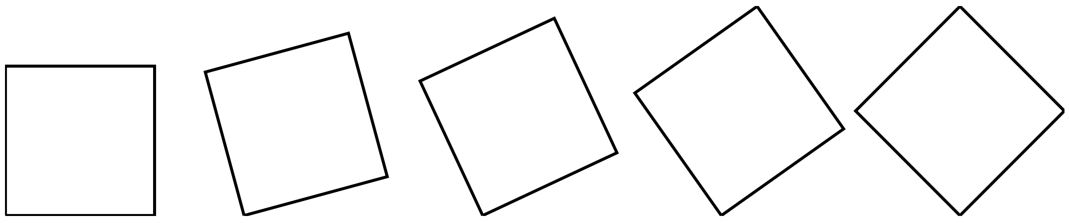
1

Directions and Angles



By now you should be getting tired of drawing figures only in *fixed* directions. In this chapter you shall learn how to change the direction in which a robot moves and draws a line in *any* direction and in relative manner. For this purpose, we present the mathematical concept of angle to indicate to a robot how far it should turn. However, if you understand clearly what an angle is and how to measure angles in degrees, you may skip Section 3 and proceed readily with the examples and experiences of Section 5.

We start by presenting the elementary messages that robots understand to change their direction. Note that we are hiding the robots from the illustrations using the message beInvisible so that you can get clearer pictures.

1 Right or Left?

In the previous chapter, you learned that a robot could face different directions using the messages east, north, northEast, northWest, south, southEast, southWest and west. However, with these messages you can not change the direction of your robot from a given angle such as 15 degrees. In addition, you cannot say that you want for example turn a robot of a quarter of a turn from the current direction.

To orientate a robot along any direction you should use the two methods turnLeft: and turnRight: which ask a robot to turn to the left or the right. As the colon at the end of the names indicates, both methods are expecting an argument. This argument is the angle between the heading of the robot before and after issuing the message. This angle is given in degrees. For example the expression pica turnLeft: 15 asks pica to turn on the left fifteen degrees from its current direction and pica turnRight: 30 turns on the right 30 degrees from its current direction. Figure 1.1 illustrates the effect of the messages turnLeft: and turnRight: first when a robot is pointing to the east and second when a robot is pointing to another direction.

Now you should practice but remember that when a new robot is created, it always points to the east or the right of the screen.

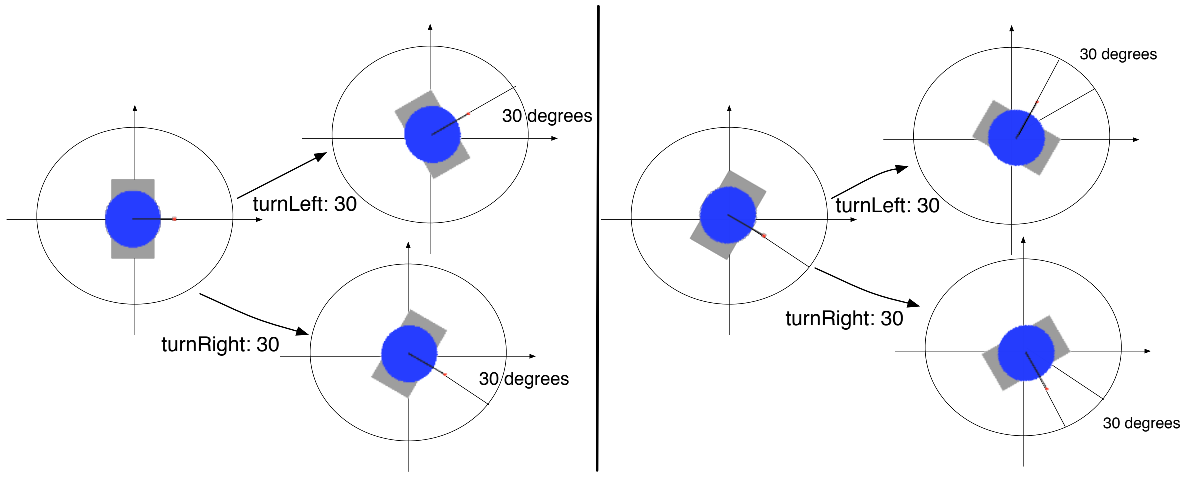


Figure 1.1: Left: turning left or right from the east position. Right: turning left of right from another position.

Experiment 1.1
Read the following scripts Problem One (Script 1.1) and Problem Two (Script 1.2) and try to guess what the created robot will do. Then experiment with the scripts: Change the angle values for example to determine what is the angle so that the robot turns a quarter, an half of a complete turn or a complete turn. If you have problem with the notion of angle read carefully the Section 3 before continuing.

Script 1.1 (Problem One)

```
| pica |
pica :=  Bot new.
pica go: 100.
pica turnLeft: 45.
pica go: 50.
pica turnLeft: 45.
pica go: 100
```

Script 1.2 (Problem Two)

```
| pica |
pica := Bot new.
pica go: 100.
pica turnRight: 60.
pica go: 100.
pica turnLeft: 60.
pica go: 100
```

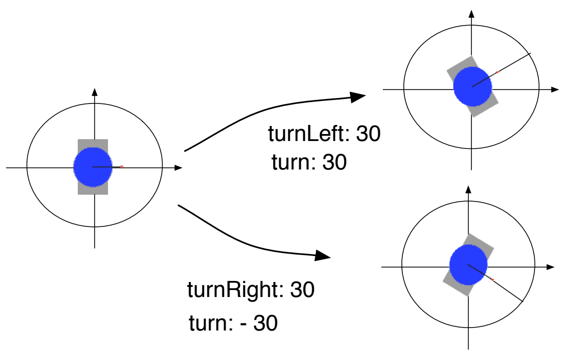


Figure 1.2: Some angles starting from the east directions.

About Mathematical Conventions. In mathematics, there exist some conventions for the rotations: a rotation with a negative angle is clockwise and with a positive angle inverse clockwise. In this book, you can also follow the mathematical conventions, using the message turn:. Hence, the message turnLeft: anAngle is equal to the message turn: anAngle while, the message turnRight: anAngle is equal to turn: - anAngle that is the angle negated as shown by the Figure 1.2.

2 Absolute versus Relative Orientation

Now you should feel confident that you can ask a robot to execute any drawing made with lines of any sort. Before going further, have you understood the difference between orienting a robot using the methods north, south, east, and west and using the methods turnLeft: and turnRight:? Figure 1.3 shows some angles which are measured starting from the east direction.

To explain the difference we ask you to complete the following experiments 1.2, 1.3 and 1.4.

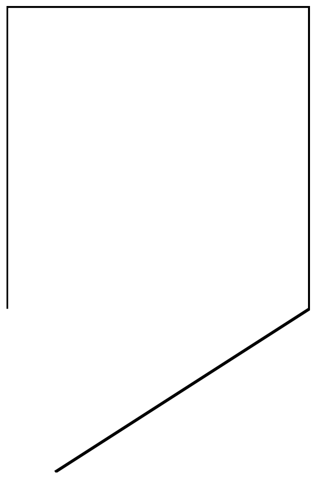
Experiment 1.2 (*A Relative Square*)
Define a script drawing a square using the methods turnLeft: or turnRight:.

Experiment 1.3
Now, add the following line: pica turnLeft: 33. before the first line containing a go: message in the script you just defined. As you can see one still obtains a square, but it is tilted.

Experiment 1.4
Finally type the Script 1.3 that draws a square but only using the methods north, south, east and west that we presented in the previous chapter.

Script 1.3 (*A Broken Square)*

```
| pica |
pica := Bot new.
pica turnLeft: 33.
pica go: 100.
pica north.
pica go: 100.
pica west.
pica go: 100.
pica south.
pica go: 100.
```



Do you still obtain a square? No! The first side drawn by the robot is slanted whereas the other sides are either horizontal or vertical. What these two scripts Script 1.3 and Script 1.2 show you is the difference between *relative* and *absolute* direction changes.

- The methods north, south, east and west change direction in an *absolute* manner. The direction to which the robot will point does *not* depend on the current direction in which it is pointing.
- The methods turnLeft: and turnRight: change direction in a *relative* manner. The direction to which the robot will point *depends* on its current direction.

Figure 1.3 shows the equivalence between relative moves starting with a robot pointing to the east and absolute moves. As you know know, this equivalence is not correct if the robot is pointing to any other direction than the east. It is worth to note that turning 180 degrees makes the robot pointing in the opposite direction which is often used in the scripts.

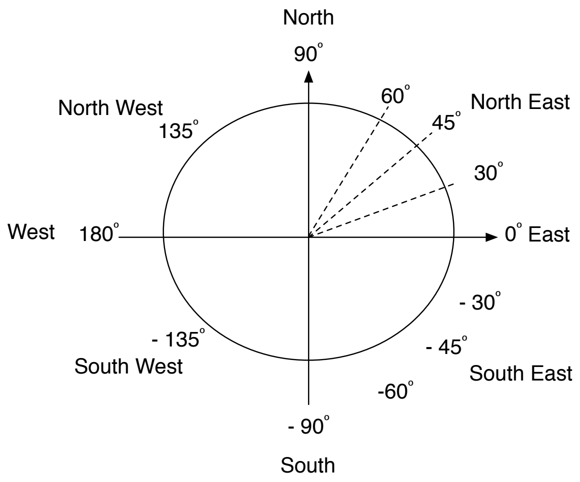


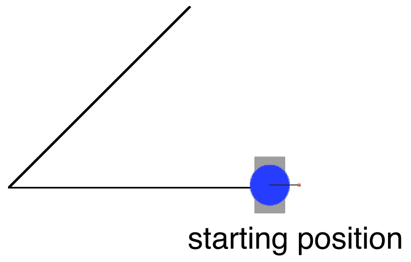
Figure 1.3: Comparing absolutes and relative angles starting from the east direction.

3 The Right Angle of Things

As you should know by now, a newly created robot is pointing east, that is, toward the right hand side of the screen. If we ask this robot to turn left by 90 degrees, it will end up heading north. If we ask it to turn right by 90 degrees, it will end up heading south. Here is a script (1.4) illustrating what a turn left by 45 degrees is doing. To help you following it we added the starting position of the robot.

Script 1.4 (*The meaning of angles)*

```
| pica |
pica := Bot new.
pica west.
pica go: 100.
pica east.
pica turnLeft: 45.
pica go: 100.
```



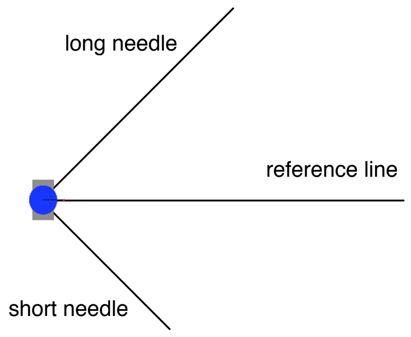
The first part of script 1.4, up to the line `pica east`, draws an horizontal line to indicate the eastern direction. The last part draws the line in the direction 45 degrees left of the east direction. You can vary the value of the angle to see what other values represent. Try the values 60, 120, 180, 240, 360, and 420. In particular, note that a turn by 180 degrees amounts to going back on the previous direction.

Do you see any difference between 60 and 420? No. Any two angle values whose difference is 360 or any multiple thereof are equivalent. Try an angle value of 1860 ($1860 = 60 + 360 \times 5$). The result is the same as that with angle values 60 and 420. 360 degrees represents a single turn. As far as orienting oneself, it does not matter whether you add one or more full turns to the orientation. Keep this in mind when dealing with angles.

Now, let us try the method `turnRight:`. Script 1.5 draw two needles of and old watch and a reference line. It uses two robots, that you can use to investigate the correspondence between a left or a right turn. We added comments surrounded by " and use different font effects to help you to identify the different parts in the script. Note that you do not have to type these comments as they do not get executed.

Script 1.5 (*The meaning of angles (2))*

```
| pica daly |
pica := Bot new.
pica jump: 200.              "drawing the reference line"
pica turnLeft: 180.
pica go: 200.
pica turnLeft: 180.
pica color: Color blue.
pica turnLeft: 45.           "drawing the long needle"
pica go: 150.
daly := Bot new.
daly color: Color red.
daly turnRight: 45.          "drawing the short needle"
daly go: 100.
```



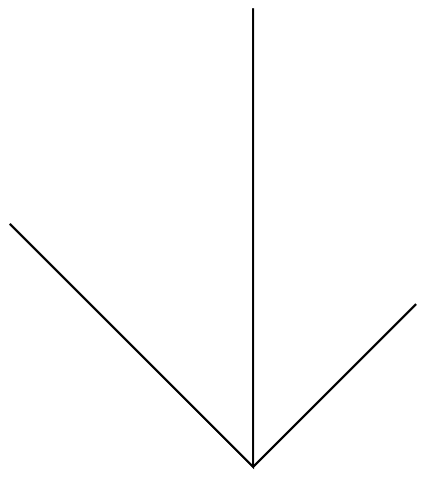
In script 1.5, the code in italic draws the reference line — that is the line representing the direction of the robot before issuing a turn method — using the fact that a turn by 180 degrees amounts to go back in the opposite direction. The reference line is also the longest lines drawn. Thus, the reference line will still be visible if the lines drawn by the robots fall on top of it. Then in normal font is the code that draws the longest needle (using pica) and in bold the code drawing the short needle using the variable pica. This is similar to the needles of an old time clock.

Experiment 1.5
Repeat the same series of angle values for both robots, that is, change the angle value for the two turn methods. Then, compare the effect of the method turnLeft: 60 (for pica) and turnRight: 300 (for daly). You can see that turning left by 60 degrees is the same as turning right 300 degrees because the sum of the two values is 360 degrees, that is a full turn.

Experiment 1.6
Now let us see what happens when turning from another direction. Here is the same script as Script 1.4 but showing the effect of turning from the north direction.

Script 1.6 (*The meaning of angles (3))*

```
| pica marge |
pica := Bot new.
pica north.
pica jump: 200.
pica turnLeft: 180.
pica go: 200.
pica turnLeft: 180.
pica color: Color blue.
pica turnLeft: 45.
pica go: 150.
daly := Bot new.
daly north.
daly color: Color red.
daly turnRight: 45.
daly go: 100.
```



Experiment 1.7
Continue to experiment with the Script 1.6 by changing the direction of reference. For the comparison to be meaningful, you have also to orient daly to the same direction than pica after creating it. Try any angle values you like and try to predict what the resulting drawing will look like before executing the script. Continue experimenting with that script until your predictions are good.

Note that it is important that you can always predict what is going to happen before executing a script because a computer blindly executes statements, even the silliest ones.

4 A Robot Clock

We have mentioned that the lines drawn in the Script 1.6 were akin to that of an old time watch. The analogy is more than real. The notion of degrees is strongly correlated to that of hours, because the ancient peoples discovered the notion of time by measuring the angle of the sun (or some star) respective to a direction of reference. However with the previous code you could draw a clock indicating an hour that was not true, that is the small needle pointing to the north and the long one to the south.

Now you will study the relationship between the small needle and the long one to represent a *real* hour: if the long needle is pointing on the south then the short one should be half of the hour distance between two numbers. Modify Script 1.6 and proceed as follows:

1. Keep the direction of reference to the north (this is how Script 1.6 is written). The reference line indicates noon.
2. Use the method `turnRight:` for both robots. A turn to the right is what the needles of an old time clock are doing.
3. You can ask `pica` to draw the minute needle by multiplying the number of the minutes you want to indicate by 6. For example, 20 after the hour is shown with the message `turnRight: 120` ($120 = 6 \times 20$).
4. You can ask `marge` to draw the hour needle by multiplying the number of the hours you want to indicate by 30. For example, 2 o'clock is shown with the message `turnRight: 60` ($60 = 30 \times 2$).

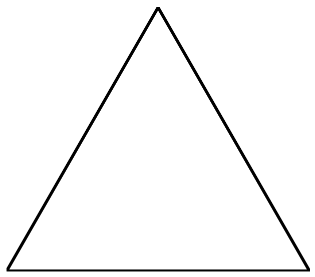
Try to indicate a few hours of your choice with that script.

5 Simple Drawings

To begin with, here is a script drawing a triangle:

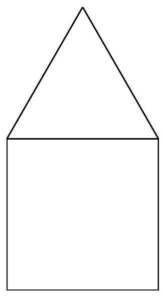
Script 1.7 (*A triangle)*

```
| pica |
pica := Bot new.
pica go: 100.
pica turnLeft: 120.
pica go: 100.
pica turnLeft: 120.
pica go: 100
```



Now, you are ready to draw a house.

Experiment 1.8
Draw a house as shown on the right. Try to draw houses of different shapes.

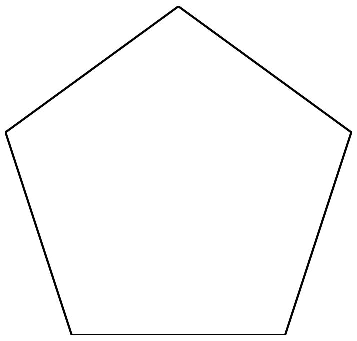


6 Regular polygons

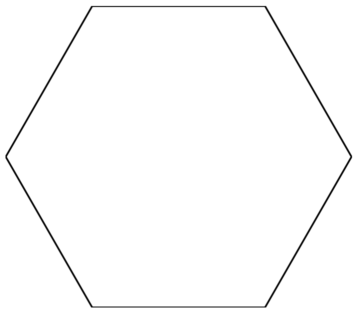
A regular polygon is a figure made with line segments of the same length. An equilateral triangle is a regular polygon with 3 sides. A square is a regular polygon with 4 sides. For example, the Script 1.7 that draws an equilateral triangle, whose side length is 100 pixels, is obtained by asking pica to go forward a given distance and to turn 120 degrees left or right, the two commands repeated 3 times. Note that there is a relationship between the number of sides and the angle: the angle is equal to 360 degrees divided by the number of sides.

Program a robot to draw a regular polygon with any number of sides by asking it to go for some length and then turn left or right by 360 degrees divided by the number of sides; this sequence must be repeated as many times as there are sides.

Experiment 1.9
Try to draw a pentagon, that is a regular polygon with five sides of 100 pixel length.

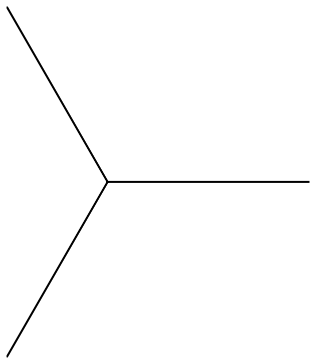


Experiment 1.10
Try to draw a hexagon, that is a regular polygon with six sides of 100 pixel length.



Maybe, you are just curious to see how far you can go. Using the cut and paste facility of the Bot Workspace, you can actually generate a regular polygon with a large number of sides. If you are in the mood, go on increasing the number of sides. However in the next chapter we will show you how this can be solved.

Experiment 1.11
Draw the symbol shows by the figure on the right.



Summary

1. A robot can be oriented *relatively* to its *current* direction using the methods turnLeft: and turnRight:.
2. The parameter given to the methods turnLeft: and turnRight: is given in degrees.
3. Turning 360 degrees corresponds to a full turn.
4. Turning 180 degrees corresponds to an half turn.
5. Angles values whose difference is a multiple of 360 degrees are equivalent.

Here is the list of the methods which you have learned in this chapter.

Method	Description	Example
turnLeft:	Ask the robot to change its direction to a number of given degrees to the left	pica turnLeft: 30
turnRight:	Ask the robot to change its direction to a number of given degrees to the right	pica turnRight: 30
turn:	Ask the robot to change its direction to a number of given degrees following the mathematical convention: to the left is the number is positive and to the right if it is negative	pica turn: -30
beInvisible	Hide the receiver	pica beInvisible
beVisible	Show the receiver	pica beVisible