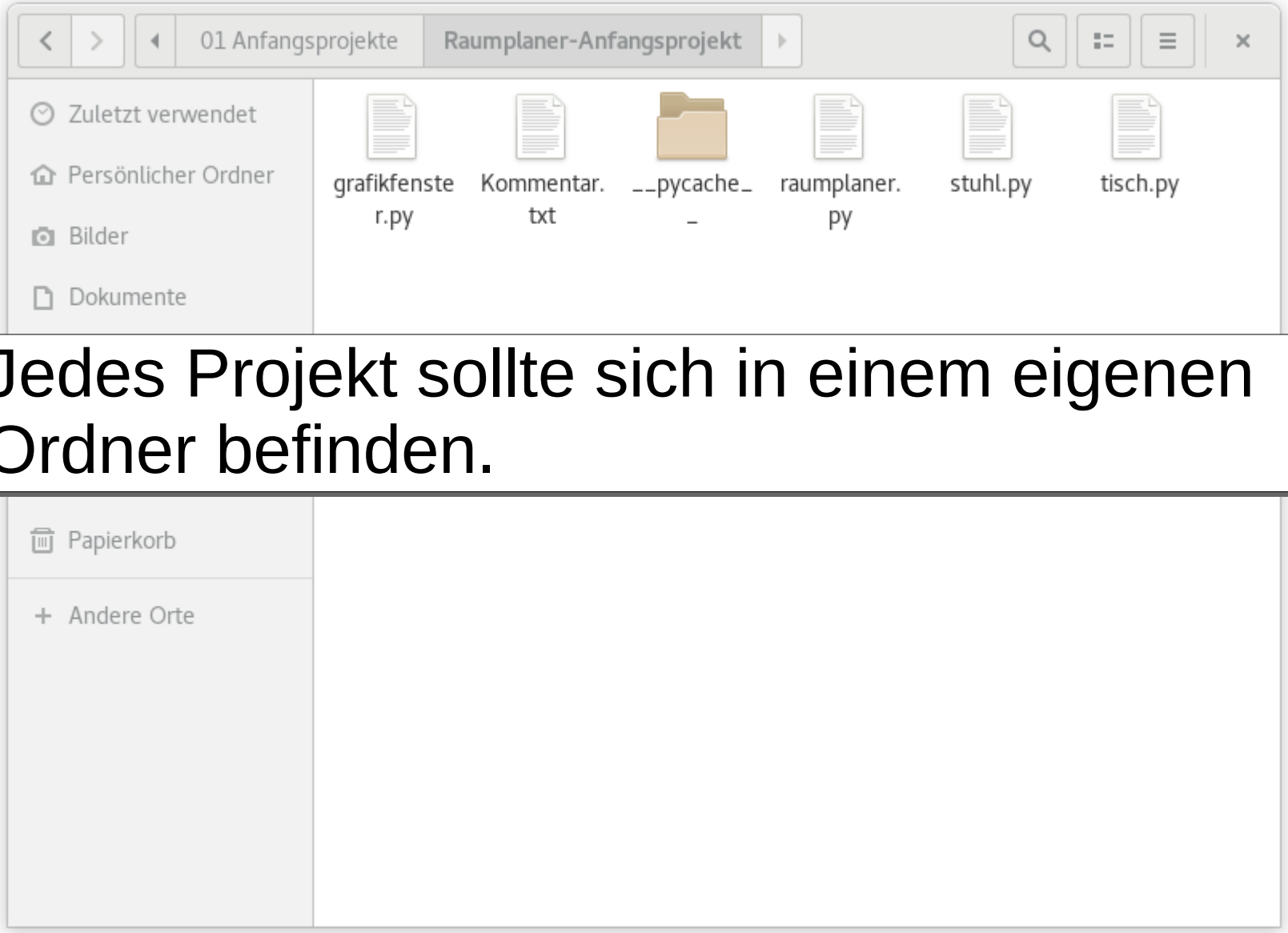
Fachausdrücke

Erläuterung von
grundlegenden
Fachausdrücken

am Anfangsprojekt
und der Klasse Stuhl

Fachausdrücke



Jedes Projekt sollte sich in einem eigenen Ordner befinden.

Fachausdrücke

Dieses Projekt enthält vier Module.

Bei Python ist es nicht zwingend, es empfiehlt sich aber, jede Klasse in einer eigenen Datei zu halten.

Fachausdrücke

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

File Edit Format Run Options Window Help

```
# stuhl.py

from grafikfenster import *
from math import radians

"""
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Im Kopf der Datei finden wir ...

- einen Kommentar mit dem Dateinamen,

Fachausdrücke

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

File Edit Format Run Options Window Help

```
# stuhl.py

from grafikfenster import *
from math import radians

"""
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Im Kopf der Datei finden wir ...

- die Importe von eigenen Modulen,

Fachausdrücke

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

File Edit Format Run Options Window Help

```
# stuhl.py

from grafikfenster import *
from math import radians

"""
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Im Kopf der Datei finden wir ...

- die Importe von Python-Modulen.

Fachausdrücke

Die Klassendefinition

- beginnt mit dem Wort (*Bezeichner*) class

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
### ------------------------------------------------
class Stuhl():
    """Klasse Stuhl
    ermoeglicht das Zeichnen und Bearbeiten eines
    Stuhl-Symbols fuer den Raumplaner"""

    def __init__(self, sichtbar=False):
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Fachausdrücke

Die Klassendefinition

- enthält dann den Namen der Klasse

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
###-------------------------------------------------
class Stuhl():
    """Klasse Stuhl
    ermoeglicht das Zeichnen und Bearbeiten eines
    Stuhl-Symbols fuer den Raumplaner"""

    def __init__(self, sichtbar=False):
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Fachausdrücke

Die Klassendefinition

- benötigt danach eine (*hier leere*) Klammer.

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
###-------------------------------------------------
class Stuhl():
    """Klasse Stuhl
    ermoeglicht das Zeichnen und Bearbeiten eines
    Stuhl-Symbols fuer den Raumplaner"""

    def __init__(self, sichtbar=False):
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

In die Klammer können Namen von Klassen eingefügt werden, von denen diese Klasse erbt. Zur Vererbung ... später!

Fachausdrücke

Immer wichtig: Es folgt ein Kommentar mit einer Beschreibung der Aufgabe der Klasse.

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
from math import radians

### ------------------------------------------------
class Stuhl():
    """Klasse Stuhl
    ermoeglicht das Zeichnen und Bearbeiten eines
    Stuhl-Symbols fuer den Raumplaner"""

    def __init__(self, sichtbar=False):
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Kommentare an dieser Stelle benötigen drei Anführungsstriche.
Sie werden als Attribut __doc__ der Klasse gespeichert.

Fachausdrücke

Es folgt der Konstruktor der Klasse.

Er ist die Methode, die aufgerufen wird, wenn ein Objekt (Instanz) der Klasse erzeugt werden soll.

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
def __init__(self, sichtbar=False):
    """einfacher Konstruktor"""
    self.x=20
    self.y=20
    self.b=40
    self.t=40
    self.w=270
    self.f="blue"
    self.s=sichtbar
    if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Daher finden wir in der Klassendefinition der RaumplanerApp den Aufruf des Konstruktors
stuhl = Stuhl(True)
zum Erzeugen des Stuhlobjekts mit dem Namen stuhl

Fachausdrücke

Der Konstruktor (*es kann bei Python nur einen geben*) beginnt wie bei jeder Methode mit dem einfach eingerückten Wort (*Bezeichner*) ***def*** (*für define*) und er heißt immer ***__init__***

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
    def __init__(self, sichtbar=False):
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Alle Methoden beginnen mit dem einfach eingerückten Wort (Bezeichner) ***def*** *(für define).*

Fachausdrücke

Der erste Parameter ***self*** bezeichnet das Objekt selbst (*so etwas wie „ich“*) und ist im Konstruktor zwingend notwendig.

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
class Stuhl():
    """Klasse Stuhl
    ermoeglicht das Zeichnen und Bearbeiten eines
    Stuhl-Symbols fuer den Raumplaner"""

    def __init__(self, sichtbar=False):
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Fachausdrücke

Der zweite Parameter ***sichtbar*** gibt die Möglichkeit, dem Konstruktor verschiedene Werte beim Aufruf zu übergeben.

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
class Stuhl():
    """Klasse Stuhl
    ermoeglicht das Zeichnen und Bearbeiten eines
    Stuhl-Symbols fuer den Raumplaner"""

    def __init__(self, sichtbar=False):
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

*In diesem Beispiel wird die Möglichkeit von Python ausgenutzt, Parameter mit vordefinierten Werten zu verwenden. Steht dort nur **sichtbar**, muss beim Aufruf zwingend ein Wert übergeben werden.*

Fachausdrücke

Der Konstruktor dient zum Initialisieren (*deshalb der Name __init__*) der Attributwerte, beispielsweise wird dem x-Attribut der Wert 20 zugewiesen.

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
class Stuhl():
    """Klasse Stuhl
    ermoeglicht das Zeichnen und Bearbeiten eines
    Stuhl-Symbols fuer den Raumplaner"""

    def __init__(self, sichtbar=False):
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

*Alle Attribute sind dadurch gekennzeichnet, dass ihre Namen den Vorsatz **self.** haben.*
Man kann das als „das x vom Objekt“ („mein x“) lesen.

Fachausdrücke

Von der Farbe wird ihr Name gespeichert. Das ist eine Zeichenkette, ein **String**. Strings werden zwischen Anführungszeichen gesetzt.

stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
Stuhl-Symbols fuer den Raumplaner"""

def __init__(self, sichtbar=False):
    """einfacher Konstruktor"""
    self.x=20
    self.y=20
    self.b=40
    self.t=40
    self.w=270
    self.f="blue"
    self.s=sichtbar
    if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

*Eine Besonderheit von Python ist, dass man sowohl einfache als auch doppelte Anführungsstriche verwenden kann. Allerdings muss am **Ende** dasselbe Zeichen stehen wie am **Anfang**.*

Fachausdrücke

if leitet eine Verzweigung des Programms ein.

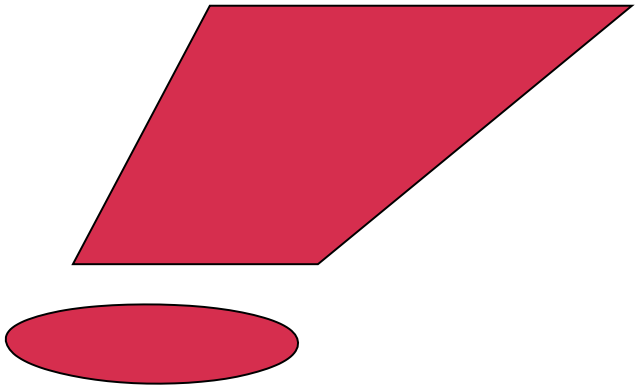
stuhl.py - /home/nutzer/Dokumente/LI/2021-LI-OO/Projekte/01 Anfangsprojekte/Raumplan...

```
from grafikfenster import *
from math import radians

### ------------------------------------------------
class Stuhl():
    """Klasse Stuhl
    ermoeglicht das Zeichnen und Bearbeiten eines
    Stuhl-Symbols fuer den Raumplaner"""

    def __init__(self, sichtbar=False):
        """einfacher Konstruktor"""
        self.x=20
        self.y=20
        self.b=40
        self.t=40
        self.w=270
        self.f="blue"
        self.s=sichtbar
        if sichtbar: self.Zeige()
```

Ln: 1 Col: 0

Hier findet eine einfache Verzweigung statt.
*Hat der Parameter **sichtbar** den Wert **True** wird die Methode **Zeige()** aufgerufen. Dann wird gezeichnet, anderenfalls passiert nichts mehr.*

Python speziell



Python schreibt
die beiden Wahrheitswerte
am Anfang groß:

Fachausdrücke

```
def GibFarbe(self):
    """Get-Methode fuer die Farbe"""
    return self.f
```

Methoden sind Aufgaben, die ein Objekt bei ihrem Aufruf bearbeiten kann.

```
def BewegeHorizontal(self, weite):
    """Veraendernde Methode fuer die x-Position"""
    self.Verberge()
    self.x += weite
    self.Zeige()

def BewegeVertikal(self, weite):
    """Veraendernde Methode fuer die y-Position"""
    self.Verberge()
    self.y += weite
    self.Zeige()

def Drehe(self, winkel):
    """Veraendernde Methode fuer die Orientierung [Winkel]"""
    self.Verberge()
    self.w += winkel
    self.Zeige()

def Verberge(self):
    """Veraendernde Methode fuer die Sichtbarkeit mit Wert False"""
    self.s = False
    Zeichenflaeche.GibZeichenflaeche().Entferne(self)

def Zeige(self):
    """Veraendernde Methode fuer die Sichtbarkeit mit Wert True"""
    self.s = True
    Zeichenflaeche.GibZeichenflaeche().Zeichne(self)
```

Bei der Objektorientierung spricht man vom „message-passing-style“:
Das Objekt erhält von einem anderen eine Botschaft, die es auffordert, eine Aufgabe zu bearbeiten.

Fachausdrücke

```
    def GibFarbe(self):
        """Get-Methode fuer die Farbe"""
        return self.f
```

Methodenkopf:

def <Name>(self,<Parameter>):

```
    def BewegeHorizontal(self, weite):
        """Veraendernde Methode fuer die x-Position"""
        self.Verberge()
        self.x += weite
        self.Zeige()

    def BewegeVertikal(self, weite):
        """Veraendernde Methode fuer die y-Position"""
        self.Verberge()
        self.y += weite
        self.Zeige()

    def Drehe(self, winkel):
        """Veraendernde Methode fuer die Orientierung [Winkel]"""
        self.Verberge()
        self.w += winkel
        self.Zeige()

    def Verberge(self):
        """Veraendernde Methode fuer die Sichtbarkeit mit Wert False"""
        self.s = False
        Zeichenflaeche.GibZeichenflaeche().Entferne(self)

    def Zeige(self):
        """Veraendernde Methode fuer die Sichtbarkeit mit Wert True"""
        self.s = True
        Zeichenflaeche.GibZeichenflaeche().Zeichne(self)
```

nach Doppelpunkt

doppelte Einrückung

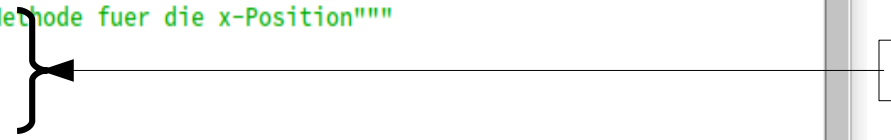
Fachausdrücke

```
    def GibFarbe(self):
        """Get-Methode fuer die Farbe"""
        return self.f
```

Der Methodenrumpf nach dem Doppelpunkt im Kopf

- ist weiter eingerückt

```
    def BewegeHorizontal(self, weite):
        """Veraendernde Methode fuer die x-Position"""
        self.Verberge()
        self.x += weite
        self.Zeige()

    def BewegeVertikal(self, weite):
        """Veraendernde Methode fuer die y-Position"""
        self.Verberge()
        self.y += weite
        self.Zeige()

    def Drehe(self, winkel):
        """Veraendernde Methode fuer die Orientierung [Winkel]"""
        self.Verberge()
        self.w += winkel
        self.Zeige()

    def Verberge(self):
        """Veraendernde Methode fuer die Sichtbarkeit mit Wert False"""
        self.s = False
        Zeichenflaeche.GibZeichenflaeche().Entferne(self)

    def Zeige(self):
        """Veraendernde Methode fuer die Sichtbarkeit mit Wert True"""
        self.s = True
        Zeichenflaeche.GibZeichenflaeche().Zeichne(self)
```



Fachausdrücke

```
    def GibFarbe(self):
        """Get-Methode fuer die Farbe"""
        return self.f
```

Der Methodenrumpf enthält (möglichst) im Kommentar eine

- Beschreibung der Aufgabe

```
    def BewegeHorizontal(self, weite):
        """Veraendernde Methode fuer die x-Position"""
        self.Verberge()
        self.x += weite
        self.Zeige()

    def BewegeVertikal(self, weite):
        """Veraendernde Methode fuer die y-Position"""
        self.Verberge()
        self.y += weite
        self.Zeige()

    def Drehe(self, winkel):
        """Veraendernde Methode fuer die Orientierung [Winkel]"""
        self.Verberge()
        self.w += winkel
        self.Zeige()

    def Verberge(self):
        """Veraendernde Methode fuer die Sichtbarkeit mit Wert False"""
        self.s = False
        Zeichenflaeche.GibZeichenflaeche().Entferne(self)

    def Zeige(self):
        """Veraendernde Methode fuer die Sichtbarkeit mit Wert True"""
        self.s = True
        Zeichenflaeche.GibZeichenflaeche().Zeichne(self)
```

Kommentar

Der Kommentar wird mit der Methode gespeichert, kann ausgelesen werden und wird in der PyShell als Tooltip angezeigt.

Fachausdrücke

```
def GibFarbe(self):
    """Get-Methode fuer die Farbe"""
    return self.f
```

Der Methodenrumpf

- enthält die Anweisungen

```
def BewegeHorizontal(self, weite):
    """Veraendernde Methode fuer die x-Position"""
    self.Verberge()
    self.x += weite
    self.Zeige()

def BewegeVertikal(self, weite):
    """Veraendernde Methode fuer die y-Position"""
    self.Verberge()
    self.y += weite
    self.Zeige()

def Drehe(self, winkel):
    """Veraendernde Methode fuer die Orientierung [Wink
    self.Verberge()
    self.w += winkel
    self.Zeige()

def Verberge(self):
    """Veraendernde Methode fuer die Sichtbarkeit mit W
    self.s = False
    Zeichenflaeche.GibZeichenflaeche().Entferne(self)

def Zeige(self):
    """Veraendernde Methode fuer die Sichtbarkeit mit W
    self.s = True
    Zeichenflaeche.GibZeichenflaeche().Zeichne(self)
```

Wertzuweisung

Wertzuweisungen mit einem einfachen Gleichheitszeichen können wie im Beispiel Berechnungen sein.
Hier wird zum Wert der y-Koordinate die als Parameter der Methode übergebene ***weite*** *addiert und das Ergebnis im y-Attribut abgespeichert.*
Inhaltsgleich wäre:
self.y = self.y + weite

Fachausdrücke

```
def GibFarbe(self):
    """Get-Methode fuer die Farbe"""
    return self.f

def GibSichtbar(self):
    """Get-Methode fuer die Sichtbarkeit"""
    return self.s
```

Rückgabewert

Rückgabewert

Methoden können mit Rückgabewert

- funktional aufgerufen werden

```
def BewegeVertikal(self, weite):
    """Veraendernde Methode fuer die y-Position"""
    self.Verberge()
    self.y += weite
    self.Zeige()

def Drehe(self, winkel):
    """Veraendernde Methode fuer die Orientierung [Winkel]"""
    self.Verberge()
    self.w += winkel
    self.Zeige()

def Verberge(self):
    """Veraendernde Methode fuer die Sichtbarkeit mit Wert False"""
    self.s = False
    Zeichenflaeche.GibZeichenflaeche().Entferne(self)

def Zeige(self):
    """Veraendernde Methode fuer die Sichtbarkeit mit Wert True"""
    self.s = True
    Zeichenflaeche.GibZeichenflaeche().Zeichne(self)
```

*Soll die Methode am Ende der Bearbeitung einen Wert an das aufrufende Objekt zurückgeben, muss dieser hinter dem Wort **return** stehen.*

Fachausdrücke

```
    def GibFarbe(self):
        """Get-Methode fuer die Farbe"""
        return self.f
```

Methoden können ohne Rückgabewert

- prozedural aufgerufen werden

```
    def BewegeHorizontal(self, weite):
        """Veraendernde Methode fuer die x-Position"""
        self.Verberge()
        self.x += weite
        self.Zeige()

    def BewegeVertikal(self, weite):
        """Veraendernde Methode fuer die y-Position"""
        self.Verberge()
        self.y += weite
        self.Zeige()

    def Drehe(self, winkel):
        """Veraendernde Methode fuer die Orientierung [Winkel]"""
        self.Verberge()
        self.w += winkel
        self.Zeige()

    def Verberge(self):
        """Veraendernde Methode fuer die Sichtbarkeit mit Wert False"""
        self.s = False
        Zeichenflaeche.GibZeichenflaeche().Entferne(self)

    def Zeige(self):
        """Veraendernde Methode fuer die Sichtbarkeit mit Wert True"""
        self.s = True
        Zeichenflaeche.GibZeichenflaeche().Zeichne(self)
```

ohne Rückgabe

ohne Rückgabe

ohne Rückgabe

ohne Rückgabe

ohne Rückgabe

Fachausdrücke

In dieser Methode wird beschrieben, wie gezeichnet werden soll. Dazu wird beschrieben, wie ein Zeichenstift arbeiten müsste.

File Edit Format Run Options Window Help

```
def GibFigur(self):
    """definiert und transformiert die zu zeichnende Figur"""
    gc = Zeichenflaeche.GibZeichenflaeche().GibGC()
    path = gc.CreatePath()

    path.MoveToPoint(0, 0)
    path.AddLineToPoint(self.b, 0)
    path.AddLineToPoint(self.b*1.1, self.t)
    path.AddLineToPoint(-self.b*0.1, self.t)
    path.AddLineToPoint(0, 0)
    path.AddLineToPoint(0, -self.t*0.1)
    path.AddLineToPoint(self.b, -self.t*0.1)
    path.AddLineToPoint(self.b, 0)

    gc.PushState()
    gc.Translate(self.x+self.b/2, self.y+self.t/2)
    gc.Rotate(radians(self.w))
    gc.Translate(-self.b/2, -self.t/2)
    transformation = gc.GetTransform()
    gc.PopState()
    path.Transform(transformation)
    return path
```

Ln: 1 Col: 0

Stift bewegen ohne Zeichnen

Stift bewegen mit Zeichnen

Rückgabeobjekt

Fachausdrücke

Soll die Klasse Stuhl unabhängig von der Raumplaner-Anwendung getestet werden können, benötigt sie eine eigene App.

File Edit Format Run Options Window Help

```
### ------------------------------------------------
class StuhlApp(wx.App):
    """Test-Anwendung speziell fuer Stuhl"""
    def OnInit(self):
        self.fenster = GrafikFenster(None, "Raumplaner-Grafik")
        self.SetTopWindow(self.fenster)
        self.fenster.Show(True)
        self.fenster.panel.Refresh()
        self.fenster.ZeigeShellFrame()
        #self.fenster.ZeigeFillingFrame()
        self.TestAnwendung()
        return True

    def TestAnwendung(self):
        """Allein fuer die Testanwendung:"""
        global stuhl
        stuhl=Stuhl(True)

### ------------------------------------------------
if __name__ == '__main__':
    app = StuhlApp(redirect=False)
    # Parameterwert True, wenn Ausgaben in der Standard E/A angezeigt werden sollen
    app.MainLoop()
```

Ln: 1 Col: 0

Die Abfrage am Schluss verhindert den Aufruf des Konstruktors der StuhlApp, wenn stuhl.py importiert wird, also nicht das Hauptprogramm (main) ist.