KOJO

An Introduction

By Lalit and Vibha Pant

Play . Learn

www.kogics.net

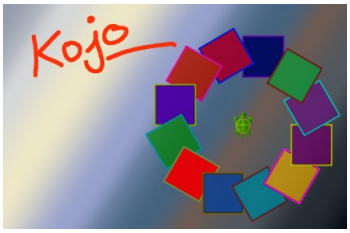
Front Matter

This is the first in a series of e-books about using Kojo to play with Computer Programming, Art, and Math. This book:

- Introduces the Kojo Environment.
- Makes you familiar with the different features of the Environment within the context of a guided activity.

This book is based on Kojo Version: 2.1

Jan 30, 2014



More information about Kojo is available at http://www.kogics.net/kojo

Table of Contents

1 What is Kojo

Kojo is a fun and friendly graphical environment for playing with and understanding computer programming, math, and other related subjects. You can think of Kojo as:

- A Gym – where you can exercise your brain.
- A Studio – where you can create paintings.
- A Lab – where you can experiment with mathematical and scientific ideas.

When you do traditional Art – you paint on a *canvas* using *oil or water colors* as a medium. With Kojo, you paint on your *computer screen* using *computer programming* as a medium.

You normally do math with *pen and paper*. With Kojo, you use *computer programs* to play and interact with Mathematical ideas.

1.1 Why work with Kojo?

- To practice and develop your systematic thinking skills – by learning and doing computer programming.

Why learn computer programming?

- It's fun!
- The world around you is driven more and more by computer programs. It's good to be in on the secret. Computer programming has been called the *New Literacy* for the 21st century.
- It's a universal human activity, and gives you an opportunity to interact with your peers anywhere around the world.
- To exercise your brain!
- To develop your logical and creative thinking skills.
- To learn to use the computer as a virtual laboratory to play with ideas!
- To hone your reading and writing skills.

- To practice and develop your creative and artistic thinking skills – by creating beautiful paintings using geometric shapes.

- To get better at Math:
    - By developing the kind of thinking skills required by Math (which are very similar to the thinking skills required by computer programming).
    - By using computer programs to create mathematical objects, thereby understanding them better.
    - By interacting and experimenting with Mathematical concepts inside a Virtual Lab.
    - By *applying Math* within the activities and projects that you do within Kojo.
- To gain a deeper understanding of computers, and to become proficient at using them.

And much more! The Kojo Page (http://www.kogics.net/kojo) gives you more information about the kinds of things that you can learn as you play with Kojo.

1.2 An overview of Kojo

To get a good overview of the kinds of things that you can do with Kojo, go through the Kojo Overview story:

- Click on the *Stories -> Kojo Overview* menu item. This will run the story.
- Navigate within the story with the help of the buttons at the bottom of the Storyteller window.

What's a Kojo story?

A story is an interactive presentation that runs within Kojo. Stories are written using a combination of two computer languages – Scala and HTML.

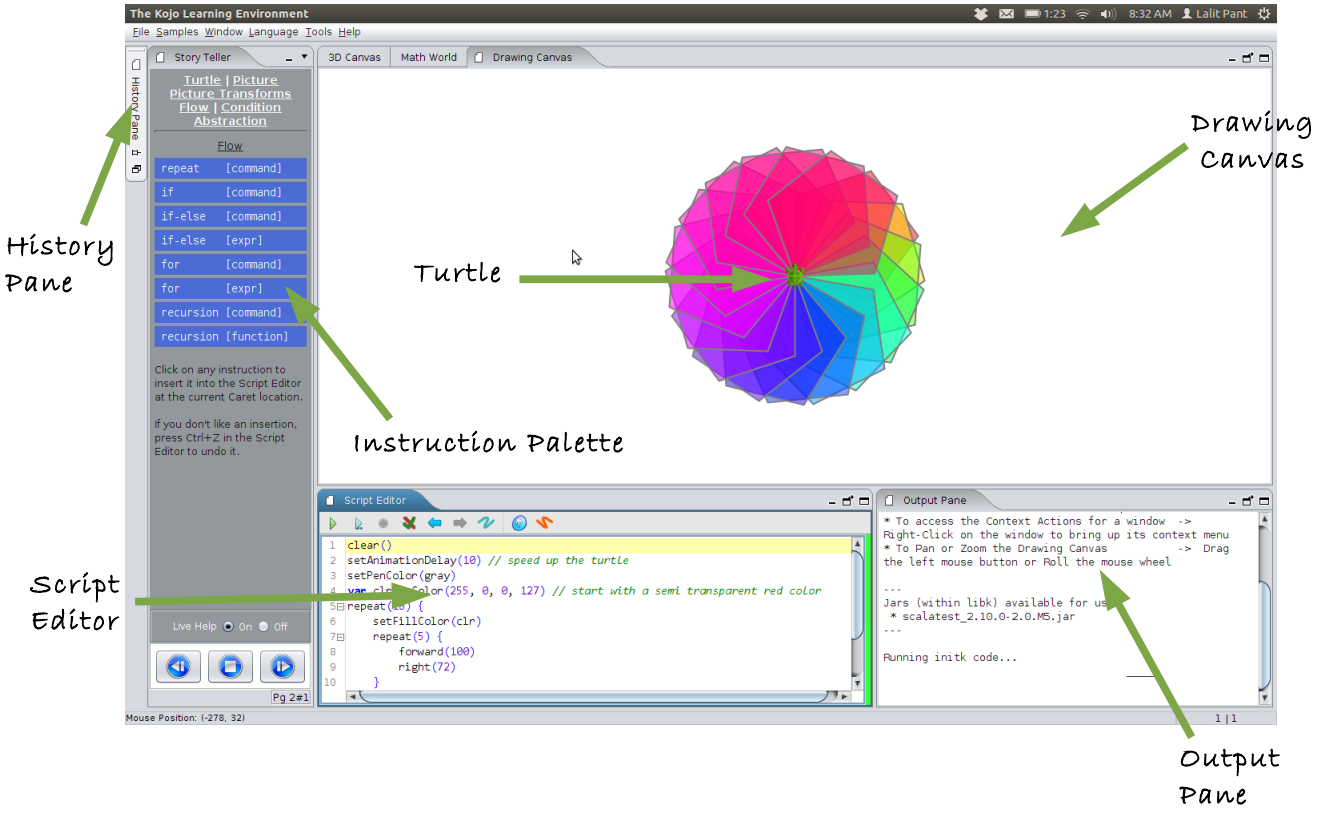
Kojo has many different components that allow you to work with different subject areas – on fun and exciting projects. Some of these components are:

- The Turtle Module – for turtle graphics (in which you drive an on-screen turtle to make paintings).
- The Pictures Module (which is really an extension of the Turtle Module) – for making more complex pictures using shapes made by turtles, and then using these pictures in animations and games.
- The Staging Module – for 2D graphics, animations, and games (this is an alternative approach to animation and game making).

- Mathworld – for experimenting with Math, and creating Math aware shapes.
- The Storyteller – for creating and playing stories (which are interactive presentations).
- The Music Module – for Music Composition and Playing.

2 The Kojo User Interface

Let's start getting familiar with Kojo by looking at the different elements of the Kojo User Interface...



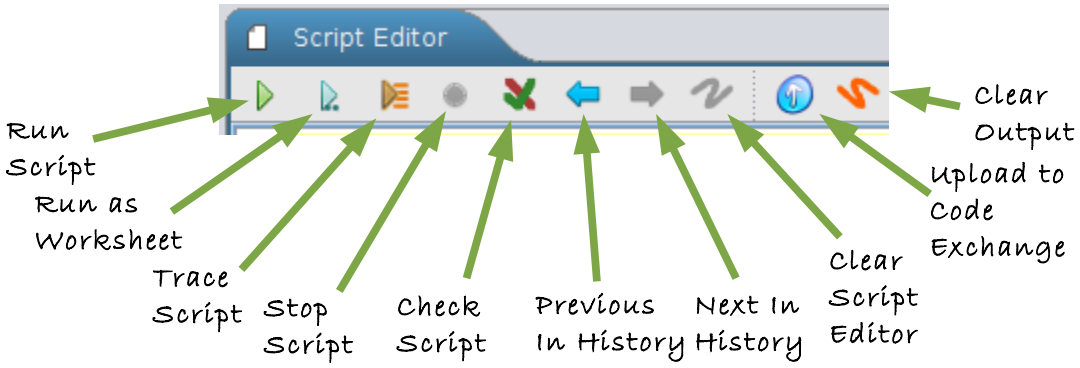
Within Kojo, you get a pet turtle to do your bidding – to make geometric shapes and paintings. You tell the turtle what to do, and the turtle does it for you. The main features of the Kojo User Interface are:

- **The Turtle** – this is the creature that follows your instructions to draw things on the screen.
- **The Script Editor** – this is where you write your programs (or scripts) that instruct the Turtle to do things on the screen.

> A **program** is a series of instructions for the computer.
>
> A **script** is a small program.

- **The Instruction Palette** – lets you write your programs *visually*, by pointing and clicking with the mouse.

- **The Drawing Canvas** – this is the area of the screen where your drawing shows up.
- **The Output Window** – this is where Kojo tells you things that you need to know, like the fact that there's a problem with the script that you want the Turtle to follow.
- **The History Pane** – stores all the commands and scripts that you run, so that you can easily call them up again.
- **The Script Editor Toolbar** – has buttons that allow you to do core tasks within Kojo:



  - The **Run Script b**utton – runs your script i.e. the contents of the Script Editor.
  - The **Run as Worksheet** button – runs your script as a worksheet, to let you see expression types and values in-line, right within the Script Editor (don't worry if that does not make sense right now).
  - The **Trace Script** button – lets you trace your script to let you see, line by line, what it does.
  - The **Stop b**utton – allows you to stop a script that is running. It also allows you to stop runaway scripts that are taking too long to finish.
  - The **Check Script** for Errors button – helps you to precisely locate errors in large scripts.
  - The **Previous in History** button – calls up, within the Script Editor, the last command/script that you ran.
  - The **Next in History** button – along with the previous button, allows you to move back and forth within your command/script history.
  - The **Clear Script Editor** button – Clears the script editor, making it easy for you to start writing a new script.
  - The **Upload to Code Exchange** button – let's you upload your code to the Kojo Code Exchange, to share it with people around the world.
  - The **Clear Output** button – Clears the output pane, making it easy for you to look at the output of scripts that you run after that.

3 A Guided tour of Kojo

In this section, you will get to work your way through the creation of a simple Geometric painting. Along the way you'll learn about:

- The programming language used within Kojo.
- Making step-by-step progress towards the creation of a program that generates the desired painting.
- Manipulating the text of your programs.
- Recovering from errors.
- Working with the History Pane.
- Saving the program in a file after it is done.
- Loading the program from the saved file within a new Kojo session.

3.1 The Programming Language

Kojo supports a very powerful language called Scala. Everything that you write within the Script Editor will be in the Scala language.

More information about Scala is available on the Scala website: http://www.scala-lang.org.

You might find it interesting to know that the Kojo Environment itself is written in the Scala programming language.

One of the great things about Scala is that it makes it very simple to do simple things (while still allowing much more complex things with a little more effort).

In this section, you will be looking at a very small and simple subset of Scala that will allow you to explore Kojo. This will include some turtle commands and a flow control command.

3.1.1 Kojo Commands

You use commands within a program to:

- Get the program to take some action (like drawing something on the screen).
- Have some effect that modifies the future behavior of the program (like changing the colors used by the program to draw things on the screen).

Let's look at a few predefined Kojo commands:

`forward(numberOfSteps)` – moves the turtle forward; the `forward` command takes one input – the number of steps (in pixels) to move forward.

Example – if you run the command:

```
forward(100)
```

The turtle will move forward 100 steps and draw a line along the path that it travels.

`right()` – turns the turtle right through 90 degrees at its current position.

`clear()` – clears the canvas area and gets the turtle back to its original position.

`setPenColor(color)` – specifies the color of the pen that the turtle draws with.

Example: `setPenColor(blue)` will make a line drawn after the command blue in color.

`setFillColor(color)` – specifies the interior fill color of the figures drawn by the turtle.

Example: `setFillColor(green)` will make a square drawn after the command green in color.

3.1.2 Flow control with repeat

Used individually, the commands above allow you to instruct the Turtle to do a particular thing as described. You can also put many commands together in a sequence to instruct the turtle to do a series of things. For example, the following script gets the turtle to make a particular geometric shape (can you figure out what it is without running the script?):

```
forward(100)
right()
forward(100)
right()
forward(100)
right()
forward(100)
right()
```

How about if you want to do a certain number of things, and then go back and repeat them? For example, you can see that in the previous example, the forward and right commands are used over an over again. Surely there's a better way of doing this.

There is! The repeat command allows you to instruct the turtle to repeat a sequence of other commands for a specified number of times. Using repeat, the previous script becomes:

```
repeat (4) {
  forward(100)
  right()
}
```

Isn't the intent of the script (the shape it makes) much clearer now?

How does Kojo know how many commands to repeat? You tell it by putting the commands to be repeated within a *block* – which is enclosed within an opening and a closing brace. Here's another example of a block with multiple commands inside it:

```
{
    forward(100)
    left(45)
    back(100)
    right(45)
}
```

With a basic knowledge of the commands described in this section, you are now in a position to start exploring how to create a simple drawing with Kojo.

3.2 Taking Kojo for a spin

Let's get our hands dirty and try out a few things!

3.2.1 Making a simple Shape

Let's start with something that you have seen before:

```
repeat (4) {
  forward(100)
  right()
}
```

Type this into the Script Editor. But wait, you don't have to type in all of the above. Kojo is there to help you with a feature called code-completion.

Code is the name given to the instructions in your script, because they represent a special *code* that the computer understands.

There are two different types of code:

- The readable text of your script is called *source code*. It is also sometimes just called *the source* or *the code*
- The actual binary instructions that are understood by the computer are called *machine code*.

Special programs called compilers or interpreters covert from source code to machine code. Kojo has an interpreter inside it that converts the *source code* that you write into *machine code* before running it.

Code-completion (also known as auto-completion or IntelliSense) is a feature that enables Kojo to automatically complete a word (of source code) that you are typing based on the current context and the letters that you have already typed. Code-completion will save you a lot of typing grunt-work as you write programs within Kojo.

Code-completion will also provide you opportunities to explore new commands within Kojo.

Let's get Kojo to do some of your typing work for you. Punch in the word *rep* and then press Control+Space (Control and Space keys together) to activate code completion.

Kojo will infer that you are trying to write a repeat statement, and will input the following into the Script Editor:

```
repeat (n) {


}
```

Kojo does not know how many times you want to repeat something, so it just puts an *n* within the parenthesis (where *n* stands for a number that you need to provide). Very conveniently, the *n* is highlighted, and the cursor is positioned right over it, so that you can just type in the repeat count that you want. If you type in 4, n will be replaced by 4.

Now, just hit the *Tab* key. Kojo will automatically position your cursor at the right location on the next line for you to start entering the first command within your repeat block.

Next, type in forw. The script editor should now contain:

```
repeat (4) {
    forw
}
```

Hit Control+Space to have Kojo fill in the forward command for you.

Proceed in this way till you have the script for making a square typed in. Now click on the *Run Script* button. The turtle will start moving and make a nice square for you.

Congratulations! You just wrote your first Kojo program.

Now type in the following code and run it (everything after a '//' on a line is a comment for human consumption and is ignored by Kojo):

```
clear()
setAnimationDelay(10)
setPenColor(yellow)
repeat(15) {
    setFillColor(blue) // click on the word blue to interactively change fill color
    repeat(4) {
        forward(100)
        right(90) // click on the number 90 to interactively change turn angle
    }
    right(360/15)
}
```

Try to understand what the program does. To determine what any particular command does, bring up a code-completion popup for that command. Code-completion popups include context specific online help.

Also, trace the program using the **Trace Script** button in the Script Editor toolbar. This will give you good insights into how the program works.

3.2.2 Accessing History

The History Feature gives you a convenient way to access the commands/scripts that have been run by you .

You can access and switch between recently run commands in three ways:

- Using the History Pane.
- Using the History buttons in the Script Editor toolbar.
- Using the Keyboard:
    - Control+Up-Arrow loads the previous command/script in the script editor.
    - Control+Down-Arrow loads the next command/script in the script editor.

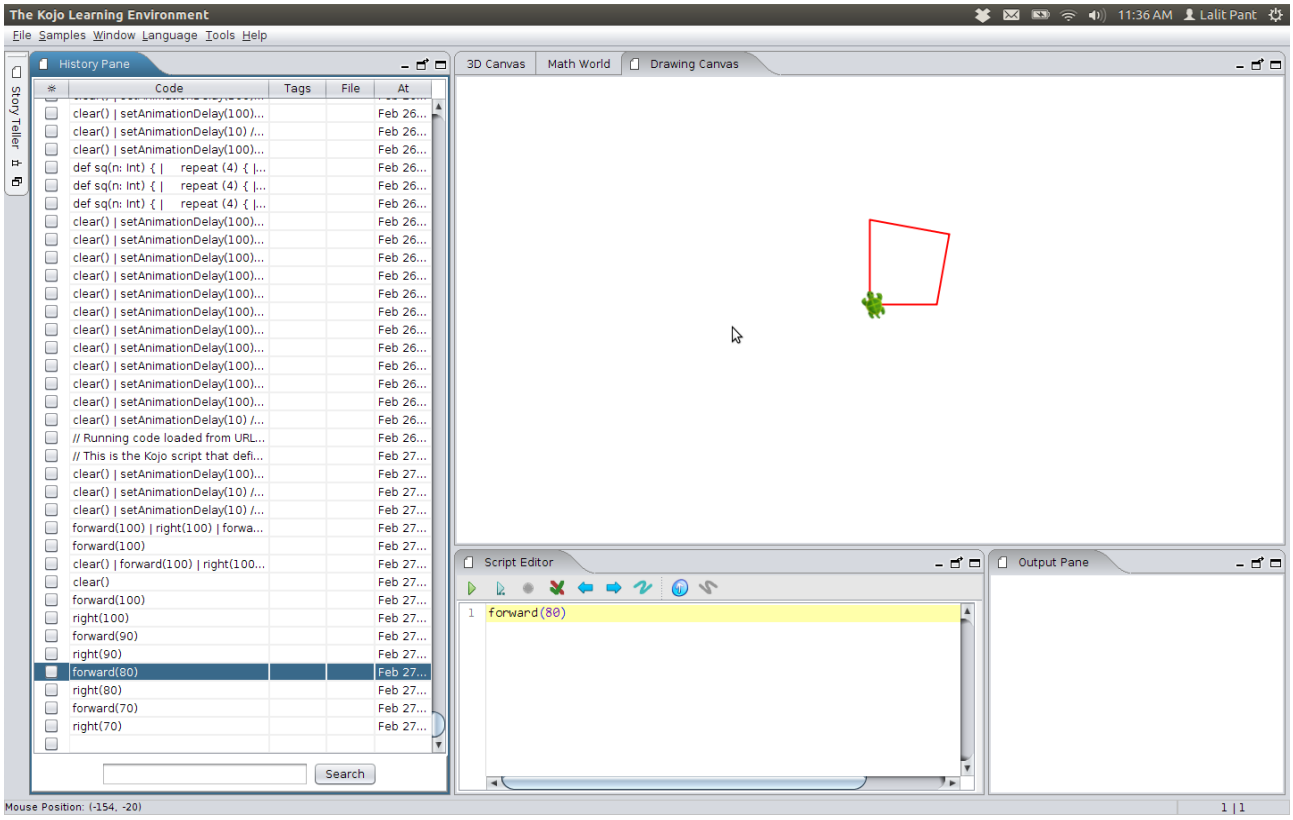
Let's see how you can use the History Pane to access history.

First, click on the Restore button in the minimized **History Pane** on the left of the Kojo Workspace to show the History window (which is minimized by default):

Then, run the following commands, one by one, to create some items in your history:

- `clear()`
- `forward(100)`
- `right(100)`
- `forward(90)`
- `right(90)`
- `forward(80)`
- `right(80)`
- `forward(70)`
- `right(70)`

The commands that you just ran should be visible in the History Pane, as shown below:



Click on the `right(90)` command; notice that it appears in the Script Editor.

Click on the `forward(70)` command. See it show up in the Script Editor.

Try this with a few other commands in your history. Notice that you can run the commands that show up in the Script Editor after clicking the History Pane – just like you would run something that you typed in yourself.

Now, try accessing the commands in your history using the history buttons in the Script Editor toolbar.

Next, try accessing the commands in your history using the Control+Up-Arrow and Control+Down-Arrow keys.

History Features

The History facility within Kojo gives you some useful features to locate code that you have written in the past:

- Tags – you can tag any item in history with words of your choice, to help you locate it later via the Search feature.
- Stars – you can *star* any item by checking a box next to it – to make it easy to visually locate it in the future.

- Search – allows you to search for items in your History that match the given text. The search is case-sensitive, and is done across script text, tags, and filenames (if your script lives in a file).

3.2.3 Playing with Script Text

As you work with Kojo, you will be spending a lot of time within the Script Editor. It is useful to know about some of the powerful capabilities of this window...

Formatting

When you write a computer program, you have two different audiences for the program:

- The computer, which will run your program
- Other programmers, who will read your program, understand it, learn from it, and possibly extend it.

When you format your code, you make it easier for other programmers to understand the structure of your code. Take a look at the following two snippets of code, one badly formatted, and the other nicely formatted:

```
repeat(4){ forward(100)
right()}
```

```
repeat (4) {
    forward(100)
    right()
}
```

Which one do you think is easier to understand?

Kojo makes it very easy to format the code that you have written:

- Press Ctrl+Shift+F
- **Or** Use the *Format Source* menu item in the Script Editor context menu.

The one thing that you need to do to get Kojo to help you with code formatting – is to get your new-lines right. For example, let's take the badly formatted code that we looked at earlier:

```
repeat(4){ forward(100)
right()}
```

To make code nicely formatable, you should follow the rule that an opening brace should never have any text after it on the same line, and a closing brace should never have any text before or after it on

the same line. Applying this rule to the code above, we get:

```
repeat(4) {
forward(100)
right()
}
```

Now, if we press Ctrl+Shift+F, Kojo will format the above code for us and make it look like this:

```
repeat(4) {
    forward(100)
    right()
}
```

That is exactly what we wanted!

Selecting Text

You can select a fragment of text by using either the Mouse or the Keyboard:

- Mouse – left-click and then drag your Mouse pointer over the text fragment that you want to select.
- Keyboard – go to the beginning of the text fragment that you want to select, press the Shift key, and then move the cursor by using the right/down arrow keys till you get to the end of the text fragment.

Exercises

1. Using the History Pane, pull up the script for making a square. Try to select the text within the repeat block by using first the Mouse, and then the Keyboard.

Why would you want to know how to select text? Because once you have a selected fragment of text, you can do things with it:

Copy & Paste

Let's say you have some code to move the turtle:

```
forward(100)
right()
```

You now want to add some additional commands, say `forward(100)` and `right(45)`, below the existing commands. This is very easy to do with Copy & Paste:

- Select the two lines that contain the existing commands.
- Copy them into the clipboard by pressing Ctrl+C or using the Copy command from the Script Editor context menu.
- Move the cursor to the line where you want to add the two new commands.
- Paste the lines from the clipboard into the location specified by the cursor by pressing Ctrl+V or using the Paste command from the Script Editor context menu.

Your code should now look like this:

```
forward(100)
right()
forward(100)
right()
```

You can now go into the fourth line and add *45* within the `right()` command to get the code that you want.

Do you agree that using Copy & Paste is a powerful way to add new code to your program? Or do you think it's easier to type in all the things that you want in your program with the help of code-completion? In either case, it's useful to know both techniques.

Cut & Paste

Let's now say that you have two lines of code within your program:

```
forward(100)
right()
```

You now want to reverse the order of these two commands. A very easy way to do this is to:

- Select the first line.
- Cut it (by pressing Ctrl+X or using the Cut command from the context menu) to delete it from the program and put it into the Clipboard.
- Move the cursor to the line below the right() command.
- Paste the line from the clipboard into the location specified by the cursor.

Voila! You've done what you wanted without having to type anything in. That was easy, wasn't it?

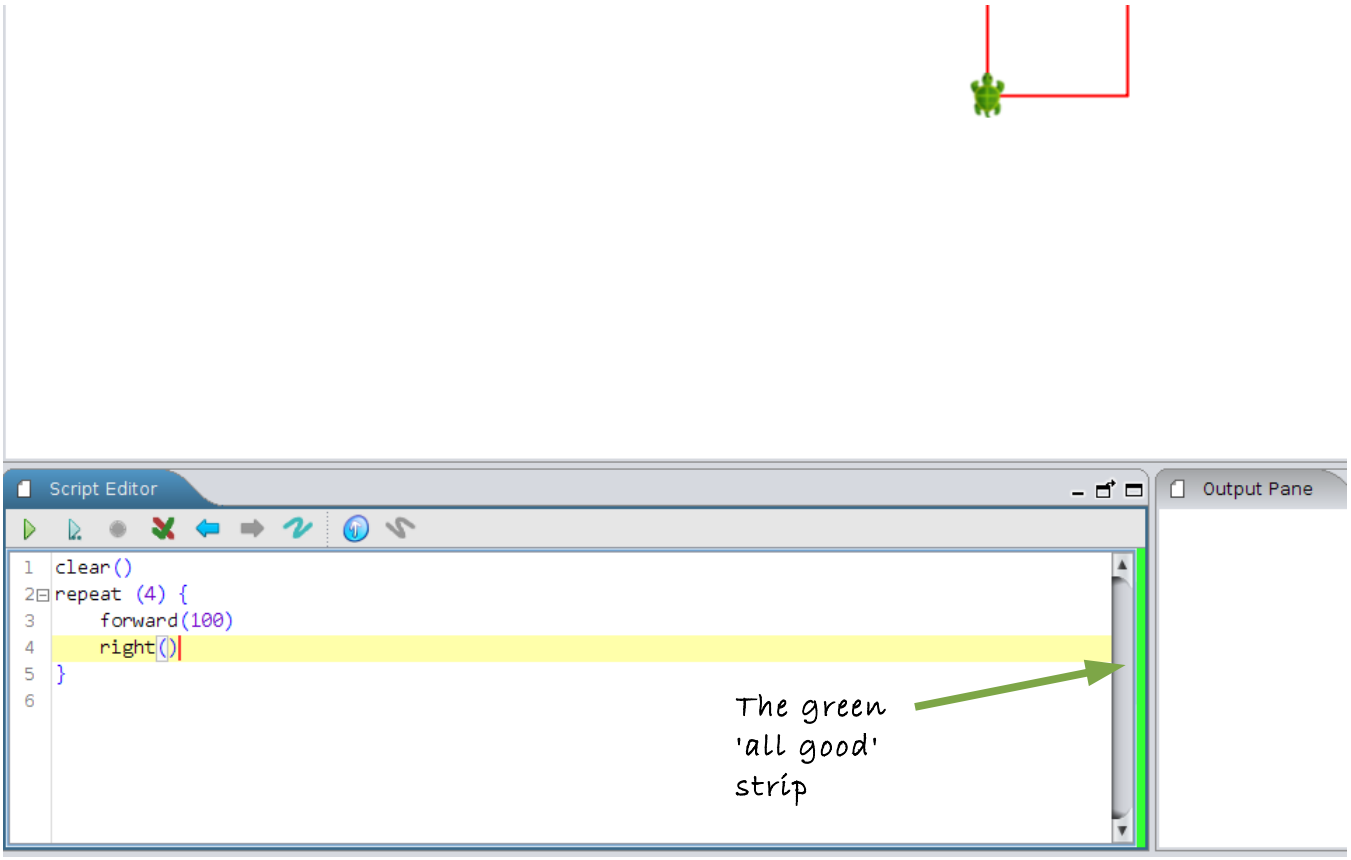
Running Selected Text

Once you select one or more lines of text, clicking on the Run button will run just those lines of code, as opposed to running all the code within the Script Editor. This helps with experimentation and debugging in your programs.

3.2.4 Recovering from Errors

Once again, run the script to make a square:

```
repeat (4) {
  forward(100)
  right()
}
```



Notice that the Script Editor window displays a green strip on its right margin. This is meant to indicate that the script has no errors.

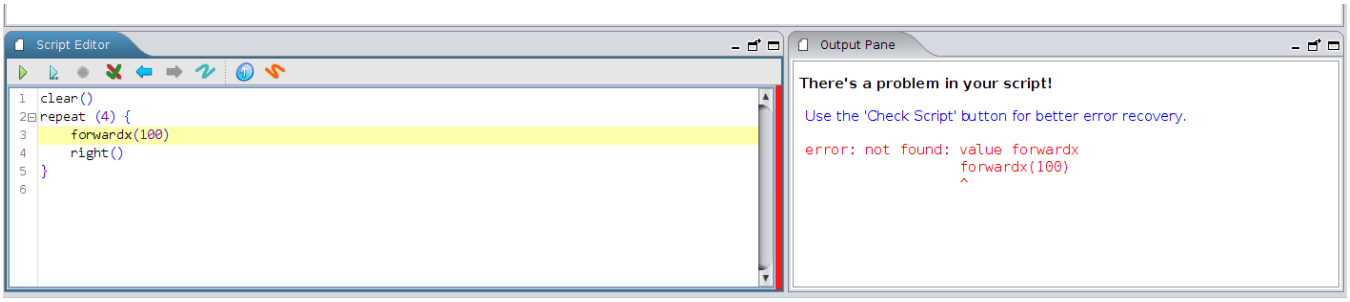
Some of the things in your script that will be considered errors are:

- References to non-existent commands .
- Misspelled commands.
- Misplaced or Mismatched braces – {}
- Misplaced or Mismatched parentheses – ()

Let's see what Kojo does if your script does have an error. In the script that you just ran, mis-spell the forward command by adding an x at the end. The script editor should now look like this:

```
repeat (4) {
  forwardx(100)
  right()
}
```

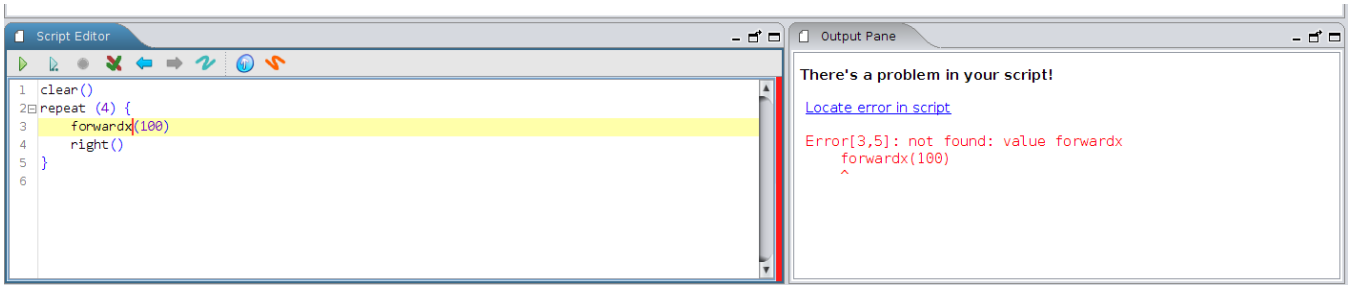
Go ahead and run this script.



Notice the following:

- The right margin of the script editor is red. This indicates that there is an error in the script.
- The output window has an error message describing exactly what the error is.
- The output window has a recommendation for you to use the *Check Script* button.

Try the recommendation – click on the *Check Script* button.



The right margin of the Script Editor is red again (indicating an error), and you see a similar error message. But there's something new. There's a hyper-link right at the top that allows you to locate the error in your script (a hyper-link is something that allows you to jump from one location to another).

Click on the link. You will find that Kojo will take you to the exact location in your program where the error is present. You can now fix the error.

The Check Script button is very useful when you're hunting down and fixing errors in your scripts (especially for larger scripts) – because it allows you to jump to the location in the Script Editor where the problem is present. This helps greatly with first identifying and then fixing errors.

3.2.5 Saving and Loading your work

You are now done working on your masterpiece for the day. You are confident that you will not lose any work that you have done – because Kojo saves every command/scripts that is run, and makes it available within the History Pane. But you might still want the facility of saving a particular script that you wrote into a file (to save you the trouble of hunting for it within the History Pane).

You can do this by using the *File → Save As* menu item in the Popup menu that comes up.

In a future session of Kojo, if you want to load this script into the Script Editor, you can do this via the *File → Open* menu item.

In both the above cases, the Kojo Script Editor associates your script with a file. As you do additional work on your script, you can save the work back to the file by clicking on the *File → Save* menu item or by pressing Control+S.

Exercises

- Save the contents of the Script Editor in a file called test-save.kojo.
- Shut down and then restart Kojo.
- Load the test-save.kojo script into the Script Editor.
- Make sure the Script Editor contains the code that you saved.

4 More User Interface features

4.1 Context Sensitive Actions

Context-sensitive actions are things that you can do with a particular window. Right-clicking on a window shows a popup with the available actions for that window.

These actions are called context-sensitive because they depend on the window (the context) that you are currently working with.

Exercises

1. Right click on the Script Editor and make a note of the available actions. Do the same for the Output Pane. Are the actions for the two windows different?

4.2 Panning and Zooming

Within the Drawing Canvas, you can:

- Pan (move the drawing around) by pressing the left mouse button and dragging.
- Zoom (make the drawing bigger/smaller) by using the mouse scroll-wheel or by pressing the Shift key, and then pressing the left mouse button and dragging.

You can reset pan and zoom levels in two different ways:

- Right-clicking on the Drawing Canvas to bring up the context menu, and then clicking on the *Reset Pan and Zoom* menu item.
- Resizing the Drawing Canvas (resizing a window means making it bigger or smaller by dragging one of its borders).

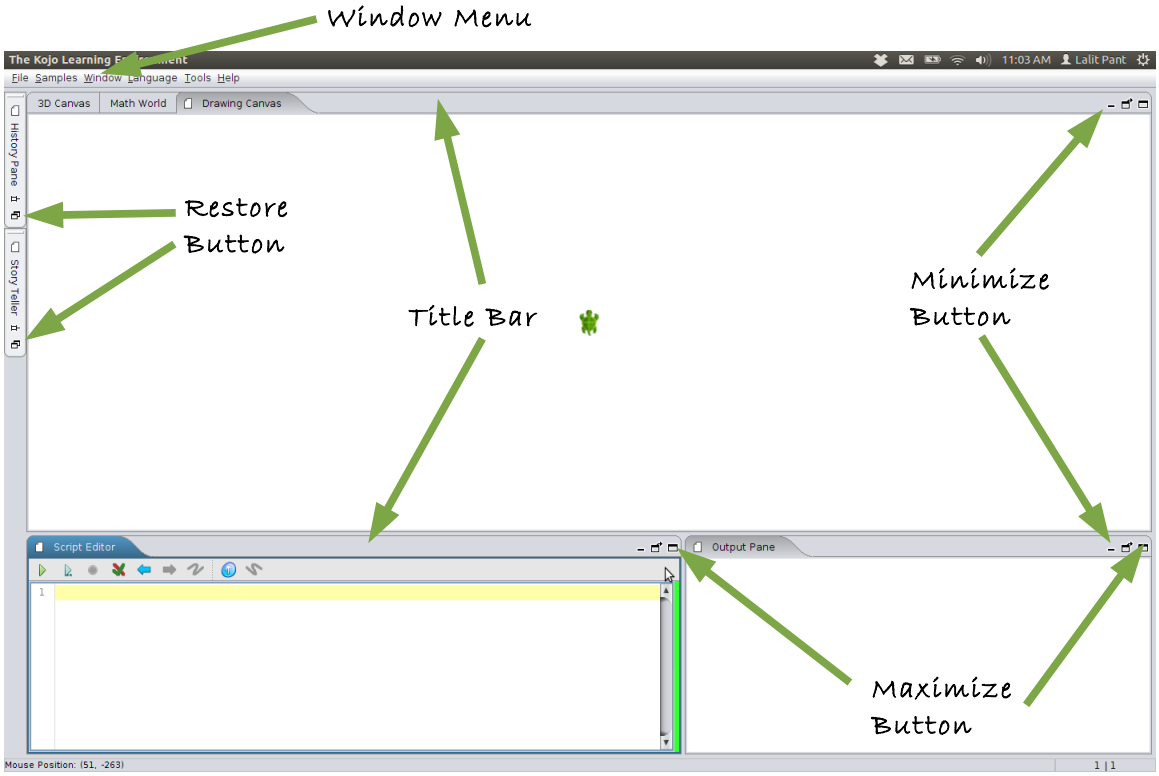
Exercises

1. Pan the Drawing Canvas (by pressing the left Mouse button and dragging) to move the turtle around.

2. Zoom the Canvas (by pressing the right Mouse button and dragging) to make the Turtle bigger or smaller.

4.3 Window Management

Kojo has a flexible workspace. The following figure shows you the workspace and highlights some of the elements that let you interact with the different Kojo windows.



The following are some of the things that you can do to control the windows within the Kojo workspace:

4.3.1 Maximizing Windows

When you maximize a window, it grows in size to fill up all the screen space allocated to Kojo. You can maximize a window by:

- Double-clicking on its title bar.

- Pressing Ctrl+M when it has focus.
- Or clicking on its Maximize button. Once a window is maximized, its Maximize button changes to a Restore button.

You can restore a maximized window to its earlier size by:

- Double-clicking on its title bar.
- Pressing Ctrl+M when it has focus.
- Or clicking on its Restore button.

Exercises

1. Maximize the Drawing Canvas, and then restore it to its original size.
2. Maximize the Script Editor, and then restore it to its original size.

4.3.2 Minimizing Windows

When you minimize a window, it shrinks and becomes a label at the side of the Kojo workspace. You can minimize a window by clicking on its Minimize button.

If you click on the label of a minimized window, it will slide out to show you its contents. If you click in the Kojo workspace outside the window, it will slide back in.

To restore a minimized window, you need to click on its Restore button.

Exercises

1. Minimize the Script Editor, pop it out by clicking your mouse pointer on the minimized window label, minimize it again by clicking outside the window, and then restore it to its original size.
2. Restore the History Pane so that it's visible within the workspace, and then minimize it.

4.3.3 Moving and Docking Windows

If you want to change the arrangement of the different windows within the Kojo workspace, you can click on a window's title bar and drag it to the new location that you want to place it at.

Note: If you mess up the Kojo workspace while playing with the different windows, you can click on the

Window → Default Perspective menu item to restore the windows to their default locations.

5 Concluding Checklist

After reading and playing along with this book, you should be comfortable doing the following:

- Identifying the various Kojo windows and their functions.
- Using the Script Editor toolbar buttons.
- Running a script.
- Locating errors within a script.
- Using the History Pane to locate previously run scripts.
- Saving a script to a file.
- Loading a script from a file.
- Formatting code within the Script Editor.
- Cutting, Copying, and Pasting text within the Script Editor.
- Running only a few lines from within the Script Editor by first selecting them.
- Panning and zooming the Drawing Canvas.
- Accessing context sensitive actions within the different windows.