Programming in Python

A Brief Introduction with TigerJython and Turtle-Graphics

Dr Tobias Kohn

© 2015–2019, Dr Tobias Kohn
http://jython.tobiaskohn.ch/

Get TigerJython from:
https://www.tjgroup.ch/engl/index.php
https://webtigerjython.ethz.ch/

1 Turtle-Graphics

Basics You are going to control a small turtle with a pen on the screen. When the turtle is moving, it draws a trace with its pen, creating pictures. For the purpose of this script we use «TigerJython», which is a dialect of *Python*.

Turtle graphics is just one of many extension modules that come with Python. At the beginning of your program, you need to explicitly load (import) the module for turtle graphics. In TigerJython, the module for turtle graphics is called «gturtle» (the «g» stands for «graphics»). Once it is loaded, use `makeTurtle()` (line 2) to create a new window with the turtle in it.

```
from gturtle import *
makeTurtle()

# Your program comes here
```

Make sure you type the first two lines exactly as shown here. Even the parentheses following `makeTurtle` are necessary. However, anything following a hash `#` will be ignored by Python.

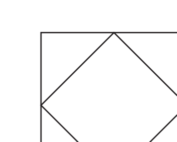
Example: Drawing a Triangle This program has the turtle draw a triangle with one right angle and two $45°$ angles. To get a $45°$ angle, the turtle has to turn by $135°$! Note how we use `sqrt(2)` to get the square root of 2.

```
1 from gturtle import *
2 makeTurtle()
3
4 forward(100 * sqrt(2))
5 right(135)
6 forward(100)
7 right(90)
8 forward(100)
```

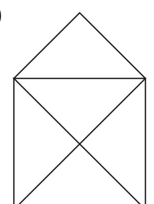
EXERCISES

1. Have your turtle draw one or both of these two shapes:

(a)



(b)



© 2019, Dr Tobias Kohn

Controlling the Turtle The turtle can move forward in a straight line, turn left or right (without moving), or draw a dot at its current position.

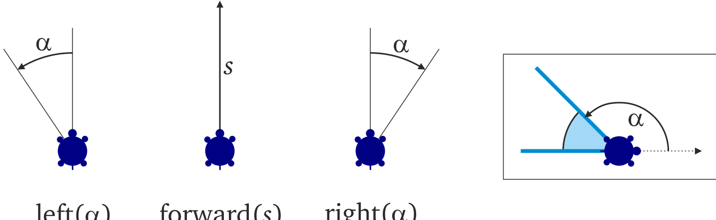


Figure 1: `forward(s)` makes the turtle move forward s pixels, `left(a)` and `right(a)` turn the turtle on its spot to the left or right, respectively. When drawing a shape, note that the angle given is the *outside angle* of the polygon (on the far right).

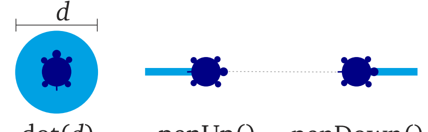


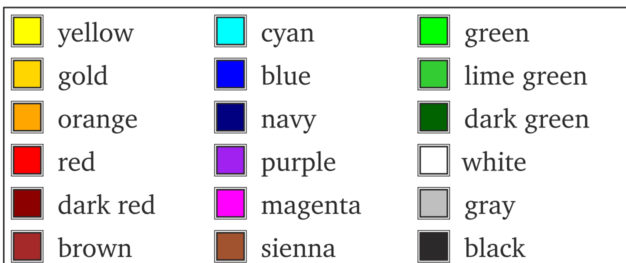
Figure 2: `dot(d)` draws a single dot with diameter d. Use `penUp()` and `penDown()` to move the turtle without drawing its trace.

Overview of Turtle Commands

`forward(s)`	Move s pixels forward.
`left(w)`	Turn w degrees to the left.
`right(w)`	Turn w degrees to the right.
`dot(d)`	Draw a dot with diameter d.
`setPenColor("c")`	Use c for the pen's colour.
`penWidth(b)`	Set the pen's width to w.
`penUp()`	Raise the pen to not draw anymore.
`penDown()`	Lower the pen to continue drawing.
`clear("c")`	Fill the entire window with the colour c.
`setFillColor("c")`	Set the colour to fill shapes.
`startPath()`	Start drawing a shape to fill.
`fillPath()`	Fill the shape.
`hideTurtle()`	Make the turtle invisible and fast.
`showTurtle()`	Make the turtle visible again.
`speed(-1)`	Set the turtle to maximum speed.
`setPos(x, y)`	Place the turtle at (x, y).
`moveTo(x, y)`	Draw a line to the coordinates (x, y).
`label("text")`	Write the given text at the current position.

© 2019, Dr Tobias Kohn

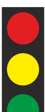
Colour and Linewidth You can choose both the colour, and the width of the pen, with which the turtle is drawing its trace. To choose the colour, use `setPenColor("colour")` (do not forget the quotation marks around the colour) and `setPenWidth(width)` (numbers have no quotation marks), e. g. `setPenColour("red")` and `setPenWidth(3)`.



yellow	cyan	green
gold	blue	lime green
orange	navy	dark green
red	purple	white
dark red	magenta	gray
brown	sienna	black

Figure 3: This is a small selection of colours available for `setPenColor("colour")`

Example: Trafic Lights The turtle draws trafic lights. The black box is actually just a broad line.



```
from gturtle import *
makeTurtle()

setPenColor("black")
penWidth(40)
forward(80)

left(180)
penUp()

setPenColor("red")
dot(30)
forward(40)

setPenColor("yellow")
dot(30)
forward(40)

setPenColor("green")
dot(30)
forward(40)
```

© 2019, Dr Tobias Kohn

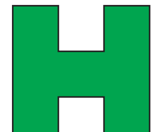
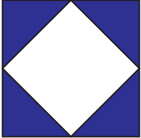
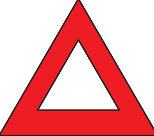
Filling Areas With Colour In order to fill an area, you can draw so many lines that they completely cover the area. Luckily, if you give it a shape, the turtle already knows how to fill it with a colour, saving us a lot of work. You have to use `startPath()` to tell where you start drawing the shape to fill. With `fillPath()` you then make the turtle fill this shape with colour.

Example: A Square in Red and White Between the (marked) commands `startPath()` and `fillPath()`, the turtle covers the entire area with colour. After `fillPath()`, it goes back to just drawing the traces.

```
from gturtle import *
makeTurtle()

setPenColor("red")
setFillColor("red")

startPath()    # <-
right(45)
forward(100)
left(90)
forward(100)
fillPath()     # <-

left(90)
forward(100)
left(90)
forward(100)
```

EXERCISES

2. When drawing one of the following shapes, make sure you only cover the area that is supposed to be coloured.

© 2019, Dr Tobias Kohn

2 Repetitions

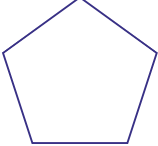
Loops If you want to draw a square, you have to repeat the same two lines four times. However, instead of repeating yourself four times as on the left, you can ask Python to do the repeating. `repeat 4:` tells Python to repeat all subsequent lines that are indented four times.

```
from gturtle import *
makeTurtle()
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
forward(100)
left(90)
```

```
from gturtle import *
makeTurtle()

repeat 4:
    forward(100)
    left(90)
```

Make sure to indent all lines that should be repeated by the same number of spaces.

Note: `repeat` is a feature of the TigerJython-dialect and not available in other versions of Python.



Example: Draw a Pentagon This program has the turtle draw a pentagon instead of a square. The number of repetitions if now five. The last command `fillPath()` is not part of the repetition scheme. It is only executed *after* all repetitions above have completed.

```
1  from gturtle import *
2  makeTurtle()
3
4  right(90)
5  startPath()
6  setFillColor("sky blue")
7  repeat 5:
8      forward(100)
9      left(72)
10 fillPath()
```

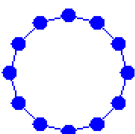
The $360°$-Rule Whenever the turtle draws a closed shape, it will have turned around by a full circle, i. e. $360°$. For a pentagon, the turtle is turning five times, and hence $360°/5 = 72°$ each time.

© 2019, Dr Tobias Kohn

Python can do the division «360/5» on its own:

```
repeat 5:
    forward(100)
    left(360 / 5)
```

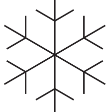
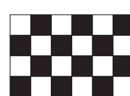
Example: The Necklace With twelve vertices, this necklace looks almost round. Indeed, a polygon with 36 or more vertices can usually not be distiniguished from a circle.



```
1 from gturtle import *
2 makeTurtle()
3
4 left(75)
5 repeat 12:
6     dot(10)
7     forward(20)
8     right(30)
9 hideTurtle()
```

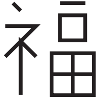
EXERCISES

3. Have the turtle draw the snowflake as seen below on the left.

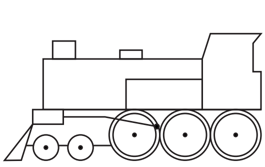
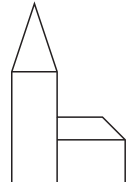
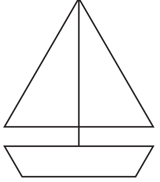
 

4. Have the turtle draw a small chess board as seen above in the middle, or the stairs above on the right.

5. Have the turtle draw the chinese character for «good fortune» as seen on the right.



6. Have the turtle draw a sailboat, or any of the other two shapes in the picture below. In order to draw the circles of the locomotive, you can either use `dot(d)`, or draw a regular polygon with 36 vertices.

© 2019, Dr Tobias Kohn

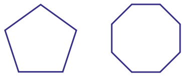
3 Make Your Own Language

New Commands Programming is all about extending the programming language and adding new commands. In Python, you use `def` to define a new command, which you can then use as often as you want. A command for drawing an equilateral triangle might look like this:

```
from gturtle import *
makeTurtle()

def triangle():
    repeat 3:
        forward(100)
        left(120)

triangle()   # Actually draw the triangle
right(90)
triangle()   # And yet another one
```

Parameters The true power of defining new commands comes using the same program code with different values. After the command's name, you can specify one or more parameters. When you actually call (use) the command, you fill in the numbers for these parameters.

```
from gturtle import *
makeTurtle()

def square(side):
    repeat 4:
        forward(side)
        right(90)

square(80)           # side = 80
setPenColor("red")
square(60)           # side = 60
```



Example: Polygons This ploygon command has two parameters: n and $side$.

```
from gturtle import *
makeTurtle()

def polygon(n, side):
    repeat n:
        forward(side)
        left(360 / n)

polygon(5, 120)    # n = 5, side = 120
```

© 2019, Dr Tobias Kohn

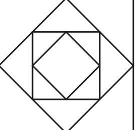
Changing Parameter Values Inside your command, you can change the value of a parameter as often and whenever you like. If t is your parameter, use `t += 1` to increase its value by one, `t -= 1` to decrease it by one, or even `t = 123` to set it to a specific value altogether.

```
from gturtle import *

def spiral(s):
    repeat 30:
        forward(s)
        s += 3
        left(60)

makeTurtle()
spiral(2)
```

Example: A Heart By slightly changing the angle instead of the step length, we get a heart shape.

```
from gturtle import *

def heart_shape_L(angle):
    repeat 16:
        forward(10)
        left(angle)
        angle -= 2

def heart_shape_R(angle):
    repeat 16:
        forward(10)
        right(angle)
        angle -= 2

makeTurtle()
setPenWidth(2)
heart_shape_L(30)
home()
heart_shape_R(30)
```

EXERCISES



7. Have the turtle draw the figure with nested squares. You can double the value of a parameter s by `s *= 2`, or use `s /= 2` to cut it in half.

© 2019, Dr Tobias Kohn

4 Animation

An animation consists basically of three steps. First, you draw a picture (the frame) onto the screen. Then you wait for a few milliseconds. And finally, you update the elements in the picture to their next positions. In Python code:

```
repeat 1000:
    draw_frame()
    delay(40)
    update_frame()
```

Example: Turning Square To make a shape rotate, all we have to do is turn the turtle a little bit each time. The updating of the frame is thus very simple here.

```
1  from gturtle import *
2
3  def square(s):
4      repeat 4:
5          forward(s)
6          right(90)
7
8  def draw_frame():
9      clear()
10     forward(150)
11     right(135)
12     square(150 * sqrt(2))
13     left(135)
14     back(150)
15
16 makeTurtle()
17 hideTurtle()
18 setPenWidth(3)
19 repeat 1000:
20     draw_frame()
21     delay(40)
22     right(2)   # update frame
```

EXERCISES

8. Code a program that displays a simple watch. It has a single hand for the seconds going around.

© 2019, Dr Tobias Kohn

Example: Random Walk Image the turtle holding a red latern that glows in the dark. Since the turtle cannot see in the dark, it changes its directly randomly for each new frame. We keep the angle between $-45°$ and $+45°$ to avoid sharp turns.

```
from gturtle import *
from random import randint

def draw_frame():
    clear("black")
    dot(15)
def update_frame():
    angle = randint(-45, 45)
    left(angle)
    forward(4)
makeTurtle()
hideTurtle()
setPenColor("red")
penUp()
repeat 10000:
    key = getKeyCode()
    if key != 0:
        break
    draw_frame()
    delay(20)
    update_frame()
dispose()    # close window
```

In each repetition of the loop, we check if a key is pressed. If the key code is not zero (a key is pressed), then we break out of the loop. You can use the keys to completely control the moving dot, and thus eventually create a computer game.

```
repeat 1000:
    clear("navy")
    dot(20)
    delay(20)
    key = getKeyCode()
    if key == 37:    # LEFT
        left(90)
        forward(5)
        right(90)
    if key == 39:    # RIGHT
        right(90)
        forward(5)
        left(90)
    if key == 38:    # UP
        forward(5)
    if key == 40:    # DOWN
        back(5)
    if key == 32:    # SPACE
        break
```

© 2019, Dr Tobias Kohn

5 Using Coordinates

Instead of telling the turtle how to turn at each point, you can also specify a shape through the coordinates of its vertices. Python then goes through our list and moves the turtle to each vertex (x, y) in turn.

```
from gturtle import *

SHAPE = [(80, 90), (4, 60), (50, -10)]

makeTurtle()
setPos(50, -10)
for (x, y) in SHAPE:
    # for each vertex (x, y) do the following:
    moveTo(x, y)
```

Example: Plotting the Graph of a Function This program plots the graph of the function $f(x)$ given by:

$$f(x) = \frac{x \cdot (x - 75) \cdot (x + 75)}{3000}$$

First, we define the function f, which returns a value computed from its parameter x. When drawing the actual function, we calculate the y-coordinate for each x-coordinate by using the function $f(x)$.

```
from gturtle import *

def f(x):
    return x * (x - 75) * (x + 75) / 3000

makeTurtle()
setPenWidth(2)
x = -100
y = f(-100)
setPos(x, y)
repeat 200:
    x += 1
    y = f(x)
    moveTo(x, y)
```

Example: Ball In the Box By using coordinates, we can start to create more elaborate animations and simulations. In this example, we have the ball between two (invisible) walls on the left and on the right, from where it is thrown back.

The global variables X, Y are the current position of our ball. S_X and S_Y give its speed. Since we want to use these global variables inside

© 2019, Dr Tobias Kohn

our commands, we say so through `global X, Y` in the first line of the respective function.

Whenever the x-coordinate gets larger than 200, we reverse the speed S_X in this direction. This looks like a reflection, and you can do the same for top and bottom with y-coordinates.

```
from gturtle import *

X, Y = -20, 90
S_X, S_Y = 4.5, -0.75

def draw_frame():
    global X, Y
    clear("black")
    setPos(X, Y)
    dot(20)

def update_frame():
    global X, Y, S_X, S_Y
    X += S_X
    Y += S_Y
    if X > 200:
        S_X = -S_X
    if X < -200:
        S_X = -S_X

makeTurtle()
hideTurtle()
penUp()
setPenColor("red")
repeat 500:
    draw_frame()
    delay(10)
    update_frame()
```

EXERCISES

9. Complete the program above by having the ball also being reflected at the floor below and the ceiling above. You may even want to draw the entire box as part of the `draw_frame()` command.

10. At the moment, `clear("black")` clears the entire screen with black. However, if we use a colour that is slightly translucent, the old image can still shine through. Use `makeColor()` to make a color translucent: `clear(makeColor("black", 0.25))`.

© 2019, Dr Tobias Kohn

Many Objects *Objects* act like containers for variables. In this program, we create an object for each ball, which then holds the position and speed of the ball. This way, you can have as many balls as you like, each with its own position, speed, colour, or even size.

```
from gturtle import *

class ball:
    def __init__(self, x, y, s_x, s_y, color):
        self.x = x
        self.y = y
        self.s_x = s_x
        self.s_y = s_y
        self.color = color

    def draw(self):
        setPos(self.x, self.y)
        setPenColor(self.color)
        dot(10)

    def update(self):
        self.x += self.s_x
        self.y += self.s_y

balls = [ ball(30, 40, 3, 6, "red"),
          ball(-50, 60, 5, -1, "gold") ]

def draw_frame():
    global balls
    clear("black")
    for ball in balls:
        ball.draw()

def update_frame():
    global balls
    for ball in balls:
        ball.update()

makeTurtle()
hideTurtle()
repeat 100:
    draw_frame()
    delay(40)
    update_frame()
    if getKeyCode() == 32:
        balls.append(ball(-40, 50, 2, -3, "cyan"))
```

Whenever you hit the space bar, a new ball is added, thanks to the last two lines. If you use `randint()` (see page 12), you can add the balls at random positions.

© 2019, Dr Tobias Kohn

6 Colour Gradients and Replacing Colours

Making Colours Use the function `makeColor(r, g, b)` to create your own colour by telling the computer, how much *red*, *green*, and *blue* your colour should contain. Each value must be given as a number in the range 0.0 – 1.0. For instance, `makeColor(1.0, 0.7, 0)` gives you a nice orange.

Depending on your system, you will have to replace the `"black"` *in line 16 by* `"#000000"` *or something similar.*

Replacing Colours In this example, we first draw a black star. Then we scan it line by line and replace all black pixels by another colour to give the star a more interesting look.

```
1  from gturtle import *
2
3  def star(s):
4      left(18)
5      startPath()
6      repeat 5:
7          forward(s)
8          right(72)
9          forward(s)
10         left(144)
11     fillPath()
12     right(18)
13
14 def replace_line(s):
15     repeat s:
16         if getPixelColorStr() == "black":
17             dot(1)
18         forward(1)
19
20 makeTurtle()
21 hideTurtle()
22 setPos(100, 0)
23 star(40)
24 setPos(0, 0)
25 penUp()
26 i = 0
27 repeat 100:
28     c = makeColor(1.0, 1.0 - i, 0.0)
29     setPenColor(c)
30     i += 1 / 100
31     right(90)
32     replace_line(150)
33     back(150)
34     left(90)
35     forward(1)
```

© 2019, Dr Tobias Kohn

7 The Turtle Finds Its Way Through A Maze

The Basic Grid In this section, we no longer work with turtle graphics. Instead, the turtle lives in a grid with blocks and walls, which we can use to form to be a veritable maze. When creating the grid, you can specify its dimensions.

```
from tjgrids import *
makeTurtle(20, 15)
```

Controlling the Turtle The commands for moving the turtle are very similar to what we had before, but turning always happens in $90°$ and no longer requires an angle.

`forward(s)`	Move s cells forward.
`left()`	Turn $90°$ degrees to the left.
`right()`	Turn $90°$ degrees to the right.
`setPos(x, y)`	Place the turtle at (x, y).
`setCell(n)`	Set a cell's value to n.
`getCell()`	Return a cell's current value.
`canGoForward()`	Checks if the turtle can move forward.
`canGoRight()`	Checks if the turtle can move to the right.
`canGoLeft()`	Checks if the turtle can move to the left.

Each cell can hold a small number, which is represented through its colour. «0» is an empty, white cell, «1» is a wall, «2» red, «3» yellow, «4» green, «5» blue, and «6» black. Use `setCell()` to set the value of the cell the turtle is currently in.

```
from tjgrids import *
makeTurtle(20, 20)

setPos(10, 10)
setCell(2)
forward()
setCell(3)
right()
forward()
setCell(2)
right()
forward()
setCell(3)
```

© 2019, Dr Tobias Kohn

You may also want the turtle to react to a cell's colour. For instance, the turtle could turn left on yellow cells, and right on red cells as in the following program.

```
from tjgrids import *
makeTurtle(20, 20)

setPos(10, 10)
repeat 30:
    if getCell() == 2:
        setCell(3)
        right()
    elif getCell() == 3:
        setCell(2)
        left()
    else:
        setCell(2)
        left()
        left()
    forward(1)
```

Finding a Way through a Maze In this program, the turtle is searching for a red cell. Whenever it can, it will turn and go to the right. Otherwise it tries to move forward, or turn to the left. This simple rule will, indeed, get you out of many mazes – although it might not be very efficient.

```
from tjgrids import *
makeGrid("labyrinth1")

while getCell() != 2:
    if canGoRight():
        right()
        forward(1)
    elif canGoForward():
        forward(1)
    else:
        left()
```

EXERCISES

11. How could you trap a turtle that uses the above strategy to find its way out of a maze?

12. Find a different strategy to get out of the maze. You might want to use the colours to mark cells where the turtle has already been.

© 2019, Dr Tobias Kohn