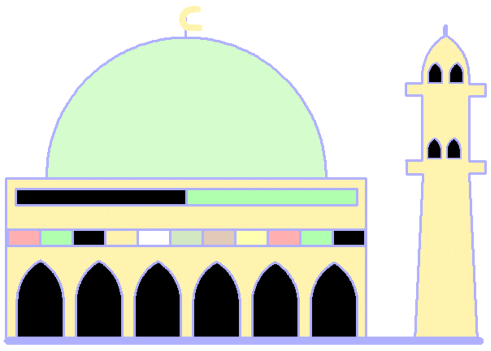
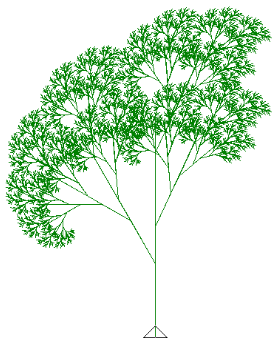
Modul Praktikum

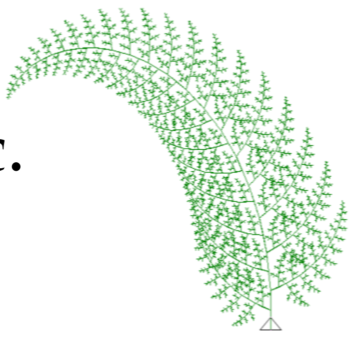


Pemrograman bahasa LOGO

dengan FMSLOGO



Drs. Sahid, MSc.

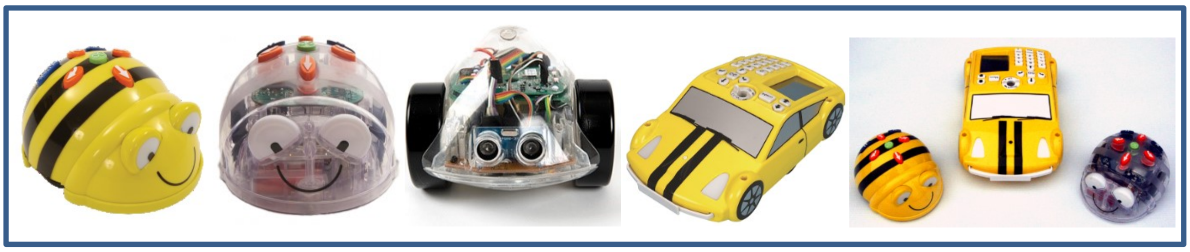


Daftar Isi

Bab 1. Pendahuluan

A. Apakah Logo?

Logo[1] adalah suatu bahasa pemrograman untuk pendidikan yang berorientasi grafis dan logika (berbasis "robot" kura-kura), yang dirancang pada 1967 oleh Daniel G. Bobrow, Wally Feurzeig, Seymour Papert dan Cynthia Solomon dari suatu lembaga riset di Cambridge, Massachusetts yang bernama **Bolt, Beranek and Newman (BBN)**. Landasan intelektual Logo adalah kecerdasasn buatan, logika matematika dan psikologi perkembangan. Logo merupakan bahasa komputer yang kaya kemampuan dan diturunkan dari LISP, yang merupakan bahasa kecerdasan buatan. Akan tetapi, yang lebih penting, Logo adalah bahasa untuk belajar. Tujuan pengembangan Logo adalah untuk menciptakan sebuah papan matematis yang dapat digunakan anak untuk bermain dengan kata dan kalimat. Logo dirancang sebagai perangkat computer untuk riset yang cukup handal, namun cukup sederhana sehingga disukai oleh anak-anak.



Gambar robot-robot kura-kura buatan perusahaan Terrapin Software (Bee-Bot, Blue-Bot, InO-Bot, Pro Bot, masing-masing dilengkapi tombol untuk menggerakkannya).
Sumber: https://www.terrapinlogo.com)

Bahasa ini awalnya disusun untuk memungkinkan apa yang disebut oleh Papert sebagai *"body-syntonic reasoning"* atau "penalaran swa-sintonik" yang artinya, siswa dapat memahami (dan memprediksi dan memikirkan) gerakan kura-kura dengan membayangkan apa yang akan mereka lakukan jika mereka adalah kura-kura. Fitur terbaik Logo adalah "kura-kura grafis" atau kura-kura virtual (berbentuk gambar kura-kura, kursor segitiga, dll.), yang dilengkapi dengan pena pada layar, yang

[1] Nama Logo berasal dari bahasa Yunani *logos* yang berarti *kata* guna membedakannya dari bahasa pemrograman lain yang berorientasi pada bilangan.

mengerti perintah untuk bergerak dan menghasilkan gambar di layar. Menggambar di Logo pada dasarnya dilakukan dengan memberi perintah kepada kura-kura virtual sehingga jejak pergerakannya membentuk gambar yang diinginkan. Dengan fitur seperti ini, anak-anak akan dengan cepat belajar menggerakkan dan mengubah arah kura-kura dengan menggunakan perintah yang mudah diingat dan intuitif.

B. Apa Saja Nilai Pendidikan Logo?

Logo adalah alat yang tepat untuk mengajarkan proses belajar dan berpikir. Logo memberikan lingkungan yang membuat siswa dapat bertindak sebagai guru. Sebagai guru, mereka harus (Terrapin Software, 2013):

a. memahami pengetahuan yang akan diajarkan;
b. merencanakan sebuah pendekatan untuk menyampaikan pengetahuan tersebut;
c. mengurai pengetahuan menjadi bagian-bagian kecil yang mudah dimengerti;
d. mengetahui bagaimana cara mengkomunikasikan pengetahuan secara jelas;
e. membangun pengetahuan baru sebagai dasar pembelajaran selanjutnya;
f. menyadari dan membangun pengetahuan yang sudah dimiliki siswa;
g. mampu menerima gagasan baru; dan
h. menanggapi kesalahpahaman dan kesalahan pembelajar (yaitu komputer).

Siswa melakukan hal-hal tersebut di Logo dengan cara (Terrapin Software, 2013):

a. bereksperimen dengan perintah-perintah Logo untuk memahami maksudnya dan meyakini kegunaan dan penggunaanya,
b. merencanakan tugas dan mengaturnya ke dalam berbagai komponennya,
c. menulis serangkaian perintah untuk melakukan setiap tugas kecil,
d. menulis/menyusun sebuah program untuk melakukan semua tugas dengan urutan yang benar,
e. mengevaluasi program guna menilai apakah tugas dilakukan secara benar, dan
f. mengoreksi/membetulkan program dengan menemukan dan memperbaiki kesalahan atau merestrukturisasi pendekatan yang dipakai.

Penggunaan Logo tidak terbatas pada topik atau subjek tertentu. Bahasa ini sangat berguna untuk mengeksplorasi matematika, karena kura-kura grafis Logo menyediakan lingkungan matematika alami. Karena kura-kura bergerak dalam jarak dan berubah arah dengan derajat, mempelajari geometri dengan menggambar dan menyelidiki poligon dan bilangan (hasil perhitungan) membuat Logo menjadi alat belajar yang hebat.

Di antara sifat alami Logo adalah: ramah, dapat diperluas, pemaaf, fleksibel, dan berdaya guna (Terrapin Software, 2013). Logo mudah dipelajari; kita dapat menghubungkan dengan kura-kura dan menggunakannya sebagai objek untuk dipikirkan. Bahasa ini dapat mengerti perintah baru (perintah atau program yang dittulis oleh pengguna). Logo juga memberikan umpan balik langsung melalui pesan yang bermanfaat dan informatif. Logo dapat dipakai oleh pengguna awam mulai dari anak prasekolah hingga mahasiswa matematika di universitas. Contoh berikut memberi gambaran tentang siapa saja yang menggunakan Logo (Terrapin Software, 2013).

1. Siswa kelas satu di New Hampshire menggunakan Logo versi *single-toystroke* untuk memindahkan kura-kura dan mengeksplorasi bangun-bangun geometri dan garis.
2. Siswa kelas dua dan tiga di Arizona belajar tentang poligon menggunakan Logo.
3. Siswa kelas empat di California memprogram permainan *golf miniatur* di Logo.
4. Siswa kelas lima di Massachusetts mempelajari geografi negara mereka dengan menggambar peta di Logo.
5. Siswa sekolah menengah di Kentucky menggunakan Logo untuk mengendalikan robot.
6. Siswa perempuan tingkat SMA menaklukkan "fobia matematika" dalam sebuah program musim panas di Mount Holyoke College di Massachusetts dengan menggunakan Logo.
7. Siswa sekolah menengah di California menggunakan Logo untuk membuat plot tersebar di kelas statistik mereka.
8. Siswa sekolah menengah di Colorado belajar memprogram menggunakan Logo.
9. Siswa lumpuh di Pennsylvania menggunakan perangkat *single-switch* dengan Logo untuk memindahkan kura-kura dan membuat desain.
10. Seorang profesor di Massachusetts Institute of Technology menggunakan versi khusus Logo untuk mengajarkan teori musik.
11. Instruktur di University of California di Berkeley mengajar siswa ilmu komputer bagaimana memprogram menggunakan Logo.
12. Siswa di seluruh dunia menggunakan Logo untuk belajar.

Sebagai tambahan, penulis juga telah memperkenalkan bahasa pemrograman Logo kepada para mahasiswa Matematika dan Pendidikan Matematika di Jurusa

Pendidikan Matematika FMIPA UNY sebagai bagian dari kuliah Algoritma dan Pemrograman. Penulis juga memperkenalkan Bahasa Logo kepada guru-guru Matematika dari negara-negara ASEAN peserta pelatihan di SEAMEO QITEP in Mathematics Yogyakarta.

Situs web Perangkat Lunak Terrapin (http://www.terrapinlogo.com/why-use-logo.php, 2013) menyebutkan beberapa laporan dari para guru yang mengatakan bahwa Logo menawarkan kepada siswa banyak keuntungan yang terukur (pemahaman sudut, peningkatan keterampilan estimasi, dll.) dan Logo juga membantu siswa dengan pengembangan pribadi, sikap terhadap pembelajaran, kedalaman pemahaman, kreativitas, dan manfaat jangka panjang lainnya. Para guru antusias mengajar dengan Logo, dan siswa belajar dengan Logo. Logo memberi siswa:

- kesempatan untuk menjadi guru, bukan pembelajar pasif, dan untuk mengendalikan komputer;
- umpan balik langsung, visual, dan tidak menghakimi;
- kesempatan untuk mengeksplorasi dan mencoba ide-ide mereka sendiri;
- kesempatan untuk menemukan konsep sendiri;
- kesempatan untuk mengeksplorasi konsep yang biasanya diajarkan jauh di kemudian hari;
- sarana belajar dan memahami dengan langkah mereka sendiri;
- kesempatan untuk menganalisis tugas dan merencanakan serangkaian instruksi untuk menyelesaikannya;
- model untuk menyelesaikan masalah dengan memecahnya menjadi masalah-masalah yang lebih sederhana;
- kesempatan untuk berhasil di bidang-bidang di mana mereka mungkin merasa tidak nyaman di lingkungan yang tidak mengancam; dan
- kegembiraan dalam belajar.

Situs web Perangkat Lunak Terrapin tersebut juga menyebutkan bahwa berdasarkan penelitian, siswa yang menggunakan Logo:

- merencanakan sesuatu secara lebih efisien,
- menyajikan tugas-tugas perencanaan secara berbeda,
- memiliki pemahaman geometri yang meningkat,
- tekun dalam menyelesaikan masalah,
- lebih baik dalam menyelesaikan konflik,
- lebih mandiri, dan

- menunjukkan interaksi sosial yang diinginkan.

Selanjutnya, situs web Perangkat Lunak Terrapin mengutip ringkasan Doug Clements pada hasil penelitian Logo (Logo Exchange, Januari 1988) sebagai berikut:

> *"Potensi logo untuk mengembangkan ide-ide geometris akan terpenuhi sejauh guru membantu membentuk pengalaman Logo siswa mereka. Siswa tidak secara otomatis mentransfer pengetahuan yang diperoleh dalam satu situasi ke situasi lain. Pengulangan tidak cukup. Pertanyaan yang menyebabkan siswa merefleksikan apa yang mereka lakukan sangat berperan."*

Akhirnya situs web Perangkat Lunak Terrapin menyimpulkan bahwa:

> *"Guru berperan sangat penting bagi keberhasilan siswa. Kunci untuk belajar adalah guru. Logo adalah alat yang baik, tetapi satu-satunya alat yang digunakan oleh kebanyakan siswa untuk meningkatkan potensi mereka sendiri. Guru yang baik melibatkan siswa dalam belajar, memberi mereka pilihan, menciptakan lingkungan yang membantu, dan mengubah kesalahan menjadi peluang untuk dieksplorasi. Logo adalah alat yang sempurna untuk para guru ini, karena desainnya membuatnya ramah, dapat diperluas, pemaaf, fleksibel dan kuat. Atribut ini dibangun ke dalam bahasa Logo, menjadikan Logo sebagai alat yang sangat ampuh untuk belajar. "*

Sebagai bahasa pemrograman yang sederhana, Logo menyediakan semua alat yang dibutuhkan untuk membuat program dengan tingkat kecanggihan apa pun. Logo mudah dipelajari, namun menawarkan kekuatan kepada pengguna yang berpengalaman; dikatakan memiliki "tidak ada ambang batas dan tidak ada langit-langit." Dengan demikian, siswa yang dapat membaca dapat belajar dengan Logo, sementara pengguna komputer yang berpengalaman menemukan bahwa Logo menyediakan lingkungan pemrograman yang kuat dan fleksibel. Oleh karena itu, Logo tidak hanya bahasa pemrograman, tetapi juga bahasa untuk belajar; bahasa yang mendorong siswa untuk mengeksplorasi, belajar, dan berpikir.

Menggunakan Logo tidak hanya memindahkan kura-kura atau menerapkan matematika, tetapi juga memikirkan proses - bagaimana Anda melakukan apa yang Anda lakukan. Dalam Logo, proses membuat produk (seperti grafik) lebih penting daripada hasil akhir. Lebih menarik dan mendidik untuk melihat bagaimana grafik atau gambar dibuat daripada melihat grafik atau gambar itu sendiri.

As a simple programming language, Logo provides all the tools needed to create programs of any degree of sophistication. Logo is easy to learn, yet offers power to the experienced user; it is said to have "no threshold and no ceiling." Thus, students who are able to read can learn with Logo, while experienced computer users find that Logo provides a powerful and flexible programming environment. Therefore, Logo is not only a programming language, but also a language for learning; a language that encourages students to explore, to learn, and to think.

Using Logo is not just moving a turtle or applying mathematics, but also thinking about processes – how you are doing what you are doing. In Logo, the process of creating a product (such as a graphics) is more important than the finished result. It is more interesting and educational to look at how a graphics or a picture was created than to look at the graphics or picture itself.

C. How to Use Logo in Mathematics Classroom?

Logo can be integrated into mathematics curriculum to teach students how to think and discover. Using Logo in the mathematics classroom is very easy. The focus in most classrooms is not on programming, but on the thinking processes that Logo encourages. The following table show several topics in mathematics and other areas that can be done in Logo.

Table 1 Some Area of Educational Uses of Logo

Area	Topics	How to do it
Mathematics	*estimation*	working with distances and angles
	polygons	using REPEAT to create regular shapes
	perimeter and area	investigating number relationships
	symmetry	drawing with point and line symmetry
	coordinates	plotting points and graphing lines
	probability	using Logo's random number generator
	functions	writing functions that output values
	algebra	graphing linear and quadratic equations
	geometry	drawing and measuring lines and angles
	trigonometry	using Logo's sine and cosine functions
	fractals	combining graphics and recursion
Programming	*proper techniques*	writing structured programs
	program design	breaking down a problem into smaller tasks
	flow of control	learning about branching and conditionals

Area	Topics	How to do it
	variables and recursion	using the power of the language
	data handling	manipulating numbers, words and lists
Language Arts	*sentence structure*	generating Logo sentences that follow the rules of grammar and parts of speech
	creative writing	writing and illustrating poems
	word structure	writing programs that rhyme words, pluralize nouns, and conjugate verbs
Social Studies	*directions*	translating the turtle's heading into compass points
	cartography	making maps using Logo graphics
	foreign languages	creating foreign language command names
Science	*robotics*	controlling robotics devices through Logo
	sensors	attaching light and touch sensors to the computer and reading the output
	simulations	running physics experiments
Fine Arts	*computer art*	using Logo's graphics capabilities
	music	using Logo's sound-generating abilities
	dance	choreographing the turtle
	multimedia	capturing Logo graphics on videotape playing video through Logo

Teachers takes most important role to students' learning. They may use several approaches to teaching Logo, or guiding or facilitating students' Logo learning. When students want or have to create a challenging task, the teacher can serves as a facilitator, helping them understand new Logo commands that might be useful; suggesting different approaches to do the task; helping determine where and why things are not going as expected; and offering support, suggestions and encouragement, if needed. Using Logo requires participations and interaction between student and teacher and between student and student. Logo author and educator Donna Beardon explains what is needed (Terrapin Software, 2013):

> *"Effective Logo teachers walk a fine line between free exploration and preplanned curricula: Logo is powerful, but Logo is not magical. Insights are powerful, but insights are not magical. Insights occur as a result of trial and thinking. Insights are what Logo is so good at promoting if the ingredients are all there. These ingredients include:*
> *1. steps toward understanding;*
> *2. significant problems that challenge and entice students to think hard;*

3. time to get in tune with the activity."

The learner of Logo must be able to discover important concepts by knowing enough information given by the teacher. When students discover concepts, they "own" that knowledge and they will have a much deeper understanding than if they are told or shown by the teacher. To facilitate students' learning with Logo, projects should be organized as real-life projects, where there is constant communication with peers and with facilitators. Students need to work together, solve new problems, and learn to work cooperatively. Learning with Logo, they are encouraged to share their knowledge, insights, discoveries, and strategies. In addition, students may record in journals how they chose their project, new commands that they learned during the project, important discoveries that they made, problems that they ran into and how they solved them, and how they tested and debugged their program. After finishing a project students may present the result to share with others within the school or with students in a pen-pal school, creating bulletin board displays and library exhibits, writing articles for school or local newspapers, making a parent's night presentation, or sending projects to computer education magazines and journals for publication.

Bab 2. Using LOGO with FMSLogo

Now is the time to start learning and using Logo in mathematics teaching and learning! However, before you can use Logo in mathematics teaching and learning you must first learn how to use Logo. Logo is the ideal language to the first learner who wants to create computer program, which is to command computer to do something he like. It is because the following reasons.

1. *Logo is an easy language to learn.*
    - The commands are English-like and easy to remember.
    - There are friendly and helpful error messages.
    - It is interactive, so the user gets immediate feedback.
2. *Logo promotes good programming habits.*
    - It encourages the development of short (simple) programs (procedures).
    - Procedures are then used as building blocks by other procedures.
    - There are no line numbers that encourage jumping around the code.
3. *Logo allows users to create and reuse procedures.*
    - As an extensible language, Logo allows users to add new commands.
    - Users can run and test procedures independently.
    - It is easy to use procedures from one program in another.
    - It encourages the creation of a library of often-used procedures.
4. *Logo programs are easy to write and maintain.*
    - Programs are easy to debug with tracing and pausing tools.
    - Users can use long and descriptive variable and procedure names.
    - Error messages are detailed and helpful.

There is much computer software that implements Logo language, ranging from commercial one like Microworld to freeware like FMSLogo. FMSLogo is a free implementation of Logo for Windows system operation. FMSLogo can be downloaded from the project portal on SourceForge (http://sourceforge.net/projects/fmslogo). FMSLogo includes support for: "standard" Logo parsing, turtle graphics, text in all available system fonts, 1024 independent turtles, bitmapped turtles, saving and loading images in BMP format, creating windows dialog boxes, event driven programming (mouse, keyboard, timer), controlling multimedia devices (WAV sound files, CD-ROM control, MIDI devices, etc.), 3D perspective drawing (wire-frame and solids), and creating animated GIFs.

In addition, FMSLogo has the following features.

1) FMSLogo has a simple GUI that encourages learning.
2) FMSLogo provides support for exploring diverse disciplines, including mathematics, engineering, art, music, and robotics.
3) FMSLogo runs fine on computers that are 10 years old.
4) FMSLogo has a strong, international user community with over a decade of classroom experience.

These features enable FMSLogo as ideal computer software introduced to elementary school students through which they can learn how to command computer and crate interesting graphics. The following tips and notes may useful in learning and teaching Logo language.

- ✓ Logo does not distinguish CAPITAL and small letters for all commands and variables.
- ✓ Everything preceded by (or written behind) semicolon (;) will be ignored by LOGO, meaning that it is just a comment.
- ✓ Many commands have short abbreviations, such FD for FORWARD, BK for BACKWARD, RT for RIGHT, LT for LEFT, PU for PENUP, PD for PENDOWN, and so on. For clarity, all code samples use the full command name. However, slow typers like children; they should use the short version.
- ✓ Logo is unforgiving of typos. If you type "FORWORD 10", Logo will display an error message like "I don't know how to FORWORD". You may feel that this negative reinforcement can damage student's motivation. Please remember that the error message is because *the computer* (neither the student nor you) is so stupid that it can't even understand that by "FORWORD" we mean "FORWARD". By remembering this will make students never give up whenever see such kind of error message.
- ✓ Every time you create original artwork, it is a good idea to copy the commands or programs that created the artwork and the artwork into a word processor like MS Word, for example, or prints them. This is a way to reward for thinking of something original and will be helpful for next creation.
- ✓ Please feel free to do whatever you want with Logo, instead of always following certain written instruction. For beginning, following written instructions or examples is okay, but for the next you must be able to create your own instruction for Logo to create some pictures or graphics.

The next parts explain how to learn Logo using FMSLogo.

1. Getting Started with Turtle Graphics

In this session you will learn the following things in FMSLogo.

1. The turtle follows your instructions and only your instructions. The turtle draws a line wherever it walks.
2. Command FORWARD (or FD) makes the turtle go forward and BACKWORD (or BK) makes it go backward. Each command is followed by a number determining how far the movement.
3. Command RIGHT (or RT) makes the turtle turn right in place and LEFT (LT) makes it turn left. The number that follows it is in degrees, such as 90 for a quarter-turn.
4. Command PENUP (or PU) makes the turtle movement results in no line and PENDOWN (or PD) makes the turtle movement results in a line.
5. Command SHOWTURTLE (or ST) make the turtle seen and HIDETURTLE (or HT) hides the turtle.
6. Command REPEAT to instruct the turtle do the same command(s) several times.

After installing FMSLogo, please start it as starting other Windows program. When FMSLogo start up on your computer, you will see the FMSLogo screen on your desktop like **Figure 1**.

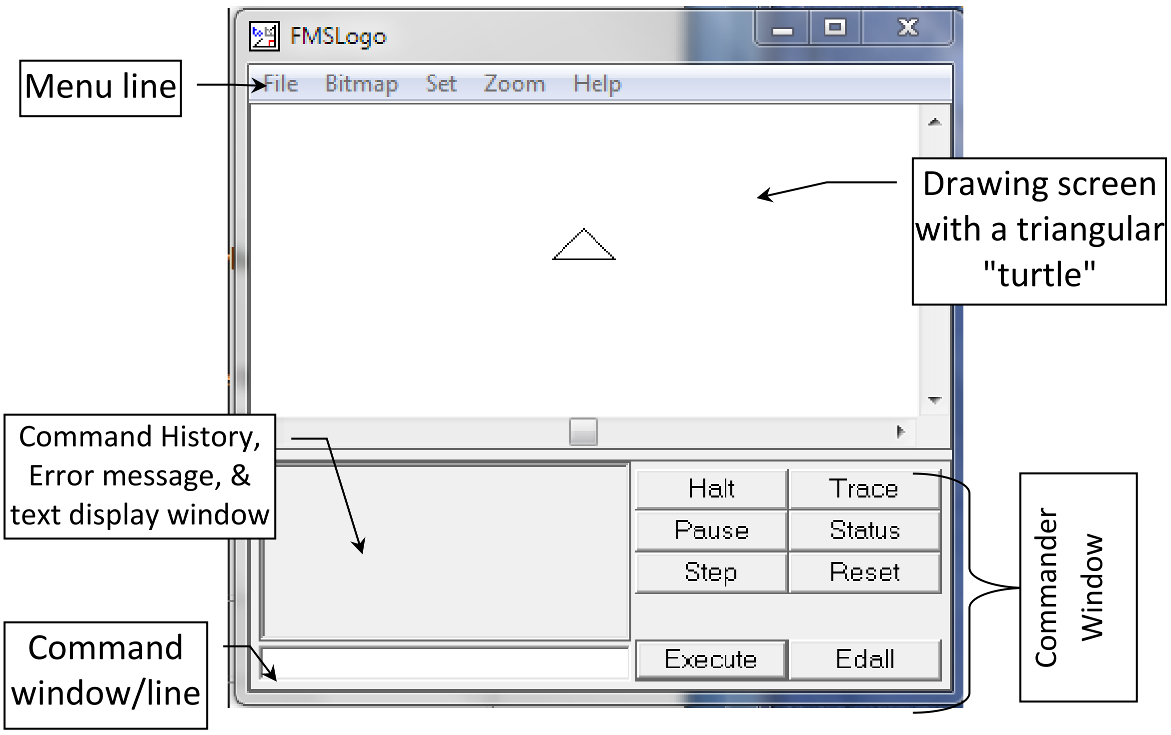

FMSLogo
Menu line
File Bitmap Set Zoom Help
Drawing screen with a triangular "turtle"
Command History, Error message, & text display window
Halt
Trace
Pause
Status
Step
Reset
Execute
Edall
Commander Window
Command window/line

Figure 1 FMSLogo Screen

The FMSLogo screen consists of several parts: title (FMSLogo), menu line (File, Bitmap, Set, Zoom, Help), drawing screen/window, command history/error message/text display window, command window/line, and a collection of commander buttons (**Halt**, **Pause**, **Step**, **Trace**, **Status**, **Reset**, **Execute**, **Edall**). Please explore the available menu, and if you are getting confused just select the Help menu. The drawing screen is a place where the turtle (Δ) (a triangular cursor) stays. The turtle has a "hidden pen", so he draws a line wherever he goes. This means you can draw pictures just by telling him to walk around. Under the drawing window there is a grey window (output box) which is the place where command history (previous commands you typed), error messages, textual output and texts are displayed. You can select an instruction on the output box to be re-run by clicking on the desired line or by using the arrow keys. When a line is clicked, it is automatically copied to the input box, and then you can edit it. Double-clicking on a line in the output box runs it as a Logo instruction. Under the output box there is a command window or input box, where you enter instructions for turtle. To have Logo run the instruction you typed, press the **ENTER** key or the **Execute** button. Pressing the **Up/Down** arrow keys automatically jumps to the output box.

Here are some basic instructions that the turtle already understands.

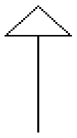
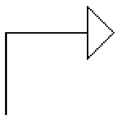
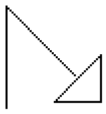
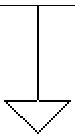
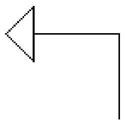
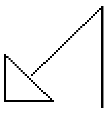
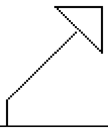
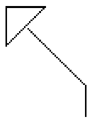
Table 2 Some Basic LOGO Commands

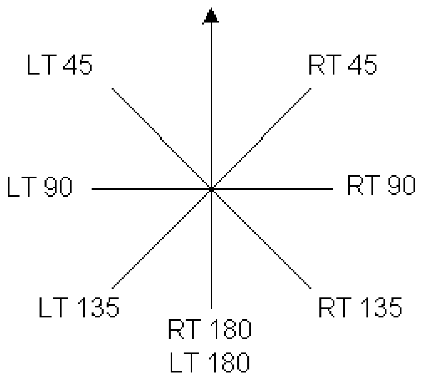
Procedure	**Example**	**What Happens**
FORWARD *number* (or FD *number*)	FORWARD 100 (or FD 100)	The turtle walks forward 100 screen dots.
BACK *number* (or BK *number*)	BACK 150 (or BK 150)	The turtle walks backward 150 screen dots.
RIGHT *number* (or RT *number*)	RIGHT 90 (or RT 90)	The turtle turns to the right.
LEFT *number* (or LT *number*)	LEFT 90 (or LT 90)	The turtle turns to the left.
CLEARSCREEN (or CS)	CLEARSCREEN (or CS)	The turtle erases everything that he has drawn and goes back to where he started.
PENUP (or PU)	PENUP (or PU)	After this command, the next turtle moves will result in no lines.
PENDOWN (or PD)	PENDOWN (or PD)	After this command, the next turtle moves will result in lines.
SHOWTURTLE (or ST)	SHOWTURTLE (or ST)	Display the turtle.
HIDETURTLE (or HT)	HIDETURTLE (or HT)	Hide the turtle (this does not mean PENUP).
CLEARTEXT (or CT)	CLEARTEXT (or CT)	Clears all text within the command history window

WAIT *delay*	WAIT 120	Delays further execution for about 2 second (120/60) second.

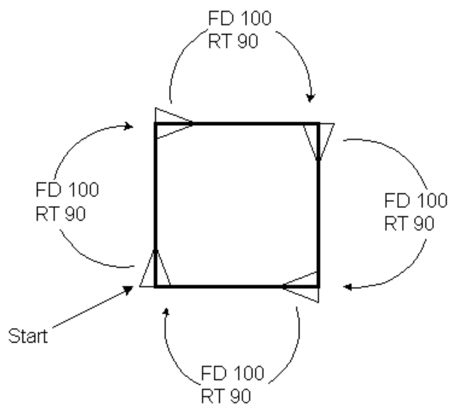
The value of number after command FORWARD or FD and BACKWARD or BK determine how far (in pixel units or screen dots) the turtle will walk. The value of number after command RIGHT or RT and LEFT or LT is the number of "degrees" the turtle will rotate. Positive degree is clockwise, and negative degree is counterclockwise.

Here are some pictures showing the results of using of the commands RIGHT (RT) and LEFT (LT).

RIGHT 0 or RIGHT 360 or LEFT 0 or LEFT 360		RIGHT 90 or LEFT 270		RIGHT 135 or LEFT 225	
RIGHT 180 or LEFT 180		RIGHT 270 or LEFT 90		RIGHT 225 or LEFT 135	
RIGHT 45 or LEFT 315				RIGHT 135 or LEFT 45	



Example:



To get the turtle to draw a shape, like a square, you can give him the instructions he needs to "walk" around the shape of a square, as illustrated on the picture beside. The steps to draw a square are as follows.

1) Draw a line segment.
2) Turn right.
3) Draw the second line segment of the same length.

4) Turn right.
5) Draw the third line segment of the same length.
6) Turn right.
7) Draw the last line segment of the same length.
8) Turn right to get the initial position of the turtle.

Activity:

1. Write and enter the instructions to make the turtle draw a square.
2. After you succeed, clear the drawing screen.
3. Write and enter the instructions to make the turtle draw a square but with different size from before.

The series of commands you wrote may look like as follows.

```
FORWARD 75
RIGHT 90
FORWARD 75
RIGHT 90
FORWARD 75
RIGHT 90
FORWARD 75
RIGHT 90
```

or

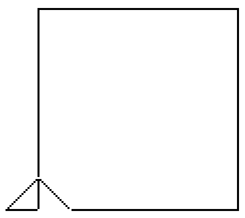
```
FD 75
RT 90
FD 75
RT90
FD 75
RT 90
FD 75
RT9
```

That was pretty good, but it was a lot of typing of the same commands. To ask the turtle do the same commands several times, we can use the REPEAT command.

Procedure	**Example**	**What Happens**
REPEAT *number* [*instructions*]	REPEAT 4 [FORWARD 100 RIGHT 90]	The turtle does the following instructions four times: moves forward 100, turns right 90°. The result in a square (when in **pendown** mode)

Instructions inside the square bracket will be done as much as the numbers after REPEAT command. With the REPEAT command, we can draw a square with just two non-repeating instructions.

```
REPEAT 4 [FORWARD 100 RIGHT 90]
```

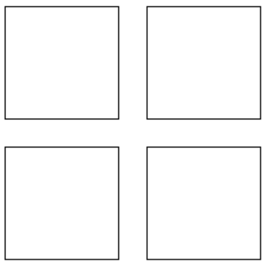


Activity:

1. Use REPEAT, FORWARD, and RIGHT commands with different numbers, and see what the result is. Clear the screen every time you want to use different numbers.
2. Use REPEAT, FORWARD, and RIGHT commands to draw an isosceles triangle.
3. Use REPEAT, FORWARD, and RIGHT commands to draw a pentagon (regular 5 sides).
4. Use REPEAT, FORWARD, and RIGHT commands to draw a hexagon (regular 6 sides).
5. Use REPEAT, FORWARD, and RIGHT commands to draw a heptagon (regular 7 sides).
6. Use REPEAT, FORWARD, and RIGHT commands to draw an octagon (regular 8 sides).
7. Use REPEAT, FORWARD, and RIGHT commands to draw a circle. If you succeed, draw circles with different radii.

So far you have seen that every time you move the turtle, it will result in a line. To move the turtle without drawing a line, first you must lift the turtle's pen up, using the PENUP or PU command. This is similar to using a pen, when it is lift up, it cannot be used to write or draw. To write or draw with a pen, it must be lift down. Similarly, to make the turtle draw a line while it moves its pen must be lift down with the PENDOWN or PD command. This is the default mode. To understand more about these commands, see what happen in the drawing screen with the following instructions.

```
CS
REPEAT 4 [BK 100 RT 90] PU FD 25 PD
REPEAT 4 [PD 100 LT 90] PU RT 90 FD 25 PD
REPEAT 4 [FD 100 LT 90] PU RT 90 FD 25 PD
REPEAT 4 [FD 100 LT 90]
```

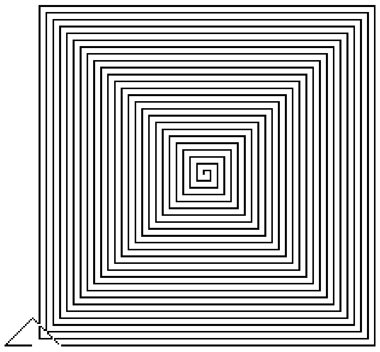


```
HT
```

See what is the different if the commands PU and PD never used in the above instructions. Try it! If you want to see the turtle again, type the command ST.

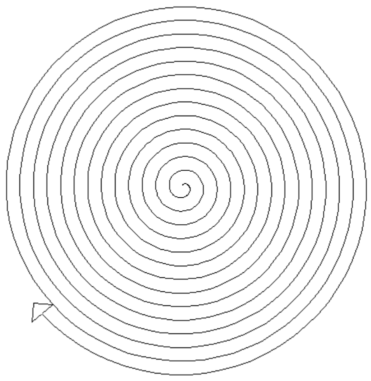
The REPEAT command automatically uses the REPCOUNT counting variable to store how many times the repetition has be done. This variable is very useful for creating more interesting drawing. With REPCOUNT the looping command REPEAT can be used to draw pictures with parts that look similar, but are NOT exactly the same. For example, to draw a square spiral, you can start at the center and repeatedly draw the edges of a square, making each edge a little longer than the previous one. You do this in Logo with the REPCOUNT loop counter. REPCOUNT is always equal to the number of times the loop has been run. It is 1 the first time the loop is run, 2 on the second, 3 on the third, and so on. Here is a program that uses REPCOUNT to draw a square spiral.

```
REPEAT 100 [FORWARD REPCOUNT*2 RIGHT 90]
```



Activity:

1. Type in the above Logo instructions to draw the square spiral as seen above.
2. Modify numbers in the above instructions to draw a triangle spiral.
3. Pick some other number for the angle and see what happens.
4. The square spiral multiplied the FORWARD amount by 2. What do you think happens if you change the 2 to a 3? Try it and see if you're right.
5. Draw a pentagonal spiral.
6. Draw an hexagonal spiral.
7. Draw an heptagonal spiral.
8. Draw a 12 sides spiral.
9. Can you draw a circular spiral as picture below? Do trial and error until you succeed.



Review & Challenge Questions

1) What command moves the turtle forward?
2) What command moves the turtle backward?
3) What command turns the turtle to its right?
4) What command turns the turtle to its left?
5) How many degrees do you turn to make a corner of square?
6) How do you repeat the same command over and over again?
7) What happens if you give FORWARD a really big number?
8) What happens if you give RIGHT a really big number (trick question!)?
9) How do you teach the turtle a new command?
10) If you have a bunch of commands that draws a shapes, how can you change those commands to make it draw the same shape twice is large?
11) Why is asking the turtle to turn by an amount greater than 359 unnecessary?
12) How can you draw something with curvy lines, like a circle?
13) How can you turn left by using the RIGHT command?
14) How can you go backwards using just the FORWARD command?
15) Why is giving two move instructions in a row unnecessary?
16) Why is giving two turn instructions in a row unnecessary?
17) How can you create a solid shape, like a square whose insides are completely black?

18) How can you create concentric polygons with different length of sides, like concentric squares, concentric triangles, concentric pentagons, etc?

2. Turtle and Its Pen Properties

a. Multiple Turtles

FMSLogo supports multiple turtles. Each turtle maintains its own heading, position, and pen control. Turtles can be individually represented as a picture using BITMAPTURTLE. By default, all turtles share the same pen mode, pen color, and pen size, but you can make individual turtles have their own pen by supplying an optional input to SETTURTLE.

Table 3 Some Multiple Turtle Commands

Multiple Turtle Commands	**Description**
SETTURTLE *index* (SETTURTLE *index* *hasownpen*)	Selects turtle *index* for control. Turtles start at index 0, which is the default turtle (0 ≤ *index* ≤ 1023). If the turtle was not previously activated, then all turtles from zero to *index* are activated, so don't choose turtle 100 and 200 if only 2 turtles are needed. Having thousands of active turtles may slow FMSLogo down considerably as it pauses to redraw every turtle after every instruction. From now until a different turtle is selected, all turtle commands refers to the newly selected turtle. Pressing the **Reset** button or running CLEARSCREEN will select turtle 0 and deactivate all other turtles. If the optional boolean *hasownpen* input is supplied as "true, then the selected turtle, as well as all newly activated turtles, will have their own PENMODE, PENSIZE, and PENCOLOR. If no *hasownpen* input is given and SETTURTLE does not activate any new turtles, then the selected turtle's pen remains unchanged. If no *hasownpen* input is given and SETTURTLE activates a new turtle, then the turtle's initial pen uses the initial pen that FMSLogo uses.
SETTURTLEMODE ***mode***	Sets the current bitmap mode for the currently selected turtle to *mode*. The *mode* input must be a number from 1 to 9. The meaning of each value is given by the following table: **Mode** | **Meaning** 1 | Take copy of memory then copy to screen

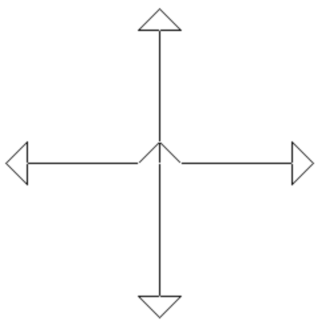
2	Take copy of memory OR with copy of screen then copy to screen
3	Take copy of memory AND with copy of screen then copy to screen
4	Take copy of memory XOR with copy of screen then copy to screen
5	Take copy of memory AND with invert of screen then copy to screen
6	Take invert of memory then copy to screen
7	Take copy of memory OR with copy of screen then invert to screen
8	Take invert of memory OR with copy of screen then copy to screen
9	Take invert of screen then copy to screen

SETTURTLEMODE only works if the selected turtle is currently bitmapped with BITMAPTURTLE. It enables bitmapped turtles to paste their images in different ways.

BITMAPTURTLE	Maps the current turtle to the corresponding bitmap buffer. BITMAPTURTLE can map turtle 0 to bitmap 0, turtle 1 to bitmap 1, turtle 2 to bitmap 2, etc., on a per-turtle basis.
NOBITMAPTURTLE	Restores the turtle to its normal/original triangle shape (it does not erase the corresponding bitmap.)
ASK *turtle instructionlist*	Selects to the turtle whose index matches the *turtle* input, runs *instructionlist*, then reselects the previous turtle

Example:

```
REPEAT 4 [ASK REPCOUNT [RIGHT 90*REPCOUNT FORWARD 100]
```



Please explain the above instructions and its result.

b. Turtle, Pen and Window Control

Some basic turtle and window control commands have been explained at the beginning. Here are more commands to control the turtle and FMSLogo window.

Table 4 Some Turtle, Pen & Window Control Commands

Control Commands	Description
SHOWTURTLE ST	Makes the turtle visible
HIDETURTLE HT	Makes the turtle invisible
WRAP	Tells the turtle to enter "wrap" mode. This is the turtle's initial mode. From now on, if the turtle is asked to move past the boundary of the graphics window, it will "wrap around" and reappear at the opposite edge of the window. The top edge wraps to the bottom edge, while the left edge wraps to the right edge.
WINDOW	Tells the turtle to enter "window" mode. From now on, if the turtle is asked to move past the boundary of the graphics window, it will move off screen. The visible graphics window is considered as just part of an infinite graphics plane; the turtle can be anywhere on the plane. If you lose the turtle, HOME will bring it back to the center of the window.
FENCE	Tells the turtle to enter "fence" mode. From now on, if the turtle is asked to move past the boundary of the graphics window, it will move as far as it can and then stop at the edge with an "out of bounds" error message.
FULLSCREEN FS	Undocks the commander window from the screen window so that you can resize it to maximize the space available for graphics. The commander window will always be on top of the screen window, so you may find it useful to minimize the commander window after undocking it.
TEXTSCREEN TS	Undocks the commander window from the screen window so that you can resize it to maximize the space available for text interaction.
SPLITSCREEN SS	Docks the commander window and the screen window inside the same frame and separates the two with a movable splitter bar. This leaves some room for text interaction while also keeping most of the graphics window visible. This is the default configuration.
CLEAN	Erases all lines that the turtle has drawn on the graphics window. The turtle's state (position, heading, pen mode, etc.) is not changed.
CLEARSCREEN CS	Removes all turtles created with SETTURTLE so that there is only one turtle and fills the drawing screen with SCREENCOLOR and sends the turtle to its initial position (center) and heading.

Control Commands	**Description**
SLOWDRAW *slowness*	Redefines FORWARD, BACK, FD, and BK to take longer so that you can see the turtle as it moves (otherwise, the turtle moves as fast as it can). The *slowness* input controls how long it takes for turtle to complete each movement instruction. The larger the *slowness* input, the slower the turtle moves. Running SLOWDRAW 0 restores FORWARD, BACK, FD, and BK to their original speed. The *slowness* input must be a non-negative integer.
SETPIXEL *colorvector*	Sets the pixel under the turtle to the color described by *colorvector* (a three item list of [red green blue] intensities, each of which must be an integer from 0 to 255). SETPIXEL is similar to running SETPENSIZE 1 SETPENCOLOR colorvector PENPAINT FORWARD 0, except that SETPIXEL is faster and doesn't affect the pen state.
PENDOWN PD	Sets the pen's position to "down", without changing its mode
PENUP PU	Sets the pen's position to "up", without changing its mode
PENPAINT PPT	Sets the pen's position to "down" and the pen's mode to "paint". In paint mode, the pen draws in the color described by PENCOLOR. This is the default mode.
PENERASE PE	Sets the pen's position to "down" and the pen's mode to "erase". In erase mode, the pen draws in the color described by SCREENCOLOR.
PENREVERSE PX	Sets the pen's position to "down" and the pen's mode to "reverse". In reverse mode, the pen draws in a color that is the bitwise negation of whatever pixel is under the pen.
PENNORMAL	Sets the pen's state to the startup default of down and paint
SETPENSIZE *size*	Set the pen's thickness. The *size* input must be either a positive integer that denotes the pen's width.

c. Pen Color, Screen Color, and Fill Color

Logo gives two ways to choose a color. One way is to pick a color from a set of 16 commonly used colors. Another way is to create a combination of different amounts of red, green, and blue. The latter way gives a choice of over 16 *million* different colors.

To change the color of the lines that the turtle draws use the SETPENCOLOR (or SETPC) command. To change the color of the screen (or background) use the SETSCREENCOLOR (or SETSC) command. These commands require the color to use. There are two ways to determine the color. The first is using color index. The other way is using a combination of different amounts of basic colors (red, green, and blue or [R G B]).

Table 5 Color Indexes, Names, and [R G B] Codes

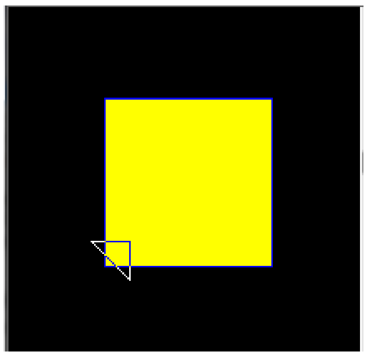
Color Index	Color Name	[R G B]	Color
0	black	[0 0 0]	⬛
1	blue	[0 0 255]	🟦
2	green	[0 255 0]	🟩
3	cyan (light blue)	[0 255 255]	🟦
4	red	[255 0 0]	🟥
5	magenta (reddish purple)	[255 0 255]	🟪
6	yellow	[255 255 0]	🟨
7	white	[255 255 255]	⬜
8	brown	[155 96 59]	🟫
9	light brown	[197 136 18]	🟧
10	dark green	[100 162 64]	🟩
11	darkish blue	[120 187 187]	🟦
12	tan	[255 149 119]	🟧
13	plum (purplish)	[144 113 208]	🟪
14	orange	[255 163 0]	🟧
15	gray	[183 183 183]	▨

Every color is made up of red, green, and blue. Taken together, red, green, and blue are called "the primary colors" (sometime called RGB colors) because every color can be created just by mixing red, green, and blue together with certain intensity. The amount of color intensity is from 0 to 255, resulting 16.7 million different colors by mixing different amounts of red, green, and blue. By using RGB values, you can pick any of the colors in the table above and more that 16 million others. Instead of specifying a number from 0 - 15, you specify a triple of numbers, like this SETPENCOLOR [187 187 187].

To fill in a region of the graphics window containing the turtle and bounded by lines that have been drawn earlier use the FILL command. It fills using the current value of FLOODCOLOR that can be set by using SETFLOODCOLOR (or SETFC).

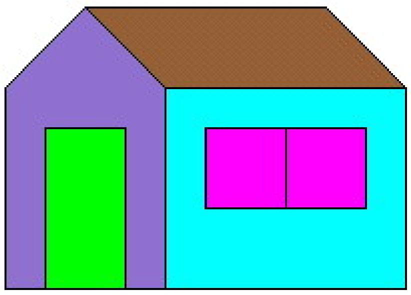
Example (Draw a square with blue lines, yellow fill, on black background):

```
SETSC "black
SETPC "blue
REPEAT 4 [FD 100 RT 90]
PU rt 45 fd 5
SETFC "yellow FILL
```



Activity:

1. Draw the following picture with Logo.
2. Create other interesting colorful pictures, such as buildings, animals, and so forth with Logo.



d. Turtle Movement

In addition to FORWARD and BACKWARD, there are many other commands to move the turtles. The following table lists these commands.

Table 6 Some Turtle Movement Commands

Motion Commands	Description
FORWARD *distance* FD *distance*	Moves the turtle forward in the direction that it's headed by the specified distance, measured in turtle steps (pixels)
BACK *distance* BK *distance*	Moves the turtle backward, exactly opposite to the direction that it's headed, by the specified distance, measured in turtle steps
LEFT *angle* LT *angle*	Turns the turtle counterclockwise by the specified angle, measured in degrees (1/360 of a circle)
RIGHT *angle* RT *angle*	Turns the turtle clockwise by the specified angle, measured in degrees

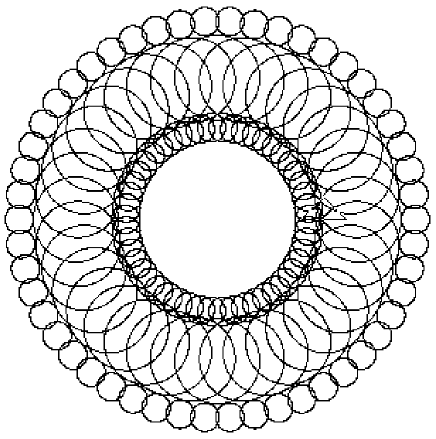
Motion Commands	**Description**
SETPOS *position*	Moves the turtle to an absolute X,Y coordinate. The *position* input is a list of two numbers, the X and Y coordinates. There is often some confusion on how to call SETPOS with variables. Let's look at some code: **MAKE "x 0** **MAKE "y 100** **SETPOS [:x :y] ; will fail** **SETPOS (list :x :y) ; will work** The first case is a list that contains 2 words ":x and ":y. In the second case a list is constructed containing the value of :x and :y.
POS	Outputs the turtle's current position, as a list of two numbers, the X and Y coordinates
SETX *xcor*	Moves the turtle along the X axis from its current position to a new absolute X coordinate. The *xcor* input is the new X coordinate.
SETY *ycor*	Moves the turtle along the Y axis from its current position to a new absolute Y coordinate
SETXY *xcor ycor*	Moves the turtle to an absolute X,Y coordinate.
XCOR	Outputs the turtle's X coordinate
YCOR	Outputs the turtle's Y coordinate
HOME	Moves the turtle to the center of the screen and sets the turtle's heading, pitch, and roll to 0.
SETHEADING *angle* SETH *angle*	Turns the turtle to a new absolute heading. The *angle* input is the heading in degrees clockwise from the positive Y-axis.
HEADING	Outputs the turtle's heading in degrees
TOWARDS *position*	Outputs (in degrees) the heading (direction), at which the turtle should be headed so that it would point from its current position towards the *position* ([X Y]) input.
DISTANCE *position*	Outputs the distance the turtle must travel along a straight line to reach the position given as the *position* ([X Y]) input
ELLIPSEARC *angle crosswise.semiaxis inline.semiaxis startangle*	Draws part of (or all of) an ellipse based on the turtle heading, turtle position and given inputs. ELLIPSEARC does not move the turtle. The center-point of the elliptic arc is the turtle's current position. The elliptic arc starts at a location given by *startangle* and sweeps an angle equal to *angle* in the clockwise direction. The size and shape of the elliptic arc is determined by the *crosswise.semiaxis* and *inline.semiaxis*

Motion Commands	**Description**
	inputs. The *crosswise.semiaxis* input is the distance from the turtle to the ellipse in the direction perpendicular turtle's current heading. The *inline.semiaxis* input is the distance from the turtle to the ellipse in the direction which the turtle is currently heading. The *startangle* input is relative to the turtle's current heading. A *startangle* of 0 indicates that the elliptic arc should start *inline.semiaxis* turtle steps directly behind the turtle. The *startangle* is measured in a clockwise direction, so a value of 90 indicates that the elliptic arc should start to the turtle's left. The start position will always be on the same ellipse, regardless of *startangle*. For example, a *startangle* of 90 will start *crosswise.semiaxis* turtle steps to the turtle's left. Normally the elliptic arc is draw clockwise, but if the *angle* input is negative, then the elliptic arc is draw counter-clockwise.
ELLIPSEARC2 *angle inline.semiaxis crosswise.semiaxis startangle*	ELLIPSEARC2 moves the turtle along an ellipse based on the turtle heading and given inputs. The elliptic arc starts at the turtle's current position and sweeps by the amount of *angle* in the clockwise direction. Normally, the turtle moves forward in a clockwise direction, but if the *angle* input is negative, then the turtle will move backward in a counter-clockwise direction. The *startangle* input should always be zero.
ELLIPSE *crosswise.semiaxis inline.semiaxis*	Draws an ellipse based on the turtle heading, turtle position, and given inputs. The center-point of the ellipse is the turtle's current position. ELLIPSE does not move the turtle.
ELLIPSE2 *crosswise.semiaxis inline.semiaxis*	Moves the turtle clockwise in an ellipse.
ARC *angle radius*	Draws an arc (part of a circle) based on the turtle heading, turtle position, and inputs. The center-point of the arc is the turtle's current position. ARC does not move the turtle. The arc sweeps an amount given by the *angle* input. The size (curvature) of the arc is based on the *radius* input.

Motion Commands	Description
	If *radius* is positive, then the arc starts *radius* turtle steps behind the turtle. If *radius* is negative, then the arc starts *radius* turtle steps in front of the turtle. In both cases, the arc sweeps an angle equal to the *angle* input. If *angle* is positive, then the arc is drawn in a clockwise direction. If *angle* is negative, then the arc is drawn in a counter-clockwise direction. If *angle* is 360 or greater (or -360 or less), then ARC draws a circle.
ARC2 *angle radius*	Moves the turtle along an arc (part of a circle).
CIRCLE *radius*	Draws a circle based on the turtle's position and the *radius* input. The center-point of the circle is the turtle's current position. CIRCLE does not move the turtle.
CIRCLE2 *radius*	Moves the turtle clockwise along a circle of the given radius. The turtle ends in the same position in which it starts.

Review & Challenge Questions

1. What is the different between SETPOS and SETXY commands?
2. What is the different between ELLIPSEARC and ELLIPSEARC2 commands?
3. What is the different between ELLIPSE and ELLIPSE2 commands?
4. What is the different between ARC and ARC2 commands?
5. What is the different between CIRCLE and CIRCLE2 commands?
6. What is the similarity between TOWARDS and DISTANCE commands?
7. Draw a triangle, a rectangular, and a square using SETPOS command (and other necessary commands).
8. Draw a triangle, a rectangular, and a square using SETXY command (and other necessary commands).
9. Draw a triangle, a rectangular, and a square using TOWARDS command (and other necessary commands).
10. Draw a triangle, a rectangular, and a square using DISTANCE command (and other necessary commands).
11. Draw concentric circles using CIRCLE command (and other necessary commands).
12. Draw a picture like the following.



Bab 3. Displaying Texts and Doing Calculation

So far you already learnt that Logo can be used to draw some pictures, including geometrical shapes. In this section you will learn that Logo can also display some texts, numbers, and do some mathematical calculations and display the results. You can display texts/numbers on the drawing screen or on the command history window. The command to display texts on drawing screen is LABEL, while command to display text on the command history window is PRINT, TYPE and SHOW.

a. LABEL command

The syntax to use the LABEL command is

```
LABEL text
```

that will print *text* on the drawing screen. The value of *text* can be:

- a single character and a single word (or string), that must be preceded by a double quoted ("), such as:

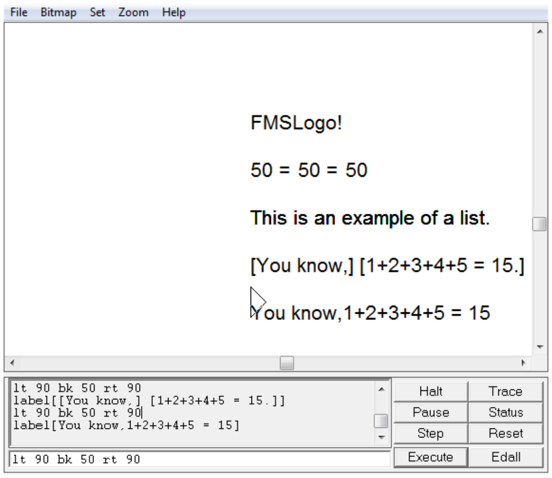
```
rt 90 LABEL "FMSLogo!
```

- a number or a calculation result, such as:

```
lt 90 bk 50 rt 90 LABEL 50
PU FD 30 LABEL "= FD 20 LABEL 5*10
FD 30 LABEL "= FD 20 LABEL 3*(1+2+3+4+5)+5
```

- a list (something written inside a square bracket), such as:

```
lt 90 fd 50 rt 90
label [This is an example of a list.]
lt 90 bk 50 rt 90
label[[You know,] [1+2+3+4+5 = 15.]]
lt 90 bk 50 rt 90
label[You know,1+2+3+4+5 = 15]
```



```
FMSLogo!

50 = 50 = 50

This is an example of a list.

[You know,] [1+2+3+4+5 = 15.]

You know,1+2+3+4+5 = 15
```

```
lt 90 bk 50 rt 90
label[[You know,] [1+2+3+4+5 = 15.]]
lt 90 bk 50 rt 90
label[You know,1+2+3+4+5 = 15]
```

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

- etc.

The following are some points that must be understand about LABEL command.

- As you can see in the above example, the handle (the origin) of the string is the top-left corner of the string. Sometimes the text can be drawn at the turtle heading and sometimes it cannot. Sometimes what is on the screen will not be exactly what you print.
- The position of the text is determined by the location of the turtle.
- The angle of the text is determined by the heading (direction) of the turtle.
- The color of the text will be the value of PENCOLOR, which can be set with the SETPENCOLOR command.

  SETPENCOLOR *color*
  SETPENCOLOUR *color*
  SETPC *color*

  These syntaxes set the pen color to *color*, which can either be a color index or a color vector. A color index is a number from 0 to 15. A color vector is a list of [**red green blue**] intensities, each in the range of 0 – 255. The pen color affects any line drawn by the turtle's moves, and any text that is drawn with the LABEL command.

- The text will use the font described by LABELFONT, which can be set with the SETLABELFONT command or from the menu **Set →Label Font**.

  The available fonts depend on your computer.

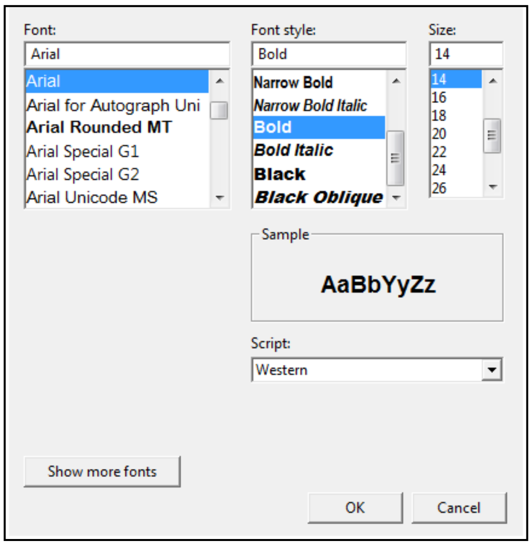


Figure 2 The FMSLogo Dialog for Setting Label Font

SETLABELFONT *font*

The *font* input must be a list of the form:

[[FaceName] Height Width Orientation Weight Italic Underline StrikeOut CharSet OutPrecision ClipPrecision Quality PitchAndFamily].

The meaning of each item within the list is explained by the following table.

Table 7 Font Characteristics

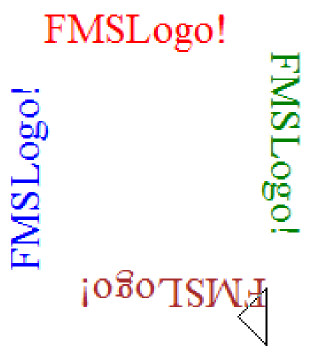
Item	**Meaning**
FaceName	A name or a list of font's typefaces
Height	An integer that specifies the font's height
Width	An integer that specifies the font's width
Orientation	An integer that specifies the font's orientation, in degrees
Weight	An integer (ranges from 0 to 900 in increments 100 with 0 means use default weight) that specifies the font's weight
Italic	An integer. The font is italicized if and only if this is non-zero.
Underlined	An integer (non-zero = underlined font)
StrikeOut	An integer (non-zero = horizontal line through the font)
CharSet	An integer that specifies the font's character set.
OutPrecision	An integer that specifies the font's output precision.
Quality	An integer that specifies the font's quality (0 = the default quality, i.e. the current operating system settings; 1 = draft quality; 2 = proof quality; 3 =no anti-aliasing; 4 = anti-aliasing)

PitchAndFamily	An integer that specifies the font's pitch and family.

If the selected font typeface is not available on the system, then SETLABELFONT will list all fonts that are available. It is better to use the easier way using the menu **Set → Label Font** to select the font for displaying label.

Example:

```
CS SETPC "blue
LABEL "FMSLogo!
FD 150 RT 90 FD 25 SETPC "red
LABEL "FMSLogo!
FD 150 RT 90 FD 25 SETPC "green
LABEL "FMSLogo!
FD 150 RT 90 FD 25 SETPC "brown
LABEL "FMSLogo!
```



b. PRINT, TYPE and SHOW Commands

PRINT *thing*
PR *thing*
(**PRINT** *thing1 thing2 ...*)
(**PR** *thing1 thing2 ...*)
TYPE *thing*
(**TYPE** *thing1 thing2 ...*)
SHOW *thing*
(**SHOW** *thing1 thing2 ...*)

The PRINT (or PR) will prints the input or inputs (*thing, thing1, thing2, …*) to the current write stream (initially the command-history window). All the inputs are printed on a single line, separated by spaces, ending with a newline. If an input is a list, square brackets are not printed around it, but brackets are printed around sub-lists. Braces are always printed around arrays.

Example:

```
PRINT "FMSLogo!
FMSLogo!
PR [You already know that 1+2+3+4+5=] PR 1+2+3+4+5
You already know that 1+2+3+4+5=
15
(PR "5! "= "1*2*3*4*5 "= 1*2*3*4*5)
5! = 1*2*3*4*5 = 120
```

The TYPE command will concatenate the input or inputs into one string that will be displayed with the PRINT command. The concatenated string will not end with a newline character and multiple inputs are not separated by spaces.

Example:

```
TYPE [You must know that 1+2+3+4+5 = ] (PRINT " 1+2+3+4+5)
You must know that 1+2+3+4+5 = 15
```

The SHOW command prints the input or inputs to the current write stream, just like PRINT, except that if an input is a list it is printed inside square brackets

Example:

```
(SHOW [You must know that 1+2+3+4+5 = ] 1+2+3+4+5)
[You must know that 1+2+3+4+5 =] 15
```

Another example of use of REPCOUNT and REPEAT is to print number sequences. Originally, the REPCOUNT will result in integer sequence 1, 2, 3, 4, 5, …, until the number after the command REPEAT. This is very useful to generate other sequences.

Example:

```
PRINT [An even number sequence]
An even number sequence
REPEAT 10 [TYPE 2*(REPCOUNT-1) type ", ] show 20
0,2,4,6,8,10,12,14,16,18,20
PRINT [An odd number sequence]
An odd number sequence
REPEAT 10 [TYPE 2*REPCOUNT-1 type ", ] show 21
1,3,5,7,9,11,13,15,17,19,21
PRINT [A square number sequence]
```

```
A square number sequence
REPEAT 10 [TYPE POWER REPCOUNT 2 type ", ] show 121
1,4,9,16,25,36,49,64,81,100,121
```

Can you figure out how to get the sequence of numbers from 10 to 20? How about all odd numbers from 1 to 19? How about multiples of 7 between 14 and 77?

Activity:

1. Using REPEAT and REPCOUNT, print all odd numbers between 20 and 50.
2. Using REPEAT and REPCOUNT, print all even numbers between 20 and 50.
3. Using REPEAT and REPCOUNT, print all multiple of 5 between 20 and 50.
4. Experiment with REPEAT and REPCOUNT to create your own interesting pictures. You can start with any sample that uses REPEAT and modify the loop to use "REPCOUNT" or you can write your own from scratch.

c. Doing Mathematics with Logo

Table 8 FMSLogo Mathematical Functions

Mathematics Calculation	**Logo**	**Description/Examples**
Addition	SUM *number1 number2* (SUM *number1 number2 …*) *number1+number2*	SHOW 2 + 3 5 SHOW (SUM 1 2 3) 6
Subtraction	DIFFERENCE *number1 number2* *number1* **-** *number2*	SHOW 3 - 2 1 SHOW DIFFERENCE 3 2 1
Multiplication	PRODUCT *number1 number2* (PRODUCT *number1 number2 …*) *number1*number2*	**SHOW 2 * 3** 6 **SHOW (PRODUCT 2 3 4)** 24
Division	QUOTIENT *dividend divisor* (QUOTIENT *divisor*) *dividend / divisor*	QUOTIENT outputs an integer if and only if *dividend* is a multiple of *divisor*. When the single input form is called, QUOTIENT outputs the reciprocal of the input. **SHOW 6 / 3** 2 **SHOW (QUOTIENT 5 2)** 2.5 **SHOW (QUOTIENT 2)** 0.5

Mathematics Calculation	**Logo**	**Description/Examples**
Remainder	REMAINDER *integer1 integer2*	SHOW REMAINDER 10 3 1 SHOW REMAINDER -10 3 -1 SHOW REMAINDER -10 2 0
Modulo	MODULO *integer1 integer2*	SHOW MODULO 10 3 1 SHOW MODULO -10 3 2 SHOW MODULO -10 5 0
Rounding	ROUND *number* INT *number*	SHOW INT 8.2 8 SHOW INT 8.7 8 SHOW ROUND 8.2 8 SHOW ROUND 8.7 9
Absolute Value	ABS *number*	SHOW ABS -7 7 SHOW ABS 3.4 3.4
Power	POWER *base exponent*	SHOW POWER 2 5 32 SHOW 5 2 25
Exponential	EXP *number*	e^{number} (e =2.718281828...) SHOW EXP 1 2.71828182845905
Square Root	SQRT *number*	SHOW SQRT 4 2 SHOW SQRT 2 1.4142135623731
Logarithm	LOG10 *num* LN *num*	SHOW LOG10 1 0 SHOW LOG10 10 1 SHOW LOG10 100 2 SHOW LN EXP 1 1
π	PI	SHOW PI 3.14159265358979
Trigonometric function	**SIN** *angle* **COS** *angle* **TAN** *angle*	(show sin 0 sin 30 sin 90) 0 0.5 1 (show cos 0 cos 60 cos 180)

Mathematics Calculation	Logo	Description/Examples
	(*angle* is in degree) **RADSIN** *angle* **RADCOS** *angle* **RADTAN** *angle* (*angle* is in radian, $1\,rad\ =\ 180^0/\pi$)	1 0.5 -1 (show tan 0 tan 45 tan 60) 0 1 1.73205080756888 (show radsin 0 radsin pi/6 radsin pi/2) 0 0.5 1 (show radcos 0 radcos pi/3 radcos pi) 1 0.5 -1 (show radtan 0 radtan pi/4 radtan pi/3) 0 1 1.73205080756888
Trigonometric function Inverse	**ARCSIN** *num* **ARCCOS** *num* **ARCTAN** *num* ($-1 \leq num \leq 1$) output range [-90, 90] **RADARCSIN** *num* **RADARCCOS** *num* **RADARCTAN** *num* ($-1 \leq num \leq 1$) output range [-pi/2, pi/2]	(show arcsin 0 arcsin 1 arcsin 0.5) 0 90 30 (show arccos 0 arccos 1 arccos 0.5) 90 0 60 (show arctan 0 arctan 0.5 arctan 1) 0 26.565051177078 45 SHOW (RADARCSIN 1) = PI/2 true SHOW (RADARCCOS (SQRT 2)/2) = PI/4 true SHOW (RADARCTAN 1) = PI/4 True

In addition to the above mathematical functions, Logo also provides a function to generate random numbers. There are two ways to add randomness. The first is with random num-bers. The second is by picking an element at random from a list.

Generating Random Numbers

Logo provides RANDOM command to generate random numbers. The syntax is below.

RANDOM *number*

This command will generate a uniform random integer from 0 to the *number* input (not including the *number* input). The *number* input must be a nonnegative integer.

For example, the instruction

```
RANDOM 10
```

will output any of these integers **with equal probability** (the meaning of the word random): 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Computers are inherently deterministic, which means that, given a set of inputs, computers always follow a program to produce the exact same output. This makes it difficult for a computer to generate truly random numbers. For this reason, when a computer generates a random number, it is often called a pseudo-random number, which means "fake random number". Here's how RANDOM works. Imagine that RANDOM has a giant list of numbers that appear in a predetermined, random-looking order. Each time you run RANDOM, it reads the next number from the list, scales it to the requested range, and outputs the result. If RANDOM always started at the top of this list, then it would always produce the same sequence of numbers each time Logo was run, which wouldn't appear random at all. Of course, RANDOM can't just pick a random place in the list to start, because computers can't create true randomness. To get around this problem, when Logo starts up, it uses the current time to determine where in the list it should start getting numbers from. As long as you start each program at a different time, the random sequence will be different, which is usually good enough to simulate true randomness. But if you manage to start two instances of Logo at exactly the same time, then RANDOM would output the exact same sequence of numbers.

Example:

```
REPEAT 5 [SHOW RANDOM 10]
6
8
3
0
9
repeat 5 [show random 100]
33
12
85
70
91
```

Normally, you want RANDOM to generate different values each time your program is run. But sometimes you may want to produce the exact same sequence of numbers. In these cases, you can use RERANDOM to reset the location in the list where RANDOM will get its next number from.

The command RERANDOM makes the results of RANDOM reproducible. Ordinarily the sequence of random numbers is different each time Logo is used. If you need the same sequence of pseudo-random numbers repeatedly, e.g. to debug a program, say RERANDOM before the first invocation of RANDOM. If you need more than one repeatable sequence, you can give RERANDOM an integer input; each possible input selects a unique sequence of numbers.

Example:

```
rerandom repeat 5 [show random 100]
58
66
25
75
90
rerandom repeat 5 [show random 100]
58
66
25
75
90
```

The RANDOM command can be used to generate random numbers in other ranges. For example,

```
(RANDOM 10) + 10
```

will returns a random number from 10 to 19. Likewise,

```
(RANDOM 10) * 10
```

returns a random multiple of 10 from 0 to 90 (a number in the set {0, 10, 20, 30, 40, 50, 60, 70, 80, 90}).

Activity:

1. How would you generate a random number from -5 to 5?
2. How would you generate a random *even* number from 0 to 30?
3. How would you generate five random numbers of multiple of 3 from 3 to 30?

Random Elements in a List

Random numbers are great, but let's say you already have some values in mind and you just want Logo to pick one at random. In this case, the PICK command to select randomly a number from a list of number.

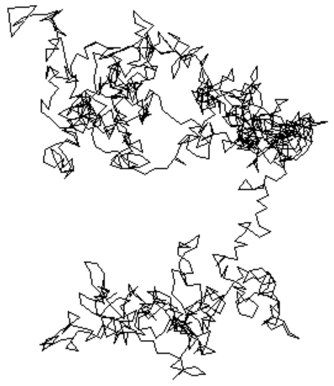
Command Syntax	Example	What Happens
PICK *list*	PICK [1 4 9 16 25]	Returns either 1, 4, 9, 16, or 25 at random.
RUN *instruction*	RUN PICK [[FD RANDOM 100] [BK RANDOM 100]]	Goes either forward or backward at random.

Notice how PICK and RUN can be combined to execute a command at random.

Random numbers can be used to generate interesting graphics by generating random moving, random directions, and random colors.

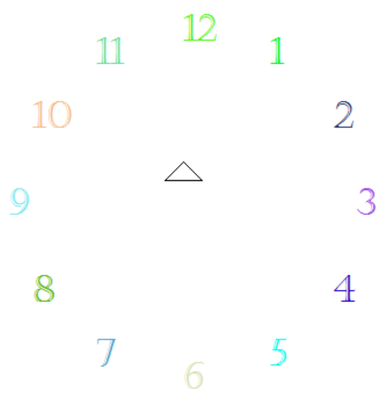
Example (random walk):

```
REPEAT 1000 [FORWARD 10 RIGHT RANDOM 360]
```



Example (Printing colorful numbers in circular analog clock):

```
SETLABELFONT [[Castellar] -32 0 0 400 0 0 0 0 3 2 1 18]
cs rt 30 repeat 12 [fd 150 setpc (list random 256 random 256 random 256) rt (3-
repcount)*30 label repcount lt (3-repcount)*30 bk 150 rt 30 wait 10] lt 30
```



Bab 4. Writing Logo Programs: Teach the Turtle New Commands

In this section you will get the following experiences.

1. You can teach the turtle new commands by using the built-in ones.
2. You can use **Edall** button to open the Logo editor to write and edit Logo programs.
3. You can save and load Logo programs written with the Logo editor.

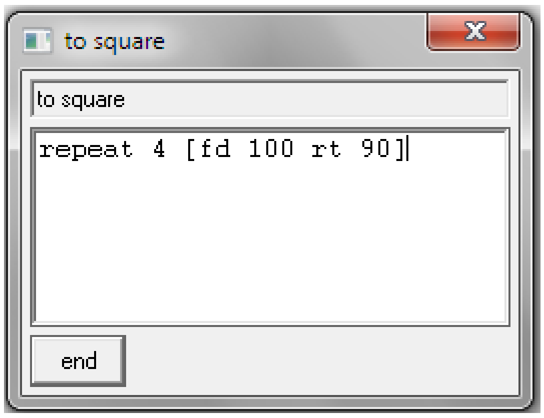
One time, someone, after trial and error, may succeed in writing complicated instruction to draw interesting picture. Next time he/she may want to generate the same or similar picture without rewrite again the complicated instructions that already forgotten. Instead of remembering or rewriting the instructions, Logo provides an easy way to remember the instructions by storing them in a program. A Logo program is just a series of instructions preceded by **TO** command and ended by **END** command. The structure of Logo program is

```
TO name (input1 input2 …)
instruction1 instruction2 instruction3 …
END
```

with *name* the desired name of program by which the program should be called as a new command to Logo. A program name must be non separated words, preceded by a letter, can not contain arithmetic operations nor special characters like "?", "!", "%", etc. The *input1*, *input2*, etc. are variables as optional inputs that can be replaced by some values when the program name is called in the Logo command line window. To program in Logo means to teach Logo understand new commands that are not available in the Logo original program. Therefore, the program name must be different from any Logo words. Programmers usually use their own words to name their programs, like SQUARE, TRIANGLE, CIRCLE, etc. The same as original Logo words, program names are also not CASE sensitive, so program name **SQUARE**, **Square**, **square**, **squARE** are the name.

a. Program Without Parameter

There are two ways to write a new program. The first way is by writing TO followed by the program name, at the command window, then press ENTER. Then, Logo will display a simple program editor like **Figure 2**.



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

Figure 3 The Logo Simple Program Editor

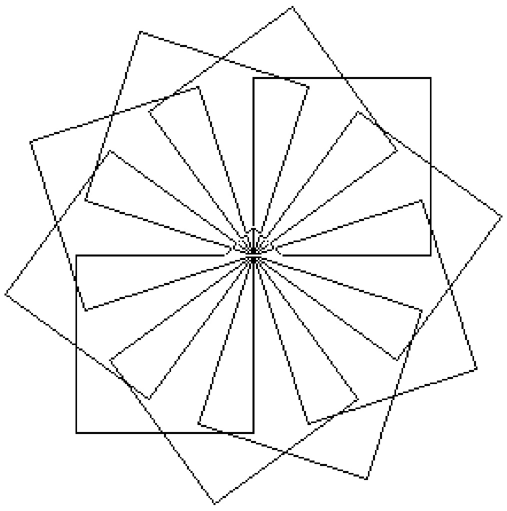
As it can be seen in the above example, the square program contains instructions to draw a square of side length 100 pixels. After finish writing the instructions, to close the program, just click the **end** button. You will see the following messages in the command history window.

```
to square
square defined
```

It means that Logo now understand a new command called SQUARE. To run the program, just write SQUARE in the command window, and you will see Logo will draw a square.

Now, you can use the SQUARE command to create an interesting picture. Do this.

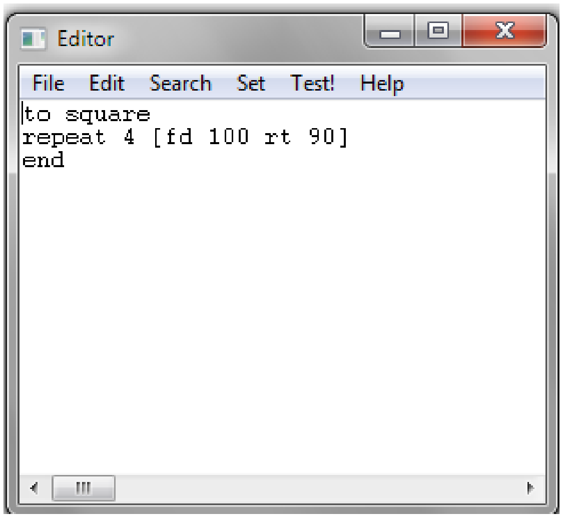
```
repeat 10 [lt 36 square]
```



Activity:

1. Write the program SQUARE as explained above.
2. Create new interesting pictures by using the program SQUARE.

The other way to write (and edit) Logo programs is by writing EDALL command and press ENTER or using **Edall** shortcut button. Just do it, you will see the FMSLogo editor like **Figure 3**.



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

Figure 4 The FMSLogo Full-featured Program Editor

In the editor you can write new programs, every program is enclosed by the pair of TO and END commands. For example, you may add new program to draw a parallelogram as follows.

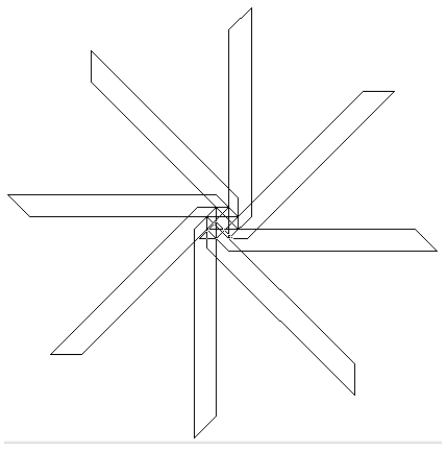
```
to parallelogram
rt 45
```

```
repeat 2 [fd 200 rt 45 fd 30 rt 135]
end
```

To check instructions within a program, select (block) the instruction and then click the **Test!** command in the editor. If the result is what you expect, then the program is correct. To close the editor to go to the main FMSLogo windows, click **File → Save and Exit** or press **Ctrl+D**. Now, you have two program called SQUARE and PARALLELOGRAM. Try to call the PARALLELOGRAM command in the command window, and see the result.

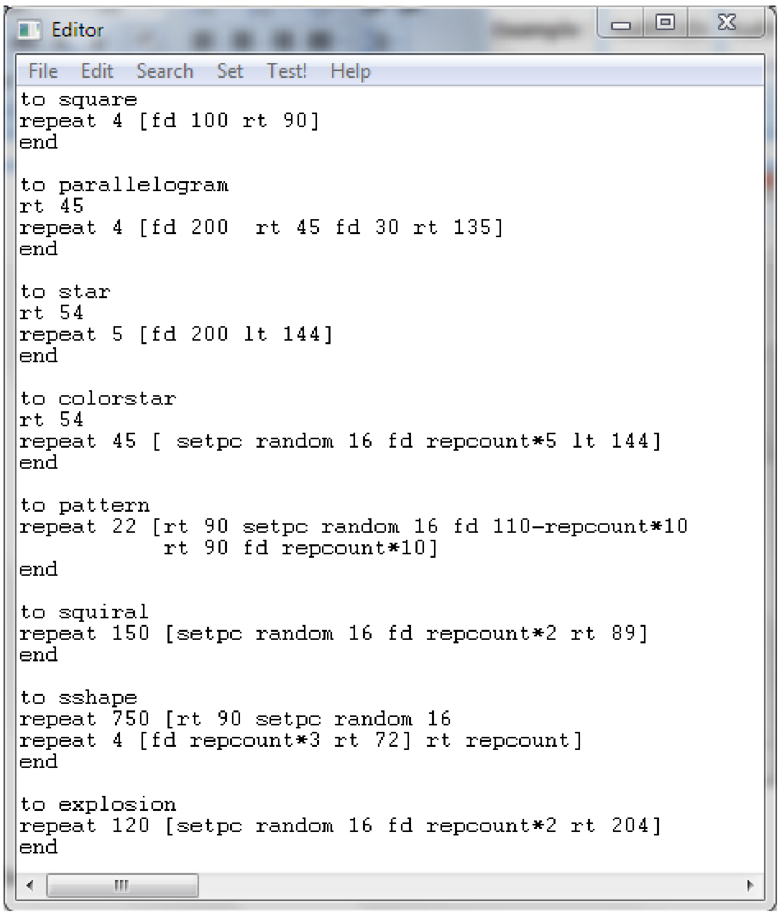
Now, you can use the PARALLELOGRAM command to create another picture, a fan. Do this.

```
cs repeat 8 [parallelogram fd 30 lt 180]
```



When you close the editor, by clicking **File → Save and Exit** or pressing **Ctrl+D**, the programs are just stored in the Logo workspace (memory). They are not stored in the hard disk yet. To save the programs into hard disk or other storing devices, from the main FMSLogo window, click **File → Save As…**. Name your file as your desire. Next time you add new or modify programs to save again, just click **File → Save**. The file extension of Logo programs is **.LGO**. If you want to run your stored program, use **File →Load**. After that you can run the program stored in the loaded file. Figure 4 below shows FMSLogo editor with several sample programs.

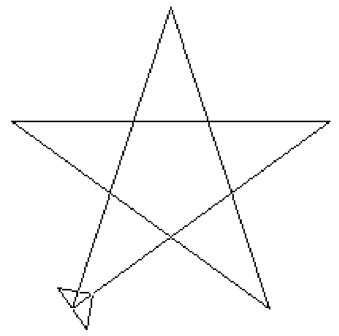
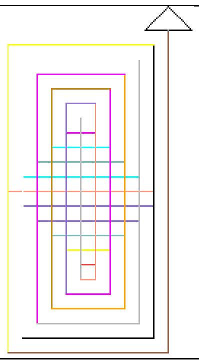
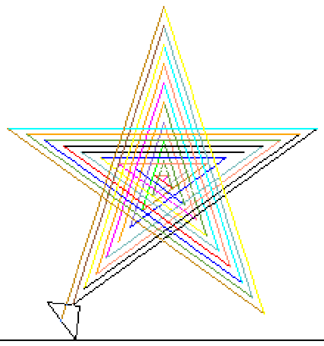
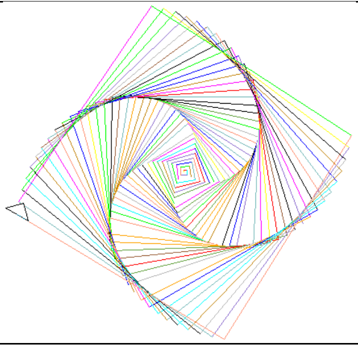
Because every new program is considered as new command by Logo, it can be used in any other program, even in its definition (this is called recursive program).

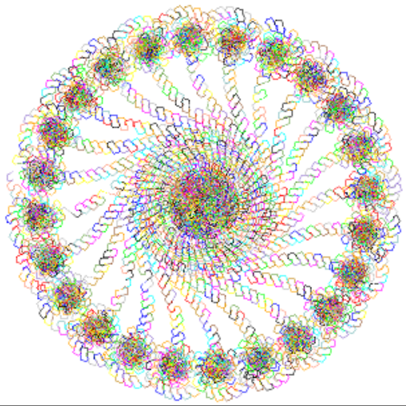
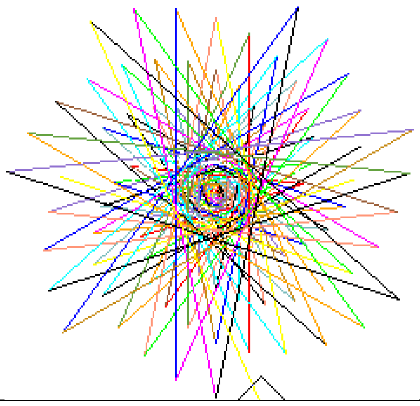


```
to square
repeat 4 [fd 100 rt 90]
end

to parallelogram
rt 45
repeat 4 [fd 200  rt 45 fd 30 rt 135]
end

to star
rt 54
repeat 5 [fd 200 lt 144]
end

to colorstar
rt 54
repeat 45 [ setpc random 16 fd repcount*5 lt 144]
end

to pattern
repeat 22 [rt 90 setpc random 16 fd 110-repcount*10
           rt 90 fd repcount*10]
end

to squiral
repeat 150 [setpc random 16 fd repcount*2 rt 89]
end

to sshape
repeat 750 [rt 90 setpc random 16
repeat 4 [fd repcount*3 rt 72] rt repcount]
end

to explosion
repeat 120 [setpc random 16 fd repcount*2 rt 204]
end
```

Figure 5 FMSLogo editor with several sample programs

The following are some results of the sample programs.

	
star	PATTERN
	

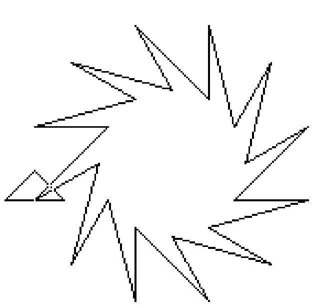
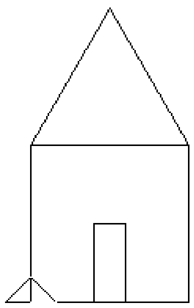
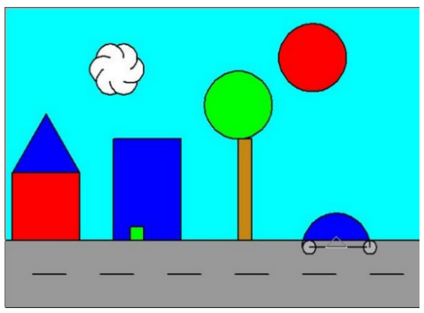
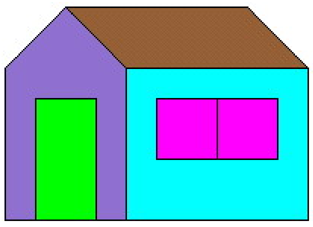
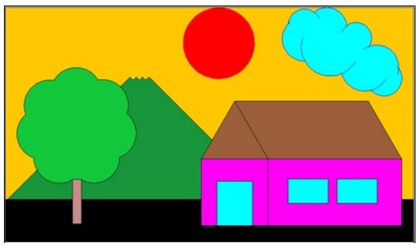
COLORSTAR	SQUIRAL
	
Repeat 24 [SSHAPE]	EXPLOSION

Review & Challenge Questions

1. How do you define a program in Logo?
2. Write the programs to draw the above pictures.
3. How can you create the sequence of odd numbers (1, 3, 5, 7, ...) with REPEAT and REPCOUNT?
4. For some values of X (for example, 60) the code

   REPEAT 100 [SETPC RANDOM REPCOUNT FD REPCOUNT RT X]

   draws a "stable" colorful picture – it looks like the same shape inside itself.
Others
   values (for example, 61) draw "twisty" pictures. What makes a value stable?
What
   makes it twisty? Is there a limit to the number of stable pictures can you draw
with this
   pattern?
5. Why is it that, when you repeat the SQUIRAL program, it ends up going around in a circle, but when you repeat the COLORSTAR program, it goes off in a straight line?
6. Why does the SSHAPE command curve inward than outward (and back again), instead of either spiraling inward forever or spiraling outward forever?
7. Write Logo programs to draw the following pictures (Use the required commands already explained).

b. Program with Parameter

To this end, it is expected that you already able to draw some geometrical shapes or other patterns and designs with Logo program. However, so far the pictures, patterns, or design you can create with programs, may have fixed sizes. How if you want to create flexible pictures, patterns, or designs with different sizes without changing the programs? Consider the following problems.

1. Drawing a rectangle with given sizes.
2. Drawing an equilateral triangle with certain side length.
3. Drawing regular polygon of certain number of sides.
4. Drawing flowers with certain number of leaves and certain pattern.

All these can be done by using parameters in the programs. Parameters of a program are variables written after program name, preceded by a colon (:), separated by space. Inside a program, you may define some "local" variables using MAKE command and the variable name must be preceded by a double quote mark (") followed by its value (in the form of fixed number or a formula). When using variables (either they are program parameters or local variables) in a formula, each of them must be preceded by a colon.

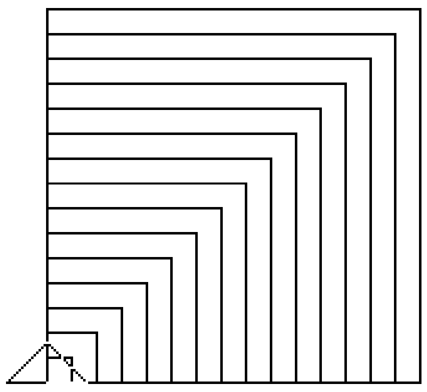
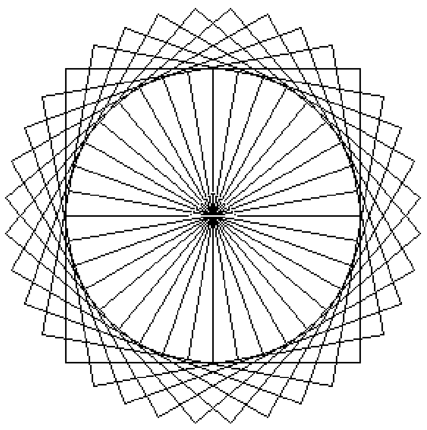
Example (Drawing Squares of Any Size):

```
TO SQUARE :LENGTH
 REPEAT 4 [ FD :LENGTH RT 90 ]
END
```

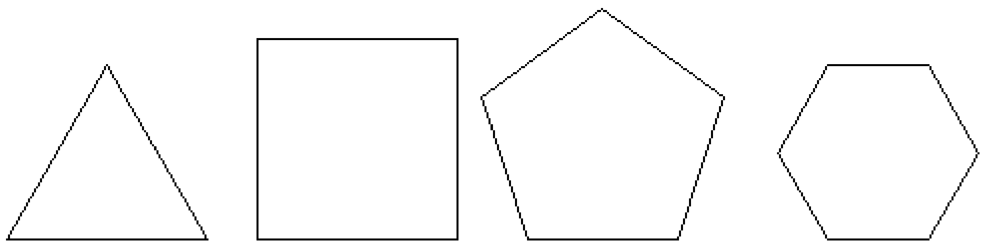
The above SQUARE program has a parameter ":LENGTH". This parameter must be replaced with a number when calling the program SQUARE, such as "SQUARE 10" to tell the turtle to make a square with sides that are 10 pixels long.

Furthermore, we can use our more powerful SQUARE program to draw an interesting design.

```
REPEAT 15 [ SQUARE REPCOUNT * 10 ]
REPEAT 36 [ SETXY 0 0 LT 10 SQUARE 100 ]
```

Example (Drawing a Regular Polygon with any Number of Sides and Side Length)**:**



The steps to draw a regular polygon of n sides are as follows.

1. Turn right 90 degrees.
2. Move forward as far as the side length.
3. Turn left 360/n degrees.
4. Repeat steps 2-3 n times.

Based on the above algorithm, we can write the following Logo program to draw a regular polygon of certain number of sides and side length.

```
TO POLYGON :SIDES :LENGTH
 RT 90
```

```
  REPEAT :SIDES [FD :LENGTH LT 360 / :SIDES ]
END
```

The program POLYGON has two parameters, namely SIDES and LENGTH to determine the number of sides and the length of each side. When calling the program POLYGON two numbers representing the number of sides (must be a positive integer) and the length of the side must be supplied.

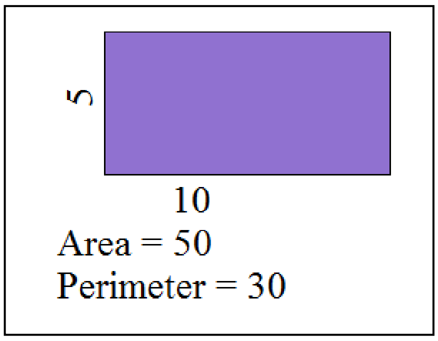
Activity:

1. Type in the program POLYGON and play around with it. What happens when you give it a number less than 3? What happens when you give it a large number, like 50?
2. Modify the program POLYGON so the region of the regular polygon is filled with color.
3. Create your own program that takes some parameter(s). You can start from scratch or you can start with any sample in this paper or other resources.

Example (Drawing a rectangle with certain length and width and calculate its area and its perimeter):

This example shows how Logo can be used to do mathematics calculation, in addition to drawing geometrical shapes. Study the sample program and see one of sample result.

```
to arectangle :length :width
 pd
 repeat 2 [fd 20*:width rt 90 fd 20*:length rt 90]
 pu ht setfc random 15 rt 45 fd :length fill
 lt 45 bk :length
 setxy xcor-40 ycor+10*:width label :width rt 90
 setxy xcor+8*:length ycor-10*:width label :length
 setxy xcor-8*:length ycor-30 label (list "Area "= :length*:width)
 setxy xcor ycor-30 label (list "Perimeter "= 2*(:length+:width))
end
```

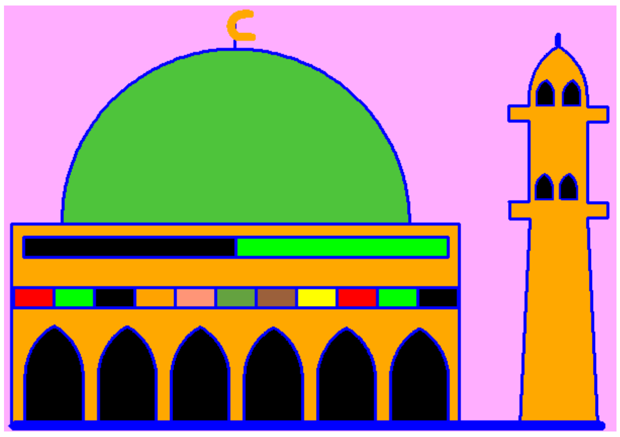


Activity:

1. Type in the program ARECTANGLE and play around with it with different sizes.
2. Write similar program to draw a right triangle with certain height and base and calculate its area and perimeter, and then play around with it with different sizes.
3. Write similar program to draw a parallelogram with certain length and height and calculate its area and perimeter, then play around with it with different sizes.
4. Write similar program to draw a circle with certain radius and calculate its area and circumference, then play around with it with different sizes.

Review & Challenge Questions

1. How much does the turtle turn to create a regular hexagon?
2. How do you define a program that takes a parameter in Logo?
3. How can you make shorter name for FORWARD than FD?
4. How can you write a POLYGON command that draws all polygons with the same circumference?
5. Write a Logo program to draw a colorful mosque, like below.



Bab 5. Playing Music with Logo

We can play music in the same way as we make drawings using Logo programs. FMSLogo provides some commands to play sounds, multimedia files and musical instruments like MIDI. Below are some basic FMSLogo commands to play music.

PLAYWAVE *wavefile flags*

This command plays a wave file (.WAV). The *wavefile* input must be a filename (including the full folder/directory location) of a WAVE file. It may either be a word or a list of words. The *flags* input is an integer that describes how the sound should be played. The meaning is given by the following table.

Flag	Behavior
0	Synchronous does not stop until completed.
1	Asynchronous returns immediately while sound is still playing.
2	Don't use the default sound if the specified one cannot be found.
4	In memory sound (not supported).
8	Continue to loop the sound until another PLAYWAVE instruction is run.
16	Don't stop an already playing sound.

These flags can be combined by adding them together.

Example:

Assuming there is a Wav file name CARDINAL.wav in the folder **C:\\Users\\Syahid\\ Music**, the following instruction will play the WAV file repeatedly, until the next instructions are run.

```
; loop the play
PLAYWAVE " C:\ \Users\ \Syahid\ \Music\ \cardinal.wav 1+8
WAIT 120       ; wait two seconds
PLAYWAVE [] 0  ; stop the sound
```

TONE *frequency duration*

This command plays a tone on the PC speaker. The *frequency* input must be a number and is measured in hertz (cycles per second). The larger the frequency the higher the pitch. The *duration* input must be a number and is measured in milliseconds. The larger the duration the longer the tone will play for. FMSLogo cannot run other command while a sound is being played.

Example (hear the sound):

```
TONE 1000 200
REPEAT 50 [TONE REPCOUNT*RANDOM 500 250 ]
```

SOUND *soundlist*

This command plays a sequence of tones on the PC speaker. The *soundlist* input must be a list of pairs. Each pair specifies a frequency (in hertz) and duration (clock ticks). The larger the frequency the higher the pitch. The larger the duration the longer the sound will play for. The sound will only come out the PC speaker. FMSLogo cannot run other command while a sound vector is being played

Example (hear the sounds):

```
SOUND [1000 2000]
REPEAT 50 [SOUND (list (RANDOM REPCOUNT)*400 600)]
```

Creating music, such as songs, using the primitive Logo commands TONE and SOUND needs to specify the frequency and duration values. This is not so easy. The resulting music is also not so nice, because it uses PC speaker. In order to make it easier to play music and creating songs with Logo, we can use a helper program called **player.lgo** written by Michael C. Koss. This program can be downloaded freely from the Internet address http://mckoss.com/logo/player.htm. After downloaded and saved into hard disk, the file needs to be loaded into FMSLogo by using menu **File → Load**. Your computer needs to have a sound card or MIDI instrument attached in order to use this program. To test that this is working in FMSLogo, type:

```
do re mi fa sol la ti high do
```

in the command window and hear the sounds of the notes.

The **player.lgo** file defines some musical aspects, as follows.

- **Notes** (Commands to play a midi note)**:** Do, Re, Mi, Fa, Sol, La, Ti
- **Durations :** Beats *value*, Eighth (=beats 0.5), Quarter (= beats 1), Half (=beats 2), Whole (= beats 4)
- **Octave Commands:** Dinasaur, Low, Middle, High

- **Incidentals:** Flat, Natural, Sharp
- **Musical Instruments:** Piano, Harpsichord, Musicbox, Guitar, Hendrix, Violin, Trumpet, Brass, Flute, Banjo, Steeldrum, Tom, Gun

The default (or initial) music defined in the **player.lgo** is using duration 8 beats, quarter duration, middle octave, natural incidental, and instrument piano. This condition can be reset with the RESET command refined in the file.

Playing music and creating songs in FMSLogo now can be done just by using the above commands. You can type in notes, and Logo will play them in the same order you type them in, such as:

```
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re do
```

that makes a sound like a piano. To make sounds like different instruments, you can type for example:

```
half guitar
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do

half musicbox
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do

half Harpsichord
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do

half Hendrix
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do

half trumpet
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do

half brass
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do
```

```
half flute
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do

half banjo
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do

half Steeldrum
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do

half tom
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do

half gun
do re mi fa sol la ti half high do
quarter do middle ti la sol fa mi re whole do
```

We can also make "random" music:

```
TO SONG
   REPEAT 24 [NOTE RANDOM 12]
END
```

We can even make a "picture" of the music:

```
TO SONG2
   REPEAT 24 [PLAYDRAW RANDOM 12 REPCOUNT]
END

TO PLAYDRAW :NOTE :WHERE
   SETXY :WHERE*12 :NOTE*12
   NOTE :NOTE
END
```

Example (Traditional Javanese Song):

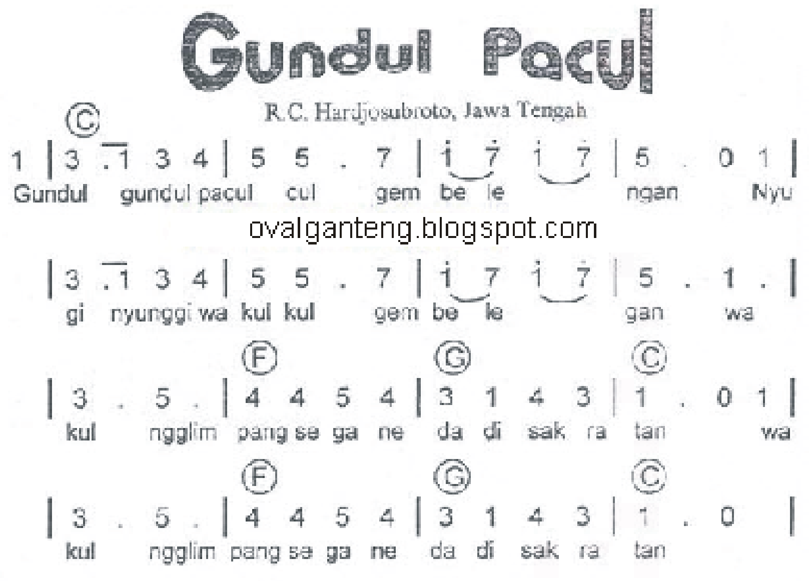

Gundul Pacul
R.C. Hardjosubroto, Jawa Tengah
ovalganteng.blogspot.com
Gundul gundul pacul cul gem be le ngan Nyu
gi nyunggi wa kul kul gem be le gan wa
kul ngglim pang se ga ne da di sak ra tan wa
kul ngglim pang se ga ne da di sak ra tan

Figure 6 Gundul Pacul Song with Number Notes and Lyrics

The following is Logo program to play the "Gundul Pacul" song.

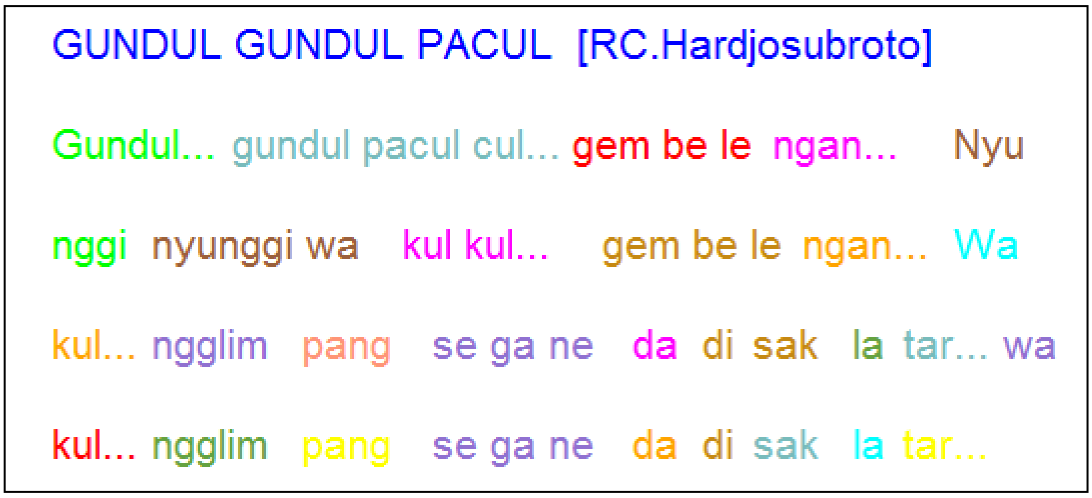
```
to gundulpacul :instrument
  reset
  run :instrument
  whole do beats 6 mi whole do mi fa sol beats 8 sol whole ti
    high do ti do ti beats 8 sol whole do
  beats 6 mi whole do mi fa sol beats 8 sol whole ti high do
    ti do ti beats 6 sol whole do
  beats 6 mi sol half fa whole fa sol fa mi do fa mi beats 8
    do whole do
  beats 6 mi sol half fa whole fa sol fa mi do fa mi beats 16
    do
end
```

To play the song, type GUNDULPACUL followed by the instrument's name preceded by a double quote ("), e.g. GUNDULPACUL "VIOLIN.

The following is a modification of the above song that displaying colorful lyrics in the drawing screen.

```
to gundulsong :instrument
reset
run :instrument
cs rt 90 pu ht
setxy -200 200 setpc random 15 label (list "GUNDUL "GUNDUL "PACUL
"[RC.Hardjosubroto]) wait 120
```

```
setxy -200 150 setpc random 15 label "Gundul... whole do beats 6 mi
setxy -110 150 setpc random 15 label (list "gundul "pacul "cul...)
whole do mi fa sol beats 8 sol
setxy 60 150 setpc random 15 label (list "gem "be "le)
whole ti high do ti do ti
setxy 160 150 setpc random 15 label "ngan... beats 8 sol
setxy 250 150 setpc random 15 label "Nyu     whole do
setxy -200 100 setpc random 15 label "nggi   beats 6 mi
setxy -150 100 setpc random 15 label (list "nyunggi "wa) whole do mi fa
setxy -25 100 setpc random 15 label (list "kul "kul...) sol beats 8 sol
setxy 75 100 setpc random 15 label (list "gem "be "le)  whole ti high do ti do ti
setxy 175 100 setpc random 15 label "ngan...  beats 6 sol
setxy 250 100 setpc random 15 label "Wa       whole do
setxy -200 50 setpc random 15 label "kul...   beats 6 mi
setxy -150 50 setpc random 15 label "ngglim   sol
setxy -75 50 setpc random 15 label "pang     half fa
setxy -10 50 setpc random 15 label (list "se "ga "ne)  whole fa sol fa
setxy 90 50 setpc random 15 label "da mi
setxy 125 50 setpc random 15 label "di do
setxy 150 50 setpc random 15 label "sak fa
setxy 200 50 setpc random 15 label "la mi
setxy 225 50 setpc random 15 label "tar... beats 8 do
setxy 275 50 setpc random 15 label "wa whole do
setxy -200 0 setpc random 15 label "kul...   beats 6 mi
setxy -150 0 setpc random 15 label "ngglim   sol
setxy -75 0 setpc random 15 label "pang     half fa
setxy -10 0 setpc random 15 label (list "se "ga "ne)  whole fa sol fa
setxy 90 0 setpc random 15 label "da mi
setxy 125 0 setpc random 15 label "di do
setxy 150 0 setpc random 15 label "sak fa
setxy 200 0 setpc random 15 label "la mi
setxy 225 0 setpc random 15 label "tar... beats 16 do
end
```



```
GUNDUL GUNDUL PACUL  [RC.Hardjosubroto]

Gundul... gundul pacul cul... gem be le  ngan...    Nyu

nggi  nyunggi wa   kul kul...   gem be le  ngan...  Wa

kul... ngglim   pang   se ga ne   da  di  sak   la  tar... wa

kul... ngglim   pang   se ga ne   da  di  sak   la  tar...
```

Activity:

1. Look for some WAV files and play them with PLAYWAVE Logo command.
2. Download the file player.lgo from http://mckoss.com/logo/player.htm and saved on your hard disk. After finished, load it into FMSLogo. Try to play some notes.
3. Write the above samples, and play on FMSLogo.
4. Look for some song songs that interesting and familiar for you, and then write Log programs to play the songs.

Bab 6. Using LOGO in Mathematics Classroom

After learning and practicing Logo, you should already get some ideas on how to use Logo in teaching and learning mathematics in primary school.

The following are some steps that may be appropriate.

1. Explain and demonstrate to the kids how to start Logo program, such as FMSLogo.
2. Explain and demonstrate to the kids some basic Logo commands (turtle movement, turning, using color), ask them to practice and explore the commands. Explain to them about the steps in creating design as follows.
   a. **Plan**
      1) Write down what you are going to need to make the design.
      2) Write down how you will make a design-step by step.

   b. **Create**
      3) Follow the plan and make your design.
      4) Write down the changes you make to your plan and your design.

   c. **Evaluation**
      5) Reflect on what you did in the investigation, design, plan and create stages.
      6) Evaluate your design. What did you do well? What did you do not so well? How could you improve?
      7) Compare it to what you said you would make in your design.
      8) Compare it to the list of what makes a good design in LOGO.
3. Ask them to make some designs on paper and ask them how they create it, how do they instruct Logo do make the same design.
4. Start using Logo to draw some simple pictures, such as geometrical shapes. Ask them to explore more and create different pictures.
5. Show them some pictures and ask them to create the pictures in Logo.
6. Ask them to create good designs on paper first and then on Logo. Ask them why their design is good, then compare with the condition of **good designs must use movements and must be eye catching, and should have color, different shapes, and variety of thing**.
7. Explain and demonstrate to the kids how to do mathematics calculation in Logo.
8. Explain and demonstrate to the kids how to write Logo program using FMSLogo editor, so they are able to recreate their pictures.
9. Ask them to go to the Internet and search some designs and pictures created with Logo, and then ask then to write the sample programs.

10. Explain and demonstrate to the kids how to play sounds and music in Logo. Give them the file **player.lgo** and ask them to load it into FMSLogo. Ask them to play some notes and to explore more notes and instruments.
11. Demonstrate to them a simple children song played with Logo, then ask them to write and play the song in Logo.
12. Finally ask them to combine creating good designs or picture and a song or music. This will challenge them and be interesting to them.

Some examples of interesting artworks created using Logo can be found at the **FMSLogo Workshop** web site (http://fmslogo.sourceforge.net/workshop/), **Logo 15-word challenge** web site (http://www.mathcats.com/gallery/15wordcontest.html).

Referrences

Anonymous. *FMSLogo 6.28.0 Tutorial*.

Abhay B. Joshi and Sandesh R. Gaikwad (2012). *Logo Programming (Part 1) - a creative and fun way to learn mathematics and problem-solving* (2 ed). Pune: SPARK Institute. **http://works.bepress.com/abhay_joshi/1**.

Abhay B. Joshi and Sandesh R. Gaikwad (2012). *Logo Programming (Part 2) - a creative and fun way to learn mathematics and problem-solving* (2 ed). Pune: SPARK Institute. **http://works.bepress.com/abhay_joshi/2**.

Anonimous. *FMSLogo: An Educational Programming Environment*. **http://fmslogo.sourceforge.net/**

David Costanzo. *Logo Workshop*. **http://fmslogo.sourceforge.net/workshop/**

Terrapin Software (2013). *Why Use Logo*? **http://www.terrapinlogo.com/why-use-logo.php**.

Wikipedia (2019). *Logo (programming language)*. **http://en.wikipedia.org/wiki/Logo_(programming_language)**.