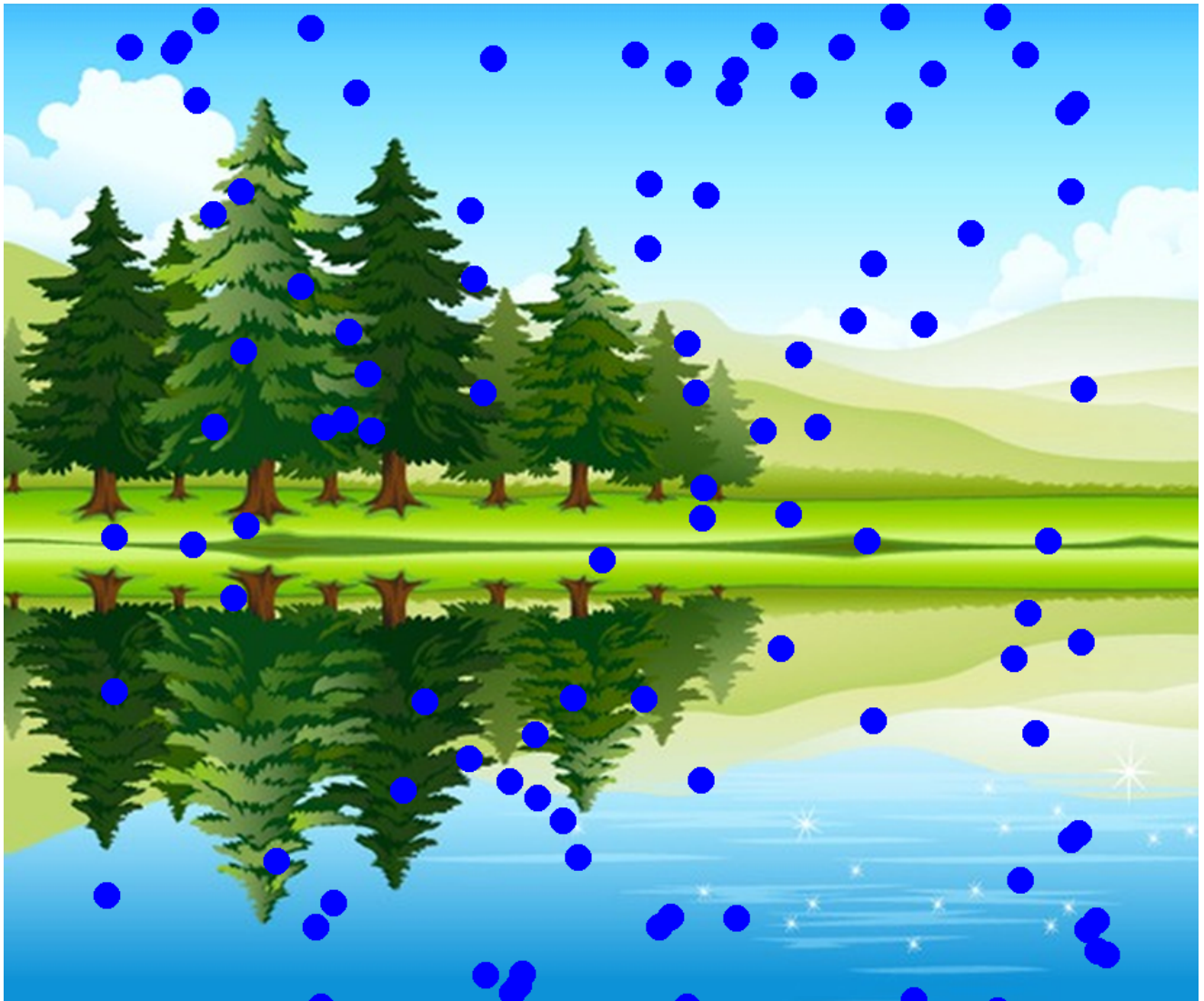


# Python Turtle Raindrops

Difficulty: 🌶️🌶️🌶️🌶️🌶️

Language: Python

Requires: Laptop with Python, Thonny or Mu



We'll use python and turtle to make it rain. Once you've got it raining, have a go at making other things move around on the screen.

The drops could be snow, stars, faces or footballs.

We'll use some elements you've seen - variables, random numbers and functions.

We'll be introducing one new element - lists.

And you'll get to do some simple animation! Stuff that moves!

## Drawing a raindrop

Lets start by setting up turtle to draw fast with `speed(0)` , hide the turtle with `hideturtle` , and pull up the pen with `penup` .

```
import turtle

t = turtle.Turtle()
t.speed(0)
t.hideturtle()
t.penup()
```

Save this in a file name like `rain.py` and run it.

To draw a simple drop we can use a blue circle. Add this:



```
t.shape("circle")
t.color("blue")

t.goto(0, 0)
t.stamp()
```

`t.shape` changes the turtles shape, `t.goto` jumps to a set of coordinates.

By using `t.stamp` , we can leave behind a stamp, an image of the turtles current shape on the canvas where it stands.

## Drawing multiple drops

We are going to want to stamp a blue circle many times - so let's move the drawing code into a function:

```
t.shape("circle")
t.color("blue")

t.goto(0, 0)
t.stamp()
```

```
def draw_drop(x, y):
    t.goto(x, y)
    t.stamp()

draw_drop(0, 0)
draw_drop(30, -40)
draw_drop(50, 20)
```

`x` is how far across the screen from the left, `y` is how far up the screen from the bottom. There is a negative number there. This is because `0, 0` is the middle of screen - so to go further down, or left, we need to subtract from `0` to get there. When you run this, it should draw 3 raindrops.



## More rain

There are many raindrops in rain. Let's use a list to hold them:

```
draw_drop(0, 0)
draw_drop(30, -40)
draw_drop(50, 20)

drops = [[0, 0], [30, -40], [50, 20]]

for drop in drops:
    draw_drop(drop[0], drop[1])
```

Drops is a list of (x,y) pairs - each a small list too. When we draw this - `x` is `drop[0]` and `y` is `drop[1]`. This should show the same 3 drops as before, but you can change the list to add more drops.

## Random rain

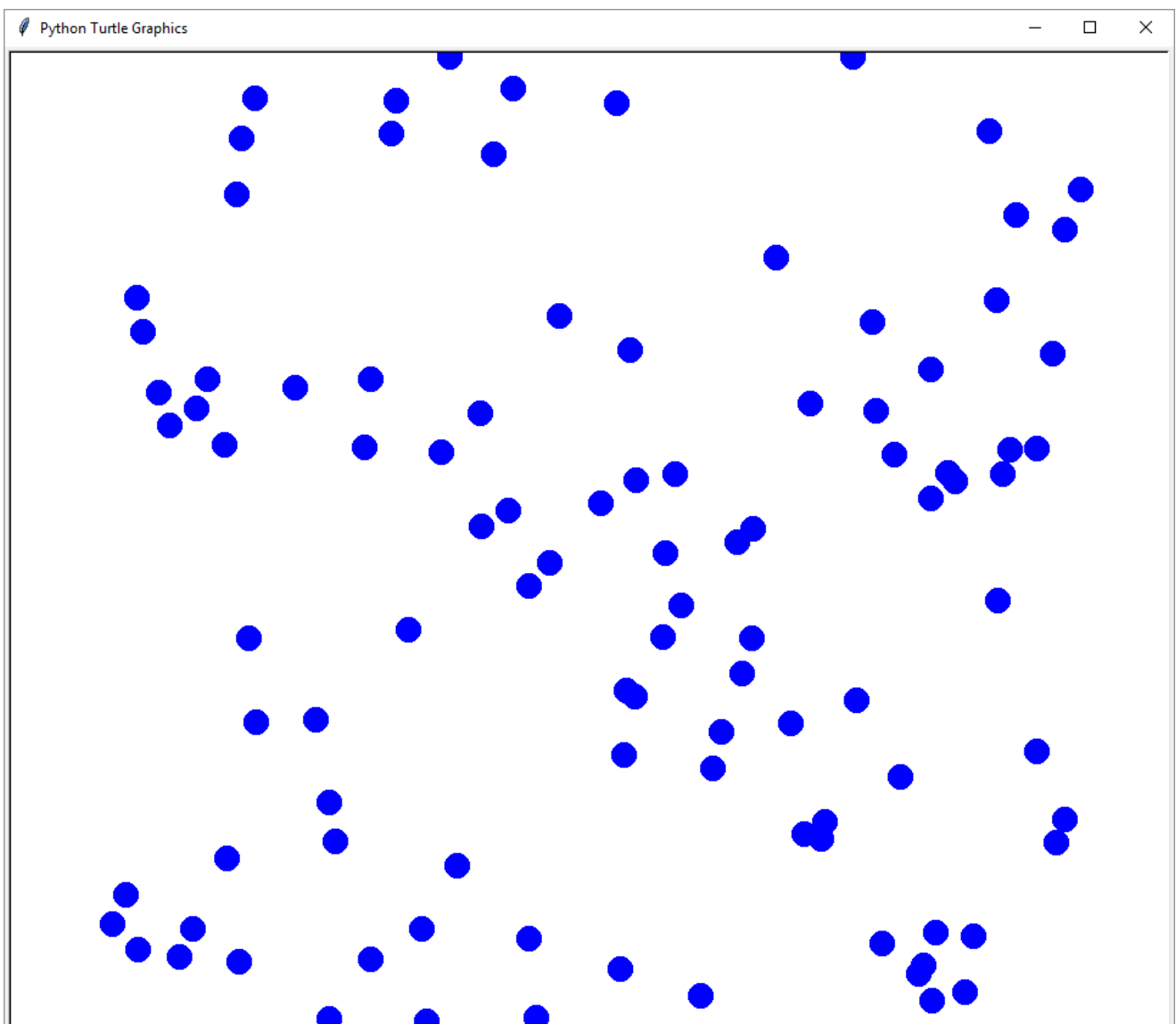
Now we can make the list bigger. Lets add 100 raindrops using `random` to scatter them around the screen. First we need to import random at the top of our code:

```
import turtle
import random
```

Then we replace our fixed list with an empty list, and fill it with random drop positions:

```
drops = [[0, 0], [30, 40], [50, 20]]
drops = []
for n in range(100):
    drop = [random.randint(-400, 400), random.randint(-400,
400) ]
    drops.append(drop)

for drop in drops:
    draw_drop(drop[0], drop[1])
```



Each time you run it - you'll get different drops!

## Checkpoint

Your code at this point should look like this:

```
import turtle
import random

t = turtle.Turtle()
t.speed(0)
t.hideturtle()
t.penup()
t.shape("circle")
t.color("blue")

def draw_drop(x, y):
    t.goto(x, y)
    t.stamp()

drops = []
for n in range(100):
    drop = [random.randint(-400, 400), random.randint(-400, 400)]
    drops.append(drop)

for drop in drops:
    draw_drop(drop[0], drop[1])
```

## Preparing to animate

You may have noticed that drawing the drops was a bit slow - one drop at a time. If we are going to animate this, we need to be able to draw a lot faster. Add the bold line near the top of the file.

Note that this should be `turtle` and not `t`.

```
import turtle
import random

turtle.tracer(0, 0)
t = turtle.Turtle()
t.speed(0)
```

This tells the turtle not to animate itself, so we can animate instead.

This will be very quick, but it's actually drawn on a background/hidden screen. To actually see it you'll need to add this at the end of the code:

```

for drop in drops:
    draw_drop(drop[0], drop[1])

turtle.update()

```

This will now make the random raindrops draw much faster.

## Moving the raindrops

We can start to make these raindrops move now.

```

drops = []
for n in range(100):
    drop = [random.randint(-400, 400), random.randint(-400,
400) ]
    drops.append( drop )

for drop in drops:
    draw_drop(drop[0], drop[1])

turtle.update()

while True:
    t.clear()
    for drop in drops:
        drop[1] -= 3
        draw_drop(drop[0], drop[1])
    turtle.update()

```

Our animation is in the while loop. It clears the drawings, then moves them down by 3 (subtracting 3 from Y), and draws the drop.

After drawing all drops, we update the screen. This makes a different picture every time, which will look like the drops are moving.

You'll note all the drops fall off the screen here. You may see an "invalid command name" and a large number when you close the window, don't worry - this can be ignored for now.

## Rain from the top again

We can stop them falling off. The bottom of the screen here is -400. So if we are below that, we can put them back at the top. Add the following in the loop after we subtract 3 from the drop y:

```

for drop in drops:
    drop[1] -= 3
    if drop[1] < -400:
        drop[1] = 400
    draw_drop(drop[0], drop[1])
turtle.update()

```

## Checkpoint 2

Your code at this point should look like this:

```

import turtle
import random

turtle.tracer(0, 0)
t = turtle.Turtle()
t.speed(0)
t.hideturtle()
t.penup()
t.shape("circle")
t.color("blue")

def draw_drop(x, y):
    t.goto(x, y)
    t.stamp()

drops = []
for n in range(100):
    drop = [random.randint(-400, 400), random.randint(-400,
400) ]
    drops.append(drop)

while True:
    t.clear()
    for drop in drops:
        drop[1] -= 3
        if drop[1] < -400:
            drop[1] = 400
        draw_drop(drop[0], drop[1])
    turtle.update()

```

## Ideas to try

### Further drop parameters

You can try using a 3rd item in the lists - for speed, or raindrop size ( `t.shapesize` ). Let's try using it for speed.

When creating the drop, we can try a number between 2 and 4:

```
for n in range(100):
    drop = [random.randint(-400, 400), random.randint(-400,
400), random.randint(2,4)]
    drops.append(drop)
```

You can then use this 3rd parameter for speed instead of -3:

```
for drop in drops:
    drop[1] -= 3
    drop[1] -= drop[2]
    if drop[1] < -400:
        drop[1] = 400
```

This now moves different drops at different speeds, giving a feeling of depth.

## Try changing the rain drops

Rain drops can be snow flakes too. Try changing the shape to a triangle, or a square using turtle commands. You can change the colour, size, put the pen down and do standard turtle drawing commands, or even use GIF or PNG images.

```
t.penup()
t.shape("circle")
t.color("blue")
screen = turtle.Screen()
image = "myimage.gif"
screen.addshape(image)
t.shape(image)

def draw_drop(x, y):
```

Now when you stamp - it will be your image instead of the circles. Happy faces? Footballs? Spaceships? Stars?

## Background images

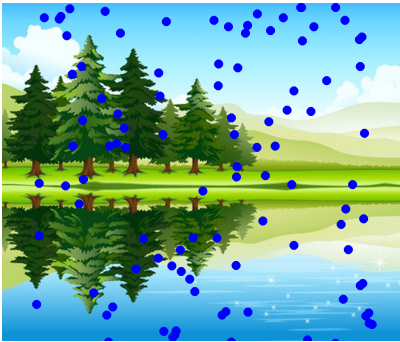
You can use `turtle.bgcolor` near the top of the code to change the background colour, or you can use a GIF or PNG as a background image:



```
import turtle
import random

screen = turtle.Screen()
screen.bgpic("lake-background.gif")

turtle.tracer(0, 0)
t = turtle.Turtle()
```



You can create a rainy, space or snowy scenes. Making the dots small with a space background. Get creative!

Stars:





```
t.shape("circle")
t.color("lightblue")
turtle.bgcolor("black")
screen = turtle.Screen()
screen.bgpic("space-background.png")
t.shapesize(0.1)
```

## Turtle Colours

This is a limited list. Look up "TK colours" for more names. You can also use three numbers for red, green and blue to mix your

own colours: `t.color((172, 38, 53))` **Sample.**

Colour Name	Sample
red	<div style="display: inline-block; width: 20px; height: 15px; background-color: red; border: 1px solid black;"></div>
blue	<div style="display: inline-block; width: 20px; height: 15px; background-color: blue; border: 1px solid black;"></div>
green	<div style="display: inline-block; width: 20px; height: 15px; background-color: green; border: 1px solid black;"></div>
yellow	<div style="display: inline-block; width: 20px; height: 15px; background-color: yellow; border: 1px solid black;"></div>

Colour Name	Sample
salmon	
orange	
black	
white	

It's worth trying other colour names and seeing what works.

## Turtle reference

Command	Effect
<code>t = turtle.Turtle()</code>	Make a turtle called t
<code>turtle.tracer(0,0)</code>	Turn off tracer animation - makes it very fast
<code>turtle.update()</code>	Make a screen update - handy when fast
<code>turtle.done()</code>	Program finished, wait for window to close
<code>t.clear()</code>	Clear everything drawn by this turtle
<code>t.speed(0)</code>	Make this turtle fast
<code>t.penup()</code>	Pull the pen up - don't draw lines
<code>t.pendown()</code>	Put the pen down - draw a line
<code>t.hideturtle()</code>	Hide the turtle - don't draw it
<code>t.goto(x, y)</code>	Jump to position x, y. 0, 0 is the middle
<code>t.stamp()</code>	Stamp the current turtle shape
<code>t.shape("shape")</code>	Change shape. Try "turtle", "circle", "square"
<code>t.shapesize(0.1)</code>	Change the size of the shape
<code>t.color("color")</code>	Change color. Try "red", "green", "blue"
<code>t.forward(100)</code>	go forward 100 pixels
<code>t.left(90)</code>	turn left 90 degrees
<code>t.right(45)</code>	turn right 45 degrees
<code>turtle.bgcolor("black")</code>	Set the screen background to "black"
<code>turtle.window_height()</code>	Get the height of the window
<code>turtle.window_width()</code>	Get the width of the window