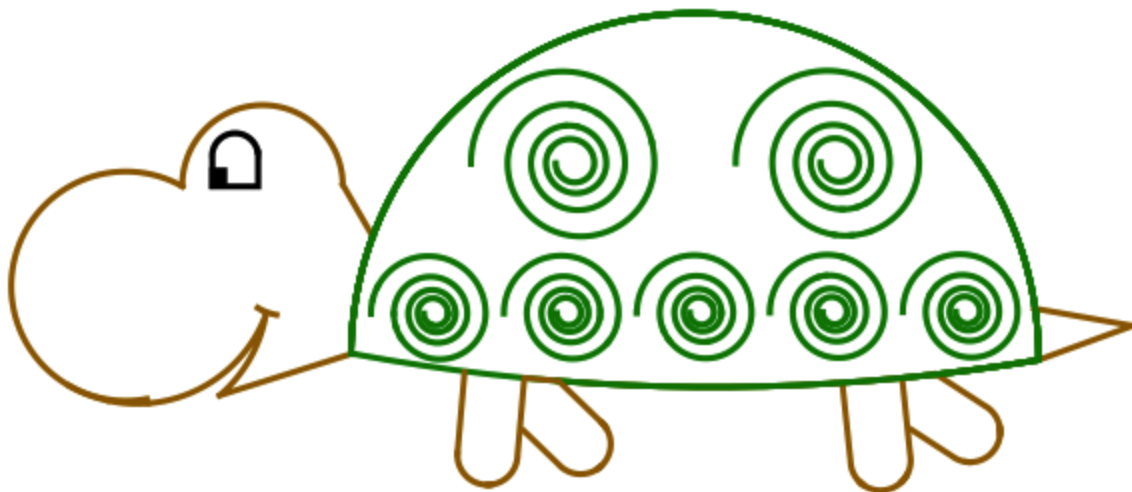


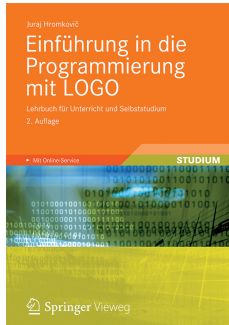
Heidi Gebauer Juraj Hromkovič Lucia Keller
Ivana Kosírová Giovanni Serafini Björn Steffen

Programming in LOGO



Programming in LOGO

The content of this script is taken from the textbook *Einführung in die Programmierung mit LOGO*, lectures 1–7. The complete textbook consists of a total of 15 lectures and contains many additional exercises and explanations, as well as comments for teachers.



Juraj Hromkovič. *Einführung in die Programmierung mit LOGO: Lehrbuch für Unterricht und Selbststudium*, 2. Aufl., Springer Vieweg 2012. ISBN: 978-3-8348-1852-2.

Version 3.0, February 26, 2014, SVN-Rev: 13903

Translation in English: Sandro Feuz

Redaction: Hans-Joachim Böckenhauer, Ivana Kosírová

Programming environment

The examples and exercises in the script are designed for the XLogo programming environment, which can be freely obtained from the website xlogo.tuxfamily.org.

For the examples to work properly, the language within XLogo has to be set to English.

Rights of use

The ABZ provides the presented material free of charge for internal usage and educational purposes by teachers and educational institutions.

ABZ

The Center for Informatics Education (ABZ) of ETH Zurich supports schools and teachers, who would like to establish or expand educational activities in the area of computer science. The support ranges from individual consultation and on-site teaching by ETH professors and the ABZ team, to training courses for teachers and maintenance of course material.

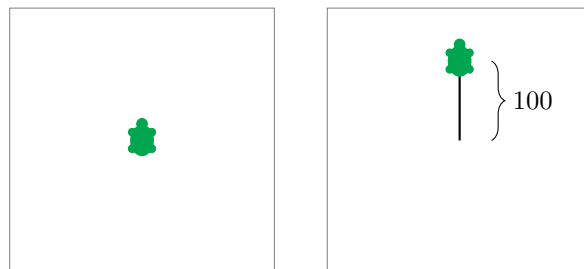
www.abz.inf.ethz.ch

1 Basic Instructions

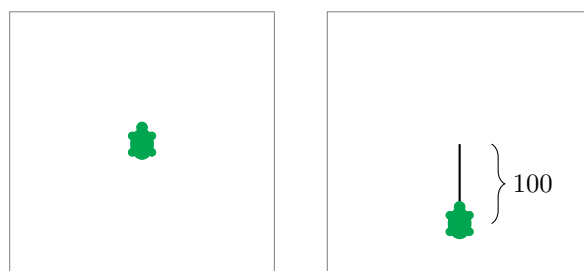
A **command** is an instruction, which the computer can understand and execute. In principle, the computer only understands very basic commands, which can then be combined to form more complicated instructions. Such a sequence of commands is called a **computer program**. Writing computer programs is not easy. There are programs which consist of millions of commands. To keep track of such a complicated program, it is very important to approach the task of writing a program in a structured and well thought out manner. This is what we will learn in this programming course.

Drawing Straight Lines

The command **forward 100** or **fd 100** moves the turtle 100 steps forward:



With the command **back 100** or **bk 100**, you can move the turtle backwards by 100 steps:



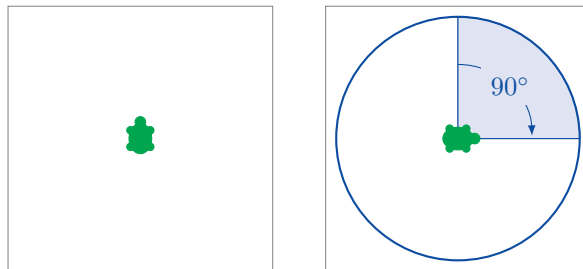
Clearing and Restarting

The command `cs` clears the entire screen and moves the turtle to its initial starting position.

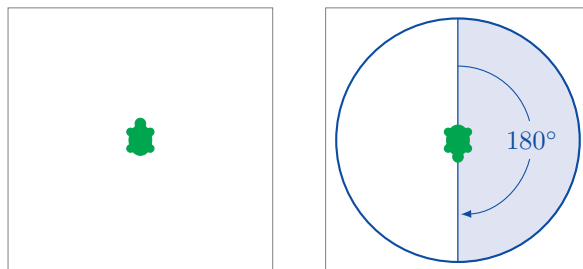
Turning

The turtle always moves in the direction it currently faces.

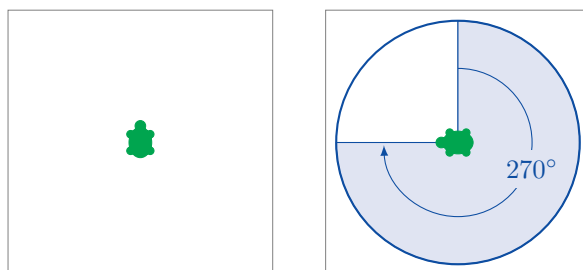
Using the commands `right 90` or `rt 90`, you can turn the turtle 90° to the right. This corresponds to a quarter circle:



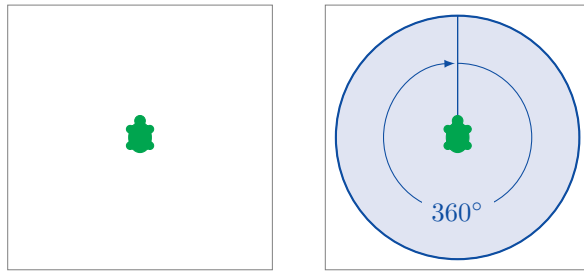
The command `right 180` or `rt 180` turns the turtle 180° to the right. This corresponds to a half-turn:



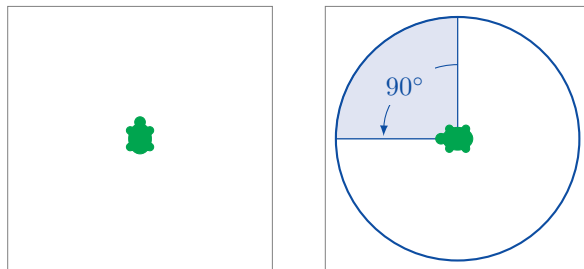
`right 270` or `rt 270` turns the turtle 270° to the right:



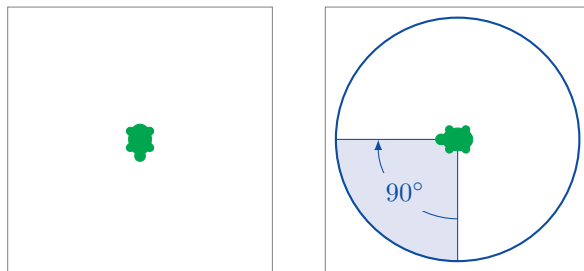
The commands **right 360** and **rt 360** turn the turtle 360° to the right. This corresponds to a whole turn:



Using the commands **left 90** or **lt 90**, the turtle turns 90° to the left:



Note that the directions of the turns are interpreted from the point of view of the turtle, illustrated by the following example using the command **rt 90**:



Programming

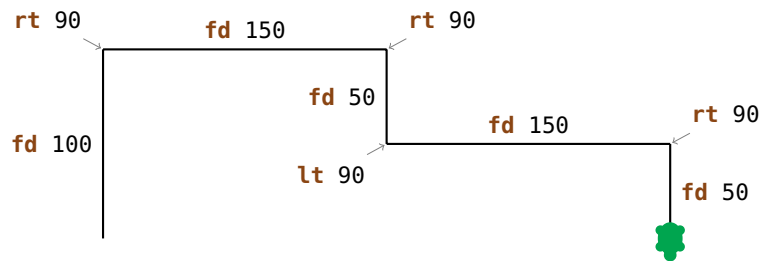
To **program** means writing multiple commands in sequence.

Exercise 1

Copy and execute the following program:

```
fd 100
rt 90
fd 150
rt 90
fd 50
lt 90
fd 150
rt 90
fd 50
```

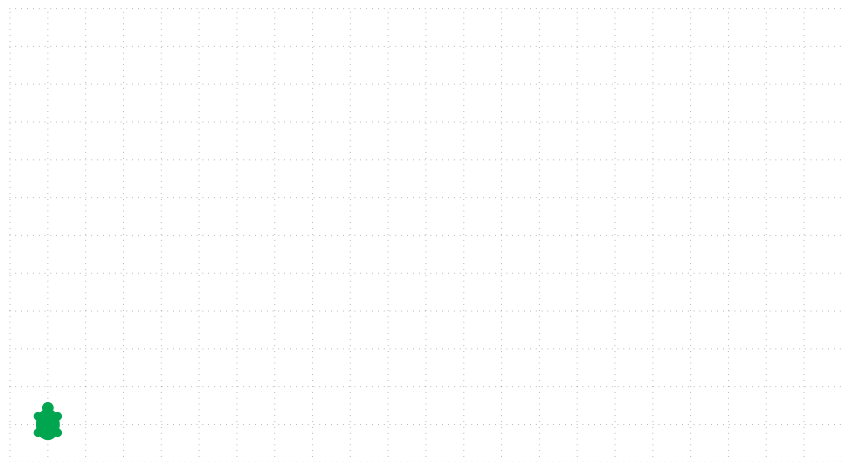
Did you get the following image?



Exercise 2

Copy the following program and execute it:

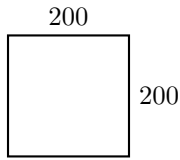
```
fd 100
rt 90
fd 200
rt 90
fd 80
rt 90
fd 100
rt 90
fd 50
```



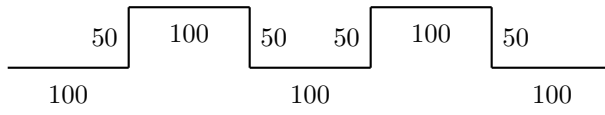
Draw the resulting picture in the given grid and mark which command has drawn each section of the image (just as we did in Exercise 1).

Exercise 3

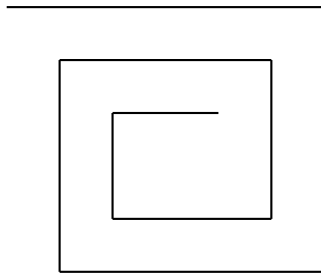
Write a program for each of the following images. For all images, you can choose the starting position of the turtle.



(a)

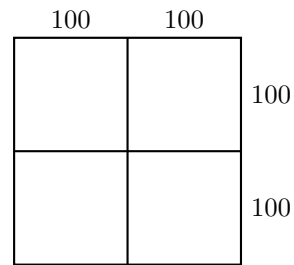


(b)



You can choose the size.

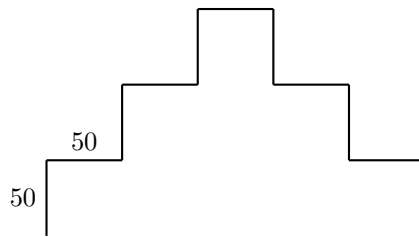
(c)



(d)

Exercise 4

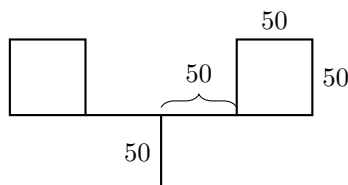
Write a program that produces the following image:



Can you rewrite your program so that it only uses the commands `fd 50` and `rt 90`?

Exercise 5

Anna would like to draw the following image. Can you help her?



2 The Command **repeat**

When we want to draw a square with sides of length 100,



we can do this with the following program:

```
fd 100  
rt 90  
fd 100  
rt 90  
fd 100  
rt 90  
fd 100  
rt 90
```

We note that the two commands

```
fd 100  
rt 90
```

are repeated four times. Wouldn't it be much simpler to tell the computer that it should just repeat these two commands four times instead of writing them four times in a row?

We can do exactly this with the following:

repeat	4	[fd 100 rt 90]
Command to repeat a program	Number of repetitions	Sequence of commands to be repeated

Exercise 6

Copy and execute the following program:

```
fd 75 lt 90
fd 75 lt 90
fd 75 lt 90
fd 75 lt 90
```

What image does the program draw? Can you use the command **repeat** to shorten the program?

Exercise 7

Copy the following program and find out what it produces:

```
fd 50 rt 60
fd 50 rt 60
fd 50 rt 60
fd 50 rt 60
fd 50 rt 60
fd 50 rt 60
```

Shorten the program by using the command **repeat**.

Exercise 8

Use the command **repeat** to write a program that draws a square having sides of length 200.

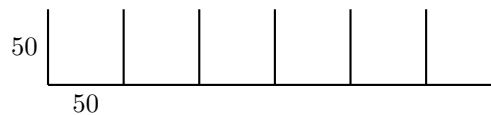
Exercise 9

Copy the following program:

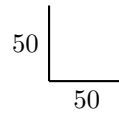
```
fd 100 rt 120
fd 100 rt 120
fd 100 rt 120
```

What is the result if you execute the program? Use the command **repeat** to shorten the program.

We are interested in drawing the following image using the command **repeat**:



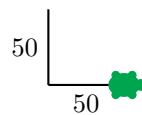
Prior to actually drawing, we have to think about what the repetitive pattern is. For example, we could use the following image as the repetitive pattern:



This image can be drawn with the following program. It assumes that we are starting in the lower left corner:

```
fd 50 bk 50 rt 90 fd 50
```

After executing the program, the turtle is positioned in the lower right corner facing right:

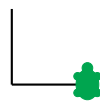


All we have to do now is get the turtle to face upwards, so that we can draw the image one more time. We can use the command **lt 90** to accomplish that.

Let us execute the whole program to check the progress we made so far:

```
fd 50 bk 50 rt 90 fd 50  
lt 50
```

We get the desired result:



If we now execute the program once again, we will get the following:



So, we see that our idea works and we can now repeat the program 6 times:

```
repeat 6 [ fd 50 bk 50 rt 90 fd 50 lt 90 ]
```


 pattern orienting

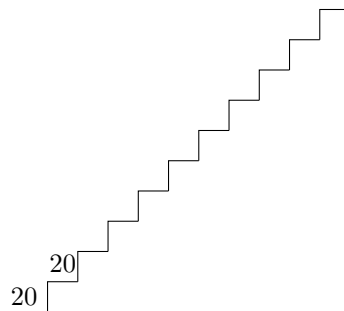
Many of the exercises can be solved with this approach. Always remember that you should first find the repetitive pattern. Then you write one program to draw the *pattern* and another program to *orientate* the turtle so that it faces the correct direction for the next repetition of the pattern. The structure of your final program should then look as follows:

repeat *Number of repetitions* [*pattern orienting*]

Exercise 10

Drawing staircases.

(a) Draw a staircase consisting of 10 steps of size 20 each.



- First find the repetitive pattern and write a program drawing it.
- Think about how to write a program that makes the turtle face the correct direction for the next repetition of the pattern.
- Put together both programs to solve the task.

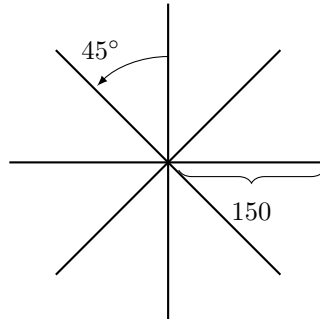
(b) Draw a staircase with 5 steps of size 50 each.

(c) Draw a staircase with 20 steps of size 10 each.

Exercise 11

We are now going to draw stars.

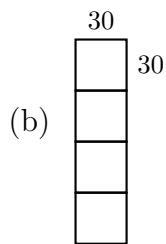
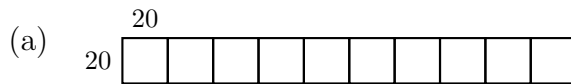
(a) Draw the following star.



(b) The star has eight rays, each of length 150. Can you draw a star with 16 rays of length 100?

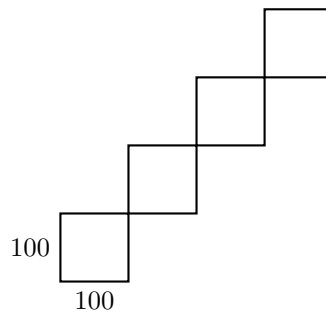
Exercise 12

Draw the following pictures with a program:



Exercise 13

Draw the following picture with a program:



Exercise 14

Copy and execute the following program:

```
repeat 4 [fd 100 rt 90]
rt 90
repeat 4 [fd 100 rt 90]
rt 90
repeat 4 [fd 100 rt 90]
rt 90
repeat 4 [fd 100 rt 90]
rt 90
```

What does it draw? Can you make this program even shorter?

Walking Mode

Our turtle is usually in the **pen mode**. This means the turtle has a pen attached to it and, whenever it moves, a line is drawn.

In the **walking mode**, however, the turtle moves without drawing. You can switch to the walking mode using the following command:

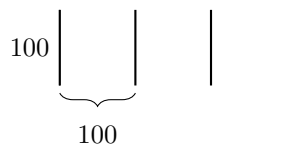
penup or **pu** for short.

To switch back to pen mode, we use the following command:

pendown or **pd** for short.

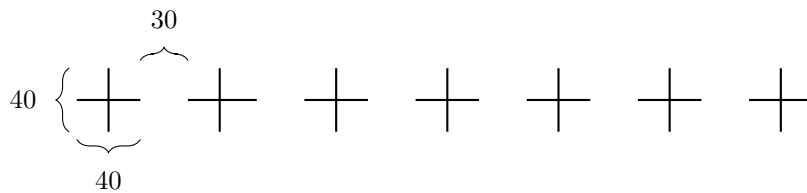
Exercise 15

Draw the following image with a program:



Exercise 16

Write a program that draws the following image:



3 Naming and Calling Programs

We can give a name to each program we have written so far. If we then write the name of the program in the command line, the program gets executed.

The program to draw a square with sides of length 100 is:

```
repeat 4 [fd 100 rt 90]
```

We can name this program **SQUARE100** in the following way:

```
to SQUARE100
repeat 4 [fd 100 rt 90]
end
```

We have written the same program twice. Once with and once without giving it a name.

To write a program with a name, we have to use the **editor**. Such programs will be indicated by a gray box. As soon as we are finished writing the program, we have to click on the button with the turtle to close the editor.

Everyone can choose their own name for the program. We have chosen **SQUARE100**, because we wanted to indicate that the program draws a square with sides of length 100. The only restrictions on the name are that it consists only of letters and digits, and that it is only one word (it must not contain any spaces).

After having written the program, nothing will be drawn yet. We have only named the program, but we have not executed it yet. If we now type the name

SQUARE100

into the command line, then the program **repeat 4 [fd 100 rt 90]** gets executed. The screen will show:



Let us go back to Exercise 12(a). We could simplify the solution by first writing a program for the repetitive pattern, a square with sides of length 20, and giving this program an appropriate name:

```
to SQUARE20
repeat 4 [fd 20 rt 90]
end
```

After drawing `SQUARE20`, the turtle is located in the lower left corner of the square:



To draw the next square, we have to move the turtle to the lower right corner. We can do that with the following program:

```
rt 90 fd 20 lt 90
```

We label this program as well:

```
to POSITION20
rt 90 fd 20 lt 90
end
```

Using those two programs we can write a program for Exercise 12(a) as follows:

```
repeat 10 [SQUARE20 POSITION20]
```

We can then label the resulting program as well. For example:

```
to ROW10
repeat 10 [SQUARE20 POSITION20]
end
```

The programs `SQUARE20` and `POSITION20` are called **subprograms** of program `ROW10`.

Exercise 17

Write a program to solve Exercise 12(b) that uses a subprogram to draw a square with sides of length 30. Your final program should look as follows:

```
repeat 4 [SQUARE30 POSITION30]
```

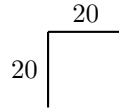
Hence, you have to figure out the subprograms `SQUARE30` and `POSITION30`.

Exercise 18

Use program **SQUARE100** as a subprogram to draw the image from Exercise 13.

Exercise 19

Write a program that draws a step



and use it as a subprogram to solve Exercise 10(a).

Exercise 20

Find another solution to Exercise 11(a) using the following program as a subprogram:

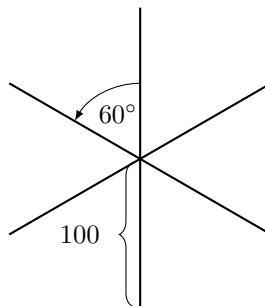
```
to LINE  
fd 150 bk 150  
end
```

Exercise 21

Write the following program in the editor:

```
to RAY  
fd 100 bk 200 fd 100  
end
```

Use the program **RAY** as a subprogram of a program **STAR6** to draw the following image:



Exercise 22

Solve Exercise 15 and Exercise 16 once again using subprograms.

Exercise 23

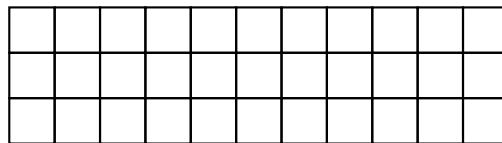
We have already written the program `ROW10`. What does the following program do?

```
ROW10 fd 20 lt 90 fd 200 rt 90
```

Test your idea using the computer.

Exercise 24

Write a program that draws the following image:



Exercise 25

Drawing squares of different sizes.

- Write a program that draws a square with sides of length 50 and name it `SQUARE50`. Execute your program to check whether it works as intended.
- Write a program that draws a square with sides of length 75.
- Execute the program

```
SQUARE50  
SQUARE75  
SQUARE100
```

What does the resulting image look like?

- How could you change the above program to add three even bigger squares?

Building Houses

In the following, we want to help an architect building a housing complex. To keep the construction as easy as possible, he is planning to build all the houses in the same way. We propose the following house design:

```
to HOUSE
rt 90
repeat 4 [fd 50 rt 90]
lt 60 fd 50 rt 120 fd 50 lt 150
end
```

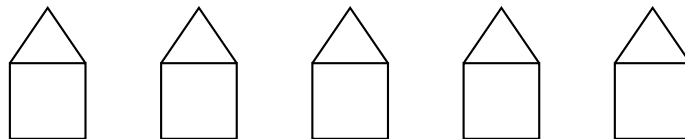
This program draws the following house:



Exercise 26

Where does the turtle start drawing the house? Think about the path the turtle takes when drawing the house using the program `HOUSE`. Where is the turtle located at the end of the execution? Draw the image and describe the effect of each command the same way we did in Exercise 1.

The architect builds the house according to our program and is satisfied with the result. Therefore, he wants to use the program to build a whole block of houses. The final block should look as follows:



Since all the houses look the same, he can use the program `HOUSE` 5 times without having to think about the design of each house. He lets the turtle draw the left-most house and then tells it to move to the starting point for the second house:



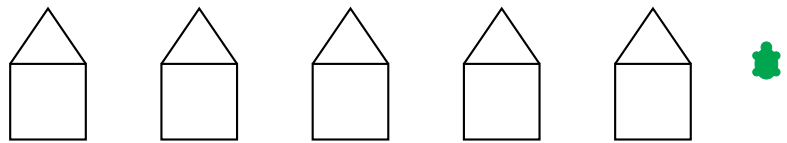
The architect does this using the program:

```
HOUSE rt 90 pu fd 50 lt 90 pd
```

Being positioned correctly, the turtle can now draw another house in the same way and move to the starting point of the next house. The process should be repeated until all five houses have been drawn. That is, we have to repeat the above program five times to get a row with five houses. The resulting program will be called **HOUSEROW**:

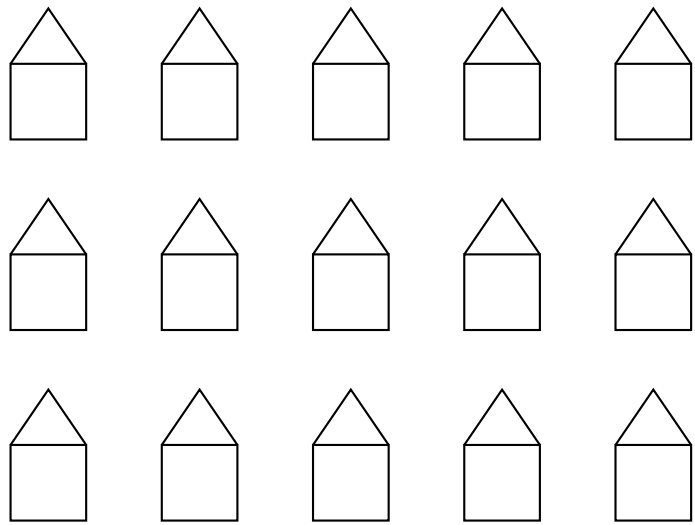
```
to HOUSEROW
repeat 5 [HOUSE rt 90 pu fd 50 lt 90 pd]
end
```

At the end of the execution, the turtle is located at the spot where the next house could be drawn:



Exercise 27

At this point we would like to extend the housing complex by additional streets. Use the program **HOUSEROW** as a building block to draw the following image:



Hint: After completion of a row, the turtle has to be moved to the correct position for building the next row.

Thick Lines and Black Squares

Exercise 28

Drawing thick lines using the program **THICK**.
Label the following program with the name **THICK**

```
fd 100  
rt 90  
fd 1  
rt 90  
fd 100  
rt 180
```

and then write in the command line

```
THICK
```

What does the turtle draw? Use a pencil to draw how the image was formed.

Exercise 29

Repeat program **THICK** 100 times using the following commands

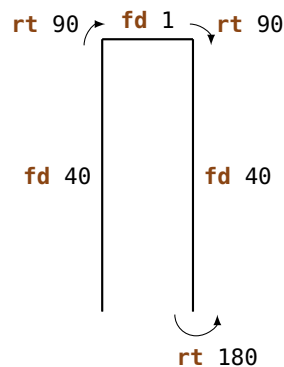
```
repeat 100 [THICK]
```

What does the resulting image look like?

Exercise 30

In this exercise we will be drawing thick lines. In Exercise 28, we have already seen that a thick line can be drawn as follows:

```
to THICK40  
fd 40  
rt 90  
fd 1  
rt 90  
fd 40  
rt 180  
end
```



Two normal lines are drawn so close together that they appear to be one thick line.

Copy the program **THICK40** and try it out.

Exercise 31

A thick line of length 40 can also be seen as rectangle of width 1 and length 40. After an execution of `THICK40` the turtle faces up. Executing the program one more time will therefore repaint the second line. We get a rectangle of width 2 and length 40. Each additional execution adds one line. When repeating `THICK40` 40 times, we get a square with sides of length 40. Try it out by repeating `THICK40` 40 times.

Write a program named `BLACK40`, which draws a black square in this way with sides of length 40.

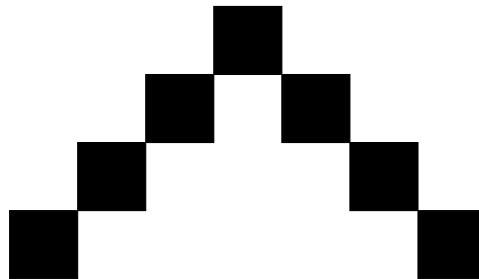
Exercise 32

Draw the following image using the program `BLACK40`:



Exercise 33

Use the program `BLACK40` to draw the following image:



Exercise 34

Draw the following image:



Exercise 35

Write a program that draws the following image:



Exercise 36

The architect decides to order the roof for the houses from another vendor. That is, he gets two types of building blocks: One called **ROOF** and another one called **BASE**. Write two programs to draw the two building blocks. Combine those programs to form a new program **HOUSE1** that draws a house.

Exercise 37

The houses in Exercise 27 are very simple. Try to be creative and come up with a new design for a house. Use your house to build a whole building complex.

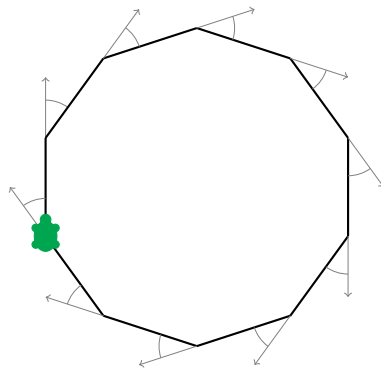
4 Regular Polygons and Circles

Regular Polygons

A regular k -gon is a polygon with k corners and k sides of equal length. To draw a regular 10-gon by pencil, you have to draw 10 lines and after every line, you need to turn (change the direction) “a little bit”.

How much do we need to turn?

When drawing a regular polygon, we turn the turtle multiple times but at the very end it always reaches its starting point and faces its initial direction.



This means that, while drawing, it has been rotated by a full 360° . When we draw a regular 10-gon, we turn exactly ten times and each time by the same angle. The angle by which we turn therefore:

$$\frac{360^\circ}{10} = 36^\circ$$

Thus we need to turn the turtle by 36° degrees each time: **rt 36**. Let us try this out by writing the following program:

```
repeat 10 [ fd 50      rt 36 ]
           side length  turning by 36°
```

Exercise 38

Draw the following regular polygons:

- (a) a regular 5-gon with sides of length 180,
- (b) a regular 12-gon with sides of length 50,
- (c) a regular 4-gon with sides of length 200,
- (d) a regular 6-gon with sides of length 100,
- (e) a regular 3-gon with sides of length 200,
- (f) a regular 18-gon with sides of length 20.

When drawing a regular 7-gon, we encounter the problem that 360 is not divisible by 7 without a remainder. In those cases, we let the computer calculate the result for us by writing

```
360/7
```

(“/” tells the computer to “divide”). The computer then figures out the result. We can therefore draw a regular 7-gon with sides of length 100 as:

```
repeat 7 [fd 100 rt 360/7]
```

Try it out.

Drawing Circles

We cannot draw exact circles by only using **fd** und **rt**. You might have noticed, however, that a regular polygon with a lot of corners almost looks like a circle. That is, we can draw circles by drawing polygons with very short sides and many corners.

Exercise 39

Test the following programs:

```
repeat 360 [fd 1 rt 1]  
repeat 180 [fd 3 rt 2]  
repeat 360 [fd 2 rt 1]  
repeat 360 [fd 3.5 rt 1]
```

3.5 means 3 and a half steps.

Exercise 40

- (a) How would you draw a very small circle? Write a program for that.
- (b) How would you draw a big circle? Write a program for that.

Exercise 41

Try to draw the following half circles. You can choose the sizes by yourself:



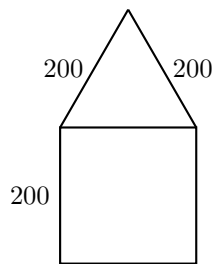
(a)



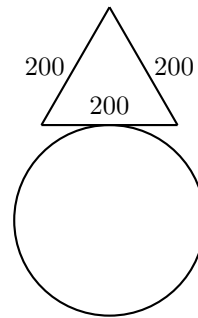
(b)

Exercise 42

Use what you have just learned to draw the following images. You can choose the size of the circle by yourself:



(a)



(b)

Freestyle Drawing

Draw a 7-gon by:

```
repeat 7 [fd 100 rt 360/7]
```

Then turn the turtle by 10° using the command

```
rt 10
```

Repeat both programs a couple of times and look at the resulting image. After each 7-gon we turn the turtle by 10° with `rt 10`. If we want to get back to our starting position, then we have to repeat it

$$\frac{360^\circ}{10^\circ} = 36$$

times. Therefore, we want to look at what the following program produces:

```
repeat 36 [repeat 7 [fd 100 rt 360/7] rt 10]
```

Exercise 43

Draw a regular polygon with 12 corners and sides of length 70. Turn it 18 times until you have reached your starting position.

Remark: You can first write a program to draw the 12-gon with sides of length 70 and label it `POLYGON12`, for example. Afterwards, you only need to complete the following program:


















```
repeat 18 [POLYGON12 rt ... ]
```

Exercise 44

Invent an exercise similar to Exercise 43. Write a program that solves your exercise.

Colors

To draw beautiful images, we need various colors. The turtle can draw not only black lines, but also lines of many different colors. Each color is assigned a number. The following table contains a list of all possible colors:

0		5		9		13	
1		6		10		14	
2		7		11		15	
3		8		12		16	
4							

Using the command

setpencolor	X
Command to change the color	Number of the desired color

the turtle changes the current color to the color given by the number **X**. We can use a shorter version of the same command: **setpc**.

Using colors, we can draw amazing patterns like for instance the pattern that is produced by the following program. First we create two named programs that draw circles of different sizes.

```
to CIRCLE3
repeat 360 [fd 3 rt 1]
end

to CIRCLE1
repeat 360 [fd 1 rt 1]
end
```

Now we use these circles to design patterns similar to the ones we have already seen:

```
to PATTERN3
repeat 36 [CIRCLE3 rt 10]
end

to PATTERN1
repeat 18 [CIRCLE1 rt 20]
end
```

Let us try the same thing using colors:

```
setpc 2
PATTERN3 rt 2
setpc 3
PATTERN3 rt 2

setpc 4
PATTERN3 rt 2
setpc 5
PATTERN3 rt 2
```

```
setpc 6
PATTERN1 rt 2
setpc 15
PATTERN1 rt 2
```

```
setpc 8
PATTERN1 rt 2
setpc 9
PATTERN1 rt 2
```

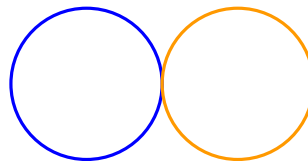
Feel free to continue and add even more patterns. Or you can come up with a totally new pattern.

Exercise 45

Use the program **PATTERN3** to draw the corresponding image with the orange color. Then use the command **setpc 7** to choose the color white. What happens if you execute **PATTERN3** again?

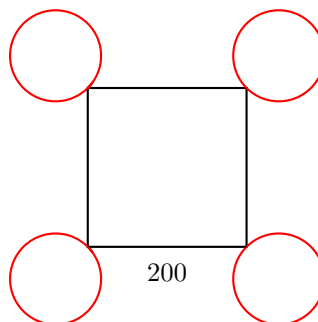
Exercise 46

Draw the following image. The turtle starts at the common point of both circles (their intersection point).



Exercise 47

Write a program that draws the following image. You can choose the size of the circle by yourself.



5 Programs with Parameters

In Lesson 3, we have learned how to assign names to programs and how to use these names to call the programs and draw the desired images. Then we have learned in Lesson 4 how to draw regular polygons. It is very tedious that we have to write a new program for every polygon with a different number of corners.

Let us look at the following three programs:

```
repeat 7 [fd 50 rt 360/7]
repeat 12 [fd 50 rt 360/12]
repeat 18 [fd 50 rt 360/18]
```

All three programs are very similar and only differ in the yellow numbers **7**, **12** and **18**. Those numbers define the number of corners of the polygon. In the following, we want to write a program that works for any polygon, no matter how many corners it has:

```
to POLYGON :CORNER
repeat :CORNER [fd 50 rt 360/:CORNER]
end
```

What did we do? Wherever the number of corners of the polygon appears, we wrote the name **:CORNER** instead of the actual number. In order for the computer to know that we want to be able to choose the number of corners freely, we have to write **:CORNER** also after the name of the program.

By typing the command **POLYGON 12** into the command line, the computer replaces the name **:CORNER** by the number **12** everywhere it appears:

```
repeat :CORNER [fd 50 rt 360/:CORNER]
```

12 12

Try it out:

```
POLYGON 3
POLYGON 4
POLYGON 5
POLYGON 6
```

We call `:CORNER` a **parameter**. In the example above, the values 3, 4, 5 and 6 are called **values of the parameter** `:CORNER`. The computer knows that this is a parameter because of the `:`. That is why, wherever a parameter appears, it has to have a `:` in front of the name.

Exercise 48

Each of the following programs draws a square of a different size.

```
repeat 4 [fd 100 rt 90]
repeat 4 [fd 50 rt 90]
repeat 4 [fd 200 rt 90]
```

The yellow numbers 100, 50, 200 can be seen as values of a parameter, which sets the size of the square. Write a program with a parameter `:SIZE` to draw squares of any size:

```
to SQUARE :SIZE
...
end
```

Exercise 49

The following programs draw circles of different sizes:

```
repeat 360 [fd 1 rt 1]
repeat 360 [fd 12 rt 1]
repeat 360 [fd 3 rt 1]
```

Write a program with a parameter to draw circles of any size. Try out your program using 1, 2, 3, 4 and 5 as different parameter values. You can choose the name of the parameter by yourself, but do not forget that there has to be a colon in front of the parameter.

Exercise 50

Do you still remember how to draw thick lines (Exercise 28)? Write a program with a parameter that can draw thick lines of any length.

Hint: You can start off by writing programs that draw lines of length 100 and length 50 to figure out what the parameter of your program should be.

Exercise 51

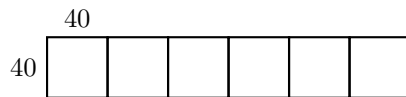
Write a program with a parameter that draws regular triangles of any size. Use your program to draw triangles of sizes

20, 40, 60, 80, 100, 120, 140, 160 and 180

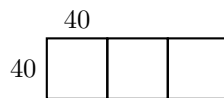
one after another. What do you get?

Exercise 52

Now we would like to draw squares of size 40, one next to another. Write a program **SQUARES** with a parameter **:AM**. The parameter **:AM** determines how many squares will be drawn. Hence, when we call **SQUARES 6**, the turtle draws the following image:

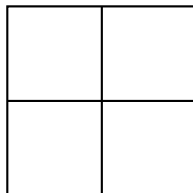


This image will be drawn after the call of **SQUARES 3**:



Exercise 53

Write a program that draws the following image consisting from 4 squares. The size of the square is determined by a parameter.

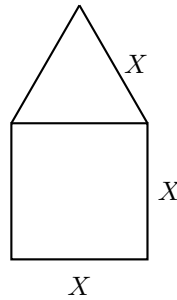


Exercise 54

Write a program with a parameter that can draw regular 6-gons of any size. Try out your program by drawing regular 6-gons of the sizes 40, 60 and 80.

Exercise 55

Write a program with a parameter `:X` that can draw houses of the following type:



Programs with Multiple Parameters

A program can have more than one parameter. When we are drawing polygons, for example, our program might have a parameter `:CORNER`, for the number of corners, and a parameter `:SIZE`, for the length of the sides.

In the following programs, the parameter `:CORNER` is marked yellow and the parameter `:SIZE` is marked green:

```
repeat 13 [fd 100 rt 360/13]
repeat 3 [fd 300 rt 360/3]
repeat 17 [fd 10 rt 360/17]
repeat 60 [fd 3 rt 360/60]
```

We can then write a program with two parameters that can draw any regular polygon:

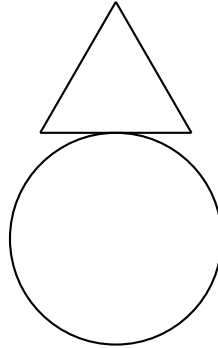
```
to POLY :CORNER :SIZE
repeat :CORNER [fd :SIZE rt 360/:CORNER]
end
```

Test the program `POLY` using the following calls:

```
POLY 12 60
POLY 12 45
POLY 8 30
POLY 9 30
POLY 7 31
POLY 11 50
```


Exercise 56

Write a program with two parameters that can draw the following image. The size of the circle as well as the size of the triangle should be freely choosable.



Exercise 57

The program

```
fd 100 rt 90 fd 200 rt 90 fd 100 rt 90 fd 200
```

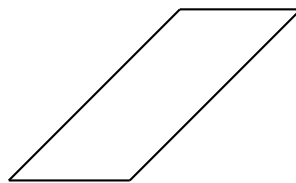
draws a rectangle of height 100 and width 200. Copy the program to see whether it really works. Write another program with two parameters that can draw rectangles of any height and width.

Exercise 58

The following program:

```
repeat 2 [rt 45 fd 200 rt 45 fd 100 rt 90]
```

draws a parallelogram:



Write a program with two parameters that can draw this kind of parallelograms with sides of any size.

Exercise 59

Draw a flower as follows. Start off with a circle using

```
POLY 360 2
```

then turn the turtle a little bit to the right

```
rt 20
```

and draw another circle

```
POLY 360 2
```

Repeat this multiple times:

```
rt 20 POLY 360 2 rt 20 POLY 360 2 ...
```

When the flower is finished, the turtle should be positioned at its initial position. The turtle will then have drawn 18 circles having turned by 20° between each of them. In total the turtle will have turned by $18 \cdot 20^\circ = 360^\circ$.

We can write the whole program as:

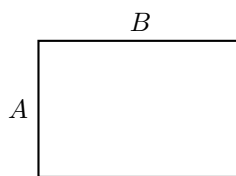
```
repeat 18 [POLY 360 2 rt 20]
```

Try it out.

- You can also draw a flower with 10 or even 20 petals (circles). How would you do that? Write a program and try it out.
- Can you write a program with a parameter that draws a flower with any number of petals (circles)?
- Can you write a program that uses the following values as parameters:
 - the number of petals (circles) and
 - the size of the circles?

Exercise 60

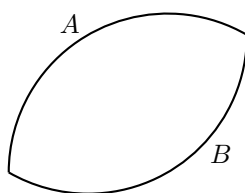
Write a program to draw any rectangle with any color:



This means that not only the height A and the width B but also the color should be freely choosable.

6 Drawing Flowers and Passing Parameters to Subprograms

A leaf



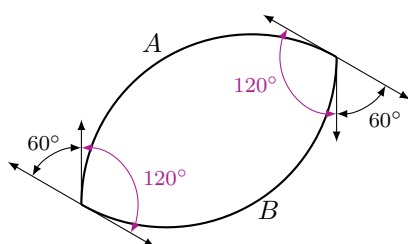
can be viewed as two arcs A and B that are glued together. We can use the following program to draw an arc:

```
repeat 120 [fd 2 rt 1]
```

Try it out.

We note that the program is very similar to the program for drawing a circle. Instead of making 360 small steps with small turns in between, however, we only repeat 120 times `[fd 2 rt 1]` and therefore only draw a third of a circle ($\frac{360^\circ}{3} = 120^\circ$).

The question that remains is how much we have to turn the turtle before we can start drawing the second arc B , which will form the lower part of the leaf. Let us look at the following illustration:



If we want to reach our initial position after having drawn the whole leaf, we will have to turn the turtle by 360° in total. While drawing part A , we turn the turtle by 120° and while drawing part B , we turn it by another 120° . Therefore, the remaining angle is

$$360^\circ - 120^\circ - 120^\circ = 120^\circ.$$

We split 120° equally between the two rotations at both spikes of the leaf:

$$\frac{120^\circ}{2} = 60^\circ.$$

Finally, we get the following program:

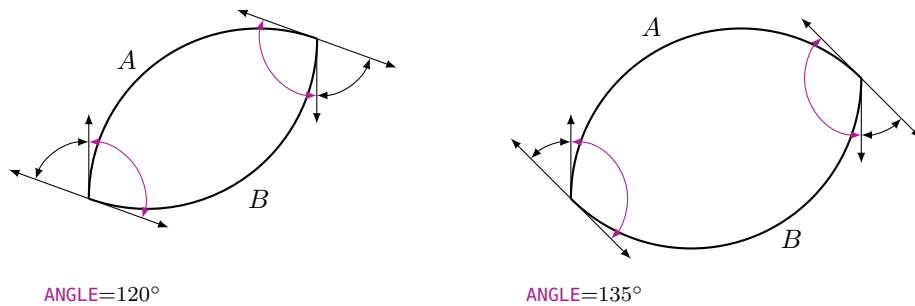
```
repeat 120 [fd 2 rt 1]
rt 60
repeat 120 [fd 2 rt 1]
rt 60
```

or even simpler:

```
repeat 2 [repeat 120 [fd 2 rt 1] rt 60]
```

Try it out.

Now, we would like to be able to draw narrower leaves (where the parts *A* and *B* are shorter) or wider leaves (where the parts *A* and *B* are longer).



We again use a program with a parameter for this. Let us call the parameter `:ANGLE`. The angle to turn by at the spike of the leaf can then be calculated as follows:

Before starting with part *B* of the leaf, we must have completed half of the total turn, that is, $\frac{360^\circ}{2} = 180^\circ$. The angle we have to turn at the spike of the leaf is then given by

$$180^\circ - \text{:ANGLE}.$$

We can now write our program in the editor:

```
to LEAF :ANGLE
repeat 2 [repeat :ANGLE [fd 2 rt 1] rt 180-:ANGLE]
end
```

Try the program by calling it from the command line as follows:

```
LEAF 20  
LEAF 40  
LEAF 60  
LEAF 80  
LEAF 100
```

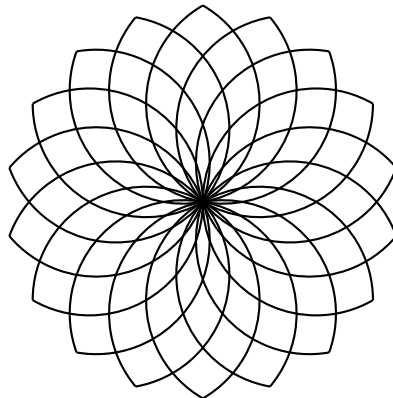
What happens?

Exercise 61

Freestyle drawing.

Start off by drawing a flower using the following program:

```
LEAF 100  
rt 20  
LEAF 100  
rt 20  
LEAF 100  
....
```



How many times do you have to repeat the commands **LEAF** and **rt 20** to completely draw the flower?

Write the program for the flower in just one line by using an appropriate **repeat** command. (Keep in mind that all turns **rt** in between the leaves have to add up to 360°).

Exercise 62

The command `fd 2` in the program `LEAF` defines the size of the circle, out of which we cut the arc. We can replace the value `2` by a parameter called `:SIZE`. Write a program

```
LEAVES :ANGLE :SIZE
```

with the two parameters `:ANGLE` and `:SIZE`, so that we can control both the length and the size of the leaves. Try out your program using the following program calls:

```
LEAVES 100 1  
LEAVES 100 1.5  
rt 100  
LEAVES 80 2  
LEAVES 80 2.5
```

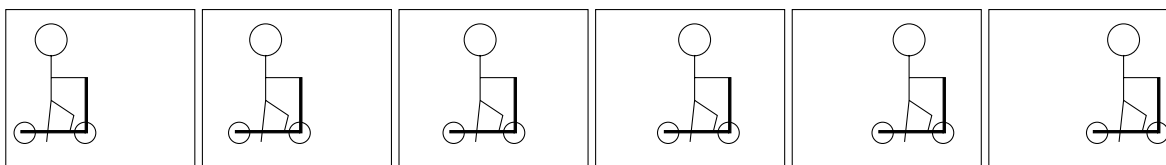
Then turn the turtle by 80° to the right and repeat the commands above.

Exercise 63

Think about other patterns you can draw.

7 Programming Animations

Do you know how to make animated movies? It works exactly the same as in the flipbook. First you draw a few pictures, which are each only a little bit different one from another. In the following picture for example, the boy on the kickboard always moves a little bit from one picture to next picture:

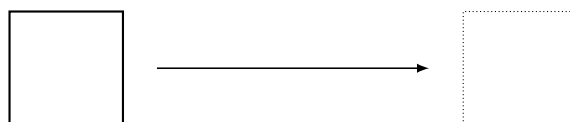


When you put the pictures one on top of another and scroll them quickly with your thumb, you get the feeling, that the boy really moves from the left to the right. Moving pictures are called **Animations**.

In this Lesson we will learn, how we can program animations with the help of the turtle.

How to Draw a Square that Leaves Traces

In our first animation, we choose a figure, which is not too complicated and which we know already for a long time: We will move a square from left to right.



We know the program, that draws a square, already from before:

```
to SQUARE100
repeat 4 [fd 100 rt 90]
end
```

Once the square is drawn, we move the turtle a bit to the right and we draw another square again. We repeat this a couple of times.

In the following program, we draw 120 of these squares:

```
to SQUAREMOVE  
repeat 120 [SQUARE100 rt 90 fd 4 lt 90]  
end
```

Exercise 64

Write the program `SQUARE100` and `SQUAREMOVE` in the Editor and try `SQUAREMOVE` out. What will be drawn?

You can see, that the traces of *all* squares are drawn. But for our animation, we would like to see in each step only the last square and we want to erase the previous traces.

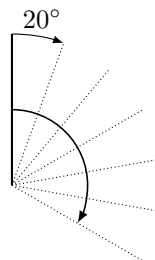


Exercise 65

Make the square move from the bottom to the top instead of from the left to the right.

Exercise 66

Write a program for a line segment of the length 20. Use this program in order to move the line segment clockwise around the lower end:



How to Draw a Square and How to Erase it Again

To cover the tracks, we have to learn how to erase images, which we have just drawn. For this purpose, the turtle has to use an eraser instead of a pen. The turtle changes from the pen mode to the erasing mode with the new command **penerase** or the shorter version **pe**.

Exercise 67

Think about what the program `SQUARE100 pe SQUARE100` does without using the computer.

To make the turtle draw again, we need the new command: **penpaint** or the shorter version **ppt**. We can directly apply the new command in our program of Exercise 67.

The program now looks like this:

```
SQUARE100 pe SQUARE100 ppt
```

Exercise 68

Try out the program above. What will happen? Can you explain it?

How to Make a Square Wait a Bit

As you have surely realized in Exercise 68, the square is erased immediately after it has been drawn. We do not even notice that a square has been drawn. Therefore, before we erase the square, we have to make the computer wait a bit.

We can do this as follows:

<code>wait</code>	4
Wait command	Wait time

Exercise 69

Try out the program:

```
SQUARE100 wait 4 pe SQUARE100 ppt
```

How to Move a Square From Left to Right

Now we can include the commands for waiting and erasing the square into our program **SQUAREMOVE**:

```
to SQUAREMOVE
repeat 120 [SQUARE100 wait 4 pe SQUARE100 rt 90 fd 4 lt 90 ppt]
end
```

Try it out. If the turtle bothers you during the animation, you can start the program with the command **hideturtle** (or shorter: **ht**), which makes the turtle disappear. You will realize that the animation gets faster. End the program with the command **showturtle** (or shorter: **st**) right before the command **end**, which makes the turtle visible again.

Exercise 70

Move a square of size 50×50 upwards.

Exercise 71

Change the program **SQUAREMOVE** so, that the square will move twice as fast to the right as before.

Exercise 72

Are you able to change the program **SQUAREMOVE** in such a way that the square will move half as fast to the right?

Exercise 73

Change the program **SQUAREMOVE** so, that the square moves from the right to the left instead.

Exercise 74

First, try to find out what the following program does and then check your assumption by executing the program:

```
to SQUAREMOVE1
ht
repeat 50 [SQUARE100 wait 5 pe SQUARE100 fd 3 rt 90 fd 3 lt 90 ppt]
SQUARE100
st
end
```

Exercise 75

First, think about what the following program does. Then check if your assumption was right with the computer.

```
to CIRCLES
ht
repeat 360 [SQUARE100 wait 4 pe SQUARE100 fd 5 rt 1 ppt]
SQUARE100
st
end
```

Exercise 76

Modify the program `CIRCLES` in order to make the square turn four times faster.

Exercise 77

What does the following program do?

```
repeat 6 [CIRCLES]
```

Exercise 78

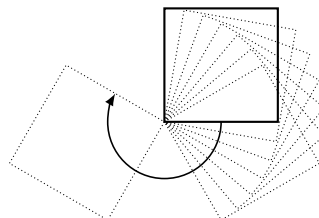
Look at the following program

```
to EARTH
repeat 45 [fd 16 rt 8]
end
```

and use it to draw an animation, in which the Earth turns around the Sun. Use your imagination to represent the Sun.

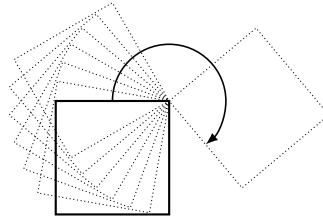
Exercise 79

Turn a square clockwise around its bottom left corner. You can choose the size of the square:



Exercise 80

Now turn a square clockwise around its top right corner:



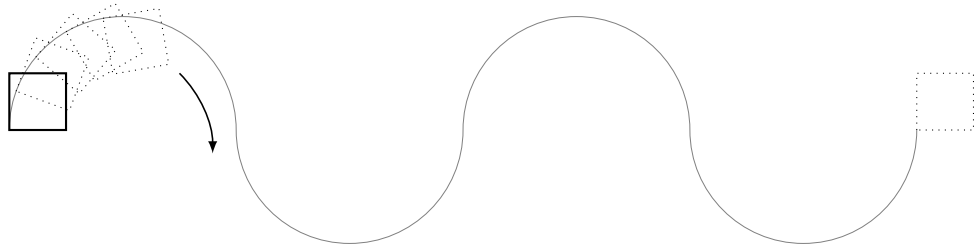
If you already know about Parameters from Lesson 5, you can work on the next exercises.

Exercise 81

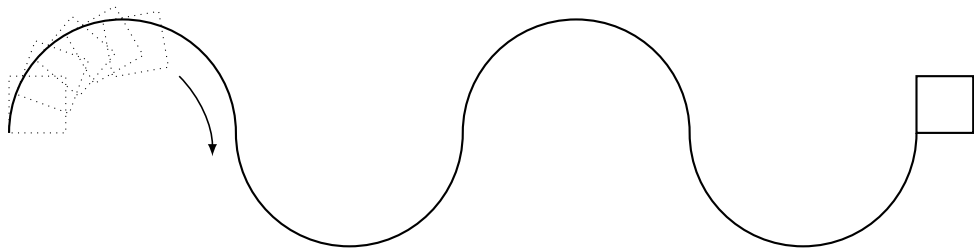
Write a program with *two Parameters* to move a square from left to right. One parameter determines the length of the square, the other parameter is for the speed of the square.

Exercise 82

- (a) Make a square walk along the path shown below, which consists of four half circles. The size of the square should be given by a parameter.

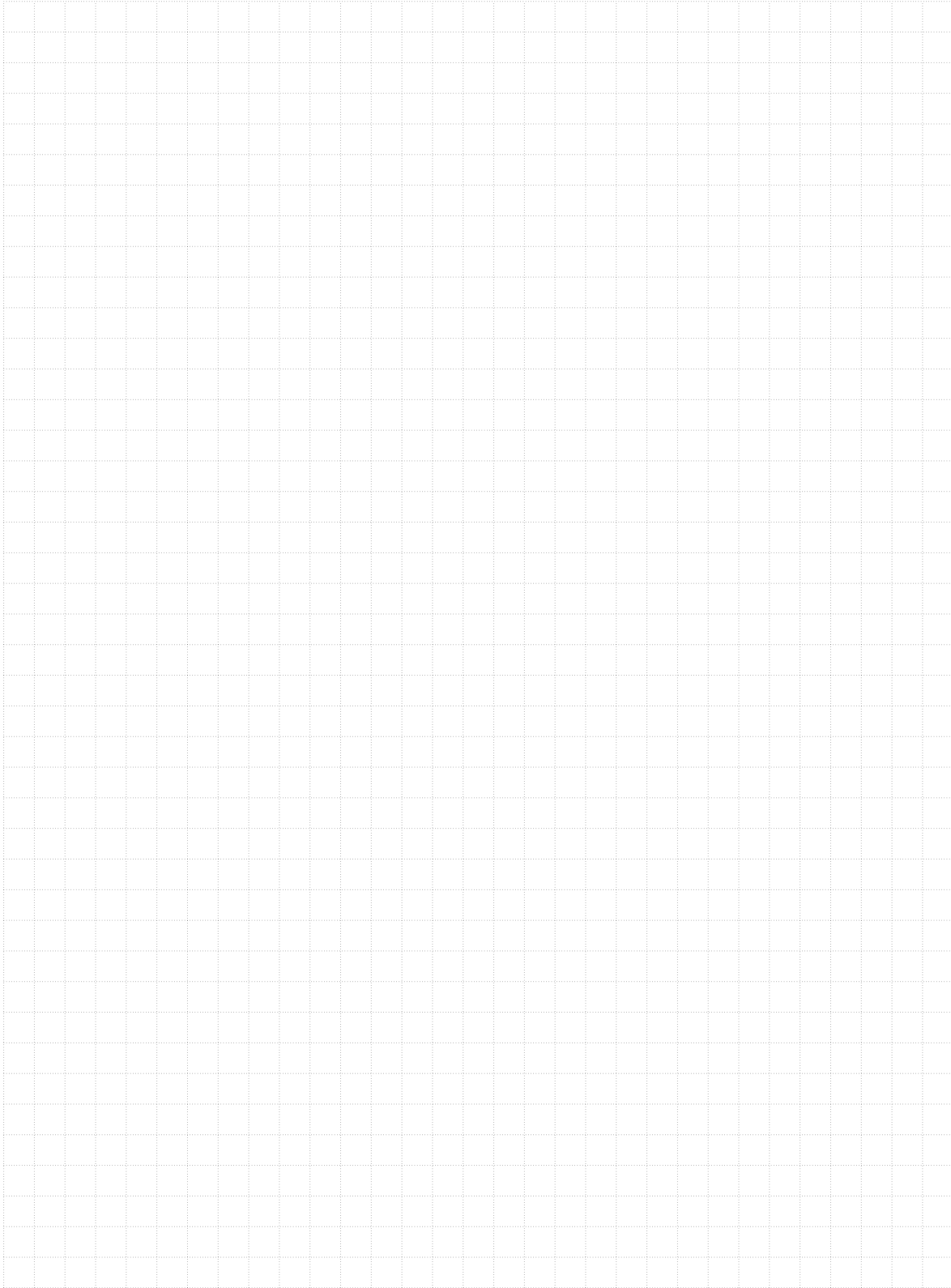


- (b) Now we also want to draw the path along which the square moves.



- (c) Are you able to extend the program in (b) in such a way, that the number of half circles can be given by a parameter?

My Notes



Overview of commands

- fd 100** take 100 steps forward
- bk 50** take 50 steps backwards
- cs** delete all and start again
- rt 90** rotate 90 degrees to the right
- lt 90** rotate 90 degrees to the left
- repeat 4 [...]** the program in [...] is four times repeated
 - pu** the turtle enters the walking mode
 - pd** the turtle returns to pen mode
- setpc 3** changes the pen color to the color 3
- to NAME** creates a program with a name
- to NAME :PARAMETER** creates a program with a name and a parameter
 - end** all programs with a name end with this command
 - pe** the turtle enters the eraser mode
 - ppt** the turtle returns to pen mode from eraser mode
- wait 5** makes the turtle wait 5 units of time



Programming in LOGO

Chair of Information Technology and Education
ETH Zurich, CAB F 15.1
Universitätstrasse 6
CH-8092 Zurich
Switzerland

www.ite.ethz.ch
www.abz.inf.ethz.ch