Save-the-word Haskell

Chris Lin

July 13, 2016

1. The easiest way: Haskell platform
   (a) GHC: the most widely used Haskell compiler.
   How to use:
   - Start (use them both, one by one):
     `ghci`
     `:set prompt "ghci> "`
   - Load up a file (provided **myfunctions.hs**):
     `:l myfunctions`
   - Reload:
     `:r`

2. Basic knowledge
   (a) Always surround a negative number with parentheses.
   (b) inequality symbol:
   `/=`
   (watch out for the difference between `4` and `"4"`)
   (c) Functions are called by writing the function name, a space and then the parameters, separated by spaces. For examples,
   `min 9 10`
   So,
   `bar (bar 3)` means `bar(bar(3))` in C.
   And there is no `bar(bar 3)` in Haskell.
   (d) Function application has the highest precedence, which means these two statements are equivalent:
   `succ 9 + max 5 4 + 1`
   `(succ 9) + (max 5 4) + 1`
   (e) If a function takes two parameters, we can also call it as an infix function by surrounding it with backticks.
   `div 92 10`
   `92 `div` 10`
   (f) Write your own functions:
   - how to make functions (contents in **myfilename**.hs):
     `doubleMe x = x + x`

- how to make use of it:
  `:l` **myfilename**
  `doubleMe 9`

- some examples:
  ```
  doubleMe x = x + x
  doubleUs x y = x*2 + y*2
  ```

  by having these, we can also run:
  `doubleUs 28 88 + doubleMe 123`

  we can also redefine the function `doubleUs` as:
  `doubleUs x y = doubleMe x + doubleMe y`

- Functions in Haskell don't have to be in any particular order, so it doesn't matter if you define `doubleMe` first and then `doubleUs` or if you do it the other way around.

- `if` statement:
  ```
  doubleSmallNumber x = if x > 100
  ........................then x
  ........................else x*2
  ```
  (Each '.' indicates a space. Because I fail to create spaces :P)
  - the `else` part is mandatory in Haskell.
  - `if` statement in Haskel is an expression:
    ```
    doubleSmallNumber' x = (if x > 100 then x else x*2)
    + 1
    ```
    notes: That apostrophe (') doesn't have any special meaning in Haskell's syntax. It's a valid character to use in a function name. We usually use ' to either denote a strict version of a function (one that isn't lazy) or a slightly modified version of a function or a variable.

- what is more:
  - Functions can't begin with uppercase letters.
  - When a function doesn't take any parameters, we usually say it's a definition (or a name):
    `conanO'Brien = "It's a-me, Conan O'Brien!"`