TURTLE

Introduction - Introduction

Turtle graphics is a popular way for introducing programming to kids. It was part of the original Logo programming language developed by Wally Feurzeig, Seymour Papert and Cynthia Solomon in 1967.

Turtle methods :

- **Move and draw –**

1. **forward() | fd()** - Move the turtle forward by the specified distance, in the direction the turtle is headed.

   Syntax - turtle.forward(distance)
   turtle.fd(distance)

2. **backward() | bk() | back()** - Move the turtle backward by distance, opposite to the direction the turtle is headed. Do not change the turtle's heading.

   Syntax - turtle.back(distance)
   turtle.bk(distance)
   turtle.backward(distance)

3. **right() | rt() -** 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 the turtle mode.

   Syntax - turtle.right(angle)
   turtle.rt(angle)

4. **left() | lt() -** 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 the turtle mode.

   Syntax - turtle.left(angle)
   turtle.lt(angle)

5. **goto() | setpos() | setposition() -** Move turtle to an absolute position. If the pen is down, draw line. Do not change the turtle's orientation.

   Syntax - turtle.goto(x, y=None)
   turtle.setpos(x, y=None)
   turtle.setposition(x, y=None)

6. **setx() -** Set the turtle's first coordinate to x, leave second coordinate unchanged**.**

   Syntax - turtle.setx(x)

7. **sety() -** Set the turtle's second coordinate to y, leave first coordinate unchanged.

   Syntax - turtle.sety(y)

setheading() | seth() - 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

Syntax - turtle.setheading(to_angle)
turtle.seth(to_angle)

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

   Syntax - turtle.home()

9. **circle() -** Draw a circle with given radius**.**

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

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

    Syntax - turtle.dot(size=None, *color)

11. **stamp() -** Stamp a copy of the turtle shape onto the canvas at the current turtle position. Return a stamp_id for that stamp

    Syntax - turtle.stamp()

12. **clearstamp() -** Delete stamp with given stampid.

    Syntax - turtle.clearstamp(stampid)

13. **clearstamps() -** Delete all or first/last n of turtle's stamps. If n is None, delete all stamps, if n > 0 delete first n stamps, else if n < 0 delete last n stamps.

    Syntax - turtle.clearstamps(n=None)

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

    Syntax - turtle.undo()

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

    Syntax - turtle.speed(speed=None)

- ## Tell Turtle's state

1. **position() | pos() –** Return the turtle's current location (x,y)
   Syntax - turtle.position()
   turtle.pos()

2. **towards() -** Return the angle between the line from turtle position to position specified by (x,y), the vector or the other turtle.
   Syntax - turtle.towards(x, y=None)

3. **xcor() -** Return the turtle's x coordinate.
   Syntax - turtle.xcor()

4. **ycor() -** Return the turtle's y coordinate.
   Syntax - turtle.ycor()

5. **heading() -** Return the turtle's current heading (value depends on the turtle mode.
   Syntax - turtle.heading()

6. **distance() -** Return the distance from the turtle to (x,y), the given vector, or the given other turtle, in turtle step units.
   Syntax - turtle.distance(x, y=None)

- **Setting and measurement**

1. **degrees() -** Set angle measurement units, i.e. set number of “degrees” for a full circle
   Syntax - turtle.degrees(fullcircle=360.0)

2. **radians() -** Set the angle measurement units to radians.
   Syntax - turtle.radians()

- Pen control
- Drawing state –

1. **pendown() | pd() | down()** – Pull the pen down – drawing when moving.
   Syntax - turtle.pendown()
   turtle.pd()
   turtle.down()

2. **penup() | pu() | up() -** Pull the pen up – no drawing when moving.
   Syntax - turtle.penup()
   turtle.pu()
   turtle.up()

3. **pensize() | width() -** Set the line thickness.
   Syntax - turtle.pensize(width=None)
   turtle.width(width=None)

4. **pen() -** 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
   Syntax - turtle.pen(pen=None, **pendict)

5. **isdown() -** Return True if pen is down, False if it’s up.

Syntax - turtle.isdown()

- **Color control -**

1. **color() -** Return or set pencolor and fillcolor.
   Syntax - turtle.color(*args)

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

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

- **Filling –**

1. filling() - Return fillstate (True if filling, False else).
   Syntax - turtle.filling()

2. **begin_fill() -** To be called just before drawing a shape to be filled.
   Syntax - turtle.begin_fill()

3. **end_fill()-** Fill the shape drawn after the last call to begin_fill().
   Syntax - turtle.end_fill()

- **More drawing control –**
  1. **reset() -** Reset all Turtles on the Screen to their initial state.
     Syntax - reset() - turtle.reset()
     turtle.resetscreen()

2. **clear() -** Delete all drawings and all turtles from the TurtleScreen. Reset the now empty TurtleScreen to its initial state: white background, no background image, no event bindings and tracing on.
   Syntax - turtle.clear()
   turtle.clearscreen()

3. **write() -** Write text - the string representation of arg - at the current turtle position according to align ("left", "center" or right") and with the given font.
   Syntax - turtle.write(arg, move=False, align="left", font=("Arial", 8, "normal"))

- **Turtle state :**
- **Visibility –**

1. **showturtle() | st() –** Make the turtle visible.
   Syntax - turtle.showturtle()
   turtle.st()

2. **hideturtle()-** Make the turtle invisible.
   Syntax - turtle.hideturtle()
   turtle.ht()

3. **isvisible()-** Return True if the Turtle is shown, False if it's hidden.
   Syntax - turtle.isvisible()

- Appearance -

1. **shape() -** Set turtle shape to shape with given name or, if name is not given, return name of current shape.
   Syntax - turtle.shape(name=None)

2. **resizemode()-** Set resizemode to one of the values: "auto", "user", "noresize". If rmode is not given, return current resizemode.

Syntax - turtle.resizemode(rmode=None)

3. **shapesize() | turtlesize() -** Return or set the pen's attributes x/y-stretchfactors and/or outline.
   Syntax - turtle.shapesize(stretch_wid=None, stretch_len=None, outline=None)
   turtle.turtlesize(stretch_wid=None, stretch_len=None, outline=None)

4. **shearfactor() -** Set or return the current shearfactor.
   Syntax - turtle.shearfactor(shear=None)

5. **settiltangle() -** Rotate the turtleshape to point in the direction specified by angle, regardless of its current tilt-angle.
   Syntax - turtle.settiltangle(angle)

6. **tiltangle**() - Set or return the current tilt-angle.
   Syntax - turtle.tiltangle(angle=None)

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

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

9. **get_shapepoly()-** Return the current shape polygon as tuple of coordinate pairs. This can be used to define a new shape or components of a compound shape.
   Syntax - turtle.get_shapepoly()

- **Using events**

1. **onclick() -** Bind fun to mouse-click events on this screen. If fun is None, existing bindings are removed.
   Syntax - turtle.onclick(fun, btn=1, add=None)
   turtle.onscreenclick(fun, btn=1, add=None)

2. **onrelease() -** Bind fun to mouse-button-release events on this turtle. If fun is None, existing bindings are removed.
   Syntax - turtle.onrelease(fun, btn=1, add=None)

3. **ondrag() -** Bind fun to mouse-move events on this turtle. If fun is None, existing bindings are removed.
   Syntax - turtle.ondrag(fun, btn=1, add=None)

- **Special Turtle methods**
  1. **begin_poly() -** Start recording the vertices of a polygon. Current turtle position is first vertex of polygon.
     Syntax - turtle.begin_poly()

  2. **end_poly()-** Stop recording the vertices of a polygon. Current turtle position is last vertex of polygon. This will be connected with the first vertex.
     Syntax - turtle.end_poly()

  3. **get_poly()-** Return the last recorded polygon.
     Syntax - turtle.get_poly()

  4. **clone()-** Create and return a clone of the turtle with same position, heading and turtle properties.
     Syntax - turtle.clone()

5. **getturtle() | getpen() -** Return the Turtle object itself. Only reasonable use: as a function to return the “anonymous turtle”
   Syntax - turtle.getturtle()
   turtle.getpen()

6. **getscreen() -** Return the TurtleScreen object the turtle is drawing on. TurtleScreen methods can then be called for that object.
   Syntax - turtle.getscreen()

7. **setundobuffer()** - Set or disable undobuffer.
   Syntax - turtle.setundobuffer(size)

8. **undobufferentries()-** Return number of entries in the undobuffer.
   Syntax - turtle.undobufferentries()

- Methods of TurtleScreen/Screen:
- Window control –
  1. **bgcolor() -** Set or return background color of the TurtleScreen.
     Syntax - turtle.bgcolor(*args)

  2. **bgpic() -** Set background image or return name of current backgroundimage.
     Syntax - turtle.bgpic(picname=None)

  3. **clear() | clearscreen() -** Delete all drawings and all turtles from the TurtleScreen.
     Syntax - turtle.clear()
     turtle.clearscreen()

  4. **reset() -** Reset all Turtles on the Screen to their initial state.
     Syntax - turtle.reset()
     turtle.resetscreen()

5. **screensize() -** If no arguments are given, return current (canvaswidth, canvasheight).
   Syntax - turtle.screensize(canvwidth=None, canvheight=None, bg=None)

6. **setworldcoordinates() -** Set up user-defined coordinate system and switch to mode “world” if necessary.
   Syntax - turtle.setworldcoordinates(llx, lly, urx, ury)

- Animation control
  1. **delay() -** Set or return the drawing delay in milliseconds.
     Syntax - turtle.delay(delay=None)

  2. **tracer() -** Turn turtle animation on/off and set delay for update drawings.
     Syntax - turtle.tracer(n=None, delay=None)

  3. **update() -** Perform a TurtleScreen update.
     Syntax - turtle.update()

- **Using screen events**
  1. **listen() -** Set focus on TurtleScreen (in order to collect key-events).
     Syntax - turtle.listen(xdummy=None, ydummy=None)

  2. **onkey() -** Bind fun to key-release event of key.
     Syntax - turtle.onkey(fun, key)
     turtle.onkeyrelease(fun, key)

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

4. **onclick() -** Bind fun to mouse-click events on this screen. If fun is None, existing bindings are removed.
   Syntax - turtle.onclick(fun, btn=1, add=None)
   turtle.onscreenclick(fun, btn=1, add=None)

5. **ontimer()-** Install a timer that calls fun after t milliseconds.
   Syntax - turtle.ontimer(fun, t=0)

6. **mainloop() -** Starts event loop - calling Tkinter's mainloop function.
   Syntax - turtle.mainloop()
   turtle.done()

- **Settings and special methods**
  1. **mode() -** Set turtle mode ("standard", "logo" or "world") and perform reset.
     Syntax - turtle.mode(mode=None)

  2. **colormode() -** Return the colormode or set it to 1.0 or 255.
     Syntax - turtle.colormode(cmode=None)

  3. **getcanvas() -** Return the Canvas of this TurtleScreen.
     Syntax - turtle.getcanvas()

  4. **getshapes() -** Return a list of names of all currently available turtle shapes.
     Syntax - turtle.getshapes()

  5. **addshape() -** Add a turtle shape to TurtleScreen's shapelist. Only thusly registered shapes can be used by issuing the command shape(shapename).
     Syntax - turtle.register_shape(name, shape=None)
     turtle.addshape(name, shape=None)

6. **turtles() -** Return the list of turtles on the screen.
   Syntax - turtle.turtles()

7. **window_height() -** Return the height of the turtle window.
   Syntax - turtle.window_height()

8. **window_width() -** Return the width of the turtle window.
   Syntax - turtle.window_width()

- **Input methods**
  1. **textinput() -** Pop up a dialog window for input of a string.
     Syntax - turtle.textinput(title, prompt)

  2. **numinput() -** Pop up a dialog window for input of a number.
     Syntax - turtle.numinput(title, prompt, default=None, minval=None, maxval=None)

- Methods specific to Screen
  1. **bye() -** Shut the turtlegraphics window.
     Syntax - turtle.bye()

  2. **exitonclick() -** Bind bye() method to mouse clicks on the Screen.
     Syntax - turtle.exitonclick()

  3. **setup() -** Set the size and position of the main window.
     Syntax - turtle.setup(width=_CFG["width"], height=_CFG["height"], startx=_CFG["leftright"], starty=_CFG["topbottom"])

  4. **title() -** Set title of turtle window to titlestring.
     Syntax - turtle.title(titlestring)