py2cd - Übersicht

Inhaltsverzeichnis

1 Grundgerüst

Jedes py2cd-Program muss folgendes Grundgerüst haben:

```
# Dies Modul muss immer importiert werden
from py2cd import *
# Für vordefinierte Farben wird dieses Modul benötigt
from py2cd.farben import *

# Das Spiel muss zuerst initialisiert werden!
Spiel.init(640, 480, "Mein Spiel") # Breite, Höhe, Titel


# RESTLICHES PROGRAMM STEHT HIER


# Startet das Spiel. Anweisungen danach werden NICHT ausgeführt
Spiel.starten()
```

Hinweis: Alle folgenden Beispiele benötigen dieses Grundgerüst. Oftmals steht es allerdings nicht dabei damit die Beispiele kürzer werden!

2 Farben

Eine Farbe wird aus 3 Komponenten aufgebaut: Rot, Grün, Blau. Die R-, G-, B-Werte dürfen dabei Werte zwischen 0 und 255 annehmen. Schwarz ist `(0, 0, 0)`, Weiß ist `(255, 255, 255)`.

```
meineFarbe = (255, 0, 0) # Rot voll, Grün und Blau nichts =>
 ↪ Rot
r = Rechteck(10, 10, 50, 50, meineFarbe)
```

3 Vorhandene Objekte

Es stehen verschiedene vordefinierte Objekte zur Verfügung. Diese können auf dem Spielfeld platziert werden, wobei sich die Position dieser Objekte immer auf deren **linke obere Ecke** bezieht!

3.1 Rechtecke

Benötigte Werte: x, y (linke obere Ecke), Breite, Höhe und Farbe.

```
r1 = Rechteck(270, 200, 100, 100, GELB)
```

3.2 Kreise

Benötigte Werte: x, y (linke obere Ecke), Radius und Farbe.

```
k1 = Kreis(390, 110, 70, MATT_GRUEN)
```

3.3 Linie

Benötigte Werte: Start- und Endpunkt, jeweils x- und y-Koordinate in einem Tupel z.B. `(100, 140)`, Farbe, Dicke (optional).

```
l1 = Linie((100, 300), (540, 300), ROT, 2)
```

3.4 Ovale

Benötigte Werte: x, y (linke obere Ecke), Breite-Radius, Höhe-Radius und Farbe.

```
o1 = Oval(100, 50, 20, 30, GELB)
```

3.5 Text

Benötigte Werte: Der Text, x, y (linke obere Ecke) und Farbe.

```
t1 = Text('Hallo', 100, 200, SCHWARZ)
```

3.6 Polygone (Vielecke)

Benötigte Werte: Liste aller Eckpunkte, jeweils x- und y-Koordinate in einem Tupel z.B. `(100, 140)`.

```
ecken_liste = [(190, 242), (195, 238), (200, 242)]
p1 = Polygon(ecken_liste, BLAU)
```

3.7 Bilder

Bilder werden über den `BilderSpeicher` verwaltet, müssen einmal geladen werden und können dann beliebig oft angezeigt werden.
Benötigte Werte: Interne Name für das Bild (frei wählbar), der Pfad/Dateiname des Bildes auf dem Computer.

```
# Das Bild EINMAL in Speicher laden.
# Das Bild liegt in dem Order 'bilder' und heißt 'scratch.png'
BildSpeicher.lade_bild("scratch", "bilder/scratch.png")

# Das Bild zweimal an verschiedenen Stellen anzeigen
bild1 = BildSpeicher.gib_bild("scratch", 100, 100)
bild2 = BildSpeicher.gib_bild("scratch", 300, 250)
```

3.8 Animationen

Animation sind einfach ein Liste von Bildern, die schnell hintereinander angezeigt werden.

```
namen_liste = ['bilder/n1.png', 'bider/n2.png']
a1 = BildAnimation(namen_liste)

# Animation steuern
a1.start() # Animation starten
a1.pause() # Animation pausieren
a1.stop() # Animation stoppen

a1.setze_wiederhole(True) # Animation endlos wiederholen
```

Hinweis: Die Bildernamen können ENTWEDER, der interne Name im Bildspeicher (z.B. oben 'scratch') ODER der Dateipfad/Dateiname des Bildes sein.

4 Objekte bewegen

Sämtliche oben aufgeführten Objekte besitzen die hier genannten Funktionen. Man ruft diese Funktionen über den Namen der Variablen in der das Objekt gespeichert ist auf. Z.B.: `r1.aendere_position(20, 20)`.

4.1 Position ändern

Ändert die Position eines Objektes **um** einen bestimmten Wert. Benötigte Werte: x-, y-Wert

```
r1 = Rechteck(20, 20, 100, 100, GELB)
r1.aendere_position(100, 100)
```

4.2 Position setzen

Setzt die Position eines Objektes **auf** einen bestimmten Wert.
Benötigte Werte: x-, y-Wert.

```
r1 = Rechteck(20, 20, 100, 100, GELB)
r1.setze_position(100, 100)
```

4.3 Position abfragen

Benötigte Werte: Keine.
Gibt zurück: Ein Tupel (x-, y-Wert)

```
r1 = Rechteck(20, 20, 100, 100, GELB)
pos = r1.position() # = (20, 20)
```

4.4 Mitte eines Objektes setzen

Setzt die Mitte eines Objektes, **NICHT** seine linke obere Ecke. Nützlich bei z.B. Kreisen.
Benötigte Werte: x-, y-Wert.

```
k1 = Kreis(450, 150, 10, ROT)
k11.mitte = (455, 155)
```

4.5 Mitte eines Objektes abfragen

Fragt die Mitte eines Objektes ab, NICHT seine linke obere Ecke.
Benötigte Werte: Keine.
Gibt zurück: Ein Tupel (x-, y-Wert)

```
k1 = Kreis(450, 150, 10, ROT)
mitte = k11.mitte # = (455, 155)
```

4.6 Abstand zum Rand setzen

Man kann auch direkt den Abstand zum Rand setzen. Dies gibt für oben, unten, rechts und links. Benötigte Werte: Abstand

```
k1 = Kreis(450, 150, 10, ROT)

# Abstand zum linken Rand: 20
k1.links = 20

# Abstand zum rechten Rand: 30
k1.rechts = 30

# Abstand zum oberen Rand: 40
k1.oben = 40

# Abstand zum unteren Rand: 50
k1.unten = 50
```

4.7 Abstand zum Rand abfragen

Genau so kann man auch den Abstand zum Rand abfragen. Benötigte Werte: Keine Gibt zurück: Abstand

```
k1 = Kreis(450, 150, 10, ROT)

# Abstand zum linken Rand
links = k1.links

# Abstand zum rechten Rand
rechts = k1.rechts

# Abstand zum oberen Rand
oben k1.oben

# Abstand zum unteren Rand
unten = k1.unten
```