Introducing the Trivia Challenge Game

The Trivia Challenge game tests a player's knowledge with a series of multiple-choice questions. The game delivers the questions as a single "episode." The episode I created to show off the program is about the mafia and is called "An Episode You Can't Refuse." All of the questions relate in some way to the mafia (although a bit indirectly at times).

The cool thing about the game is that the questions for an episode are stored in a separate file, independent of the game code. This way, it's easy to play different ones. Even better, this means that anyone with a text editor (like Notepad on Windows machines) can create their own trivia episode about whatever topic they choose—anything from anime to zoology. Figure 7.1 shows the game (and my episode) in action.

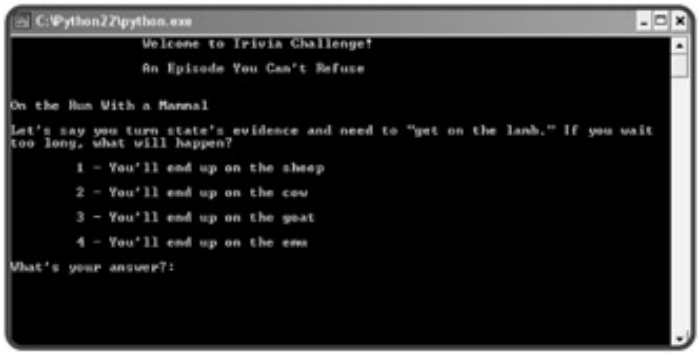

Figure 7.1: The player is always presented with four inviting choices. But only one is correct.

Back to the Trivia Challenge Game

With the basics of files and exceptions under your belt, it's time to tackle the Trivia Challenge game presented at the the chapter. One of the cool things about the program is that it reads a plain text file, so you can create your own trivi episodes with a text editor and a dash of creativity. As you'll see in the code, the text file the program reads,  `triv` needs to be in the same directory as the program file. To create your own episode full of questions, all you need to d this file with one containing your own work.

Understanding the Data File Layout

Before I go over actual code from the game, you should understand exactly how the  `trivia.txt` file is structur first line in the file is the title of the episode. The rest of the file consists of blocks of seven lines for each question. Yo as many blocks (and thus questions) as you like. Here's a generic representation of a block:

```
<category>
<question>
<answer 1>
<answer 2>
<answer 3>
<answer 4>
<correct answer>
<explanation>
```

And here's the beginning of the file I created for the game:

```
An Episode You Can't Refuse
On the Run With a Mammal
Let's say you turn state's evidence and need to "get on the lamb." If you wait /too l
will happen?
You'll end up on the sheep
You'll end up on the cow
You'll end up on the goat
You'll end up on the emu
1
A lamb is just a young sheep.
The Godfather Will Get Down With You Now
Let's say you have an audience with the Godfather of Soul. How would it be /smart to a
him?
Mr. Richard
Mr. Domino
Mr. Brown
Mr. Checker
3
James Brown is the Godfather of Soul.
```

To save space, I only show the first 15 lines of the file—two questions' worth. You can take a look at the complete file `trivia.txt`, on the CD-ROM that's included with this book.

Remember, the very first line in the file, `An Episode You Can't Refuse`, is the episode title for this game. The lines are for the first question. And the next seven lines are for the second question. So, the line `On the Run With` is the category of the first question. The category is just a clever way to introduce the next question. The next line, `Le you turn state's evidence and need to "get on the lamb." If you wait /too long, what happen?`, is the first question in the game. The next four lines, `You'll end up on the sheep`, `You'll end cow`, `You'll end up on the goat`, and `You'll end up on the emu`, are the four possible answers from player will choose. The next line, `1`, is the number of the correct answer. So in this case, the correct answer to the q first answer, `You'll end up on the sheep`. The next line, `A lamb is just a young sheep.`, explains wh answer is correct. The rest of the questions follow the same pattern.

An important thing to note is that I included a forward slash (`/`) in two of the lines. I did this to represent a newline si does not automatically wrap text when it prints it. When the program reads a line from the text file, it replaces all of th slashes with the newline character. You'll see exactly how the program does this when I go over the code.

The `open_file()` Function

The first thing I do in the program is define the function `open_file()`, which receives a file name and mode (both s returns a corresponding file object. I use `try` and `except` to trap for an `IOError` exception for input-output errors, v occur if the file doesn't exist, for example.

If I trap an exception, that means there was a problem opening the trivia file. If this happens, there's no point in conti program, so I print an appropriate message and call the `sys.exit()` function. This function raises an exception tha the termination of the program. You should only use `sys.exit()` as a last resort, when you must end a program. N didn't have to import the `sys` module to call `sys.exit()`. That's because the `sys` module is always available.

```
# Trivia Challenge
# Trivia game that reads a plain text file
# Michael Dawson - 5/3/03

def open_file(file_name, mode):
    """Open a file."""
    try:
        the_file = open(file_name, mode)
    except(IOError), e:
        print "Unable to open the file", file_name, "Ending program.\n", e
        raw_input("\n\nPress the enter key to exit.")
        sys.exit()
    else:
        return the_file
```

The `next_line()` Function

Next, I define the `next_line()` function, which receives a file object and returns the next line of text from it:

```
def next_line(the_file):
    """Return next line from the trivia file, formatted."""
    line = the_file.readline()
```

```
    line = line.replace("/", "\n")
    return line
```

However, I do one small bit of formatting to the line before I return it. I replace all forward slashes with newline charac this because Python does not automatically word wrap printed text. My procedure gives the creator of a trivia text file formatting control. He or she can indicate where newlines should go so that words don't get split across lines. Take a `triva.txt` file and the output of the Trivia Challenge game to see this in action. Try removing the forward slashes f file and check out the results.

The `next_block()` Function

The `next_block()` function reads the next block of lines for one question. It takes a file object and returns four strin of strings. It returns a string for the category, question, correct answer, and explanation. It returns a list of four strings possible answers to the question.

```
def next_block(the_file):
    """Return the next block of data from the trivia file."""
    category = next_line(the_file)

    question = next_line(the_file)

    answers = []
    for i in range(4):
        answers.append(next_line(the_file))

    correct = next_line(the_file)
    if correct:
        correct = correct[0]

    explanation = next_line(the_file)

    return category, question, answers, correct, explanation
```

If the end of the file is reached, reading a line returns the empty string. So, when the program comes to the end of `trivia.txt` , `category` gets the empty string. I check category in the `main()` function of the program. When it empty string, the game is over.

The `welcome()` Function

The `welcome()` function welcomes the player to the game and announces the episode's title. The function gets the as a string and prints it along with a welcome message.

```
def welcome(title):
    """Welcome the player and get his/her name."""
    print "\t\tWelcome to Trivia Challenge!\n"
    print "\t\t", title, "\n"
```

Setting Up the Game

Next, I create the `main()` function, which houses the main game loop. In the first part of the function, I set up the ga opening the trivia file, getting the title of the episode (the first line of the file), welcoming the player, and setting the pl to `0`.

```
def main():
    trivia_file = open_file("trivia.txt", "r")
    title = next_line(trivia_file)
    welcome(title)
    score = 0
```

Asking a Question

Next, I read the first block of lines for the first question into variables. Then, I start the `while` loop, which will continu questions as long as `category` is not the empty string. If `category` is the empty string, that means the end of the t been reached and the loop won't be entered. I ask a question by printing the category of the question, the question it four possible answers.

```
    # get first block
    category, question, answers, correct, explanation = next_block(trivia_file)
    while category:
        # ask a question
        print category
        print question
        for i in range(4):
            print "\t", i + 1, "-", answers[i]
```

Getting an Answer

Next, I get the player's answer:

```
        # get answer
        answer = raw_input("What's your answer?: ")
```

Checking an Answer

Then, I compare the player's answer to the correct answer. If they match, the player is congratulated and his or her s increased by one. If they don't match, the player is told he or she is wrong. In either case, I then display the explanati describes why the correct answer is correct. Lastly, I display the player's current score.

```
        # check answer
        if answer == correct:
            print "\nRight!",
            score += 1
        else:
           print "\nWrong.",
        print explanation
```

```
        print "Score:", score, "\n\n"
```

Getting the Next Question

Then, I call the `next_block()` function and get the block of strings for the next question. If there are no more quest `category` will get the empty string and the loop won't continue.

```
        # get next block
        category, question, answers, correct, explanation = next_block(trivia_file)
```

Ending the Game

After the loop, I close the trivia file and display the player's score:

```
    trivia_file.close()

    print "That was the last question!"
    print "You're final score is:", score
```

Starting `the main()` Function

The last lines of code start `main()` and kick off the game:

```
main()
raw_input("\n\nPress the enter key to exit.")
```

Summary

In this chapter, you learned about files and exceptions. You learned how to read from text files. You saw how to read a single character or an entire file at once. You learned several different ways to read one full line at a time, probably the most common way to read a text file. You also learned how to write to text files—everything from a single character to a list of strings. Next, you learned how to save more complex data to files through pickling and how to manage a group of pickled objects in a single file using a shelf. Then, you saw how to handle exceptions raised during the execution of a program. You saw how to trap for specific exceptions and how to write code to work around them. Finally, you saw how to put files and exceptions together through the construction of a trivia game program that allows anyone with a text editor to create their very own trivia episodes.

Challenges

1. Improve the Trivia Challenge game so that each question has a unique point value associated with it. The player's score should be the total of all the point values of the questions he or she answers correctly.

2. Improve the Trivia Challenge game so that it maintains a high-scores list in a file. The program should record the player's name and score if the player makes the list. Store the high scores using a pickled object.

3. Change the way the high-scores functionality you created in the last challenge is implemented. This time, use a plain text file to store the list.

4. Create a trivia game episode that tests a player's knowledge of Python files and exceptions.