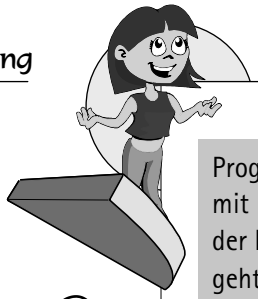


Programmentwicklung ist mit der IDLE (Python-GUI) sicherer. Du kannst mit ihr auch Grafik-Programme entwickeln, doch muss dann am Ende der Programme immer ein `mainloop()`-Aufruf stehen. Interaktive Grafik geht damit nicht.

Wenn du bei der Programmentwicklung Grafik interaktiv verwenden willst, wie wir es in diesem Buch stets gemacht haben, musst du die IDLE ohne Subprozess, also die IPI-SHELL, verwenden. Das bringt den Nachteil mit sich, dass man Programmfehler leichter übersehen kann. Ausweg: den IPI von Zeit zu Zeit neu starten!

Die Konfiguration der IDLE ohne Subprozess, – also die Konfiguration des IPI – ist in Anhang A beschrieben (Seite 430f.).

Anhang G

Das Modul turtle.py: Die Referenz

Das Turtle-Grafik-Modul `turtle.py` stellt Funktionen und Klassen bereit, die gestatten, es sowohl in prozeduraler wie auch in objektorientierter Weise zu verwenden.

In dieser Referenz werden alle *Funktionen* angegeben, mit denen du die Turtle oder das Grafik-Fenster steuern kannst.

All diese Funktionen können auch als *Methoden* einer Turtle oder als Methoden des Grafik-Fensters aufgerufen werden. Du kannst in deinen Programmen beliebig viele Turtle-Objekte verwenden, aber immer nur ein Grafik-Fenster. Dieses einzigartige Objekt hat in diesem Buch stets den Namen `screen`. (In der Computer-Fachsprache sagt man: `screen` ist ein Singleton.)

➢ Starte IPI-TURTLEGRAFIK und mach mit!

```
>>> from turtle import Screen, Turtle
>>> screen = Screen()
>>> screen.bgcolor("pink")
>>> tina = Turtle()
>>> tina.pensize(8)
>>> tina.color("red", "orange")
```

Wenn du mit nur einer (namenlosen) Turtle und mit Turtle-Grafik-Funktionen arbeiten willst, musst du alle Funktionen importieren:

```
from turtle  import *
```

Wenn du mit `Turtle`-Objekten, dem `screen`-Objekt und mit ihren Methoden arbeiten willst, genügt es, mit folgender `import`-Anweisung die beiden Klassen zu importieren:

```
from turtle import Screen, Turtle
```

Im Folgenden werden zunächst alle Funktionen beschrieben, mit denen du die Turtle steuern kannst. Sie leiten sich von entsprechenden, meist gleichnamigen Turtle-Methoden her und sind nach ihren Aufgaben in Gruppen zusammengefasst. Danach werden die Funktionen für die Steuerung des Grafik-Fensters beschrieben. Diese leiten sich von Methoden des `screen`-Objekts her.

Für alle Funktionen (und Methoden), die du importiert hast, kannst du eine ausführlichere Erklärung in der PYTHON-SHELL mit der Funktion `help()` erhalten. (Siehe Kapitel 2, Seite 54.) Zum Beispiel:

```
>>> help(goto)
>>> help(tina.pencolor)
>>> help(screen.onkeypress)
```

I. Funktionen für die Kontrolle der Turtle

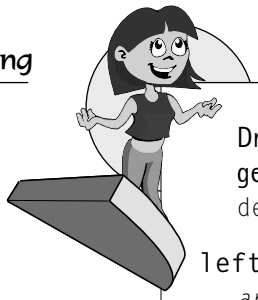
1. Turtle-Bewegung

1.a Bewegen und zeichnen

`forward(`*`distance`*`)` | `fd`
`distance`: eine Zahl
Bewegt die Turtle um die Strecke *`distance`* (in Pixel) vorwärts, und zwar in die Richtung, in die sie zeigt.

`back(`*`distance`*`)` | `bk` | `backward`
`distance`: eine Zahl
Bewegt die Turtle um die Strecke *`distance`* (in Pixel) rückwärts.

`right(`*`angle`*`)` | `rt`
`angle`: eine Zahl (ein Winkel)



Dreht die Turtle um *angle* Winkeleinheiten nach rechts. (Voreingestellte Winkeleinheit ist Grad; das kann über die Funktionen `degrees()` und `radians()` geändert werden.)

`left(`*angle*`)` | `lt`

angle: eine Zahl (ein Winkel)

Dreht die Turtle um *angle* Winkeleinheiten nach links. (Siehe auch: `right()`)

`goto(`*x, y=None*`)` | `setpos` | `setposition`

Argumente: *x*, *y*: zwei Zahlen *oder*
x: ein Paar/Vektor von zwei Zahlen (*y = None*)

Bewegt die Turtle zum Punkt mit den absoluten Koordinaten (*x*,*y*). Wenn der Zeichenstift unten ist, wird eine Strecke gezeichnet. Die Orientierung der Turtle wird nicht geändert.

`setx(`*x*`)`

x: eine Zahl

Setzt die erste Koordinate der Turtle auf *x*. Die zweite Koordinate bleibt unverändert.

`sety(`*y*`)`

y: eine Zahl

Setzt die zweite Koordinate der Turtle auf *y*. Die erste Koordinate bleibt unverändert.

`setheading(`*to_angle*`)` | `seth`

to_angle: eine Zahl (ein Winkel)

Dreht die Turtle so, dass sie in Richtung des angegebenen Winkels *to_angle* orientiert ist.

Hier sind einige gebräuchliche Richtungen in Grad:

'standard'-mode:	'logo'-mode:
0 - ost	0 - nord
90 - nord	90 - ost
180 - west	180 - süd
270 - süd	270 – west

`circle(`*radius, extent=None, steps=None*`)`

radius: eine Zahl

extent (optional): eine Zahl (ein Winkel)

steps (optional): eine ganze Zahl

Zeichnet einen Kreis(bogen) mit gegebenem *radius*. Der Kreismittelpunkt ist *radius* Einheiten links von der Turtle, wenn *radius* > 0 ist, andernfalls rechts von der Turtle. Der Winkel *extent* gibt an, welcher Teil des Kreises gezeichnet wird. Wenn *extent* fehlt, wird ein voller

Kreis gezeichnet. Kreise zeichnet die Turtle als Vielecke mit vielen Ecken, so dass sie rund ausschauen. Die Zahl *steps* legt fest, wie viele Ecken berechnet werden. Wenn sie nicht angegeben wird, wird sie automatisch optimal berechnet.

`dot(`*size=None, *color*`)`

size: eine Zahl größer oder gleich 1

color: ein Farbstring oder ein numerisches RGB-Farbtupel

Zeichnet einen Punkt mit dem Durchmesser *size*, in der Farbe *color*. Wenn *color* nicht angegeben wird, wird die Turtle-Farbe verwendet. Wenn auch *size* nicht angegeben wird, wird *pensize()+4* verwendet.

`home()`

Bewegt die Turtle zur Ausgangsposition und richtet sie in die Ausgangsorientierung (abhängig vom mode) aus.

`stamp()`

Erzeugt einen »Stempelabdruck« der Turtle-Form auf der Zeichenfläche an der aktuellen Turtle-Position und gibt eine `id` für diesen Abdruck zurück, so dass mit `clearstamp(id)` dieser Abdruck wieder gelöscht werden kann.

`clearstamp(`*stamp_id*`)`

stamp_id: Zahl, Rückgabewert eines `stamp()`-Aufrufs

Löscht den Stempelabdruck mit der angegebenen *stamp_id*.

`clearstamps(`*n=None*`)`

n: positive oder negative ganze Zahl

Wenn *n* nicht angegeben wird, werden alle Abdrücke der Turtle gelöscht. Wenn *n* > 0 ist, werden die ersten *n* Abdrücke gelöscht, wenn *n* < 0 ist, die letzten n.

`undo()`

Macht die letzte Turtle-Aktion rückgängig. Kann wiederholt aufgerufen werden. Die Anzahl der Aktionen, die rückgängig gemacht werden können, hängt von der Größe des `undo`-Puffers ab. (Kann in der Konfigurationsdatei `turtle.cfg` eingestellt werden.)

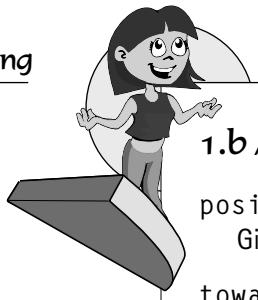
`speed(`*speed=None*`)`

speed: ganze Zahl im Bereich `0 .. 10`, mit folgender Bedeutung:

`0`: keine Animation, Turtle »springt« zum Zielpunkt.

`1 .. 10`: langsam ... schnell

Setzt die Geschwindigkeit der Turtle auf *speed* oder gibt deren Wert zurück.



1.b Abfrage des Zustands der Turtle

`position()` | `pos`
Gibt die aktuelle Position der Turtle zurück: `(x,y)` (als Vektor)

`towards(`*`x, y=None`*`)`
Argumente:
x, *y*: zwei Zahlen (Koordinaten eines Punktes) *oder*
x: ein Paar oder Vektor von zwei Zahlen (und *y=None*) *oder*
x: ein Turtle-Objekt (eine Turtle) (und *y=None*)
Gibt den Winkel zwischen der Linie von der Turtle-Position zum angegebenen Punkt und der Startorientierung der Turtle zurück (je nach Modus – `"standard"` oder `"logo"`).

`xcor()`
Gibt die x-Koordinate der Turtle zurück.

`ycor()`
Gibt die y-Koordinate der Turtle zurück.

`heading()`
Gibt die aktuelle Orientierung der Turtle zurück.

`distance(`*`x, y=None`*`)`
Argumente:
x, *y*: zwei Zahlen *oder*
x: ein Paar oder Vektor von zwei Zahlen (und *y=None*) *oder*
x: ein Turtle-Objekt (eine Turtle) (und *y=None*)
Gibt die Entfernung des Punktes (*x*, *y*) von der Turtle zurück.

1.c Einstellungen und Maßeinheiten

`degrees(`*`fullcircle=360.0`*`)`
Setzt die Einheit für die Winkelmessung auf Grad. Optionales Argument: *fullcircle* ist Anzahl der 'Grade' eines vollen Winkels, Standardwert ist 360.0).

`radians()`
Setzt die Einheit für die Winkelmessung auf Radian (Bogenmaß).

2. Steuerung des Zeichenstiftes

2.a Eigenschaften für das Zeichnen

`pendown()` | `pd` | `down`
Setzt den Zeichenstift nach unten. Zeichnet bei nachfolgenden Bewegungen.

`penup()` | `pu` | `up`
Hebt den Zeichenstift an. Zeichnet nicht bei nachfolgenden Bewegungen.

`pensize(`*`width=None`*`)` | `width`
`width`: positive Zahl
Setzt die Strichdicke auf *`width`*, wenn *`width`* angegeben ist; andernfalls wird die Strichdicke zurückgegeben. Wenn `resizemode` auf `'auto'` gestellt ist und die Turtleshape ein Polygon ist, dann wird das Polygon auch mit der Strichdicke *`width`* dargestellt.

`pen(`*`pen=None, **pendict`*`)`
`pen`: ein Dictionary mit folgenden Schlüssel-Wert-Paaren:

```
"shown"         :   True/False
"pendown"       :   True/False
"pencolor"      :   Farbstring oder Farb-Zahlentripel
"fillcolor"     :   Farbstring oder Farb-Zahlentripel
"pensize"       :   positive Zahl
"speed"         :   ganze Zahl im Bereich 0..10
"resizemode"    :   "auto" or "user" or "noresize"
"outline"       :   positive Zahl
"stretchfactor" :   (positive Zahl, positive Zahl)
"tilt"          :   Zahl (integer oder float)
"shearfactor"   :   positive Zahl
```

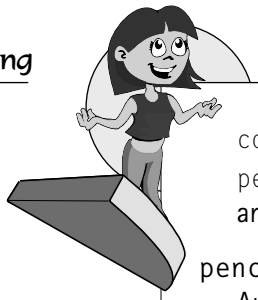
Setzt die Eigenschaften des Zeichenstiftes entsprechend des als Argument übergebenen Dictionarys *`pen`* und/oder der an *`**pendict`* übergebenen Schlüsselwort-Argumente. Wenn keine Argumente angegeben werden, wird das *`pen`*-Dictionary zurückgegeben.

`isdown()`
Gibt `True` zurück, falls der Zeichenstift unten ist, `False` sonst.

2.b Farben einstellen und abfragen

Im Folgenden steht *`s`* stets für einen Farbstring (siehe Anhang D) und *`r`*, *`g`*, *`b`* für drei Zahlen zur numerischen Beschreibung des Rot-, Grün- und Blauanteils von RGB-Farben. *`r`*, *`g`* und *`b`* müssen im Bereich `0 .. colormode` liegen. Die Farbe Lachsrosa etwa wird nach `colormode(1)` durch das Tripel `(1.0, 0.5, 0.25)` angegeben, nach `colormode(255)` durch das Tripel `(128, 64, 32)`.

`color(`*`*args`*`)`
Es gibt Aufruf-Formate mit 0, 1, 2, 3 oder 6 Argumenten.
`color(`*`s`*`)`, `color(`*`(r,g,b)`*`)`, `color(`*`r,g,b`*`)`: Setzt die Zeichenfarbe und die Füllfarbe auf den gegebenen Wert.
`color(`*`s1, s2`*`)`, `color(`*`(r1,g1,b1), (r2,g2,b2)`*`)` *oder*



`color(r1, g1, b1, r2, g2, b2)`: Gleichwertig mit `pencolor(s1)` und `fillcolor(s2)` – und entsprechend für die anderen Formate.

`pencolor(*args)`
Aufruf-Formate: `pencolor()` *oder*
`pencolor(s)`, `pencolor((r,g,b))`, `pencolor(r,g,b)`
Setzt die Zeichenfarbe oder gibt ihren Wert zurück, wenn kein Argument übergeben wird.

`fillcolor(*args)`
Aufruf-Formate: `fillcolor()` *oder*
`fillcolor(s)`, `fillcolor((r,g,b))`, `fillcolor(r,g,b)`
Setzt die Füllfarbe oder gibt ihren Wert zurück, wenn kein Argument übergeben wird.

2.c Füllen

`filling()`
Gibt den Füll-Zustand ab. Gibt `True` zurück, wenn Füllen mit `begin_fill()` eingeschaltet wurde, `False` sonst.

Infolge eines Bugs steht `filling()` in Python 3.1.1 nur als Methode der Klasse `Turtle` zur Verfügung, nicht aber als Funktion.

`begin_fill()`
Beginnt das Zeichnen einer Figur, die gefüllt werden soll.

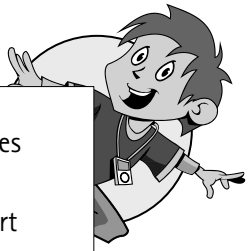
`end_fill()`
Füllt die Figur, die nach dem Aufruf von `begin_fill()` gezeichnet wurde, mit der Füllfarbe.

2.d Weitere Funktionen für die Grafik

`reset()`
Löscht die Zeichnungen der Turtle vom Grafik-Fenster. Setzt die Turtle in den Mittelpunkt des Fensters und alle Attribute auf ihre Anfangswerte (mit Ausnahme der Turtle-Gestalt).

`clear()`
Löscht die Zeichnungen der Turtle vom Grafik-Fenster. Die Turtle wird nicht bewegt, Zeichnungen anderer Turtles werden nicht beeinflusst.

```
write(arg,move=False,align='left',
              font=('Arial',8,'normal'))
```
arg: String, der geschrieben werden soll
move: `True` oder `False`



align: 'left', 'center' oder 'right' zur Ausrichtung des Textes
font: ein Dreiertupel zur Beschreibung der Schriftart
Schreibt Text an die aktuelle Turtle-Position, entsprechend dem Wert von *align* in der für *font* angegebenen Schriftart. Wenn *move* True ist, wird die Turtle zum rechten unteren Ende des Textes bewegt.

3. Zustand der Turtle

showturtle() | st
Macht die Turtle sichtbar.

hideturtle() | ht
Macht die Turtle unsichtbar.

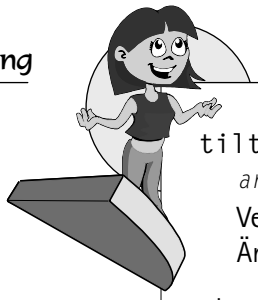
isvisible() | shownp
Gibt True zurück, wenn die Turtle sichtbar ist, False sonst.

shape(*name=None*)
name: ein String, der eine registrierte Turtleshape bezeichnet.
Setzt die Gestalt der Turtle auf die Gestalt mit dem angegebenen *name*.

resizemode(*rmode=None*)
rmode: "auto", "user", "noresize"
Setzt resizemode auf einen der drei Werte mit folgender Wirkung:
"auto": Turtle-Größe passt sich automatisch der pensize an.
"user": Turtle-Größe wird vom Benutzer mit shapesize() festgelegt.
"noresize": keine Größenänderung der Turtle
Wenn kein Argument angegeben ist, wird der aktuelle resizemode zurückgegeben.
Anmerkung: Wird shapesize() mit Argumenten aufgerufen, so setzt dies automatisch den resizemode auf den Wert "user".

shapesize(*stretch_wid=None, stretch_len=None, outline=None*) | turtlesize
stretch_wid, stretch_len, outline: drei positive Zahlen
Setzt Streckfaktoren für die Turtle-Shape und die Randstärke oder gibt diese zurück. *stretch_len* betrifft Streckung in die Länge, *stretch_wid* die in die Breite.

tiltangle(*angle=None*)
angle: eine Zahl (ein Winkel)
Verdreht die Turtle*shape* auf *angle* bezogen auf ihre Orientierung oder gibt den Verdrehungswinkel *angle* zurück. Ändert *nicht* die Orientierung der Turtle.



`tilt(angle)`
`angle`: eine Zahl (ein Winkel)
Verdreht die Turtle um `angle` bezogen auf ihren aktuellen Tilt-Winkel. Ändert *nicht* die Orientierung der Turtle.

`shearfactor(shear=None)`
`shear`: eine Zahl (ein Winkel)
Setzt den Scherungsfaktor für die Turtle-Shape oder gibt ihn zurück, wenn `shear` nicht angegeben ist. Die Turtle-Gestalt wird einer Scherung unterworfen, wobei `shear` der Tangens des Scherungswinkels ist. Ändert *nicht* die Orientierung der Turtle.

`shapetransform(t11=None, t12=None, t21=None, t22=None)`
`t11, t12, t21, t22` sind vier Zahlen, die eine Transformationsmatrix festlegen. Wenn alle vier nicht angegeben sind, wird die Transformationsmatrix als Vierertupel zurückgegeben. Die Turtle-Shape wird entsprechend dieser Matrix transformiert und die Streckfaktoren, der Verdrehungswinkel und der Scherungsfaktor werden entsprechend eingestellt. Ändert *nicht* die Orientierung der Turtle.

`get_shapepoly()`
Gibt die aktuelle Turtle-Form als Tupel von Koordinatenpaaren zurück.

4. Turtle-Ereignisse

`onclick(fun, btn=1)`
`fun`: Funktion mit 2 Argumenten
`btn`: 1 oder 2 oder 3 für die drei Maustasten
Bindet den Aufruf von `fun` an das Mausklick-Ereignis auf die Turtle. Die Koordinaten des angeklickten Punktes werden beim Aufruf von `fun` als Argumente übergeben.

`onrelease(fun, btn)`
`fun`, `btn`: wie bei `onclick()`
Bindet den Aufruf von `fun` an das Ereignis »Maustaste loslassen auf der Turtle«. Die Koordinaten des angeklickten Punktes werden beim Aufruf an `fun` als Argumente übergeben.

`ondrag(fun, btn)`
`fun`, `btn`: wie bei `onclick()`
In der Folge wird `fun` während des »Ziehens« der Turtle (mit gedrückter Maustaste `btn`) wiederholt aufgerufen. Die Koordinaten der aktuellen Mausposition werden jeweils an `fun` als Argumente übergeben.

5. Spezielle Turtle-Funktionen

`begin_poly()`
Startet die Aufzeichnung der Eckpunkte eines Polygons mit der aktuellen Turtle-Position.

`end_poly()`
Beendet die Aufzeichnung der Eckpunkte eines Polygons mit der aktuellen Turtle-Position. Dieser letzte Punkt wird mit dem ersten verbunden, wenn das Polygon gezeichnet wird.

`get_poly()`
Gibt das zuletzt aufgezeichnete Polygon zurück.

`clone()`
Erzeugt einen Klon (eine exakte Kopie) der Turtle mit gleicher Position, Orientierung und Eigenschaften und gibt diesen zurück.

`getturtle() | getpen`
Gibt das Turtle-Objekt selbst zurück. Einzige sinnvolle Verwendung: als Funktion, die die »namenlose Turtle« zurückgibt.

`getscreen()`
Gibt das TurtleScreen-Objekt zurück, auf dem die Turtles zeichnen.

`setundobuffer(`*`size`*`)`
`size`: eine positive Zahl oder `None`
Stellt die Größe des `undo`-Puffers ein. Falls *`size`* gleich `None` ist, wird der `undo`-Buffer ausgeschaltet.

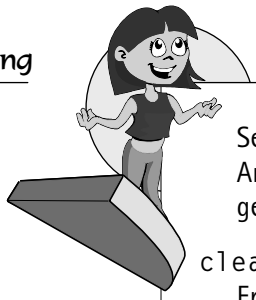
`undobufferentries()`
Gibt die Anzahl der Einträge im `undo`-Puffer zurück.

II. Funktionen für die Kontrolle des Turtle-Grafik-Fensters

6. Fenster-Aktionen

`bgcolor(`*`*args`*`)`
`args`: ein Farbstring oder ein Farbtupel
Setzt die Hintergrundfarbe des Grafik-Fensters oder gibt sie zurück.

`bgpic(`*`picname=None`*`)`
`picname`: String. Name einer `*.gif`-Datei oder `"nopic"`



Setzt ein Hintergrundbild für das Grafik-Fenster bzw. entfernt es. Ohne Argument wird der Name der aktuell verwendeten Grafik-Datei zurückgegeben.

`clearscreen()`
Entfernt alle Zeichnungen und alle Turtles vom Grafik-Fenster und setzt das leere Grafik-Fenster auf seinen Anfangszustand.
Als Methode auch mit dem Namen `clear` verfügbar.

`resetscreen()`
Setzt alle Turtles im Grafik-Fenster auf ihren Anfangszustand.
Als Methode auch mit dem Namen `reset` verfügbar.

`screensize(`*`canvwidth=None, canvheight=None, bg=None`*`)`
`canvwidth, canvheight`: zwei positive Zahlen
`bg`: Farbstring oder Farbtupel
Stellt die Größe der mittels Scrollbars erreichbaren Zeichenfläche ein, auf der die Turtles zeichnen, oder gibt ein Tupel `(widht, height)` zurück. (Die Fenster-Größe wird nicht verändert.)

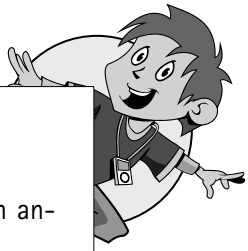
`setworldcoordinates(`*`llx, lly, urx, ury`*`)`
`llx, lly, urx, ury`: vier Zahlen
`llx, lly` sind die Benutzerkoordinaten der linken unteren Ecke, *`urx, ury`* die der oberen rechten Ecke.
Stellt ein benutzerdefiniertes Koordinatensystem ein. Falls nötig, stellt es auf den Modus `"world"` um. Wenn Modus `"world"` aktiv ist, werden alle vorhandenen Zeichnungen entsprechend dem neuen Koordinatensystem aktualisiert.

7. Kontrolle der Animation

`tracer(`*`flag=None, delay=None`*`)` **! Methode des Grafik-Fensters !**
flag: `True` oder `False`
Schaltet Turtle-Animation ein (*`flag`*`=True`) bzw. aus (*`flag`*`=False`).

`update()`
Führt ein Update des Grafik-Fensters aus. Speziell nützlich, wenn `tracer(False)` gesetzt ist.

`delay(`*`delay=None`*`)`
`delay`: positive ganze Zahl
Setzt – oder gibt zurück – das Zeitintervall zwischen zwei aufeinanderfolgenden Neuzeichnungen des Canvas in Millisekunden. Je größer *`delay`*, desto langsamer läuft die Animation ab.



8. Ereignisse des Grafik-Fensters

Im Folgenden werden für Maus-Ereignisse die Maustasten mit Zahlen angesprochen: 1 ist die linke, 2 die mittlere und 3 die rechte Maustaste.

`listen(`*`x=None, y=None`*`)`
Setzt den Fokus auf das Grafik-Fenster (siehe: `onkey()`).

`onkeyrelease(`*`fun, key=None`*`)` | `onkey`
`fun`: eine Funktion ohne Argumente
`key`: String-Kennung einer Taste, wie `"a"`,`"z"`,`"space"`,`"Escape"`
Bindet die Funktion *`fun`* an das Loslassen der Taste *`key`*. Das Grafik-Fenster muss den Fokus haben, damit `key`-Ereignisse registriert werden (siehe `listen()`).

`onkeypress(`*`fun, key=None`*`)` | `onkeyrelease`
`fun`: eine Funktion ohne Argumente
`key`: wie bei `onkeyrelease`
Bindet die Funktion *`fun`* an das Loslassen der Taste *`key`*. Das Grafik-Fenster muss den Fokus haben (siehe `listen()`).

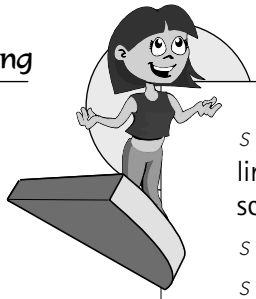
`onscreenclick(fun, btn=1)`
`fun`: Funktion mit 2 Argumenten
`btn`: 1 oder 2 oder 3 für die Maustasten
Bindet den Aufruf von *`fun`* an das Mausklick-Ereignis ins Grafik-Fenster mit der entsprechenden Maustaste. Die Koordinaten des angeklickten Punktes werden beim Aufruf an *`fun`* als Argumente übergeben.
Anmerkung: Als Methode auch mit dem Namen `onclick()` verfügbar.

`ontimer(`*`fun, t=0`*`)`
`fun`: Funktion ohne Argumente
`t`: positive ganze Zahl
Setzt eine Uhr in Gang, die die Funktion *`fun`* nach *`t`* Millisekunden aufruft.

`mainloop()`
Setzt die Ereignis-Schleife in Gang. Muss die letzte Anweisung in einem ereignisorientierten Turtle-Grafik-Programm sein.

9. Einstellungen und spezielle Funktionen

`setup(`*`width=0.5, height=0.75, startx=None, starty=None`*`)`
`width`, *`height`*: ganze Zahlen (Breite bzw. Höhe in Pixel), **oder** Kommazahlen <= 1.0, Bruchteil der Bildschirmbreite / -höhe



startx, *starty*: Zahlen. Wenn positiv, Startposition in Pixel vom linken/oberen Bildschirmrand. Wenn negativ vom rechten/unteren Bildschirmrand.

startx=None zentriert das Grafik-Fenster horizontal,

starty=None zentriert das Grafik-Fenster vertikal am Bildschirm.

mode(*mode*=*None*)

mode: die Zeichenkette 'standard', 'logo' oder 'world'

Setzt den Turtle-Modus auf 'standard' oder 'logo' und führt ein reset() aus. Modus 'standard' ist kompatibel mit turtle.py. Modus 'logo' ist kompatibel mit den meisten Turtle-Grafik-Implementierungen in Logo. Modus 'world' verwendet benutzerdefinierte Koordinaten. (Siehe setworldcoordinates())

Modus	Anfängliche Orientierung der Turtle	positive Winkel
'standard'	nach rechts (Osten)	im Gegenuhrzeigersinn
'logo'	nach oben (Norden)	im Uhrzeigersinn
'world'	unwesentlich (nach rechts)	im Gegenuhrzeigersinn

In diesem Buch wird ausschließlich der Modus 'logo' verwendet!

colormode(*cmode*=*None*)

cmode: 1.0 oder 255

Setzt colormode auf 1.0 oder 255. Ohne Argument gibt es den colormode zurück.

getcanvas()

Gibt das Canvas-Objekt zurück, auf dem die Turtles zeichnen.

getshapes()

Gibt eine Liste der Namen der aktuell registrierten Turtle-Shapes zurück.

register_shape(*name*, *shape*=None) | addshape

name: ein String

shape: ein Polygon oder ein Objekt der Klasse Shape

Füge eine Turtle-Form zum shape-Dictionary des Grafik-Fensters hinzu. Folgende Fälle sind möglich:

(1) *name* ist der Dateiname einer gif-Datei und *shape* ist None: Meldet das entsprechende Bild als Turtle-Gestalt an.

(2) *name* ist eine beliebige Zeichenkette und *shape* ist ein Tupel von Koordinaten-Paaren. Registriert die entsprechende Polygon-Form.

(3) *name* ist eine beliebige Zeichenkette und *shape* ist ein zusammengesetztes *Shape*-Objekt. Registriert die entsprechend zusammengesetzte Turtle-Form.

`turtles()`
Gibt eine Liste aller Turtles in dem Grafik-Fenster zurück.

`title(`*titlestring*`)`
titlestring : ein String, der im Titelbalken des Grafik-Fensters erscheint
Setzt den *title* des Turtle-Grafik-Fensters auf *titlestring*.

`window_height()`
Gibt die Höhe des Turtle-Grafik-Fensters zurück.

`window_width()`
Gibt die Breite des Turtle-Grafik-Fensters zurück.

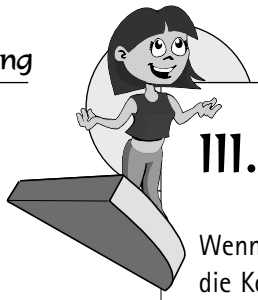
`bye()`
Schließt das Turtle-Grafik-Fenster.

`exitonclick()`
Bindet `bye()` an Mausklicks auf das Grafik-Fenster.

10. Eingabe-Funktionen

`textinput(`*title, prompt*`)`
title, prompt: zwei Strings
Öffnet grafischen Eingabedialog für einen String. *title* ist der Titel des Fensters, *prompt* beschreibt, was einzugeben ist.

`numinput(`*title, prompt, default=None, minval=None, maxval=None*`)`
title, prompt: zwei Strings wie bei `textinput()`
default: Standard-Vorgabewert
minval, maxval: minimaler und maximaler erlaubter Eingabewert
Öffnet grafischen Eingabedialog für eine Zahleneingabe.
Methoden des Grafik-Fensters sind Methoden dieses Objekts.



III. Die Datei »turtle.cfg«

Wenn `turtle.py` importiert oder geladen wird, liest das Modul zunächst die Konfigurations-Datei `turtle.cfg`. Das ist eine Textdatei, in der einige Einstellungen festgelegt werden. Für das Buch »Python für Kids« hat sie folgenden Inhalt:

```
width = 400
height = 300
canvwidth = 360
canvheight = 270
shape = turtle
mode = logo
importvec = False
language = german
```

Du kannst diese Datei editieren und so das Modul `turtle` deinen Bedürfnissen anpassen.

- `width` und `height` legen die Größe des Grafik-Fensters fest.
- `canvwidth` und `canvheight` legen die Größe der Zeichenfläche fest. Du kannst sie größer als das Fenster wählen und bekommst dann Scrollbars an den Fensterkanten, mit denen du die Zeichenfläche verschieben kannst.
- `shape` legt die Gestalt der Turtle fest. Änderst du den Eintrag beispielsweise auf `shape = arrow`, so startet das Turtle-Grafik-Fenster immer mit einer Turtle in Dreiecksgestalt.

Weitergehende Informationen über den Gebrauch von `turtle.cfg` und das Modul `turtle.py` im Allgemeinen findest du auf der Seite *http://python4kids.at*.