**About the author**



Don Watkins - Educator, education technology specialist,  entrepreneur, open source advocate. M.A. in Educational Psychology, MSED in Educational Leadership, Linux system administrator, CCNA, virtualization using Virtual Box. Follow me at [@Don\_Watkins .](https://twitter.com/Don_Watkins)

* [index](https://docs.python.org/2/genindex.html)
* [modules](https://docs.python.org/2/py-modindex.html) |
* [next](https://docs.python.org/2/library/idle.html) |
* [previous](https://docs.python.org/2/library/scrolledtext.html) |
* https://docs.python.org/2/_static/py.png
* [Python](https://www.python.org/) »
* [Documentation](https://docs.python.org/2/index.html) »
* [The Python Standard Library](https://docs.python.org/2/library/index.html) »
* [24. Graphical User Interfaces with Tk](https://docs.python.org/2/library/tk.html) »

### [Table Of Contents](https://docs.python.org/2/contents.html)

* [24.5. turtle — Turtle graphics for Tk](https://docs.python.org/2/library/turtle.html)
  + [24.5.1. Introduction](https://docs.python.org/2/library/turtle.html#introduction)
  + [24.5.2. Overview over available Turtle and Screen methods](https://docs.python.org/2/library/turtle.html#overview-over-available-turtle-and-screen-methods)
    - [24.5.2.1. Turtle methods](https://docs.python.org/2/library/turtle.html#turtle-methods)
    - [24.5.2.2. Methods of TurtleScreen/Screen](https://docs.python.org/2/library/turtle.html#methods-of-turtlescreen-screen)
  + [24.5.3. Methods of RawTurtle/Turtle and corresponding functions](https://docs.python.org/2/library/turtle.html#methods-of-rawturtle-turtle-and-corresponding-functions)
    - [24.5.3.1. Turtle motion](https://docs.python.org/2/library/turtle.html#turtle-motion)
    - [24.5.3.2. Tell Turtle’s state](https://docs.python.org/2/library/turtle.html#tell-turtle-s-state)
    - [24.5.3.3. Settings for measurement](https://docs.python.org/2/library/turtle.html#settings-for-measurement)
    - [24.5.3.4. Pen control](https://docs.python.org/2/library/turtle.html#pen-control)
      * [24.5.3.4.1. Drawing state](https://docs.python.org/2/library/turtle.html#drawing-state)
      * [24.5.3.4.2. Color control](https://docs.python.org/2/library/turtle.html#color-control)
      * [24.5.3.4.3. Filling](https://docs.python.org/2/library/turtle.html#filling)
      * [24.5.3.4.4. More drawing control](https://docs.python.org/2/library/turtle.html#more-drawing-control)
    - [24.5.3.5. Turtle state](https://docs.python.org/2/library/turtle.html#turtle-state)
      * [24.5.3.5.1. Visibility](https://docs.python.org/2/library/turtle.html#visibility)
      * [24.5.3.5.2. Appearance](https://docs.python.org/2/library/turtle.html#appearance)
    - [24.5.3.6. Using events](https://docs.python.org/2/library/turtle.html#using-events)
    - [24.5.3.7. Special Turtle methods](https://docs.python.org/2/library/turtle.html#special-turtle-methods)
    - [24.5.3.8. Excursus about the use of compound shapes](https://docs.python.org/2/library/turtle.html#excursus-about-the-use-of-compound-shapes)
  + [24.5.4. Methods of TurtleScreen/Screen and corresponding functions](https://docs.python.org/2/library/turtle.html#methods-of-turtlescreen-screen-and-corresponding-functions)
    - [24.5.4.1. Window control](https://docs.python.org/2/library/turtle.html#window-control)
    - [24.5.4.2. Animation control](https://docs.python.org/2/library/turtle.html#animation-control)
    - [24.5.4.3. Using screen events](https://docs.python.org/2/library/turtle.html#using-screen-events)
    - [24.5.4.4. Settings and special methods](https://docs.python.org/2/library/turtle.html#settings-and-special-methods)
    - [24.5.4.5. Methods specific to Screen, not inherited from TurtleScreen](https://docs.python.org/2/library/turtle.html#methods-specific-to-screen-not-inherited-from-turtlescreen)
  + [24.5.5. The public classes of the module turtle](https://docs.python.org/2/library/turtle.html#the-public-classes-of-the-module-turtle)
  + [24.5.6. Help and configuration](https://docs.python.org/2/library/turtle.html#help-and-configuration)
    - [24.5.6.1. How to use help](https://docs.python.org/2/library/turtle.html#how-to-use-help)
    - [24.5.6.2. Translation of docstrings into different languages](https://docs.python.org/2/library/turtle.html#translation-of-docstrings-into-different-languages)
    - [24.5.6.3. How to configure Screen and Turtles](https://docs.python.org/2/library/turtle.html#how-to-configure-screen-and-turtles)
  + [24.5.7. Demo scripts](https://docs.python.org/2/library/turtle.html#demo-scripts)

#### Previous topic

[24.4. ScrolledText — Scrolled Text Widget](https://docs.python.org/2/library/scrolledtext.html)

#### Next topic

[24.6. IDLE](https://docs.python.org/2/library/idle.html)

# 24.5. [turtle](https://docs.python.org/2/library/turtle.html#module-turtle) — Turtle graphics for Tk

## 24.5.1. Introduction

Turtle graphics is a popular way for introducing programming to kids. It was part of the original **Logo** programming language developed by Wally Feurzig and Seymour Papert in **1966**.

Imagine a **robotic turtle** starting at (0, 0) in the x-y plane. After an import turtle, give it the command turtle.forward(15), and it moves (on-screen!) 15 pixels in the direction it is facing, drawing a line as it moves. Give it the command turtle.right(25), and it rotates in-place 25 degrees clockwise.

By combining together these and similar commands, intricate shapes and pictures can easily be drawn.

The [turtle](https://docs.python.org/2/library/turtle.html#module-turtle) module is an extended reimplementation of the same-named module from the Python standard distribution up to version Python 2.5.

It tries to keep the merits of the old turtle module and to be (nearly) 100% compatible with it. This means in the first place to enable the learning programmer to use all the commands, classes and methods interactively when using the module from within IDLE run with the -n switch.

The turtle module provides turtle graphics primitives, in both object-oriented and procedure-oriented ways. Because it uses [Tkinter](https://docs.python.org/2/library/tkinter.html#module-Tkinter) for the underlying graphics, it needs a version of Python installed with **Tk support**.

The object-oriented interface uses essentially two+two classes:

1. The [TurtleScreen](https://docs.python.org/2/library/turtle.html#turtle.TurtleScreen) class defines graphics windows as a playground for the drawing turtles. Its constructor needs a Tkinter.Canvas or a [ScrolledCanvas](https://docs.python.org/2/library/turtle.html#turtle.ScrolledCanvas) as argument. It should be used when [turtle](https://docs.python.org/2/library/turtle.html#module-turtle) is used as part of some application.

The function [Screen()](https://docs.python.org/2/library/turtle.html#turtle.Screen) returns a singleton object of a [TurtleScreen](https://docs.python.org/2/library/turtle.html#turtle.TurtleScreen) subclass. This function should be used when [turtle](https://docs.python.org/2/library/turtle.html#module-turtle) is used as a standalone tool for doing graphics. As a singleton object, inheriting from its class is not possible.

All methods of TurtleScreen/Screen also exist as functions, i.e. as part of the procedure-oriented interface.

1. [RawTurtle](https://docs.python.org/2/library/turtle.html#turtle.RawTurtle) (alias: [RawPen](https://docs.python.org/2/library/turtle.html#turtle.RawPen)) defines Turtle objects which draw on a [TurtleScreen](https://docs.python.org/2/library/turtle.html#turtle.TurtleScreen). Its constructor needs a Canvas, ScrolledCanvas or TurtleScreen as argument, so the RawTurtle objects know where to draw.

Derived from RawTurtle is the subclass [Turtle](https://docs.python.org/2/library/turtle.html#turtle.Turtle) (alias: Pen), which draws on “the” [Screen](https://docs.python.org/2/library/turtle.html#turtle.Screen) - instance which is automatically created, if not already present.

All methods of RawTurtle/Turtle also exist as functions, i.e. part of the procedure-oriented interface.

The procedural interface provides functions which are derived from the methods of the classes [Screen](https://docs.python.org/2/library/turtle.html#turtle.Screen) and [Turtle](https://docs.python.org/2/library/turtle.html#turtle.Turtle). They have the same names as the corresponding methods. A screen object is automatically created whenever a function derived from a Screen method is called. An (unnamed) turtle object is automatically created whenever any of the functions derived from a Turtle method is called.

To use multiple turtles an a screen one has to use the object-oriented interface.

Note

In the following documentation the argument list for functions is given. Methods, of course, have the additional first argument self which is omitted here.

## 24.5.2. Overview over available Turtle and Screen methods

### 24.5.2.1. Turtle methods

Turtle motion

Move and draw

[forward()](https://docs.python.org/2/library/turtle.html#turtle.forward) | [fd()](https://docs.python.org/2/library/turtle.html#turtle.fd)

[backward()](https://docs.python.org/2/library/turtle.html#turtle.backward) | [bk()](https://docs.python.org/2/library/turtle.html#turtle.bk) | [back()](https://docs.python.org/2/library/turtle.html#turtle.back)

[right()](https://docs.python.org/2/library/turtle.html#turtle.right) | [rt()](https://docs.python.org/2/library/turtle.html#turtle.rt)

[left()](https://docs.python.org/2/library/turtle.html#turtle.left) | [lt()](https://docs.python.org/2/library/turtle.html#turtle.lt)

[goto()](https://docs.python.org/2/library/turtle.html#turtle.goto) | [setpos()](https://docs.python.org/2/library/turtle.html#turtle.setpos) | [setposition()](https://docs.python.org/2/library/turtle.html#turtle.setposition)

[setx()](https://docs.python.org/2/library/turtle.html#turtle.setx)

[sety()](https://docs.python.org/2/library/turtle.html#turtle.sety)

[setheading()](https://docs.python.org/2/library/turtle.html#turtle.setheading) | [seth()](https://docs.python.org/2/library/turtle.html#turtle.seth)

[home()](https://docs.python.org/2/library/turtle.html#turtle.home)

[circle()](https://docs.python.org/2/library/turtle.html#turtle.circle)

[dot()](https://docs.python.org/2/library/turtle.html#turtle.dot)

[stamp()](https://docs.python.org/2/library/turtle.html#turtle.stamp)

[clearstamp()](https://docs.python.org/2/library/turtle.html#turtle.clearstamp)

[clearstamps()](https://docs.python.org/2/library/turtle.html#turtle.clearstamps)

[undo()](https://docs.python.org/2/library/turtle.html#turtle.undo)

[speed()](https://docs.python.org/2/library/turtle.html#turtle.speed)

Tell Turtle’s state

[position()](https://docs.python.org/2/library/turtle.html#turtle.position) | [pos()](https://docs.python.org/2/library/turtle.html#turtle.pos)

[towards()](https://docs.python.org/2/library/turtle.html#turtle.towards)

[xcor()](https://docs.python.org/2/library/turtle.html#turtle.xcor)

[ycor()](https://docs.python.org/2/library/turtle.html#turtle.ycor)

[heading()](https://docs.python.org/2/library/turtle.html#turtle.heading)

[distance()](https://docs.python.org/2/library/turtle.html#turtle.distance)

Setting and measurement

[degrees()](https://docs.python.org/2/library/turtle.html#turtle.degrees)

[radians()](https://docs.python.org/2/library/turtle.html#turtle.radians)

Pen control

Drawing state

[pendown()](https://docs.python.org/2/library/turtle.html#turtle.pendown) | [pd()](https://docs.python.org/2/library/turtle.html#turtle.pd) | [down()](https://docs.python.org/2/library/turtle.html#turtle.down)

[penup()](https://docs.python.org/2/library/turtle.html#turtle.penup) | [pu()](https://docs.python.org/2/library/turtle.html#turtle.pu) | [up()](https://docs.python.org/2/library/turtle.html#turtle.up)

[pensize()](https://docs.python.org/2/library/turtle.html#turtle.pensize) | [width()](https://docs.python.org/2/library/turtle.html#turtle.width)

[pen()](https://docs.python.org/2/library/turtle.html#turtle.pen)

[isdown()](https://docs.python.org/2/library/turtle.html#turtle.isdown)

Color control

[color()](https://docs.python.org/2/library/turtle.html#turtle.color)

[pencolor()](https://docs.python.org/2/library/turtle.html#turtle.pencolor)

[fillcolor()](https://docs.python.org/2/library/turtle.html#turtle.fillcolor)

Filling

[fill()](https://docs.python.org/2/library/turtle.html#turtle.fill)

[begin\_fill()](https://docs.python.org/2/library/turtle.html#turtle.begin_fill)

[end\_fill()](https://docs.python.org/2/library/turtle.html#turtle.end_fill)

More drawing control

[reset()](https://docs.python.org/2/library/turtle.html#turtle.reset)

[clear()](https://docs.python.org/2/library/turtle.html#turtle.clear)

[write()](https://docs.python.org/2/library/turtle.html#turtle.write)

Turtle state

Visibility

[showturtle()](https://docs.python.org/2/library/turtle.html#turtle.showturtle) | [st()](https://docs.python.org/2/library/turtle.html#turtle.st)

[hideturtle()](https://docs.python.org/2/library/turtle.html#turtle.hideturtle) | [ht()](https://docs.python.org/2/library/turtle.html#turtle.ht)

[isvisible()](https://docs.python.org/2/library/turtle.html#turtle.isvisible)

Appearance

[shape()](https://docs.python.org/2/library/turtle.html#turtle.shape)

[resizemode()](https://docs.python.org/2/library/turtle.html#turtle.resizemode)

[shapesize()](https://docs.python.org/2/library/turtle.html#turtle.shapesize) | [turtlesize()](https://docs.python.org/2/library/turtle.html#turtle.turtlesize)

[settiltangle()](https://docs.python.org/2/library/turtle.html#turtle.settiltangle)

[tiltangle()](https://docs.python.org/2/library/turtle.html#turtle.tiltangle)

[tilt()](https://docs.python.org/2/library/turtle.html#turtle.tilt)

Using events

[onclick()](https://docs.python.org/2/library/turtle.html#turtle.onclick)

[onrelease()](https://docs.python.org/2/library/turtle.html#turtle.onrelease)

[ondrag()](https://docs.python.org/2/library/turtle.html#turtle.ondrag)

[mainloop()](https://docs.python.org/2/library/turtle.html#turtle.mainloop) | [done()](https://docs.python.org/2/library/turtle.html#turtle.done)

Special Turtle methods

[begin\_poly()](https://docs.python.org/2/library/turtle.html#turtle.begin_poly)

[end\_poly()](https://docs.python.org/2/library/turtle.html#turtle.end_poly)

[get\_poly()](https://docs.python.org/2/library/turtle.html#turtle.get_poly)

[clone()](https://docs.python.org/2/library/turtle.html#turtle.clone)

[getturtle()](https://docs.python.org/2/library/turtle.html#turtle.getturtle) | [getpen()](https://docs.python.org/2/library/turtle.html#turtle.getpen)

[getscreen()](https://docs.python.org/2/library/turtle.html#turtle.getscreen)

[setundobuffer()](https://docs.python.org/2/library/turtle.html#turtle.setundobuffer)

[undobufferentries()](https://docs.python.org/2/library/turtle.html#turtle.undobufferentries)

[tracer()](https://docs.python.org/2/library/turtle.html#turtle.tracer)

[window\_width()](https://docs.python.org/2/library/turtle.html#turtle.window_width)

[window\_height()](https://docs.python.org/2/library/turtle.html#turtle.window_height)

### 24.5.2.2. Methods of TurtleScreen/Screen

Window control

[bgcolor()](https://docs.python.org/2/library/turtle.html#turtle.bgcolor)

[bgpic()](https://docs.python.org/2/library/turtle.html#turtle.bgpic)

[clear()](https://docs.python.org/2/library/turtle.html#turtle.clear) | [clearscreen()](https://docs.python.org/2/library/turtle.html#turtle.clearscreen)

[reset()](https://docs.python.org/2/library/turtle.html#turtle.reset) | [resetscreen()](https://docs.python.org/2/library/turtle.html#turtle.resetscreen)

[screensize()](https://docs.python.org/2/library/turtle.html#turtle.screensize)

[setworldcoordinates()](https://docs.python.org/2/library/turtle.html#turtle.setworldcoordinates)

Animation control

[delay()](https://docs.python.org/2/library/turtle.html#turtle.delay)

[tracer()](https://docs.python.org/2/library/turtle.html#turtle.tracer)

[update()](https://docs.python.org/2/library/turtle.html#turtle.update)

Using screen events

[listen()](https://docs.python.org/2/library/turtle.html#turtle.listen)

[onkey()](https://docs.python.org/2/library/turtle.html#turtle.onkey)

[onclick()](https://docs.python.org/2/library/turtle.html#turtle.onclick) | [onscreenclick()](https://docs.python.org/2/library/turtle.html#turtle.onscreenclick)

[ontimer()](https://docs.python.org/2/library/turtle.html#turtle.ontimer)

Settings and special methods

[mode()](https://docs.python.org/2/library/turtle.html#turtle.mode)

[colormode()](https://docs.python.org/2/library/turtle.html#turtle.colormode)

[getcanvas()](https://docs.python.org/2/library/turtle.html#turtle.getcanvas)

[getshapes()](https://docs.python.org/2/library/turtle.html#turtle.getshapes)

[register\_shape()](https://docs.python.org/2/library/turtle.html#turtle.register_shape) | [addshape()](https://docs.python.org/2/library/turtle.html#turtle.addshape)

[turtles()](https://docs.python.org/2/library/turtle.html#turtle.turtles)

[window\_height()](https://docs.python.org/2/library/turtle.html#turtle.window_height)

[window\_width()](https://docs.python.org/2/library/turtle.html#turtle.window_width)

Methods specific to Screen

[bye()](https://docs.python.org/2/library/turtle.html#turtle.bye)

[exitonclick()](https://docs.python.org/2/library/turtle.html#turtle.exitonclick)

[setup()](https://docs.python.org/2/library/turtle.html#turtle.setup)

[title()](https://docs.python.org/2/library/turtle.html#turtle.title)

## 24.5.3. Methods of RawTurtle/Turtle and corresponding functions

Most of the examples in this section refer to a Turtle instance called turtle.

### 24.5.3.1. Turtle motion

turtle.forward(distance)

turtle.fd(distance)

|  |  |
| --- | --- |
| **Parameters:** | **distance** – a number (integer or float) |

Move the turtle forward by the specified distance, in the direction the turtle is headed.

>>>

>>> turtle.position()

(0.00,0.00)

>>> turtle.forward(25)

>>> turtle.position()

(25.00,0.00)

>>> turtle.forward(-75)

>>> turtle.position()

(-50.00,0.00)

turtle.back(distance)

turtle.bk(distance)

turtle.backward(distance)

|  |  |
| --- | --- |
| **Parameters:** | **distance** – a number |

Move the turtle backward by distance, opposite to the direction the turtle is headed. Do not change the turtle’s heading.

>>>

>>> turtle.position()

(0.00,0.00)

>>> turtle.backward(30)

>>> turtle.position()

(-30.00,0.00)

turtle.right(angle)

turtle.rt(angle)

|  |  |
| --- | --- |
| **Parameters:** | **angle** – a number (integer or float) |

Turn turtle right by angle units. (Units are by default degrees, but can be set via the [degrees()](https://docs.python.org/2/library/turtle.html#turtle.degrees) and [radians()](https://docs.python.org/2/library/turtle.html#turtle.radians) functions.) Angle orientation depends on the turtle mode, see [mode()](https://docs.python.org/2/library/turtle.html#turtle.mode).

>>>

>>> turtle.heading()

22.0

>>> turtle.right(45)

>>> turtle.heading()

337.0

turtle.left(angle)

turtle.lt(angle)

|  |  |
| --- | --- |
| **Parameters:** | **angle** – a number (integer or float) |

Turn turtle left by angle units. (Units are by default degrees, but can be set via the [degrees()](https://docs.python.org/2/library/turtle.html#turtle.degrees) and [radians()](https://docs.python.org/2/library/turtle.html#turtle.radians) functions.) Angle orientation depends on the turtle mode, see [mode()](https://docs.python.org/2/library/turtle.html#turtle.mode).

>>>

>>> turtle.heading()

22.0

>>> turtle.left(45)

>>> turtle.heading()

67.0

turtle.goto(x, y=None)

turtle.setpos(x, y=None)

turtle.setposition(x, y=None)

|  |  |
| --- | --- |
| **Parameters:** | * **x** – a number or a pair/vector of numbers * **y** – a number or None |

If y is None, x must be a pair of coordinates or a [Vec2D](https://docs.python.org/2/library/turtle.html#turtle.Vec2D) (e.g. as returned by [pos()](https://docs.python.org/2/library/turtle.html#turtle.pos)).

Move turtle to an absolute position. If the pen is down, draw line. Do not change the turtle’s orientation.

>>>

>>> tp = turtle.pos()

>>> tp

(0.00,0.00)

>>> turtle.setpos(60,30)

>>> turtle.pos()

(60.00,30.00)

>>> turtle.setpos((20,80))

>>> turtle.pos()

(20.00,80.00)

>>> turtle.setpos(tp)

>>> turtle.pos()

(0.00,0.00)

turtle.setx(x)

|  |  |
| --- | --- |
| **Parameters:** | **x** – a number (integer or float) |

Set the turtle’s first coordinate to x, leave second coordinate unchanged.

>>>

>>> turtle.position()

(0.00,240.00)

>>> turtle.setx(10)

>>> turtle.position()

(10.00,240.00)

turtle.sety(y)

|  |  |
| --- | --- |
| **Parameters:** | **y** – a number (integer or float) |

Set the turtle’s second coordinate to y, leave first coordinate unchanged.

>>>

>>> turtle.position()

(0.00,40.00)

>>> turtle.sety(-10)

>>> turtle.position()

(0.00,-10.00)

turtle.setheading(to\_angle)

turtle.seth(to\_angle)

|  |  |
| --- | --- |
| **Parameters:** | **to\_angle** – a number (integer or float) |

Set the orientation of the turtle to to\_angle. Here are some common directions in degrees:

| **standard mode** | **logo mode** |
| --- | --- |
| 0 - east | 0 - north |
| 90 - north | 90 - east |
| 180 - west | 180 - south |
| 270 - south | 270 - west |

>>>

>>> turtle.setheading(90)

>>> turtle.heading()

90.0

turtle.home()

Move turtle to the origin – coordinates (0,0) – and set its heading to its start-orientation (which depends on the mode, see [mode()](https://docs.python.org/2/library/turtle.html#turtle.mode)).

>>>

>>> turtle.heading()

90.0

>>> turtle.position()

(0.00,-10.00)

>>> turtle.home()

>>> turtle.position()

(0.00,0.00)

>>> turtle.heading()

0.0

turtle.circle(radius, extent=None, steps=None)

|  |  |
| --- | --- |
| **Parameters:** | * **radius** – a number * **extent** – a number (or None) * **steps** – an integer (or None) |

Draw a circle with given radius. The center is radius units left of the turtle; extent – an angle – determines which part of the circle is drawn. If extent is not given, draw the entire circle. If extent is not a full circle, one endpoint of the arc is the current pen position. Draw the arc in counterclockwise direction if radius is positive, otherwise in clockwise direction. Finally the direction of the turtle is changed by the amount of extent.

As the circle is approximated by an inscribed regular polygon, steps determines the number of steps to use. If not given, it will be calculated automatically. May be used to draw regular polygons.

>>>

>>> turtle.home()

>>> turtle.position()

(0.00,0.00)

>>> turtle.heading()

0.0

>>> turtle.circle(50)

>>> turtle.position()

(-0.00,0.00)

>>> turtle.heading()

0.0

>>> turtle.circle(120, 180) # draw a semicircle

>>> turtle.position()

(0.00,240.00)

>>> turtle.heading()

180.0

turtle.dot(size=None, \*color)

|  |  |
| --- | --- |
| **Parameters:** | * **size** – an integer >= 1 (if given) * **color** – a colorstring or a numeric color tuple |

Draw a circular dot with diameter size, using color. If size is not given, the maximum of pensize+4 and 2\*pensize is used.

>>>

>>> turtle.home()

>>> turtle.dot()

>>> turtle.fd(50); turtle.dot(20, "blue"); turtle.fd(50)

>>> turtle.position()

(100.00,-0.00)

>>> turtle.heading()

0.0

turtle.stamp()

Stamp a copy of the turtle shape onto the canvas at the current turtle position. Return a stamp\_id for that stamp, which can be used to delete it by calling clearstamp(stamp\_id).

>>>

>>> turtle.color("blue")

>>> turtle.stamp()

11

>>> turtle.fd(50)

turtle.clearstamp(stampid)

|  |  |
| --- | --- |
| **Parameters:** | **stampid** – an integer, must be return value of previous [stamp()](https://docs.python.org/2/library/turtle.html#turtle.stamp) call |

Delete stamp with given stampid.

>>>

>>> turtle.position()

(150.00,-0.00)

>>> turtle.color("blue")

>>> astamp = turtle.stamp()

>>> turtle.fd(50)

>>> turtle.position()

(200.00,-0.00)

>>> turtle.clearstamp(astamp)

>>> turtle.position()

(200.00,-0.00)

turtle.clearstamps(n=None)

|  |  |
| --- | --- |
| **Parameters:** | **n** – an integer (or None) |

Delete all or first/last n of turtle’s stamps. If n is None, delete all stamps, if n > 0 delete first n stamps, else if n < 0 delete last n stamps.

>>>

>>> for i in range(8):

... turtle.stamp(); turtle.fd(30)

13

14

15

16

17

18

19

20

>>> turtle.clearstamps(2)

>>> turtle.clearstamps(-2)

>>> turtle.clearstamps()

turtle.undo()

Undo (repeatedly) the last turtle action(s). Number of available undo actions is determined by the size of the undobuffer.

>>>

>>> for i in range(4):

... turtle.fd(50); turtle.lt(80)

...

>>> for i in range(8):

... turtle.undo()

turtle.speed(speed=None)

|  |  |
| --- | --- |
| **Parameters:** | **speed** – an integer in the range 0..10 or a speedstring (see below) |

Set the turtle’s speed to an integer value in the range 0..10. If no argument is given, return current speed.

If input is a number greater than 10 or smaller than 0.5, speed is set to 0. Speedstrings are mapped to speedvalues as follows:

* “fastest”: 0
* “fast”: 10
* “normal”: 6
* “slow”: 3
* “slowest”: 1

Speeds from 1 to 10 enforce increasingly faster animation of line drawing and turtle turning.

Attention: speed = 0 means that no animation takes place. forward/back makes turtle jump and likewise left/right make the turtle turn instantly.

>>>

>>> turtle.speed()

3

>>> turtle.speed('normal')

>>> turtle.speed()

6

>>> turtle.speed(9)

>>> turtle.speed()

9

### 24.5.3.2. Tell Turtle’s state

turtle.position()

turtle.pos()

Return the turtle’s current location (x,y) (as a [Vec2D](https://docs.python.org/2/library/turtle.html#turtle.Vec2D) vector).

>>>

>>> turtle.pos()

(440.00,-0.00)

turtle.towards(x, y=None)

|  |  |
| --- | --- |
| **Parameters:** | * **x** – a number or a pair/vector of numbers or a turtle instance * **y** – a number if x is a number, else None |

Return the angle between the line from turtle position to position specified by (x,y), the vector or the other turtle. This depends on the turtle’s start orientation which depends on the mode - “standard”/”world” or “logo”).

>>>

>>> turtle.goto(10, 10)

>>> turtle.towards(0,0)

225.0

turtle.xcor()

Return the turtle’s x coordinate.

>>>

>>> turtle.home()

>>> turtle.left(50)

>>> turtle.forward(100)

>>> turtle.pos()

(64.28,76.60)

>>> print turtle.xcor()

64.2787609687

turtle.ycor()

Return the turtle’s y coordinate.

>>>

>>> turtle.home()

>>> turtle.left(60)

>>> turtle.forward(100)

>>> print turtle.pos()

(50.00,86.60)

>>> print turtle.ycor()

86.6025403784

turtle.heading()

Return the turtle’s current heading (value depends on the turtle mode, see [mode()](https://docs.python.org/2/library/turtle.html#turtle.mode)).

>>>

>>> turtle.home()

>>> turtle.left(67)

>>> turtle.heading()

67.0

turtle.distance(x, y=None)

|  |  |
| --- | --- |
| **Parameters:** | * **x** – a number or a pair/vector of numbers or a turtle instance * **y** – a number if x is a number, else None |

Return the distance from the turtle to (x,y), the given vector, or the given other turtle, in turtle step units.

>>>

>>> turtle.home()

>>> turtle.distance(30,40)

50.0

>>> turtle.distance((30,40))

50.0

>>> joe = Turtle()

>>> joe.forward(77)

>>> turtle.distance(joe)

77.0

### 24.5.3.3. Settings for measurement

turtle.degrees(fullcircle=360.0)

|  |  |
| --- | --- |
| **Parameters:** | **fullcircle** – a number |

Set angle measurement units, i.e. set number of “degrees” for a full circle. Default value is 360 degrees.

>>>

>>> turtle.home()

>>> turtle.left(90)

>>> turtle.heading()

90.0

Change angle measurement unit to grad (also known as gon,

grade, or gradian and equals 1/100-th of the right angle.)

>>> turtle.degrees(400.0)

>>> turtle.heading()

100.0

>>> turtle.degrees(360)

>>> turtle.heading()

90.0

turtle.radians()

Set the angle measurement units to radians. Equivalent to degrees(2\*math.pi).

>>>

>>> turtle.home()

>>> turtle.left(90)

>>> turtle.heading()

90.0

>>> turtle.radians()

>>> turtle.heading()

1.5707963267948966

### 24.5.3.4. Pen control

#### 24.5.3.4.1. Drawing state

turtle.pendown()

turtle.pd()

turtle.down()

Pull the pen down – drawing when moving.

turtle.penup()

turtle.pu()

turtle.up()

Pull the pen up – no drawing when moving.

turtle.pensize(width=None)

turtle.width(width=None)

|  |  |
| --- | --- |
| **Parameters:** | **width** – a positive number |

Set the line thickness to width or return it. If resizemode is set to “auto” and turtleshape is a polygon, that polygon is drawn with the same line thickness. If no argument is given, the current pensize is returned.

>>>

>>> turtle.pensize()

1

>>> turtle.pensize(10) # from here on lines of width 10 are drawn

turtle.pen(pen=None, \*\*pendict)

|  |  |
| --- | --- |
| **Parameters:** | * **pen** – a dictionary with some or all of the below listed keys * **pendict** – one or more keyword-arguments with the below listed keys as keywords |

Return or set the pen’s attributes in a “pen-dictionary” with the following key/value pairs:

* “shown”: True/False
* “pendown”: True/False
* “pencolor”: color-string or color-tuple
* “fillcolor”: color-string or color-tuple
* “pensize”: positive number
* “speed”: number in range 0..10
* “resizemode”: “auto” or “user” or “noresize”
* “stretchfactor”: (positive number, positive number)
* “outline”: positive number
* “tilt”: number

This dictionary can be used as argument for a subsequent call to [pen()](https://docs.python.org/2/library/turtle.html#turtle.pen) to restore the former pen-state. Moreover one or more of these attributes can be provided as keyword-arguments. This can be used to set several pen attributes in one statement.

>>>

>>> turtle.pen(fillcolor="black", pencolor="red", pensize=10)

>>> sorted(turtle.pen().items())

[('fillcolor', 'black'), ('outline', 1), ('pencolor', 'red'),

('pendown', True), ('pensize', 10), ('resizemode', 'noresize'),

('shown', True), ('speed', 9), ('stretchfactor', (1, 1)), ('tilt', 0)]

>>> penstate=turtle.pen()

>>> turtle.color("yellow", "")

>>> turtle.penup()

>>> sorted(turtle.pen().items())

[('fillcolor', ''), ('outline', 1), ('pencolor', 'yellow'),

('pendown', False), ('pensize', 10), ('resizemode', 'noresize'),

('shown', True), ('speed', 9), ('stretchfactor', (1, 1)), ('tilt', 0)]

>>> turtle.pen(penstate, fillcolor="green")

>>> sorted(turtle.pen().items())

[('fillcolor', 'green'), ('outline', 1), ('pencolor', 'red'),

('pendown', True), ('pensize', 10), ('resizemode', 'noresize'),

('shown', True), ('speed', 9), ('stretchfactor', (1, 1)), ('tilt', 0)]

turtle.isdown()

Return True if pen is down, False if it’s up.

>>>

>>> turtle.penup()

>>> turtle.isdown()

False

>>> turtle.pendown()

>>> turtle.isdown()

True

#### 24.5.3.4.2. Color control

turtle.pencolor(\*args)

Return or set the pencolor.

Four input formats are allowed:

pencolor()

Return the current pencolor as color specification string or as a tuple (see example). May be used as input to another color/pencolor/fillcolor call.

pencolor(colorstring)

Set pencolor to colorstring, which is a Tk color specification string, such as "red", "yellow", or "#33cc8c".

pencolor((r, g, b))

Set pencolor to the RGB color represented by the tuple of r, g, and b. Each of r, g, and b must be in the range 0..colormode, where colormode is either 1.0 or 255 (see [colormode()](https://docs.python.org/2/library/turtle.html#turtle.colormode)).

pencolor(r, g, b)

Set pencolor to the RGB color represented by r, g, and b. Each of r, g, and b must be in the range 0..colormode.

If turtleshape is a polygon, the outline of that polygon is drawn with the newly set pencolor.

>>>

>>> colormode()

1.0

>>> turtle.pencolor()

'red'

>>> turtle.pencolor("brown")

>>> turtle.pencolor()

'brown'

>>> tup = (0.2, 0.8, 0.55)

>>> turtle.pencolor(tup)

>>> turtle.pencolor()

(0.2, 0.8, 0.5490196078431373)

>>> colormode(255)

>>> turtle.pencolor()

(51, 204, 140)

>>> turtle.pencolor('#32c18f')

>>> turtle.pencolor()

(50, 193, 143)

turtle.fillcolor(\*args)

Return or set the fillcolor.

Four input formats are allowed:

fillcolor()

Return the current fillcolor as color specification string, possibly in tuple format (see example). May be used as input to another color/pencolor/fillcolor call.

fillcolor(colorstring)

Set fillcolor to colorstring, which is a Tk color specification string, such as "red", "yellow", or "#33cc8c".

fillcolor((r, g, b))

Set fillcolor to the RGB color represented by the tuple of r, g, and b. Each of r, g, and b must be in the range 0..colormode, where colormode is either 1.0 or 255 (see [colormode()](https://docs.python.org/2/library/turtle.html#turtle.colormode)).

fillcolor(r, g, b)

Set fillcolor to the RGB color represented by r, g, and b. Each of r, g, and b must be in the range 0..colormode.

If turtleshape is a polygon, the interior of that polygon is drawn with the newly set fillcolor.

>>>

>>> turtle.fillcolor("violet")

>>> turtle.fillcolor()

'violet'

>>> col = turtle.pencolor()

>>> col

(50, 193, 143)

>>> turtle.fillcolor(col)

>>> turtle.fillcolor()

(50, 193, 143)

>>> turtle.fillcolor('#ffffff')

>>> turtle.fillcolor()

(255, 255, 255)

turtle.color(\*args)

Return or set pencolor and fillcolor.

Several input formats are allowed. They use 0 to 3 arguments as follows:

color()

Return the current pencolor and the current fillcolor as a pair of color specification strings or tuples as returned by [pencolor()](https://docs.python.org/2/library/turtle.html#turtle.pencolor) and [fillcolor()](https://docs.python.org/2/library/turtle.html#turtle.fillcolor).

color(colorstring), color((r,g,b)), color(r,g,b)

Inputs as in [pencolor()](https://docs.python.org/2/library/turtle.html#turtle.pencolor), set both, fillcolor and pencolor, to the given value.

color(colorstring1, colorstring2), color((r1,g1,b1), (r2,g2,b2))

Equivalent to pencolor(colorstring1) and fillcolor(colorstring2) and analogously if the other input format is used.

If turtleshape is a polygon, outline and interior of that polygon is drawn with the newly set colors.

>>>

>>> turtle.color("red", "green")

>>> turtle.color()

('red', 'green')

>>> color("#285078", "#a0c8f0")

>>> color()

((40, 80, 120), (160, 200, 240))

See also: Screen method [colormode()](https://docs.python.org/2/library/turtle.html#turtle.colormode).

#### 24.5.3.4.3. Filling

turtle.fill(flag)

|  |  |
| --- | --- |
| **Parameters:** | **flag** – True/False (or 1/0 respectively) |

Call fill(True) before drawing the shape you want to fill, and fill(False) when done. When used without argument: return fillstate (True if filling, False else).

>>>

>>> turtle.fill(True)

>>> for \_ in range(3):

... turtle.forward(100)

... turtle.left(120)

...

>>> turtle.fill(False)

turtle.begin\_fill()

Call just before drawing a shape to be filled. Equivalent to fill(True).

turtle.end\_fill()

Fill the shape drawn after the last call to [begin\_fill()](https://docs.python.org/2/library/turtle.html#turtle.begin_fill). Equivalent to fill(False).

>>>

>>> turtle.color("black", "red")

>>> turtle.begin\_fill()

>>> turtle.circle(80)

>>> turtle.end\_fill()

#### 24.5.3.4.4. More drawing control

turtle.reset()

Delete the turtle’s drawings from the screen, re-center the turtle and set variables to the default values.

>>>

>>> turtle.goto(0,-22)

>>> turtle.left(100)

>>> turtle.position()

(0.00,-22.00)

>>> turtle.heading()

100.0

>>> turtle.reset()

>>> turtle.position()

(0.00,0.00)

>>> turtle.heading()

0.0

turtle.clear()

Delete the turtle’s drawings from the screen. Do not move turtle. State and position of the turtle as well as drawings of other turtles are not affected.

turtle.write(arg, move=False, align="left", font=("Arial", 8, "normal"))

|  |  |
| --- | --- |
| **Parameters:** | * **arg** – object to be written to the TurtleScreen * **move** – True/False * **align** – one of the strings “left”, “center” or right” * **font** – a triple (fontname, fontsize, fonttype) |

Write text - the string representation of arg - at the current turtle position according to align (“left”, “center” or right”) and with the given font. If move is true, the pen is moved to the bottom-right corner of the text. By default, move is False.

>>>

>>> turtle.write("Home = ", True, align="center")

>>> turtle.write((0,0), True)

### 24.5.3.5. Turtle state

#### 24.5.3.5.1. Visibility

turtle.hideturtle()

turtle.ht()

Make the turtle invisible. It’s a good idea to do this while you’re in the middle of doing some complex drawing, because hiding the turtle speeds up the drawing observably.

>>>

>>> turtle.hideturtle()

turtle.showturtle()

turtle.st()

Make the turtle visible.

>>>

>>> turtle.showturtle()

turtle.isvisible()

Return True if the Turtle is shown, False if it’s hidden.

>>>

>>> turtle.hideturtle()

>>> turtle.isvisible()

False

>>> turtle.showturtle()

>>> turtle.isvisible()

True

#### 24.5.3.5.2. Appearance

turtle.shape(name=None)

|  |  |
| --- | --- |
| **Parameters:** | **name** – a string which is a valid shapename |

Set turtle shape to shape with given name or, if name is not given, return name of current shape. Shape with name must exist in the TurtleScreen’s shape dictionary. Initially there are the following polygon shapes: “arrow”, “turtle”, “circle”, “square”, “triangle”, “classic”. To learn about how to deal with shapes see Screen method [register\_shape()](https://docs.python.org/2/library/turtle.html#turtle.register_shape).

>>>

>>> turtle.shape()

'classic'

>>> turtle.shape("turtle")

>>> turtle.shape()

'turtle'

turtle.resizemode(rmode=None)

|  |  |
| --- | --- |
| **Parameters:** | **rmode** – one of the strings “auto”, “user”, “noresize” |

Set resizemode to one of the values: “auto”, “user”, “noresize”. If rmode is not given, return current resizemode. Different resizemodes have the following effects:

* “auto”: adapts the appearance of the turtle corresponding to the value of pensize.
* “user”: adapts the appearance of the turtle according to the values of stretchfactor and outlinewidth (outline), which are set by [shapesize()](https://docs.python.org/2/library/turtle.html#turtle.shapesize).
* “noresize”: no adaption of the turtle’s appearance takes place.

resizemode(“user”) is called by [shapesize()](https://docs.python.org/2/library/turtle.html#turtle.shapesize) when used with arguments.

>>>

>>> turtle.resizemode()

'noresize'

>>> turtle.resizemode("auto")

>>> turtle.resizemode()

'auto'

turtle.shapesize(stretch\_wid=None, stretch\_len=None, outline=None)

turtle.turtlesize(stretch\_wid=None, stretch\_len=None, outline=None)

|  |  |
| --- | --- |
| **Parameters:** | * **stretch\_wid** – positive number * **stretch\_len** – positive number * **outline** – positive number |

Return or set the pen’s attributes x/y-stretchfactors and/or outline. Set resizemode to “user”. If and only if resizemode is set to “user”, the turtle will be displayed stretched according to its stretchfactors: stretch\_wid is stretchfactor perpendicular to its orientation, stretch\_len is stretchfactor in direction of its orientation, outline determines the width of the shapes’s outline.

>>>

>>> turtle.shapesize()

(1, 1, 1)

>>> turtle.resizemode("user")

>>> turtle.shapesize(5, 5, 12)

>>> turtle.shapesize()

(5, 5, 12)

>>> turtle.shapesize(outline=8)

>>> turtle.shapesize()

(5, 5, 8)

turtle.tilt(angle)

|  |  |
| --- | --- |
| **Parameters:** | **angle** – a number |

Rotate the turtleshape by angle from its current tilt-angle, but do not change the turtle’s heading (direction of movement).

>>>

>>> turtle.reset()

>>> turtle.shape("circle")

>>> turtle.shapesize(5,2)

>>> turtle.tilt(30)

>>> turtle.fd(50)

>>> turtle.tilt(30)

>>> turtle.fd(50)

turtle.settiltangle(angle)

|  |  |
| --- | --- |
| **Parameters:** | **angle** – a number |

Rotate the turtleshape to point in the direction specified by angle, regardless of its current tilt-angle. Do not change the turtle’s heading (direction of movement).

>>>

>>> turtle.reset()

>>> turtle.shape("circle")

>>> turtle.shapesize(5,2)

>>> turtle.settiltangle(45)

>>> turtle.fd(50)

>>> turtle.settiltangle(-45)

>>> turtle.fd(50)

turtle.tiltangle()

Return the current tilt-angle, i.e. the angle between the orientation of the turtleshape and the heading of the turtle (its direction of movement).

>>>

>>> turtle.reset()

>>> turtle.shape("circle")

>>> turtle.shapesize(5,2)

>>> turtle.tilt(45)

>>> turtle.tiltangle()

45.0

### 24.5.3.6. Using events

turtle.onclick(fun, btn=1, add=None)

|  |  |
| --- | --- |
| **Parameters:** | * **fun** – a function with two arguments which will be called with the coordinates of the clicked point on the canvas * **num** – number of the mouse-button, defaults to 1 (left mouse button) * **add** – True or False – if True, a new binding will be added, otherwise it will replace a former binding |

Bind fun to mouse-click events on this turtle. If fun is None, existing bindings are removed. Example for the anonymous turtle, i.e. the procedural way:

>>>

>>> def turn(x, y):

... left(180)

...

>>> onclick(turn) # Now clicking into the turtle will turn it.

>>> onclick(None) # event-binding will be removed

turtle.onrelease(fun, btn=1, add=None)

|  |  |
| --- | --- |
| **Parameters:** | * **fun** – a function with two arguments which will be called with the coordinates of the clicked point on the canvas * **num** – number of the mouse-button, defaults to 1 (left mouse button) * **add** – True or False – if True, a new binding will be added, otherwise it will replace a former binding |

Bind fun to mouse-button-release events on this turtle. If fun is None, existing bindings are removed.

>>>

>>> class MyTurtle(Turtle):

... def glow(self,x,y):

... self.fillcolor("red")

... def unglow(self,x,y):

... self.fillcolor("")

...

>>> turtle = MyTurtle()

>>> turtle.onclick(turtle.glow) # clicking on turtle turns fillcolor red,

>>> turtle.onrelease(turtle.unglow) # releasing turns it to transparent.

turtle.ondrag(fun, btn=1, add=None)

|  |  |
| --- | --- |
| **Parameters:** | * **fun** – a function with two arguments which will be called with the coordinates of the clicked point on the canvas * **num** – number of the mouse-button, defaults to 1 (left mouse button) * **add** – True or False – if True, a new binding will be added, otherwise it will replace a former binding |

Bind fun to mouse-move events on this turtle. If fun is None, existing bindings are removed.

Remark: Every sequence of mouse-move-events on a turtle is preceded by a mouse-click event on that turtle.

>>>

>>> turtle.ondrag(turtle.goto)

Subsequently, clicking and dragging the Turtle will move it across the screen thereby producing handdrawings (if pen is down).

turtle.mainloop()

turtle.done()

Starts event loop - calling Tkinter’s mainloop function. Must be the last statement in a turtle graphics program.

>>>

>>> turtle.mainloop()

### 24.5.3.7. Special Turtle methods

turtle.begin\_poly()

Start recording the vertices of a polygon. Current turtle position is first vertex of polygon.

turtle.end\_poly()

Stop recording the vertices of a polygon. Current turtle position is last vertex of polygon. This will be connected with the first vertex.

turtle.get\_poly()

Return the last recorded polygon.

>>>

>>> turtle.home()

>>> turtle.begin\_poly()

>>> turtle.fd(100)

>>> turtle.left(20)

>>> turtle.fd(30)

>>> turtle.left(60)

>>> turtle.fd(50)

>>> turtle.end\_poly()

>>> p = turtle.get\_poly()

>>> register\_shape("myFavouriteShape", p)

turtle.clone()

Create and return a clone of the turtle with same position, heading and turtle properties.

>>>

>>> mick = Turtle()

>>> joe = mick.clone()

turtle.getturtle()

turtle.getpen()

Return the Turtle object itself. Only reasonable use: as a function to return the “anonymous turtle”:

>>>

>>> pet = getturtle()

>>> pet.fd(50)

>>> pet

<turtle.Turtle object at 0x...>

turtle.getscreen()

Return the [TurtleScreen](https://docs.python.org/2/library/turtle.html#turtle.TurtleScreen) object the turtle is drawing on. TurtleScreen methods can then be called for that object.

>>>

>>> ts = turtle.getscreen()

>>> ts

<turtle.\_Screen object at 0x...>

>>> ts.bgcolor("pink")

turtle.setundobuffer(size)

|  |  |
| --- | --- |
| **Parameters:** | **size** – an integer or None |

Set or disable undobuffer. If size is an integer an empty undobuffer of given size is installed. size gives the maximum number of turtle actions that can be undone by the [undo()](https://docs.python.org/2/library/turtle.html#turtle.undo) method/function. If size is None, the undobuffer is disabled.

>>>

>>> turtle.setundobuffer(42)

turtle.undobufferentries()

Return number of entries in the undobuffer.

>>>

>>> while undobufferentries():

... undo()

turtle.tracer(flag=None, delay=None)

A replica of the corresponding TurtleScreen method.

Deprecated since version 2.6.

turtle.window\_width()

turtle.window\_height()

Both are replicas of the corresponding TurtleScreen methods.

Deprecated since version 2.6.

### 24.5.3.8. Excursus about the use of compound shapes

To use compound turtle shapes, which consist of several polygons of different color, you must use the helper class [Shape](https://docs.python.org/2/library/turtle.html#turtle.Shape) explicitly as described below:

1. Create an empty Shape object of type “compound”.
2. Add as many components to this object as desired, using the addcomponent() method.

For example:

>>>

>>> s = Shape("compound")

>>> poly1 = ((0,0),(10,-5),(0,10),(-10,-5))

>>> s.addcomponent(poly1, "red", "blue")

>>> poly2 = ((0,0),(10,-5),(-10,-5))

>>> s.addcomponent(poly2, "blue", "red")

1. Now add the Shape to the Screen’s shapelist and use it:

>>>

>>> register\_shape("myshape", s)

>>> shape("myshape")

Note

The [Shape](https://docs.python.org/2/library/turtle.html#turtle.Shape) class is used internally by the [register\_shape()](https://docs.python.org/2/library/turtle.html#turtle.register_shape) method in different ways. The application programmer has to deal with the Shape class only when using compound shapes like shown above!

## 24.5.4. Methods of TurtleScreen/Screen and corresponding functions

Most of the examples in this section refer to a TurtleScreen instance called screen.

### 24.5.4.1. Window control

turtle.bgcolor(\*args)

|  |  |
| --- | --- |
| **Parameters:** | **args** – a color string or three numbers in the range 0..colormode or a 3-tuple of such numbers |

Set or return background color of the TurtleScreen.

>>>

>>> screen.bgcolor("orange")

>>> screen.bgcolor()

'orange'

>>> screen.bgcolor("#800080")

>>> screen.bgcolor()

(128, 0, 128)

turtle.bgpic(picname=None)

|  |  |
| --- | --- |
| **Parameters:** | **picname** – a string, name of a gif-file or "nopic", or None |

Set background image or return name of current backgroundimage. If picname is a filename, set the corresponding image as background. If picname is "nopic", delete background image, if present. If picname is None, return the filename of the current backgroundimage.

>>>

>>> screen.bgpic()

'nopic'

>>> screen.bgpic("landscape.gif")

>>> screen.bgpic()

"landscape.gif"

turtle.clear()

turtle.clearscreen()

Delete all drawings and all turtles from the TurtleScreen. Reset the now empty TurtleScreen to its initial state: white background, no background image, no event bindings and tracing on.

Note

This TurtleScreen method is available as a global function only under the name clearscreen. The global function clear is another one derived from the Turtle method clear.

turtle.reset()

turtle.resetscreen()

Reset all Turtles on the Screen to their initial state.

Note

This TurtleScreen method is available as a global function only under the name resetscreen. The global function reset is another one derived from the Turtle method reset.

turtle.screensize(canvwidth=None, canvheight=None, bg=None)

|  |  |
| --- | --- |
| **Parameters:** | * **canvwidth** – positive integer, new width of canvas in pixels * **canvheight** – positive integer, new height of canvas in pixels * **bg** – colorstring or color-tuple, new background color |

If no arguments are given, return current (canvaswidth, canvasheight). Else resize the canvas the turtles are drawing on. Do not alter the drawing window. To observe hidden parts of the canvas, use the scrollbars. With this method, one can make visible those parts of a drawing which were outside the canvas before.

>>>

>>> screen.screensize()

(400, 300)

>>> screen.screensize(2000,1500)

>>> screen.screensize()

(2000, 1500)

e.g. to search for an erroneously escaped turtle ;-)

turtle.setworldcoordinates(llx, lly, urx, ury)

|  |  |
| --- | --- |
| **Parameters:** | * **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 |

Set up user-defined coordinate 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.

**ATTENTION**: in user-defined coordinate systems angles may appear distorted.

>>>

>>> screen.reset()

>>> screen.setworldcoordinates(-50,-7.5,50,7.5)

>>> for \_ in range(72):

... left(10)

...

>>> for \_ in range(8):

... left(45); fd(2) # a regular octagon

### 24.5.4.2. Animation control

turtle.delay(delay=None)

|  |  |
| --- | --- |
| **Parameters:** | **delay** – positive integer |

Set or return the drawing delay in milliseconds. (This is approximately the time interval between two consecutive canvas updates.) The longer the drawing delay, the slower the animation.

Optional argument:

>>>

>>> screen.delay()

10

>>> screen.delay(5)

>>> screen.delay()

5

turtle.tracer(n=None, delay=None)

|  |  |
| --- | --- |
| **Parameters:** | * **n** – nonnegative integer * **delay** – nonnegative integer |

Turn turtle animation on/off and set delay for update drawings. 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 argument sets delay value (see [delay()](https://docs.python.org/2/library/turtle.html#turtle.delay)).

>>>

>>> screen.tracer(8, 25)

>>> dist = 2

>>> for i in range(200):

... fd(dist)

... rt(90)

... dist += 2

turtle.update()

Perform a TurtleScreen update. To be used when tracer is turned off.

See also the RawTurtle/Turtle method [speed()](https://docs.python.org/2/library/turtle.html#turtle.speed).

### 24.5.4.3. Using screen events

turtle.listen(xdummy=None, ydummy=None)

Set focus on TurtleScreen (in order to collect key-events). Dummy arguments are provided in order to be able to pass [listen()](https://docs.python.org/2/library/turtle.html#turtle.listen) to the onclick method.

turtle.onkey(fun, key)

|  |  |
| --- | --- |
| **Parameters:** | * **fun** – a function with no arguments or None * **key** – a string: key (e.g. “a”) or key-symbol (e.g. “space”) |

Bind fun to key-release event of key. If fun is None, event bindings are removed. Remark: in order to be able to register key-events, TurtleScreen must have the focus. (See method [listen()](https://docs.python.org/2/library/turtle.html#turtle.listen).)

>>>

>>> def f():

... fd(50)

... lt(60)

...

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

>>> screen.listen()

turtle.onclick(fun, btn=1, add=None)

turtle.onscreenclick(fun, btn=1, add=None)

|  |  |
| --- | --- |
| **Parameters:** | * **fun** – a function with two arguments which will be called with the coordinates of the clicked point on the canvas * **num** – number of the mouse-button, defaults to 1 (left mouse button) * **add** – True or False – if True, a new binding will be added, otherwise it will replace a former binding |

Bind fun to mouse-click events on this screen. If fun is None, existing bindings are removed.

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) # remove event binding again

Note

This TurtleScreen method is available as a global function only under the name onscreenclick. The global function onclick is another one derived from the Turtle method onclick.

turtle.ontimer(fun, t=0)

|  |  |
| --- | --- |
| **Parameters:** | * **fun** – a function with no arguments * **t** – a number >= 0 |

Install a timer that calls fun after t milliseconds.

>>>

>>> running = True

>>> def f():

... if running:

... fd(50)

... lt(60)

... screen.ontimer(f, 250)

>>> f() ### makes the turtle march around

>>> running = False

### 24.5.4.4. Settings and special methods

turtle.mode(mode=None)

|  |  |
| --- | --- |
| **Parameters:** | **mode** – one of the strings “standard”, “logo” or “world” |

Set turtle mode (“standard”, “logo” or “world”) and perform reset. If mode is not given, current mode is returned.

Mode “standard” is compatible with old [turtle](https://docs.python.org/2/library/turtle.html#module-turtle). Mode “logo” is compatible with most Logo turtle graphics. Mode “world” uses user-defined “world coordinates”. **Attention**: in this mode angles appear distorted if x/y unit-ratio doesn’t equal 1.

| **Mode** | **Initial turtle heading** | **positive angles** |
| --- | --- | --- |
| “standard” | to the right (east) | counterclockwise |
| “logo” | upward (north) | clockwise |

>>>

>>> mode("logo") # resets turtle heading to north

>>> mode()

'logo'

turtle.colormode(cmode=None)

|  |  |
| --- | --- |
| **Parameters:** | **cmode** – one of the values 1.0 or 255 |

Return the colormode or set it to 1.0 or 255. Subsequently r, g, b values of color triples have to be in the range 0..cmode.

>>>

>>> screen.colormode(1)

>>> turtle.pencolor(240, 160, 80)

Traceback (most recent call last):

...

TurtleGraphicsError: bad color sequence: (240, 160, 80)

>>> screen.colormode()

1.0

>>> screen.colormode(255)

>>> screen.colormode()

255

>>> turtle.pencolor(240,160,80)

turtle.getcanvas()

Return the Canvas of this TurtleScreen. Useful for insiders who know what to do with a Tkinter Canvas.

>>>

>>> cv = screen.getcanvas()

>>> cv

<turtle.ScrolledCanvas instance at 0x...>

turtle.getshapes()

Return a list of names of all currently available turtle shapes.

>>>

>>> screen.getshapes()

['arrow', 'blank', 'circle', ..., 'turtle']

turtle.register\_shape(name, shape=None)

turtle.addshape(name, shape=None)

There are three different ways to call this function:

1. name is the name of a gif-file and shape is None: Install the corresponding image shape.

>>>

>>> screen.register\_shape("turtle.gif")

Note

Image shapes do not rotate when turning the turtle, so they do not display the heading of the turtle!

1. name is an arbitrary string and shape is a tuple of pairs of coordinates: Install the corresponding polygon shape.

>>>

>>> screen.register\_shape("triangle", ((5,-3), (0,5), (-5,-3)))

1. name is an arbitrary string and shape is a (compound) [Shape](https://docs.python.org/2/library/turtle.html#turtle.Shape) object: Install the corresponding compound shape.

Add a turtle shape to TurtleScreen’s shapelist. Only thusly registered shapes can be used by issuing the command shape(shapename).

turtle.turtles()

Return the list of turtles on the screen.

>>>

>>> for turtle in screen.turtles():

... turtle.color("red")

turtle.window\_height()

Return the height of the turtle window.

>>>

>>> screen.window\_height()

480

turtle.window\_width()

Return the width of the turtle window.

>>>

>>> screen.window\_width()

640

### 24.5.4.5. Methods specific to Screen, not inherited from TurtleScreen

turtle.bye()

Shut the turtlegraphics window.

turtle.exitonclick()

Bind bye() method to mouse clicks on the Screen.

If the value “using\_IDLE” in the configuration dictionary is False (default value), also enter mainloop. Remark: If IDLE with the -n switch (no subprocess) is used, this value should be set to True in turtle.cfg. In this case IDLE’s own mainloop is active also for the client script.

turtle.setup(width=\_CFG["width"], height=\_CFG["height"], startx=\_CFG["leftright"], starty=\_CFG["topbottom"])

Set the size and position of the main window. Default values of arguments are stored in the configuration dictionary and can be changed via a turtle.cfg file.

|  |  |
| --- | --- |
| **Parameters:** | * **width** – if an integer, a size in pixels, if a float, a fraction of the screen; default is 50% of screen * **height** – if an integer, the height in pixels, if a float, a fraction of the screen; default is 75% of screen * **startx** – if positive, starting position in pixels from the left edge of the screen, if negative from the right edge, if None, center window horizontally * **starty** – if positive, starting position in pixels from the top edge of the screen, if negative from the bottom edge, if None, center window vertically |

>>>

>>> screen.setup (width=200, height=200, startx=0, starty=0)

>>> # sets window to 200x200 pixels, in upper left of screen

>>> screen.setup(width=.75, height=0.5, startx=None, starty=None)

>>> # sets window to 75% of screen by 50% of screen and centers

turtle.title(titlestring)

|  |  |
| --- | --- |
| **Parameters:** | **titlestring** – a string that is shown in the titlebar of the turtle graphics window |

Set title of turtle window to titlestring.

>>>

>>> screen.title("Welcome to the turtle zoo!")

## 24.5.5. The public classes of the module [turtle](https://docs.python.org/2/library/turtle.html#module-turtle)

class turtle.RawTurtle(canvas)

class turtle.RawPen(canvas)

|  |  |
| --- | --- |
| **Parameters:** | **canvas** – a Tkinter.Canvas, a [ScrolledCanvas](https://docs.python.org/2/library/turtle.html#turtle.ScrolledCanvas) or a [TurtleScreen](https://docs.python.org/2/library/turtle.html#turtle.TurtleScreen) |

Create a turtle. The turtle has all methods described above as “methods of Turtle/RawTurtle”.

class turtle.Turtle

Subclass of RawTurtle, has the same interface but draws on a default [Screen](https://docs.python.org/2/library/turtle.html#turtle.Screen) object created automatically when needed for the first time.

class turtle.TurtleScreen(cv)

|  |  |
| --- | --- |
| **Parameters:** | **cv** – a Tkinter.Canvas |

Provides screen oriented methods like setbg() etc. that are described above.

class turtle.Screen

Subclass of TurtleScreen, with [four methods added](https://docs.python.org/2/library/turtle.html#screenspecific).

class turtle.ScrolledCanvas(master)

|  |  |
| --- | --- |
| **Parameters:** | **master** – some Tkinter widget to contain the ScrolledCanvas, i.e. a Tkinter-canvas with scrollbars added |

Used by class Screen, which thus automatically provides a ScrolledCanvas as playground for the turtles.

class turtle.Shape(type\_, data)

|  |  |
| --- | --- |
| **Parameters:** | **type\_** – one of the strings “polygon”, “image”, “compound” |

Data structure modeling shapes. The pair (type\_, data) must follow this specification:

| **type\_** | **data** |
| --- | --- |
| “polygon” | a polygon-tuple, i.e. a tuple of pairs of coordinates |
| “image” | an image (in this form only used internally!) |
| “compound” | None (a compound shape has to be constructed using the [addcomponent()](https://docs.python.org/2/library/turtle.html#turtle.Shape.addcomponent) method) |

addcomponent(poly, fill, outline=None)

|  |  |
| --- | --- |
| **Parameters:** | * **poly** – a polygon, i.e. a tuple of pairs of numbers * **fill** – a color the poly will be filled with * **outline** – a color for the poly’s outline (if given) |

Example:

>>>

>>> poly = ((0,0),(10,-5),(0,10),(-10,-5))

>>> s = Shape("compound")

>>> s.addcomponent(poly, "red", "blue")

>>> # ... add more components and then use register\_shape()

See [Excursus about the use of compound shapes](https://docs.python.org/2/library/turtle.html#compoundshapes).

class turtle.Vec2D(x, y)

A two-dimensional vector class, used as a helper class for implementing turtle graphics. May be useful for turtle graphics programs too. Derived from tuple, so a vector is a tuple!

Provides (for a, b vectors, k number):

* a + b vector addition
* a - b vector subtraction
* a \* b inner product
* k \* a and a \* k multiplication with scalar
* abs(a) absolute value of a
* a.rotate(angle) rotation

## 24.5.6. Help and configuration

### 24.5.6.1. How to use help

The public methods of the Screen and Turtle classes are documented extensively via docstrings. So these can be used as online-help via the Python help facilities:

* When using IDLE, tooltips show the signatures and first lines of the docstrings of typed in function-/method calls.
* Calling [help()](https://docs.python.org/2/library/functions.html#help) on methods or functions displays the docstrings:

>>>

>>> help(Screen.bgcolor)

Help on method bgcolor in module turtle:

bgcolor(self, \*args) unbound turtle.Screen method

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.

>>> screen.bgcolor("orange")

>>> screen.bgcolor()

"orange"

>>> screen.bgcolor(0.5,0,0.5)

>>> screen.bgcolor()

"#800080"

>>> help(Turtle.penup)

Help on method penup in module turtle:

penup(self) unbound turtle.Turtle method

Pull the pen up -- no drawing when moving.

Aliases: penup | pu | up

No argument

>>> turtle.penup()

* The docstrings of the functions which are derived from methods have a modified form:

>>>

>>> help(bgcolor)

Help on function bgcolor in module turtle:

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::

>>> bgcolor("orange")

>>> bgcolor()

"orange"

>>> bgcolor(0.5,0,0.5)

>>> bgcolor()

"#800080"

>>> help(penup)

Help on function penup in module turtle:

penup()

Pull the pen up -- no drawing when moving.

Aliases: penup | pu | up

No argument

Example:

>>> penup()

These modified docstrings are created automatically together with the function definitions that are derived from the methods at import time.

### 24.5.6.2. Translation of docstrings into different languages

There is a utility to create a dictionary the keys of which are the method names and the values of which are the docstrings of the public methods of the classes Screen and Turtle.

turtle.write\_docstringdict(filename="turtle\_docstringdict")

|  |  |
| --- | --- |
| **Parameters:** | **filename** – a string, used as filename |

Create and write docstring-dictionary to a Python script with the given filename. This function has to be called explicitly (it is not used by the turtle graphics classes). The docstring dictionary will be written to the Python script *filename*.py. It is intended to serve as a template for translation of the docstrings into different languages.

If you (or your students) want to use [turtle](https://docs.python.org/2/library/turtle.html#module-turtle) with online help in your native language, you have to translate the docstrings and save the resulting file as e.g. turtle\_docstringdict\_german.py.

If you have an appropriate entry in your turtle.cfg file this dictionary will be read in at import time and will replace the original English docstrings.

At the time of this writing there are docstring dictionaries in German and in Italian. (Requests please to [glingl@aon.at](mailto:glingl%40aon.at).)

### 24.5.6.3. How to configure Screen and Turtles

The built-in default configuration mimics the appearance and behaviour of the old turtle module in order to retain best possible compatibility with it.

If you want to use a different configuration which better reflects the features of this module or which better fits to your needs, e.g. for use in a classroom, you can prepare a configuration file turtle.cfg which will be read at import time and modify the configuration according to its settings.

The built in configuration would correspond to the following turtle.cfg:

width = 0.5

height = 0.75

leftright = None

topbottom = None

canvwidth = 400

canvheight = 300

mode = standard

colormode = 1.0

delay = 10

undobuffersize = 1000

shape = classic

pencolor = black

fillcolor = black

resizemode = noresize

visible = True

language = english

exampleturtle = turtle

examplescreen = screen

title = Python Turtle Graphics

using\_IDLE = False

Short explanation of selected entries:

* The first four lines correspond to the arguments of the Screen.setup() method.
* Line 5 and 6 correspond to the arguments of the method Screen.screensize().
* shape can be any of the built-in shapes, e.g: arrow, turtle, etc. For more info try help(shape).
* If you want to use no fillcolor (i.e. make the turtle transparent), you have to write fillcolor = "" (but all nonempty strings must not have quotes in the cfg-file).
* If you want to reflect the turtle its state, you have to use resizemode = auto.
* If you set e.g. language = italian the docstringdict turtle\_docstringdict\_italian.py will be loaded at import time (if present on the import path, e.g. in the same directory as [turtle](https://docs.python.org/2/library/turtle.html#module-turtle).
* The entries exampleturtle and examplescreen define the names of these objects as they occur in the docstrings. The transformation of method-docstrings to function-docstrings will delete these names from the docstrings.
* using\_IDLE: Set this to True if you regularly work with IDLE and its -n switch (“no subprocess”). This will prevent [exitonclick()](https://docs.python.org/2/library/turtle.html#turtle.exitonclick) to enter the mainloop.

There can be a turtle.cfg file in the directory where [turtle](https://docs.python.org/2/library/turtle.html#module-turtle) is stored and an additional one in the current working directory. The latter will override the settings of the first one.

The Demo/turtle directory contains a turtle.cfg file. You can study it as an example and see its effects when running the demos (preferably not from within the demo-viewer).

## 24.5.7. Demo scripts

There is a set of demo scripts in the turtledemo directory located in the Demo/turtle directory in the source distribution.

It contains:

* a set of 15 demo scripts demonstrating different features of the new module [turtle](https://docs.python.org/2/library/turtle.html#module-turtle)
* a demo viewer turtleDemo.py which can be used to view the sourcecode of the scripts and run them at the same time. 14 of the examples can be accessed via the Examples menu; all of them can also be run standalone.
* The example turtledemo\_two\_canvases.py demonstrates the simultaneous use of two canvases with the turtle module. Therefore it only can be run standalone.
* There is a turtle.cfg file in this directory, which also serves as an example for how to write and use such files.

The demoscripts are:

| **Name** | **Description** | **Features** |
| --- | --- | --- |
| bytedesign | complex classical turtlegraphics pattern | [tracer()](https://docs.python.org/2/library/turtle.html#turtle.tracer), delay, [update()](https://docs.python.org/2/library/turtle.html#turtle.update) |
| chaos | graphs Verhulst dynamics, shows that computer’s computations can generate results sometimes against the common sense expectations | world coordinates |
| clock | analog clock showing time of your computer | turtles as clock’s hands, ontimer |
| colormixer | experiment with r, g, b | [ondrag()](https://docs.python.org/2/library/turtle.html#turtle.ondrag) |
| fractalcurves | Hilbert & Koch curves | recursion |
| lindenmayer | ethnomathematics (indian kolams) | L-System |
| minimal\_hanoi | Towers of Hanoi | Rectangular Turtles as Hanoi discs (shape, shapesize) |
| paint | super minimalistic drawing program | [onclick()](https://docs.python.org/2/library/turtle.html#turtle.onclick) |
| peace | elementary | turtle: appearance and animation |
| penrose | aperiodic tiling with kites and darts | [stamp()](https://docs.python.org/2/library/turtle.html#turtle.stamp) |
| planet\_and\_moon | simulation of gravitational system | compound shapes, [Vec2D](https://docs.python.org/2/library/turtle.html#turtle.Vec2D) |
| tree | a (graphical) breadth first tree (using generators) | [clone()](https://docs.python.org/2/library/turtle.html#turtle.clone) |
| wikipedia | a pattern from the wikipedia article on turtle graphics | [clone()](https://docs.python.org/2/library/turtle.html#turtle.clone), [undo()](https://docs.python.org/2/library/turtle.html#turtle.undo) |
| yingyang | another elementary example | [circle()](https://docs.python.org/2/library/turtle.html#turtle.circle) |

Have fun!

### This Page

* [Report a Bug](https://docs.python.org/2/bugs.html)
* [Show Source](https://github.com/python/cpython/blob/2.7/Doc/library/turtle.rst)

© [Copyright](https://docs.python.org/2/copyright.html) 1990-2017, Python Software Foundation.   
The Python Software Foundation is a non-profit corporation. [Please donate.](https://www.python.org/psf/donations/)   
Last updated on Mar 27, 2017. [Found a bug](https://docs.python.org/2/bugs.html)?   
Created using [Sphinx](http://sphinx.pocoo.org/) 1.3.3.