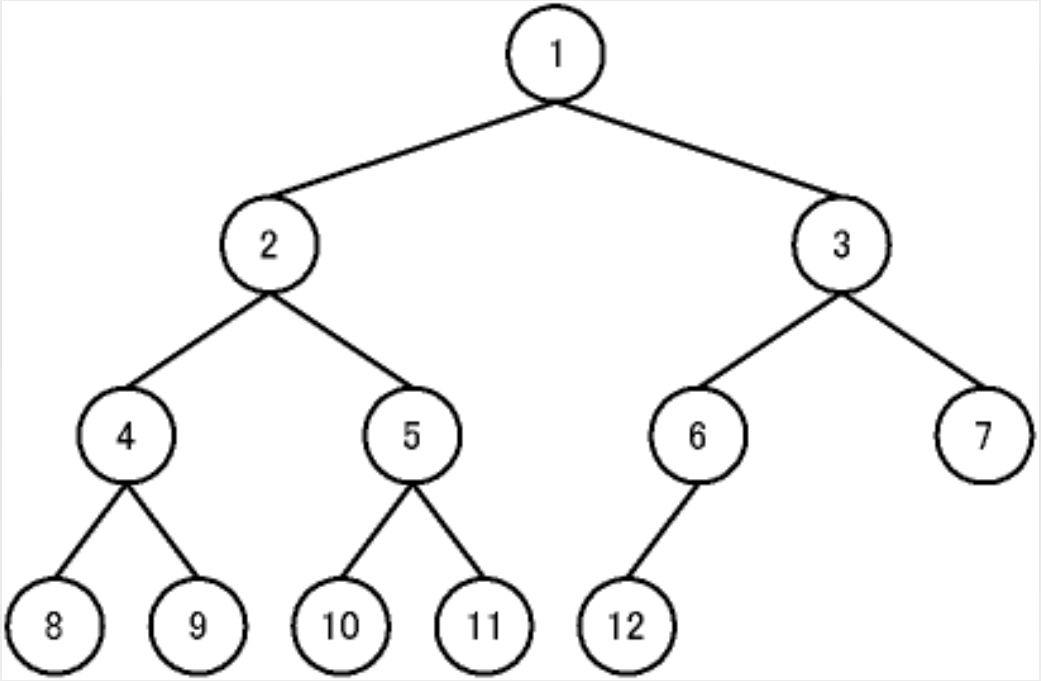
2. Betrachte alle Städte, die zwar eine **Kennzahl** haben, aber noch nicht rot markiert sind. Suche die Stadt mit der kleinsten **Kennzahl**.

3. Bezeichne diese als **aktuelle Stadt**. Weisen mehrere Städte die kleinste **Kennzahl** auf, wähle eine beliebige davon als **aktuelle Stadt**.

4. Markiere die **aktuelle Stadt** rot, zeichne die dort markierte Strecke in rot ein.

5. Falls es noch Städte gibt, die nicht rot markiert sind, weiter bei (1.)

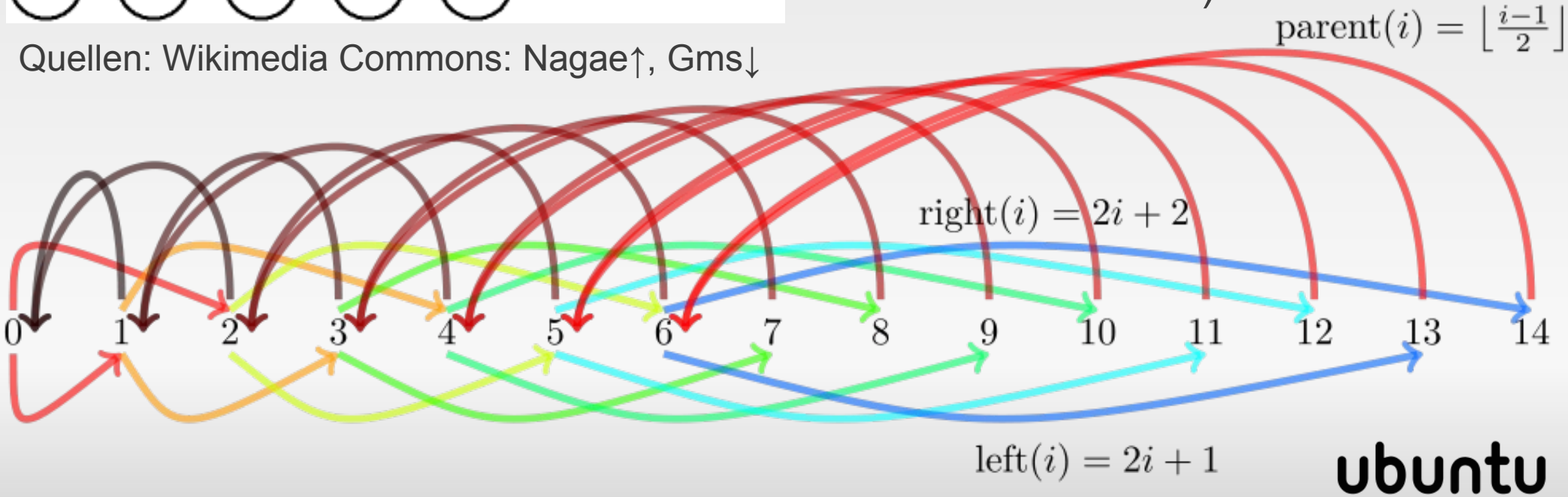
Prioritätswarteschlange: Heap



Quellen: Wikimedia Commons: Nagae↑, Gms↓

Seriell (als Array) implementiert;

Methoden:
- einfügen (immer am Ende)
- bubbleup (Platztausch mit Elter)
- entfernen (immer die Wurzel)
- sirtDown (das letzte Element auf Wurzel, P.tausch mit Kindern)



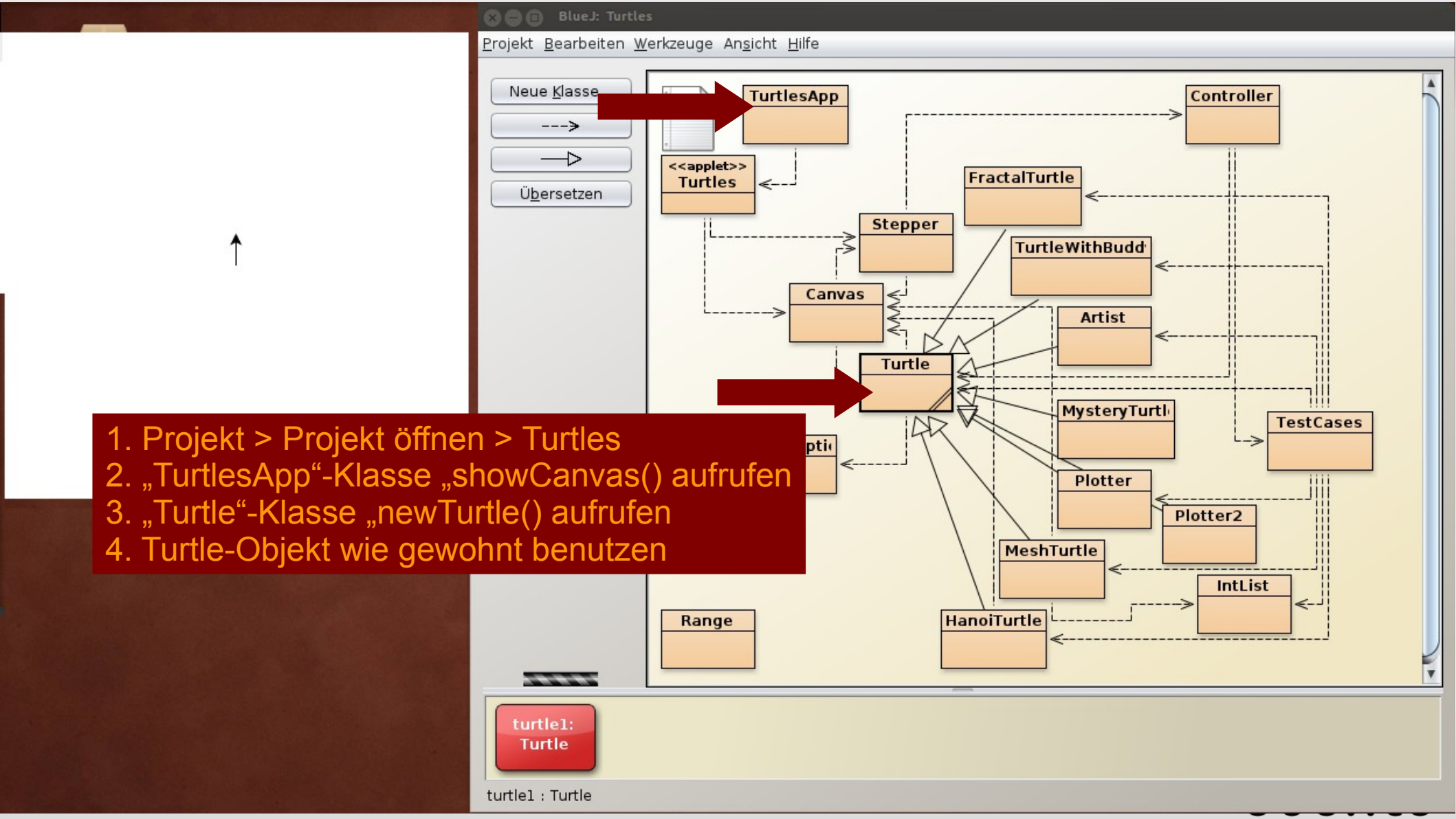
ubuntu

Tiefensuche oder Dijkstra?

- 4 Optionen (siehe Skript-Ende):
  - `textausgabe()` der Matrix auf der Konsole
  - Tiefensuche mit `starteTsuche()` an Startknoten
  - `dijkstra()` nur mit Startknoten (original) erzeugt n-Baum kürzester Entfernungen
  - `dijkstra()` modifiziert mit Start- und Zieleingabe erzeugt Pfad vom Ziel- zum Startknoten

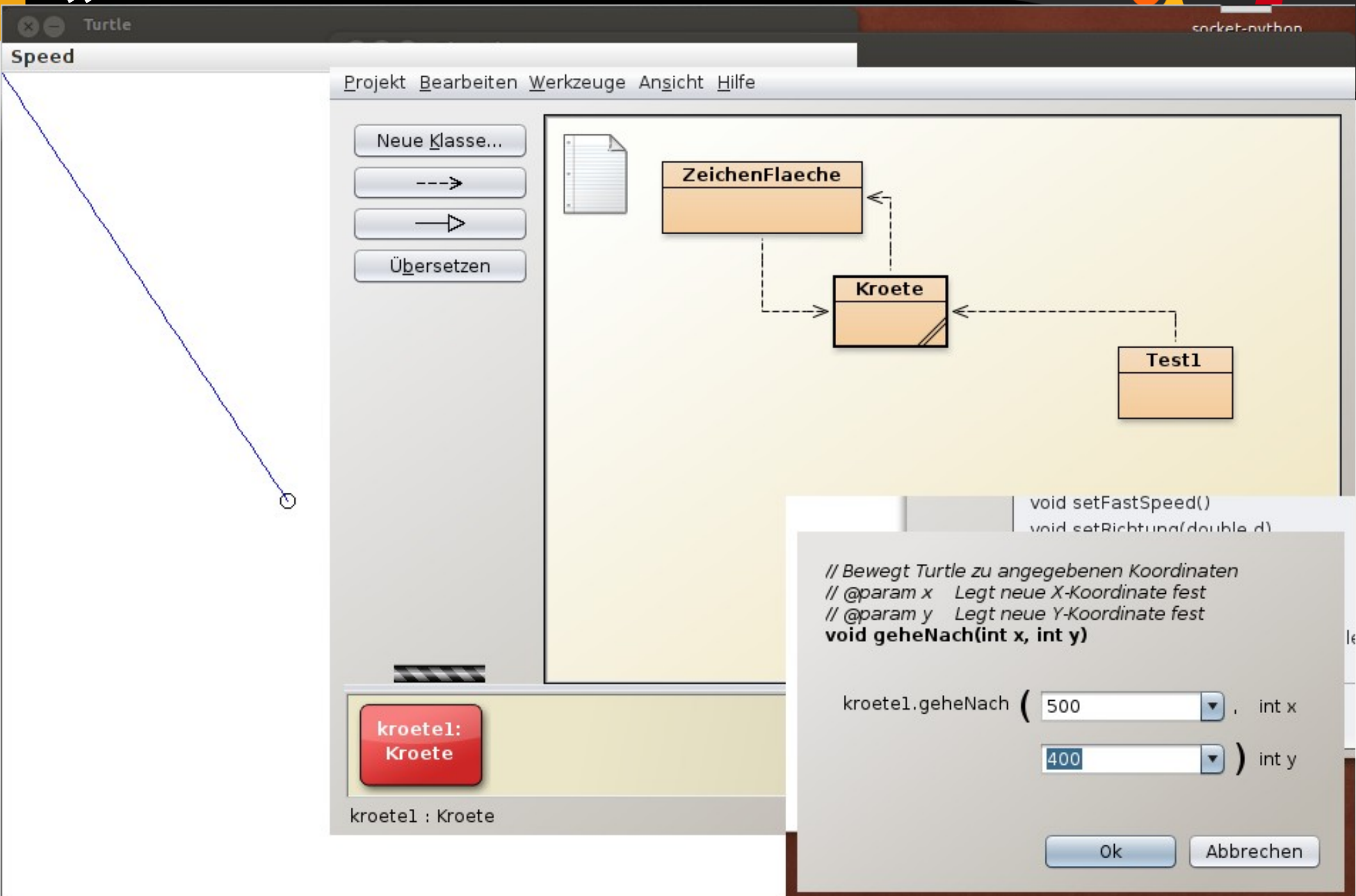
```
gm.addKante("Finning","Hagenheim",5)
gm.addKante("Mundraching","Hagenheim",9.6)

#gm.textausgabe()
gm.zeichneGraph()
#gm.starteTsuche("Burching")
gm.dijkstra("Kaufering","Greifenberg")
#gm.dijkstra("Kaufering")
exitonclick()
```

Turtle mit BlueJ



1. Projekt > Projekt öffnen > Turtles
2. „TurtlesApp“-Klasse „showCanvas() aufrufen
3. „Turtle“-Klasse „newTurtle() aufrufen
4. Turtle-Objekt wie gewohnt benutzen

„Kroete“ mit BlueJ



Versionsdifferenzen: Python 2.x und 3.x

- 2.x-Python wurde aus Kompatibilitätsgründen bis zur Version 2.7.3 weiterentwickelt; keine weiteren Verbesserungen mehr zu erwarten
- 3.x-Python ist nicht mehr abwärts kompatibel ist; für den Unterricht sind die Unterschiede jedoch relativ gering, interessant ist aber vor allem die Unicode-Unterstützung!

print „Das ist Python“, print	**print(„Das ist Python“), print ()**
type *long*	**nur noch *int***
str (8-bit)+Unicode()	***text* (Unicode)+*data* (bytes)**
1/2 = 0; 1/2. = 0	**1/2 = 0.5; 1//2 = 0**
raw_input(), input()	**input(), eval(input())**
class Klasse:	**class Klasse(object):**
Oberklasse.__init__()	**super().__init__()**

ubuntu