

This documentation was generated from the Python documentation available by typing

```
from turtle import *  
help (TurtleScreen)
```

in the Python shell. In the documentation found here the variable *turtle* is a reference to a Turtle object and *screen* is a reference to the *TurtleScreen* object. This is a subset of that documentation. The official Python 3 documentation is at <http://docs.python.org/3/>.

Method	Description
screen.addshape(name)	Same thing as <code>screen.register_shape(name)</code>
screen.bgcolor(*args)	Set or return backgroundcolor of the TurtleScreen. Arguments (if given): a color string or three numbers in the range 0..colormode or a 3-tuple of such numbers. Example (for a TurtleScreen instance named screen): <pre>>>> screen.bgcolor("orange") >>> screen.bgcolor() 'orange' >>> screen.bgcolor(0.5,0,0.5) >>> screen.bgcolor() '#800080'</pre>

Method	Description
screen.bgpic(picname=None)	<p>Set background image or return name of current backgroundimage.</p> <p>Optional argument: picname – a string, name of a gif-file or "nopic".</p> <p>If picname is a filename, set the corresponding image as background. If picname is "nopic", delete backgroundimage, if present. If picname is None, return the filename of the current backgroundimage.</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>>>> screen.bgpic() 'nopic' >>> screen.bgpic("landscape.gif") >>> screen.bgpic() 'landscape.gif'</pre>
screen.clear()	<p>Delete all drawings and all turtles from the TurtleScreen.</p> <p>Reset empty TurtleScreen to its initial state: white background, no backgroundimage, no eventbindings and tracing on.</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>screen.clear()</pre> <p>Note: this method is not available as function.</p>
screen.colormode(cmode=None)	<p>Return the colormode or set it to 1.0 or 255.</p> <p>Optional argument: cmode – one of the values 1.0 or 255</p> <p>r, g, b values of colortriples have to be in range 0..cmode.</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>>>> screen.colormode() 1.0 >>> screen.colormode(255) >>> turtle.pencolor(240,160,80)</pre>
screen.delay(delay=None)	<p>Return or set the drawing delay in milliseconds.</p> <p>Optional argument: delay – positive integer</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>>>> screen.delay(15) >>> screen.delay() 15</pre>

Method	Description
screen.getcanvas()	<p>Return the Canvas of this TurtleScreen.</p> <p>Example (for a Screen instance named screen):</p> <pre>>>> cv = screen.getcanvas() >>> cv <turtle.ScrolledCanvas instance at 0x010742D8></pre>
screen.getshapes()	<p>Return a list of names of all currently available turtle shapes.</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>>>> screen.getshapes() ['arrow', 'blank', 'circle', ... , 'turtle']</pre>
screen.listen(xdummy=None, ydummy=None)	<p>Set focus on TurtleScreen (in order to collect key-events)</p> <p>Dummy arguments are provided in order to be able to pass listen to the onclick method.</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>>>> screen.listen()</pre>
screen.mode(mode=None)	<p>Set turtle-mode ('standard', 'logo' or 'world') and perform reset.</p> <p>Optional argument: mode – one of the strings 'standard', 'logo' or 'world'</p> <p>Mode 'standard' is compatible with turtle.py. Mode 'logo' is compatible with most Logo-Turtle-Graphics. Mode 'world' uses userdefined 'worldcoordinates'. *Attention*: in this mode angles appear distorted if x/y unit-ratio doesn't equal 1. If mode is not given, return the current mode.</p> <p>Mode Initial turtle heading positive angles</p> <hr/> <p>'standard' to the right (east) counterclockwise 'logo' upward (north) clockwise</p> <p>Examples:</p> <pre>>>> mode('logo') # resets turtle heading to north >>> mode() 'logo'</pre>

Method Description

<p>screen.onclick(fun, btn=1, add=None)</p>
--

Bind fun to mouse-click event on canvas.

Arguments:

fun – a function with two arguments, the coordinates of the clicked point on the canvas.

num – the number of the mouse-button, defaults to 1

Example (for a TurtleScreen instance named screen and a Turtle instance named turtle):

```
>>> screen.onclick(turtle.goto)
```

```
### Subsequently clicking into the TurtleScreen will
### make the turtle move to the clicked point.
```

```
>>> screen.onclick(None)
```

```
### event-binding will be removed
```

<p>screen.onkey(fun, key)</p>

Bind fun to key-release event of key.

Arguments:

fun – a function with no arguments

key – a string: key (e.g. "a") or key-symbol (e.g. "space")

In order to be able to register key-events, TurtleScreen Must have focus. (See method listen.)

Example (for a TurtleScreen instance named screen and a Turtle instance named turtle):

```
>>> def f():
    turtle.fd(50)
    turtle.lt(60)
```

```
>>> screen.onkey(f, "Up")
```

```
>>> screen.listen()
```

```
### Subsequently the turtle can be moved by
### repeatedly pressing the up-arrow key,
### consequently drawing a hexagon
```

Method	Description
screen.onkeypress(fun, key=None)	<p>Bind fun to key-press event of key if key is given, or to any key-press-event if no key is given.</p> <p>Arguments: fun – a function with no arguments key – a string: key (e.g. "a") or key-symbol (e.g. "space")</p> <p>In order to be able to register key-events, TurtleScreen must have focus. (See method listen.)</p> <p>Example (for a TurtleScreen instance named screen and a Turtle instance named turtle):</p> <pre>>>> def f(): turtle.fd(50) >>> screen.onkey(f, "Up") >>> screen.listen() ### Subsequently the turtle can be moved by ### repeatedly pressing the up-arrow key, ### or by keeping pressed the up-arrow key. ### consequently drawing a hexagon.</pre>
screen.ontimer(fun, t=0)	<p>Install a timer, which calls fun after t milliseconds.</p> <p>Arguments: fun – a function with no arguments. t – a number >= 0</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>>>> running = True >>> def f(): if running: turtle.fd(50) turtle.lt(60) screen.ontimer(f, 250) >>> f() ### makes the turtle marching around >>> running = False</pre>

Method Description
<p>screen.register_shape(name, shape=None) Adds a turtle shape to TurtleScreen's shapelist.</p> <p>Arguments:</p> <p>(1) name is the name of a gif-file and shape is None. Installs the corresponding image shape. !! Image-shapes DO NOT rotate when turning the turtle, !! so they do not display the heading of the turtle!</p> <p>(2) name is an arbitrary string and shape is a tuple of pairs of coordinates. Installs the corresponding polygon shape</p> <p>(3) name is an arbitrary string and shape is a (compound) Shape object. Installs the corresponding compound shape.</p> <p>To use a shape, you have to issue the command shape(shapename).</p> <p>call: register_shape("turtle.gif") –or: register_shape("tri", ((0,0), (10,10), (-10,10)))</p> <p>Example (for a TurtleScreen instance named screen): >>> screen.register_shape("triangle", ((5,-3),(0,5),(-5,-3)))</p>
<p>screen.reset() Reset all Turtles on the Screen to their initial state.</p> <p>Example (for a TurtleScreen instance named screen): >>> screen.reset()</p>
<p>screen.screensize(canvwidth=None, canvheight=None, bg=None) Resize the canvas the turtles are drawing on.</p> <p>Optional arguments:</p> <p>canvwidth – positive integer, new width of canvas in pixels canvheight – positive integer, new height of canvas in pixels bg – colorstring or color-tuple, new backgroundcolor If no arguments are given, return current (canvwidth, canvheight)</p> <p>Do not alter the drawing window. To observe hidden parts of the canvas use the scrollbars. (Can make visible those parts of a drawing, which were outside the canvas before!)</p> <p>Example (for a Turtle instance named turtle): >>> turtle.screensize(2000,1500) ### e. g. to search for an erroneously escaped turtle :-)</p>

Method	Description
screen.setworldcoordinates(llx, lly, urx, ury)	<p>Set up a user defined coordinate-system.</p> <p>Arguments:</p> <p>llx – a number, x-coordinate of lower left corner of canvas lly – a number, y-coordinate of lower left corner of canvas urx – a number, x-coordinate of upper right corner of canvas ury – a number, y-coordinate of upper right corner of canvas</p> <p>Set up user coodinat-system and switch to mode 'world' if necessary. This performs a screen.reset. If mode 'world' is already active, All drawings are redrawn according to the new coordinates.</p> <p>But ATTENTION: in user-defined coordinatesystems angles may appear distorted. (see Screen.mode())</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>>>> screen.setworldcoordinates(-10,-0.5,50,1.5) >>> for _ in range(36): turtle.left(10) turtle.forward(0.5)</pre>
screen.title(titlestr)	<p>Set the title of the Turtle Graphics screen. The title appears in the title bar of the window.</p>
screen.tracer(n=None, delay=None)	<p>Turns turtle animation on/off and set delay for update drawings.</p> <p>Optional arguments:</p> <p>n – nonnegative integer delay – nonnegative integer</p> <p>If n is given, only each n-th regular screen update is really performed. (Can be used to accelerate the drawing of complex graphics.) Second arguments sets delay value (see RawTurtle.delay())</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>>>> screen.tracer(8, 25) >>> dist = 2 >>> for i in range(200): turtle.fd(dist) turtle.rt(90) dist += 2</pre>
screen.turtles()	<p>Return the list of turtles on the screen.</p> <p>Example (for a TurtleScreen instance named screen):</p> <pre>>>> screen.turtles() [<turtle.Turtle object at 0x00E11FB0>]</pre>

Method	Description
screen.update()	Perform a TurtleScreen update.
screen.window_height()	Return the height of the turtle window. Example (for a TurtleScreen instance named screen): >>> screen.window_height() 480
screen.window_width()	Return the width of the turtle window. Example (for a TurtleScreen instance named screen): >>> screen.window_width() 640
screen.mainloop()	Starts event loop - calling Tkinter's mainloop function. Must be last statement in a turtle graphics program. Must NOT be used if a script is run from within IDLE in -n mode (No subprocess) - for interactive use of turtle graphics. Example (for a TurtleScreen instance named screen): >>> screen.mainloop()
screen.numinput(title, prompt, default=None, minval=None, maxval=None)	Pop up a dialog window for input of a number. Arguments: title is the title of the dialog window, prompt is a text mostly describing what numerical information to input. default: default value minval: minimum value for input maxval: maximum value for input The number input must be in the range minval .. maxval if these are given. If not, a hint is issued and the dialog remains open for correction. Return the number input. If the dialog is canceled, return None. Example (for a TurtleScreen instance named screen): >>> screen.numinput("Poker", "Your stakes:", 1000, minval=10, maxval=10000)
screen.textinput(title, prompt)	Pop up a dialog window for input of a string. Arguments: title is the title of the dialog window, prompt is a text mostly describing what information to input. Return the string input If the dialog is canceled, return None. Example (for a TurtleScreen instance named screen): >>> screen.textinput("NIM", "Name of first player:")