

# diego2500garza/ColabTurtle

The program that this documentation relates to was started by Github user tolgaatam, and further modified by Github user YijiaXiong. This document follows the layout of the original [Turtle graphics Python documentation](#) in order to emphasize what is similar or different, for those familiar with the original. I did not create most of the functions in this ColabTurtle module, merely adapted it to best mimic the original turtle module and add in some additional features. Any explanations that I make are my best attempt to explain the code adapted from those two Github users, and don't reflect their thinking. As user YijiaXiong described it, this module is "an HTML based Turtle implementation, in order to work in Google Colab, Jupyter or any iPython environment."

---

## 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.*

*Imagine a robotic turtle starting at (0, 0) in the x-y plane. After an import turtle, give it the command 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 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.*

-- [From "Turtle Graphics", The Python Standard Library](#)

This turtle iteration uses SVG formatted strings and displays using an HTML object, imported from IPython.display. It also uses colormap features from [matplotlib.org](#) to create many different color possibilities.

To start and create a turtle window, you need to first call the initializeTurtle( ) function, which allows the window's size to be configured.

---

# Overview / Table of Contents

## **Turtle motion**

- Move and Draw
  - forward( ) | fd( )
  - backward( ) | back( ) | bk( )
  - right( ) | rt( )
  - left( ) | lt( )
  - setdirection( )
  - goto( ) | setpos( ) | setposition( )
  - setx( )
  - sety( )
  - circle( )
  - oval( )
  - stamp( )
  - drawline( )
  - regularpolygon( )
  - arrow( )
  - speed( )
- Tell Turtle's state
  - getx( )
  - gety( )
  - position( ) | pos( )
  - getdirection( )
  - distance( )

## **Pen Control**

- Drawing State
  - pendown( )
  - penup( )
  - getwidth( )
  - width( )
  - isdown( )
- Color Control
  - color( )
  - color\_rgb( )
  - initializeColors( )
  - setcolor( )
  - getcolor( )
- More Drawing Control
  - new\_window( )

## **Turtle's State**

- Visibility
  - hideturtle( ) | ht( )
  - showturtle( ) | st( )
  - isvisible( )

## **Window Control**

- bgcolor( )
- bgurl( )

### **Settings and Special methods**

- `window_width()`
- `window_height()`

### **Animation Control**

- `delay()`

### **Backend Functions**

---

# Turtle Motion

## Move and Draw

forward(units)

fd(units)

- Parameters: units - a number (integer or float)
- Move turtle forward by units (pixels), towards direction turtle is headed



backward(units)

back(units)

bk(units)

- Parameters: units - a number (integer or float)
- Move turtle by units (pixels), opposite direction the turtle is headed

right(degrees)

rt(degrees)

- Parameters: degrees - a number (integer or float)
- Rotate the turtle clockwise / to the right by given degrees. When initialized, turtle starts facing upward.



left(degrees)

lt(degrees)

- Parameters: degrees - a number (integer or float)
- Rotate turtle counter-clockwise / to the left by given degrees. When initialized, turtle starts facing upward.

setdirection(degrees)

- Parameters: degrees - a number (integer or float)
- Rotate the turtle to face the direction specified. Orientation to the right is considered 0 degrees. Degrees go clockwise from 0 degrees. When turtle is initialized it faces upwards by default, this is considered 270 degrees.

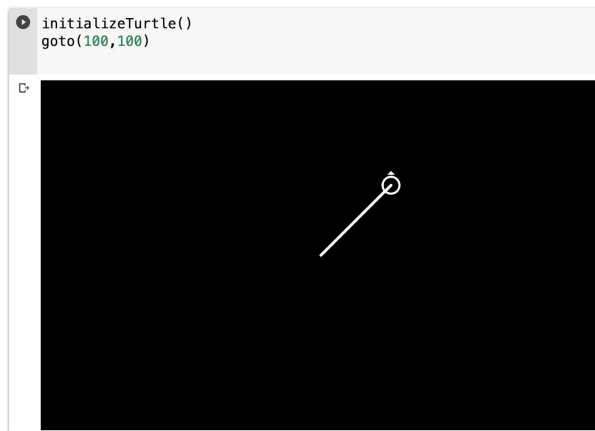


goto(given\_x, given\_y)

setpos(given\_x, given\_y)

setposition(given\_x, given\_y)

- Parameters: x - a number (integer or float) , y - a number (integer or float)
- Move turtle to an absolute position. Draws line if pen is down. Doesn't change turtle's orientation.



setx(given\_x)

- Parameters: given\_x - a number (integer or float)
- Set turtle's first coordinate to given\_x, leave second coordinate unchanged.

sety(given\_y)

- Parameters: given\_y - a number (integer or float)

- Set turtle's first coordinate to given\_y, leave second coordinate unchanged.

circle(radius, degrees = 360, steps = 180)

- Parameters: radius - a number (integer or float), degrees - a number (integer or float), steps - a number (integer)
- Draw a circle with given radius. The center is turtle's starting position. Degrees specifies if you would like only a certain amount of circle drawn. Amount of steps taken by turtle can be configured as well



oval(width, height, offset\_tilt = 0, steps=360)

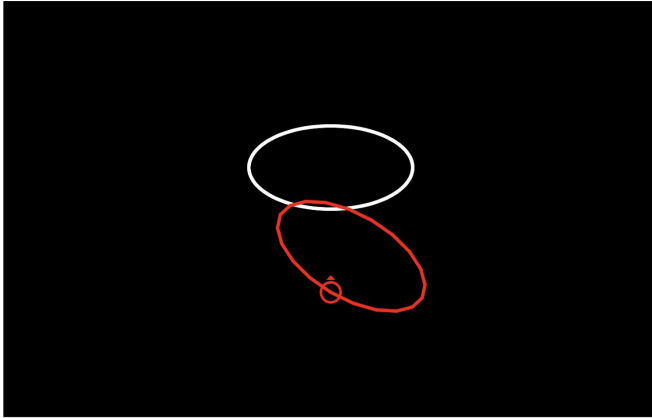
- Parameters: width - a number (integer or float), height - a number (integer or float), offset\_tilt - a number (integer or float), steps - a number (integer)
- Draw an oval with a provided width and height. Starts drawing at current location going counter-clockwise. Offset\_tilt allows you to tilt the oval in the clockwise direction. Amount of steps taken by turtle can be configured as well

```

initializeTurtle(initial_speed=13)
oval(width=100,height=50)

penup()
goto(0,-100)
color("red")
pendown()
oval(width=100, height=50, offset_tilt=30, steps=20)
# tilts 30 degrees to the right, less steps -> more jagged

```



stamp(desired\_position=position(), offset\_angle=0, pen\_width=2)

- Parameters: position - tuple of 2 numbers (integers or numbers), offset\_angle - a number (integer or float), pen\_width - a number (integer)
- Draws a stamp of a turtle at origin pointing upward by default. To stamp at current location, fill desired\_position = position(). Can configure to place stamp elsewhere, or offset clockwise by some angle.

```

initializeTurtle(initial_speed=13)

penup()
goto(100,100)
setdirection(30)
pendown()
stamp()

stamp(pen_width=4, desired_position=(-100,100), desired_offset_angle=220)
#can make pen thicker, make stamp @ certain location w/ specific angle

```



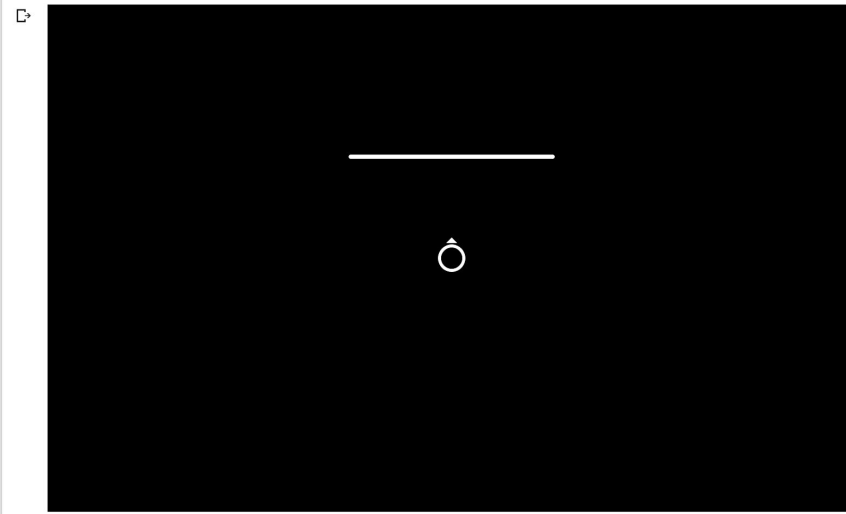
drawline(x\_1, y\_1, x\_2, y\_2)

- Parameters: x\_1, y\_1, x\_2, y\_2 - a number (integer or float)
- Draws a line from the (x\_1, y\_1) to the (x\_2, y\_2) position.

```

▶ initializeTurtle(initial_speed=13)
drawline(100,100, -100,100)

```



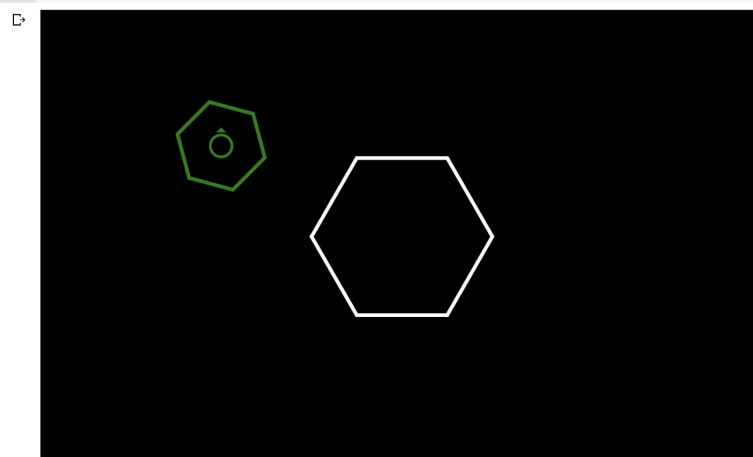
regularpolygon(num\_sides, side\_length=100, initial\_angle=0)

- Parameters: sides - a number (integer), side\_length - a number (integer or float), initial\_angle - a number (integer or float)
- Draws a regular polygon with num\_sides number of sides. Side length is default 100 pixels. initial\_angle can offset the polygon by rotating the starting turtle direction. Center is the location of the turtle before drawing.

```

▶ initializeTurtle(initial_speed=13)
regularpolygon(6)
penup()
goto(-200,100)
color("green")
regularpolygon(6, side_length=50, initial_angle=15)

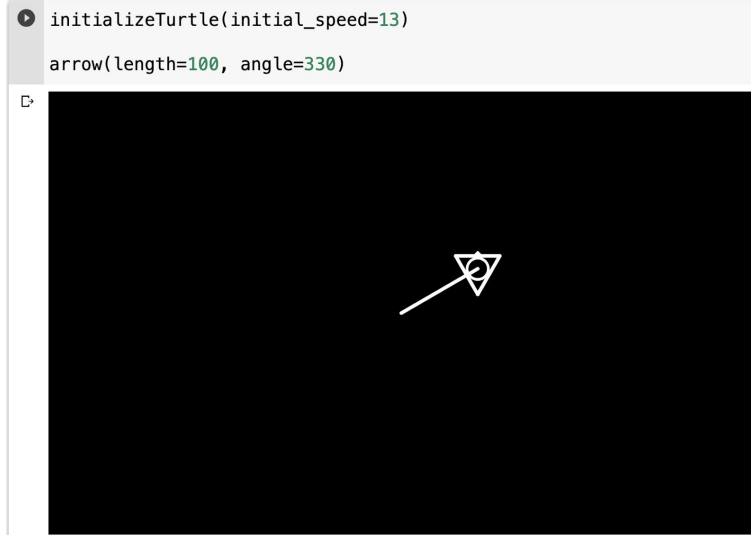
```



arrow(length, angle)

- Parameters: length - a number (integer or float), angle - a number (integer or float)
- Draws a basic arrow using regularpolygon function in direction provided with the straight line part being length units long.





speed(speed)

- Parameters: speed - an integer in range [1,13] inclusive
- Set the turtle's speed to an integer value in range 1...13. 13 is fastest, 1 is slowest.

### Tell Turtle's state

getx()

- Return the turtle's x coordinate

gety()

- Return the turtle's y coordinate

position()

pos()

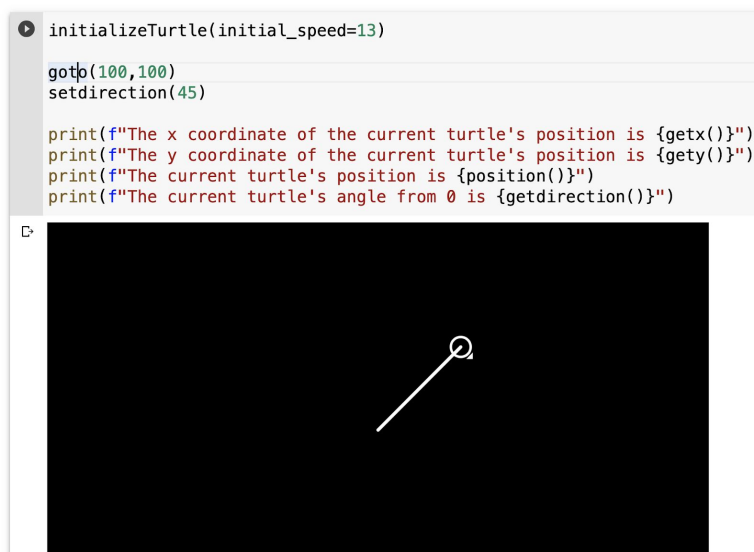
- Return the turtle's location as (x,y) coordinate

getdirection()

- Return the turtle's direction in degrees

distance(x\_2,y\_2)

- Parameters: x\_2, y\_2 - a number (integer or float)
- Return the distance from turtle's current position to given coordinates



# Pen Control

## DrawingState

pendown()

- Pull the pen down - drawing when moving

penup()

- Pull the pen up - no drawing when moving

getwidth()

- Returns the pen's width

width(width)

- Parameters: width - a number (integer)
- Changes the pen's width

isdown()

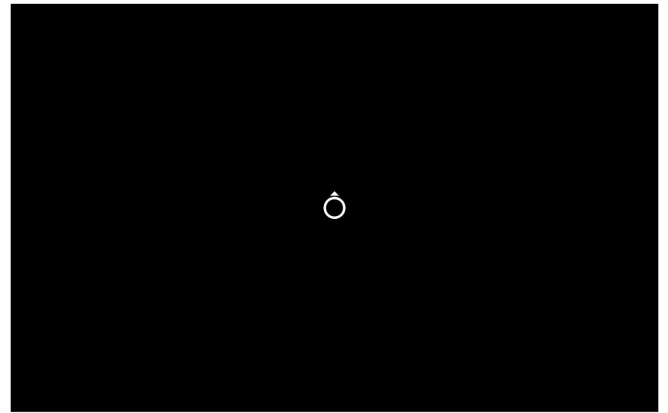
- Return's True if pen is down, False if pen is up.

```
initializeTurtle(initial_speed=13)

penup()
print(isdown())
pendown()
print(isdown())

width(30)
print(f"The pen's width is {getwidth()}")
```

↳



```
False
True
The pen's width is 30
```

## Color Control

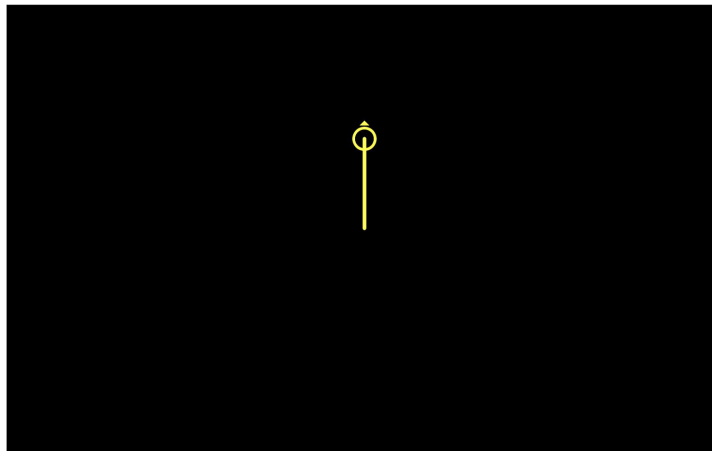
color(color)

- Parameters: string
- Changes the pen's color to one in the VALID\_COLORS list

```
initializeTurtle(initial_speed=13)

color("yellow")
forward(100)
getcolor()
```

↳



```
'yellow'
```

color\_rgb(r,g,b,a=1)

- Parameters: r - number (integer or float), g - number (integer or float), b - number (integer or float), a - number (integer or float)
- Changes the pen's color given a set of RGB values. Alpha (opacity) is default set to 1.

initializeColors(numColors=256, colormap="rainbow")

- Parameters: numColors - number (integer or float), colormap - string
- Instantiates a colormap object, which serves as a color palette. The numColors provides the number of RGB quantization levels. This sets a restriction to the amount of colors possible from the colormap. numColors = 100? Then 99 possible color values are available. numColors = 55 --> 54 possible colors. It provides a sort of limitation by dividing the colormap into chunks. The colormap comes from matplotlib. Figures to see the different colormaps available can be seen at this link: <https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html>. Choosing a colormap and setting limits to it, it instantiates a colormap object, serving as a color palette.

setcolor(n=0)

- Parameters: n - number (integer or float)
- Sets the pen color as the nth color in the color palette. initializeColors must be called beforehand. From the figures in the matplotlib, n = 0 starts on the left side, and moves to the right side, up to n = 254.

getcolor()

- Returns the pen's color as either string or rgb values

## More drawing control

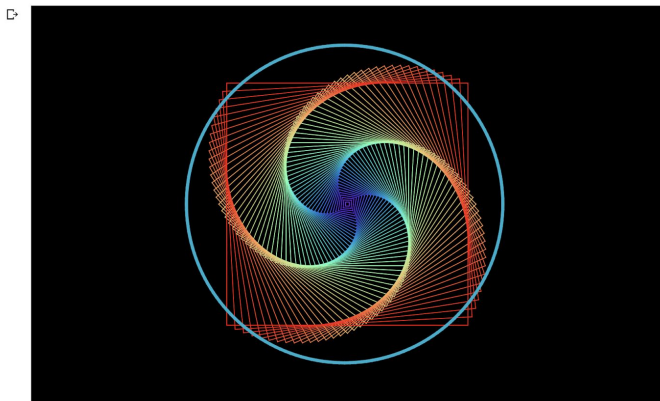
new\_window()

- Stops turtle and creates a new display (canvas) to work off of

```
initializeTurtle(initial_speed=13)
hideturtle()
width(1)
# set a color palette split into 60 chunks/colors with the rainbow colormap
initializeColors(61,"rainbow")
# cycle through each chunk/color and draw a square with each
for i in range(61):
    setcolor(i)
    n = 5 + i * 5
    square(n)
    right(3)

color_rgb(18,174,207) # can also choose color with rgb values
width(4)
circle(200)

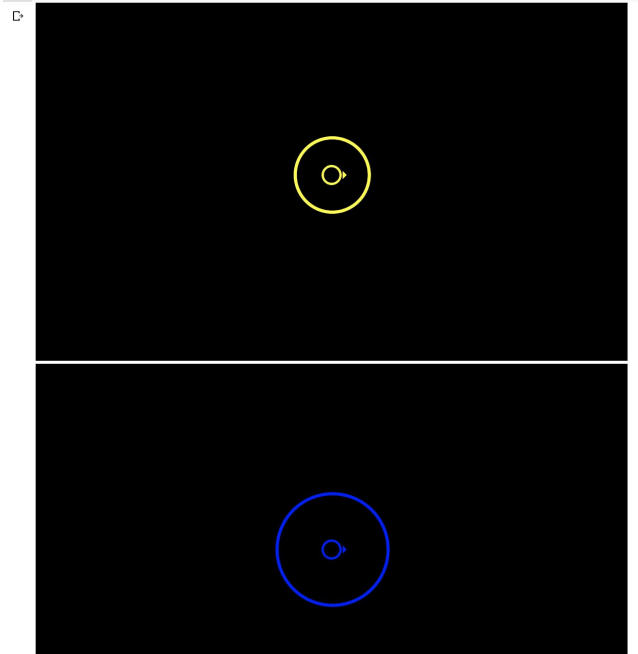
getcolor()
```



'rgb(18,174,207)'

```
initializeTurtle(initial_speed=13)

color("yellow")
circle(50)
new_window()
speed(13)
color("blue")
circle(75)
```



## Turtle's state

### Visibility

hideturtle()

ht()

- Make the turtle invisible. Good idea to do this for complex drawings. Hiding speeds up drawing

showturtle()

st()

- Make the turtle visible. Turtle is shown by default.

isvisible()

- Return True if the Turtle is shown. False if it's hidden.

```
initializeTurtle(initial_speed=13)
hideturtle()
forward(100)
print(isvisible())
```



False

## Window Control

bgcolor(color)

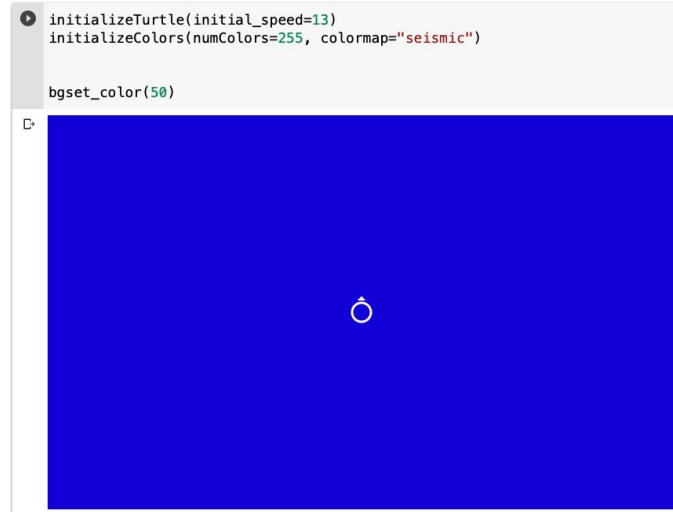
- Parameters: color - string
- Change the background color of the drawing area. Valid colors defined at VALID\_COLORS

bgcolor\_rgb(r,g,b,a=1)

- Parameters: r - number (integer or float), g - number (integer or float), b - number (integer or float), a - number (integer or float)
- Changes the background color given a set of RGB values. Alpha (opacity) is default set to 1.

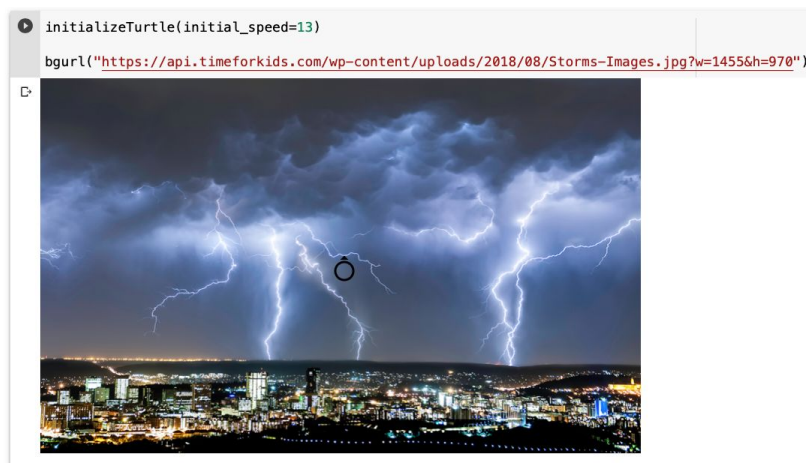
bgset\_color(n=0)

- Parameters: n - number (integer or float)
- Sets the background color as the nth color in the color palette. initializeColors must be called beforehand. From the figures in the matplotlib, n = 0 starts on the left side, and moves to the right side, up to n = 254.



`bgurl(url)`

- Parameters: url - string
- Using a picture's image address, can set it as the background picture. Resize the picture to fit in the window. Pen color changed to black. If no picture is found, background color turns white.



## Settings and Special methods

`window_width()`

- Returns the window's width

`window_height()`

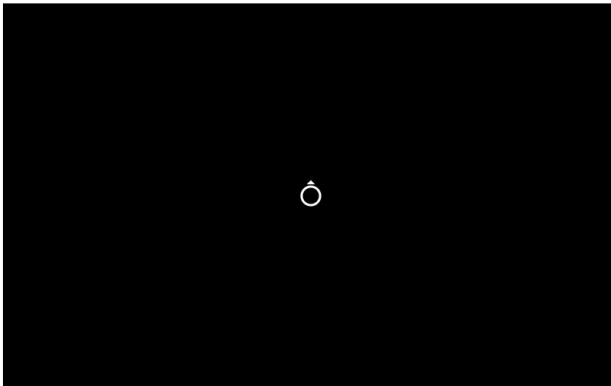
- Returns the window's height

```

initializeTurtle(initial_speed=13)

print(f"The window's height is {window_height()} pixels")
print(f"The window's width is {window_width()} pixels")

```



The window's height is 500 pixels  
The window's width is 800 pixels

## Animation Control

`delay(delay_time)`

- Parameters: `delay_time` - a number (integer or float)
- Delays the next execution by `delay_time` amount of seconds.

### Backend

`initializeTurtle(initial_speed=DEFAULT_SPEED, initial_window_size=DEFAULT_WINDOW_SIZE)`

- Parameters: `initial_speed` - an integer in range [1,13] inclusive, `initial_window_size` - tuple of 2 numbers (integers or numbers)
- Constructs the display for turtle. Can change window size here.

## Important Global Variables

`VALID_COLORS` = pre-selected basic colors that can be used for pen or background color

- ('white', 'yellow', 'orange', 'red', 'green', 'blue', 'purple', 'grey', 'black')

`COLOR_MAP` = the current color maps allowed in the program. Full list of possible color maps can be found on matplotlib.org at: <https://matplotlib.org/3.1.0/tutorials/colors/colormaps.html>

- Perceptually Uniform Sequential
  - 'viridis', 'plasma', 'inferno', 'magma', 'cividis',
- Sequential
  - 'Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds', 'YlOrBr', 'YlOrRd', 'OrRd', 'PuRd', 'RdPu', 'BuPu', 'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn'
- Sequential (2)
  - 'binary', 'gist\_yarg', 'gist\_gray', 'gray', 'bone', 'pink', 'spring', 'summer', 'autumn', 'winter', 'cool', 'Wistia', 'hot', 'afmhot', 'gist\_heat', 'copper'
- Diverging
  - 'PiYG', 'PRGn', 'BrBG', 'PuOr', 'RdGy', 'RdBu', 'RdYlBu', 'RdYlGn', 'Spectral', 'coolwarm', 'bwr', 'seismic'

- Cyclic
  - 'twilight', 'twilight\_shifted', 'hsv'
- Qualitative
  - 'Pastel1', 'Pastel2', 'Paired', 'Accent', 'Dark2', 'Set1', 'Set2', 'Set3', 'tab10', 'tab20', 'tab20b', 'tab20c'
- Miscellaneous
  - 'flag', 'prism', 'ocean', 'gist\_earth', 'terrain', 'gist\_stern', 'gnuplot', 'gnuplot2', 'CMRmap', 'cubehelix', 'brg', 'gist\_rainbow', 'rainbow', 'jet', 'nipy\_spectral', 'gist\_ncar'

`drawing_window = None` as default to make sure that user initializes turtle before doing anything else

## Helpers

These functions are not callable in Colab, but are very important and powerful for behind-the-scenes work. Anybody interested in editing the Turtle.py code or working off of it, will find these couple helper functions particularly useful. These can be found right before the Turtle Motion functions.

`_moveToNewPosition(new_pos)`

- Parameters: `new_pos` - a tuple of 2 numbers (integers or floats)
- Helper function for managing any kind of move to a given '`new_pos`' and draw lines if pen is down

`_updateDrawing()`:

- Helper functions for updating the screen using the latest positions/angles/lines etc.

`_convertx(input_x)`

- Parameters: `input_x` - a number (integer or float)
- Turns the x from the user that is in an x,y (origin (0,0) at center of display) coordinate system, to the x in the coordinate system used by the backend HTML display.

`_converty(input_y)`

- Parameters: `input_y` - a number (integer or float)
- Turns the y from the user that is in an x,y (origin (0,0) at center of display) coordinate system, to the y in the coordinate system used by the backend HTML display.