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 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!
Zusätzlich, ist jedes Objekt von einem unsichtbaren Rechteckt, dem *umgebenden Rechteck* begrenzt. Falls weiter unten sich auf die Breite oder Höhe von Obejekten bezogen wird, ist dies immer gleichbedeutend mit der Breite/Höhe des umgebenden Rechtecks.

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)`.

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

Hinweis: Einige wichtige Farben sind bereits vordefiniert.

Rechtecke

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

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

Kreise

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

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

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)
```

Ovale

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

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

Text

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

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

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)
```

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)
```

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.

3 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)`.

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)
```

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)
```

Position abfragen

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

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

x- und y-Koordinaten direkt setzen

Man kann die x- und y-Koordinaten eines Objektes auch direkt festlegen.

```
box = Rechteck(10, 10, 50, 50, ROT)

# x- und y-Koordinate festlegen
box.x = 100
box.y = 200
```

x- und y-Koordinaten abfragen

Man kann die x- und y-Koordinaten eines Objektes ganz einfach abfragen

```
box = Rechteck(10, 10, 50, 50, ROT)

# x- und y-Koordinate abfragen und ausgeben
print(box.x, box.y)
```

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)
```

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)
```

Abstand zum Rand setzen

Man kann auch direkt den Abstand zum jeweiligen Rand setzen. Dies geht 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
```

Achtung: Der Abstand zum Rand geht immer von der dem Rand am nächsten gelegenen Seite aus! Setzt man zwei konfligierende Anweisungen: `k1.rechts` und `k1.links`, so zählt natürlich nur die zuletzt ausgeführte Anweisung.

Abstand zum Rand abfragen

Wie man den Abstand zum Rand setzen kann, kann man ihn auch 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
```

Objekte zentrieren

Man kann Objekte auch mittig auf dem Spielfeld darstellen. Entweder nur horizontal, vertikal oder in beide Richtungen zentriert.
Benötigte Werte: Keine

```
# Ganz mittig zentrieren
bild.zentriere()

# Nur horizontal zentrieren
bild.zentriere_horizontal()

# Nur horizontal zentrieren
bild.zentriere_vertikal()
```

4 Spiel - Die zentrale Steuerung

Das Spiel initialisieren

Das Spiel muss **IMMER** zuerst initialisiert werden, bevor irgendetwas getan werden kann.
Benötigte Werte: Breite, Höhe, Titel

```
Spiel.init(640, 480, "Mein Spiel")
```

Das Spiel starten

Startet das Spiel und zeigt das Fenster an. Anweisungen danach werden **NICHT** ausgeführt:
Benötigte Werte: Keine

```
Spiel.starten()
```

Das Spiel beenden

Beendet das Spiel und schließt das Fenster.
Benötigte Werte: Keine

```
Spiel.beenden()
```

Den Fenstertitel setzen

Benötigte Werte: Titel

```
Spiel.setze_fenster_titel("Mein Fenster")
```

Die Fensterhintergrundfarbe ändern

Benötigte Werte: Die Farbe

```
Spiel.setze_hintergrund_farbe(BLAU)
```

Hilfsgitter einblenden

Zeichnet Gitterlinien und ein Koordinatensystem ein, um die Positionierung von Objekten zu vereinfachen. Benötigte Werte: Keine

```
Spiel.zeichne_gitter()
```

Hinweis: das Gitter ist auch nur ein Objekt und kann deshalb von anderen Objekten verdeckt werden.

Tastatureingaben

Es gibt zwei verschiedene Arten auf Tastatur eingaben zu reagieren. Beide Arten binden eine Rückruf-Funktion (Callback) an eine bestimmte Taste. Diese Rückruf-Funktion wird aufgerufen, wenn diese Taste gedrückt wird.

Solange eine Taste gedrückt wird

Die Funktion `Spiel.registriere_solange_taste_unten(taste, rueckruf_funktion)` bindet die gewünschte Taste an die gewünschte Ruckruf-Funktion. Diese Funktion wird **wiederholt** aufgerufen, solange die Taste gedrückt ist.

```
# Rückruf-Funktion
def solange_links(dt):
    print("Taste gedrückt")

# Linke Pfeiltaste 'T_LINKS', an die Rückruffunktion binden
Spiel.registriere_solange_taste_unten(T_LINKS, solange_links)
```

Hinweis: Die Rückruffunktion benötigt genau **einen** Übergabeparameter.

Wenn eine Taste gedrückt wird

Die Funktion `Spiel.registriere_taste_gedrueckt(taste, rueckruf_funktion)` bindet die gewünschte Taste an die gewünschte Ruckruf-Funktion. Diese Funktion wird **genau zweimal** aufgerufen. Einmal beim Drücken der Taste (`unten = True`) und einmal beim Loslassen (`unten = False`).

```
def leer_gedrueckt(unten, pyg_ereignis):
    if unten:
        print("Leertaste gedrückt.")
    else:
        print("Leertaste losgelassen.")

# Leertaste 'T_LEER', an die Rückruffunktion binden
Spiel.registriere_taste_gedrueckt(T_LEER, leer_gedrueckt)
```

Hinweis: Die Rückruffunktion benötigt genau **zwei** Übergabeparameter.