

Minimal program

```
import turtle

screen = turtle.Screen()
t = turtle.Turtle()

t.forward(100)
t.left(90)

turtle.done()
```

Setup and the window

<code>screen = turtle.Screen()</code>	Create (or get) the drawing window
<code>t = turtle.Turtle()</code>	Create a turtle to draw with
<code>screen.setup(width, height)</code>	Window size in pixels
<code>screen.title(text)</code>	Window title
<code>screen.bgcolor(color)</code>	Background colour
<code>screen.bgpic(file)</code>	Background image (GIF or PNG)
<code>screen.screensize(w, h)</code>	Size of the drawing area (adds scrollbars if larger than the window)
<code>screen.window_width()</code>	Current window width (window_height() for height)
<code>turtle.done()</code>	Keep the window open until you close it; put it last
<code>screen.exitonclick()</code>	Close the window when it is clicked
<code>screen.bye()</code>	Close the window from code

Move and turn

<code>t.forward(distance)</code> fd	Move forward, drawing if the pen is down
<code>t.backward(distance)</code> bk back	Move backward without turning
<code>t.right(angle)</code> rt	Turn clockwise by angle degrees
<code>t.left(angle)</code> lt	Turn counterclockwise
<code>t.goto(x, y)</code> setpos setposition	Move to a point (draws a line if the pen is down)
<code>t.teleport(x, y)</code>	Jump to a point without drawing (Python 3.12+)
<code>t.setx(x)</code> / <code>t.sety(y)</code> sety	Change only the x or only the y coordinate
<code>t.setheading(angle)</code> seth	Point in a direction: 0 east, 90 north, 180 west, 270 south
<code>t.home()</code>	Back to (0, 0), facing east
<code>t.circle(radius, extent, steps)</code>	Circle, arc or regular polygon; negative radius goes clockwise
<code>t.dot(size, color)</code>	Filled dot at the current position
<code>t.stamp()</code>	Print a copy of the turtle shape; returns an id
<code>t.clearstamp(id)</code> clearstamps	Remove one stamp (clearstamps() removes all)
<code>t.undo()</code>	Undo the last action
<code>t.speed(speed)</code>	1 slowest to 10 fast, 0 = no animation; or "slow", "fast", "fastest"

Where is the turtle?

<code>t.position()</code> pos	Current (x, y)
<code>t.xcor()</code> / <code>t.ycor()</code> ycor	Current x or y only
<code>t.heading()</code>	Current direction in degrees
<code>t.towards(x, y)</code>	Angle from the turtle to a point
<code>t.distance(x, y)</code>	Distance to a point (or to another turtle)

Pen

<code>t.pendown()</code> pd down	Draw when moving
<code>t.penup()</code> pu up	Move without drawing
<code>t.pensize(width)</code> width	Line thickness
<code>t.isdown()</code>	True if the pen is down
<code>t.pen(**settings)</code>	Set several pen options at once

Colour and fill

<code>t.color(pen, fill)</code>	Set pen and fill colour together
<code>t.pencolor(color)</code>	Line and text colour: name, hex or RGB
<code>t.fillcolor(color)</code>	Colour used by begin_fill()/end_fill()
<code>t.begin_fill()</code> ... <code>t.end_fill()</code> end_fill	Fill the shape drawn between the two calls
<code>t.filling()</code>	True while a fill is in progress
<code>screen.colormode(255)</code>	Use RGB values 0 to 255 instead of 0 to 1

Turtle shape and visibility

<code>t.shape(name)</code>	"arrow", "turtle", "circle", "square", "triangle" or "classic"
<code>t.shapesize(w, l, outline)</code> turtlesize	Stretch the shape: width, length, outline
<code>t.hideturtle()</code> ht	Hide the turtle (drawing is faster)
<code>t.showturtle()</code> st	Show it again
<code>t.isvisible()</code>	True if the turtle is shown
<code>t.tilt(angle)</code>	Rotate the shape without changing the heading
<code>screen.register_shape(file)</code> addshape	Use a GIF image as a turtle shape
<code>screen.getshapes()</code>	List the available shape names

Text and input

<code>t.write(text, move, align, font)</code>	Write text; align "left", "center" or "right"
<code>screen.textinput(title, prompt)</code>	Ask the user for text (None if cancelled)
<code>screen.numinput(title, prompt)</code>	Ask for a number, with optional default, minval, maxval

Clear and reset

<code>t.clear()</code>	Erase this turtle's drawings; position and settings stay
<code>t.reset()</code>	Erase and send the turtle home with default settings
<code>screen.reset()</code> resetscreen	Reset every turtle on the screen
<code>screen.clear()</code> clearscreen	Remove all drawings, turtles, background and events

Animation and speed

<code>screen.tracer(0)</code>	Stop redrawing after every step (much faster)
<code>screen.update()</code>	Redraw the screen once (use with <code>tracer(0)</code>)
<code>screen.ontimer(function, ms)</code>	Call a function after ms milliseconds
<code>screen.delay(ms)</code>	Pause between screen updates

Keyboard and mouse events

<code>screen.listen()</code>	Send key presses to the Turtle window
<code>screen.onkey(function, key)</code>	Run a function when a key is released: "Up", "a", "space"
<code>screen.onkeypress(function, key)</code> onkeyrelease	Run while the key is pressed (repeats when held)
<code>screen.onclick(function)</code> onscreenclick	Click anywhere; the function gets x and y
<code>t.onclick(function)</code>	Click on the turtle itself
<code>t.ondrag(function)</code> onrelease	Drag the turtle with the mouse

More turtles and saving

<code>t.clone()</code>	Copy a turtle with the same position and settings
<code>screen.turtles()</code>	List every turtle on the screen
<code>screen.getcanvas()</code>	The Tk canvas, e.g. to save the drawing as EPS

Grey names are aliases, e.g. `t.fd()` = `t.forward()`. Checked against Python 3.12's turtle module.