Documentation for ColabTurtlePlus

Larry Riddle, Sept. 2021
https://github.com/mathriddle/ColabTurtlePlus

The ColabTurtlePlus module provides turtle graphics primitives, in both object-oriented and procedure-oriented ways. The procedural interface provides functions that are derived from the methods of the classes. They have the same names as the corresponding methods. A screen object is automatically created whenever a function derived from a Screen method is called. An (unnamed) turtle object is automatically created whenever any of the functions derived from a Turtle method is called.

To use multiple turtles on a screen one has to use the object-oriented interface for turtle methods.

Installation

Create an empty code cell and type:

```
!pip install ColabTurtlePlus
```

Run the code cell to install the library.

Usage

In any code cell, import the module using

```
from ColabTurtlePlus.Turtle import *
```

See https://docs.python.org/3/library/turtle.html for more details on most of the following methods from classic turtle.py. Those with an asterisk either do not occur in the classic turtle.py or have additional parameters from the corresponding method in turtle.py

Screen Methods

Window Controls

`setup(width,height)`
Set the size of the graphics window in pixels. Note that percentages are not used in this version.

`mode(mode)`
Set turtle mode ("standard", "logo", "world", or "svg") and reset the window. If mode is not given, current mode is returned. The "svg" mode is a special mode to handle how the original ColabTurtle worked. The coordinate system is the same as that used with SVG. The upper left corner is (0,0) with positive x direction going left to right, and the positive y direction going top to bottom. Positive angles are measured clockwise with 0° pointing right.

`bgcolor()`
`bgcolor(r,g,b)`
`bgcolor([r,g,b])`
`bgcolor(colorstring)`
The color can be given as three separate color arguments as in the RGB color encoding: red,green,blue. These three numbers can be given in a single tuple as well. The color can also be given as a single color string. The following formats are accepted for this color string:

- HTML standard color names: 140 color names defined as standard (see https://www.w3schools.com/colors/colors_names.asp) . Examples: `"red"`, `"black"`, `"magenta"`, `"cyan"` etc. Can also use `"none"` for no background color.
- Hex string with 3 or 6 digits, like `"#fff"`, `"FFF"`, `"#ddfdfd"`, `"#DFDFDF"`
- RGB string, like `"rgb(10 20 30)"`, `"rgb(10, 20, 30)"`

If no argument is given, returns the current background color.

`window_width()`
Return the width of the turtle window.

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

`*setworldcoordinates(llx,lly,urx,ury, aspect)`
Set up user-defined coordinate system and switch to mode "world". Depending on the window size, angles and circles may be distorted. To avoid this, set **`aspect=True`** which will resize the drawing window so that the aspect ratio of the drawing window will match that of the axes.

- **`llx`** : x-coordinate of lower left corner of canvas (min value on x-axis)
- **`lly`** : y-coordinate of lower left corner of canvas (min value on y-axis)
- **`urx`** : x-coordinate of upper right corner of canvas (max value on x-axis)
- **`ury`** : y-coordinate of upper right corner of canvas (max value on y-axis)

`*showborder(color)`
Show a border around the graphics window. Default (no parameters) is gray. A color can be specified in a similar way as with **`bgcolor`**.

`*hideborder()`
Hide the border around the graphics window.

`clear()`
Clears all text or drawings on the screen. Deletes all turtles. Note: this method is not available as a function. Use clearscreen() instead. Because variable preserve their values in Colab cells until the runtime is restarted, it is recommended to include clearscreen() in any Colab or Jupyter cell that is expected to be re-run more than once to clear all turtles before the next execution.

`reset()`
Resets all turtles to their initial state. Note: this method is not available as a function. Use resetscreen() instead.

`*initializescreen(window_mode) / *initializeTurtle(window_mode)`
This is included for backward compatibility with earlier non-class versions of ColabTurtlePlus. Set **`window=(w,h)`** to specify a window of width w pixels and height h pixels and **`mode`** to specify either 'standard', 'logo', 'world', or 'svg'. Default values [(800,600), 'standard'] are used if an argument is not given.

SVG Methods

`*showSVG(turtle)`
Print the SVG tags to the screen. The **`turtle`** argument (True or False) determines whether or not the turtles are included.

`*saveSVG(filename, turtle)`
Save the SVG tags to the file **`filename`**. The **`turtle`** argument (True or False) determines whether or not the turtles are included. If the filename is omitted, "SVGimage.svg" will be used.

Additional screen methods

`turtles()`
Return the list of turtles on the screen.

`*drawline(x1,y1,x2,y2) | line(x1,y1,x2,y2)`
`*drawline([x1,y1],(x2,y2)) | line([x1,y1],(x2,y2))`
Draw a line from (x1,y1) to (x2,y2)

Turtle Methods

Turtle Motion - Move and Draw

`forward(units) | fd(units)`
Moves the turtle in the direction it is facing, by **`units`** pixels

`backward(units) | bk(units) | back(units)`
Moves the turtle in the opposite of the direction it is facing, by **`units`** pixels

`right(degrees) | rt(degrees)`
Turns the turtle to right by the given **`degrees`**

`left(degrees) | lt(degrees)`
Turns the turtle to left by the given **`degrees`**

`goto(x,y) | setpos(x,y) | setposition(x,y)`
`goto((x,y)) | setpos((x,y)) | setposition((x,y))`
Moves the turtle to the point defined by x,y. The coordinates can be given separately, or in a single tuple.

`setx(x)`
Moves the turtle to the given **`x`** position, the y coordinate of the turtle stays the same.

`sety(y)`
Moves the turtle to the given **`y`** position, the x coordinate of the turtle stays the same.

`setheading(to_angle) | heading(to_angle) | face(to_angle)`
Sets the orientation of the turtle to **`to_angle`**. The result depends on the mode.

`home()`
Returns the turtle to the beginning position and angle. The turtle will continue drawing during this operation if the pen is down.

`*jumpto(x,y)`
Jump immediately to the point (x,y) without drawing or animation. The coordinates can be given separately, or in a single tuple.

`*circle(radius, extent, steps)`
Draws a circle of radius **`radius`** covering an extent of **`extent`** degrees. If extent is not given, draws the complete circle. If the radius is positive, the center of the circle is to the left of the turtle and the path is drawn in the counterclockwise direction. If the radius is negative, the center of the circle is to the right of the turtle and path is drawn in the clockwise direction. Number of steps is not used here for an actual circular arc since the circle is drawn using the svg circle function. However, the step argument is available primarily for backward compatibility with classic turtle.py circle. To get a true circular arc, do NOT use steps since the circle will be drawn using SVG commands. If steps > 20, it will be assumed that an arc of a circle was intended. While this function can still be used to draw a regular polygon with 20 or fewer sides, it is better to use the regularpolygon() function (see below) to take advantage of svg commands.

`dot(size,color)`
Draw a circular dot with diameter **`size`**, using **`color`**.

`*stamp(layer)`
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). The optional **`layer`** argument determines whether the stamp appears below other items (layer=0) or above other items (layer=1) in the order that SVG draws items. So if layer=0, a stamp may be covered by a filled object, for example, even if the stamp is originally drawn on top of the object during the animation. To prevent this, set layer=1 (or any nonzero number). The default is layer=0 if no argument is given.

`clearstamp(stampid)`
Delete stamp with given stampid, which can be an integer or a tuple of integers.

`clearstamps(n)`
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.

`*speed(s)`
Sets the speed of turtle's movements. **`s`** can be a value in the interval [0,13] where 1 is the slowest and 13 is the fastest for animation. If the speed is 0, no animation is drawn and only the final result is shown. The command **`done()`** must be executed to see the final image if speed=0. If **`s`** is omitted, the function returns the current speed.

`done() | update()`
Redraws the image. Needed if speed = 0 so that final image is drawn.

`*regularPolygon(sides,length,steps)`
Moves the turtle around a regular polygon of with size sides, with length being the length of each side. The optional steps argument indicates how many sides are drawn. The initial and concluding angle is half of the exterior angle. Positive values for sides or length draws the polygon to the left of the turtle's current direction, and a negative value forether sides or length draws it to the right of the turtle's current direction. The number of sides from 3 to 10 can also be given as a string using the name of the regular polygon (i.e. "triangle", "square", "pentagon", etc.)

Turtle Motion - Tell Turtle's State

`position() | pos()`
Returns the current x,y coordinates of the turtle as a tuple.

`towards(x,y) | towards((x,y))`
Return the angle between the line from turtle position to position specified by (x,y). This depends on the turtle's start orientation which depends on the mode - standard/world/logo/svg.

`xcor() | getx()`
Returns the current x coordinate of the turtle.

`ycor() | gety()`
Returns the current y coordinate of the turtle.

`heading() | getheading()`
Returns the direction that the turtle is looking at right now, in degrees.

`distance(x,y) | distance((x,y))`
Returns the turtle's distance to a given point x,y. The coordinates can be given separately or as a single tuple.

Turtle Motion -- Setting and measurement

`degrees()`
`radians()`
Sets the angle measurement for user input requiring an angle. All internal calculations are done in degrees after radians are converted.

Pen Control - Drawing State

`pendown() | pd()`
Puts the pen down, causing the turtle movements to start drawing again.

`penup() | pu() | up()`
Lifts the pen, turtles movement will not draw anything after this function is called.

`pensize(w) | width(w)`
Changes the width of the pen. If the parameter is omitted, returns the current pen width.

`pen(pen, **pendict)`
Return or set the pen's attributes.

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

Pen Control - Color Control

`color()`
Return the current pencolor and the current fillcolor

`color(colorstring) , color((r,g,b))`
set both fillcolor and pencolor to the given value

`color(string1,string2) , color((r1,g1,b1),(r2,g2,b2))`
Equivalent to pencolor(string1) and fillcolor(string2)

`pencolor()`
`pencolor(r,g,b)`
`pencolor((r,g,b))`
`pencolor(colorstring)`
If no parameter given, returns the current pen color as string. Else, changes the pen color The color can be given as three separate color arguments as in the RGB color encoding: red,green,blue. These three numbers can be given in a single tuple as well. The color can be given as a single color string, too! The following formats are accepted for this color string:

- HTML standard color names: 140 color names defined as standard (https://www.w3schools.com/colors/colors_names.asp) . Examples: `"red"`, `"black"`, `"magenta"`, `"cyan"` etc.
- Hex string with 3 or 6 digits, like `"#fff"`, `"FFF"`, `"#ddfdfd"`, `"#DFDFDF"`
- RGB string, like `"rgb(10 20 30)"`, `"rgb(10, 20, 30)"`

`fillcolor()`
`fillcolor(r,g,b)`
`fillcolor((r,g,b))`
`fillcolor(colorstring)`
Works the same as **`pencolor`** for the fillcolor.

Pen Control - Filling

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

`begin_fill()`
To be called just before drawing a shape to be filled.

`end_fill()`
Fill the shape drawn after the last call to begin_fill().

`*fillrule(rule)`
Sets the global fill_rule (nonzero or evenodd) used by SVG to fill an object. The global default fill-rule is evenodd to match the behavior of classic turtle.py. The **`begin_fill()`** function can take an argument of 'nonzero' or 'evenodd' to set the fill_rule just for that fill.

`*fillopacity(opacity)`
Sets the global fill-opacity used by SVG to fill an object. The default is 1. The **`begin_fill()`** function can take an argument between 0 and 1 to set the fill_opacity just for that fill.

Because the fill is controlled by svg rules, the result may differ from classic turtle fill. The fill-rule and fill-opacity can be set as arguments to the begin_fill() function and will apply only to objects filled before the end_fill is called. There are two possible arguments to specify the SVG fill-rule: 'nonzero' and 'evenodd'. The default is 'evenodd' to match the fill behavior in classic turtle.py. See https://developer.mozilla.org/en-US/docs/Web/SVG/Attribute/fill-rule for details.

Note: end_fill() will use the current pen color and current pen width for the boundary of the area being filled. If you want to change the pen color or width during the fill, you can do a separate begin_fill(), end_fill() pair for each color and/or width. Or you can execute the code twice, the first time with the fill, then a second time without the fill. It would be necessary to save the turtle position and heading before starting the first pass so that you can return to that position (using jumpto) and heading to repeat the code for the second pass.

Pen Control - More Drawing Control

`reset()`
Reset most (but not all) of the default values. Clears the turtle drawing and moves turtle back to the middle of the drawing window.

`clear()`
Clear the turtle drawing on the screen. Leave the turtle in its current location.

`*delete()`
Delete the turtle and all of its drawings. Can only be used as a method and is intended primarily for when there is more than one turtle. Use clearscreen() to delete all turtles.

`write(obj, align, font)`
Writes the string equivalent of any value to the screen. **`align`** and **`font`** named parameters can be given as arguments optionally. **`align`** must be one of `"left"`,`"center"`,`"right"`. It specifies where to put the text with respect to the turtle. **`font`** must be a tuple of three values like (**`"Arial"`**, **`20`**, **`"bold"`**) . The first value is the font name (only ones that your browser natively supports should be used), second value is the font size, and the third value is font style that must be one of `"normal"`,`"bold"`,`"italic"`,`"underline"`. Note that the "move" argument in the classic turtle.py write method is not used here.

Turtle State - Visibility

`showturtle() | st()`
Makes the turtle visible.

`hideturtle() | ht()`
Makes the turtle invisible.

`isvisible()`
Returns whether turtle is currently visible as boolean.

Turtle State - Appearance

`shape(name)`
Takes a shape name **`name`** and transforms the main character's look. This library has **`'classic'`**, **`'turtle'`**, **`'circle'`**, **`'arrow'`**, **`'triangle'`**, **`'square'`**, **`'ring'`**, **`'blank'`**, and **`'turtle2'`** shapes available, with **`'classic'`** as the default. If no argument is supplied, this function returns the name of the current shape. The **`'turtle'`** shape is the one that Tolga Atam included in his original ColabTurtle version. Use **`'turtle2'`** for the polygonal turtle shape from turtle.py. The circle shape from the original ColabTurtle was renamed **`'ring'`**. (If the mode is "svg" when a turtle is created, then the 'circle' and 'ring' shapes will be the same to be backward compatible with the original ColabTurtle.)

`shapesize(stretch_wid=None, stretch_len=None, outline=None) | turtlesize()`
Scale the size of the turtle. The turtle will be displayed stretched according to its stretchfactors: **`stretch_wid`** is *stretchfactor* perpendicular to its orientation, **`stretch_len`** is stretchfactor in direction of its orientation, **`outline`** determines the width of the shapes's outline.

`shearfactor(shear)`
Set or return the current shearfactor. *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.

`tiltangle(angle)`
Set or return the current tilt-angle. If angle is given, 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).

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

Animation Controls

`*delay(delay_time)`
Delay execution of next command for given delay_time (in seconds)

`*animationOff()`
`*animationOn()`
Turn off/on animation. When off, forward/back/circle makes turtle jump and likewise left/right make the turtle turn instantly.

Additional Functions

`Screen()`
Returns the screen object. If none exists at the moment, creates a new one and returns it, else returns the existing one.

`*oldDefaults()`
Set the defaults used in the original version of ColabTurtle module for backward compatibility. This should be called as the first line after importing ColabTurtlePlus.Turtle.

`*getcolor(n)`
Returns the color string in position n in the valid color list. Requires 0 <= n <= 139.

Examples

Examples of using ColabTurtlePlus in a Colaboratory or JupyterLab notebook can be found in the examples_version2 folder on Github.