Program Design and Data Structures, 20.0 c

Course code: 1DL201, Report code: DL201, 67%, DAG, NML,
week: 44 - 02 Semester: Autumn 2017
week: 03 - 11 Semester: Spring 2018

LAB ASSIGNMENT 2

This information is not available in English. Now showing the Swedish version.

After this lab session you should be able to...

- Write simple functions.
- Explain how Haskell programs are executed.

Before the lab session

- Read the slides on functions (declarations, evaluation, etc.) from lecture 4.
- Read *the entire* assignment first.

Instructions

1. Start the Haskell interpreter with the command ghci in the Unix shell (or use the Emacs interface).
2. Use Emacs or another text editor to create a file where you can save the functions that you write!
3. Take notes while you work, to make it easier to show what you have done, and to answer questions from the assistant who grades your lab assignment.

Tip

To create a Haskell module containing functions, create a text file and give it a name with the ending ".hs" (example: *lab2.hs*).

You can now fill this file with functions and definitions. To load the file in ghci use the **:load** command: **:load lab2**

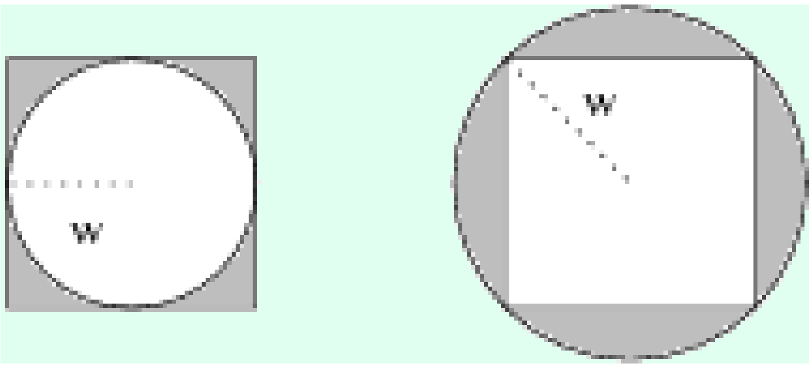
Note that (global) declarations made in a file should *not* begin with **let**: e.g., write add1 x = x+1 instead of let add1 x = x+1

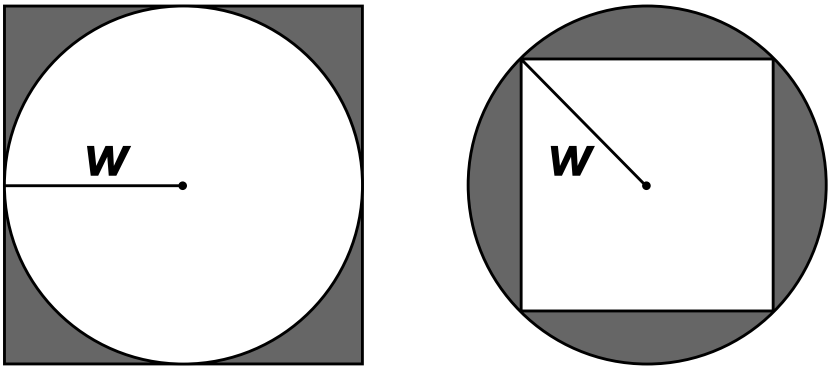
The Task

PROBLEM 1

You will write functions for computing the area of the dark gray regions of the figures. In order to compute the area of the left figure, you must compute the area of the square, and then subtract the area of the circle.

In order to compute the area on the right, you must compute the area of the circle, and then subtract the area of the square.





In order to do this, the following hints should be useful:

- pi :: Double is bound to (a floating-point approximation of) the number □.
- sqrt :: Double -> Double is a function for computing square roots.
- (^) :: Double -> Integer -> Double is a function to raise a number to some power. (There are other power functions in Haskell. In this lab, we'll only consider (^).)
- The area of a rectangle is the height times the width.
- The area of a circle is multiplied by the square of the radius.
- The Pythagorean theorem: in a right triangle, the square of the hypotenuse is equal to the sum of the squares of the other two sides (the catheti).

Write the following functions:

- squareArea :: Double -> Double which given the length of a side of a square computes its area.
- circleArea :: Double -> Double which given the radius of a circle computes its area.
- squareCircleArea :: Double -> Double which given a value corresponding to *w* in the left figure computes the area of the shaded region.
- cathetus :: Double -> Double which given the length of a hypotenuse (and the assumption that the catheti are of equal length) computes the length of a cathetus.
- circleSquareArea :: Double -> Double which given a value corresponding to *w* in the right figure computes the area of the shaded region (four circle segments).

Some examples of what we want the functions to compute:

```
> squareArea 2.0
4.0
> circleArea 2.0
12.566370614359172
> squareCircleArea 2.0
3.4336293856408275
> cathetus 2.0
1.414213562373095
> circleSquareArea 2.0
4.566370614359174
```

PROBLEM 2

Write a function rhymes :: String -> String -> Bool that takes two words as arguments. The function should return True if the words end on the same three letters. (Yes, this is a very simple definition of rhyming.) If any word is shorter than three letters, the function should return True if the words are equal. Otherwise the function should return False.

Hint 1: Use conditional expressions (if-then-else) and Boolean operators (||, &&, not) as needed to handle the different cases.
Hint 2: You may use functions that we have defined in the lectures, if you find them useful.

Some examples of what we want the function to compute:

```
> rhymes "steka" "leka"
True
> rhymes "elfel" "nackdel"
False
> rhymes "del" "el"
False
> rhymes "el" "el"
True
```

PROBLEM 3

Try to solve this problem without running the code, as an exercise in reading Haskell code. (If you don't at all succeed, you may run the code in GHCi to figure out what is going on.)

Consider the following function declaration:

```
drJeep :: String -> String -> Bool
drJeep x y =
  not (length x < length y) && drop (length x - length y) x == y
```

1. Show, *step by step*, how the interpreter computes the expression drJeep "Jultomte" "tomte". Present it in the style of the evaluation from the lecture slides.
2. Describe, in words, *how* the function does what it is supposed to do - in other words, how the function operates on a stepwise basis (not just for the particular example

where x="Jultomte" and y="tomte", but in general).

How would you describe *what* the function computes - seen from "the outside"? What would be meaningful names for the function and its arguments - names that describe what the function does and what the role of each argument is?

WHEN YOU ARE DONE

When you are done with all problems (the explorations below are optional), raise your hand or approach an assistant to have your solution graded.

If you pass this lab at least 30 minutes early and other groups are still working on it, we ask you to **help one other group**. Do not simply share your solution with them, but try to understand the (partial) solutions that they have developed so far (which may be different from yours) and the difficulties that they have. Assist them in coming up with their own solutions. Once the group that you are helping completes the lab assignment, you may leave. (If it is still early, the group that received your help should then stay to help another group.)

EXPLORATIONS

If you have finished the lab exercises and are waiting to be graded, or if you wish to explore programming in Haskell further at any time, or want to get some extra practice, have a look at the Explorations page.

Exploration problems are optional and should only be attempted after other problems are completed.
You don't have to show these answers to the lab assistants for grading.