Vorlesung 08

2.2 Module und Klienten

Module erlauben es, Funktionen für Programme in verschiedenen Dateien zu definieren. Sie unterstützen die Idee des strukturierten Programmierens.

Klienten sind Programme, die Module benutzen.

1 Neue Struktur für ein altes Programm

Wir wollen das Programm, das eine Kugel über die Leinwand rollen lässt, durch Funktionen strukturieren.

Später werden wir es in Module aufteilen.

Die Funktionen sollen dafür sorgen, dass

- der Programmtext einfacher zu verstehen ist und
- Berechnungen, die an mehreren Stellen des Programms ausgeführt werden, nur einmal programmiert werden müssen.

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern"

```
import sys, stddraw, random

# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Das Malen wird vorbereitet.
stddraw.setCanvasSize(800,800)
stddraw.setPenColor(stddraw.RED)

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    if x_pos>=1.0 or x_pos<=0.0:  dx = -dx
    if y_pos>=1.0 or y_pos<=0.0:  dy = -dy
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(anzeigedauer)
```

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern"

```
import sys, stddraw, random

# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Das Malen wird vorbereitet.
stddraw.setCanvasSize(800,800)
stddraw.setPenColor(stddraw.RED)

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    if x_pos>=1.0 or x_pos<=0.0:  dx = -dx
    if y_pos>=1.0 or y_pos<=0.0:  dy = -dy
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(anzeigedauer)
```

```
bildAnzeigen(x_pos,y_pos)
```

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern'"

```
import sys, stddraw, random

# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Das Malen wird vorbereitet.
stddraw.setCanvasSize(800,800)
stddraw.setPenColor(stddraw.RED)
```

leinwandVorbereiten()

```
# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    if x_pos>=1.0 or x_pos<=0.0:  dx = -dx
    if y_pos>=1.0 or y_pos<=0.0:  dy = -dy
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(anzeigedauer)
```

bildAnzeigen(x_pos,y_pos)

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern'"

```
import sys, stddraw, random

def leinwandVorbereiten():
    stddraw.setCanvasSize(800,800)
    stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(40)
```

```
import sys, stddraw, random

# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Das Malen wird vorbereitet.
stddraw.setCanvasSize(800,800)
stddraw.setPenColor(stddraw.RED)
```

leinwandVorbereiten()

```
# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    if x_pos>=1.0 or x_pos<=0.0:  dx = -dx
    if y_pos>=1.0 or y_pos<=0.0:  dy = -dy
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(anzeigedauer)
```

bildAnzeigen(x_pos,y_pos)

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern"

```
import sys, stddraw, random

def leinwandVorbereiten():
    stddraw.setCanvasSize(800,800)
    stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(40)
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    if x_pos>=1.0 or x_pos<=0.0:  dx = -dx
    if y_pos>=1.0 or y_pos<=0.0:  dy = -dy
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern"

```
import sys, stddraw, random

def leinwandVorbereiten():
    stddraw.setCanvasSize(800,800)
    stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(40)
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:                    randtest(pos) liefert -1 oder 1
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    if x_pos>=1.0 or x_pos<=0.0:  dx = -dx
    if y_pos>=1.0 or y_pos<=0.0:  dy = -dy
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern"

```
import sys, stddraw, random

def leinwandVorbereiten():
    stddraw.setCanvasSize(800,800)
    stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(40)


def randTest(pos):
    if pos<=0.0 or pos>=1.0: return -1
    else: return 1
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:                    randtest(pos) liefert -1 oder 1
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    if x_pos>=1.0 or x_pos<=0.0:  dx = -dx
    if y_pos>=1.0 or y_pos<=0.0:  dy = -dy
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern"

```
import sys, stddraw, random

def leinwandVorbereiten():
   stddraw.setCanvasSize(800,800)
   stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
   stddraw.clear(stddraw.YELLOW)
   stddraw.filledCircle(x_pos, y_pos, 0.05)
   stddraw.show(40)


def randTest(pos):
   if pos<=0.0 or pos>=1.0: return -1
   else: return 1
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    dx = dx * randTest(x_pos)
    dy = dy * randTest(y_pos)
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern"

```
import sys, stddraw, random

def leinwandVorbereiten():
    stddraw.setCanvasSize(800,800)
    stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(40)


def randTest(pos):
    if pos<=0.0 or pos>=1.0: return -1
    else: return 1
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    dx = dx * randTest(x_pos)
    dy = dy * randTest(y_pos)
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

bewegeKugel(x_pos,y_pos, dx,dy)
liefert neue Position der Kugel

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern"

```
import sys, stddraw, random

def leinwandVorbereiten():
   stddraw.setCanvasSize(800,800)
   stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
   stddraw.clear(stddraw.YELLOW)
   stddraw.filledCircle(x_pos, y_pos, 0.05)
   stddraw.show(40)


def randTest(pos):
   if pos<=0.0 or pos>=1.0: return -1
   else: return 1


def bewegeKugel(x_pos,y_pos, dx,dy):
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    return x_pos, y_pos
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    dx = dx * randTest(x_pos)
    dy = dy * randTest(y_pos)
    # Neue Position der Kugel bestimmen.
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

bewegeKugel(x_pos,y_pos, dx,dy)
liefert neue Position der Kugel

Bessere Strukturierung des Kugel-Programms

Funktionen „auslagern"

```
import sys, stddraw, random

def leinwandVorbereiten():
   stddraw.setCanvasSize(800,800)
   stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
   stddraw.clear(stddraw.YELLOW)
   stddraw.filledCircle(x_pos, y_pos, 0.05)
   stddraw.show(40)


def randTest(pos):
   if pos<=0.0 or pos>=1.0: return -1
   else: return 1


def bewegeKugel(x_pos,y_pos, dx,dy):
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    return x_pos, y_pos
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    dx = dx * randTest(x_pos)
    dy = dy * randTest(y_pos)
    # Neue Position der Kugel bestimmen.
    x_pos, y_pos = bewegeKugel(x_pos,y_pos, dx,dy)
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

Bessere Strukturierung des Kugel-Programms


```
import sys, stddraw, random

def leinwandVorbereiten():
   stddraw.setCanvasSize(800,800)
   stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
   stddraw.clear(stddraw.YELLOW)
   stddraw.filledCircle(x_pos, y_pos, 0.05)
   stddraw.show(40)


def randTest(pos):
   if pos<=0.0 or pos>=1.0: return -1
   else: return 1


def bewegeKugel(x_pos,y_pos, dx,dy):
    x_pos = x_pos + dx
    y_pos = y_pos + dy
    return x_pos, y_pos
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    dx = dx * randTest(x_pos)
    dy = dy * randTest(y_pos)
    # Neue Position der Kugel bestimmen.
    x_pos, y_pos = bewegeKugel(x_pos,y_pos, dx,dy)
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

Bessere Strukturierung des Kugel-Programms


```
import sys, stddraw, random

def leinwandVorbereiten():
    stddraw.setCanvasSize(800,800)
    stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(40)


def randTest(pos):
    if pos<=0.0 or pos>=1.0: return -1
    else: return 1


def bewegeKugel(x_pos,y_pos, dx,dy):
     x_pos = x_pos + dx
     y_pos = y_pos + dy
     return x_pos, y_pos
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    dx = dx * randTest(x_pos)
    dy = dy * randTest(y_pos)
    # Neue Position der Kugel bestimmen.
    x_pos, y_pos = bewegeKugel(x_pos,y_pos, dx,dy)
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

Tupel position :

position[0] ≙ x_pos

position[1] ≙ y_pos

Bessere Strukturierung des Kugel-Programms


```
import sys, stddraw, random

def leinwandVorbereiten():
    stddraw.setCanvasSize(800,800)
    stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(x_pos, y_pos):
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(x_pos, y_pos, 0.05)
    stddraw.show(40)


def randTest(pos):
    if pos<=0.0 or pos>=1.0: return -1
    else: return 1


def bewegeKugel(x_pos,y_pos, dx,dy):
     x_pos = x_pos + dx
     y_pos = y_pos + dy
     return x_pos, y_pos
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
x_pos = random.random()
y_pos = random.random()

# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    dx = dx * randTest(x_pos)
    dy = dy * randTest(y_pos)
    # Neue Position der Kugel bestimmen.
    x_pos, y_pos = bewegeKugel(x_pos,y_pos, dx,dy)
    # Bild mit der Kugel malen.
    bildAnzeigen(x_pos,y_pos)
```

Tupel position :

position[0] ≙ x_pos
position[1] ≙ y_pos

Bessere Strukturierung des Kugel-Programms


```
import sys, stddraw, random

def leinwandVorbereiten():
   stddraw.setCanvasSize(800,800)
   stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(position):
   stddraw.clear(stddraw.YELLOW)
   stddraw.filledCircle(position[0],position[1], 0.05)
   stddraw.show(40)


def randTest(pos):
   if pos<=0.0 or pos>=1.0: return -1
   else: return 1


def bewegeKugel(position, dx,dy):
    x_pos = position[0] + dx
    y_pos = position[1] + dy
    return x_pos, y_pos
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
position = ( random.random(), random.random() )


# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    dx = dx * randTest(position[0])
    dy = dy * randTest(position[1])
    # Neue Position der Kugel bestimmen.
    position = bewegeKugel(position, dx,dy)

    # Bild mit der Kugel malen.
    bildAnzeigen(position)
```

Bessere Strukturierung des Kugel-Programms


```
import sys, stddraw, random

def leinwandVorbereiten():
    stddraw.setCanvasSize(800,800)
    stddraw.setPenColor(stddraw.RED)


def bildAnzeigen(position):
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(position[0],position[1], 0.05)
    stddraw.show(40)


def randTest(pos):
    if pos<=0.0 or pos>=1.0: return -1
    else: return 1


def bewegeKugel(position, dx,dy):
    x_pos = position[0] + dx
    y_pos = position[1] + dy
    return x_pos, y_pos
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
dx = float(sys.argv[1])
dy = float(sys.argv[2])

# Die Startposition der Kugel wird zufällig bestimmt.
position = ( random.random(), random.random() )


# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    dx = dx * randTest(position[0])
    dy = dy * randTest(position[1])
    # Neue Position der Kugel bestimmen.
    position = bewegeKugel(position, dx,dy)

    # Bild mit der Kugel malen.
    bildAnzeigen(position)
```

Tupel richtung

richtung[0] ≙ dx

richtung[1] ≙ dy

Bessere Strukturierung des Kugel-Programms


```
import sys, stddraw, random

def leinwandVorbereiten():
   stddraw.setCanvasSize(800,800)
   stddraw.setPenColor(stddraw.RED)

def bildAnzeigen(position):
   stddraw.clear(stddraw.YELLOW)
   stddraw.filledCircle(position[0],position[1], 0.05)
   stddraw.show(40)

def randTest(pos):
   if pos<=0.0 or pos>=1.0: return -1
   else: return 1

def bewegeKugel(position, richtung):
   x_pos = position[0] + richtung[0]
   y_pos = position[1] + richtung[1]
   return x_pos, y_pos

def richtungAnpassen(position, richtung):
   dx = richtung[0] * randTest(position[0])
   dy = richtung[1] * randTest(position[1])
   return dx, dy
```

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
richtung = ( float(sys.argv[1]), float(sys.argv[2]) )


# Die Startposition der Kugel wird zufällig bestimmt.
position = ( random.random(), random.random() )


# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    richtung = richtungAnpassen( position, richtung )

    # Neue Position der Kugel bestimmen.
    position = bewegeKugel(position, richtung)

    # Bild mit der Kugel malen.
    bildAnzeigen(position)
```

Bessere Strukturierung des Kugel-Programms


```
import sys, stddraw, random

def leinwandVorbereiten():
    stddraw.setCanvasSize(800,800)
    stddraw.setPenColor(stddraw.RED)

def bildAnzeigen(position):
    stddraw.clear(stddraw.YELLOW)
    stddraw.filledCircle(position[0],position[1], 0.05)
    stddraw.show(40)

def randTest(pos):
    if pos<=0.0 or pos>=1.0: return -1
    else: return 1

def bewegeKugel(position, richtung):
    x_pos = position[0] + richtung[0]
    y_pos = position[1] + richtung[1]
    return x_pos, y_pos

def richtungAnpassen(position, richtung):
    dx = richtung[0] * randTest(position[0])
    dy = richtung[1] * randTest(position[1])
    return dx, dy
```

Kommentare fehlen ;)

```
# Die Schrittweite der Kugel wird von der Kommandozeile gelesen.
richtung = ( float(sys.argv[1]), float(sys.argv[2]) )


# Die Startposition der Kugel wird zufällig bestimmt.
position = ( random.random(), random.random() )


# Die Leinwand wird vorbereitet.
leinwandVorbereiten()

# Lasse die Kugel rollen.
while True:
    # Schrittweite ändern, falls Kugel an einem Rand ist.
    richtung = richtungAnpassen( position, richtung )

    # Neue Position der Kugel bestimmen.
    position = bewegeKugel(position, richtung)

    # Bild mit der Kugel malen.
    bildAnzeigen(position)
```

2 Module und Klienten

Strukturierung eines Programms durch Module

Ein *Modul* ist eine Datei (z.B. mit dem Namen `moduldatei.py`) mit einem Python-Programm, deren Definitionen von Funktionen (und „Konstanten“) man in anderen Programmen nutzen will.

Die Anweisung `import moduldatei`

- führt die Datei `moduldatei.py` aus
- erlaubt es, die Funktionen (und Konstanten) aus `moduldatei.py` zu benutzen.
  Eine Funktion `f()`, die in `moduldatei.py` definiert ist,
  kann dann als `moduldatei.f()` im Klienten benutzt werden.

Ein Programm mit einer Import-Anweisung heißt auch *Klient* des importierten Moduls.

Um ein Modul zur Benutzung bereitzustellen,
schreibt man ein *application programming interface* (API, Anwendungsschnittstelle)
mit den Funktions-Signaturen und Beschreibungen, was die Funktionen machen.

Das Kugel-Programm im Stück und „modularisiert“

kugel.py

```
import sys, stddraw, random

def leinwandVorbereiten():
   stddraw.setCanvasSize(800,800)
   stddraw.setPenColor(stddraw.RED)

def bildAnzeigen(position):
   stddraw.clear(stddraw.YELLOW)
   stddraw.filledCircle(position[0],position[1], 0.05)
   stddraw.show(40)

def randTest(pos):
   if pos<=0.0 or pos>=1.0: return -1
   else: return 1

def bewegeKugel(position, richtung):
   x_pos = position[0] + richtung[0]
   y_pos = position[1] + richtung[1]
   return x_pos, y_pos

def richtungAnpassen(position, richtung):
   dx = richtung[0] * randTest(position[0])
   dy = richtung[1] * randTest(position[1])
   return dx, dy

#---- Das Hauptprogramm  ----------------------------------------

richtung = ( float(sys.argv[1]), float(sys.argv[2]) )
position = ( random.random(), random.random() )
leinwandVorbereiten()
while True:
    richtung = richtungAnpassen( position, richtung )
    position = bewegeKugel(position, richtung)
    bildAnzeigen(position)
```

kugelFunktionen.py

```
import stddraw

def leinwandVorbereiten():
   stddraw.setCanvasSize(800,800)
   stddraw.setPenColor(stddraw.RED)

def bildAnzeigen(position):
   stddraw.clear(stddraw.YELLOW)
   stddraw.filledCircle(position[0],position[1], 0.05)
   stddraw.show(40)

def randTest(pos):
   if pos<=0.0 or pos>=1.0: return -1
   else: return 1

def bewegeKugel(position, richtung):
   x_pos = position[0] + richtung[0]
   y_pos = position[1] + richtung[1]
   return x_pos, y_pos

def richtungAnpassen(position, richtung):
   dx = richtung[0] * randTest(position[0])
   dy = richtung[1] * randTest(position[1])
   return dx, dy
```

kugel_8.py

```
import kugelFunktionen, sys, random

richtung = ( float(sys.argv[1]), float(sys.argv[2]) )
position = ( random.random(), random.random() )
kugelFunktionen.leinwandVorbereiten()
while True:
    richtung = kugelFunktionen.richtungAnpassen( position, richtung )
    position = kugelFunktionen.bewegeKugel(position, richtung)
    kugelFunktionen.bildAnzeigen(position)
```

Application programming interface (API, Anwendungsschnittstelle)

Um ein Modul zur Benutzung bereitzustellen,
schreibt man ein *application programming interface* (API, Anwendungsschnittstelle).

API für das Modul `kugelFunktionen`:

Funktion	Beschreibung
`leinwandVorbereiten()`	stellt die Leinwandgröße von `stddraw` auf 800×800 Pixel und die Stiftfarbe auf rot
`bildAnzeigen(pos)`	zeigt mit `stddraw` ein Bild mit gelbem Hintergrund und rotem Kreis an Position `pos`
`randTest(n)`	gibt 1 zurück, falls $0 < n < 1$; sonst wird -1 zurückgegeben
`bewegeKugel(pos, dir)`	`pos` und `dir` sind Tupel aus jeweils zwei Zahlen; es wird die Position zurückgegeben, die man von `pos` mit einem Schritt in Richtung `dir` erreicht
`richtungAnpassen(pos, dir)`	`pos` und `dir` sind Tupel aus jeweils zwei Zahlen; es wird die Richtung zurückgegeben, in die die Kugel an Position `pos`, die in Richtung `dir` rollt, weiterrollen kann

Programmier-Auftrag: lasse viele Kugeln hin- und herrollen

Benutzung des Moduls `kugelFunktionen.py`

Statt einer Kugel (wie bei `kugel_8.py`) sollen viele Kugeln über die Leinwand rollen.

In `kugel_8.py` wird die Kugel durch ihre Position und ihre Richtung beschrieben.
Um sie in *einem* Objekt zu beschreiben, können wir das Tupel `(position, richtung)` nehmen.

Viele Kugeln können als Array aus solchen Tupeln beschrieben werden.
Alles, was man in `kugel_8.py` für eine Kugel macht, macht man im neuen Programm für viele Kugeln.

Die **Daten** des Programms:

- ein Array mit einem Eintrag für jede Kugel
- jeder Eintrag ist ein Tupel aus Position und Richtung
- (*Position* und *Richtung* sind jeweils Tupel aus 2 Zahlenwerten für die x- und die y-Koordinate)

Programmier-Auftrag: lasse viele Kugeln hin- und herrollen

Benutzung des Moduls `kugelFunktionen.py`

Statt einer Kugel (wie bei `kugel_8.py`) sollen viele Kugeln über die Leinwand rollen.

In `kugel_8.py` wird die Kugel durch ihre Position und ihre Richtung beschrieben.
Um sie in *einem* Objekt zu beschreiben, können wir das Tupel `(position, richtung)` nehmen.

Viele Kugeln können als Array aus solchen Tupeln beschrieben werden.
Alles, was man in `kugel_8.py` für eine Kugel macht, macht man im neuen Programm für viele Kugeln.

Die **Daten** des Programms:

- ein Array mit einem Eintrag für jede Kugel
- jeder Eintrag ist ein Tupel aus Position und Richtung
- (*Position* und *Richtung* sind jeweils Tupel aus 2 Zahlenwerten für die x- und die y-Koordinate)

Die grobe Struktur des Programmes

Das alte Programm mit einer Kugel:

1. Lies die Richtung der Kugel ein.
2. Erzeuge zufällig eine Position der Kugel.
3. Wiederhole:
   - **3.1** bewege die Kugel einen Schritt
   - **3.2** zeige das Bild mit der Kugel an

1. Lies die Anzahl der Kugeln ein.
2. Erzeuge Positionen und Richtungen für alle Kugeln.
3. Wiederhole:
   - **3.1** bewege jede Kugel einen Schritt
   - **3.2** zeige das Bild mit den Kugeln an

Was können wir aus dem Modul `kugelFunktionen.py` benutzen?

- `leinwandVorbereiten()`
- `richtungAnpassen()`
- `bewegeKugel()`

Was muss neu gemacht werden?

- `bildAnzeigen()` zeigt nur ein Bild mit einer Kugel
  und muss für „viele Kugeln“ neu geschrieben werden.

Die grobe Struktur des Programmes

Das alte Programm mit einer Kugel:

1. Lies die Richtung der Kugel ein.
2. Erzeuge zufällig eine Position der Kugel.
3. Wiederhole:
   - **3.1** bewege die Kugel einen Schritt
   - **3.2** zeige das Bild mit der Kugel an

1. Lies die Anzahl der Kugeln ein.
2. Erzeuge Positionen und Richtungen für alle Kugeln.
3. Wiederhole:
   - **3.1** bewege jede Kugel einen Schritt
   - **3.2** zeige das Bild mit den Kugeln an

Was können wir aus dem Modul `kugelFunktionen.py` benutzen?

- `leinwandVorbereiten()`
- `richtungAnpassen()`
- `bewegeKugel()`

Was muss neu gemacht werden?

- `bildAnzeigen()` zeigt nur ein Bild mit einer Kugel
  und muss für „viele Kugeln“ neu geschrieben werden.

Das Programm kugeln.py mit import kugelFunktionen

```
# kugeln.py
import kugelFunktionen
import sys, random
# Zeige das Bild mit allen Kugeln an.
def bildAnzeigen(kugeln):
   stddraw.clear(stddraw.YELLOW)                                  # Lösche das alte Bild und färbe den Hintergrund gelb.
   for k in kugeln:  stddraw.filledCircle(k[0][0],k[0][1], 0.05)  # Male jede Kugel als Kreis an ihre Position.
   stddraw.show(40)                                               # Zeige das Bild an.
#------- Hauptprogramm ----------------------------------------------------------------------------------------------------
# Lies die Anzahl n der Kugeln von der Kommandozeile.
n = int(sys.argv[1])
# Erzeuge n Kugeln mit zufälliger Position und Richtung.
kugeln = []
for i in range(n):
  position = ( random.random(), random.random() )                 # Wähle zufällig eine Position der Kugel.
  richtung = ( random.random()/20, random.random()/20 )           # Wähle zufällig eine Richtung der Kugel.
  kugeln += [ (position, richtung) ]                              # Speichere Position und Richtung der Kugel.
# Die Leinwand wird vorbereitet.
leinwandVorbereiten()
# Lasse die Kugeln rollen.
while True:
    for i in range(len(kugeln)):                                  # Für alle Kugeln:
      richtung = kugelFunktionen.richtungAnpassen( kugeln[i][0], kugeln[i][1] )  #    bestimme die Richtung der Kugel
      position = kugelFunktionen.bewegeKugel( kugeln[i][0], richtung )           #    bewege die Kugel und bestimme ihre neue Position
      kugeln[i] = ( position, richtung )                          #    speichere Position und Richtung der Kugel
    # Male das Bild mit den Kugeln.
    kugelFunktionen.bildAnzeigen(kugeln)
```

Varianten von `import`

Die Anweisung `from moduldatei import f`

- führt die Datei `moduldatei.py` aus und
- erlaubt es, die Funktion `f` aus `moduldatei.py` zu benutzen.
  Sie wird im Klienten als `f()` benutzt.

Varianten von `import`

Die Anweisung `from moduldatei import *`

- führt die Datei `moduldatei.py` aus und
- erlaubt es, alle Funktionen aus `moduldatei.py` zu benutzen.
  Sie werden im Klienten direkt mit ihren Namen aus `moduldatei.py` benutzt.

Varianten von `import`

Die Anweisung `from moduldatei import f as g`

- führt die Datei `moduldatei.py` aus und
- erlaubt es, die Funktion `f` aus `moduldatei.py` zu benutzen.
  Sie wird im Klienten als `g()` benutzt.

Das Programm kugeln_2.py mit from kugelFunktionen import *

```
# kugeln_2.py
from kugelFunktionen import *
import sys, random
# Zeige das Bild mit allen Kugeln an.
def bildAnzeigen(kugeln):
   stddraw.clear(stddraw.YELLOW)                                  # Lösche das alte Bild und färbe den Hintergrund gelb.
   for k in kugeln:  stddraw.filledCircle(k[0][0],k[0][1], 0.05)  # Male jede Kugel als Kreis an ihre Position.
   stddraw.show(40)                                               # Zeige das Bild an.
#------- Hauptprogramm ----------------------------------------------------------------------------------------------------------
# Lies die Anzahl n der Kugeln von der Kommandozeile.
n = int(sys.argv[1])
# Erzeuge n Kugeln mit zufälliger Position und Richtung.
kugeln = []
for i in range(n):
  position = ( random.random(), random.random() )                 # Wähle zufällig eine Position der Kugel.
  richtung = ( random.random()/20, random.random()/20 )           # Wähle zufällig eine Richtung der Kugel.
  kugeln += [ (position, richtung) ]                              # Speichere Position und Richtung der Kugel.
# Die Leinwand wird vorbereitet.
leinwandVorbereiten()
# Lasse die Kugeln rollen.
while True:
    for i in range(len(kugeln)):                                  # Für alle Kugeln:
      richtung = richtungAnpassen( kugeln[i][0], kugeln[i][1] )   #    bestimme die Richtung der Kugel
      position = bewegeKugel( kugeln[i][0], richtung )            #    bewege die Kugel und bestimme ihre neue Position
      kugeln[i] = ( position, richtung )                          #    speichere Position und Richtung der Kugel
    # Male das Bild mit den Kugeln.
    bildAnzeigen(kugeln)
```

Die Funktion `bildAnzeigen(x)` wird

- im Modul `kugelFunktionen.py` definiert und
- im Klient `kugeln_2.py` definiert, der alle Funktionen aus `kugelFunktionen.py` importiert.

Die letzte Definition wird genommen.

Mit `import ...` hat jedes Modul seinen eigenen „Namensraum“.

Mit `from ... import *` geht man ein Risiko ein,
falls verschiedene Module Funktionen mit gleichem Namen definieren.

Die Funktion `bildAnzeigen(x)` wird

- im Modul `kugelFunktionen.py` definiert und
- im Klient `kugeln_2.py` definiert, der alle Funktionen aus `kugelFunktionen.py` importiert.

Die letzte Definition wird genommen.

Mit `import ...` hat jedes Modul seinen eigenen „Namensraum“.

Mit `from ... import *` geht man ein Risiko ein,
falls verschiedene Module Funktionen mit gleichem Namen definieren.

Bemerkungen

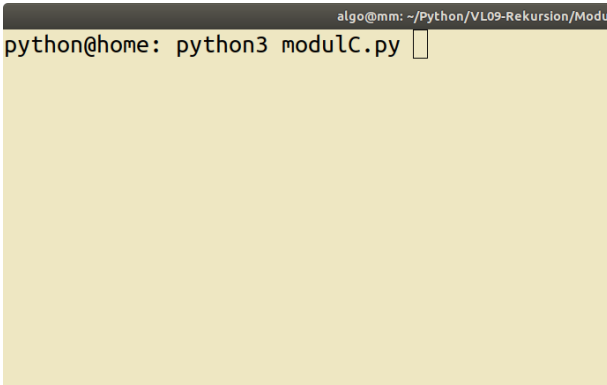
Wir wollen uns nochmal anschauen, was beim Import eines Moduls passiert.
Die folgenden Klienten und Module geben am Anfang und Ende aus,
dass sie gestartet bzw. beendet werden.

Modul `modulC.py` definiert eine Funktion und führt sie einmal aus.

Klient `klientZ.py` importiert `modulC.py` und führt einmal die Funktion daraus aus.

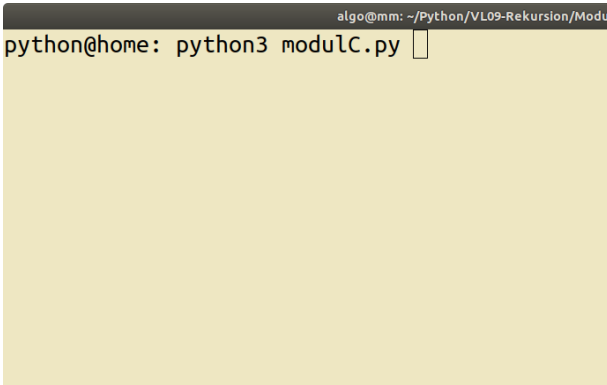
```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
   mini = min(a)
   maxi = max(a)
   return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')
```

```
# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')

import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```



```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')
```

```
# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')

import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```



```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')
```

```
# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')

import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulC.py
Start von modulC.py .
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
```

```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
   mini = min(a)
   maxi = max(a)
   return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')
```

```
# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')

import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulC.py
Start von modulC.py .
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientZ.py
```

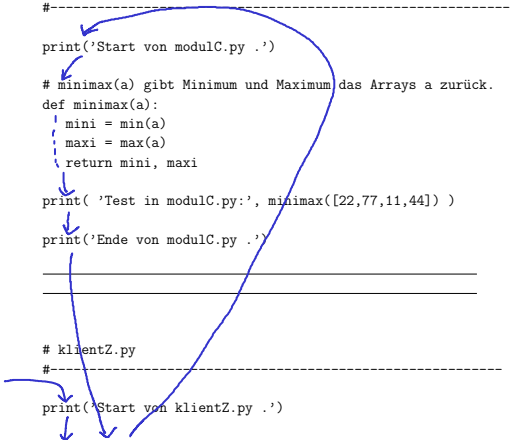
```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
   mini = min(a)
   maxi = max(a)
   return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')
```

```
# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')
import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulC.py
Start von modulC.py .
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientZ.py
```

```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')
```

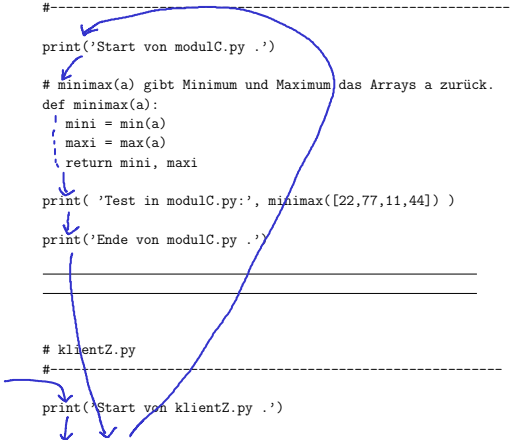
```
# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')

import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```



```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulC.py
Start von modulC.py .
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientZ.py
```

```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')
```

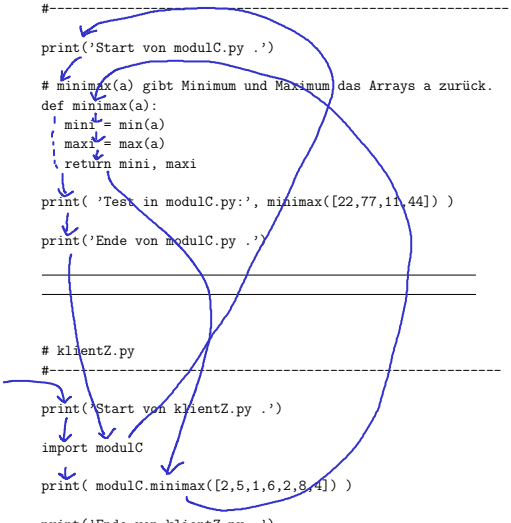
```
# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')

import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```



```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulC.py
Start von modulC.py .
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientZ.py
```

```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')
```

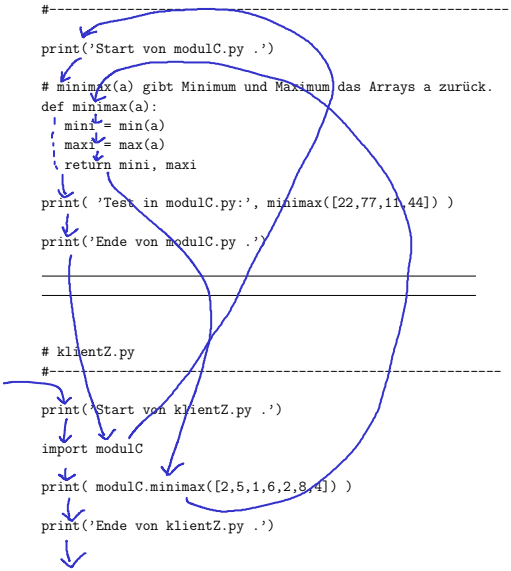
```
# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')

import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```



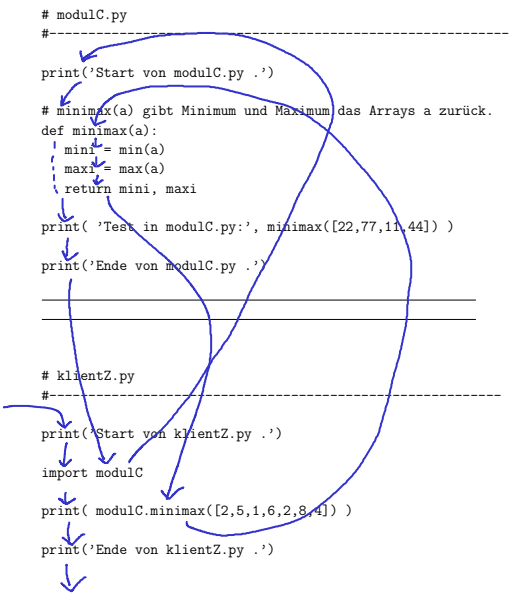
```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulC.py
Start von modulC.py .
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientZ.py
```

```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')
```

```
# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')

import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```



```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulC.py
Start von modulC.py .
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientZ.py
```



```
# modulC.py
#-------------------------------------------------------------

print('Start von modulC.py .')

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

print( 'Test in modulC.py:', minimax([22,77,11,44]) )

print('Ende von modulC.py .')

# klientZ.py
#-------------------------------------------------------------

print('Start von klientZ.py .')

import modulC

print( modulC.minimax([2,5,1,6,2,8,4]) )

print('Ende von klientZ.py .')
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulC.py
Start von modulC.py .
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientZ.py
Start von klientZ.py .
Start von modulC.py .
Test in modulC.py: (11, 77)
Ende von modulC.py .
(1, 8)
Ende von klientZ.py .
python@home:
```

Jedes Modul wird nur einmal ausgeführt!

Module kennen ihren Namen

Die Systemvariable `__name__` hat einen String als Wert.

- In einem Modul ist der Wert ihr Dateiname (ohne `.py`).
- In einem Klienten ist der Wert `'__main__'`.

Dadurch kann man in einer Datei unterscheiden, ob sie als Modul oder als Klient ausgeführt wird:

Der Ausdruck `__name__ == '__main__'` hat Wert

- `False`, wenn er in einem Modul ausgeführt wird, bzw.
- `True`, wenn er in einem Klienten ausgeführt wird.

Dadurch können wir Module so programmieren, dass

- nichts ausgeführt wird, wenn sie als Modul importiert werden, und
- eine Testfunktion ausgeführt wird, wenn sie als Klient ausgeführt werden.

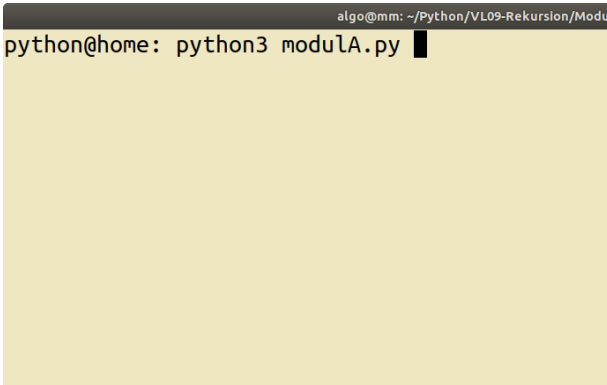
```
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
   mini = min(a)
   maxi = max(a)
   return mini, maxi

# Testfunktion
def test():
  print( '__name__ ist ' + __name__ +'.' )
  print( 'Test in modulC.py:', minimax([22,77,11,44]) )
  print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```
# klientX.py
#-----------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulA.py
```

```
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

# Testfunktion
def test():
    print( '__name__ ist ' + __name__ +'.' )
    print( 'Test in modulC.py:', minimax([22,77,11,44]) )
    print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```
# klientX.py
#-----------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```



```python
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

# Testfunktion
def test():
    print( '__name__ ist ' + __name__ +'.' )
    print( 'Test in modulC.py:', minimax([22,77,11,44]) )
    print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```python
# klientX.py
#-----------------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulA.py
__name__ ist __main__.
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
```

```
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

# Testfunktion
def test():
    print( '__name__ ist ' + __name__ +'.' )
    print( 'Test in modulC.py:', minimax([22,77,11,44]) )
    print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```
# klientX.py
#-----------------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulA.py
__name__ ist __main__.
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
```

```
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
   mini = min(a)
   maxi = max(a)
   return mini, maxi

# Testfunktion
def test():
  print( '__name__ ist ' + __name__ +'.' )
  print( 'Test in modulC.py:', minimax([22,77,11,44]) )
  print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```
# klientX.py
#-------------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulA.py
__name__ ist __main__.
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientX.py
```

```python
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
   mini = min(a)
   maxi = max(a)
   return mini, maxi

# Testfunktion
def test():
  print( '__name__ ist ' + __name__ +'.' )
  print( 'Test in modulC.py:', minimax([22,77,11,44]) )
  print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```python
# klientX.py
#-------------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulA.py
__name__ ist __main__.
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientX.py
```

```
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
   mini = min(a)
   maxi = max(a)
   return mini, maxi

# Testfunktion
def test():
  print( '__name__ ist ' + __name__ +'.' )
  print( 'Test in modulC.py:', minimax([22,77,11,44]) )
  print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```
# klientX.py
#-------------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulA.py
__name__ ist __main__.
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientX.py
```

```
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

# Testfunktion
def test():
    print( '__name__ ist ' + __name__ +'.' )
    print( 'Test in modulC.py:', minimax([22,77,11,44]) )
    print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```
# klientX.py
#-------------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulA.py
__name__ ist __main__.
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientX.py
```

```
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

# Testfunktion
def test():
    print( '__name__ ist ' + __name__ +'.' )
    print( 'Test in modulC.py:', minimax([22,77,11,44]) )
    print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```
# klientX.py
#-------------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulA.py
__name__ ist __main__.
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientX.py
```

```
# modulA.py
#-------------------------------------------------------------

# minimax(a) gibt Minimum und Maximum das Arrays a zurück.
def minimax(a):
    mini = min(a)
    maxi = max(a)
    return mini, maxi

# Testfunktion
def test():
    print( '__name__ ist ' + __name__ +'.' )
    print( 'Test in modulC.py:', minimax([22,77,11,44]) )
    print( 'Ende von modulC.py .' )

# Rufe test() auf,
#   falls das Programm nicht als Modul ausgeführt wird.
if __name__ == '__main__': test()
```

```
# klientX.py
#-------------------------------------------------------

import modulA

print('__name__ in klientX.py ist ' + __name__ +'.')

print( modulA.minimax([2,5,1,6,2,8,4]) )
```

```
algo@mm: ~/Python/VL09-Rekursion/Modu
python@home: python3 modulA.py
__name__ ist __main__.
Test in modulC.py: (11, 77)
Ende von modulC.py .
python@home:
python@home: python3 klientX.py
__name__ in klientX.py ist __main__.
(1, 8)
python@home:
```

Jedes Modul wird nur einmal ausgeführt!

Zusammenfassung

- Mit Funktionen kann man einfacher zu verstehende Programme aufschreiben.
- Durch Module kann man Funktionen für verschiedene Klienten benutzbar machen.
- Module kann man unabhängig von Klienten testen.
- Funktionen und Module helfen dabei,

  Programmieraufgaben in Teilaufgaben zu zerlegen,

  die unabhängig voneinander gelöst und zum Gesamtprogramm zusammengefügt werden.