Ch11-Turtles-Events

September 10, 2025

1 Turtle program & event handling

http://openbookproject.net/thinkcs/python/english3e/hello_little_turtles.html - can't run this notebook in Colab or other online services - there are many modules in Python that provide various useful and powerful features - TKinter, turtle, email, url, http, etc. - running turtle program multiple times from jupyter notebook crashes jupyter server… so run scripts by themselves in turtle folder

1.1 1st turtle program

[2]:
```python
import turtle
```

[3]:
```python
help(turtle)
```

```
Help on module turtle:

NAME
    turtle

MODULE REFERENCE
    https://docs.python.org/3.6/library/turtle

    The following documentation is automatically generated from the Python
    source files.  It may be incomplete, incorrect or include features that
    are considered implementation detail and may vary between Python
    implementations.  When in doubt, consult the module reference at the
    location listed above.

DESCRIPTION
    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.

----- turtle.py

This module is an extended reimplementation of turtle.py from the Python standard distribution up to Python 2.5. (See: http://www.python.org)

It tries to keep the merits of turtle.py 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.

Roughly it has the following features added:

- Better animation of the turtle movements, especially of turning the turtle. So the turtles can more easily be used as a visual feedback instrument by the (beginning) programmer.

- Different turtle shapes, gif-images as turtle shapes, user defined and user controllable turtle shapes, among them compound (multicolored) shapes. Turtle shapes can be stretched and tilted, which makes turtles very versatile geometrical objects.

- Fine control over turtle movement and screen updates via delay(), and enhanced tracer() and speed() methods.

- Aliases for the most commonly used commands, like fd for forward etc., following the early Logo traditions. This reduces the boring work of typing long sequences of commands, which often occur in a natural way when kids try to program fancy pictures on their first encounter with turtle graphics.

- Turtles now have an undo()-method with configurable undo-buffer.

- Some simple commands/methods for creating event driven programs (mouse-, key-, timer-events). Especially useful for programming games.

- A scrollable Canvas class. The default scrollable Canvas can be extended interactively as needed while playing around with the turtle(s).

- A TurtleScreen class with methods controlling background color or background image, window and canvas size and other properties of the TurtleScreen.

- There is a method, setworldcoordinates(), to install a user defined

```
      coordinate-system for the TurtleScreen.

    - The implementation uses a 2-vector class named Vec2D, derived from tuple.
      This class is public, so it can be imported by the application programmer,
      which makes certain types of computations very natural and compact.

    - Appearance of the TurtleScreen and the Turtles at startup/import can be
      configured by means of a turtle.cfg configuration file.
      The default configuration mimics the appearance of the old turtle module.

    - If configured appropriately the module reads in docstrings from a
docstring
      dictionary in some different language, supplied separately  and replaces
      the English ones by those read in. There is a utility function
      write_docstringdict() to write a dictionary with the original (English)
      docstrings to disc, so it can serve as a template for translations.

    Behind the scenes there are some features included with possible
    extensions in mind. These will be commented and documented elsewhere.

CLASSES
    builtins.Exception(builtins.BaseException)
        Terminator
    builtins.object
        Shape
    builtins.tuple(builtins.object)
        Vec2D
    tkinter.Frame(tkinter.Widget)
        ScrolledCanvas
    TNavigator(builtins.object)
        RawTurtle(TPen, TNavigator)
            Turtle
    TPen(builtins.object)
        RawTurtle(TPen, TNavigator)
            Turtle
    TurtleScreenBase(builtins.object)
        TurtleScreen

    Pen = class Turtle(RawTurtle)
     |  RawTurtle auto-creating (scrolled) canvas.
     |
     |  When a Turtle object is created or a function derived from some
     |  Turtle method is called a TurtleScreen object is automatically created.
     |
     |  Method resolution order:
     |      Turtle
     |      RawTurtle
     |      TPen
     |      TNavigator
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __init__(self, shape='classic', undobuffersize=1000, visible=True)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from RawTurtle:
     |
     |  begin_fill(self)
     |      Called just before drawing a shape to be filled.
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color("black", "red")
     |      >>> turtle.begin_fill()
     |      >>> turtle.circle(60)
     |      >>> turtle.end_fill()
     |
     |  begin_poly(self)
     |      Start recording the vertices of a polygon.
     |
     |      No argument.
     |
     |      Start recording the vertices of a polygon. Current turtle position
     |      is first point of polygon.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.begin_poly()
     |
     |  clear(self)
     |      Delete the turtle's drawings from the screen. Do not move turtle.
     |
     |      No arguments.
     |
     |      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.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.clear()
     |
     |  clearstamp(self, stampid)
     |      Delete stamp with given stampid
     |
     |      Argument:
     |      stampid - an integer, must be return value of previous stamp() call.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color("blue")
     |      >>> astamp = turtle.stamp()
     |      >>> turtle.fd(50)
     |      >>> turtle.clearstamp(astamp)
     |
     |  clearstamps(self, n=None)
     |      Delete all or first/last n of turtle's stamps.
     |
     |      Optional argument:
     |      n -- an integer
     |
     |      If n is None, delete all of pen's stamps,
     |      else if n > 0 delete first n stamps
     |      else if n < 0 delete last n stamps.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> for i in range(8):
     |      …     turtle.stamp(); turtle.fd(30)
     |      …
     |      >>> turtle.clearstamps(2)
     |      >>> turtle.clearstamps(-2)
     |      >>> turtle.clearstamps()
     |
     |  clone(self)
     |      Create and return a clone of the turtle.
     |
     |      No argument.
     |
     |      Create and return a clone of the turtle with same position, heading
     |      and turtle properties.
     |
     |      Example (for a Turtle instance named mick):
     |      mick = Turtle()
     |      joe = mick.clone()
     |
     |  dot(self, size=None, *color)
     |      Draw a dot with diameter size, using color.
     |
     |      Optional arguments:
     |      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.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.dot()
     |      >>> turtle.fd(50); turtle.dot(20, "blue"); turtle.fd(50)
     |
     |  end_fill(self)
     |      Fill the shape drawn after the call begin_fill().
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color("black", "red")
     |      >>> turtle.begin_fill()
     |      >>> turtle.circle(60)
     |      >>> turtle.end_fill()
     |
     |  end_poly(self)
     |      Stop recording the vertices of a polygon.
     |
     |      No argument.
     |
     |      Stop recording the vertices of a polygon. Current turtle position is
     |      last point of polygon. This will be connected with the first point.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.end_poly()
     |
     |  filling(self)
     |      Return fillstate (True if filling, False else).
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.begin_fill()
     |      >>> if turtle.filling():
     |      …     turtle.pensize(5)
     |      … else:
     |      …     turtle.pensize(3)
     |
     |  get_poly(self)
     |      Return the lastly recorded polygon.
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> p = turtle.get_poly()
     |      >>> turtle.register_shape("myFavouriteShape", p)
```

```
|
|  get_shapepoly(self)
|      Return the current shape polygon as tuple of coordinate pairs.
|
|      No argument.
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.shape("square")
|      >>> turtle.shapetransform(4, -1, 0, 2)
|      >>> turtle.get_shapepoly()
|      ((50, -20), (30, 20), (-50, 20), (-30, -20))
|
|  getpen = getturtle(self)
|      Return the Turtleobject itself.
|
|      No argument.
|
|      Only reasonable use: as a function to return the 'anonymous turtle':
|
|      Example:
|      >>> pet = getturtle()
|      >>> pet.fd(50)
|      >>> pet
|      <turtle.Turtle object at 0x0187D810>
|      >>> turtles()
|      [<turtle.Turtle object at 0x0187D810>]
|
|  getscreen(self)
|      Return the TurtleScreen object, the turtle is drawing  on.
|
|      No argument.
|
|      Return the TurtleScreen object, the turtle is drawing  on.
|      So TurtleScreen-methods can be called for that object.
|
|      Example (for a Turtle instance named turtle):
|      >>> ts = turtle.getscreen()
|      >>> ts
|      <turtle.TurtleScreen object at 0x0106B770>
|      >>> ts.bgcolor("pink")
|
|  getturtle(self)
|      Return the Turtleobject itself.
|
|      No argument.
|
|      Only reasonable use: as a function to return the 'anonymous turtle':
|
     |      Example:
     |      >>> pet = getturtle()
     |      >>> pet.fd(50)
     |      >>> pet
     |      <turtle.Turtle object at 0x0187D810>
     |      >>> turtles()
     |      [<turtle.Turtle object at 0x0187D810>]
     |
     |  onclick(self, fun, btn=1, add=None)
     |      Bind fun to mouse-click event on this turtle on canvas.
     |
     |      Arguments:
     |      fun --  a function with two arguments, to which will be assigned
     |              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, new binding will be added, otherwise
     |              it will replace a former binding.
     |
     |      Example for the anonymous turtle, i. e. the procedural way:
     |
     |      >>> def turn(x, y):
     |      …     left(360)
     |      …
     |      >>> onclick(turn)  # Now clicking into the turtle will turn it.
     |      >>> onclick(None)  # event-binding will be removed
     |
     |  ondrag(self, fun, btn=1, add=None)
     |      Bind fun to mouse-move event on this turtle on canvas.
     |
     |      Arguments:
     |      fun -- a function with two arguments, to which will be assigned
     |             the coordinates of the clicked point on the canvas.
     |      num -- number of the mouse-button defaults to 1 (left mouse button).
     |
     |      Every sequence of mouse-move-events on a turtle is preceded by a
     |      mouse-click event on that turtle.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.ondrag(turtle.goto)
     |
     |      Subsequently clicking and dragging a Turtle will move it
     |      across the screen thereby producing handdrawings (if pen is
     |      down).
     |
     |  onrelease(self, fun, btn=1, add=None)
     |      Bind fun to mouse-button-release event on this turtle on canvas.
     |
     |      Arguments:
     |      fun -- a function with two arguments, to which will be assigned
     |              the coordinates of the clicked point on the canvas.
     |      num --  number of the mouse-button defaults to 1 (left mouse
button).
     |
     |      Example (for a MyTurtle instance named joe):
     |      >>> class MyTurtle(Turtle):
     |      …     def glow(self,x,y):
     |      …             self.fillcolor("red")
     |      …     def unglow(self,x,y):
     |      …             self.fillcolor("")
     |      …
     |      >>> joe = MyTurtle()
     |      >>> joe.onclick(joe.glow)
     |      >>> joe.onrelease(joe.unglow)
     |
     |      Clicking on joe turns fillcolor red, unclicking turns it to
     |      transparent.
     |
     |  reset(self)
     |      Delete the turtle's drawings and restore its default values.
     |
     |      No argument.
     |
     |      Delete the turtle's drawings from the screen, re-center the turtle
     |      and set variables to the default values.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00,-22.00)
     |      >>> turtle.heading()
     |      100.0
     |      >>> turtle.reset()
     |      >>> turtle.position()
     |      (0.00,0.00)
     |      >>> turtle.heading()
     |      0.0
     |
     |  settiltangle(self, angle)
     |      Rotate the turtleshape to point in the specified direction
     |
     |      Argument: angle -- 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).
     |
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.settiltangle(45)
     |      >>> stamp()
     |      >>> turtle.fd(50)
     |      >>> turtle.settiltangle(-45)
     |      >>> stamp()
     |      >>> turtle.fd(50)
     |
     |  setundobuffer(self, size)
     |      Set or disable undobuffer.
     |
     |      Argument:
     |      size -- an integer or None
     |
     |      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() function.
     |      If size is None, no undobuffer is present.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.setundobuffer(42)
     |
     |  shape(self, name=None)
     |      Set turtle shape to shape with given name / return current
shapename.
     |
     |      Optional argument:
     |      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.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.shape()
     |      'arrow'
     |      >>> turtle.shape("turtle")
     |      >>> turtle.shape()
     |      'turtle'
     |
     |  shapesize(self, stretch_wid=None, stretch_len=None, outline=None)
     |      Set/return turtle's stretchfactors/outline. Set resizemode to
"user".
     |
     |      Optional arguments:
     |         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 orientation
     |      stretch_len is stretchfactor in direction of turtles orientation.
     |      outline determines the width of the shapes's outline.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.resizemode("user")
     |      >>> turtle.shapesize(5, 5, 12)
     |      >>> turtle.shapesize(outline=8)
     |
     |  shapetransform(self, t11=None, t12=None, t21=None, t22=None)
     |      Set or return the current transformation matrix of the turtle shape.
     |
     |      Optional arguments: t11, t12, t21, t22 -- numbers.
     |
     |      If none of the matrix elements are given, return the transformation
     |      matrix.
     |      Otherwise set the given elements and transform the turtleshape
     |      according to the matrix consisting of first row t11, t12 and
     |      second row t21, 22.
     |      Modify stretchfactor, shearfactor and tiltangle according to the
     |      given matrix.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("square")
     |      >>> turtle.shapesize(4,2)
     |      >>> turtle.shearfactor(-0.5)
     |      >>> turtle.shapetransform()
     |      (4.0, -1.0, -0.0, 2.0)
     |
     |  shearfactor(self, shear=None)
     |      Set or return the current shearfactor.
     |
     |      Optional argument: shear -- number, tangent of the shear angle
```

```
|
|       Shear the turtleshape according to the given shearfactor shear,
|       which is the tangent of the shear angle. DO NOT change the
|       turtle's heading (direction of movement).
|       If shear is not given: return the current shearfactor, i. e. the
|       tangent of the shear angle, by which lines parallel to the
|       heading of the turtle are sheared.
|
|       Examples (for a Turtle instance named turtle):
|       >>> turtle.shape("circle")
|       >>> turtle.shapesize(5,2)
|       >>> turtle.shearfactor(0.5)
|       >>> turtle.shearfactor()
|       >>> 0.5
|
|   stamp(self)
|       Stamp a copy of the turtleshape onto the canvas and return its id.
|
|       No argument.
|
|       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).
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.color("blue")
|       >>> turtle.stamp()
|       13
|       >>> turtle.fd(50)
|
|   tilt(self, angle)
|       Rotate the turtleshape by angle.
|
|       Argument:
|       angle - a number
|
|       Rotate the turtleshape by angle from its current tilt-angle,
|       but do NOT change the turtle's heading (direction of movement).
|
|       Examples (for a Turtle instance named turtle):
|       >>> turtle.shape("circle")
|       >>> turtle.shapesize(5,2)
|       >>> turtle.tilt(30)
|       >>> turtle.fd(50)
|       >>> turtle.tilt(30)
|       >>> turtle.fd(50)
|
|   tiltangle(self, angle=None)
```

```
     |      Set or return the current tilt-angle.
     |
     |      Optional argument: angle -- 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).
     |      If angle is not given: 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).
     |
     |      Deprecated since Python 3.1
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.tilt(45)
     |      >>> turtle.tiltangle()
     |
     |  turtlesize = shapesize(self, stretch_wid=None, stretch_len=None,
outline=None)
     |      Set/return turtle's stretchfactors/outline. Set resizemode to
"user".
     |
     |      Optional arguments:
     |         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 orientation
     |      stretch_len is stretchfactor in direction of turtles orientation.
     |      outline determines the width of the shapes's outline.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.resizemode("user")
     |      >>> turtle.shapesize(5, 5, 12)
     |      >>> turtle.shapesize(outline=8)
     |
     |  undo(self)
     |      undo (repeatedly) the last turtle action.
     |
     |      No argument.
     |
     |      undo (repeatedly) the last turtle action.
     |      Number of available undo actions is determined by the size of
     |      the undobuffer.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> for i in range(4):
     |      …     turtle.fd(50); turtle.lt(80)
     |      …
     |      >>> for i in range(8):
     |      …     turtle.undo()
     |      …
     |
     |  undobufferentries(self)
     |      Return count of entries in the undobuffer.
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> while undobufferentries():
     |      …     undo()
     |
     |  write(self, arg, move=False, align='left', font=('Arial', 8, 'normal'))
     |      Write text at the current turtle position.
     |
     |      Arguments:
     |      arg -- info, which is to be written to the TurtleScreen
     |      move (optional) -- True/False
     |      align (optional) -- one of the strings "left", "center" or right"
     |      font (optional) -- 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.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.write('Home = ', True, align="center")
     |      >>> turtle.write((0,0), True)
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from RawTurtle:
     |
     |  screens = []
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from TPen:
     |
     |  color(self, *args)
     |      Return or set the pencolor and fillcolor.
     |
     |      Arguments:
     |      Several input formats are allowed.
     |      They use 0, 1, 2, or 3 arguments as follows:
     |
     |      color()
     |          Return the current pencolor and the current fillcolor
     |          as a pair of color specification strings as are returned
     |          by pencolor and fillcolor.
     |      color(colorstring), color((r,g,b)), color(r,g,b)
     |          inputs as in 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.
     |      For mor info see: pencolor, fillcolor
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color('red', 'green')
     |      >>> turtle.color()
     |      ('red', 'green')
     |      >>> colormode(255)
     |      >>> color((40, 80, 120), (160, 200, 240))
     |      >>> color()
     |      ('#285078', '#a0c8f0')
     |
     |  down = pendown(self)
     |      Pull the pen down -- drawing when moving.
     |
     |      Aliases: pendown | pd | down
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.pendown()
     |
     |  fillcolor(self, *args)
     |      Return or set the fillcolor.
     |
     |      Arguments:
     |      Four input formats are allowed:
     |        - fillcolor()
     |          Return the current fillcolor as color specification string,
     |          possibly in hex-number format (see example).
     |          May be used as input to another color/pencolor/fillcolor call.
     |        - fillcolor(colorstring)
     |          s is a Tk color specification string, such as "red" or "yellow"
     |        - fillcolor((r, g, b))
     |          *a tuple* of r, g, and b, which represent, an RGB color,
     |          and each of r, g, and b are in the range 0..colormode,
     |          where colormode is either 1.0 or 255
     |        - fillcolor(r, g, b)
     |          r, g, and b represent an RGB color, and each of r, g, and b
     |          are in the range 0..colormode
     |
     |      If turtleshape is a polygon, the interior of that polygon is drawn
     |      with the newly set fillcolor.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.fillcolor('violet')
     |      >>> col = turtle.pencolor()
     |      >>> turtle.fillcolor(col)
     |      >>> turtle.fillcolor(0, .5, 0)
     |
     |  hideturtle(self)
     |      Makes the turtle invisible.
     |
     |      Aliases: hideturtle | ht
     |
     |      No argument.
     |
     |      It's a good idea to do this while you're in the
     |      middle of a complicated drawing, because hiding
     |      the turtle speeds up the drawing observably.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.hideturtle()
     |
     |  ht = hideturtle(self)
     |      Makes the turtle invisible.
     |
     |      Aliases: hideturtle | ht
     |
     |      No argument.
     |
     |      It's a good idea to do this while you're in the
     |      middle of a complicated drawing, because hiding
     |      the turtle speeds up the drawing observably.
```

```
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|
|  isdown(self)
|      Return True if pen is down, False if it's up.
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|      >>> turtle.isdown()
|      False
|      >>> turtle.pendown()
|      >>> turtle.isdown()
|      True
|
|  isvisible(self)
|      Return True if the Turtle is shown, False if it's hidden.
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|      >>> print turtle.isvisible():
|      False
|
|  pd = pendown(self)
|      Pull the pen down -- drawing when moving.
|
|      Aliases: pendown | pd | down
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pendown()
|
|  pen(self, pen=None, **pendict)
|      Return or set the pen's attributes.
|
|      Arguments:
|          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)
|         "shearfactor":   number
|         "outline"    :   positive number
|         "tilt"       :   number
|
|      This dictionary can be used as argument for a subsequent
|      pen()-call 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.
|
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.pen(fillcolor="black", pencolor="red", pensize=10)
|      >>> turtle.pen()
|      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
|      'pencolor': 'red', 'pendown': True, 'fillcolor': 'black',
|      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
|      >>> penstate=turtle.pen()
|      >>> turtle.color("yellow","")
|      >>> turtle.penup()
|      >>> turtle.pen()
|      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
|      'pencolor': 'yellow', 'pendown': False, 'fillcolor': '',
|      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
|      >>> p.pen(penstate, fillcolor="green")
|      >>> p.pen()
|      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
|      'pencolor': 'red', 'pendown': True, 'fillcolor': 'green',
|      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
|
|  pencolor(self, *args)
|      Return or set the pencolor.
|
|      Arguments:
|      Four input formats are allowed:
|        - pencolor()
|          Return the current pencolor as color specification string,
|          possibly in hex-number format (see example).
|          May be used as input to another color/pencolor/fillcolor call.
|        - pencolor(colorstring)
|          s is a Tk color specification string, such as "red" or "yellow"
|        - pencolor((r, g, b))
|           *a tuple* of r, g, and b, which represent, an RGB color,
|           and each of r, g, and b are in the range 0..colormode,
|           where colormode is either 1.0 or 255
|         - pencolor(r, g, b)
|           r, g, and b represent an RGB color, and each of r, g, and b
|           are in the range 0..colormode
|
|       If turtleshape is a polygon, the outline of that polygon is drawn
|       with the newly set pencolor.
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.pencolor('brown')
|       >>> tup = (0.2, 0.8, 0.55)
|       >>> turtle.pencolor(tup)
|       >>> turtle.pencolor()
|       '#33cc8c'
|
|   pendown(self)
|       Pull the pen down -- drawing when moving.
|
|       Aliases: pendown | pd | down
|
|       No argument.
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.pendown()
|
|   pensize(self, width=None)
|       Set or return the line thickness.
|
|       Aliases:  pensize | width
|
|       Argument:
|       width -- 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, current pensize
|       is returned.
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.pensize()
|       1
|       >>> turtle.pensize(10)   # from here on lines of width 10 are drawn
|
|   penup(self)
|       Pull the pen up -- no drawing when moving.
|
|      Aliases: penup | pu | up
|
|      No argument
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|
|  pu = penup(self)
|      Pull the pen up -- no drawing when moving.
|
|      Aliases: penup | pu | up
|
|      No argument
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|
|  resizemode(self, rmode=None)
|      Set resizemode to one of the values: "auto", "user", "noresize".
|
|      (Optional) Argument:
|      rmode -- one of the strings "auto", "user", "noresize"
|
|      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()
|        - "noresize" no adaption of the turtle's appearance takes place.
|      If no argument is given, return current resizemode.
|      resizemode("user") is called by a call of shapesize with arguments.
|
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.resizemode("noresize")
|      >>> turtle.resizemode()
|      'noresize'
|
|  showturtle(self)
|      Makes the turtle visible.
|
|      Aliases: showturtle | st
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
```

```
 |      >>> turtle.showturtle()
 |
 |  speed(self, speed=None)
 |      Return or set the turtle's speed.
 |
 |      Optional argument:
 |      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 in the following way:
 |          '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 : *no* animation takes place. forward/back makes turtle
jump
 |      and likewise left/right make the turtle turn instantly.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.speed(3)
 |
 |  st = showturtle(self)
 |      Makes the turtle visible.
 |
 |      Aliases: showturtle | st
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.hideturtle()
 |      >>> turtle.showturtle()
 |
 |  up = penup(self)
 |      Pull the pen up -- no drawing when moving.
 |
 |      Aliases: penup | pu | up
 |
 |      No argument
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.penup()
 |
 |  width = pensize(self, width=None)
 |      Set or return the line thickness.
 |
 |      Aliases:  pensize | width
 |
 |      Argument:
 |      width -- 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, current pensize
 |      is returned.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.pensize()
 |      1
 |      >>> turtle.pensize(10)   # from here on lines of width 10 are drawn
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from TPen:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from TNavigator:
 |
 |  back(self, distance)
 |      Move the turtle backward by distance.
 |
 |      Aliases: back | backward | bk
 |
 |      Argument:
 |      distance -- a number
 |
 |      Move the turtle backward by distance ,opposite to the direction the
 |      turtle is headed. Do not change the turtle's heading.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.position()
 |      (0.00, 0.00)
 |      >>> turtle.backward(30)
```

```
|      >>> turtle.position()
|      (-30.00, 0.00)
|
|  backward = back(self, distance)
|      Move the turtle backward by distance.
|
|      Aliases: back | backward | bk
|
|      Argument:
|      distance -- a number
|
|      Move the turtle backward by distance ,opposite to the direction the
|      turtle is headed. Do not change the turtle's heading.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.backward(30)
|      >>> turtle.position()
|      (-30.00, 0.00)
|
|  bk = back(self, distance)
|      Move the turtle backward by distance.
|
|      Aliases: back | backward | bk
|
|      Argument:
|      distance -- a number
|
|      Move the turtle backward by distance ,opposite to the direction the
|      turtle is headed. Do not change the turtle's heading.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.backward(30)
|      >>> turtle.position()
|      (-30.00, 0.00)
|
|  circle(self, radius, extent=None, steps=None)
|      Draw a circle with given radius.
|
|      Arguments:
|      radius -- a number
|      extent (optional) -- a number
|      steps (optional) -- an integer
|
|      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. Maybe used to draw regular
     |      polygons.
     |
     |      call: circle(radius)                  # full circle
     |      --or: circle(radius, extent)          # arc
     |      --or: circle(radius, extent, steps)
     |      --or: circle(radius, steps=6)         # 6-sided polygon
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.circle(50)
     |      >>> turtle.circle(120, 180)  # semicircle
     |
     |  degrees(self, fullcircle=360.0)
     |      Set angle measurement units to degrees.
     |
     |      Optional argument:
     |      fullcircle -  a number
     |
     |      Set angle measurement units, i. e. set number
     |      of 'degrees' for a full circle. Dafault value is
     |      360 degrees.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.left(90)
     |      >>> turtle.heading()
     |      90
     |
     |      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
     |
     |  distance(self, x, y=None)
     |      Return the distance from the turtle to (x,y) in turtle step units.
     |
     |      Arguments:
     |      x -- a number   or  a pair/vector of numbers   or   a turtle
instance
```

```
|      y -- a number       None                            None
|
|      call: distance(x, y)         # two coordinates
|      --or: distance((x, y))       # a pair (tuple) of coordinates
|      --or: distance(vec)          # e.g. as returned by pos()
|      --or: distance(mypen)        # where mypen is another turtle
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pos()
|      (0.00, 0.00)
|      >>> turtle.distance(30,40)
|      50.0
|      >>> pen = Turtle()
|      >>> pen.forward(77)
|      >>> turtle.distance(pen)
|      77.0
|
|  fd = forward(self, distance)
|      Move the turtle forward by the specified distance.
|
|      Aliases: forward | fd
|
|      Argument:
|      distance -- a number (integer or float)
|
|      Move the turtle forward by the specified distance, in the direction
|      the turtle is headed.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.forward(25)
|      >>> turtle.position()
|      (25.00,0.00)
|      >>> turtle.forward(-75)
|      >>> turtle.position()
|      (-50.00,0.00)
|
|  forward(self, distance)
|      Move the turtle forward by the specified distance.
|
|      Aliases: forward | fd
|
|      Argument:
|      distance -- a number (integer or float)
|
|      Move the turtle forward by the specified distance, in the direction
|      the turtle is headed.
```

```
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.forward(25)
|      >>> turtle.position()
|      (25.00,0.00)
|      >>> turtle.forward(-75)
|      >>> turtle.position()
|      (-50.00,0.00)
|
|  goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
|
|  heading(self)
|      Return the turtle's current heading.
|
|      No arguments.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.left(67)
|      >>> turtle.heading()
|      67.0
|
|  home(self)
|      Move turtle to the origin - coordinates (0,0).
|
|      No arguments.
|
|      Move turtle to the origin - coordinates (0,0) and set its
|      heading to its start-orientation (which depends on mode).
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.home()
|
|  left(self, angle)
|      Turn turtle left by angle units.
|
|      Aliases: left | lt
|
|      Argument:
|      angle -- a number (integer or float)
|
|      Turn turtle left by angle units. (Units are by default degrees,
|      but can be set via the degrees() and radians() functions.)
|      Angle orientation depends on mode. (See this.)
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      22.0
|      >>> turtle.left(45)
|      >>> turtle.heading()
|      67.0
|
|  lt = left(self, angle)
|      Turn turtle left by angle units.
|
|      Aliases: left | lt
|
|      Argument:
|      angle -- a number (integer or float)
|
|      Turn turtle left by angle units. (Units are by default degrees,
|      but can be set via the degrees() and radians() functions.)
|      Angle orientation depends on mode. (See this.)
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      22.0
|      >>> turtle.left(45)
|      >>> turtle.heading()
|      67.0
|
|  pos(self)
|      Return the turtle's current location (x,y), as a Vec2D-vector.
|
|      Aliases: pos | position
|
|      No arguments.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pos()
|      (0.00, 240.00)
|
|  position = pos(self)
|      Return the turtle's current location (x,y), as a Vec2D-vector.
|
|      Aliases: pos | position
|
|      No arguments.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pos()
|      (0.00, 240.00)
|
|  radians(self)
|      Set the angle measurement units to radians.
|
|      No arguments.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      90
|      >>> turtle.radians()
|      >>> turtle.heading()
|      1.5707963267948966
|
|  right(self, angle)
|      Turn turtle right by angle units.
|
|      Aliases: right | rt
|
|      Argument:
|      angle -- a number (integer or float)
|
|      Turn turtle right by angle units. (Units are by default degrees,
|      but can be set via the degrees() and radians() functions.)
|      Angle orientation depends on mode. (See this.)
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      22.0
|      >>> turtle.right(45)
|      >>> turtle.heading()
|      337.0
|
|  rt = right(self, angle)
|      Turn turtle right by angle units.
|
|      Aliases: right | rt
|
|      Argument:
|      angle -- a number (integer or float)
|
|      Turn turtle right by angle units. (Units are by default degrees,
|      but can be set via the degrees() and radians() functions.)
|      Angle orientation depends on mode. (See this.)
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      22.0
|      >>> turtle.right(45)
|      >>> turtle.heading()
|      337.0
|
|  seth = setheading(self, to_angle)
|      Set the orientation of the turtle to to_angle.
|
|      Aliases:  setheading | seth
|
|      Argument:
|      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
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.setheading(90)
|      >>> turtle.heading()
|      90
|
|  setheading(self, to_angle)
|      Set the orientation of the turtle to to_angle.
|
|      Aliases:  setheading | seth
|
|      Argument:
|      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
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.setheading(90)
|      >>> turtle.heading()
|      90
|
|  setpos = goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
|
|  setposition = goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
|
|  setx(self, x)
|      Set the turtle's first coordinate to x
|
|      Argument:
|      x -- a number (integer or float)
|
|      Set the turtle's first coordinate to x, leave second coordinate
|      unchanged.
|
|      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 240.00)
     |      >>> turtle.setx(10)
     |      >>> turtle.position()
     |      (10.00, 240.00)
     |
     |  sety(self, y)
     |      Set the turtle's second coordinate to y
     |
     |      Argument:
     |      y -- a number (integer or float)
     |
     |      Set the turtle's first coordinate to x, second coordinate remains
     |      unchanged.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 40.00)
     |      >>> turtle.sety(-10)
     |      >>> turtle.position()
     |      (0.00, -10.00)
     |
     |  towards(self, x, y=None)
     |      Return the angle of the line from the turtle's position to (x, y).
     |
     |      Arguments:
     |      x -- a number   or  a pair/vector of numbers   or   a turtle
instance
     |      y -- a number       None                            None
     |
     |      call: distance(x, y)         # two coordinates
     |      --or: distance((x, y))       # a pair (tuple) of coordinates
     |      --or: distance(vec)          # e.g. as returned by pos()
     |      --or: distance(mypen)        # where mypen is another turtle
     |
     |      Return the angle, between the line from turtle-position to position
     |      specified by x, y and the turtle's start orientation. (Depends on
     |      modes - "standard" or "logo")
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.pos()
     |      (10.00, 10.00)
     |      >>> turtle.towards(0,0)
     |      225.0
     |
     |  xcor(self)
     |      Return the turtle's x coordinate.
     |
     |      No arguments.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> reset()
     |      >>> turtle.left(60)
     |      >>> turtle.forward(100)
     |      >>> print turtle.xcor()
     |      50.0
     |
     |  ycor(self)
     |      Return the turtle's y coordinate
     |      ---
     |      No arguments.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> reset()
     |      >>> turtle.left(60)
     |      >>> turtle.forward(100)
     |      >>> print turtle.ycor()
     |      86.6025403784
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from TNavigator:
     |
     |  DEFAULT_ANGLEOFFSET = 0
     |
     |  DEFAULT_ANGLEORIENT = 1
     |
     |  DEFAULT_MODE = 'standard'
     |
     |  START_ORIENTATION = {'logo': (0.00,1.00), 'standard': (1.00,0.00),
'wo…

    RawPen = class RawTurtle(TPen, TNavigator)
     |  Animation part of the RawTurtle.
     |  Puts RawTurtle upon a TurtleScreen and provides tools for
     |  its animation.
     |
     |  Method resolution order:
     |      RawTurtle
     |      TPen
     |      TNavigator
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __init__(self, canvas=None, shape='classic', undobuffersize=1000,
visible=True)
```

```
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  begin_fill(self)
 |      Called just before drawing a shape to be filled.
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.color("black", "red")
 |      >>> turtle.begin_fill()
 |      >>> turtle.circle(60)
 |      >>> turtle.end_fill()
 |
 |  begin_poly(self)
 |      Start recording the vertices of a polygon.
 |
 |      No argument.
 |
 |      Start recording the vertices of a polygon. Current turtle position
 |      is first point of polygon.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.begin_poly()
 |
 |  clear(self)
 |      Delete the turtle's drawings from the screen. Do not move turtle.
 |
 |      No arguments.
 |
 |      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.
 |
 |      Examples (for a Turtle instance named turtle):
 |      >>> turtle.clear()
 |
 |  clearstamp(self, stampid)
 |      Delete stamp with given stampid
 |
 |      Argument:
 |      stampid - an integer, must be return value of previous stamp() call.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.color("blue")
 |      >>> astamp = turtle.stamp()
 |      >>> turtle.fd(50)
 |      >>> turtle.clearstamp(astamp)
 |
     |  clearstamps(self, n=None)
     |      Delete all or first/last n of turtle's stamps.
     |
     |      Optional argument:
     |      n -- an integer
     |
     |      If n is None, delete all of pen's stamps,
     |      else if n > 0 delete first n stamps
     |      else if n < 0 delete last n stamps.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> for i in range(8):
     |      …     turtle.stamp(); turtle.fd(30)
     |      …
     |      >>> turtle.clearstamps(2)
     |      >>> turtle.clearstamps(-2)
     |      >>> turtle.clearstamps()
     |
     |  clone(self)
     |      Create and return a clone of the turtle.
     |
     |      No argument.
     |
     |      Create and return a clone of the turtle with same position, heading
     |      and turtle properties.
     |
     |      Example (for a Turtle instance named mick):
     |      mick = Turtle()
     |      joe = mick.clone()
     |
     |  dot(self, size=None, *color)
     |      Draw a dot with diameter size, using color.
     |
     |      Optional arguments:
     |      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.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.dot()
     |      >>> turtle.fd(50); turtle.dot(20, "blue"); turtle.fd(50)
     |
     |  end_fill(self)
     |      Fill the shape drawn after the call begin_fill().
     |
     |       No argument.
     |
     |       Example (for a Turtle instance named turtle):
     |       >>> turtle.color("black", "red")
     |       >>> turtle.begin_fill()
     |       >>> turtle.circle(60)
     |       >>> turtle.end_fill()
     |
     |   end_poly(self)
     |       Stop recording the vertices of a polygon.
     |
     |       No argument.
     |
     |       Stop recording the vertices of a polygon. Current turtle position is
     |       last point of polygon. This will be connected with the first point.
     |
     |       Example (for a Turtle instance named turtle):
     |       >>> turtle.end_poly()
     |
     |   filling(self)
     |       Return fillstate (True if filling, False else).
     |
     |       No argument.
     |
     |       Example (for a Turtle instance named turtle):
     |       >>> turtle.begin_fill()
     |       >>> if turtle.filling():
     |       …     turtle.pensize(5)
     |       … else:
     |       …     turtle.pensize(3)
     |
     |   get_poly(self)
     |       Return the lastly recorded polygon.
     |
     |       No argument.
     |
     |       Example (for a Turtle instance named turtle):
     |       >>> p = turtle.get_poly()
     |       >>> turtle.register_shape("myFavouriteShape", p)
     |
     |   get_shapepoly(self)
     |       Return the current shape polygon as tuple of coordinate pairs.
     |
     |       No argument.
     |
     |       Examples (for a Turtle instance named turtle):
     |       >>> turtle.shape("square")
     |       >>> turtle.shapetransform(4, -1, 0, 2)
```

```
     |      >>> turtle.get_shapepoly()
     |      ((50, -20), (30, 20), (-50, 20), (-30, -20))
     |
     |  getpen = getturtle(self)
     |
     |  getscreen(self)
     |      Return the TurtleScreen object, the turtle is drawing  on.
     |
     |      No argument.
     |
     |      Return the TurtleScreen object, the turtle is drawing  on.
     |      So TurtleScreen-methods can be called for that object.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> ts = turtle.getscreen()
     |      >>> ts
     |      <turtle.TurtleScreen object at 0x0106B770>
     |      >>> ts.bgcolor("pink")
     |
     |  getturtle(self)
     |      Return the Turtleobject itself.
     |
     |      No argument.
     |
     |      Only reasonable use: as a function to return the 'anonymous turtle':
     |
     |      Example:
     |      >>> pet = getturtle()
     |      >>> pet.fd(50)
     |      >>> pet
     |      <turtle.Turtle object at 0x0187D810>
     |      >>> turtles()
     |      [<turtle.Turtle object at 0x0187D810>]
     |
     |  onclick(self, fun, btn=1, add=None)
     |      Bind fun to mouse-click event on this turtle on canvas.
     |
     |      Arguments:
     |      fun --  a function with two arguments, to which will be assigned
     |              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, new binding will be added, otherwise
     |              it will replace a former binding.
     |
     |      Example for the anonymous turtle, i. e. the procedural way:
     |
     |      >>> def turn(x, y):
```

```
     |      …     left(360)
     |      …
     |      >>> onclick(turn)  # Now clicking into the turtle will turn it.
     |      >>> onclick(None)  # event-binding will be removed
     |
     |  ondrag(self, fun, btn=1, add=None)
     |      Bind fun to mouse-move event on this turtle on canvas.
     |
     |      Arguments:
     |      fun -- a function with two arguments, to which will be assigned
     |             the coordinates of the clicked point on the canvas.
     |      num -- number of the mouse-button defaults to 1 (left mouse button).
     |
     |      Every sequence of mouse-move-events on a turtle is preceded by a
     |      mouse-click event on that turtle.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.ondrag(turtle.goto)
     |
     |      Subsequently clicking and dragging a Turtle will move it
     |      across the screen thereby producing handdrawings (if pen is
     |      down).
     |
     |  onrelease(self, fun, btn=1, add=None)
     |      Bind fun to mouse-button-release event on this turtle on canvas.
     |
     |      Arguments:
     |      fun -- a function with two arguments, to which will be assigned
     |              the coordinates of the clicked point on the canvas.
     |      num --  number of the mouse-button defaults to 1 (left mouse
button).
     |
     |      Example (for a MyTurtle instance named joe):
     |      >>> class MyTurtle(Turtle):
     |      …     def glow(self,x,y):
     |      …             self.fillcolor("red")
     |      …     def unglow(self,x,y):
     |      …             self.fillcolor("")
     |      …
     |      >>> joe = MyTurtle()
     |      >>> joe.onclick(joe.glow)
     |      >>> joe.onrelease(joe.unglow)
     |
     |      Clicking on joe turns fillcolor red, unclicking turns it to
     |      transparent.
     |
     |  reset(self)
     |      Delete the turtle's drawings and restore its default values.
     |
     |      No argument.
     |
     |      Delete the turtle's drawings from the screen, re-center the turtle
     |      and set variables to the default values.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00,-22.00)
     |      >>> turtle.heading()
     |      100.0
     |      >>> turtle.reset()
     |      >>> turtle.position()
     |      (0.00,0.00)
     |      >>> turtle.heading()
     |      0.0
     |
     |  settiltangle(self, angle)
     |      Rotate the turtleshape to point in the specified direction
     |
     |      Argument: angle -- 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).
     |
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.settiltangle(45)
     |      >>> stamp()
     |      >>> turtle.fd(50)
     |      >>> turtle.settiltangle(-45)
     |      >>> stamp()
     |      >>> turtle.fd(50)
     |
     |  setundobuffer(self, size)
     |      Set or disable undobuffer.
     |
     |      Argument:
     |      size -- an integer or None
     |
     |      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() function.
     |      If size is None, no undobuffer is present.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.setundobuffer(42)
     |
     |  shape(self, name=None)
     |      Set turtle shape to shape with given name / return current 
shapename.
     |
     |      Optional argument:
     |      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.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.shape()
     |      'arrow'
     |      >>> turtle.shape("turtle")
     |      >>> turtle.shape()
     |      'turtle'
     |
     |  shapesize(self, stretch_wid=None, stretch_len=None, outline=None)
     |      Set/return turtle's stretchfactors/outline. Set resizemode to 
"user".
     |
     |      Optional arguments:
     |         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 orientation
     |      stretch_len is stretchfactor in direction of turtles orientation.
     |      outline determines the width of the shapes's outline.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.resizemode("user")
     |      >>> turtle.shapesize(5, 5, 12)
     |      >>> turtle.shapesize(outline=8)
     |
     |  shapetransform(self, t11=None, t12=None, t21=None, t22=None)
     |      Set or return the current transformation matrix of the turtle shape.
     |
     |      Optional arguments: t11, t12, t21, t22 -- numbers.
     |
     |      If none of the matrix elements are given, return the transformation
     |      matrix.
     |      Otherwise set the given elements and transform the turtleshape
     |      according to the matrix consisting of first row t11, t12 and
     |      second row t21, 22.
     |      Modify stretchfactor, shearfactor and tiltangle according to the
     |      given matrix.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("square")
     |      >>> turtle.shapesize(4,2)
     |      >>> turtle.shearfactor(-0.5)
     |      >>> turtle.shapetransform()
     |      (4.0, -1.0, -0.0, 2.0)
     |
     |  shearfactor(self, shear=None)
     |      Set or return the current shearfactor.
     |
     |      Optional argument: shear -- number, tangent of the shear angle
     |
     |      Shear the turtleshape according to the given shearfactor shear,
     |      which is the tangent of the shear angle. DO NOT change the
     |      turtle's heading (direction of movement).
     |      If shear is not given: return the current shearfactor, i. e. the
     |      tangent of the shear angle, by which lines parallel to the
     |      heading of the turtle are sheared.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.shearfactor(0.5)
     |      >>> turtle.shearfactor()
     |      >>> 0.5
     |
     |  stamp(self)
     |      Stamp a copy of the turtleshape onto the canvas and return its id.
     |
     |      No argument.
     |
     |      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).
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color("blue")
     |      >>> turtle.stamp()
     |      13
     |      >>> turtle.fd(50)
     |
     |  tilt(self, angle)
     |      Rotate the turtleshape by angle.
     |
     |      Argument:
     |      angle - a number
     |
     |      Rotate the turtleshape by angle from its current tilt-angle,
     |      but do NOT change the turtle's heading (direction of movement).
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.tilt(30)
     |      >>> turtle.fd(50)
     |      >>> turtle.tilt(30)
     |      >>> turtle.fd(50)
     |
     |  tiltangle(self, angle=None)
     |      Set or return the current tilt-angle.
     |
     |      Optional argument: angle -- 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).
     |      If angle is not given: 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).
     |
     |      Deprecated since Python 3.1
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.tilt(45)
     |      >>> turtle.tiltangle()
     |
     |  turtlesize = shapesize(self, stretch_wid=None, stretch_len=None,
outline=None)
```

```
|
|  undo(self)
|      undo (repeatedly) the last turtle action.
|
|      No argument.
|
|      undo (repeatedly) the last turtle action.
|      Number of available undo actions is determined by the size of
|      the undobuffer.
|
|      Example (for a Turtle instance named turtle):
|      >>> for i in range(4):
|      …     turtle.fd(50); turtle.lt(80)
|      …
|      >>> for i in range(8):
|      …     turtle.undo()
|      …
|
|  undobufferentries(self)
|      Return count of entries in the undobuffer.
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> while undobufferentries():
|      …     undo()
|
|  write(self, arg, move=False, align='left', font=('Arial', 8, 'normal'))
|      Write text at the current turtle position.
|
|      Arguments:
|      arg -- info, which is to be written to the TurtleScreen
|      move (optional) -- True/False
|      align (optional) -- one of the strings "left", "center" or right"
|      font (optional) -- 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.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.write('Home = ', True, align="center")
|      >>> turtle.write((0,0), True)
|
|  ----------------------------------------------------------------------
|  Data and other attributes defined here:
```

```
|
|  screens = []
|
|  ----------------------------------------------------------------------
|  Methods inherited from TPen:
|
|  color(self, *args)
|      Return or set the pencolor and fillcolor.
|
|      Arguments:
|      Several input formats are allowed.
|      They use 0, 1, 2, or 3 arguments as follows:
|
|      color()
|          Return the current pencolor and the current fillcolor
|          as a pair of color specification strings as are returned
|          by pencolor and fillcolor.
|      color(colorstring), color((r,g,b)), color(r,g,b)
|          inputs as in 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.
|      For mor info see: pencolor, fillcolor
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.color('red', 'green')
|      >>> turtle.color()
|      ('red', 'green')
|      >>> colormode(255)
|      >>> color((40, 80, 120), (160, 200, 240))
|      >>> color()
|      ('#285078', '#a0c8f0')
|
|  down = pendown(self)
|      Pull the pen down -- drawing when moving.
|
|      Aliases: pendown | pd | down
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pendown()
|
|  fillcolor(self, *args)
|      Return or set the fillcolor.
|
|      Arguments:
|      Four input formats are allowed:
|        - fillcolor()
|          Return the current fillcolor as color specification string,
|          possibly in hex-number format (see example).
|          May be used as input to another color/pencolor/fillcolor call.
|        - fillcolor(colorstring)
|          s is a Tk color specification string, such as "red" or "yellow"
|        - fillcolor((r, g, b))
|          *a tuple* of r, g, and b, which represent, an RGB color,
|          and each of r, g, and b are in the range 0..colormode,
|          where colormode is either 1.0 or 255
|        - fillcolor(r, g, b)
|          r, g, and b represent an RGB color, and each of r, g, and b
|          are in the range 0..colormode
|
|      If turtleshape is a polygon, the interior of that polygon is drawn
|      with the newly set fillcolor.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.fillcolor('violet')
|      >>> col = turtle.pencolor()
|      >>> turtle.fillcolor(col)
|      >>> turtle.fillcolor(0, .5, 0)
|
|  hideturtle(self)
|      Makes the turtle invisible.
|
|      Aliases: hideturtle | ht
|
|      No argument.
|
|      It's a good idea to do this while you're in the
|      middle of a complicated drawing, because hiding
|      the turtle speeds up the drawing observably.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|
|  ht = hideturtle(self)
|      Makes the turtle invisible.
|
|      Aliases: hideturtle | ht
|
|      No argument.
```

```
 |
 |      It's a good idea to do this while you're in the
 |      middle of a complicated drawing, because hiding
 |      the turtle speeds up the drawing observably.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.hideturtle()
 |
 |  isdown(self)
 |      Return True if pen is down, False if it's up.
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.penup()
 |      >>> turtle.isdown()
 |      False
 |      >>> turtle.pendown()
 |      >>> turtle.isdown()
 |      True
 |
 |  isvisible(self)
 |      Return True if the Turtle is shown, False if it's hidden.
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.hideturtle()
 |      >>> print turtle.isvisible():
 |      False
 |
 |  pd = pendown(self)
 |      Pull the pen down -- drawing when moving.
 |
 |      Aliases: pendown | pd | down
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.pendown()
 |
 |  pen(self, pen=None, **pendict)
 |      Return or set the pen's attributes.
 |
 |      Arguments:
 |          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)
|         "shearfactor":   number
|         "outline"    :   positive number
|         "tilt"       :   number
|
|      This dictionary can be used as argument for a subsequent
|      pen()-call 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.
|
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.pen(fillcolor="black", pencolor="red", pensize=10)
|      >>> turtle.pen()
|      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
|      'pencolor': 'red', 'pendown': True, 'fillcolor': 'black',
|      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
|      >>> penstate=turtle.pen()
|      >>> turtle.color("yellow","")
|      >>> turtle.penup()
|      >>> turtle.pen()
|      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
|      'pencolor': 'yellow', 'pendown': False, 'fillcolor': '',
|      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
|      >>> p.pen(penstate, fillcolor="green")
|      >>> p.pen()
|      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
|      'pencolor': 'red', 'pendown': True, 'fillcolor': 'green',
|      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
|
|  pencolor(self, *args)
|      Return or set the pencolor.
|
|      Arguments:
|      Four input formats are allowed:
|        - pencolor()
|          Return the current pencolor as color specification string,
|          possibly in hex-number format (see example).
|          May be used as input to another color/pencolor/fillcolor call.
|        - pencolor(colorstring)
|          s is a Tk color specification string, such as "red" or "yellow"
|        - pencolor((r, g, b))
|          *a tuple* of r, g, and b, which represent, an RGB color,
|          and each of r, g, and b are in the range 0..colormode,
|          where colormode is either 1.0 or 255
|        - pencolor(r, g, b)
|          r, g, and b represent an RGB color, and each of r, g, and b
|          are in the range 0..colormode
|
|      If turtleshape is a polygon, the outline of that polygon is drawn
|      with the newly set pencolor.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pencolor('brown')
|      >>> tup = (0.2, 0.8, 0.55)
|      >>> turtle.pencolor(tup)
|      >>> turtle.pencolor()
|      '#33cc8c'
|
|  pendown(self)
|      Pull the pen down -- drawing when moving.
|
|      Aliases: pendown | pd | down
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pendown()
|
|  pensize(self, width=None)
|      Set or return the line thickness.
|
|      Aliases:  pensize | width
|
|      Argument:
|      width -- 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, current pensize
|      is returned.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pensize()
|      1
|      >>> turtle.pensize(10)   # from here on lines of width 10 are drawn
```

```
|
|  penup(self)
|      Pull the pen up -- no drawing when moving.
|
|      Aliases: penup | pu | up
|
|      No argument
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|
|  pu = penup(self)
|      Pull the pen up -- no drawing when moving.
|
|      Aliases: penup | pu | up
|
|      No argument
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|
|  resizemode(self, rmode=None)
|      Set resizemode to one of the values: "auto", "user", "noresize".
|
|      (Optional) Argument:
|      rmode -- one of the strings "auto", "user", "noresize"
|
|      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()
|        - "noresize" no adaption of the turtle's appearance takes place.
|      If no argument is given, return current resizemode.
|      resizemode("user") is called by a call of shapesize with arguments.
|
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.resizemode("noresize")
|      >>> turtle.resizemode()
|      'noresize'
|
|  showturtle(self)
|      Makes the turtle visible.
|
|      Aliases: showturtle | st
|
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.hideturtle()
     |      >>> turtle.showturtle()
     |
     |  speed(self, speed=None)
     |      Return or set the turtle's speed.
     |
     |      Optional argument:
     |      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 in the following way:
     |          '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 : *no* animation takes place. forward/back makes turtle
jump
     |      and likewise left/right make the turtle turn instantly.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.speed(3)
     |
     |  st = showturtle(self)
     |      Makes the turtle visible.
     |
     |      Aliases: showturtle | st
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.hideturtle()
     |      >>> turtle.showturtle()
     |
     |  up = penup(self)
     |      Pull the pen up -- no drawing when moving.
     |
     |      Aliases: penup | pu | up
     |
     |      No argument
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.penup()
     |
     |  width = pensize(self, width=None)
     |      Set or return the line thickness.
     |
     |      Aliases:  pensize | width
     |
     |      Argument:
     |      width -- 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, current pensize
     |      is returned.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.pensize()
     |      1
     |      >>> turtle.pensize(10)   # from here on lines of width 10 are drawn
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors inherited from TPen:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from TNavigator:
     |
     |  back(self, distance)
     |      Move the turtle backward by distance.
     |
     |      Aliases: back | backward | bk
     |
     |      Argument:
     |      distance -- a number
     |
     |      Move the turtle backward by distance ,opposite to the direction the
     |      turtle is headed. Do not change the turtle's heading.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 0.00)
     |      >>> turtle.backward(30)
     |      >>> turtle.position()
     |      (-30.00, 0.00)
     |
     |  backward = back(self, distance)
     |      Move the turtle backward by distance.
     |
     |      Aliases: back | backward | bk
     |
     |      Argument:
     |      distance -- a number
     |
     |      Move the turtle backward by distance ,opposite to the direction the
     |      turtle is headed. Do not change the turtle's heading.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 0.00)
     |      >>> turtle.backward(30)
     |      >>> turtle.position()
     |      (-30.00, 0.00)
     |
     |  bk = back(self, distance)
     |      Move the turtle backward by distance.
     |
     |      Aliases: back | backward | bk
     |
     |      Argument:
     |      distance -- a number
     |
     |      Move the turtle backward by distance ,opposite to the direction the
     |      turtle is headed. Do not change the turtle's heading.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 0.00)
     |      >>> turtle.backward(30)
     |      >>> turtle.position()
     |      (-30.00, 0.00)
     |
     |  circle(self, radius, extent=None, steps=None)
     |      Draw a circle with given radius.
     |
     |      Arguments:
     |      radius -- a number
     |      extent (optional) -- a number
     |      steps (optional) -- an integer
     |
     |      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. Maybe used to draw regular
     |      polygons.
     |
     |      call: circle(radius)                  # full circle
     |      --or: circle(radius, extent)          # arc
     |      --or: circle(radius, extent, steps)
     |      --or: circle(radius, steps=6)         # 6-sided polygon
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.circle(50)
     |      >>> turtle.circle(120, 180)  # semicircle
     |
     |  degrees(self, fullcircle=360.0)
     |      Set angle measurement units to degrees.
     |
     |      Optional argument:
     |      fullcircle -  a number
     |
     |      Set angle measurement units, i. e. set number
     |      of 'degrees' for a full circle. Dafault value is
     |      360 degrees.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.left(90)
     |      >>> turtle.heading()
     |      90
     |
     |      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
     |
     |  distance(self, x, y=None)
     |      Return the distance from the turtle to (x,y) in turtle step units.
```

```
     |
     |      Arguments:
     |      x -- a number   or  a pair/vector of numbers   or   a turtle
instance
     |      y -- a number       None                            None
     |
     |      call: distance(x, y)         # two coordinates
     |      --or: distance((x, y))       # a pair (tuple) of coordinates
     |      --or: distance(vec)          # e.g. as returned by pos()
     |      --or: distance(mypen)        # where mypen is another turtle
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.pos()
     |      (0.00, 0.00)
     |      >>> turtle.distance(30,40)
     |      50.0
     |      >>> pen = Turtle()
     |      >>> pen.forward(77)
     |      >>> turtle.distance(pen)
     |      77.0
     |
     |  fd = forward(self, distance)
     |      Move the turtle forward by the specified distance.
     |
     |      Aliases: forward | fd
     |
     |      Argument:
     |      distance -- a number (integer or float)
     |
     |      Move the turtle forward by the specified distance, in the direction
     |      the turtle is headed.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 0.00)
     |      >>> turtle.forward(25)
     |      >>> turtle.position()
     |      (25.00,0.00)
     |      >>> turtle.forward(-75)
     |      >>> turtle.position()
     |      (-50.00,0.00)
     |
     |  forward(self, distance)
     |      Move the turtle forward by the specified distance.
     |
     |      Aliases: forward | fd
     |
     |      Argument:
```

```
|      distance -- a number (integer or float)
|
|      Move the turtle forward by the specified distance, in the direction
|      the turtle is headed.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.forward(25)
|      >>> turtle.position()
|      (25.00,0.00)
|      >>> turtle.forward(-75)
|      >>> turtle.position()
|      (-50.00,0.00)
|
|  goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
|
|  heading(self)
|      Return the turtle's current heading.
|
|      No arguments.
|      
|      Example (for a Turtle instance named turtle):
|      >>> turtle.left(67)
|      >>> turtle.heading()
|      67.0
|  
|  home(self)
|      Move turtle to the origin - coordinates (0,0).
|      
|      No arguments.
|      
|      Move turtle to the origin - coordinates (0,0) and set its
|      heading to its start-orientation (which depends on mode).
|      
|      Example (for a Turtle instance named turtle):
|      >>> turtle.home()
|  
|  left(self, angle)
|      Turn turtle left by angle units.
|      
|      Aliases: left | lt
|      
|      Argument:
|      angle -- a number (integer or float)
|      
|      Turn turtle left by angle units. (Units are by default degrees,
|      but can be set via the degrees() and radians() functions.)
|      Angle orientation depends on mode. (See this.)
|      
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      22.0
|      >>> turtle.left(45)
|      >>> turtle.heading()
|      67.0
|  
|  lt = left(self, angle)
|      Turn turtle left by angle units.
|      
|      Aliases: left | lt
|      
|      Argument:
|      angle -- a number (integer or float)
|      
|      Turn turtle left by angle units. (Units are by default degrees,
|      but can be set via the degrees() and radians() functions.)
|      Angle orientation depends on mode. (See this.)
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      22.0
|      >>> turtle.left(45)
|      >>> turtle.heading()
|      67.0
|
|  pos(self)
|      Return the turtle's current location (x,y), as a Vec2D-vector.
|
|      Aliases: pos | position
|
|      No arguments.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pos()
|      (0.00, 240.00)
|
|  position = pos(self)
|      Return the turtle's current location (x,y), as a Vec2D-vector.
|
|      Aliases: pos | position
|
|      No arguments.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pos()
|      (0.00, 240.00)
|
|  radians(self)
|      Set the angle measurement units to radians.
|
|      No arguments.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      90
|      >>> turtle.radians()
|      >>> turtle.heading()
|      1.5707963267948966
|
|  right(self, angle)
|      Turn turtle right by angle units.
|
|      Aliases: right | rt
|
|      Argument:
```

```
|       angle -- a number (integer or float)
|
|       Turn turtle right by angle units. (Units are by default degrees,
|       but can be set via the degrees() and radians() functions.)
|       Angle orientation depends on mode. (See this.)
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.heading()
|       22.0
|       >>> turtle.right(45)
|       >>> turtle.heading()
|       337.0
|
|   rt = right(self, angle)
|       Turn turtle right by angle units.
|
|       Aliases: right | rt
|
|       Argument:
|       angle -- a number (integer or float)
|
|       Turn turtle right by angle units. (Units are by default degrees,
|       but can be set via the degrees() and radians() functions.)
|       Angle orientation depends on mode. (See this.)
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.heading()
|       22.0
|       >>> turtle.right(45)
|       >>> turtle.heading()
|       337.0
|
|   seth = setheading(self, to_angle)
|       Set the orientation of the turtle to to_angle.
|
|       Aliases:  setheading | seth
|
|       Argument:
|       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
|
|     Example (for a Turtle instance named turtle):
|     >>> turtle.setheading(90)
|     >>> turtle.heading()
|     90
|
|  setheading(self, to_angle)
|     Set the orientation of the turtle to to_angle.
|
|     Aliases:  setheading | seth
|
|     Argument:
|     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
|
|     Example (for a Turtle instance named turtle):
|     >>> turtle.setheading(90)
|     >>> turtle.heading()
|     90
|
|  setpos = goto(self, x, y=None)
|     Move turtle to an absolute position.
|
|     Aliases: setpos | setposition | goto:
|
|     Arguments:
|     x -- a number      or     a pair/vector of numbers
|     y -- a number             None
|
|     call: goto(x, y)         # two coordinates
|     --or: goto((x, y))       # a pair (tuple) of coordinates
|     --or: goto(vec)          # e.g. as returned by pos()
|
|     Move turtle to an absolute position. If the pen is down,
|     a line will be drawn. The turtle's orientation does not change.
|
|     Example (for a Turtle instance named turtle):
|     >>> 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)
|
|  setposition = goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
|
|  setx(self, x)
|      Set the turtle's first coordinate to x
|
|      Argument:
|      x -- a number (integer or float)
|
     |      Set the turtle's first coordinate to x, leave second coordinate
     |      unchanged.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 240.00)
     |      >>> turtle.setx(10)
     |      >>> turtle.position()
     |      (10.00, 240.00)
     |
     |  sety(self, y)
     |      Set the turtle's second coordinate to y
     |
     |      Argument:
     |      y -- a number (integer or float)
     |
     |      Set the turtle's first coordinate to x, second coordinate remains
     |      unchanged.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 40.00)
     |      >>> turtle.sety(-10)
     |      >>> turtle.position()
     |      (0.00, -10.00)
     |
     |  towards(self, x, y=None)
     |      Return the angle of the line from the turtle's position to (x, y).
     |
     |      Arguments:
     |      x -- a number   or  a pair/vector of numbers   or   a turtle
instance
     |      y -- a number       None                            None
     |
     |      call: distance(x, y)         # two coordinates
     |      --or: distance((x, y))       # a pair (tuple) of coordinates
     |      --or: distance(vec)          # e.g. as returned by pos()
     |      --or: distance(mypen)        # where mypen is another turtle
     |
     |      Return the angle, between the line from turtle-position to position
     |      specified by x, y and the turtle's start orientation. (Depends on
     |      modes - "standard" or "logo")
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.pos()
     |      (10.00, 10.00)
     |      >>> turtle.towards(0,0)
     |      225.0
     |
     |  xcor(self)
     |      Return the turtle's x coordinate.
     |
     |      No arguments.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> reset()
     |      >>> turtle.left(60)
     |      >>> turtle.forward(100)
     |      >>> print turtle.xcor()
     |      50.0
     |
     |  ycor(self)
     |      Return the turtle's y coordinate
     |      ---
     |      No arguments.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> reset()
     |      >>> turtle.left(60)
     |      >>> turtle.forward(100)
     |      >>> print turtle.ycor()
     |      86.6025403784
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from TNavigator:
     |
     |  DEFAULT_ANGLEOFFSET = 0
     |
     |  DEFAULT_ANGLEORIENT = 1
     |
     |  DEFAULT_MODE = 'standard'
     |
     |  START_ORIENTATION = {'logo': (0.00,1.00), 'standard': (1.00,0.00),
'wo…

    class RawTurtle(TPen, TNavigator)
     |  Animation part of the RawTurtle.
     |  Puts RawTurtle upon a TurtleScreen and provides tools for
     |  its animation.
     |
     |  Method resolution order:
     |      RawTurtle
     |      TPen
     |      TNavigator
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __init__(self, canvas=None, shape='classic', undobuffersize=1000,
visible=True)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  begin_fill(self)
     |      Called just before drawing a shape to be filled.
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color("black", "red")
     |      >>> turtle.begin_fill()
     |      >>> turtle.circle(60)
     |      >>> turtle.end_fill()
     |
     |  begin_poly(self)
     |      Start recording the vertices of a polygon.
     |
     |      No argument.
     |
     |      Start recording the vertices of a polygon. Current turtle position
     |      is first point of polygon.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.begin_poly()
     |
     |  clear(self)
     |      Delete the turtle's drawings from the screen. Do not move turtle.
     |
     |      No arguments.
     |
     |      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.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.clear()
     |
     |  clearstamp(self, stampid)
     |      Delete stamp with given stampid
     |
     |      Argument:
     |      stampid - an integer, must be return value of previous stamp() call.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color("blue")
     |      >>> astamp = turtle.stamp()
     |      >>> turtle.fd(50)
     |      >>> turtle.clearstamp(astamp)
     |
     |  clearstamps(self, n=None)
     |      Delete all or first/last n of turtle's stamps.
     |
     |      Optional argument:
     |      n -- an integer
     |
     |      If n is None, delete all of pen's stamps,
     |      else if n > 0 delete first n stamps
     |      else if n < 0 delete last n stamps.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> for i in range(8):
     |      …     turtle.stamp(); turtle.fd(30)
     |      …
     |      >>> turtle.clearstamps(2)
     |      >>> turtle.clearstamps(-2)
     |      >>> turtle.clearstamps()
     |
     |  clone(self)
     |      Create and return a clone of the turtle.
     |
     |      No argument.
     |
     |      Create and return a clone of the turtle with same position, heading
     |      and turtle properties.
     |
     |      Example (for a Turtle instance named mick):
     |      mick = Turtle()
     |      joe = mick.clone()
     |
     |  dot(self, size=None, *color)
     |      Draw a dot with diameter size, using color.
     |
     |      Optional arguments:
     |      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.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.dot()
     |      >>> turtle.fd(50); turtle.dot(20, "blue"); turtle.fd(50)
```

```
|
|  end_fill(self)
|      Fill the shape drawn after the call begin_fill().
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.color("black", "red")
|      >>> turtle.begin_fill()
|      >>> turtle.circle(60)
|      >>> turtle.end_fill()
|
|  end_poly(self)
|      Stop recording the vertices of a polygon.
|
|      No argument.
|
|      Stop recording the vertices of a polygon. Current turtle position is
|      last point of polygon. This will be connected with the first point.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.end_poly()
|
|  filling(self)
|      Return fillstate (True if filling, False else).
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.begin_fill()
|      >>> if turtle.filling():
|      …     turtle.pensize(5)
|      … else:
|      …     turtle.pensize(3)
|
|  get_poly(self)
|      Return the lastly recorded polygon.
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> p = turtle.get_poly()
|      >>> turtle.register_shape("myFavouriteShape", p)
|
|  get_shapepoly(self)
|      Return the current shape polygon as tuple of coordinate pairs.
|
|      No argument.
```

```
 |
 |      Examples (for a Turtle instance named turtle):
 |      >>> turtle.shape("square")
 |      >>> turtle.shapetransform(4, -1, 0, 2)
 |      >>> turtle.get_shapepoly()
 |      ((50, -20), (30, 20), (-50, 20), (-30, -20))
 |
 |  getpen = getturtle(self)
 |
 |  getscreen(self)
 |      Return the TurtleScreen object, the turtle is drawing  on.
 |
 |      No argument.
 |
 |      Return the TurtleScreen object, the turtle is drawing  on.
 |      So TurtleScreen-methods can be called for that object.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> ts = turtle.getscreen()
 |      >>> ts
 |      <turtle.TurtleScreen object at 0x0106B770>
 |      >>> ts.bgcolor("pink")
 |
 |  getturtle(self)
 |      Return the Turtleobject itself.
 |
 |      No argument.
 |
 |      Only reasonable use: as a function to return the 'anonymous turtle':
 |
 |      Example:
 |      >>> pet = getturtle()
 |      >>> pet.fd(50)
 |      >>> pet
 |      <turtle.Turtle object at 0x0187D810>
 |      >>> turtles()
 |      [<turtle.Turtle object at 0x0187D810>]
 |
 |  onclick(self, fun, btn=1, add=None)
 |      Bind fun to mouse-click event on this turtle on canvas.
 |
 |      Arguments:
 |      fun --  a function with two arguments, to which will be assigned
 |              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, new binding will be added, otherwise
 |              it will replace a former binding.
```

```
      |
      |      Example for the anonymous turtle, i. e. the procedural way:
      |
      |      >>> def turn(x, y):
      |      …     left(360)
      |      …
      |      >>> onclick(turn)  # Now clicking into the turtle will turn it.
      |      >>> onclick(None)  # event-binding will be removed
      |
      |  ondrag(self, fun, btn=1, add=None)
      |      Bind fun to mouse-move event on this turtle on canvas.
      |
      |      Arguments:
      |      fun -- a function with two arguments, to which will be assigned
      |             the coordinates of the clicked point on the canvas.
      |      num -- number of the mouse-button defaults to 1 (left mouse button).
      |
      |      Every sequence of mouse-move-events on a turtle is preceded by a
      |      mouse-click event on that turtle.
      |
      |      Example (for a Turtle instance named turtle):
      |      >>> turtle.ondrag(turtle.goto)
      |
      |      Subsequently clicking and dragging a Turtle will move it
      |      across the screen thereby producing handdrawings (if pen is
      |      down).
      |
      |  onrelease(self, fun, btn=1, add=None)
      |      Bind fun to mouse-button-release event on this turtle on canvas.
      |
      |      Arguments:
      |      fun -- a function with two arguments, to which will be assigned
      |              the coordinates of the clicked point on the canvas.
      |      num --  number of the mouse-button defaults to 1 (left mouse
button).
      |
      |      Example (for a MyTurtle instance named joe):
      |      >>> class MyTurtle(Turtle):
      |      …     def glow(self,x,y):
      |      …             self.fillcolor("red")
      |      …     def unglow(self,x,y):
      |      …             self.fillcolor("")
      |      …
      |      >>> joe = MyTurtle()
      |      >>> joe.onclick(joe.glow)
      |      >>> joe.onrelease(joe.unglow)
      |
      |      Clicking on joe turns fillcolor red, unclicking turns it to
      |      transparent.
      |
      |  reset(self)
      |      Delete the turtle's drawings and restore its default values.
      |
      |      No argument.
      |
      |      Delete the turtle's drawings from the screen, re-center the turtle
      |      and set variables to the default values.
      |
      |      Example (for a Turtle instance named turtle):
      |      >>> turtle.position()
      |      (0.00,-22.00)
      |      >>> turtle.heading()
      |      100.0
      |      >>> turtle.reset()
      |      >>> turtle.position()
      |      (0.00,0.00)
      |      >>> turtle.heading()
      |      0.0
      |
      |  settiltangle(self, angle)
      |      Rotate the turtleshape to point in the specified direction
      |
      |      Argument: angle -- 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).
      |
      |
      |      Examples (for a Turtle instance named turtle):
      |      >>> turtle.shape("circle")
      |      >>> turtle.shapesize(5,2)
      |      >>> turtle.settiltangle(45)
      |      >>> stamp()
      |      >>> turtle.fd(50)
      |      >>> turtle.settiltangle(-45)
      |      >>> stamp()
      |      >>> turtle.fd(50)
      |
      |  setundobuffer(self, size)
      |      Set or disable undobuffer.
      |
      |      Argument:
      |      size -- an integer or None
      |
      |      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() function.
     |      If size is None, no undobuffer is present.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.setundobuffer(42)
     |
     |  shape(self, name=None)
     |      Set turtle shape to shape with given name / return current
shapename.
     |
     |      Optional argument:
     |      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.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.shape()
     |      'arrow'
     |      >>> turtle.shape("turtle")
     |      >>> turtle.shape()
     |      'turtle'
     |
     |  shapesize(self, stretch_wid=None, stretch_len=None, outline=None)
     |      Set/return turtle's stretchfactors/outline. Set resizemode to
"user".
     |
     |      Optional arguments:
     |         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 orientation
     |      stretch_len is stretchfactor in direction of turtles orientation.
     |      outline determines the width of the shapes's outline.
```

```
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.resizemode("user")
|      >>> turtle.shapesize(5, 5, 12)
|      >>> turtle.shapesize(outline=8)
|
|  shapetransform(self, t11=None, t12=None, t21=None, t22=None)
|      Set or return the current transformation matrix of the turtle shape.
|
|      Optional arguments: t11, t12, t21, t22 -- numbers.
|
|      If none of the matrix elements are given, return the transformation
|      matrix.
|      Otherwise set the given elements and transform the turtleshape
|      according to the matrix consisting of first row t11, t12 and
|      second row t21, 22.
|      Modify stretchfactor, shearfactor and tiltangle according to the
|      given matrix.
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.shape("square")
|      >>> turtle.shapesize(4,2)
|      >>> turtle.shearfactor(-0.5)
|      >>> turtle.shapetransform()
|      (4.0, -1.0, -0.0, 2.0)
|
|  shearfactor(self, shear=None)
|      Set or return the current shearfactor.
|
|      Optional argument: shear -- number, tangent of the shear angle
|
|      Shear the turtleshape according to the given shearfactor shear,
|      which is the tangent of the shear angle. DO NOT change the
|      turtle's heading (direction of movement).
|      If shear is not given: return the current shearfactor, i. e. the
|      tangent of the shear angle, by which lines parallel to the
|      heading of the turtle are sheared.
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.shape("circle")
|      >>> turtle.shapesize(5,2)
|      >>> turtle.shearfactor(0.5)
|      >>> turtle.shearfactor()
|      >>> 0.5
|
|  stamp(self)
|      Stamp a copy of the turtleshape onto the canvas and return its id.
|
     |      No argument.
     |
     |      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).
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color("blue")
     |      >>> turtle.stamp()
     |      13
     |      >>> turtle.fd(50)
     |
     |  tilt(self, angle)
     |      Rotate the turtleshape by angle.
     |
     |      Argument:
     |      angle - a number
     |
     |      Rotate the turtleshape by angle from its current tilt-angle,
     |      but do NOT change the turtle's heading (direction of movement).
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.tilt(30)
     |      >>> turtle.fd(50)
     |      >>> turtle.tilt(30)
     |      >>> turtle.fd(50)
     |
     |  tiltangle(self, angle=None)
     |      Set or return the current tilt-angle.
     |
     |      Optional argument: angle -- 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).
     |      If angle is not given: 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).
     |
     |      Deprecated since Python 3.1
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.tilt(45)
     |      >>> turtle.tiltangle()
     |
     |  turtlesize = shapesize(self, stretch_wid=None, stretch_len=None,
outline=None)
     |
     |  undo(self)
     |      undo (repeatedly) the last turtle action.
     |
     |      No argument.
     |
     |      undo (repeatedly) the last turtle action.
     |      Number of available undo actions is determined by the size of
     |      the undobuffer.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> for i in range(4):
     |      …     turtle.fd(50); turtle.lt(80)
     |      …
     |      >>> for i in range(8):
     |      …     turtle.undo()
     |      …
     |
     |  undobufferentries(self)
     |      Return count of entries in the undobuffer.
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> while undobufferentries():
     |      …     undo()
     |
     |  write(self, arg, move=False, align='left', font=('Arial', 8, 'normal'))
     |      Write text at the current turtle position.
     |
     |      Arguments:
     |      arg -- info, which is to be written to the TurtleScreen
     |      move (optional) -- True/False
     |      align (optional) -- one of the strings "left", "center" or right"
     |      font (optional) -- 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.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.write('Home = ', True, align="center")
     |      >>> turtle.write((0,0), True)
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes defined here:
     |
     |  screens = []
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from TPen:
     |
     |  color(self, *args)
     |      Return or set the pencolor and fillcolor.
     |
     |      Arguments:
     |      Several input formats are allowed.
     |      They use 0, 1, 2, or 3 arguments as follows:
     |
     |      color()
     |          Return the current pencolor and the current fillcolor
     |          as a pair of color specification strings as are returned
     |          by pencolor and fillcolor.
     |      color(colorstring), color((r,g,b)), color(r,g,b)
     |          inputs as in 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.
     |      For mor info see: pencolor, fillcolor
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color('red', 'green')
     |      >>> turtle.color()
     |      ('red', 'green')
     |      >>> colormode(255)
     |      >>> color((40, 80, 120), (160, 200, 240))
     |      >>> color()
     |      ('#285078', '#a0c8f0')
     |
     |  down = pendown(self)
     |      Pull the pen down -- drawing when moving.
     |
     |      Aliases: pendown | pd | down
     |
     |      No argument.
```

```
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pendown()
|
|  fillcolor(self, *args)
|      Return or set the fillcolor.
|
|      Arguments:
|      Four input formats are allowed:
|        - fillcolor()
|          Return the current fillcolor as color specification string,
|          possibly in hex-number format (see example).
|          May be used as input to another color/pencolor/fillcolor call.
|        - fillcolor(colorstring)
|          s is a Tk color specification string, such as "red" or "yellow"
|        - fillcolor((r, g, b))
|          *a tuple* of r, g, and b, which represent, an RGB color,
|          and each of r, g, and b are in the range 0..colormode,
|          where colormode is either 1.0 or 255
|        - fillcolor(r, g, b)
|          r, g, and b represent an RGB color, and each of r, g, and b
|          are in the range 0..colormode
|
|      If turtleshape is a polygon, the interior of that polygon is drawn
|      with the newly set fillcolor.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.fillcolor('violet')
|      >>> col = turtle.pencolor()
|      >>> turtle.fillcolor(col)
|      >>> turtle.fillcolor(0, .5, 0)
|
|  hideturtle(self)
|      Makes the turtle invisible.
|
|      Aliases: hideturtle | ht
|
|      No argument.
|
|      It's a good idea to do this while you're in the
|      middle of a complicated drawing, because hiding
|      the turtle speeds up the drawing observably.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|
|  ht = hideturtle(self)
|      Makes the turtle invisible.
```

```
 |
 |      Aliases: hideturtle | ht
 |
 |      No argument.
 |
 |      It's a good idea to do this while you're in the
 |      middle of a complicated drawing, because hiding
 |      the turtle speeds up the drawing observably.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.hideturtle()
 |
 |  isdown(self)
 |      Return True if pen is down, False if it's up.
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.penup()
 |      >>> turtle.isdown()
 |      False
 |      >>> turtle.pendown()
 |      >>> turtle.isdown()
 |      True
 |
 |  isvisible(self)
 |      Return True if the Turtle is shown, False if it's hidden.
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.hideturtle()
 |      >>> print turtle.isvisible():
 |      False
 |
 |  pd = pendown(self)
 |      Pull the pen down -- drawing when moving.
 |
 |      Aliases: pendown | pd | down
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.pendown()
 |
 |  pen(self, pen=None, **pendict)
 |      Return or set the pen's attributes.
 |
 |      Arguments:
 |          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)
 |         "shearfactor":   number
 |         "outline"    :   positive number
 |         "tilt"       :   number
 |
 |      This dictionary can be used as argument for a subsequent
 |      pen()-call 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.
 |
 |
 |      Examples (for a Turtle instance named turtle):
 |      >>> turtle.pen(fillcolor="black", pencolor="red", pensize=10)
 |      >>> turtle.pen()
 |      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
 |      'pencolor': 'red', 'pendown': True, 'fillcolor': 'black',
 |      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
 |      >>> penstate=turtle.pen()
 |      >>> turtle.color("yellow","")
 |      >>> turtle.penup()
 |      >>> turtle.pen()
 |      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
 |      'pencolor': 'yellow', 'pendown': False, 'fillcolor': '',
 |      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
 |      >>> p.pen(penstate, fillcolor="green")
 |      >>> p.pen()
 |      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
 |      'pencolor': 'red', 'pendown': True, 'fillcolor': 'green',
 |      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
 |
 |  pencolor(self, *args)
 |      Return or set the pencolor.
 |
 |      Arguments:
```

```
|      Four input formats are allowed:
|        - pencolor()
|          Return the current pencolor as color specification string,
|          possibly in hex-number format (see example).
|          May be used as input to another color/pencolor/fillcolor call.
|        - pencolor(colorstring)
|          s is a Tk color specification string, such as "red" or "yellow"
|        - pencolor((r, g, b))
|          *a tuple* of r, g, and b, which represent, an RGB color,
|          and each of r, g, and b are in the range 0..colormode,
|          where colormode is either 1.0 or 255
|        - pencolor(r, g, b)
|          r, g, and b represent an RGB color, and each of r, g, and b
|          are in the range 0..colormode
|
|      If turtleshape is a polygon, the outline of that polygon is drawn
|      with the newly set pencolor.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pencolor('brown')
|      >>> tup = (0.2, 0.8, 0.55)
|      >>> turtle.pencolor(tup)
|      >>> turtle.pencolor()
|      '#33cc8c'
|
|  pendown(self)
|      Pull the pen down -- drawing when moving.
|
|      Aliases: pendown | pd | down
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pendown()
|
|  pensize(self, width=None)
|      Set or return the line thickness.
|
|      Aliases:  pensize | width
|
|      Argument:
|      width -- 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, current pensize
|      is returned.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pensize()
|      1
|      >>> turtle.pensize(10)   # from here on lines of width 10 are drawn
|
|  penup(self)
|      Pull the pen up -- no drawing when moving.
|
|      Aliases: penup | pu | up
|
|      No argument
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|
|  pu = penup(self)
|      Pull the pen up -- no drawing when moving.
|
|      Aliases: penup | pu | up
|
|      No argument
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|
|  resizemode(self, rmode=None)
|      Set resizemode to one of the values: "auto", "user", "noresize".
|
|      (Optional) Argument:
|      rmode -- one of the strings "auto", "user", "noresize"
|
|      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()
|        - "noresize" no adaption of the turtle's appearance takes place.
|      If no argument is given, return current resizemode.
|      resizemode("user") is called by a call of shapesize with arguments.
|
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.resizemode("noresize")
|      >>> turtle.resizemode()
|      'noresize'
|
|  showturtle(self)
|      Makes the turtle visible.
|
|      Aliases: showturtle | st
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|      >>> turtle.showturtle()
|
|  speed(self, speed=None)
|      Return or set the turtle's speed.
|
|      Optional argument:
|      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 in the following way:
|          '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 : *no* animation takes place. forward/back makes turtle
jump
|      and likewise left/right make the turtle turn instantly.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.speed(3)
|
|  st = showturtle(self)
|      Makes the turtle visible.
|
|      Aliases: showturtle | st
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|      >>> turtle.showturtle()
```

```
|
|  up = penup(self)
|      Pull the pen up -- no drawing when moving.
|
|      Aliases: penup | pu | up
|
|      No argument
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|
|  width = pensize(self, width=None)
|      Set or return the line thickness.
|
|      Aliases:  pensize | width
|
|      Argument:
|      width -- 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, current pensize
|      is returned.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pensize()
|      1
|      >>> turtle.pensize(10)   # from here on lines of width 10 are drawn
|
|  ----------------------------------------------------------------------
|  Data descriptors inherited from TPen:
|
|  __dict__
|      dictionary for instance variables (if defined)
|
|  __weakref__
|      list of weak references to the object (if defined)
|
|  ----------------------------------------------------------------------
|  Methods inherited from TNavigator:
|
|  back(self, distance)
|      Move the turtle backward by distance.
|
|      Aliases: back | backward | bk
|
|      Argument:
|      distance -- a number
```

```
|
|      Move the turtle backward by distance ,opposite to the direction the
|      turtle is headed. Do not change the turtle's heading.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.backward(30)
|      >>> turtle.position()
|      (-30.00, 0.00)
|
|  backward = back(self, distance)
|      Move the turtle backward by distance.
|
|      Aliases: back | backward | bk
|
|      Argument:
|      distance -- a number
|
|      Move the turtle backward by distance ,opposite to the direction the
|      turtle is headed. Do not change the turtle's heading.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.backward(30)
|      >>> turtle.position()
|      (-30.00, 0.00)
|
|  bk = back(self, distance)
|      Move the turtle backward by distance.
|
|      Aliases: back | backward | bk
|
|      Argument:
|      distance -- a number
|
|      Move the turtle backward by distance ,opposite to the direction the
|      turtle is headed. Do not change the turtle's heading.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.backward(30)
|      >>> turtle.position()
|      (-30.00, 0.00)
|
|  circle(self, radius, extent=None, steps=None)
|      Draw a circle with given radius.
|
|      Arguments:
|      radius -- a number
|      extent (optional) -- a number
|      steps (optional) -- an integer
|
|      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. Maybe used to draw regular
|      polygons.
|
|      call: circle(radius)                  # full circle
|      --or: circle(radius, extent)          # arc
|      --or: circle(radius, extent, steps)
|      --or: circle(radius, steps=6)         # 6-sided polygon
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.circle(50)
|      >>> turtle.circle(120, 180)  # semicircle
|
|  degrees(self, fullcircle=360.0)
|      Set angle measurement units to degrees.
|
|      Optional argument:
|      fullcircle -  a number
|
|      Set angle measurement units, i. e. set number
|      of 'degrees' for a full circle. Dafault value is
|      360 degrees.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.left(90)
|      >>> turtle.heading()
|      90
|
|      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
     |
     |  distance(self, x, y=None)
     |      Return the distance from the turtle to (x,y) in turtle step units.
     |
     |      Arguments:
     |      x -- a number   or  a pair/vector of numbers   or   a turtle
instance
     |      y -- a number       None                            None
     |
     |      call: distance(x, y)         # two coordinates
     |      --or: distance((x, y))       # a pair (tuple) of coordinates
     |      --or: distance(vec)          # e.g. as returned by pos()
     |      --or: distance(mypen)        # where mypen is another turtle
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.pos()
     |      (0.00, 0.00)
     |      >>> turtle.distance(30,40)
     |      50.0
     |      >>> pen = Turtle()
     |      >>> pen.forward(77)
     |      >>> turtle.distance(pen)
     |      77.0
     |
     |  fd = forward(self, distance)
     |      Move the turtle forward by the specified distance.
     |
     |      Aliases: forward | fd
     |
     |      Argument:
     |      distance -- a number (integer or float)
     |
     |      Move the turtle forward by the specified distance, in the direction
     |      the turtle is headed.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 0.00)
     |      >>> turtle.forward(25)
     |      >>> turtle.position()
     |      (25.00,0.00)
     |      >>> turtle.forward(-75)
     |      >>> turtle.position()
     |      (-50.00,0.00)
     |
     |  forward(self, distance)
     |      Move the turtle forward by the specified distance.
```

```
|
|      Aliases: forward | fd
|
|      Argument:
|      distance -- a number (integer or float)
|
|      Move the turtle forward by the specified distance, in the direction
|      the turtle is headed.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.forward(25)
|      >>> turtle.position()
|      (25.00,0.00)
|      >>> turtle.forward(-75)
|      >>> turtle.position()
|      (-50.00,0.00)
|
|  goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
```

```
 |
 |  heading(self)
 |      Return the turtle's current heading.
 |
 |      No arguments.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.left(67)
 |      >>> turtle.heading()
 |      67.0
 |
 |  home(self)
 |      Move turtle to the origin - coordinates (0,0).
 |
 |      No arguments.
 |
 |      Move turtle to the origin - coordinates (0,0) and set its
 |      heading to its start-orientation (which depends on mode).
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.home()
 |
 |  left(self, angle)
 |      Turn turtle left by angle units.
 |
 |      Aliases: left | lt
 |
 |      Argument:
 |      angle -- a number (integer or float)
 |
 |      Turn turtle left by angle units. (Units are by default degrees,
 |      but can be set via the degrees() and radians() functions.)
 |      Angle orientation depends on mode. (See this.)
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.heading()
 |      22.0
 |      >>> turtle.left(45)
 |      >>> turtle.heading()
 |      67.0
 |
 |  lt = left(self, angle)
 |      Turn turtle left by angle units.
 |
 |      Aliases: left | lt
 |
 |      Argument:
 |      angle -- a number (integer or float)
```

```
|
|       Turn turtle left by angle units. (Units are by default degrees,
|       but can be set via the degrees() and radians() functions.)
|       Angle orientation depends on mode. (See this.)
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.heading()
|       22.0
|       >>> turtle.left(45)
|       >>> turtle.heading()
|       67.0
|
|   pos(self)
|       Return the turtle's current location (x,y), as a Vec2D-vector.
|
|       Aliases: pos | position
|
|       No arguments.
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.pos()
|       (0.00, 240.00)
|
|   position = pos(self)
|       Return the turtle's current location (x,y), as a Vec2D-vector.
|
|       Aliases: pos | position
|
|       No arguments.
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.pos()
|       (0.00, 240.00)
|
|   radians(self)
|       Set the angle measurement units to radians.
|
|       No arguments.
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.heading()
|       90
|       >>> turtle.radians()
|       >>> turtle.heading()
|       1.5707963267948966
|
|   right(self, angle)
|       Turn turtle right by angle units.
```

```
|
|       Aliases: right | rt
|
|       Argument:
|       angle -- a number (integer or float)
|
|       Turn turtle right by angle units. (Units are by default degrees,
|       but can be set via the degrees() and radians() functions.)
|       Angle orientation depends on mode. (See this.)
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.heading()
|       22.0
|       >>> turtle.right(45)
|       >>> turtle.heading()
|       337.0
|
|   rt = right(self, angle)
|       Turn turtle right by angle units.
|
|       Aliases: right | rt
|
|       Argument:
|       angle -- a number (integer or float)
|
|       Turn turtle right by angle units. (Units are by default degrees,
|       but can be set via the degrees() and radians() functions.)
|       Angle orientation depends on mode. (See this.)
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.heading()
|       22.0
|       >>> turtle.right(45)
|       >>> turtle.heading()
|       337.0
|
|   seth = setheading(self, to_angle)
|       Set the orientation of the turtle to to_angle.
|
|       Aliases:  setheading | seth
|
|       Argument:
|       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
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.setheading(90)
|      >>> turtle.heading()
|      90
|
|  setheading(self, to_angle)
|      Set the orientation of the turtle to to_angle.
|
|      Aliases:  setheading | seth
|
|      Argument:
|      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
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.setheading(90)
|      >>> turtle.heading()
|      90
|
|  setpos = goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
|
|  setposition = goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
|
|  setx(self, x)
|      Set the turtle's first coordinate to x
```

```
     |
     |      Argument:
     |      x -- a number (integer or float)
     |
     |      Set the turtle's first coordinate to x, leave second coordinate
     |      unchanged.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 240.00)
     |      >>> turtle.setx(10)
     |      >>> turtle.position()
     |      (10.00, 240.00)
     |
     |  sety(self, y)
     |      Set the turtle's second coordinate to y
     |
     |      Argument:
     |      y -- a number (integer or float)
     |
     |      Set the turtle's first coordinate to x, second coordinate remains
     |      unchanged.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 40.00)
     |      >>> turtle.sety(-10)
     |      >>> turtle.position()
     |      (0.00, -10.00)
     |
     |  towards(self, x, y=None)
     |      Return the angle of the line from the turtle's position to (x, y).
     |
     |      Arguments:
     |      x -- a number   or  a pair/vector of numbers   or   a turtle
instance
     |      y -- a number       None                            None
     |
     |      call: distance(x, y)         # two coordinates
     |      --or: distance((x, y))       # a pair (tuple) of coordinates
     |      --or: distance(vec)          # e.g. as returned by pos()
     |      --or: distance(mypen)        # where mypen is another turtle
     |
     |      Return the angle, between the line from turtle-position to position
     |      specified by x, y and the turtle's start orientation. (Depends on
     |      modes - "standard" or "logo")
     |
     |      Example (for a Turtle instance named turtle):
```

```
     |      >>> turtle.pos()
     |      (10.00, 10.00)
     |      >>> turtle.towards(0,0)
     |      225.0
     |
     |  xcor(self)
     |      Return the turtle's x coordinate.
     |
     |      No arguments.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> reset()
     |      >>> turtle.left(60)
     |      >>> turtle.forward(100)
     |      >>> print turtle.xcor()
     |      50.0
     |
     |  ycor(self)
     |      Return the turtle's y coordinate
     |      ---
     |      No arguments.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> reset()
     |      >>> turtle.left(60)
     |      >>> turtle.forward(100)
     |      >>> print turtle.ycor()
     |      86.6025403784
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from TNavigator:
     |
     |  DEFAULT_ANGLEOFFSET = 0
     |
     |  DEFAULT_ANGLEORIENT = 1
     |
     |  DEFAULT_MODE = 'standard'
     |
     |  START_ORIENTATION = {'logo': (0.00,1.00), 'standard': (1.00,0.00),
'wo…

    class ScrolledCanvas(tkinter.Frame)
     |  Modeled after the scrolled canvas class from Grayons's Tkinter book.
     |
     |  Used as the default canvas, which pops up automatically when
     |  using turtle graphics functions or the Turtle class.
     |
     |  Method resolution order:
```

```
     |      ScrolledCanvas
     |      tkinter.Frame
     |      tkinter.Widget
     |      tkinter.BaseWidget
     |      tkinter.Misc
     |      tkinter.Pack
     |      tkinter.Place
     |      tkinter.Grid
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __init__(self, master, width=500, height=350, canvwidth=600,
canvheight=500)
     |      Construct a frame widget with the parent MASTER.
     |
     |      Valid resource names: background, bd, bg, borderwidth, class,
     |      colormap, container, cursor, height, highlightbackground,
     |      highlightcolor, highlightthickness, relief, takefocus, visual,
width.
     |
     |  addtag(self, *args, **kw)
     |
     |  addtag_above(self, *args, **kw)
     |
     |  addtag_all(self, *args, **kw)
     |
     |  addtag_below(self, *args, **kw)
     |
     |  addtag_closest(self, *args, **kw)
     |
     |  addtag_enclosed(self, *args, **kw)
     |
     |  addtag_overlapping(self, *args, **kw)
     |
     |  addtag_withtag(self, *args, **kw)
     |
     |  adjustScrolls(self)
     |      Adjust scrollbars according to window- and canvas-size.
     |
     |  bbox(self, *args)
     |      'forward' method, which canvas itself has inherited…
     |
     |  bind(self, *args, **kwargs)
     |      'forward' method, which canvas itself has inherited…
     |
     |  canvasx(self, *args, **kw)
     |
     |  canvasy(self, *args, **kw)
     |
     |  cget(self, *args, **kwargs)
     |      'forward' method, which canvas itself has inherited…
     |
     |  config(self, *args, **kwargs)
     |      'forward' method, which canvas itself has inherited…
     |
     |  coords(self, *args, **kw)
     |
     |  create_arc(self, *args, **kw)
     |
     |  create_bitmap(self, *args, **kw)
     |
     |  create_image(self, *args, **kw)
     |
     |  create_line(self, *args, **kw)
     |
     |  create_oval(self, *args, **kw)
     |
     |  create_polygon(self, *args, **kw)
     |
     |  create_rectangle(self, *args, **kw)
     |
     |  create_text(self, *args, **kw)
     |
     |  create_window(self, *args, **kw)
     |
     |  dchars(self, *args, **kw)
     |
     |  delete(self, *args, **kw)
     |
     |  dtag(self, *args, **kw)
     |
     |  find(self, *args, **kw)
     |
     |  find_above(self, *args, **kw)
     |
     |  find_all(self, *args, **kw)
     |
     |  find_below(self, *args, **kw)
     |
     |  find_closest(self, *args, **kw)
     |
     |  find_enclosed(self, *args, **kw)
     |
     |  find_overlapping(self, *args, **kw)
     |
     |  find_withtag(self, *args, **kw)
     |
     |  focus_force(self)
     |      'forward' method, which canvas itself has inherited…
     |
     |  gettags(self, *args, **kw)
     |
     |  icursor(self, *args, **kw)
     |
     |  index(self, *args, **kw)
     |
     |  insert(self, *args, **kw)
     |
     |  itemcget(self, *args, **kw)
     |
     |  itemconfig(self, *args, **kw)
     |
     |  itemconfigure(self, *args, **kw)
     |
     |  move(self, *args, **kw)
     |
     |  onResize(self, event)
     |      self-explanatory
     |
     |  postscript(self, *args, **kw)
     |
     |  reset(self, canvwidth=None, canvheight=None, bg=None)
     |      Adjust canvas and scrollbars according to given canvas size.
     |
     |  scale(self, *args, **kw)
     |
     |  scan_dragto(self, *args, **kw)
     |
     |  scan_mark(self, *args, **kw)
     |
     |  select_adjust(self, *args, **kw)
     |
     |  select_clear(self, *args, **kw)
     |
     |  select_from(self, *args, **kw)
     |
     |  select_item(self, *args, **kw)
     |
     |  select_to(self, *args, **kw)
     |
     |  tag_bind(self, *args, **kw)
     |
     |  tag_lower(self, *args, **kw)
```

```
|
|  tag_raise(self, *args, **kw)
|
|  tag_unbind(self, *args, **kw)
|
|  type(self, *args, **kw)
|
|  unbind(self, *args, **kwargs)
|      'forward' method, which canvas itself has inherited…
|
|  xview(self, *args, **kw)
|
|  xview_moveto(self, *args, **kw)
|
|  xview_scroll(self, *args, **kw)
|
|  yview(self, *args, **kw)
|
|  yview_moveto(self, *args, **kw)
|
|  yview_scroll(self, *args, **kw)
|
|  ----------------------------------------------------------------------
|  Methods inherited from tkinter.BaseWidget:
|
|  destroy(self)
|      Destroy this and all descendants widgets.
|
|  ----------------------------------------------------------------------
|  Methods inherited from tkinter.Misc:
|
|  __getitem__ = cget(self, key)
|      Return the resource value for a KEY given as string.
|
|  __repr__(self)
|      Return repr(self).
|
|  __setitem__(self, key, value)
|
|  __str__(self)
|      Return the window path name of this widget.
|
|  after(self, ms, func=None, *args)
|      Call function once after given time.
|
|      MS specifies the time in milliseconds. FUNC gives the
|      function which shall be called. Additional parameters
|      are given as parameters to the function call.  Return
```

```
     |      identifier to cancel scheduling with after_cancel.
     |
     |  after_cancel(self, id)
     |      Cancel scheduling of function identified with ID.
     |
     |      Identifier returned by after or after_idle must be
     |      given as first parameter.
     |
     |  after_idle(self, func, *args)
     |      Call FUNC once if the Tcl main loop has no event to
     |      process.
     |
     |      Return an identifier to cancel the scheduling with
     |      after_cancel.
     |
     |  anchor = grid_anchor(self, anchor=None)
     |      The anchor value controls how to place the grid within the
     |      master when no row/column has any weight.
     |
     |      The default anchor is nw.
     |
     |  bell(self, displayof=0)
     |      Ring a display's bell.
     |
     |  bind_all(self, sequence=None, func=None, add=None)
     |      Bind to all widgets at an event SEQUENCE a call to function FUNC.
     |      An additional boolean parameter ADD specifies whether FUNC will
     |      be called additionally to the other bound function or whether
     |      it will replace the previous function. See bind for the return
value.
     |
     |  bind_class(self, className, sequence=None, func=None, add=None)
     |      Bind to widgets with bindtag CLASSNAME at event
     |      SEQUENCE a call of function FUNC. An additional
     |      boolean parameter ADD specifies whether FUNC will be
     |      called additionally to the other bound function or
     |      whether it will replace the previous function. See bind for
     |      the return value.
     |
     |  bindtags(self, tagList=None)
     |      Set or get the list of bindtags for this widget.
     |
     |      With no argument return the list of all bindtags associated with
     |      this widget. With a list of strings as argument the bindtags are
     |      set to this list. The bindtags determine in which order events are
     |      processed (see bind).
     |
     |  clipboard_append(self, string, **kw)
     |      Append STRING to the Tk clipboard.
     |
     |      A widget specified at the optional displayof keyword
     |      argument specifies the target display. The clipboard
     |      can be retrieved with selection_get.
     |
     |  clipboard_clear(self, **kw)
     |      Clear the data in the Tk clipboard.
     |
     |      A widget specified for the optional displayof keyword
     |      argument specifies the target display.
     |
     |  clipboard_get(self, **kw)
     |      Retrieve data from the clipboard on window's display.
     |
     |      The window keyword defaults to the root window of the Tkinter
     |      application.
     |
     |      The type keyword specifies the form in which the data is
     |      to be returned and should be an atom name such as STRING
     |      or FILE_NAME.  Type defaults to STRING, except on X11, where the
default
     |      is to try UTF8_STRING and fall back to STRING.
     |
     |      This command is equivalent to:
     |
     |      selection_get(CLIPBOARD)
     |
     |  columnconfigure = grid_columnconfigure(self, index, cnf={}, **kw)
     |      Configure column INDEX of a grid.
     |
     |      Valid resources are minsize (minimum size of the column),
     |      weight (how much does additional space propagate to this column)
     |      and pad (how much space to let additionally).
     |
     |  configure(self, cnf=None, **kw)
     |      Configure resources of a widget.
     |
     |      The values for resources are specified as keyword
     |      arguments. To get an overview about
     |      the allowed keyword arguments call the method keys.
     |
     |  deletecommand(self, name)
     |      Internal function.
     |
     |      Delete the Tcl command provided in NAME.
     |
     |  event_add(self, virtual, *sequences)
     |      Bind a virtual event VIRTUAL (of the form <<Name>>)
     |      to an event SEQUENCE such that the virtual event is triggered
     |      whenever SEQUENCE occurs.
     |
     |  event_delete(self, virtual, *sequences)
     |      Unbind a virtual event VIRTUAL from SEQUENCE.
     |
     |  event_generate(self, sequence, **kw)
     |      Generate an event SEQUENCE. Additional
     |      keyword arguments specify parameter of the event
     |      (e.g. x, y, rootx, rooty).
     |
     |  event_info(self, virtual=None)
     |      Return a list of all virtual events or the information
     |      about the SEQUENCE bound to the virtual event VIRTUAL.
     |
     |  focus = focus_set(self)
     |      Direct input focus to this widget.
     |
     |      If the application currently does not have the focus
     |      this widget will get the focus if the application gets
     |      the focus through the window manager.
     |
     |  focus_displayof(self)
     |      Return the widget which has currently the focus on the
     |      display where this widget is located.
     |
     |      Return None if the application does not have the focus.
     |
     |  focus_get(self)
     |      Return the widget which has currently the focus in the
     |      application.
     |
     |      Use focus_displayof to allow working with several
     |      displays. Return None if application does not have
     |      the focus.
     |
     |  focus_lastfor(self)
     |      Return the widget which would have the focus if top level
     |      for this widget gets the focus from the window manager.
     |
     |  focus_set(self)
     |      Direct input focus to this widget.
     |
     |      If the application currently does not have the focus
     |      this widget will get the focus if the application gets
     |      the focus through the window manager.
     |
     |  getboolean(self, s)
     |      Return a boolean value for Tcl boolean values true and false given
as parameter.
     |
     |  getdouble(self, s)
     |
     |  getint(self, s)
     |
     |  getvar(self, name='PY_VAR')
     |      Return value of Tcl variable NAME.
     |
     |  grab_current(self)
     |      Return widget which has currently the grab in this application
     |      or None.
     |
     |  grab_release(self)
     |      Release grab for this widget if currently set.
     |
     |  grab_set(self)
     |      Set grab for this widget.
     |
     |      A grab directs all events to this and descendant
     |      widgets in the application.
     |
     |  grab_set_global(self)
     |      Set global grab for this widget.
     |
     |      A global grab directs all events to this and
     |      descendant widgets on the display. Use with caution -
     |      other applications do not get events anymore.
     |
     |  grab_status(self)
     |      Return None, "local" or "global" if this widget has
     |      no, a local or a global grab.
     |
     |  grid_anchor(self, anchor=None)
     |      The anchor value controls how to place the grid within the
     |      master when no row/column has any weight.
     |
     |      The default anchor is nw.
     |
     |  grid_bbox(self, column=None, row=None, col2=None, row2=None)
     |      Return a tuple of integer coordinates for the bounding
     |      box of this widget controlled by the geometry manager grid.
     |
     |      If COLUMN, ROW is given the bounding box applies from
     |      the cell with row and column 0 to the specified
     |      cell. If COL2 and ROW2 are given the bounding box
     |       starts at that cell.
     |
     |       The returned integers specify the offset of the upper left
     |       corner in the master widget and the width and height.
     |
     |   grid_columnconfigure(self, index, cnf={}, **kw)
     |       Configure column INDEX of a grid.
     |
     |       Valid resources are minsize (minimum size of the column),
     |       weight (how much does additional space propagate to this column)
     |       and pad (how much space to let additionally).
     |
     |   grid_location(self, x, y)
     |       Return a tuple of column and row which identify the cell
     |       at which the pixel at position X and Y inside the master
     |       widget is located.
     |
     |   grid_propagate(self, flag=['_noarg_'])
     |       Set or get the status for propagation of geometry information.
     |
     |       A boolean argument specifies whether the geometry information
     |       of the slaves will determine the size of this widget. If no argument
     |       is given, the current setting will be returned.
     |
     |   grid_rowconfigure(self, index, cnf={}, **kw)
     |       Configure row INDEX of a grid.
     |
     |       Valid resources are minsize (minimum size of the row),
     |       weight (how much does additional space propagate to this row)
     |       and pad (how much space to let additionally).
     |
     |   grid_size(self)
     |       Return a tuple of the number of column and rows in the grid.
     |
     |   grid_slaves(self, row=None, column=None)
     |       Return a list of all slaves of this widget
     |       in its packing order.
     |
     |   image_names(self)
     |       Return a list of all existing image names.
     |
     |   image_types(self)
     |       Return a list of all available image types (e.g. photo bitmap).
     |
     |   keys(self)
     |       Return a list of all resource names of this widget.
     |
     |   lift = tkraise(self, aboveThis=None)
     |      Raise this widget in the stacking order.
     |
     |  lower(self, belowThis=None)
     |      Lower this widget in the stacking order.
     |
     |  mainloop(self, n=0)
     |      Call the mainloop of Tk.
     |
     |  nametowidget(self, name)
     |      Return the Tkinter instance of a widget identified by
     |      its Tcl name NAME.
     |
     |  option_add(self, pattern, value, priority=None)
     |      Set a VALUE (second parameter) for an option
     |      PATTERN (first parameter).
     |
     |      An optional third parameter gives the numeric priority
     |      (defaults to 80).
     |
     |  option_clear(self)
     |      Clear the option database.
     |
     |      It will be reloaded if option_add is called.
     |
     |  option_get(self, name, className)
     |      Return the value for an option NAME for this widget
     |      with CLASSNAME.
     |
     |      Values with higher priority override lower values.
     |
     |  option_readfile(self, fileName, priority=None)
     |      Read file FILENAME into the option database.
     |
     |      An optional second parameter gives the numeric
     |      priority.
     |
     |  pack_propagate(self, flag=['_noarg_'])
     |      Set or get the status for propagation of geometry information.
     |
     |      A boolean argument specifies whether the geometry information
     |      of the slaves will determine the size of this widget. If no argument
     |      is given the current setting will be returned.
     |
     |  pack_slaves(self)
     |      Return a list of all slaves of this widget
     |      in its packing order.
     |
     |  place_slaves(self)
```

```
 |      Return a list of all slaves of this widget
 |      in its packing order.
 |
 |  propagate = pack_propagate(self, flag=['_noarg_'])
 |      Set or get the status for propagation of geometry information.
 |
 |      A boolean argument specifies whether the geometry information
 |      of the slaves will determine the size of this widget. If no argument
 |      is given the current setting will be returned.
 |
 |  quit(self)
 |      Quit the Tcl interpreter. All widgets will be destroyed.
 |
 |  register = _register(self, func, subst=None, needcleanup=1)
 |      Return a newly created Tcl function. If this
 |      function is called, the Python function FUNC will
 |      be executed. An optional function SUBST can
 |      be given which will be executed before FUNC.
 |
 |  rowconfigure = grid_rowconfigure(self, index, cnf={}, **kw)
 |      Configure row INDEX of a grid.
 |
 |      Valid resources are minsize (minimum size of the row),
 |      weight (how much does additional space propagate to this row)
 |      and pad (how much space to let additionally).
 |
 |  selection_clear(self, **kw)
 |      Clear the current X selection.
 |
 |  selection_get(self, **kw)
 |      Return the contents of the current X selection.
 |
 |      A keyword parameter selection specifies the name of
 |      the selection and defaults to PRIMARY.  A keyword
 |      parameter displayof specifies a widget on the display
 |      to use. A keyword parameter type specifies the form of data to be
 |      fetched, defaulting to STRING except on X11, where UTF8_STRING is
tried
 |      before STRING.
 |
 |  selection_handle(self, command, **kw)
 |      Specify a function COMMAND to call if the X
 |      selection owned by this widget is queried by another
 |      application.
 |
 |      This function must return the contents of the
 |      selection. The function will be called with the
 |      arguments OFFSET and LENGTH which allows the chunking
 |      of very long selections. The following keyword
 |      parameters can be provided:
 |      selection - name of the selection (default PRIMARY),
 |      type - type of the selection (e.g. STRING, FILE_NAME).
 |
 |  selection_own(self, **kw)
 |      Become owner of X selection.
 |
 |      A keyword parameter selection specifies the name of
 |      the selection (default PRIMARY).
 |
 |  selection_own_get(self, **kw)
 |      Return owner of X selection.
 |
 |      The following keyword parameter can
 |      be provided:
 |      selection - name of the selection (default PRIMARY),
 |      type - type of the selection (e.g. STRING, FILE_NAME).
 |
 |  send(self, interp, cmd, *args)
 |      Send Tcl command CMD to different interpreter INTERP to be executed.
 |
 |  setvar(self, name='PY_VAR', value='1')
 |      Set Tcl variable NAME to VALUE.
 |
 |  size = grid_size(self)
 |      Return a tuple of the number of column and rows in the grid.
 |
 |  slaves = pack_slaves(self)
 |      Return a list of all slaves of this widget
 |      in its packing order.
 |
 |  tk_bisque(self)
 |      Change the color scheme to light brown as used in Tk 3.6 and before.
 |
 |  tk_focusFollowsMouse(self)
 |      The widget under mouse will get automatically focus. Can not
 |      be disabled easily.
 |
 |  tk_focusNext(self)
 |      Return the next widget in the focus order which follows
 |      widget which has currently the focus.
 |
 |      The focus order first goes to the next child, then to
 |      the children of the child recursively and then to the
 |      next sibling which is higher in the stacking order.  A
 |      widget is omitted if it has the takefocus resource set
 |      to 0.
```

```
     |
     |  tk_focusPrev(self)
     |      Return previous widget in the focus order. See tk_focusNext for
details.
     |
     |  tk_setPalette(self, *args, **kw)
     |      Set a new color scheme for all widget elements.
     |
     |      A single color as argument will cause that all colors of Tk
     |      widget elements are derived from this.
     |      Alternatively several keyword parameters and its associated
     |      colors can be given. The following keywords are valid:
     |      activeBackground, foreground, selectColor,
     |      activeForeground, highlightBackground, selectBackground,
     |      background, highlightColor, selectForeground,
     |      disabledForeground, insertBackground, troughColor.
     |
     |  tk_strictMotif(self, boolean=None)
     |      Set Tcl internal variable, whether the look and feel
     |      should adhere to Motif.
     |
     |      A parameter of 1 means adhere to Motif (e.g. no color
     |      change if mouse passes over slider).
     |      Returns the set value.
     |
     |  tkraise(self, aboveThis=None)
     |      Raise this widget in the stacking order.
     |
     |  unbind_all(self, sequence)
     |      Unbind for all widgets for event SEQUENCE all functions.
     |
     |  unbind_class(self, className, sequence)
     |      Unbind for all widgets with bindtag CLASSNAME for event SEQUENCE
     |      all functions.
     |
     |  update(self)
     |      Enter event loop until all pending events have been processed by
Tcl.
     |
     |  update_idletasks(self)
     |      Enter event loop until all idle callbacks have been called. This
     |      will update the display of windows but not process events caused by
     |      the user.
     |
     |  wait_variable(self, name='PY_VAR')
     |      Wait until the variable is modified.
     |
     |      A parameter of type IntVar, StringVar, DoubleVar or
     |      BooleanVar must be given.
     |
     |  wait_visibility(self, window=None)
     |      Wait until the visibility of a WIDGET changes
     |      (e.g. it appears).
     |
     |      If no parameter is given self is used.
     |
     |  wait_window(self, window=None)
     |      Wait until a WIDGET is destroyed.
     |
     |      If no parameter is given self is used.
     |
     |  waitvar = wait_variable(self, name='PY_VAR')
     |      Wait until the variable is modified.
     |
     |      A parameter of type IntVar, StringVar, DoubleVar or
     |      BooleanVar must be given.
     |
     |  winfo_atom(self, name, displayof=0)
     |      Return integer which represents atom NAME.
     |
     |  winfo_atomname(self, id, displayof=0)
     |      Return name of atom with identifier ID.
     |
     |  winfo_cells(self)
     |      Return number of cells in the colormap for this widget.
     |
     |  winfo_children(self)
     |      Return a list of all widgets which are children of this widget.
     |
     |  winfo_class(self)
     |      Return window class name of this widget.
     |
     |  winfo_colormapfull(self)
     |      Return true if at the last color request the colormap was full.
     |
     |  winfo_containing(self, rootX, rootY, displayof=0)
     |      Return the widget which is at the root coordinates ROOTX, ROOTY.
     |
     |  winfo_depth(self)
     |      Return the number of bits per pixel.
     |
     |  winfo_exists(self)
     |      Return true if this widget exists.
     |
     |  winfo_fpixels(self, number)
     |      Return the number of pixels for the given distance NUMBER
```

```
     |      (e.g. "3c") as float.
     |
     |  winfo_geometry(self)
     |      Return geometry string for this widget in the form
"widthxheight+X+Y".
     |
     |  winfo_height(self)
     |      Return height of this widget.
     |
     |  winfo_id(self)
     |      Return identifier ID for this widget.
     |
     |  winfo_interps(self, displayof=0)
     |      Return the name of all Tcl interpreters for this display.
     |
     |  winfo_ismapped(self)
     |      Return true if this widget is mapped.
     |
     |  winfo_manager(self)
     |      Return the window manager name for this widget.
     |
     |  winfo_name(self)
     |      Return the name of this widget.
     |
     |  winfo_parent(self)
     |      Return the name of the parent of this widget.
     |
     |  winfo_pathname(self, id, displayof=0)
     |      Return the pathname of the widget given by ID.
     |
     |  winfo_pixels(self, number)
     |      Rounded integer value of winfo_fpixels.
     |
     |  winfo_pointerx(self)
     |      Return the x coordinate of the pointer on the root window.
     |
     |  winfo_pointerxy(self)
     |      Return a tuple of x and y coordinates of the pointer on the root
window.
     |
     |  winfo_pointery(self)
     |      Return the y coordinate of the pointer on the root window.
     |
     |  winfo_reqheight(self)
     |      Return requested height of this widget.
     |
     |  winfo_reqwidth(self)
     |      Return requested width of this widget.
     |
     |  winfo_rgb(self, color)
     |      Return tuple of decimal values for red, green, blue for
     |      COLOR in this widget.
     |
     |  winfo_rootx(self)
     |      Return x coordinate of upper left corner of this widget on the
     |      root window.
     |
     |  winfo_rooty(self)
     |      Return y coordinate of upper left corner of this widget on the
     |      root window.
     |
     |  winfo_screen(self)
     |      Return the screen name of this widget.
     |
     |  winfo_screencells(self)
     |      Return the number of the cells in the colormap of the screen
     |      of this widget.
     |
     |  winfo_screendepth(self)
     |      Return the number of bits per pixel of the root window of the
     |      screen of this widget.
     |
     |  winfo_screenheight(self)
     |      Return the number of pixels of the height of the screen of this
widget
     |      in pixel.
     |
     |  winfo_screenmmheight(self)
     |      Return the number of pixels of the height of the screen of
     |      this widget in mm.
     |
     |  winfo_screenmmwidth(self)
     |      Return the number of pixels of the width of the screen of
     |      this widget in mm.
     |
     |  winfo_screenvisual(self)
     |      Return one of the strings directcolor, grayscale, pseudocolor,
     |      staticcolor, staticgray, or truecolor for the default
     |      colormodel of this screen.
     |
     |  winfo_screenwidth(self)
     |      Return the number of pixels of the width of the screen of
     |      this widget in pixel.
     |
     |  winfo_server(self)
     |      Return information of the X-Server of the screen of this widget in
     |      the form "XmajorRminor vendor vendorVersion".
     |
     |  winfo_toplevel(self)
     |      Return the toplevel widget of this widget.
     |
     |  winfo_viewable(self)
     |      Return true if the widget and all its higher ancestors are mapped.
     |
     |  winfo_visual(self)
     |      Return one of the strings directcolor, grayscale, pseudocolor,
     |      staticcolor, staticgray, or truecolor for the
     |      colormodel of this widget.
     |
     |  winfo_visualid(self)
     |      Return the X identifier for the visual for this widget.
     |
     |  winfo_visualsavailable(self, includeids=False)
     |      Return a list of all visuals available for the screen
     |      of this widget.
     |
     |      Each item in the list consists of a visual name (see winfo_visual),
a
     |      depth and if includeids is true is given also the X identifier.
     |
     |  winfo_vrootheight(self)
     |      Return the height of the virtual root window associated with this
     |      widget in pixels. If there is no virtual root window return the
     |      height of the screen.
     |
     |  winfo_vrootwidth(self)
     |      Return the width of the virtual root window associated with this
     |      widget in pixel. If there is no virtual root window return the
     |      width of the screen.
     |
     |  winfo_vrootx(self)
     |      Return the x offset of the virtual root relative to the root
     |      window of the screen of this widget.
     |
     |  winfo_vrooty(self)
     |      Return the y offset of the virtual root relative to the root
     |      window of the screen of this widget.
     |
     |  winfo_width(self)
     |      Return the width of this widget.
     |
     |  winfo_x(self)
     |      Return the x coordinate of the upper left corner of this widget
     |      in the parent.
```

```
 |
 |  winfo_y(self)
 |      Return the y coordinate of the upper left corner of this widget
 |      in the parent.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from tkinter.Misc:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from tkinter.Pack:
 |
 |  forget = pack_forget(self)
 |      Unmap this widget and do not use it for the packing order.
 |
 |  info = pack_info(self)
 |      Return information about the packing options
 |      for this widget.
 |
 |  pack = pack_configure(self, cnf={}, **kw)
 |      Pack a widget in the parent widget. Use as options:
 |      after=widget - pack it after you have packed widget
 |      anchor=NSEW (or subset) - position widget according to
 |                                given direction
 |      before=widget - pack it before you will pack widget
 |      expand=bool - expand widget if parent size grows
 |      fill=NONE or X or Y or BOTH - fill widget if widget grows
 |      in=master - use master to contain this widget
 |      in_=master - see 'in' option description
 |      ipadx=amount - add internal padding in x direction
 |      ipady=amount - add internal padding in y direction
 |      padx=amount - add padding in x direction
 |      pady=amount - add padding in y direction
 |      side=TOP or BOTTOM or LEFT or RIGHT -  where to add this widget.
 |
 |  pack_configure(self, cnf={}, **kw)
 |      Pack a widget in the parent widget. Use as options:
 |      after=widget - pack it after you have packed widget
 |      anchor=NSEW (or subset) - position widget according to
 |                                given direction
 |      before=widget - pack it before you will pack widget
 |      expand=bool - expand widget if parent size grows
 |      fill=NONE or X or Y or BOTH - fill widget if widget grows
```

```
     |      in=master - use master to contain this widget
     |      in_=master - see 'in' option description
     |      ipadx=amount - add internal padding in x direction
     |      ipady=amount - add internal padding in y direction
     |      padx=amount - add padding in x direction
     |      pady=amount - add padding in y direction
     |      side=TOP or BOTTOM or LEFT or RIGHT -  where to add this widget.
     |
     |  pack_forget(self)
     |      Unmap this widget and do not use it for the packing order.
     |
     |  pack_info(self)
     |      Return information about the packing options
     |      for this widget.
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from tkinter.Place:
     |
     |  place = place_configure(self, cnf={}, **kw)
     |      Place a widget in the parent widget. Use as options:
     |      in=master - master relative to which the widget is placed
     |      in_=master - see 'in' option description
     |      x=amount - locate anchor of this widget at position x of master
     |      y=amount - locate anchor of this widget at position y of master
     |      relx=amount - locate anchor of this widget between 0.0 and 1.0
     |                    relative to width of master (1.0 is right edge)
     |      rely=amount - locate anchor of this widget between 0.0 and 1.0
     |                    relative to height of master (1.0 is bottom edge)
     |      anchor=NSEW (or subset) - position anchor according to given
direction
     |      width=amount - width of this widget in pixel
     |      height=amount - height of this widget in pixel
     |      relwidth=amount - width of this widget between 0.0 and 1.0
     |                        relative to width of master (1.0 is the same width
     |                        as the master)
     |      relheight=amount - height of this widget between 0.0 and 1.0
     |                         relative to height of master (1.0 is the same
     |                         height as the master)
     |      bordermode="inside" or "outside" - whether to take border width of
     |                                         master widget into account
     |
     |  place_configure(self, cnf={}, **kw)
     |      Place a widget in the parent widget. Use as options:
     |      in=master - master relative to which the widget is placed
     |      in_=master - see 'in' option description
     |      x=amount - locate anchor of this widget at position x of master
     |      y=amount - locate anchor of this widget at position y of master
     |      relx=amount - locate anchor of this widget between 0.0 and 1.0
     |                      relative to width of master (1.0 is right edge)
     |      rely=amount - locate anchor of this widget between 0.0 and 1.0
     |                      relative to height of master (1.0 is bottom edge)
     |      anchor=NSEW (or subset) - position anchor according to given
direction
     |      width=amount - width of this widget in pixel
     |      height=amount - height of this widget in pixel
     |      relwidth=amount - width of this widget between 0.0 and 1.0
     |                        relative to width of master (1.0 is the same width
     |                        as the master)
     |      relheight=amount - height of this widget between 0.0 and 1.0
     |                         relative to height of master (1.0 is the same
     |                         height as the master)
     |      bordermode="inside" or "outside" - whether to take border width of
     |                                          master widget into account
     |
     |  place_forget(self)
     |      Unmap this widget.
     |
     |  place_info(self)
     |      Return information about the placing options
     |      for this widget.
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from tkinter.Grid:
     |
     |  grid = grid_configure(self, cnf={}, **kw)
     |      Position a widget in the parent widget in a grid. Use as options:
     |      column=number - use cell identified with given column (starting with
0)
     |      columnspan=number - this widget will span several columns
     |      in=master - use master to contain this widget
     |      in_=master - see 'in' option description
     |      ipadx=amount - add internal padding in x direction
     |      ipady=amount - add internal padding in y direction
     |      padx=amount - add padding in x direction
     |      pady=amount - add padding in y direction
     |      row=number - use cell identified with given row (starting with 0)
     |      rowspan=number - this widget will span several rows
     |      sticky=NSEW - if cell is larger on which sides will this
     |                    widget stick to the cell boundary
     |
     |  grid_configure(self, cnf={}, **kw)
     |      Position a widget in the parent widget in a grid. Use as options:
     |      column=number - use cell identified with given column (starting with
0)
     |      columnspan=number - this widget will span several columns
     |      in=master - use master to contain this widget
```

```
 |      in_=master - see 'in' option description
 |      ipadx=amount - add internal padding in x direction
 |      ipady=amount - add internal padding in y direction
 |      padx=amount - add padding in x direction
 |      pady=amount - add padding in y direction
 |      row=number - use cell identified with given row (starting with 0)
 |      rowspan=number - this widget will span several rows
 |      sticky=NSEW - if cell is larger on which sides will this
 |                    widget stick to the cell boundary
 |
 |  grid_forget(self)
 |      Unmap this widget.
 |
 |  grid_info(self)
 |      Return information about the options
 |      for positioning this widget in a grid.
 |
 |  grid_remove(self)
 |      Unmap this widget but remember the grid options.
 |
 |  location = grid_location(self, x, y)
 |      Return a tuple of column and row which identify the cell
 |      at which the pixel at position X and Y inside the master
 |      widget is located.

class Shape(builtins.object)
 |  Data structure modeling shapes.
 |
 |  attribute _type is one of "polygon", "image", "compound"
 |  attribute _data is - depending on _type a poygon-tuple,
 |  an image or a list constructed using the addcomponent method.
 |
 |  Methods defined here:
 |
 |  __init__(self, type_, data=None)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  addcomponent(self, poly, fill, outline=None)
 |      Add component to a shape of type compound.
 |
 |      Arguments: poly is a polygon, i. e. a tuple of number pairs.
 |      fill is the fillcolor of the component,
 |      outline is the outline color of the component.
 |
 |      call (for a Shapeobject namend s):
 |      --   s.addcomponent(((0,0), (10,10), (-10,10)), "red", "blue")
 |
 |      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()
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)

class Terminator(builtins.Exception)
 |  Will be raised in TurtleScreen.update, if _RUNNING becomes False.
 |
 |  This stops execution of a turtle graphics script.
 |  Main purpose: use in the Demo-Viewer turtle.Demo.py.
 |
 |  Method resolution order:
 |      Terminator
 |      builtins.Exception
 |      builtins.BaseException
 |      builtins.object
 |
 |  Data descriptors defined here:
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.Exception:
 |
 |  __init__(self, /, *args, **kwargs)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  __new__(*args, **kwargs) from builtins.type
 |      Create and return a new object.  See help(type) for accurate
signature.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.BaseException:
 |
 |  __delattr__(self, name, /)
 |      Implement delattr(self, name).
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __reduce__(…)
 |      helper for pickle
 |
 |  __repr__(self, /)
 |      Return repr(self).
 |
 |  __setattr__(self, name, value, /)
 |      Implement setattr(self, name, value).
 |
 |  __setstate__(…)
 |
 |  __str__(self, /)
 |      Return str(self).
 |
 |  with_traceback(…)
 |      Exception.with_traceback(tb) --
 |      set self.__traceback__ to tb and return self.
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from builtins.BaseException:
 |
 |  __cause__
 |      exception cause
 |
 |  __context__
 |      exception context
 |
 |  __dict__
 |
 |  __suppress_context__
 |
 |  __traceback__
 |
 |  args

class Turtle(RawTurtle)
 |  RawTurtle auto-creating (scrolled) canvas.
 |
 |  When a Turtle object is created or a function derived from some
 |  Turtle method is called a TurtleScreen object is automatically created.
 |
 |  Method resolution order:
 |      Turtle
 |      RawTurtle
 |      TPen
 |      TNavigator
 |      builtins.object
 |
 |  Methods defined here:
 |
 |  __init__(self, shape='classic', undobuffersize=1000, visible=True)
 |      Initialize self.  See help(type(self)) for accurate signature.
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from RawTurtle:
 |
 |  begin_fill(self)
 |      Called just before drawing a shape to be filled.
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.color("black", "red")
 |      >>> turtle.begin_fill()
 |      >>> turtle.circle(60)
 |      >>> turtle.end_fill()
 |
 |  begin_poly(self)
 |      Start recording the vertices of a polygon.
 |
 |      No argument.
 |
 |      Start recording the vertices of a polygon. Current turtle position
 |      is first point of polygon.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.begin_poly()
 |
 |  clear(self)
 |      Delete the turtle's drawings from the screen. Do not move turtle.
 |
 |      No arguments.
 |
 |      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.
 |
 |      Examples (for a Turtle instance named turtle):
 |      >>> turtle.clear()
 |
 |  clearstamp(self, stampid)
 |      Delete stamp with given stampid
 |
 |      Argument:
     |      stampid - an integer, must be return value of previous stamp() call.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color("blue")
     |      >>> astamp = turtle.stamp()
     |      >>> turtle.fd(50)
     |      >>> turtle.clearstamp(astamp)
     |
     |  clearstamps(self, n=None)
     |      Delete all or first/last n of turtle's stamps.
     |
     |      Optional argument:
     |      n -- an integer
     |
     |      If n is None, delete all of pen's stamps,
     |      else if n > 0 delete first n stamps
     |      else if n < 0 delete last n stamps.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> for i in range(8):
     |      …     turtle.stamp(); turtle.fd(30)
     |      …
     |      >>> turtle.clearstamps(2)
     |      >>> turtle.clearstamps(-2)
     |      >>> turtle.clearstamps()
     |
     |  clone(self)
     |      Create and return a clone of the turtle.
     |
     |      No argument.
     |
     |      Create and return a clone of the turtle with same position, heading
     |      and turtle properties.
     |
     |      Example (for a Turtle instance named mick):
     |      mick = Turtle()
     |      joe = mick.clone()
     |
     |  dot(self, size=None, *color)
     |      Draw a dot with diameter size, using color.
     |
     |      Optional arguments:
     |      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.
```

```
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.dot()
|       >>> turtle.fd(50); turtle.dot(20, "blue"); turtle.fd(50)
|
|   end_fill(self)
|       Fill the shape drawn after the call begin_fill().
|
|       No argument.
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.color("black", "red")
|       >>> turtle.begin_fill()
|       >>> turtle.circle(60)
|       >>> turtle.end_fill()
|
|   end_poly(self)
|       Stop recording the vertices of a polygon.
|
|       No argument.
|
|       Stop recording the vertices of a polygon. Current turtle position is
|       last point of polygon. This will be connected with the first point.
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.end_poly()
|
|   filling(self)
|       Return fillstate (True if filling, False else).
|
|       No argument.
|
|       Example (for a Turtle instance named turtle):
|       >>> turtle.begin_fill()
|       >>> if turtle.filling():
|       …     turtle.pensize(5)
|       … else:
|       …     turtle.pensize(3)
|
|   get_poly(self)
|       Return the lastly recorded polygon.
|
|       No argument.
|
|       Example (for a Turtle instance named turtle):
|       >>> p = turtle.get_poly()
|       >>> turtle.register_shape("myFavouriteShape", p)
|
|  get_shapepoly(self)
|      Return the current shape polygon as tuple of coordinate pairs.
|
|      No argument.
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.shape("square")
|      >>> turtle.shapetransform(4, -1, 0, 2)
|      >>> turtle.get_shapepoly()
|      ((50, -20), (30, 20), (-50, 20), (-30, -20))
|
|  getpen = getturtle(self)
|      Return the Turtleobject itself.
|
|      No argument.
|
|      Only reasonable use: as a function to return the 'anonymous turtle':
|
|      Example:
|      >>> pet = getturtle()
|      >>> pet.fd(50)
|      >>> pet
|      <turtle.Turtle object at 0x0187D810>
|      >>> turtles()
|      [<turtle.Turtle object at 0x0187D810>]
|
|  getscreen(self)
|      Return the TurtleScreen object, the turtle is drawing  on.
|
|      No argument.
|
|      Return the TurtleScreen object, the turtle is drawing  on.
|      So TurtleScreen-methods can be called for that object.
|
|      Example (for a Turtle instance named turtle):
|      >>> ts = turtle.getscreen()
|      >>> ts
|      <turtle.TurtleScreen object at 0x0106B770>
|      >>> ts.bgcolor("pink")
|
|  getturtle(self)
|      Return the Turtleobject itself.
|
|      No argument.
|
|      Only reasonable use: as a function to return the 'anonymous turtle':
|
|      Example:
     |      >>> pet = getturtle()
     |      >>> pet.fd(50)
     |      >>> pet
     |      <turtle.Turtle object at 0x0187D810>
     |      >>> turtles()
     |      [<turtle.Turtle object at 0x0187D810>]
     |
     |  onclick(self, fun, btn=1, add=None)
     |      Bind fun to mouse-click event on this turtle on canvas.
     |
     |      Arguments:
     |      fun --  a function with two arguments, to which will be assigned
     |              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, new binding will be added, otherwise
     |              it will replace a former binding.
     |
     |      Example for the anonymous turtle, i. e. the procedural way:
     |
     |      >>> def turn(x, y):
     |      …     left(360)
     |      …
     |      >>> onclick(turn)  # Now clicking into the turtle will turn it.
     |      >>> onclick(None)  # event-binding will be removed
     |
     |  ondrag(self, fun, btn=1, add=None)
     |      Bind fun to mouse-move event on this turtle on canvas.
     |
     |      Arguments:
     |      fun -- a function with two arguments, to which will be assigned
     |             the coordinates of the clicked point on the canvas.
     |      num -- number of the mouse-button defaults to 1 (left mouse button).
     |
     |      Every sequence of mouse-move-events on a turtle is preceded by a
     |      mouse-click event on that turtle.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.ondrag(turtle.goto)
     |
     |      Subsequently clicking and dragging a Turtle will move it
     |      across the screen thereby producing handdrawings (if pen is
     |      down).
     |
     |  onrelease(self, fun, btn=1, add=None)
     |      Bind fun to mouse-button-release event on this turtle on canvas.
     |
     |      Arguments:
```

```
     |      fun -- a function with two arguments, to which will be assigned
     |              the coordinates of the clicked point on the canvas.
     |      num --  number of the mouse-button defaults to 1 (left mouse
button).
     |
     |      Example (for a MyTurtle instance named joe):
     |      >>> class MyTurtle(Turtle):
     |      …     def glow(self,x,y):
     |      …             self.fillcolor("red")
     |      …     def unglow(self,x,y):
     |      …             self.fillcolor("")
     |      …
     |      >>> joe = MyTurtle()
     |      >>> joe.onclick(joe.glow)
     |      >>> joe.onrelease(joe.unglow)
     |
     |      Clicking on joe turns fillcolor red, unclicking turns it to
     |      transparent.
     |
     |  reset(self)
     |      Delete the turtle's drawings and restore its default values.
     |
     |      No argument.
     |
     |      Delete the turtle's drawings from the screen, re-center the turtle
     |      and set variables to the default values.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00,-22.00)
     |      >>> turtle.heading()
     |      100.0
     |      >>> turtle.reset()
     |      >>> turtle.position()
     |      (0.00,0.00)
     |      >>> turtle.heading()
     |      0.0
     |
     |  settiltangle(self, angle)
     |      Rotate the turtleshape to point in the specified direction
     |
     |      Argument: angle -- 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).
     |
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.settiltangle(45)
     |      >>> stamp()
     |      >>> turtle.fd(50)
     |      >>> turtle.settiltangle(-45)
     |      >>> stamp()
     |      >>> turtle.fd(50)
     |
     |  setundobuffer(self, size)
     |      Set or disable undobuffer.
     |
     |      Argument:
     |      size -- an integer or None
     |
     |      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() function.
     |      If size is None, no undobuffer is present.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.setundobuffer(42)
     |
     |  shape(self, name=None)
     |      Set turtle shape to shape with given name / return current
shapename.
     |
     |      Optional argument:
     |      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.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.shape()
     |      'arrow'
     |      >>> turtle.shape("turtle")
     |      >>> turtle.shape()
     |      'turtle'
     |
     |  shapesize(self, stretch_wid=None, stretch_len=None, outline=None)
     |      Set/return turtle's stretchfactors/outline. Set resizemode to
"user".
     |
     |      Optional arguments:
     |         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 orientation
     |      stretch_len is stretchfactor in direction of turtles orientation.
     |      outline determines the width of the shapes's outline.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.resizemode("user")
     |      >>> turtle.shapesize(5, 5, 12)
     |      >>> turtle.shapesize(outline=8)
     |
     |  shapetransform(self, t11=None, t12=None, t21=None, t22=None)
     |      Set or return the current transformation matrix of the turtle shape.
     |
     |      Optional arguments: t11, t12, t21, t22 -- numbers.
     |
     |      If none of the matrix elements are given, return the transformation
     |      matrix.
     |      Otherwise set the given elements and transform the turtleshape
     |      according to the matrix consisting of first row t11, t12 and
     |      second row t21, 22.
     |      Modify stretchfactor, shearfactor and tiltangle according to the
     |      given matrix.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("square")
     |      >>> turtle.shapesize(4,2)
     |      >>> turtle.shearfactor(-0.5)
     |      >>> turtle.shapetransform()
     |      (4.0, -1.0, -0.0, 2.0)
     |
     |  shearfactor(self, shear=None)
     |      Set or return the current shearfactor.
     |
     |      Optional argument: shear -- number, tangent of the shear angle
     |
     |      Shear the turtleshape according to the given shearfactor shear,
     |      which is the tangent of the shear angle. DO NOT change the
     |      turtle's heading (direction of movement).
     |      If shear is not given: return the current shearfactor, i. e. the
     |      tangent of the shear angle, by which lines parallel to the
     |      heading of the turtle are sheared.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.shearfactor(0.5)
     |      >>> turtle.shearfactor()
     |      >>> 0.5
     |
     |  stamp(self)
     |      Stamp a copy of the turtleshape onto the canvas and return its id.
     |
     |      No argument.
     |
     |      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).
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.color("blue")
     |      >>> turtle.stamp()
     |      13
     |      >>> turtle.fd(50)
     |
     |  tilt(self, angle)
     |      Rotate the turtleshape by angle.
     |
     |      Argument:
     |      angle - a number
     |
     |      Rotate the turtleshape by angle from its current tilt-angle,
     |      but do NOT change the turtle's heading (direction of movement).
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.tilt(30)
     |      >>> turtle.fd(50)
     |      >>> turtle.tilt(30)
     |      >>> turtle.fd(50)
     |
     |  tiltangle(self, angle=None)
     |      Set or return the current tilt-angle.
```

```
     |
     |      Optional argument: angle -- 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).
     |      If angle is not given: 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).
     |
     |      Deprecated since Python 3.1
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.shape("circle")
     |      >>> turtle.shapesize(5,2)
     |      >>> turtle.tilt(45)
     |      >>> turtle.tiltangle()
     |
     |  turtlesize = shapesize(self, stretch_wid=None, stretch_len=None,
outline=None)
     |      Set/return turtle's stretchfactors/outline. Set resizemode to
"user".
     |
     |      Optional arguments:
     |         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 orientation
     |      stretch_len is stretchfactor in direction of turtles orientation.
     |      outline determines the width of the shapes's outline.
     |
     |      Examples (for a Turtle instance named turtle):
     |      >>> turtle.resizemode("user")
     |      >>> turtle.shapesize(5, 5, 12)
     |      >>> turtle.shapesize(outline=8)
     |
     |  undo(self)
     |      undo (repeatedly) the last turtle action.
     |
     |      No argument.
```

```
 |
 |      undo (repeatedly) the last turtle action.
 |      Number of available undo actions is determined by the size of
 |      the undobuffer.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> for i in range(4):
 |      …     turtle.fd(50); turtle.lt(80)
 |      …
 |      >>> for i in range(8):
 |      …     turtle.undo()
 |      …
 |
 |  undobufferentries(self)
 |      Return count of entries in the undobuffer.
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> while undobufferentries():
 |      …     undo()
 |
 |  write(self, arg, move=False, align='left', font=('Arial', 8, 'normal'))
 |      Write text at the current turtle position.
 |
 |      Arguments:
 |      arg -- info, which is to be written to the TurtleScreen
 |      move (optional) -- True/False
 |      align (optional) -- one of the strings "left", "center" or right"
 |      font (optional) -- 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.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.write('Home = ', True, align="center")
 |      >>> turtle.write((0,0), True)
 |
 |  ----------------------------------------------------------------------
 |  Data and other attributes inherited from RawTurtle:
 |
 |  screens = []
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from TPen:
```

```
 |
 |  color(self, *args)
 |      Return or set the pencolor and fillcolor.
 |
 |      Arguments:
 |      Several input formats are allowed.
 |      They use 0, 1, 2, or 3 arguments as follows:
 |
 |      color()
 |          Return the current pencolor and the current fillcolor
 |          as a pair of color specification strings as are returned
 |          by pencolor and fillcolor.
 |      color(colorstring), color((r,g,b)), color(r,g,b)
 |          inputs as in 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.
 |      For mor info see: pencolor, fillcolor
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.color('red', 'green')
 |      >>> turtle.color()
 |      ('red', 'green')
 |      >>> colormode(255)
 |      >>> color((40, 80, 120), (160, 200, 240))
 |      >>> color()
 |      ('#285078', '#a0c8f0')
 |
 |  down = pendown(self)
 |      Pull the pen down -- drawing when moving.
 |
 |      Aliases: pendown | pd | down
 |
 |      No argument.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.pendown()
 |
 |  fillcolor(self, *args)
 |      Return or set the fillcolor.
 |
 |      Arguments:
 |      Four input formats are allowed:
```

```
|        - fillcolor()
|          Return the current fillcolor as color specification string,
|          possibly in hex-number format (see example).
|          May be used as input to another color/pencolor/fillcolor call.
|        - fillcolor(colorstring)
|          s is a Tk color specification string, such as "red" or "yellow"
|        - fillcolor((r, g, b))
|          *a tuple* of r, g, and b, which represent, an RGB color,
|          and each of r, g, and b are in the range 0..colormode,
|          where colormode is either 1.0 or 255
|        - fillcolor(r, g, b)
|          r, g, and b represent an RGB color, and each of r, g, and b
|          are in the range 0..colormode
|
|      If turtleshape is a polygon, the interior of that polygon is drawn
|      with the newly set fillcolor.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.fillcolor('violet')
|      >>> col = turtle.pencolor()
|      >>> turtle.fillcolor(col)
|      >>> turtle.fillcolor(0, .5, 0)
|
|  hideturtle(self)
|      Makes the turtle invisible.
|
|      Aliases: hideturtle | ht
|
|      No argument.
|
|      It's a good idea to do this while you're in the
|      middle of a complicated drawing, because hiding
|      the turtle speeds up the drawing observably.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|
|  ht = hideturtle(self)
|      Makes the turtle invisible.
|
|      Aliases: hideturtle | ht
|
|      No argument.
|
|      It's a good idea to do this while you're in the
|      middle of a complicated drawing, because hiding
|      the turtle speeds up the drawing observably.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|
|  isdown(self)
|      Return True if pen is down, False if it's up.
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|      >>> turtle.isdown()
|      False
|      >>> turtle.pendown()
|      >>> turtle.isdown()
|      True
|
|  isvisible(self)
|      Return True if the Turtle is shown, False if it's hidden.
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|      >>> print turtle.isvisible():
|      False
|
|  pd = pendown(self)
|      Pull the pen down -- drawing when moving.
|
|      Aliases: pendown | pd | down
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pendown()
|
|  pen(self, pen=None, **pendict)
|      Return or set the pen's attributes.
|
|      Arguments:
|          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)
|         "shearfactor":   number
|         "outline"    :   positive number
|         "tilt"       :   number
|
|      This dictionary can be used as argument for a subsequent
|      pen()-call 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.
|
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.pen(fillcolor="black", pencolor="red", pensize=10)
|      >>> turtle.pen()
|      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
|      'pencolor': 'red', 'pendown': True, 'fillcolor': 'black',
|      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
|      >>> penstate=turtle.pen()
|      >>> turtle.color("yellow","")
|      >>> turtle.penup()
|      >>> turtle.pen()
|      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
|      'pencolor': 'yellow', 'pendown': False, 'fillcolor': '',
|      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
|      >>> p.pen(penstate, fillcolor="green")
|      >>> p.pen()
|      {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
|      'pencolor': 'red', 'pendown': True, 'fillcolor': 'green',
|      'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
|
|  pencolor(self, *args)
|      Return or set the pencolor.
|
|      Arguments:
|      Four input formats are allowed:
|        - pencolor()
|          Return the current pencolor as color specification string,
|          possibly in hex-number format (see example).
|          May be used as input to another color/pencolor/fillcolor call.
|        - pencolor(colorstring)
|          s is a Tk color specification string, such as "red" or "yellow"
|        - pencolor((r, g, b))
|          *a tuple* of r, g, and b, which represent, an RGB color,
|          and each of r, g, and b are in the range 0..colormode,
|          where colormode is either 1.0 or 255
|        - pencolor(r, g, b)
|          r, g, and b represent an RGB color, and each of r, g, and b
|          are in the range 0..colormode
|
|      If turtleshape is a polygon, the outline of that polygon is drawn
|      with the newly set pencolor.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pencolor('brown')
|      >>> tup = (0.2, 0.8, 0.55)
|      >>> turtle.pencolor(tup)
|      >>> turtle.pencolor()
|      '#33cc8c'
|
|  pendown(self)
|      Pull the pen down -- drawing when moving.
|
|      Aliases: pendown | pd | down
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pendown()
|
|  pensize(self, width=None)
|      Set or return the line thickness.
|
|      Aliases:  pensize | width
|
|      Argument:
|      width -- 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, current pensize
|      is returned.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.pensize()
|      1
|      >>> turtle.pensize(10)   # from here on lines of width 10 are drawn
|
|  penup(self)
|      Pull the pen up -- no drawing when moving.
|
|      Aliases: penup | pu | up
```

```
|
|      No argument
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|
|  pu = penup(self)
|      Pull the pen up -- no drawing when moving.
|
|      Aliases: penup | pu | up
|
|      No argument
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.penup()
|
|  resizemode(self, rmode=None)
|      Set resizemode to one of the values: "auto", "user", "noresize".
|
|      (Optional) Argument:
|      rmode -- one of the strings "auto", "user", "noresize"
|
|      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()
|        - "noresize" no adaption of the turtle's appearance takes place.
|      If no argument is given, return current resizemode.
|      resizemode("user") is called by a call of shapesize with arguments.
|
|
|      Examples (for a Turtle instance named turtle):
|      >>> turtle.resizemode("noresize")
|      >>> turtle.resizemode()
|      'noresize'
|
|  showturtle(self)
|      Makes the turtle visible.
|
|      Aliases: showturtle | st
|
|      No argument.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.hideturtle()
|      >>> turtle.showturtle()
```

```
     |
     |  speed(self, speed=None)
     |      Return or set the turtle's speed.
     |
     |      Optional argument:
     |      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 in the following way:
     |          '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 : *no* animation takes place. forward/back makes turtle
jump
     |      and likewise left/right make the turtle turn instantly.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.speed(3)
     |
     |  st = showturtle(self)
     |      Makes the turtle visible.
     |
     |      Aliases: showturtle | st
     |
     |      No argument.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.hideturtle()
     |      >>> turtle.showturtle()
     |
     |  up = penup(self)
     |      Pull the pen up -- no drawing when moving.
     |
     |      Aliases: penup | pu | up
     |
     |      No argument
     |
     |      Example (for a Turtle instance named turtle):
```

```
 |      >>> turtle.penup()
 |
 |  width = pensize(self, width=None)
 |      Set or return the line thickness.
 |
 |      Aliases:  pensize | width
 |
 |      Argument:
 |      width -- 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, current pensize
 |      is returned.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.pensize()
 |      1
 |      >>> turtle.pensize(10)   # from here on lines of width 10 are drawn
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors inherited from TPen:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from TNavigator:
 |
 |  back(self, distance)
 |      Move the turtle backward by distance.
 |
 |      Aliases: back | backward | bk
 |
 |      Argument:
 |      distance -- a number
 |
 |      Move the turtle backward by distance ,opposite to the direction the
 |      turtle is headed. Do not change the turtle's heading.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.position()
 |      (0.00, 0.00)
 |      >>> turtle.backward(30)
 |      >>> turtle.position()
```

```
|      (-30.00, 0.00)
|
|  backward = back(self, distance)
|      Move the turtle backward by distance.
|
|      Aliases: back | backward | bk
|
|      Argument:
|      distance -- a number
|
|      Move the turtle backward by distance ,opposite to the direction the
|      turtle is headed. Do not change the turtle's heading.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.backward(30)
|      >>> turtle.position()
|      (-30.00, 0.00)
|
|  bk = back(self, distance)
|      Move the turtle backward by distance.
|
|      Aliases: back | backward | bk
|
|      Argument:
|      distance -- a number
|
|      Move the turtle backward by distance ,opposite to the direction the
|      turtle is headed. Do not change the turtle's heading.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
|      (0.00, 0.00)
|      >>> turtle.backward(30)
|      >>> turtle.position()
|      (-30.00, 0.00)
|
|  circle(self, radius, extent=None, steps=None)
|      Draw a circle with given radius.
|
|      Arguments:
|      radius -- a number
|      extent (optional) -- a number
|      steps (optional) -- an integer
|
|      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. Maybe used to draw regular
     |      polygons.
     |
     |      call: circle(radius)                  # full circle
     |      --or: circle(radius, extent)          # arc
     |      --or: circle(radius, extent, steps)
     |      --or: circle(radius, steps=6)         # 6-sided polygon
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.circle(50)
     |      >>> turtle.circle(120, 180)  # semicircle
     |
     |  degrees(self, fullcircle=360.0)
     |      Set angle measurement units to degrees.
     |
     |      Optional argument:
     |      fullcircle -  a number
     |
     |      Set angle measurement units, i. e. set number
     |      of 'degrees' for a full circle. Dafault value is
     |      360 degrees.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.left(90)
     |      >>> turtle.heading()
     |      90
     |
     |      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
     |
     |  distance(self, x, y=None)
     |      Return the distance from the turtle to (x,y) in turtle step units.
     |
     |      Arguments:
     |      x -- a number   or  a pair/vector of numbers   or   a turtle
instance
     |      y -- a number       None                            None
```

```
 |
 |      call: distance(x, y)         # two coordinates
 |      --or: distance((x, y))       # a pair (tuple) of coordinates
 |      --or: distance(vec)          # e.g. as returned by pos()
 |      --or: distance(mypen)        # where mypen is another turtle
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.pos()
 |      (0.00, 0.00)
 |      >>> turtle.distance(30,40)
 |      50.0
 |      >>> pen = Turtle()
 |      >>> pen.forward(77)
 |      >>> turtle.distance(pen)
 |      77.0
 |
 |  fd = forward(self, distance)
 |      Move the turtle forward by the specified distance.
 |
 |      Aliases: forward | fd
 |
 |      Argument:
 |      distance -- a number (integer or float)
 |
 |      Move the turtle forward by the specified distance, in the direction
 |      the turtle is headed.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.position()
 |      (0.00, 0.00)
 |      >>> turtle.forward(25)
 |      >>> turtle.position()
 |      (25.00,0.00)
 |      >>> turtle.forward(-75)
 |      >>> turtle.position()
 |      (-50.00,0.00)
 |
 |  forward(self, distance)
 |      Move the turtle forward by the specified distance.
 |
 |      Aliases: forward | fd
 |
 |      Argument:
 |      distance -- a number (integer or float)
 |
 |      Move the turtle forward by the specified distance, in the direction
 |      the turtle is headed.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.position()
 |      (0.00, 0.00)
 |      >>> turtle.forward(25)
 |      >>> turtle.position()
 |      (25.00,0.00)
 |      >>> turtle.forward(-75)
 |      >>> turtle.position()
 |      (-50.00,0.00)
 |
 |  goto(self, x, y=None)
 |      Move turtle to an absolute position.
 |
 |      Aliases: setpos | setposition | goto:
 |
 |      Arguments:
 |      x -- a number      or     a pair/vector of numbers
 |      y -- a number             None
 |
 |      call: goto(x, y)         # two coordinates
 |      --or: goto((x, y))       # a pair (tuple) of coordinates
 |      --or: goto(vec)          # e.g. as returned by pos()
 |
 |      Move turtle to an absolute position. If the pen is down,
 |      a line will be drawn. The turtle's orientation does not change.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> 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)
 |
 |  heading(self)
 |      Return the turtle's current heading.
 |
 |      No arguments.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.left(67)
 |      >>> turtle.heading()
 |      67.0
 |
 |  home(self)
 |      Move turtle to the origin - coordinates (0,0).
 |
 |      No arguments.
 |
 |      Move turtle to the origin - coordinates (0,0) and set its
 |      heading to its start-orientation (which depends on mode).
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.home()
 |
 |  left(self, angle)
 |      Turn turtle left by angle units.
 |
 |      Aliases: left | lt
 |
 |      Argument:
 |      angle -- a number (integer or float)
 |
 |      Turn turtle left by angle units. (Units are by default degrees,
 |      but can be set via the degrees() and radians() functions.)
 |      Angle orientation depends on mode. (See this.)
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.heading()
 |      22.0
 |      >>> turtle.left(45)
 |      >>> turtle.heading()
 |      67.0
 |
 |  lt = left(self, angle)
 |      Turn turtle left by angle units.
 |
 |      Aliases: left | lt
 |
 |      Argument:
 |      angle -- a number (integer or float)
 |
 |      Turn turtle left by angle units. (Units are by default degrees,
 |      but can be set via the degrees() and radians() functions.)
 |      Angle orientation depends on mode. (See this.)
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.heading()
 |      22.0
 |      >>> turtle.left(45)
 |      >>> turtle.heading()
 |      67.0
 |
 |  pos(self)
 |      Return the turtle's current location (x,y), as a Vec2D-vector.
 |
 |      Aliases: pos | position
 |
 |      No arguments.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.pos()
 |      (0.00, 240.00)
 |
 |  position = pos(self)
 |      Return the turtle's current location (x,y), as a Vec2D-vector.
 |
 |      Aliases: pos | position
 |
 |      No arguments.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.pos()
 |      (0.00, 240.00)
 |
 |  radians(self)
 |      Set the angle measurement units to radians.
 |
 |      No arguments.
 |
 |      Example (for a Turtle instance named turtle):
 |      >>> turtle.heading()
 |      90
 |      >>> turtle.radians()
 |      >>> turtle.heading()
 |      1.5707963267948966
 |
 |  right(self, angle)
 |      Turn turtle right by angle units.
 |
 |      Aliases: right | rt
 |
 |      Argument:
 |      angle -- a number (integer or float)
 |
 |      Turn turtle right by angle units. (Units are by default degrees,
 |      but can be set via the degrees() and radians() functions.)
 |      Angle orientation depends on mode. (See this.)
```

```
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      22.0
|      >>> turtle.right(45)
|      >>> turtle.heading()
|      337.0
|
|  rt = right(self, angle)
|      Turn turtle right by angle units.
|
|      Aliases: right | rt
|
|      Argument:
|      angle -- a number (integer or float)
|
|      Turn turtle right by angle units. (Units are by default degrees,
|      but can be set via the degrees() and radians() functions.)
|      Angle orientation depends on mode. (See this.)
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.heading()
|      22.0
|      >>> turtle.right(45)
|      >>> turtle.heading()
|      337.0
|
|  seth = setheading(self, to_angle)
|      Set the orientation of the turtle to to_angle.
|
|      Aliases:  setheading | seth
|
|      Argument:
|      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
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.setheading(90)
|      >>> turtle.heading()
|      90
|
|  setheading(self, to_angle)
|      Set the orientation of the turtle to to_angle.
|
|      Aliases:  setheading | seth
|
|      Argument:
|      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
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.setheading(90)
|      >>> turtle.heading()
|      90
|
|  setpos = goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
|
|  setposition = goto(self, x, y=None)
|      Move turtle to an absolute position.
|
|      Aliases: setpos | setposition | goto:
|
|      Arguments:
|      x -- a number      or     a pair/vector of numbers
|      y -- a number             None
|
|      call: goto(x, y)         # two coordinates
|      --or: goto((x, y))       # a pair (tuple) of coordinates
|      --or: goto(vec)          # e.g. as returned by pos()
|
|      Move turtle to an absolute position. If the pen is down,
|      a line will be drawn. The turtle's orientation does not change.
|
|      Example (for a Turtle instance named turtle):
|      >>> 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)
|
|  setx(self, x)
|      Set the turtle's first coordinate to x
|
|      Argument:
|      x -- a number (integer or float)
|
|      Set the turtle's first coordinate to x, leave second coordinate
|      unchanged.
|
|      Example (for a Turtle instance named turtle):
|      >>> turtle.position()
```

```
     |      (0.00, 240.00)
     |      >>> turtle.setx(10)
     |      >>> turtle.position()
     |      (10.00, 240.00)
     |
     |  sety(self, y)
     |      Set the turtle's second coordinate to y
     |
     |      Argument:
     |      y -- a number (integer or float)
     |
     |      Set the turtle's first coordinate to x, second coordinate remains
     |      unchanged.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.position()
     |      (0.00, 40.00)
     |      >>> turtle.sety(-10)
     |      >>> turtle.position()
     |      (0.00, -10.00)
     |
     |  towards(self, x, y=None)
     |      Return the angle of the line from the turtle's position to (x, y).
     |
     |      Arguments:
     |      x -- a number   or  a pair/vector of numbers   or   a turtle
instance
     |      y -- a number       None                            None
     |
     |      call: distance(x, y)         # two coordinates
     |      --or: distance((x, y))       # a pair (tuple) of coordinates
     |      --or: distance(vec)          # e.g. as returned by pos()
     |      --or: distance(mypen)        # where mypen is another turtle
     |
     |      Return the angle, between the line from turtle-position to position
     |      specified by x, y and the turtle's start orientation. (Depends on
     |      modes - "standard" or "logo")
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.pos()
     |      (10.00, 10.00)
     |      >>> turtle.towards(0,0)
     |      225.0
     |
     |  xcor(self)
     |      Return the turtle's x coordinate.
     |
     |      No arguments.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> reset()
     |      >>> turtle.left(60)
     |      >>> turtle.forward(100)
     |      >>> print turtle.xcor()
     |      50.0
     |
     |  ycor(self)
     |      Return the turtle's y coordinate
     |      ---
     |      No arguments.
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> reset()
     |      >>> turtle.left(60)
     |      >>> turtle.forward(100)
     |      >>> print turtle.ycor()
     |      86.6025403784
     |
     |  ----------------------------------------------------------------------
     |  Data and other attributes inherited from TNavigator:
     |
     |  DEFAULT_ANGLEOFFSET = 0
     |
     |  DEFAULT_ANGLEORIENT = 1
     |
     |  DEFAULT_MODE = 'standard'
     |
     |  START_ORIENTATION = {'logo': (0.00,1.00), 'standard': (1.00,0.00),
'wo…

    class TurtleScreen(TurtleScreenBase)
     |  Provides screen oriented methods like setbg etc.
     |
     |  Only relies upon the methods of TurtleScreenBase and NOT
     |  upon components of the underlying graphics toolkit -
     |  which is Tkinter in this case.
     |
     |  Method resolution order:
     |      TurtleScreen
     |      TurtleScreenBase
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __init__(self, cv, mode='standard', colormode=1.0, delay=10)
     |      Initialize self.  See help(type(self)) for accurate signature.
     |
     |  addshape = register_shape(self, name, shape=None)
     |
     |  bgcolor(self, *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):
     |      >>> screen.bgcolor("orange")
     |      >>> screen.bgcolor()
     |      'orange'
     |      >>> screen.bgcolor(0.5,0,0.5)
     |      >>> screen.bgcolor()
     |      '#800080'
     |
     |  bgpic(self, picname=None)
     |      Set background image or return name of current backgroundimage.
     |
     |      Optional argument:
     |      picname -- a string, name of a gif-file or "nopic".
     |
     |      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.
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.bgpic()
     |      'nopic'
     |      >>> screen.bgpic("landscape.gif")
     |      >>> screen.bgpic()
     |      'landscape.gif'
     |
     |  clear(self)
     |      Delete all drawings and all turtles from the TurtleScreen.
     |
     |      No argument.
     |
     |      Reset empty TurtleScreen to its initial state: white background,
     |      no backgroundimage, no eventbindings and tracing on.
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.clear()
     |
     |      Note: this method is not available as function.
     |
     |  clearscreen = clear(self)
     |
     |  colormode(self, cmode=None)
     |      Return the colormode or set it to 1.0 or 255.
     |
     |      Optional argument:
     |      cmode -- one of the values 1.0 or 255
     |
     |      r, g, b values of colortriples have to be in range 0..cmode.
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.colormode()
     |      1.0
     |      >>> screen.colormode(255)
     |      >>> pencolor(240,160,80)
     |
     |  delay(self, delay=None)
     |      Return or set the drawing delay in milliseconds.
     |
     |      Optional argument:
     |      delay -- positive integer
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.delay(15)
     |      >>> screen.delay()
     |      15
     |
     |  getcanvas(self)
     |      Return the Canvas of this TurtleScreen.
     |
     |      No argument.
     |
     |      Example (for a Screen instance named screen):
     |      >>> cv = screen.getcanvas()
     |      >>> cv
     |      <turtle.ScrolledCanvas instance at 0x010742D8>
     |
     |  getshapes(self)
     |      Return a list of names of all currently available turtle shapes.
     |
     |      No argument.
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.getshapes()
     |      ['arrow', 'blank', 'circle', … , 'turtle']
     |
     |  listen(self, xdummy=None, ydummy=None)
     |      Set focus on TurtleScreen (in order to collect key-events)
```

```
|
|      No arguments.
|      Dummy arguments are provided in order
|      to be able to pass listen to the onclick method.
|
|      Example (for a TurtleScreen instance named screen):
|      >>> screen.listen()
|
|  mode(self, mode=None)
|      Set turtle-mode ('standard', 'logo' or 'world') and perform reset.
|
|      Optional argument:
|      mode -- one of the strings 'standard', 'logo' or 'world'
|
|      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.
|
|           Mode      Initial turtle heading     positive angles
|       ------------|-------------------------|-------------------
|        'standard'    to the right (east)       counterclockwise
|          'logo'        upward    (north)         clockwise
|
|      Examples:
|      >>> mode('logo')   # resets turtle heading to north
|      >>> mode()
|      'logo'
|
|  onclick(self, fun, btn=1, add=None)
|      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)
|
|      >>> screen.onclick(goto)
|      >>> # Subsequently clicking into the TurtleScreen will
|      >>> # make the turtle move to the clicked point.
|      >>> screen.onclick(None)
|
|  onkey(self, fun, key)
|      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):
|
|      >>> def f():
|      …     fd(50)
|      …     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
|
|  onkeypress(self, fun, key=None)
|      Bind fun to key-press event of key if key is given,
|      or to any key-press-event if no key is given.
|
|      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():
|      …     fd(50)
|      …     lt(60)
|      …
|      >>> screen.onkeypress(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.
|
|  onkeyrelease = onkey(self, fun, key)
|
|  onscreenclick = onclick(self, fun, btn=1, add=None)
|
|  ontimer(self, fun, t=0)
|      Install a timer, which calls fun after t milliseconds.
|
|      Arguments:
|      fun -- a function with no arguments.
|      t -- a number >= 0
|
|      Example (for a TurtleScreen instance named screen):
|
|      >>> running = True
|      >>> def f():
|      …     if running:
|      …             fd(50)
|      …             lt(60)
|      …             screen.ontimer(f, 250)
|      …
|      >>> f()   # makes the turtle marching around
|      >>> running = False
|
|  register_shape(self, name, shape=None)
|      Adds a turtle shape to TurtleScreen's shapelist.
|
|      Arguments:
|      (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!
|      (2) name is an arbitrary string and shape is a tuple
|          of pairs of coordinates. Installs the corresponding
|          polygon shape
|      (3) name is an arbitrary string and shape is a
|          (compound) Shape object. Installs the corresponding
|          compound shape.
|      To use a shape, you have to issue the command shape(shapename).
|
|      call: register_shape("turtle.gif")
|      --or: register_shape("tri", ((0,0), (10,10), (-10,10)))
|
|      Example (for a TurtleScreen instance named screen):
|      >>> screen.register_shape("triangle", ((5,-3),(0,5),(-5,-3)))
|
|  reset(self)
|      Reset all Turtles on the Screen to their initial state.
|
|      No argument.
|
|      Example (for a TurtleScreen instance named screen):
|      >>> screen.reset()
```

```
     |
     |  resetscreen = reset(self)
     |
     |  screensize(self, canvwidth=None, canvheight=None, bg=None)
     |      Resize the canvas the turtles are drawing on.
     |
     |      Optional arguments:
     |      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 (canvaswidth,
canvasheight)
     |
     |      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!)
     |
     |      Example (for a Turtle instance named turtle):
     |      >>> turtle.screensize(2000,1500)
     |      >>> # e.g. to search for an erroneously escaped turtle ;-)
     |
     |  setworldcoordinates(self, llx, lly, urx, ury)
     |      Set up a user defined coordinate-system.
     |
     |      Arguments:
     |      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 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.
     |
     |      But ATTENTION: in user-defined coordinatesystems angles may appear
     |      distorted. (see Screen.mode())
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.setworldcoordinates(-10,-0.5,50,1.5)
     |      >>> for _ in range(36):
     |      …     left(10)
     |      …     forward(0.5)
     |
     |  tracer(self, n=None, delay=None)
     |      Turns turtle animation on/off and set delay for update drawings.
     |
     |      Optional arguments:
     |      n -- nonnegative  integer
     |      delay -- nonnegative  integer
     |
     |      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())
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.tracer(8, 25)
     |      >>> dist = 2
     |      >>> for i in range(200):
     |      …     fd(dist)
     |      …     rt(90)
     |      …     dist += 2
     |
     |  turtles(self)
     |      Return the list of turtles on the screen.
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.turtles()
     |      [<turtle.Turtle object at 0x00E11FB0>]
     |
     |  update(self)
     |      Perform a TurtleScreen update.
     |
     |  window_height(self)
     |      Return the height of the turtle window.
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.window_height()
     |      480
     |
     |  window_width(self)
     |      Return the width of the turtle window.
     |
     |      Example (for a TurtleScreen instance named screen):
     |      >>> screen.window_width()
     |      640
     |
     |  ----------------------------------------------------------------------
     |  Methods inherited from TurtleScreenBase:
     |
     |  mainloop(self)
     |      Starts event loop - calling Tkinter's mainloop function.
     |
     |      No argument.
     |
     |      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()
     |
     |  numinput(self, 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 imput
     |      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)
     |
     |  textinput(self, 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:")
     |
     |  ----------------------------------------------------------------------
     |  Data descriptors inherited from TurtleScreenBase:
     |
     |  __dict__
     |      dictionary for instance variables (if defined)
     |
     |  __weakref__
     |      list of weak references to the object (if defined)

    class Vec2D(builtins.tuple)
     |  A 2 dimensional vector class, used as a helper class
     |  for implementing turtle graphics.
     |  May be useful for turtle graphics programs also.
     |  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
     |     |a| absolute value of a
     |     a.rotate(angle) rotation
     |
     |  Method resolution order:
     |      Vec2D
     |      builtins.tuple
     |      builtins.object
     |
     |  Methods defined here:
     |
     |  __abs__(self)
     |
     |  __add__(self, other)
     |      Return self+value.
     |
     |  __getnewargs__(self)
     |
     |  __mul__(self, other)
     |      Return self*value.n
     |
     |  __neg__(self)
     |
     |  __repr__(self)
     |      Return repr(self).
     |
     |  __rmul__(self, other)
     |      Return self*value.
     |
     |  __sub__(self, other)
     |
     |  rotate(self, angle)
     |      rotate self counterclockwise by angle
     |
     |  ----------------------------------------------------------------------
     |  Static methods defined here:
     |
     |  __new__(cls, x, y)
     |      Create and return a new object.  See help(type) for accurate
signature.
```

```
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  ----------------------------------------------------------------------
 |  Methods inherited from builtins.tuple:
 |
 |  __contains__(self, key, /)
 |      Return key in self.
 |
 |  __eq__(self, value, /)
 |      Return self==value.
 |
 |  __ge__(self, value, /)
 |      Return self>=value.
 |
 |  __getattribute__(self, name, /)
 |      Return getattr(self, name).
 |
 |  __getitem__(self, key, /)
 |      Return self[key].
 |
 |  __gt__(self, value, /)
 |      Return self>value.
 |
 |  __hash__(self, /)
 |      Return hash(self).
 |
 |  __iter__(self, /)
 |      Implement iter(self).
 |
 |  __le__(self, value, /)
 |      Return self<=value.
 |
 |  __len__(self, /)
 |      Return len(self).
 |
 |  __lt__(self, value, /)
 |      Return self<value.
 |
 |  __ne__(self, value, /)
 |      Return self!=value.
 |
 |  count(…)
 |      T.count(value) -> integer -- return number of occurrences of value
```

```
     |
     |  index(…)
     |      T.index(value, [start, [stop]]) -> integer -- return first index of
value.
     |      Raises ValueError if the value is not present.

FUNCTIONS
    Screen()
        Return the singleton screen object.
        If none exists at the moment, create a new one and return it,
        else return the existing one.

    addshape(name, shape=None)
        Adds a turtle shape to TurtleScreen's shapelist.

        Arguments:
        (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!
        (2) name is an arbitrary string and shape is a tuple
            of pairs of coordinates. Installs the corresponding
            polygon shape
        (3) name is an arbitrary string and shape is a
            (compound) Shape object. Installs the corresponding
            compound shape.
        To use a shape, you have to issue the command shape(shapename).

        call: register_shape("turtle.gif")
        --or: register_shape("tri", ((0,0), (10,10), (-10,10)))

        Example:
        >>> register_shape("triangle", ((5,-3),(0,5),(-5,-3)))

    back(distance)
        Move the turtle backward by distance.

        Aliases: back | backward | bk

        Argument:
        distance -- a number

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

        Example:
        >>> position()
        (0.00, 0.00)
        >>> backward(30)
        >>> position()
        (-30.00, 0.00)

    backward(distance)
        Move the turtle backward by distance.

        Aliases: back | backward | bk

        Argument:
        distance -- a number

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

        Example:
        >>> position()
        (0.00, 0.00)
        >>> backward(30)
        >>> position()
        (-30.00, 0.00)

    begin_fill()
        Called just before drawing a shape to be filled.

        No argument.

        Example:
        >>> color("black", "red")
        >>> begin_fill()
        >>> circle(60)
        >>> end_fill()

    begin_poly()
        Start recording the vertices of a polygon.

        No argument.

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

        Example:
        >>> begin_poly()

    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'

    bgpic(picname=None)
        Set background image or return name of current backgroundimage.

        Optional argument:
        picname -- a string, name of a gif-file or "nopic".

        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.

        Example:
        >>> bgpic()
        'nopic'
        >>> bgpic("landscape.gif")
        >>> bgpic()
        'landscape.gif'

    bk(distance)
        Move the turtle backward by distance.

        Aliases: back | backward | bk

        Argument:
        distance -- a number

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

        Example:
        >>> position()
        (0.00, 0.00)
        >>> backward(30)
        >>> position()
        (-30.00, 0.00)

    bye()
        Shut the turtlegraphics window.
        Example:
        >>> bye()

    circle(radius, extent=None, steps=None)
        Draw a circle with given radius.

        Arguments:
        radius -- a number
        extent (optional) -- a number
        steps (optional) -- an integer

        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. Maybe used to draw regular
        polygons.

        call: circle(radius)                  # full circle
        --or: circle(radius, extent)          # arc
        --or: circle(radius, extent, steps)
        --or: circle(radius, steps=6)         # 6-sided polygon

        Example:
        >>> circle(50)
        >>> circle(120, 180)  # semicircle

    clear()
        Delete the turtle's drawings from the screen. Do not move

        No arguments.

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

        Examples:
        >>> clear()

    clearscreen()
        Delete all drawings and all turtles from the TurtleScreen.
        No argument.

        Reset empty TurtleScreen to its initial state: white background,
        no backgroundimage, no eventbindings and tracing on.

        Example:
        >>> clear()

        Note: this method is not available as function.

    clearstamp(stampid)
        Delete stamp with given stampid

        Argument:
        stampid - an integer, must be return value of previous stamp() call.

        Example:
        >>> color("blue")
        >>> astamp = stamp()
        >>> fd(50)
        >>> clearstamp(astamp)

    clearstamps(n=None)
        Delete all or first/last n of turtle's stamps.

        Optional argument:
        n -- an integer

        If n is None, delete all of pen's stamps,
        else if n > 0 delete first n stamps
        else if n < 0 delete last n stamps.

        Example:
        >>> for i in range(8):
        …     stamp(); fd(30)
        …
        >>> clearstamps(2)
        >>> clearstamps(-2)
        >>> clearstamps()

    clone()
        Create and return a clone of the

        No argument.

        Create and return a clone of the turtle with same position, heading
        and turtle properties.
        Example (for a Turtle instance named mick):
        mick = Turtle()
        joe = mick.clone()

    color(*args)
        Return or set the pencolor and fillcolor.

        Arguments:
        Several input formats are allowed.
        They use 0, 1, 2, or 3 arguments as follows:

        color()
            Return the current pencolor and the current fillcolor
            as a pair of color specification strings as are returned
            by pencolor and fillcolor.
        color(colorstring), color((r,g,b)), color(r,g,b)
            inputs as in 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.
        For mor info see: pencolor, fillcolor

        Example:
        >>> color('red', 'green')
        >>> color()
        ('red', 'green')
        >>> colormode(255)
        >>> color((40, 80, 120), (160, 200, 240))
        >>> color()
        ('#285078', '#a0c8f0')

    colormode(cmode=None)
        Return the colormode or set it to 1.0 or 255.

        Optional argument:
        cmode -- one of the values 1.0 or 255

        r, g, b values of colortriples have to be in range 0..cmode.

        Example:
        >>> colormode()
        1.0
        >>> colormode(255)
        >>> pencolor(240,160,80)

    degrees(fullcircle=360.0)
        Set angle measurement units to degrees.

        Optional argument:
        fullcircle -  a number

        Set angle measurement units, i. e. set number
        of 'degrees' for a full circle. Dafault value is
        360 degrees.

        Example:
        >>> left(90)
        >>> heading()
        90

        Change angle measurement unit to grad (also known as gon,
        grade, or gradian and equals 1/100-th of the right angle.)
        >>> degrees(400.0)
        >>> heading()
        100

    delay(delay=None)
        Return or set the drawing delay in milliseconds.

        Optional argument:
        delay -- positive integer

        Example:
        >>> delay(15)
        >>> delay()
        15

    distance(x, y=None)
        Return the distance from the turtle to (x,y) in turtle step units.

        Arguments:
        x -- a number   or  a pair/vector of numbers   or   a turtle instance
        y -- a number       None                            None

        call: distance(x, y)         # two coordinates
        --or: distance((x, y))       # a pair (tuple) of coordinates
        --or: distance(vec)          # e.g. as returned by pos()
        --or: distance(mypen)        # where mypen is another turtle

        Example:
        >>> pos()
        (0.00, 0.00)
        >>> distance(30,40)
        50.0
        >>> pen = Turtle()
        >>> pen.forward(77)
        >>> distance(pen)
        77.0

    done = mainloop()
        Starts event loop - calling Tkinter's mainloop function.

        No argument.

        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:
        >>> mainloop()

    dot(size=None, *color)
        Draw a dot with diameter size, using color.

        Optional arguments:
        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.

        Example:
        >>> dot()
        >>> fd(50); dot(20, "blue"); fd(50)

    down()
        Pull the pen down -- drawing when moving.

        Aliases: pendown | pd | down

        No argument.

        Example:
        >>> pendown()

    end_fill()
        Fill the shape drawn after the call begin_fill().

        No argument.
        Example:
        >>> color("black", "red")
        >>> begin_fill()
        >>> circle(60)
        >>> end_fill()

    end_poly()
        Stop recording the vertices of a polygon.

        No argument.

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

        Example:
        >>> end_poly()

    exitonclick()
        Go into mainloop until the mouse is clicked.

        No arguments.

        Bind bye() method to mouseclick on TurtleScreen.
        If "using_IDLE" - value in configuration dictionary is False
        (default value), enter mainloop.
        If IDLE with -n switch (no subprocess) is used, this value should be
        set to True in turtle.cfg. In this case IDLE's mainloop
        is active also for the client script.

        This is a method of the Screen-class and not available for
        TurtleScreen instances.

        Example:
        >>> exitonclick()

    fd(distance)
        Move the turtle forward by the specified distance.

        Aliases: forward | fd

        Argument:
        distance -- a number (integer or float)

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

        Example:
```

```
    >>> position()
    (0.00, 0.00)
    >>> forward(25)
    >>> position()
    (25.00,0.00)
    >>> forward(-75)
    >>> position()
    (-50.00,0.00)

fillcolor(*args)
    Return or set the fillcolor.

    Arguments:
    Four input formats are allowed:
      - fillcolor()
        Return the current fillcolor as color specification string,
        possibly in hex-number format (see example).
        May be used as input to another color/pencolor/fillcolor call.
      - fillcolor(colorstring)
        s is a Tk color specification string, such as "red" or "yellow"
      - fillcolor((r, g, b))
        *a tuple* of r, g, and b, which represent, an RGB color,
        and each of r, g, and b are in the range 0..colormode,
        where colormode is either 1.0 or 255
      - fillcolor(r, g, b)
        r, g, and b represent an RGB color, and each of r, g, and b
        are in the range 0..colormode

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

    Example:
    >>> fillcolor('violet')
    >>> col = pencolor()
    >>> fillcolor(col)
    >>> fillcolor(0, .5, 0)

filling()
    Return fillstate (True if filling, False else).

    No argument.

    Example:
    >>> begin_fill()
    >>> if filling():
    …     pensize(5)
    … else:
    …     pensize(3)
```

```
forward(distance)
    Move the turtle forward by the specified distance.

    Aliases: forward | fd

    Argument:
    distance -- a number (integer or float)

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

    Example:
    >>> position()
    (0.00, 0.00)
    >>> forward(25)
    >>> position()
    (25.00,0.00)
    >>> forward(-75)
    >>> position()
    (-50.00,0.00)

get_poly()
    Return the lastly recorded polygon.

    No argument.

    Example:
    >>> p = get_poly()
    >>> register_shape("myFavouriteShape", p)

get_shapepoly()
    Return the current shape polygon as tuple of coordinate pairs.

    No argument.

    Examples:
    >>> shape("square")
    >>> shapetransform(4, -1, 0, 2)
    >>> get_shapepoly()
    ((50, -20), (30, 20), (-50, 20), (-30, -20))

getcanvas()
    Return the Canvas of this TurtleScreen.

    No argument.

    Example:
```

```
    >>> cv = getcanvas()
    >>> cv
    <turtle.ScrolledCanvas instance at 0x010742D8>

getpen()
    Return the Turtleobject itself.

    No argument.

    Only reasonable use: as a function to return the 'anonymous turtle':

    Example:
    >>> pet = getturtle()
    >>> pet.fd(50)
    >>> pet
    <Turtle object at 0x0187D810>
    >>> turtles()
    [<Turtle object at 0x0187D810>]

getscreen()
    Return the TurtleScreen object, the turtle is drawing  on.

    No argument.

    Return the TurtleScreen object, the turtle is drawing  on.
    So TurtleScreen-methods can be called for that object.

    Example:
    >>> ts = getscreen()
    >>> ts
    <TurtleScreen object at 0x0106B770>
    >>> ts.bgcolor("pink")

getshapes()
    Return a list of names of all currently available turtle shapes.

    No argument.

    Example:
    >>> getshapes()
    ['arrow', 'blank', 'circle', … , 'turtle']

getturtle()
    Return the Turtleobject itself.

    No argument.

    Only reasonable use: as a function to return the 'anonymous turtle':
```

```
    Example:
    >>> pet = getturtle()
    >>> pet.fd(50)
    >>> pet
    <Turtle object at 0x0187D810>
    >>> turtles()
    [<Turtle object at 0x0187D810>]

goto(x, y=None)
    Move turtle to an absolute position.

    Aliases: setpos | setposition | goto:

    Arguments:
    x -- a number      or     a pair/vector of numbers
    y -- a number             None

    call: goto(x, y)         # two coordinates
    --or: goto((x, y))       # a pair (tuple) of coordinates
    --or: goto(vec)          # e.g. as returned by pos()

    Move turtle to an absolute position. If the pen is down,
    a line will be drawn. The turtle's orientation does not change.

    Example:
    >>> tp = pos()
    >>> tp
    (0.00, 0.00)
    >>> setpos(60,30)
    >>> pos()
    (60.00,30.00)
    >>> setpos((20,80))
    >>> pos()
    (20.00,80.00)
    >>> setpos(tp)
    >>> pos()
    (0.00,0.00)

heading()
    Return the turtle's current heading.

    No arguments.

    Example:
    >>> left(67)
    >>> heading()
    67.0
```

```
hideturtle()
    Makes the turtle invisible.

    Aliases: hideturtle | ht

    No argument.

    It's a good idea to do this while you're in the
    middle of a complicated drawing, because hiding
    the turtle speeds up the drawing observably.

    Example:
    >>> hideturtle()

home()
    Move turtle to the origin - coordinates (0,0).

    No arguments.

    Move turtle to the origin - coordinates (0,0) and set its
    heading to its start-orientation (which depends on mode).

    Example:
    >>> home()

ht()
    Makes the turtle invisible.

    Aliases: hideturtle | ht

    No argument.

    It's a good idea to do this while you're in the
    middle of a complicated drawing, because hiding
    the turtle speeds up the drawing observably.

    Example:
    >>> hideturtle()

isdown()
    Return True if pen is down, False if it's up.

    No argument.

    Example:
    >>> penup()
    >>> isdown()
    False
    >>> pendown()
    >>> isdown()
    True

isvisible()
    Return True if the Turtle is shown, False if it's hidden.

    No argument.

    Example:
    >>> hideturtle()
    >>> print isvisible():
    False

left(angle)
    Turn turtle left by angle units.

    Aliases: left | lt

    Argument:
    angle -- a number (integer or float)

    Turn turtle left by angle units. (Units are by default degrees,
    but can be set via the degrees() and radians() functions.)
    Angle orientation depends on mode. (See this.)

    Example:
    >>> heading()
    22.0
    >>> left(45)
    >>> heading()
    67.0

listen(xdummy=None, ydummy=None)
    Set focus on TurtleScreen (in order to collect key-events)

    No arguments.
    Dummy arguments are provided in order
    to be able to pass listen to the onclick method.

    Example:
    >>> listen()

lt(angle)
    Turn turtle left by angle units.

    Aliases: left | lt
    Argument:
    angle -- a number (integer or float)

    Turn turtle left by angle units. (Units are by default degrees,
    but can be set via the degrees() and radians() functions.)
    Angle orientation depends on mode. (See this.)

    Example:
    >>> heading()
    22.0
    >>> left(45)
    >>> heading()
    67.0

mainloop()
    Starts event loop - calling Tkinter's mainloop function.

    No argument.

    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:
    >>> mainloop()

mode(mode=None)
    Set turtle-mode ('standard', 'logo' or 'world') and perform reset.

    Optional argument:
    mode -- one of the strings 'standard', 'logo' or 'world'

    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.

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

    Examples:
    >>> mode('logo')   # resets turtle heading to north
    >>> mode()
    'logo'
```

```
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 imput
    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:
    >>> numinput("Poker", "Your stakes:", 1000, minval=10, maxval=10000)

onclick(fun, btn=1, add=None)
    Bind fun to mouse-click event on this turtle on canvas.

    Arguments:
    fun --  a function with two arguments, to which will be assigned
            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, new binding will be added, otherwise
            it will replace a former binding.

    Example for the anonymous turtle, i. e. the procedural way:

    >>> def turn(x, y):
    …     left(360)
    …
    >>> onclick(turn)  # Now clicking into the turtle will turn it.
    >>> onclick(None)  # event-binding will be removed

ondrag(fun, btn=1, add=None)
    Bind fun to mouse-move event on this turtle on canvas.

    Arguments:
    fun -- a function with two arguments, to which will be assigned
           the coordinates of the clicked point on the canvas.
    num -- number of the mouse-button defaults to 1 (left mouse button).

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

    Example:
```

```
    >>> ondrag(goto)

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

onkey(fun, key)
    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:

    >>> def f():
    …     fd(50)
    …     lt(60)
    …
    >>> onkey(f, "Up")
    >>> listen()

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

onkeypress(fun, key=None)
    Bind fun to key-press event of key if key is given,
    or to any key-press-event if no key is given.

    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():
    …     fd(50)
    …     lt(60)
    …
    >>> onkeypress(f, "Up")
    >>> 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.

onkeyrelease(fun, key)
    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:

    >>> def f():
    …     fd(50)
    …     lt(60)
    …
    >>> onkey(f, "Up")
    >>> listen()

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

onrelease(fun, btn=1, add=None)
    Bind fun to mouse-button-release event on this turtle on canvas.

    Arguments:
    fun -- a function with two arguments, to which will be assigned
            the coordinates of the clicked point on the canvas.
    num --  number of the mouse-button defaults to 1 (left mouse button).

    Example (for a MyTurtle instance named joe):
    >>> class MyTurtle(Turtle):
    …     def glow(self,x,y):
    …             self.fillcolor("red")
    …     def unglow(self,x,y):
    …             self.fillcolor("")
    …
    >>> joe = MyTurtle()
    >>> joe.onclick(joe.glow)
    >>> joe.onrelease(joe.unglow)

    Clicking on joe turns fillcolor red, unclicking turns it to
    transparent.
```

```
onscreenclick(fun, btn=1, add=None)
    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)

    >>> onclick(goto)
    >>> # Subsequently clicking into the TurtleScreen will
    >>> # make the turtle move to the clicked point.
    >>> onclick(None)

ontimer(fun, t=0)
    Install a timer, which calls fun after t milliseconds.

    Arguments:
    fun -- a function with no arguments.
    t -- a number >= 0

    Example:

    >>> running = True
    >>> def f():
    …     if running:
    …             fd(50)
    …             lt(60)
    …             ontimer(f, 250)
    …
    >>> f()   # makes the turtle marching around
    >>> running = False

pd()
    Pull the pen down -- drawing when moving.

    Aliases: pendown | pd | down

    No argument.

    Example:
    >>> pendown()

pen(pen=None, **pendict)
    Return or set the pen's attributes.
    Arguments:
        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)
       "shearfactor":   number
       "outline"    :   positive number
       "tilt"       :   number

    This dictionary can be used as argument for a subsequent
    pen()-call 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.


    Examples:
    >>> pen(fillcolor="black", pencolor="red", pensize=10)
    >>> pen()
    {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
    'pencolor': 'red', 'pendown': True, 'fillcolor': 'black',
    'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
    >>> penstate=pen()
    >>> color("yellow","")
    >>> penup()
    >>> pen()
    {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
    'pencolor': 'yellow', 'pendown': False, 'fillcolor': '',
    'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}
    >>> p.pen(penstate, fillcolor="green")
    >>> p.pen()
    {'pensize': 10, 'shown': True, 'resizemode': 'auto', 'outline': 1,
    'pencolor': 'red', 'pendown': True, 'fillcolor': 'green',
    'stretchfactor': (1,1), 'speed': 3, 'shearfactor': 0.0}

pencolor(*args)
    Return or set the pencolor.

    Arguments:
```

```
    Four input formats are allowed:
      - pencolor()
        Return the current pencolor as color specification string,
        possibly in hex-number format (see example).
        May be used as input to another color/pencolor/fillcolor call.
      - pencolor(colorstring)
        s is a Tk color specification string, such as "red" or "yellow"
      - pencolor((r, g, b))
        *a tuple* of r, g, and b, which represent, an RGB color,
        and each of r, g, and b are in the range 0..colormode,
        where colormode is either 1.0 or 255
      - pencolor(r, g, b)
        r, g, and b represent an RGB color, and each of r, g, and b
        are in the range 0..colormode

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

    Example:
    >>> pencolor('brown')
    >>> tup = (0.2, 0.8, 0.55)
    >>> pencolor(tup)
    >>> pencolor()
    '#33cc8c'

pendown()
    Pull the pen down -- drawing when moving.

    Aliases: pendown | pd | down

    No argument.

    Example:
    >>> pendown()

pensize(width=None)
    Set or return the line thickness.

    Aliases:  pensize | width

    Argument:
    width -- 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, current pensize
    is returned.
    Example:
    >>> pensize()
    1
    >>> pensize(10)   # from here on lines of width 10 are drawn

penup()
    Pull the pen up -- no drawing when moving.

    Aliases: penup | pu | up

    No argument

    Example:
    >>> penup()

pos()
    Return the turtle's current location (x,y), as a Vec2D-vector.

    Aliases: pos | position

    No arguments.

    Example:
    >>> pos()
    (0.00, 240.00)

position()
    Return the turtle's current location (x,y), as a Vec2D-vector.

    Aliases: pos | position

    No arguments.

    Example:
    >>> pos()
    (0.00, 240.00)

pu()
    Pull the pen up -- no drawing when moving.

    Aliases: penup | pu | up

    No argument

    Example:
    >>> penup()

radians()
```

```
    Set the angle measurement units to radians.

    No arguments.

    Example:
    >>> heading()
    90
    >>> radians()
    >>> heading()
    1.5707963267948966

register_shape(name, shape=None)
    Adds a turtle shape to TurtleScreen's shapelist.

    Arguments:
    (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!
    (2) name is an arbitrary string and shape is a tuple
        of pairs of coordinates. Installs the corresponding
        polygon shape
    (3) name is an arbitrary string and shape is a
        (compound) Shape object. Installs the corresponding
        compound shape.
    To use a shape, you have to issue the command shape(shapename).

    call: register_shape("turtle.gif")
    --or: register_shape("tri", ((0,0), (10,10), (-10,10)))

    Example:
    >>> register_shape("triangle", ((5,-3),(0,5),(-5,-3)))

reset()
    Delete the turtle's drawings and restore its default values.

    No argument.

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

    Example:
    >>> position()
    (0.00,-22.00)
    >>> heading()
    100.0
    >>> reset()
    >>> position()
    (0.00,0.00)
    >>> heading()
    0.0

resetscreen()
    Reset all Turtles on the Screen to their initial state.

    No argument.

    Example:
    >>> reset()

resizemode(rmode=None)
    Set resizemode to one of the values: "auto", "user", "noresize".

    (Optional) Argument:
    rmode -- one of the strings "auto", "user", "noresize"

    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()
      - "noresize" no adaption of the turtle's appearance takes place.
    If no argument is given, return current resizemode.
    resizemode("user") is called by a call of shapesize with arguments.


    Examples:
    >>> resizemode("noresize")
    >>> resizemode()
    'noresize'

right(angle)
    Turn turtle right by angle units.

    Aliases: right | rt

    Argument:
    angle -- a number (integer or float)

    Turn turtle right by angle units. (Units are by default degrees,
    but can be set via the degrees() and radians() functions.)
    Angle orientation depends on mode. (See this.)

    Example:
    >>> heading()
    22.0
    >>> right(45)
    >>> heading()
    337.0

rt(angle)
    Turn turtle right by angle units.

    Aliases: right | rt

    Argument:
    angle -- a number (integer or float)

    Turn turtle right by angle units. (Units are by default degrees,
    but can be set via the degrees() and radians() functions.)
    Angle orientation depends on mode. (See this.)

    Example:
    >>> heading()
    22.0
    >>> right(45)
    >>> heading()
    337.0

screensize(canvwidth=None, canvheight=None, bg=None)
    Resize the canvas the turtles are drawing on.

    Optional arguments:
    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 (canvaswidth, canvasheight)

    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!)

    Example (for a Turtle instance named turtle):
    >>> turtle.screensize(2000,1500)
    >>> # e.g. to search for an erroneously escaped turtle ;-)

seth(to_angle)
    Set the orientation of the turtle to to_angle.

    Aliases:  setheading | seth

    Argument:
    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

    Example:
    >>> setheading(90)
    >>> heading()
    90

setheading(to_angle)
    Set the orientation of the turtle to to_angle.

    Aliases:  setheading | seth

    Argument:
    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

    Example:
    >>> setheading(90)
    >>> heading()
    90

setpos(x, y=None)
    Move turtle to an absolute position.

    Aliases: setpos | setposition | goto:

    Arguments:
    x -- a number      or     a pair/vector of numbers
    y -- a number             None
    call: goto(x, y)         # two coordinates
    --or: goto((x, y))       # a pair (tuple) of coordinates
    --or: goto(vec)          # e.g. as returned by pos()

    Move turtle to an absolute position. If the pen is down,
    a line will be drawn. The turtle's orientation does not change.

    Example:
    >>> tp = pos()
    >>> tp
    (0.00, 0.00)
    >>> setpos(60,30)
    >>> pos()
    (60.00,30.00)
    >>> setpos((20,80))
    >>> pos()
    (20.00,80.00)
    >>> setpos(tp)
    >>> pos()
    (0.00,0.00)

setposition(x, y=None)
    Move turtle to an absolute position.

    Aliases: setpos | setposition | goto:

    Arguments:
    x -- a number      or     a pair/vector of numbers
    y -- a number             None

    call: goto(x, y)         # two coordinates
    --or: goto((x, y))       # a pair (tuple) of coordinates
    --or: goto(vec)          # e.g. as returned by pos()

    Move turtle to an absolute position. If the pen is down,
    a line will be drawn. The turtle's orientation does not change.

    Example:
    >>> tp = pos()
    >>> tp
    (0.00, 0.00)
    >>> setpos(60,30)
    >>> pos()
    (60.00,30.00)
    >>> setpos((20,80))
    >>> pos()
    (20.00,80.00)
    >>> setpos(tp)
    >>> pos()
    (0.00,0.00)

settiltangle(angle)
    Rotate the turtleshape to point in the specified direction

    Argument: angle -- 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).


    Examples:
    >>> shape("circle")
    >>> shapesize(5,2)
    >>> settiltangle(45)
    >>> stamp()
    >>> fd(50)
    >>> settiltangle(-45)
    >>> stamp()
    >>> fd(50)

setundobuffer(size)
    Set or disable undobuffer.

    Argument:
    size -- an integer or None

    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() function.
    If size is None, no undobuffer is present.

    Example:
    >>> setundobuffer(42)

setup(width=0.5, height=0.75, startx=None, starty=None)
    Set the size and position of the main window.

    Arguments:
    width: as integer a size in pixels, as float a fraction of the
      Default is 50% of
    height: as integer the height in pixels, as float a fraction of the
       Default is 75% of
    startx: if positive, starting position in pixels from the left
      edge of the screen, if negative from the right edge
      Default, startx=None is to center window horizontally.
    starty: if positive, starting position in pixels from the top
      edge of the screen, if negative from the bottom edge
      Default, starty=None is to center window vertically.

    Examples:
    >>> setup (width=200, height=200, startx=0, starty=0)

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

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

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

setworldcoordinates(llx, lly, urx, ury)
    Set up a user defined coordinate-system.

    Arguments:
    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 coodinat-system and switch to mode 'world' if necessary.
    This performs a reset. If mode 'world' is already active,
    all drawings are redrawn according to the new coordinates.

    But ATTENTION: in user-defined coordinatesystems angles may appear
    distorted. (see Screen.mode())

    Example:
    >>> setworldcoordinates(-10,-0.5,50,1.5)
    >>> for _ in range(36):
    …     left(10)
    …     forward(0.5)

setx(x)
    Set the turtle's first coordinate to x

    Argument:
    x -- a number (integer or float)

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

    Example:
    >>> position()
    (0.00, 240.00)
    >>> setx(10)
    >>> position()
    (10.00, 240.00)

sety(y)
    Set the turtle's second coordinate to y

    Argument:
    y -- a number (integer or float)

    Set the turtle's first coordinate to x, second coordinate remains
    unchanged.

    Example:
    >>> position()
    (0.00, 40.00)
    >>> sety(-10)
    >>> position()
    (0.00, -10.00)

shape(name=None)
    Set turtle shape to shape with given name / return current shapename.

    Optional argument:
    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.

    Example:
    >>> shape()
    'arrow'
    >>> shape("turtle")
    >>> shape()
    'turtle'

shapesize(stretch_wid=None, stretch_len=None, outline=None)
    Set/return turtle's stretchfactors/outline. Set resizemode to "user".

    Optional arguments:
       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 orientation
    stretch_len is stretchfactor in direction of turtles orientation.
    outline determines the width of the shapes's outline.

    Examples:
    >>> resizemode("user")
    >>> shapesize(5, 5, 12)
    >>> shapesize(outline=8)

shapetransform(t11=None, t12=None, t21=None, t22=None)
    Set or return the current transformation matrix of the turtle shape.

    Optional arguments: t11, t12, t21, t22 -- numbers.

    If none of the matrix elements are given, return the transformation
    matrix.
    Otherwise set the given elements and transform the turtleshape
    according to the matrix consisting of first row t11, t12 and
    second row t21, 22.
    Modify stretchfactor, shearfactor and tiltangle according to the
    given matrix.

    Examples:
    >>> shape("square")
    >>> shapesize(4,2)
    >>> shearfactor(-0.5)
    >>> shapetransform()
    (4.0, -1.0, -0.0, 2.0)

shearfactor(shear=None)
    Set or return the current shearfactor.

    Optional argument: shear -- number, tangent of the shear angle

    Shear the turtleshape according to the given shearfactor shear,
    which is the tangent of the shear angle. DO NOT change the
    turtle's heading (direction of movement).
    If shear is not given: return the current shearfactor, i. e. the
    tangent of the shear angle, by which lines parallel to the
    heading of the turtle are sheared.

    Examples:
    >>> shape("circle")
    >>> shapesize(5,2)
    >>> shearfactor(0.5)
    >>> shearfactor()
    >>> 0.5

showturtle()
    Makes the turtle visible.

    Aliases: showturtle | st

    No argument.

    Example:
    >>> hideturtle()
    >>> showturtle()

speed(speed=None)
    Return or set the turtle's speed.

    Optional argument:
    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 in the following way:
        '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 : *no* animation takes place. forward/back makes turtle jump
    and likewise left/right make the turtle turn instantly.

    Example:
    >>> speed(3)

st()
    Makes the turtle visible.

    Aliases: showturtle | st

    No argument.
    Example:
    >>> hideturtle()
    >>> showturtle()

stamp()
    Stamp a copy of the turtleshape onto the canvas and return its id.

    No argument.

    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).

    Example:
    >>> color("blue")
    >>> stamp()
    13
    >>> fd(50)

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:
    >>> textinput("NIM", "Name of first player:")

tilt(angle)
    Rotate the turtleshape by angle.

    Argument:
    angle - a number

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

    Examples:
    >>> shape("circle")
    >>> shapesize(5,2)
    >>> tilt(30)
    >>> fd(50)
    >>> tilt(30)
    >>> fd(50)
```

```
tiltangle(angle=None)
    Set or return the current tilt-angle.

    Optional argument: angle -- 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).
    If angle is not given: 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).

    Deprecated since Python 3.1

    Examples:
    >>> shape("circle")
    >>> shapesize(5,2)
    >>> tilt(45)
    >>> tiltangle()

title(titlestring)
    Set title of turtle-window

    Argument:
    titlestring -- a string, to appear in the titlebar of the
                   turtle graphics window.

    This is a method of Screen-class. Not available for TurtleScreen-
    objects.

    Example:
    >>> title("Welcome to the turtle-zoo!")

towards(x, y=None)
    Return the angle of the line from the turtle's position to (x, y).

    Arguments:
    x -- a number   or  a pair/vector of numbers   or   a turtle instance
    y -- a number       None                            None

    call: distance(x, y)         # two coordinates
    --or: distance((x, y))       # a pair (tuple) of coordinates
    --or: distance(vec)          # e.g. as returned by pos()
    --or: distance(mypen)        # where mypen is another turtle

    Return the angle, between the line from turtle-position to position
    specified by x, y and the turtle's start orientation. (Depends on
    modes - "standard" or "logo")
    Example:
    >>> pos()
    (10.00, 10.00)
    >>> towards(0,0)
    225.0

tracer(n=None, delay=None)
    Turns turtle animation on/off and set delay for update drawings.

    Optional arguments:
    n -- nonnegative  integer
    delay -- nonnegative  integer

    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())

    Example:
    >>> tracer(8, 25)
    >>> dist = 2
    >>> for i in range(200):
    …     fd(dist)
    …     rt(90)
    …     dist += 2

turtles()
    Return the list of turtles on the

    Example:
    >>> turtles()
    [<turtle.Turtle object at 0x00E11FB0>]

turtlesize(stretch_wid=None, stretch_len=None, outline=None)
    Set/return turtle's stretchfactors/outline. Set resizemode to "user".

    Optional arguments:
       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 orientation
    stretch_len is stretchfactor in direction of turtles orientation.
    outline determines the width of the shapes's outline.
    Examples:
    >>> resizemode("user")
    >>> shapesize(5, 5, 12)
    >>> shapesize(outline=8)

undo()
    undo (repeatedly) the last turtle action.

    No argument.

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

    Example:
    >>> for i in range(4):
    …     fd(50); lt(80)
    …
    >>> for i in range(8):
    …     undo()
    …

undobufferentries()
    Return count of entries in the undobuffer.

    No argument.

    Example:
    >>> while undobufferentries():
    …     undo()

up()
    Pull the pen up -- no drawing when moving.

    Aliases: penup | pu | up

    No argument

    Example:
    >>> penup()

update()
    Perform a TurtleScreen update.

width(width=None)
    Set or return the line thickness.
    Aliases:  pensize | width

    Argument:
    width -- 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, current pensize
    is returned.

    Example:
    >>> pensize()
    1
    >>> pensize(10)   # from here on lines of width 10 are drawn

window_height()
    Return the height of the turtle window.

    Example:
    >>> window_height()
    480

window_width()
    Return the width of the turtle window.

    Example:
    >>> window_width()
    640

write(arg, move=False, align='left', font=('Arial', 8, 'normal'))
    Write text at the current turtle position.

    Arguments:
    arg -- info, which is to be written to the TurtleScreen
    move (optional) -- True/False
    align (optional) -- one of the strings "left", "center" or right"
    font (optional) -- 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.

    Example:
    >>> write('Home = ', True, align="center")
    >>> write((0,0), True)
    write_docstringdict(filename='turtle_docstringdict')
        Create and write docstring-dictionary to file.

        Optional argument:
        filename -- a string, used as filename
                    default value is turtle_docstringdict

        Has to be called explicitly, (not used by the turtle-graphics classes)
        The docstring dictionary will be written to the Python script
<filname>.py
        It is intended to serve as a template for translation of the docstrings
        into different languages.

    xcor()
        Return the turtle's x coordinate.

        No arguments.

        Example:
        >>> reset()
        >>> left(60)
        >>> forward(100)
        >>> print xcor()
        50.0

    ycor()
        Return the turtle's y coordinate
        ---
        No arguments.

        Example:
        >>> reset()
        >>> left(60)
        >>> forward(100)
        >>> print ycor()
        86.6025403784

DATA
    __all__ = ['ScrolledCanvas', 'TurtleScreen', 'Screen', 'RawTurtle', 'T…

FILE
    /Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/turtle.py
```

[1]:
```python
import turtle                # Allows us to use turtles
wn = turtle.Screen()         # Creates a playground for turtles
```

```
alex = turtle.Turtle()    # Create a turtle, assign to alex

alex.forward(50)          # Tell alex to move forward by 50 units
alex.left(90)             # Tell alex to turn by 90 degrees
alex.forward(30)          # Complete the second side of a rectangle

wn.mainloop()             # Wait for user to close window
```

1.1.1 add some colors

[2]:
```
import turtle
wn = turtle.Screen()
wn.bgcolor("lightgreen")      # Set the window background color
wn.title("Hello, Tess!")      # Set the window title

tess = turtle.Turtle()
tess.color("blue")            # Tell tess to change her color
tess.pensize(3)               # Tell tess to set her pen width

tess.forward(50)
tess.left(120)
tess.forward(50)

wn.mainloop()
```

[ ]:
```

```