This course has already ended.
The latest instance of the course can be found at: O1: 2023

« Chapter 9.2: Comparing, Sorting, and Grouping Course materials Week 10 »
CS-A1110 / Week 9 / Chapter 9.3: Peeveli

Luet oppimateriaalin englanninkielistä versiota. Mainitsit kuitenkin taustakyselyssä osaavasi suomea. Siksi **suosittelemme, että käytät suomenkielistä versiota**, joka on testatumpi ja hieman laajempi ja muutenkin mukava.
Suomenkielinen materiaali kyllä esittelee englanninkielisetkin termit. Myös suomenkielisessä materiaalissa käytetään ohjelmien koodissa englanninkielisiä nimiä kurssin alkupään johdantoesimerkeissä kukuunottamatta.
Voit vaihtaa kieltä A+:n valikon yläreunassa olevasta painikkeesta. Tai tästä: **Vaihda suomeksi**.

Chapter 9.3: Peeveli

About This Page

Questions Answered: What higher-order methods do `String`s have? Can I write an algorithm that operates on strings? My friend claims to be good at Hangman; how can I teach them a lesson?

Topics: Additional practice on implementing algorithms, strings, `Map`s, and higher-order methods.

What Will I Do? Program, once you work out what the program is supposed to do. A single programming assignment makes up most of this chapter.

Rough Estimate of Workload? Five hours? You don't need to write a great amount of code, but it will take time to get to know the program's domain, come up with a solution, translate the solution into code, and test the program. Don't get stuck; ask for help.

Points Available: C90.

Related Modules: Peeveli **(new)**.

Background for Upcoming Assignments

A recap

The facts listed below should be already familiar. Revisit the earlier chapters or the Scala Reference for details as needed.

- Collections such as buffers and vectors have many first-order methods (such as `take`, `contains`, and `indexOf`; Chapter 4.2) and higher-order methods (such as `filter`, `map`, and `maxBy`; Chapters 6.3, 6.4, and 9.2).
- `String`s consist of elements of type `Char`, each of which represents a single letter or other character (Chapter 5.2).
- `String`s have many methods that are specific to string processing (such as `trim` and `toUpperCase`; Chapter 5.2).
- `String`s are objects; they are collections with characters as elements (Chapter 5.2). The things you can do with a collection, such as loop over it with `for`, you can also do with a `String` (Chapter 5.6).
- `String`s share many first-order methods with other collections (such as `take`, `contains`, and `indexOf`; Chapter 5.2).

Now, knowing that `String`s are collections, it's natural that they too have *higher-order* methods, many of which you already know.

Examples of higher-order methods on `String`s

Let's use `foreach` to print each of a string's elements (characters):

```
"H!!".foreach(println)H
!
!
```

Now let's take a longer string and `filter` out everything but lower-case letters. We'll be assisted by the `isLower` method on `Char`s:

```
"let's offroad!".filter( _.isLower )res0: String = etsoffroad
```

`sorted` sorts characters as per their natural ordering (Chapter 9.2), which is defined by Unicode:

```
"let's offroad!".sortedres1: String = " 'adeffoorst"
```

Note that in Unicode, the upper-case alphabet comes before the lower-case one.
With `sortBy`, we can — for instance — sort the letters by their lower-case equivalents:

```
"let's offroad!".sortBy( _.toLower )res2: String = " 'adeffLoorst"
```

Calling `toLower` on a `Char` produces its lower-case version.
Observe the difference between this output and the earlier one.
The `map` method constructs its return value by applying its parameter function to each character in the string. Here are a couple of examples:

```
"let's offroad!".map( char => if (char.isLower) char.toUpper else char.toLower )res3: String = lET'S OFFROAD!
"H!!".map( _.isLower )res4: IndexedSeq[Boolean] = Vector(false, true, false)
```

Peeveli, a Word Game

Background: the game of Hangman


A, L and N have been correctly guessed. Five guesses have missed the mark, so the riddler has drawn a stick figure with 1) a head, 2) a body, 3, 4) two arms, and 5) a leg. The next incorrect guess means the second leg is drawn and the guesser loses.

Hangman is a classic word game for two: one player — the riddler — picks a word and the other player — the guesser — tries to figure out what the word is, guessing one letter at a time. The length of the word is known to both players. Each time the guesser picks a letter, the riddler must reveal all those positions in the target word where that letter appears. If the letter doesn't appear in the target word at all, the guess is a miss and the riddler draws an additional element into a line drawing of a hanging stick figure.

If the guesser misses a few times, the drawing will be completed and the guesser loses the game. The guesser wins in case they manage to reveal the entire word.

That's pretty much all there is to it, but you can read up on the game in Wikipedia if you like.

There are many digital variants of Hangman in which the computer takes the role of the riddler and the human player is the guesser. Just like the traditional pen-and-paper version, these variants are based on the premise that the guesser can trust the riddler.

Our take on Hangman is different. In the game we're about to program, the riddler is something called Peeveli, and it doesn't play fair.

Cheating at Hangman

In a regular game of Hangman, the riddler picks the target word at the start of the game and scrupulously reveals letters as the correct answers accumulate. But what if the riddler wasn't trustworthy?

Imagine the scene. The game has just started, you are the guesser, and the riddler has just — unbeknownst to you — picked the target word BAZAAR. The concealed word initially looks like this:

```
_ _ _ _ _ _
```

If you now guess A, the riddler should reveal three copies of that letter. But they could instead mentally switch their original target word for an A-less one — BELIEF is an example — and inform you that there is no A and your guess is a miss.

Imagine another scene. You've managed to guess all but one of the letters in this five-letter word, but the next incorrect guess with lose you the game.

```
_ A S I S
```

Suppose you haven't guessed the letters B and O yet. There are only two possible answers: BASIS and OASIS. However, if you pick B, the riddler can claim they were thinking of OASIS — and vice versa. You can't beat the dishonest riddler!

Introduction to Peeveli

The name Peeveli

"Peeveli" is an archaic Finnish word for the Devil. It derives from the Old Swedish "bövel", which additionally meant a hangman. These days, the Finnish word is comparatively rare and is employed almost exclusively as a semi-humorous exclamation or very mild curse. The name of our game thus captures the diabolical nature of our virtual gallows expert as well as, perhaps, the reaction of the exasperated guesser.

In the game of **Peeveli**, the computer is the riddler and the human player is the guesser. The game is challenging because the computer cheats methodically. It uses a vast vocabulary in combination with a devious algorithm: it doesn't actually pick a target word at all, instead keeping track of all the existing words that continue to be plausible correct solutions, given the previous guesses and the letters already revealed as a consequence. Whenever the player guesses a letter, Peeveli tries to keep its options open: it selects which (if any) letters to reveal so that there are as many potential solutions left as possible.

Imagine a scenario with you as the riddler. The target word is supposed to have four letters. Further imagine that there are no four-letter words in English apart from the ten listed below. The target word must therefore be one of those ten.

```
ALSO AREA AUNT GURU HAWK IDEA JUJU PLAY TUNA ZERO
```

Your opponent's first guess is the letter A. What should you do?

First, you should see where the A's appear in the list of known four-letter words.

```
ALSO AREA AUNT GURU HAWK IDEA JUJU PLAY TUNA ZERO
```

Now you can identify a few groups of words with different patterns of A's in them:

Group	Description	Words
`A_A_`	Begins and ends with an A	AREA
`A___`	One A as the first letter.	ALSO, AUNT
`_A__`	One A as the second letter	HAWK
`__A_`	One A as the third letter.	PLAY
`___A`	One A as the last letter.	IDEA, TUNA
`____`	No A's at all.	GURU, JUJU, ZERO

This basically means that you need to pick one of six alternatives. A good basic strategy is to pick the largest group, which in this instance is the last one. So you inform your opponent that the word has no A's and memorize the fact that there are three plausible solutions left: GURU, JUJU, and ZERO.

If the vocabulary had additionally contained APEX and ARMY, which begin with an A, the group `A___` would have been the largest. In that scenario, you would have revealed the initial A and memorized the plausible solutions ALSO, APEX, ARMY, and AUNT.

The art of deception

In this chapter, we'll always use the basic strategy of picking the largest group. If you wish, you can reflect on why that isn't always optimal and how you might come up with even craftier algorithms.

Suppose you've chosen the group with GURU, JUJU, and ZERO, and your opponent's next guess is the letter U. This results in the groups `_U_U` (with GURU and JUJU) and `____` (with ZERO). The former is larger, so you tell your opponent there are U's in the second and fourth slots.

Whenever your opponent guesses a letter that doesn't appear in any of the plausible solutions, you have only a single group that encompasses all the words that remain. Obviously, you'll then pick that group.

Occasionally, you'll find that there are multiple groups of equal size. In such cases, you could pick an arbitrary group or break the tie by choosing the group that reveals the fewest letters.

The Peeveli module

The Peeveli game doesn't draw a hanging figure; it tallies
missed guesses by displaying letters in red. The letters start
appearing when the player has only a small number of incorrect
guesses left.

The module Peeveli contains a partial implementation of the game described above. The GUI is in working order but the riddler is sorely lacking in smarts.

Task description

1. Try playing the game as given. The app object is `o1.peeveli.gui.PeeveliApp`. You'll notice that:
   - The riddler is really lenient, not at all what we intended. It accepts all guesses and always reveals one more letter of the target word on each guess.
   - The game doesn't end when it should.
   - You can change vocabularies in the menu.
   - You can enable a testing mode where Peeveli uses the text console to print out all the remaining plausible solutions after each guess. (The given riddler meekly prunes down the list to just one solution, though.)
2. Study the module's Scaladocs and program code.
3. Implement the missing parts that make Peeveli work as cleverly as we planned above.

A clarification

In the above description of Peeveli's algorithm, we mentioned that when word groups are equal in size, it might be a good idea to pick the group that reveals fewer letters to the guesser. **That is completely optional in this assignment.** Implement that additional bit of deviilry only if you want an additional challenge. Otherwise, just pick any of the equally sized groups.

Instructions and hints

- You'll need to edit only `GameState.scala`.
- It's a good idea to implement the private method `reveal` and use it in `guessLetter`.
- When you write literals, remember that whereas `String` literals go in double quotation marks as in `"myString"`, `Char` literals such as `'c'` use single quotes.
- `groupBy` (Chapter 9.2)

Points 0 / 90 My submissions 5 / 10 Deadline Wednesday, 11 November 2020, 12:00
To be submitted alone or in groups of 2

This course has been archived (Tuesday, 31 August 2021, 23:59).

Assignment 6 (Peeveli)

Select your files for grading

GameState.scala

Choose File No file chosen

Submit

Feedback

Please note that this section must be completed individually. Even if you worked on this chapter with a pair, each of you should submit the form separately.

Accepted My submissions 1

This course has been archived (Tuesday, 31 August 2021, 23:59).

Time spent: (*) Required

Please estimate the total number of minutes you spent on this chapter (reading, assignments, etc.). You don't have to be exact, but if you can produce an estimate to within 15 minutes or half an hour, that would be great.

240

Written comment or question:

You aren't required to give written feedback. Nevertheless, please do ask something, give feedback, or reflect on your learning! (However, the right place to ask urgent questions about programs that you're currently working on isn't this form but Piazza or the lab sessions. We can't guarantee that anyone will even see anything you type here before the weekly deadline.)

Submit an update

Credits

Thousands of students have given feedback that has contributed to this ebook's design. Thank you!

The ebook's chapters, programming assignments, and weekly bulletins have been written in Finnish and translated into English by Juha Sorva.

The appendices (glossary, Scala reference, FAQ, etc.) are by Juha Sorva unless otherwise specified on the page.

The automatic assessment of the assignments has been developed by: (in alphabetical order) Riku Autio, Nikolas Drosdek, Joonatan Honkamaa, Jaakko Kantojärvi, Niklas Kröger, Teemu Lehtinen, Strasdosky Otewa, Timi Seppälä, Teemu Sirkiä, and Aleksi Vartiainen.

The illustrations at the top of each chapter, and the similar drawings elsewhere in the ebook, are the work of Christina Lassheikki.

The animations that detail the execution Scala programs have been designed by Juha Sorva and Teemu Sirkiä. Teemu Sirkiä and Riku Autio did the technical implementation, relying on Teemu's Jsvee and Kelmu toolkits.

The other diagrams and interactive presentations in the ebook are by Juha Sorva.

The O1Library software has been developed by Aleksi Lukkarinen and Juha Sorva. Several of its key components are built upon Aleksi's SMCL library.

The pedagogy of using O1Library for simple graphical programmers (such as `Plc`) is inspired by the textbooks *How to Design Programs* by Flatt, Felleisen, Findler, and Krishnamurthi and *Picturing Programs* by Stephen Bloch.

The course platform A+ was originally created at Aalto's LeTech research group as a student project. The open-source project is now shepherded by the Computer Science department's edu-tech team and hosted by the department's IT services. Markku Riekkinen is the current lead developer; dozens of Aalto students and others have also contributed.

The A+ Courses plugin, which supports A+ and O1 in IntelliJ IDEA, is another open-source project. It was created by Nikolai Denissov, Olli Kiljunen, and Nikolas Drosdek with input from Juha Sorva, Otto Seppälä, Arto Hellas, and others.

For O1's current teaching staff, please see Chapter 1.1.

Additional credits for this page

Peeveli is a derivative of a similar programming assignment designed by Keith Schwarz.


a drop of ink