Scratch 1.3

Michael Badger

Chapter No. 7
"Games of Fortune"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.7 "Games of Fortune"

A synopsis of the book’s content

Information on where to buy this book

About the Author

Michael Badger is a technical communicator with a history of helping others use their computer software and technology. For fun, Michael reads computer books and blogs about technology. When he finally decides to disconnect, he spends his spare time fishing, growing pigs, raising honeybees, and tending the family.

Michael also wrote Zenoss Core Network and System Monitoring, a step-by-step guide to configuring, using, and adapting the free Open Source network monitoring system.

Share your feedback about this book at `http://www.scratchguide.com`.

Writers work hours at a time in isolation, but bringing a book from concept to finished product requires the support of many people. My support starts at home with my wife Christie's encouragement and my son Cameron's early bedtime. Cameron, when you're old enough to read, we'll create some games together.

Early in this project, I received thoughtful, timely, and sensible feedback from my editor David Barnes. Thanks for the great advice.

I thank the reviewer, who took time to read and respond critically to my work for which I will be forever grateful. Know that I appreciate every correction, suggestion, and improvement he offered. This book benefits from his attention.

Scratch 1.3

When we program, we solve problems. In order to solve problems, we think, take action, and reflect upon our efforts. Scratch teaches us to program using a fun, accessible environment that's as easy as dragging and dropping blocks from one part of the screen to another.

In this book, we will program games, stories, and animations using hands-on examples that get us thinking and tinkering. For each project, we start with a series of steps to build something. Then, we pause to put our actions into context so that we can relate our code to the actions on Scratch's stage. Throughout each chapter, you'll encounter challenges that encourage you to experiment and learn.

As you begin working through the examples in the book, you won't be able to stop your imagination, and the ideas will stream as fast as you can think of them. Write them down. You'll quickly realize there are a lot of young minds in your home, classroom, or community group that could benefit from Scratch's friendly face. Teach them, please.

What This Book Covers

Chapter 1 provides an overview of Scratch, its features, and how it can help you teach 21st century learning skills to your children and students.

Chapter 2 guides us through the installation of Scratch on Windows, OS X, and Linux. This chapter also helps you run the Scratch programming environment from a USB flash drive.

Chapter 3 explores the Scratch interface and allows us to create some simple scripts that demonstrate how easily we can build a project. This is a high-speed tour of Scratch that gets us tinkering and thinking about what's possible.

Chapter 4 teaches us how to create an animated birthday card and a slideshow of our favorite photos.

Chapter 5 allows us to horse around as we develop a barnyard humor book that lets us narrate multiple scenes. There's no need to hold the applause.

Chapter 6 takes a classic pong game and gives it a little personality by adding a troll, switching levels, and keeping score.

Chapter 7 takes us to the fortune-teller, but before we learn the random answers to all our deepest questions, we must create our game using the Magic 8 ball's fortunes.

Chapter 8 uses mathematical formulas and graphs to help us answer the question, "Would you rather have a dollar that doubles every day or a lump sum of money?" The answer may surprise you.

Chapter 9 explains how to share your project with the Scratch community and how to promote it to you friends and fans.

Chapter 10 shows us how to connect an external sensor board to our computer and delivers real-world stimuli as input to Scratch projects.

7
Games of Fortune

In Chapter 6, we learned how easy it is to create projects that incorporate dynamic information using variables. However, variables have a limitation; they store only one value at a time. Sometimes, we want a variable to store multiple values.

Welcome to lists. In Scratch, a list allows us to associate one list (a variable) with multiple items or values in much the same way we create a list before going to the grocery store.

In this chapter, we will take a trip to the fortune-teller to demonstrate lists, and I predict you'll learn how to:

- Store and retrieve information in lists
- Add and remove items from the lists
- Keep track of items in a list by using a counter
- Identify intervals using the **mod** block
- Use if/else control blocks to make decisions

That's a lot to process, but we can do it.

Fortune-teller

Most of us enjoy a good circus, carnival, or county fair. There's fun, food, and fortunes. Aah, yes, what would a fair be without the fortune-teller's tent? By the end of the chapter, you'll know everything you need to spin your fortunes and amaze your friends with your wisdom.

Before we start the first exercise, create a new project and add two sprites. The first sprite will be the seeker. The second sprite will be the teller. Choose any sprites you want. My seeker will be a clam and my teller will be a snowman. If you want to add a background, go ahead.

Time for action – create a list of questions

In order to have a successful fortune-telling, we need two things: a question and an answer. Let's start by defining some questions and answers:

1. Select the **seeker** from the list of sprites.
2. From the **Variables** palette, click the **Make a list** button.
3. In the list name dialog box, type **questions** and select **For this sprite only**.
4. Click **OK** to create the list. Several new blocks display in the **Variables** palette, and an empty block titled **seeker questions** displays on the stage.

5. Let's think about a couple of questions we may be tempted to ask, such as the following:
   - Will my hair fall out?
   - How many children will I have?

6. Let's add our proposed questions to the questions list. Click the **plus sign** located in the bottom-left corner of the seeker questions box (on the stage) to display a text input field. Type **Will my hair fall out?**

7. Press the plus sign again and enter the second question**: How many children will I have?** We now have two questions in our list.

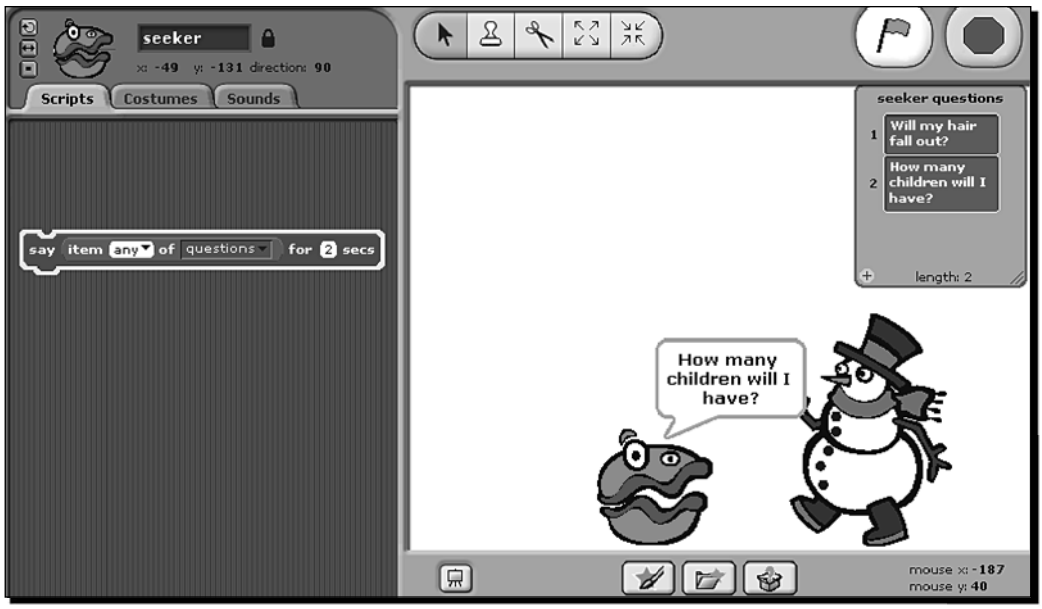
To automatically add the next item in the list, press enter.

8. Let's add a **say for 2 secs** block to the scripts area of the **seeker** sprite so that we can start the dialog.

9. From the **Variables** palette, drag the **item of questions** block to the input value of the **say for 2 secs** block.

10. Double-click on the block and the seeker asks, "**Will my hair fall out?**"

11. Change the value on the **item** block to **last** and double-click the block again. This time the seeker asks, "**How many children will I have?**"



What just happened?

I'm certain you could come up with a hundred different questions to ask a fortune-teller. Don't worry, you'll get your chance to ask more questions later.

Did you notice that the new list we created behaved a lot like a variable? We were able to make the questions list private; we don't want our teller to peek at our questions, after all. Also, the list became visible on the screen allowing us to edit the contents.

The most notable difference is that we added more than one item, and each item corresponds to a number. We essentially created a numbered list.

If you work with other programming languages, then you might refer to lists as arrays.

Because the seeker's questions were contained in a list, we used the **item** block to provide special instructions to the **say** block in order to ask the question. The first value of the **item** block was position, which defaulted to one. The second value was the name of the list, which defaulted to questions.

In contrast, if we used a variable to store a question, we would only need to supply the name of the variable to the **say** block. We saw those examples in Chapter 6.

Have a go hero

Create an answers list for the **teller** sprite, and add several items to the list. Remember, there are no wrong answers in this exercise.

Work with an item in a list

We can use lists to group related items, but accessing the items in the list requires an extra level of specificity. We need to know the name of the list and the position of the item within the list before we can do anything with the values.

The following table shows the available ways to access a specific item in a list.

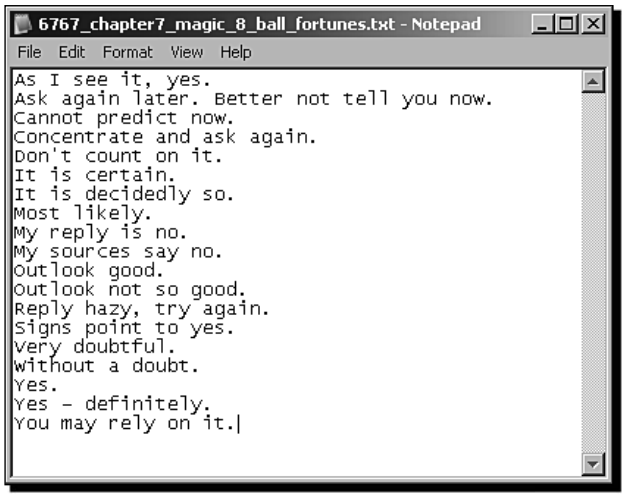
Position	Description	Uses
First	Identifies the first item in the list.	Insert, delete, replace, or retrieve the first item in the list.
Any	Selects a random item in the list.	Insert, replace, or retrieve a random item in the list.
Last	Selects the last item in the list.	Insert, delete, replace, or retrieve the last item in the list.

Position	Description	Uses
Variable	Specifies a variable that contains a number instead of the default first, any, or last values.	Use the variable to store the position of an item in the list, then insert, delete, replace, retrieve the item that corresponds to the value.
Manual input	Enters a specific item number.	Insert, delete, replace, or retrieve a constant item number, such as item 5.

Import a list

Entering one item at a time via the Scratch interface is functional, but the small size of the list box can be difficult to use when we need to add a large number of items. Fortunately, we can create a text file outside of Scratch and then import it into our list.

Before you continue with the exercise, build your own text file with the fortunes you want to use in response to the seeker's questions. Enter one fortune per line in a text editor, such as Notepad.



You can use any fortunes you want, but I'm going to use a list of common Magic 8 ball responses that I found on Wikipedia at `http://en.wikipedia.org/wiki/Magic_8_ball`. My list will contain 19 items to start.

Time for action – import a list of fortunes

If you haven't already done so, create a new list and name it **answers**.

1. To import the list, right-click on the answers list and choose **import...**. Note: This will erase any items you previously added to the answers list.

2. In the **Import List** dialog box, browse to the fortunes file you saved on your computer prior to starting this exercise and import it.

3. Think of a new fortune that is not yet in your answers list. We are going to add it to the list. I'll use "No comment" as my new fortune.

4. Now, let's add the new fortune to our list. From the Variables palette, find the **add** block. Replace the default value **thing** with your new fortune. Make sure the answers list is selected.

5. Double-click on the **add** block to add the new item to the end of the answers list.

What just happened?

I don't know about you, but I had a much easier time typing in a text editor than I did typing in the list monitor. Actually, if you took my lead, you copied the Magic 8 ball® responses from Wikipedia and pasted them into a text file. That way, all you had to do was to clean up some formatting.

We did add a new fortune to the list via the **add** block. The **add** block always places the new value at the end of the list. Now, my list contains 20 items. As an alternative, we could have updated the original fortunes text file and then re-imported the list.

Reasons to import

It's true that importing a list makes list creation easy. However, we can use the Import List functionality in other creative ways. For example, we could create a game that instructs the player to create a unique list that he or she can import into Scratch. That way, each player can customize the game.

If we create a math game, we could ask the player for a varied set of numbers to make the problems different. You get the idea.

Export a list

Sometimes, we may want to export a list from Scratch to a file. For example, as people play a game, we may want to collect all the scores into a separate list that we can later export to a text file on our computer.

As we create our projects, the contents of a list may change from the original list we imported. In that case, we may want to export the new list.

To export a list, right-click on the list monitor and click **export**. The file automatically saves to the root installation directory for Scratch. In Windows, that is `C:\Program Files\Scratch`.

Pop quiz

1. If you wanted to group ten related items together, you would create a new:
    - Variable
    - Numbers block
    - List
    - Forever loop
2. If you want to add many items to a list at one time, how would you do it?
    - Type each item one at a time into the **add** block
    - Type each item into the list monitor that displays on the stage
    - Export the list from Scratch
    - Create a list in a text file and import it via Scratch

Your fortune is ...

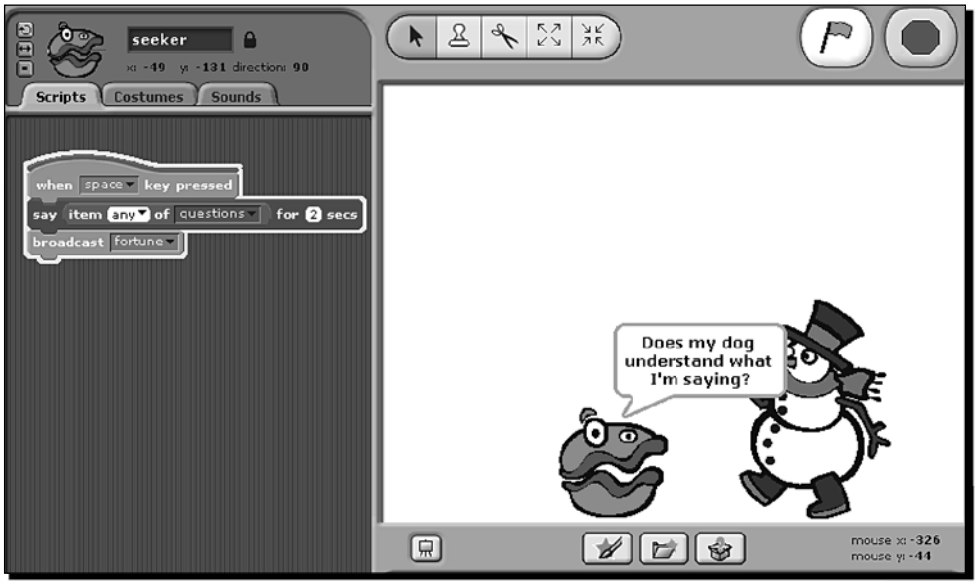
Take a moment to add more questions to the **seeker** sprite's question list.

Now that we have a list of several questions for our seeker to ask and a list of fortunes for our teller to answer, let's create the script that randomly selects a question.

Time for action – tell me a fortune

Before we begin, hide the lists from the stage by right-clicking on each list and selecting **hide** so that they are out of the way. Let's start with the **seeker** sprite. We should have a **say** block in the scripts area from our earlier exercise:

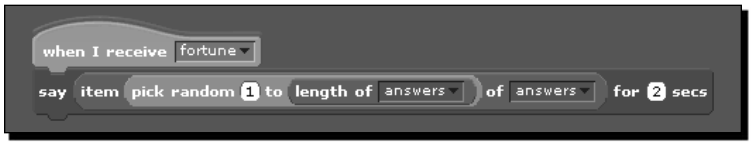
1. We need a way to control our seeker. From the **Control** palette, drag the **when space key pressed** block and attach it to the **say** block.
2. From the position drop-down list in the **item** block, select **any**.Press the space key to ask a question. Each time you press the space key, a random question from the list displays.
3. We need to let the teller know we've asked a question, and that we expect an answer. From the **Control** palette, add the **broadcast** block.
4. Add a new broadcast message titled **fortune**.



5. Every question deserves an answer. Select the **teller** sprite so that we can create the script to provide an answer.
6. From the **Control** palette, drag the **when I receive** block into the scripts area, and make sure **fortune** displays as the message value.
7. From the **Looks** palette, add the **say for 2 secs** block to the **when I receive** block.
8. Let's pick a random item from the answers list. From the **Variables** palette, add the **item** block as the input to the **say** block.

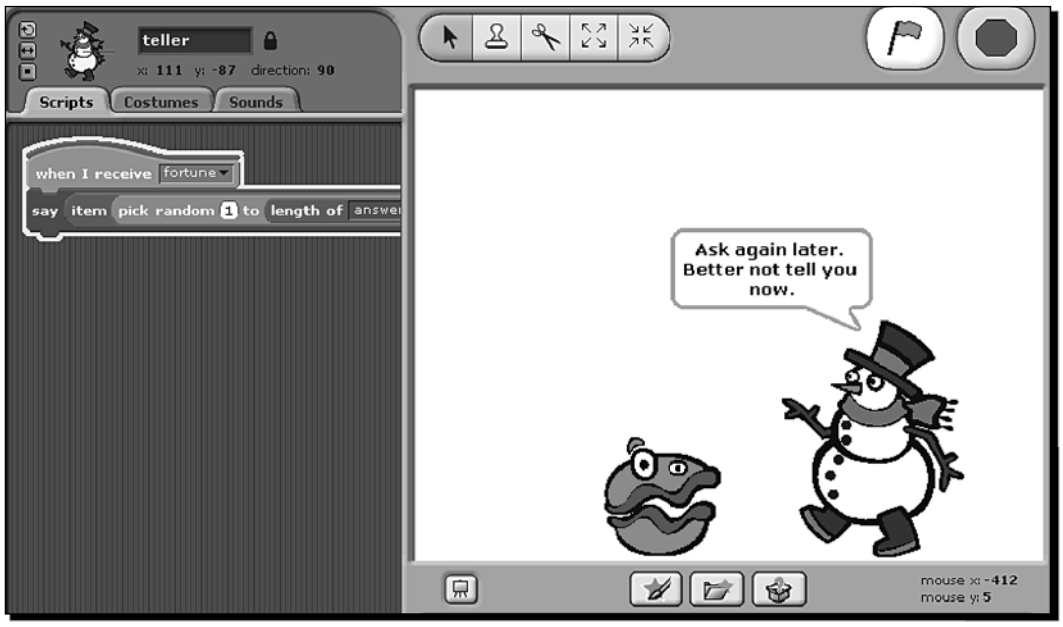
9. For the **seeker** sprite, we pick a random question using the **any** position of the **item** block. This time, we're going to take a slightly different approach to build a more flexible, but slightly more complicated script. From the **Numbers** palette, add the **pick random** block as the position value of the **item** block.

10. From the **Variables** palette, add the **length of** block to the second value of the **pick random** block. Make sure the answers list is selected for the value in the **length of** block. The following screenshot shows the script for the **teller** sprite:



11. Press the Space bar to make the seeker ask a question. The teller will respond with a fortune.

Every item in a list is represented by a number, and when we want to manipulate a certain value in the list, it's the number we refer to. The **list length** block gives us a way to always know how many items are in a list. In our example, our list of answers contains 20 items.



What just happened?

You are now prepared for a life in the carnival. Based on our script, we can answer any question that may come our way, and our response will seem profound. At least that's what we'll choose to believe.

Our seeker asked a question, and our teller doled out a random fortune using items from their respective lists. The script we used for the **teller** sprite was decidedly more complex than the script for the **seeker** sprite.

In the seeker's script, we used the default **any** value to select a random item from the list. For the teller, we replaced the **any** value with a **pick random** block. The **length of questions** block gave the **pick random** block the maximum number of items to choose from. In my example, the script selected a number between 1 and 20 because my questions list contained 20 items.

You're probably asking yourself why we would write the more complex code when we can achieve the same result with less. If you are asking that question, good for you.

The answer, naturally, depends on what we want to accomplish. If we only want to select a random item from our list, then the simple script for the seeker suffices. However, let's say we want to instruct our teller to answer every fifth question with a positive response, and all other questions get a random response.

Let's see how we can modify the teller's script to guarantee a positive fortune for every fifth question.

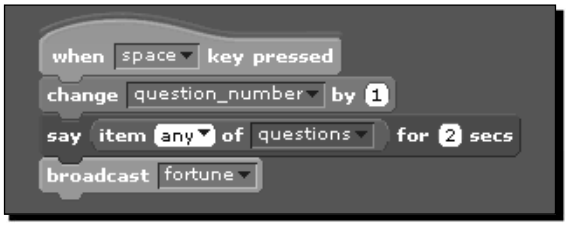
Time for action – force a positive fortune

Before we begin this exercise, re-order your answers list so that all the positive responses are at the end. Edit the list items in your text editor and then re-import the list. Make a note of where the first positive response begins:

1. Our first task is to set up a variable to count how many questions the seeker asks so that we can calculate whether or not it's time to answer positively. Select the **stage** from the sprites list and create a new variable named **question_number.**
2. Add the **when flag clicked** block to the scripts area.
3. From the **Variables** palette, add the **set question_number to 0** block to the **when flag clicked** block. Now, we have a way to reset our calculation.

4. Next, we need to assign the **question_number** variable a value when the seeker asks a question. Click on the **seeker** sprite to display the scripts area.

5. From the **Variables** palette, snap the **change question_number by 1** block in place between the **when space key pressed** and the **say** blocks.



6. Next, make the **teller** sprite give a specific range of answers based on the question. Select the **teller** sprite to display the scripts area.

7. From the **Control** palette, add the **if/else** block to the **when I receive** block. Snap the current **say** block in place after the **if** block.

8. Change the first input value on the **pick random** block to reflect the item number that begins your positive responses. In my example, that value is 11.

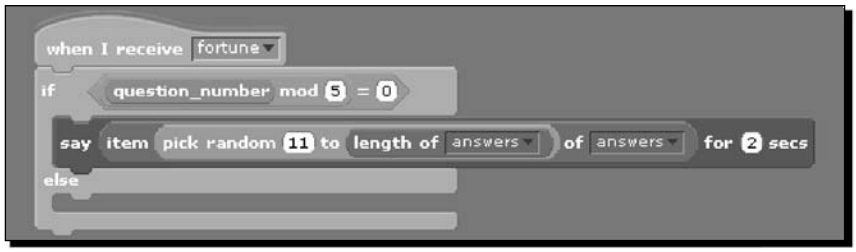
9. Now, we need to supply a condition to the **if** statement to test whether or not we should issue a positive response. From the **Numbers** palette, add the **=** block to the **if** block.

10. Drag the **mod** block into the value to the left of the **=** sign. Change the value to the right of the equals sign to **0**.

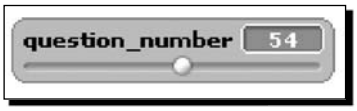
11. We're going to use the **mod** block to divide **question_number** by 5 so that we can calculate the remainder. Add the **question_number** block to the first value of the **mod** block.

12. Change the second value of the **mod** block to **5** so that the block reads **question_number mod 5**.

13. Test your script by pressing the Space bar. With our current setup, the teller responds only on the fifth question, and it's always a positive response.



If you double-click on the variable monitor block on the stage, a slider will display. Use the slider to assign a number to the variable before you run the script as a way to test.



What just happened?

We asked our teller to issue only a positive response every fifth question, but we needed a way to let our **teller** sprite determine when the fifth question was asked. We set up the **question_number** variable as a way to count the question, and the seeker script updated the value of the **question_number** variable each time we pressed the space key. We call that a counter variable.

The **mod** block gave us the logic we needed to let the teller calculate whether or not to issue a positive response. The **mod** block divided the first number (**question_number**) by the second (**5**) and returned the remainder.

The **teller** sprite used the **if** block to compare the remainder to zero. We chose zero because when question_number is a multiple of 5, the remainder was zero. When the remainder was zero, we executed the code in the **if** block; otherwise, the code in the **else** block ran.

Let's evaluate some mod calculations using a divisor of 5:

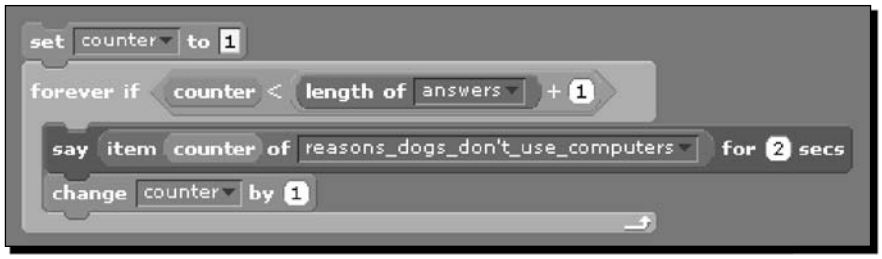
- 25 mod 5 is 0
- 32 mod 5 is 2
- 67 mod 5 is 3

In our script, question 25 guarantees a positive response, while questions 32 and 67 do not.

Counters

When we need to know how many items we've processed like we did in our "force a positive fortune" exercise, we use a counter variable. A counter variable is just an arbitrary name I chose so that we can easily associate that we are using a variable to count the steps in some process.

For example, iterating through each item in a list is a common example of using a variable to count the current list item's position. Consider the block of code in the following screenshot:



Imagine if we had used a list to store our jokes in Chapter 5. Our scripts would have been much simpler to construct. The sample code in the screenshot uses the counter variable in several ways. It sets the value to 1 prior to checking the condition in the **forever if** block. It uses the number assigned to counter to determine if we've processed all the items in the list. If counter is less than the number of items in the list, the block runs. At the end of the block, we increment counter's value by 1, and the **forever if** block checks the new value of counter.

Keep track of intervals with mod

If we identify an interval, then we can create a pattern of events based on the interval. We already saw an example where we look for the fifth occurrence of an event, but what if we wanted to make our sprite dance after 100 seconds elapse? A mod calculation helps us identify the interval. Assuming our timer starts at zero, the expression "current_time mod 100 is 0" becomes a check to identify every 100th second.

In our project example, we used the **mod** block to select certain items from our list, but we could program any number of events based on our interval, such as select items from a totally different list, change backgrounds or costumes, or we could use the mod calculation to do nothing at all.

Have a go hero

Give mod a try. Make the **seeker** sprite do something on every fifth response. Examples of things you might try include issuing a response to the teller, applying a graphical transformation, or jumping for joy.

If/else

In earlier chapters, we became familiar with **forever**, **forever if**, and **if** concepts. Each of these concepts checks a condition and then runs if the condition is met. We don't define what happens when the condition is not met.

In contrast, the **if/else** control block evaluates a condition, and if the evaluation is true, the code in the **if** block executes. If the **if** condition evaluates to false, then the code in the **else** block executes.

Think of the ultimatums you give your children, or your parents gave you. If you clean your room, you get ice cream. Or else, you go to bed without a snack.

Pop quiz

1. The **mod** block:
   - Modifies a number in the list
   - Creates a variable that tracks an interval
   - Transforms the sprite into a leprechaun
   - Divides two numbers and returns the remainder
2. We use a counter variable to:
   - Track how many times an event occurs
   - Identify how many sprites we have in the project
   - Select a random item from a list
   - Add a new item to a specific position in the list

Repeat the fortune

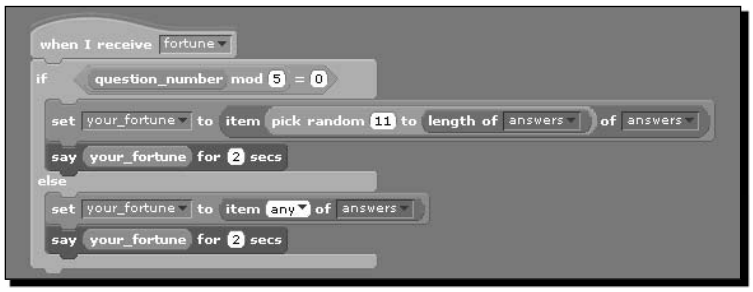
Up to this point, we've only stored numeric values in our variables, but variables can store text too. Let's add a script to our **teller** sprite to repeat the question.

Time for action – my fortune is what?

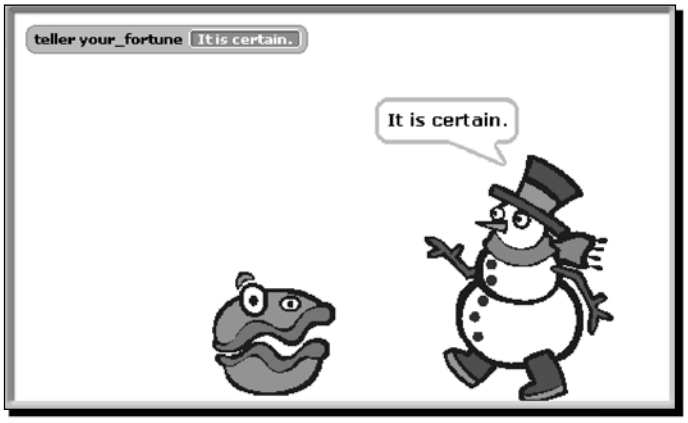
What if we get distracted and miss our fortune? That would be tragic, so let's add some code to the **teller** sprite that repeats the last fortune told:

1. Create a private variable for the **teller** sprite named **your_fortune**. We'll use this variable to keep track of the fortune.
2. We'll start with the blocks inside the **if** statement. From the **Variables** palette, snap the **set to** block in place between the **if** block and the **say** block. Select **your_fortune** as the variable so that the block reads **set your_fortune to 0**.
3. We need to replace the **0** value in the **set to** block with the code that selects a random item from the answers list. Remember, we're working with the **if** block right now. Click on the **item** block and drag the entire block into the **to** value of the **set to** block.
4. From the **Variables** palette, drag the **your_fortune** block into the value of the **say** block. Refer to the following screenshot for the new script:

5. Apply the same changes to the **else** block. Refer to the following screenshot for help:



6. Get a fortune by pressing the Space bar. Notice that the your_fortune monitor reports the teller's fortune.

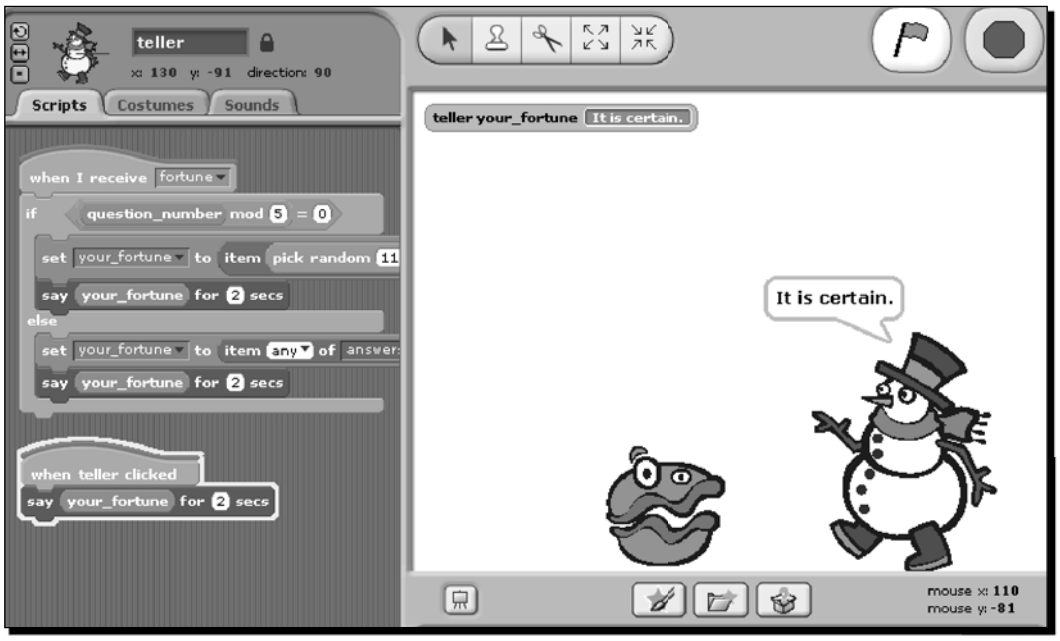


7. Now, we need a way to prompt the teller to repeat the message. We'll add a **when clicked** control block to the **teller** sprite. From the **Control** palette, add the **when teller clicked** block to the scripts area.

8. From the Looks palette, add the **say for 2 secs** block to the **when teller clicked** block.

9. From the Variables palette, add the **your_fortune** block as the message value for the **say** block.

10. Click the **teller** sprite to repeat the last fortune.



What just happened?

Like thoughts, fortunes can be fleeting. That's why we captured our fortune in the **your_fortune** variable. We replaced where in our script we selected the fortune. Our logic to pick a random item from the answers list became the input to the **set to** block. And for the **say** block, we replaced the message value with the **your_fortune** variable.

Now, if we get a fortune we like, we can make the teller repeat it over and over.

Holding text in a variable

In earlier chapters, we typed text directly into the **say** block. This means that if we want to change what the sprite says, we have to find the **say** block in the script and change the message.

When we work with dynamic data, we sometimes want to capture that data so that we can use it later. We already learned how we can use variables to store numeric data, but having the ability to store text adds a whole new level of functionality to our projects, such as simulating dynamic conversations.

Other programming languages often refer to a text phrase as a **string**.

Variables are one of the most important programming concepts we can use. A majority of the projects you create will need a variable, especially as the projects become more interactive and complex.

Text entry limitations

Now, it is a logical time for us to build a way that prompts the user to type a question for the teller. However, the current version of Scratch (1.3) doesn't allow us to prompt the user for text input that we can use in our scripts.

Next steps

Our focus has been on using lists and variables to drive the action in our game. If you wanted to expand on this game, try adding graphical effects to the sprites and creating a background to begin to tell a story.

Using the concepts in this chapter, you could create other games of chance, such as blackjack, bingo, or a lottery number predictor.

Summary

We have seen the power that lists and variables give us to create dynamic, flexible, and fun projects. As you worked through this chapter, you probably realized that we could have used lists in some of our previous projects. For example, our scripts in Chapter 5 would have been much smaller had we used lists.

We learned how to manipulate lists, a special kind of variable, to collect a group of related items. We continued to explore variables by using them to store text. The **mod** block helped us identify intervals as we iterated through our list. We used a variable to keep track of the interval so that we could program specific events based on when an interval occurred.

Our programming knowledge has been accumulating nicely to this point even though you may not know it. We're ready to apply everything we've learned, plus more, in our next project. Get your money.

Where to buy this book

You can buy Scratch 1.3 from the Packt Publishing website:
`http://www.packtpub.com/scratch-1-3-beginners-guide/book`

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com