Skulpt Turtle API Documentaion

```
import turtle

t = turtle.Turtle()
```

turtle.back = 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)
```

turtle.backward

```
Help on function backward in turtle:

turtle.backward = 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)
```

turtle.begin_fill

```
Help on function begin_fill in turtle:

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

    No argument.

    Example:
    >>> begin_fill()
    >>> forward(100)
    >>> left(90)
    >>> forward(100)
    >>> left(90)
    >>> forward(100)
    >>> left(90)
    >>> forward(100)
    >>> end_fill()
```

turtle.bgcolor

```
Help on function bgcolor in turtle:
```

```
turtle.bgcolor = 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'
```

turtle.bk

```
Help on function bk in turtle:

turtle.bk = 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)
```

turtle.circle

```
Help on function circle in turtle:

turtle.circle = 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
```

turtle.clear

```
Help on function clear in turtle:

turtle.clear = 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()
```

turtle.clear

```
Help on function clear in turtle:

turtle.clear = 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()
```

turtle.color

```
Help on function color in turtle:

turtle.color = 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')
```

turtle.colormode

```
Help on function colormode in turtle:

turtle.colormode = 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)
```

turtle.delay

```
Help on function delay in turtle:

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

    Optional argument:
    delay -- positive integer

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

turtle.distance

```
Help on function distance in turtle:

turtle.distance = 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
```

turtle.dot

```
Help on function dot in turtle:

turtle.dot = 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)
```

turtle.down

```
Help on function down in turtle:

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

    Aliases: pendown | pd | down

    No argument.

    Example:
    >>> pendown()
```

turtle.end_fill

```
Help on function end_fill in turtle:

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

    No argument.

    Example:
    >>> begin_fill()
    >>> forward(100)
    >>> left(90)
    >>> forward(100)
    >>> left(90)
    >>> forward(100)
    >>> left(90)
    >>> forward(100)
    >>> end_fill()
```

turtle.exitonclick

```
Help on function exitonclick in turtle:

turtle.exitonclick = 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()
```

turtle.fd

```
Help on function fd in turtle:

turtle.fd = 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)
```

turtle.fill

```
Help on function fill in turtle:

turtle.fill = fill(flag=None)
    Call fill(True) before drawing a shape to fill, fill(False) when done.

    Optional argument:
    flag -- True/False (or 1/0 respectively)

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

    Example:
    >>> fill(True)
    >>> forward(100)
    >>> left(90)
    >>> forward(100)
    >>> left(90)
    >>> forward(100)
    >>> left(90)
    >>> forward(100)
    >>> fill(False)
```

turtle.fillcolor

```
Help on function fillcolor in turtle:

turtle.fillcolor = 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)
```

turtle.forward

```
Help on function forward in turtle:

turtle.forward = 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)
```

turtle.goto_$rw$

```
no Python documentation found for 'turtle.goto_$'
```

turtle.heading

```
Help on function heading in turtle:

turtle.heading = heading()
    Return the turtle's current heading.

    No arguments.

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

turtle.hideturtle

```
Help on function hideturtle in turtle:

turtle.hideturtle = 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()
```

turtle.home

```
Help on function home in turtle:

turtle.home = 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()
```

turtle.ht

```
Help on function ht in turtle:

turtle.ht = 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()
```

turtle.isdown

```
Help on function isdown in turtle:

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

    No argument.

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

turtle.isvisible

```
Help on function isvisible in turtle:

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

    No argument.

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

turtle.left

```
Help on function left in turtle:

turtle.left = 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
```

turtle.lt

```
Help on function lt in turtle:

turtle.lt = 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
```

turtle.pd

```
Help on function pd in turtle:

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

    Aliases: pendown | pd | down

    No argument.

    Example:
    >>> pendown()
```

turtle.pencolor

```
Help on function pencolor in turtle:

turtle.pencolor = 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'
```

turtle.pendown

```
Help on function pendown in turtle:

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

    Aliases: pendown | pd | down

    No argument.

    Example:
    >>> pendown()
```

turtle.pensize

```
Help on function pensize in turtle:

turtle.pensize = 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
```

turtle.penup

```
Help on function penup in turtle:

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

    Aliases: penup | pu | up

    No argument

    Example:
    >>> penup()
```

turtle.pos

```
Help on function pos in turtle:

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

    Aliases: pos | position

    No arguments.
    Example:
    >>> pos()
    (0.00, 240.00)
```

turtle.position

```
Help on function position in turtle:

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

    Aliases: pos | position

    No arguments.

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

turtle.pu

```
Help on function pu in turtle:

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

    Aliases: penup | pu | up

    No argument

    Example:
    >>> penup()
```

turtle.reset

```
Help on function reset in turtle:

turtle.reset = 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
```

turtle.right

```
Help on function right in turtle:

turtle.right = 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
```

turtle.rt

```
Help on function rt in turtle:

turtle.rt = 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
```

turtle.seth

```
Help on function seth in turtle:

turtle.seth = 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
```

turtle.setheading

```
Help on function setheading in turtle:

turtle.setheading = 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
```

turtle.setpos

```
Help on function setpos in turtle:

turtle.setpos = 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)
```

turtle.setposition

```
Help on function setposition in turtle:

turtle.setposition = 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)
```

turtle.setup

```
Help on function setup in turtle:

turtle.setup = 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
```

turtle.setworldcoordinates

```
Help on function setworldcoordinates in turtle:

turtle.setworldcoordinates = 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)
```

turtle.setworldcoordinates

```
Help on function setworldcoordinates in turtle:

turtle.setworldcoordinates = 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)
```

turtle.setx

```
Help on function setx in turtle:

turtle.setx = 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)
```

turtle.sety

```
Help on function sety in turtle:

turtle.sety = 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)
```

turtle.shape

```
Help on function shape in turtle:

turtle.shape = 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'
```

turtle.showturtle

```
Help on function showturtle in turtle:

turtle.showturtle = showturtle()
    Makes the turtle visible.

    Aliases: showturtle | st

    No argument.

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

turtle.speed

```
Help on function speed in turtle:

turtle.speed = 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)
```

turtle.st

```
Help on function st in turtle:

turtle.st = st()
```

```
    Makes the turtle visible.

    Aliases: showturtle | st

    No argument.

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

turtle.stamp

```
Help on function stamp in turtle:

turtle.stamp = 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)
```

turtle.title

```
Help on function title in turtle:

turtle.title = 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!")
```

turtle.towards

```
Help on function towards in turtle:

turtle.towards = 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
```

turtle.tracer

```
Help on function tracer in turtle:

turtle.tracer = tracer(flag=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
```

turtle.tracer

```
Help on function tracer in turtle:

turtle.tracer = tracer(flag=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
```

turtle.turtles

```
Help on function turtles in turtle:

turtle.turtles = turtles()
    Return the list of turtles on the

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

turtle.up

```
Help on function up in turtle:

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

    Aliases: penup | pu | up

    No argument

    Example:
```

```
    >>> penup()
```

turtle.update

```
Help on function update in turtle:

turtle.update = update()
    Perform a TurtleScreen update.
```

turtle.update

```
Help on function update in turtle:

turtle.update = update()
    Perform a TurtleScreen update.
```

turtle.width

```
Help on function width in turtle:

turtle.width = 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
```

turtle.window_height

```
Help on function window_height in turtle:

turtle.window_height = window_height()
    Return the height of the turtle window.

    No argument.

    Example (for a TurtleScreen instance named screen):
    >>> screen.window_height()
    480
```

turtle.window_width

```
Help on function window_width in turtle:

turtle.window_width = window_width()
    Returns the width of the turtle window.

    No argument.

    Example (for a TurtleScreen instance named screen):
    >>> screen.window_width()
    640
```

turtle.write

```
Help on function write in turtle:

turtle.write = 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)
```

turtle.xcor

```
Help on function xcor in turtle:

turtle.xcor = xcor()
    Return the turtle's x coordinate.

    No arguments.

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

turtle.ycor

```
Help on function ycor in turtle:

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

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