Programming I: Functional Programming in Haskell
Solutions to Unassessed Exercises

Set 1: Basics

Note: Ignore the line breaks in the solutions - this is simply to aid readability

1. Notes:

(a)-(e) The operator `-` is left-associative; `^` is right associative. (Note, in (d) you get division by zero.)

(f)-(h) The precedence of `^` is higher than `*` which in turn is higher than `-`. For a full list of precedences see either the Haskell book or the Haskell web pages.

(l) The expression `(ord 'A' - ord 'a')` computes the ordinal difference between upper and lower case letters; this works because `'a'..'z'` and `'A'..'Z'` are numbered consecutively in the Unicode/ASCII coding scheme. Note that we could have chosen any base character for this (e.g. `'K'` and `'k'`). By subtracting this number from the ordinal value of any upper case character, e.g. `'X'`, we obtain the lower case equivalent (i.e. `'x'` in this case). This is a general trick for converting between upper and lower case characters.

(o)-(p) Because of the inaccuracy of floating-point arithmetic a small error is introduced when computing some function applications. Although `sqrt` and `(^2)` are inverses, the mechanics of computer arithmetic means that a composition of the two will not always deliver the original number. The result of (p) is almost zero, but not quite. For this reason we must be careful when reasoning about floating-point arithmetic, especially when very large numbers of calculations are involved (the errors mount up!).

Remark: yes, `(^2)` really is a valid Haskell function. More of this later.

2. (a) `123 `mod` 10`
   (b) `456 `div` 10 `mod` 10`
   (c) `chr (ord 'a' + 7)`

3. (a) `not (2 * 3 == 10)`
   (b) `(3 == 4) == True` (without the parentheses it can't decide which way to bracket the expression)
   (c) `(if True then 1 else 2) == 3` (conditional expressions extend as far to the right as possible)
   (d) `ord (if 3 == 4 then 'a' else 'b') + 1`
   (e) `(8 > 2 ^ if 3 == 4 then 2 else 3) == False`

4. (a) `True` because `2 < 4`, the two 1s being the same
   (b) `True` because `(4, 9) > (1, 10)`, as `4 > 1`
   (c) `True` because `'b' < 'c'`, all other terms being the same

5. `let s = 8473 in (s `div` 3600, s `mod` 3600 `div` 60, s `mod` 60)`

6. `let (r, t) = (1, pi/4) in (r * cos t, r * sin t)`

   The answer is `(0.7071067811865476,0.7071067811865475)`.

7. (a) `(3, 8, 24)` – the x and y are *bound* to 3 and 8 respectively
   (b) Mattern matching error – the pattern `(x, y)` and the value `(1, 2, 3)` cannot be matched as they have different structure
   (c) `1`
   (d) `(5, '*')` because `p` has been defined to be `(6, 5)`; the expression `(p, '*')` is the same as `((6, 5,), '*')` and so can be matched with the pattern `((a, b), c)`
   (e) `(True, (True, 1))` – the `p` is bound to the pair `(True, 1)`

8. `let (q, r) = quotRem 24 10 in (q * 10 + q) * 100 + (r * 10 + r)`

   The answer is 2244. In general, this duplicates the first and second digits of the decimal representation of n.

9. (a) `[ch | ch <- "As you like it", ord ch > 106]`
   (b) ``[x | x <- [13,26..1000], x `mod` 10 == 3]``
   (c) `[(m, n, m * n) | m <- [2..12], n <- [2..12]]`
   (d) `[(c, s) | c <- "JQKA", s <- "CDHS"]`
   (e) `[(m, n) | m <- [1..100], n <- [1..100], m < n, (m + n) == (m - n) ^ 2]`