nclab

Learning for Industry 4.0

Visual Introduction to

Python

with Turtle Tina

Visual Introduction to Python with Turtle Tina

Visual Introduction to Python with Turtle Tina

Dr. Pavel Solin

Revision October 12, 2018

About the Author

Dr. Pavel Solin is Professor of Applied and Computational Mathematics at the University of Nevada, Reno. He loves computers, computing, and open source software. He works on advanced computer simulation projects and is the author of several books and many scientific articles in international journals. He also wrote the complete Turtle functionality in Python.

Acknowledgment

We would like to thank many teachers for class-testing the course, and for providing useful feedback that helps us improve the textbook, the self-paced course, and the Turtle language itself.

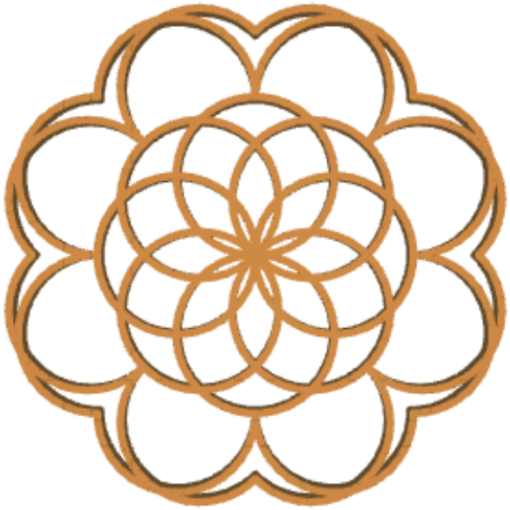
Graphics Design:

TR-Design `http://tr-design.cz`

Preface

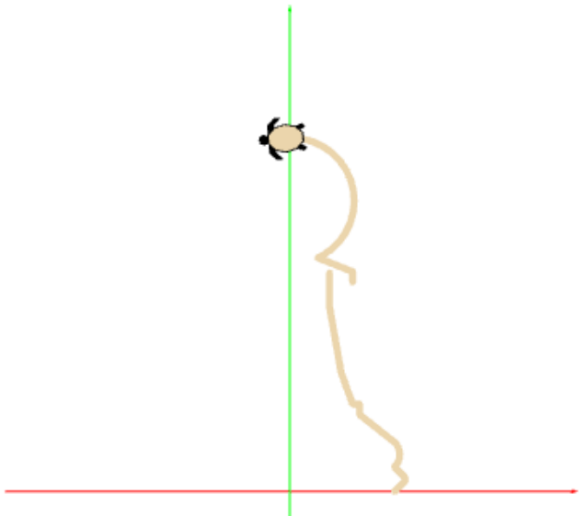
Turtle Tina[1] is an intermediate computer programming course that teaches basic programming logic and the syntax of the Python programming language. This course is a step up from Karel Coding that uses both simple logic and simple syntax, but it is simpler than the full Python programming course with full logic and full syntax. Using loops, nested loops, functions, and other programming concepts, the Turtle can draw beautiful patterns

which in turn can be extruded for 3D printing:

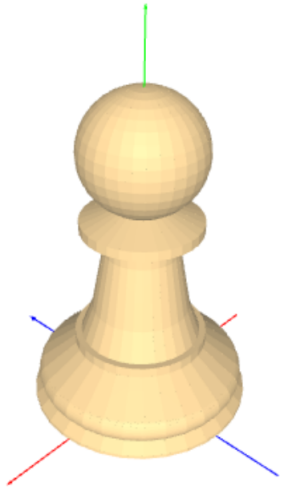


[1] This document was prepared using the LaTeX module in NCLab

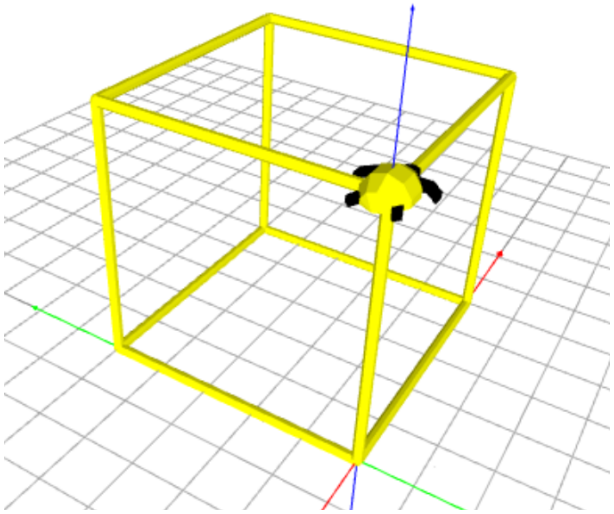
The Turtle can also draw contours in the XY plane



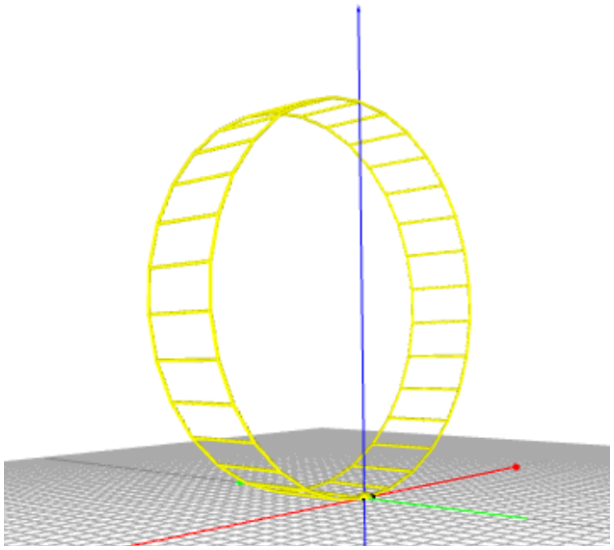
and revolve them about the Y axis, creating rotational solids, shells, and surfaces:



In the 3D mode, the Turtle can swim in all three spatial directions and create awesome wireframe models. Of course they can be 3D-printed as well.



This is super fun!



Besides teaching you Python coding, the Turtle will allow you to develop strong spatial reasoning skills, unleash your creativity, build awesome 3D-printable designs, and participate in weekly NCLab competitions.

Good luck!

Pavel

Contents

1 Introduction

The Turtle Programming module in NCLab can be accessed via Free Apps ⟶ Programming, and the self-paced course Turtle Tina via Courses. This document does not replace the course - it is meant to serve as a reference material for those who already know Turtle Programming.

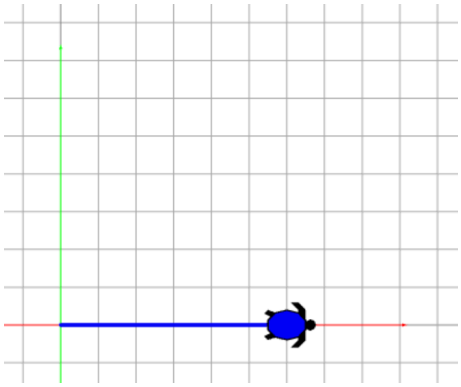
The NCLab Turtle is a descendant of Logo - an educational programming language designed in 1967 by Daniel G. Bobrow, Wally Feurzeig, Seymour Papert and Cynthia Solomon. Due to its simplicity, many different implementations can be found on the web. The NCLab version is mostly compatible with the other ones out there, but it is unique for its ability to produce 3D pendants and even rotational 3D objects, and export them as STL files for 3D printing. Both the Turtle program files and the resulting STL files can be saved in your NCLab account, and you can publish them on the web.

2 Drawing the first line

The following program creates a new NCLabTurtle named `t` at position (0, 0), moves it 60 steps forward, and shows the line that it has drawn. To run the program, press the green Play button.

```
1 t = NCLabTurtle()
2 t.go(60)
3 t.show()
```

This is the corresponding output:



3 Default settings

The default position of the Turtle is at the origin (0, 0) and its default angle is 0 (facing East). The default line width is 1, default line height is 0 (the trace is 2D in the XY plane) and default line color is blue. The red and green lines with arrows are the axes of the coordinate system (red = x, green = y). The size of the grid square is 10×10 steps. This size can be customized in Settings. Both the axes and the grid can be turned off in the menu. The turtle's pen is down by default.

4 Basic Turtle commands

4.1 NCLabTurtle()

The line

```
t = NCLabTurtle()
```

creates a new NCLabTurtle named `t` at position (0, 0). Defaults from Section 3 apply. The line

```
t = NCLabTurtle(40, 30)
```

creates a new NCLabTurtle named `t` at position (40, 30).

4.2 show()

The line

```
t.show()
```

displays the trace of turtle `t` as well as the turtle itself if visible (it is visible by default).

4.3 go(), forward(), fd()

The line

```
t.go(30)
```

will move turtle `t` 30 steps forward. When the pen is down, the turtle will draw a line.

4.4 left(), lt()

The line

```
t.left(45)
```

will turn turtle `t` left 45 degrees.

4.5 right(), rt()

The line

```
t.right(90)
```

will turn turtle `t` right 90 degrees.

4.6 penup(), pu()

The line

```
t.penup()
```

will lift pen of turtle `t` up. When turtle `t` moves after that, she will not be drawing a line.

4.7 pendown(), pd()

The line

```
t.pendown()
```

will put pen of turtle `t` down. When turtle `t` moves after that, she will be drawing a line. When a new turtle is created, the pen is down by default.

4.8 back(), backward(), bk()

The line

```
t.back(50)
```

will move turtle `t` back 50 steps. NOTE: The turtle is not drawing while backing.

4.9 color()

The line

```
t.color(MAGENTA)
```

will set the color of turtle `t` to magenta. There are many predefined colors like this - if you can name a color, chances are that it will be available. Use all capital letters for colors. In addition, it is possible to define custom colors as triplets of RGB values. For example, [0, 0, 0] is the same as BLACK, [180, 0, 0] is the same as RED, [255, 0, 0] is the same as LIGHTRED, etc. When a new turtle is created, its default color is BLUE.

4.10 width()

The line

```
t.width(2)
```

will set the line width of turtle `t` to 2. When a new turtle is created, its default line width is 1.

4.11 height()

The line

```
t.height(5)
```

will set the (vertical) line height of turtle `t` to 5. When a new turtle is created, its default line height is 0.

4.12 angle()

The line

```
t.angle(30)
```

will set the angle of turtle `t` to 45 degrees. When a new turtle is created, its default angle is 0 (facing East). Angle 90 faces North, angle 180 faces West, angle -90 faces South, etc.

4.13 goto()

The line

```
t.goto(50, 20)
```

will move turtle `t` to coordinates (50, 20). If pen is down, then the turtle will draw a line. The turtle's final angle will depend on where she came from.

4.14 home()

The line

```
t.home()
```

will move turtle `t` to coordinates (0, 0) and set angle to 0 degrees (facing East). The turtle is not drawing while going home.

4.15 hide(), invisible()

The line

```
t.hide()
```

will make sure that turtle `t` is not displayed when `t.show()` is called.

4.16 reveal(), visible()

The line

```
t.reveal()
```

will make sure that turtle `t` is displayed when `t.show()` is called.

4.17 arc()

The line

```
t.arc(120, 50, 'l')
```

will draw a left arc of angle 120 degrees and radius 50. The line

```
t.arc(60, 10, 'r')
```

will draw a right arc of angle 60 degrees and radius 10.

5 Basic Python commands

5.1 The for-loop

The for-loop is part of the Python programming language. The program

```
for i in range(5):
  print(i)
```

will print the numbers 0, 1, 2, 3, 4 one number per line:

```
0
1
2
3
4
```

Here `i` is the counting index. In the first cycle i = 0, in the second cycle, i = 1 etc. Now let's apply it to the Turtle. The program

```
for j in range(4):
  tina.go(50)
  tina.left(90)
```

makes the Turtle `tina` draw a square of edge length 10 steps. In other words, the turtle will go 10 steps forward, then turn left 90 degrees, and after that the same is repeated three more times.

5.2 The if-else condition

The if-else condition is another basic element of the Python programming language. The program

```
a = 7
if a > 0:
  print("The number a is positive.")
else:
  print("The number a is negative or zero.")
```

will print

```
The number a is positive.
```

And the program

```
a = -2
if a > 0:
  print("The number a is positive.")
else:
  print("The number a is negative or zero.")
```

will print

```
The number a is negative or zero.
```

5.3 The while-loop

The while-loop is suitable for situations when it is not known how many repetitions will be needed. Such as, for example, when we want to add numbers 1, 2, 3, 4, 5, ... etc. until their sum exceeds 1000:

```
sum = 0
counter = 1
while sum < 1000:
  sum += counter
  counter += 1
print("Sum =", sum)
```

On line 1, a variable named `sum` is initialized with zero. On line 2, variable `counter` is initialized with 1. Line 3 checks if `sum` is less than 1000. If so, lines 4 and 5 and executed, and the program returns to line 3. If not, then the loop ends and the program continues to line 6 where the sum is printed. On line 4, the current value of the counter is added to the sum, and on line 5 the counter is increased by one.

5.4 Custom functions and the command `def`

The command `def` is used to define custom functions. What is a custom function? Sometimes there is a group of actions that you need to do more than once. And perhaps in different parts of the main program. Then it makes sense to create a custom function for that group of commands.

For example, the following custom function `hexagon(T, s)` will make Turtle `T` draw a haxagon of side length `s`:

```
def hexagon(T, s):
  for i in range(6):
    T.go(s)
    T.left(60)
```

As always, do not forget the colons at the end of lines 1 and 2, and pay good attention to the indentation.

6 Advanced Turtle commands

6.1 isdown()

The line

```
t.isdown()
```

will return True is turtle `t`'s pen is down, and False otherwise.

6.2 setpos(), setposition()

The line

```
t.setpos(10, 20)
```

is the same as

```
t.goto(10, 20)
```

6.3 setx()

The line

```
t.setx(20)
```

will move turtle `t` to x-coordinate 20 while leaving its y-coordinate unchanged.

6.4 sety()

The line

```
t.sety(30)
```

will move turtle `t` to y-coordinate 30 while leaving its x-coordinate unchanged.

6.5 getx()

The line

```
t.getx()
```

will return the turtle's x-coordinate.

6.6 gety()

The line

```
t.gety()
```

will return the turtle's y-coordinate.

6.7 getcolor()

The line

```
t.getcolor()
```

will return the turtle's color.

6.8 getwidth()

The line

```
t.getwidth()
```

will return the turtle's line width.

6.9 getheight()

The line

```
t.getheight()
```

will return the turtle's (vertical) line height.

6.10 getangle()

The line

```
t.getangle()
```

will return the turtle's angle.

7 3D modeling commands

7.1 extrude()

The line

```
t.extrude(5)
```

will extrude the turtle's trace to 3D, making it a 3D object of thickness 5.

7.2 rosol()

The line

```
t.rosol()
```

will revolve the turtle's trace about the y-axis, creating a rotationally-symmetric 3D solid. NOTE: The turtle's trace should be in the first quadrant for this command to work properly. Optional parameters are angle (default value 360 degrees) and angular subdivision (default value 36).

7.3 rosurf()

The line

```
t.rosurf()
```

will revolve the turtle's trace about the y-axis, creating a rotationally-symmetric 3D surface. 3D surfaces are paper-thin and so they cannot be 3D-printed. NOTE: The turtle's trace should be in the first quadrant for this command to work properly. Optional parameters are angle (default value 360 degrees) and angular subdivision (default value 36).

7.4 roshell()

Formerly this command was called `revolve()`. The line

```
t.roshell()
```

will revolve the turtle's trace about the y-axis, creating a rotationally-symmetric 3D shell. NOTE: The turtle's trace should be in the first quadrant for this command to work properly. Be careful: Rotational shells have more than twice the number of facets than rotational solids or surfaces. Therefore, this operation may take a longer computing time. Optional parameters are angle (default value 360 degrees) and angular subdivision (default value 36).

7.5 spiral()

The line

```
t.spiral(720, 30, 48)
```

will spiral the turtle's trace about the y-axis. 720 means two full revolutions, 30 means an elevation gain 30 per revolution, and 48 is the angular subdivision per revolution.

7.6 export()

With the program

```
out = t.export()
SHOW(out)
```

Turtle `t` will export its 2D or 3D geometry to an object named `out`. This object can then be used as part of a more complex 2D or 3D model. In the above example, it is just displayed.

8 The 3D Turtle

The 3D Turtle (in the following we'll call it just Turtle) allows you to move and draw in all three spatial directions X, Y and Z. Most commands are the same, so let's just focus on the new ones.

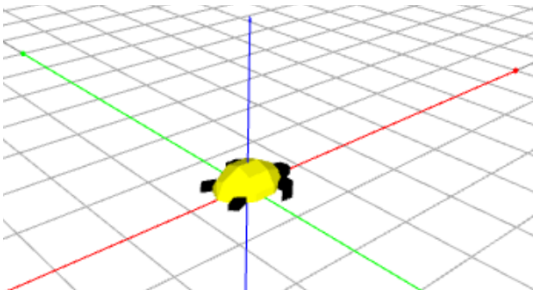
8.1 NCLabTurtle3D()

To create a new 3D Turtle `t`, type

```
t = NCLabTurtle3D()
```

8.2 Initial position

When the Turtle is created, she stands at the origin (0, 0, 0) and faces East (positive X direction). Her belly lies in the XY plane. This is the same in 3D as it was in 2D:

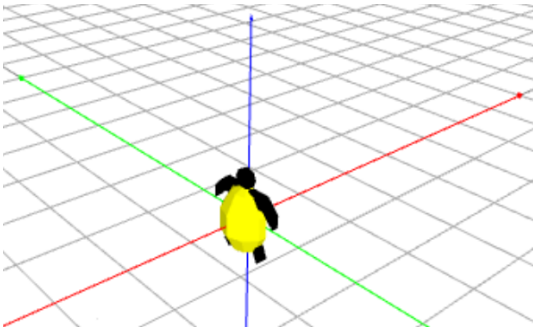


Recall that the X axis is red, Y axis green, and Z axis blue.

8.3 The bellyplane

The Turtle can turn left and right in the 3D space exactly in the same way as she did in the XY plane. She is turning in the plane that is associated with her belly – let's call it the bellyplane. In the image above, her bellyplane is the XY plane. That's the same as it was in 2D, and therefore turning in that plane is exactly the same as turning in 2D.

But in the image below, the Turtle faces the positive Z direction and her bellyplane is the YZ plane.

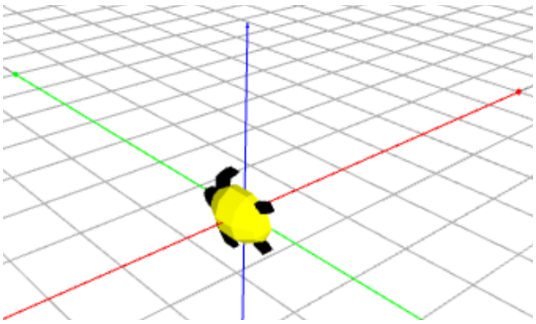


8.4 left(), right()

Consider again the last image above – the Turtle faces the positive Z direction and her bellyplane is the YZ plane. Now when she turns left 90 degrees,

```
t.left(90)
```

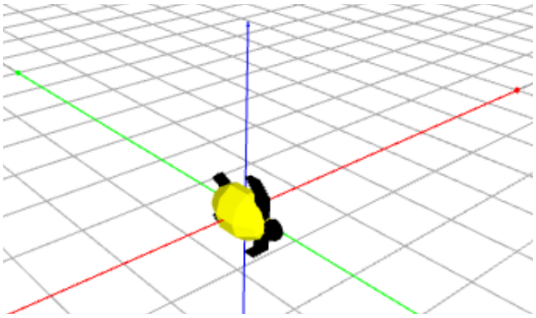
she will turn in her bellyplane, and face the positive Y direction:



We are using 90 degree angles here for simplicity, of course the Turtle can use arbitrary angles. If she turns 90 degrees right instead,

```
t.right(90)
```

she will be facing the negative Y direction:

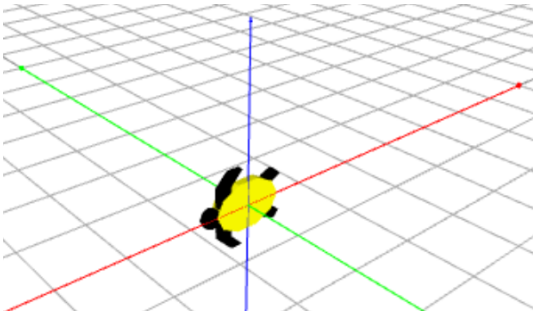


8.5 up(), down()

The up() and down() commands are again relative to the Turtle's bellyplane. Consider the last image above. Now when she turns up 90 degrees,

```
t.left(90)
```

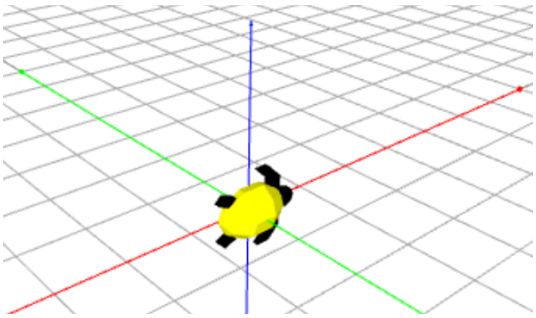
she will be facing the negative X direction and her bellyplane will be the XZ plane:



When she turns down 90 degrees instead,

```
t.down(90)
```

she will be facing the positive X direction, her bellyplane will be the XZ plane again, but her body will be on the opposite side of the XZ plane:

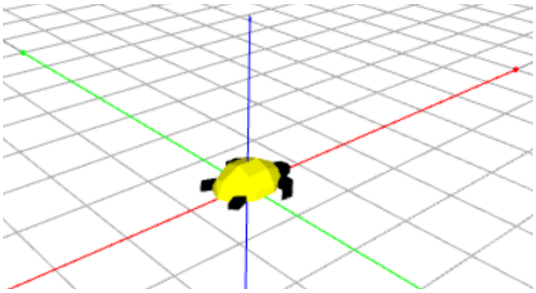


8.6 roll()

The Turtle uses the roll() command to roll about her axis exactly in the same way airplanes do. Consider the last image above. With the command

```
t.roll(90, 'l')
```

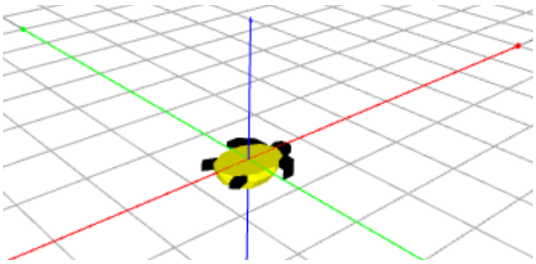
she will be facing positive X direction and her bellyplane will be the XY plane:



The 'l' in the roll() command stands for "to the left". She can also roll to the right using 'r' as the second parameter of the roll() command. When she rolls 90 degrees to the right instead,

```
t.roll(90, 'r')
```

she will be facing positive X direction, her bellyplane will be the XY plane, but her body will be on the opposite side of the XY plane:



8.7 Flight analogy - commands yaw(), pitch() and roll()

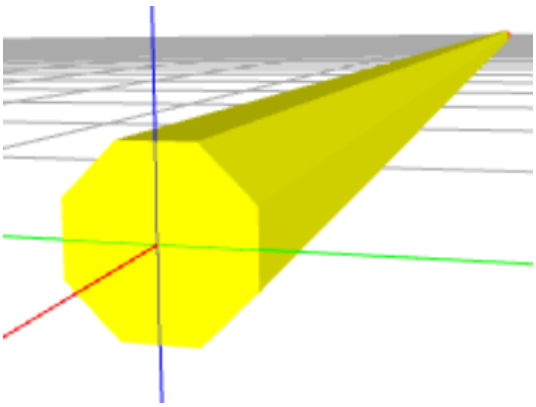
An airplane can turn in three different ways – using yaw, pitch and roll. You already know how the roll() command works.

The yaw() command is doing the same thing as left() and right(). When used with a positive angle, such as yaw(45), the Turtle will turn to the left. When used with a negative angle, such as yaw(-30), the Turtle turns to the right.

The pitch() command is doing the same thing as up() and down(). When used with a positive angle, such as pitch(10), the Turtle will lift her nose up. When used with a negative angle, such as pitch(20), the Turtle turns her nose down.

8.8 Line types and the edges() command

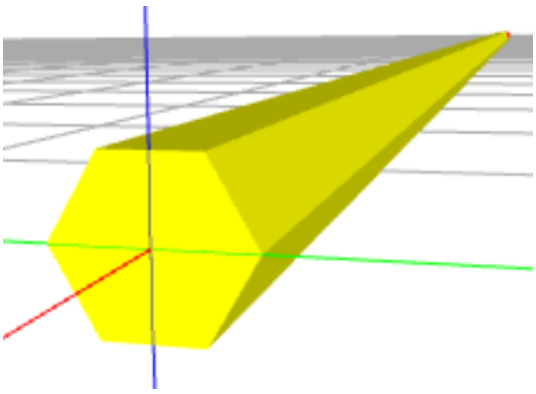
The default cross-section of the Turtle's 3D line is an octagon:



This can be changed using the edges() command. For example, with

```
t.edges(6)
```

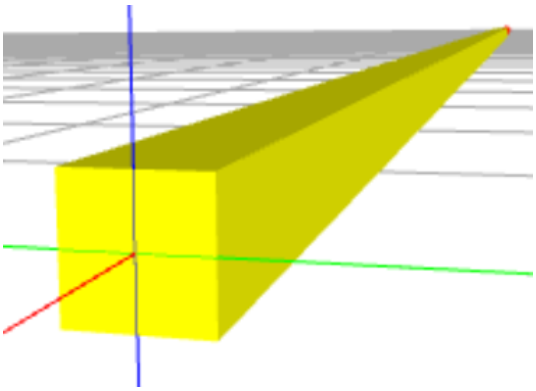
the cross-section will be a hexagon:



And with

```
t.edges(4)
```

it will be a square:



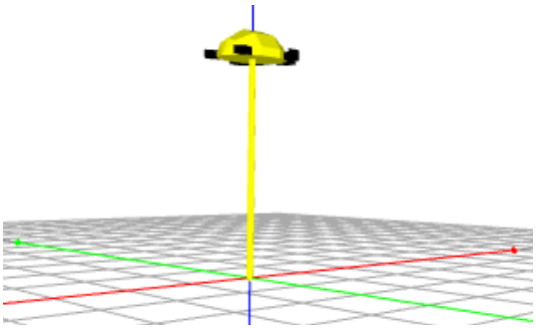
and so on. The lowest number of edges is three.

8.9 Difference in the goto() command

The goto() command accepts three coordinates of the end point, as expected. But there is one more difference: When the Turtle arrives at the target point, she will reset all her angles to zero. In other words, her angles will be as if she was just created. To illustrate this, the code

```
t = NCLabTurtle3D()
t.color(YELLOW)
t.goto(0, 0, 30)
t.show()
```

will produce:



8.10 angles()

The command angles(leftangle, upangle, rollangle) will set the Turtle's three angles. Here leftangle is the angle between her axis and the XZ plane, upangle is the angle between her axis and the XY plane, and rollangle is her roll angle about her axis. Initially all three are zero. For example, the following line will set these angles to 45, 30 and 20, respectively:

```
t.angles(45, 30, 20)
```

8.11 getangles()

This command can be used to find the Turtle's three angles in degrees. Sample use:

```
aleft, aup, aroll = t.getangles()
```

8.12 printlines()

This command is useful for debugging. It will print in text form all the lines where the Turtle went. For example, the code

```
t = NCLabTurtle3D()
t.go(30)
t.up(90)
t.go(20)
t.left(90)
t.go(10)
t.printlines()
```

will produce the output:

```
---
Start: 0 0 0
End:   30 0 0
X: 1 0 0
Y: 0 1 0
Z: 0 0 1
---
Start: 30 0 0
End:   30.0 0.0 20.0
X: 0.0 0.0 1.0
Y: 0 1 0
```

```
Z: -1.0 0.0 0.0
---
Start: 30.0 0.0 20.0
End:   30.0 10.0 20.0
X: 0.0 1.0 0.0
Y: -0.0 0.0 -1.0
Z: -1.0 0.0 0.0
```

Here Start and End are the starting and end points of each line, and the vectors X, Y, Z are three unit vectors that define the local coordinate system associated with the Turtle.