```
Prelude> :?
 Commands available from the prompt:

   <statement>                 evaluate/run <statement>
   :                           repeat last command
   :{\n ..lines.. \n:}\n       multiline command
   :add [*]<module> ...        add module(s) to the current target set
   :browse[!] [[*]<mod>]       display the names defined by module <mod>
                               (!: more details; *: all top-level names)
   :cd <dir>                   change directory to <dir>
   :cmd <expr>                 run the commands returned by <expr>::IO String
   :complete <dom> [<rng>] <s> list completions for partial input string
   :ctags[!] [<file>]          create tags file <file> for Vi (default: "tags")
                               (!: use regex instead of line number)
   :def <cmd> <expr>           define command :<cmd> (later defined command has
                               precedence, ::<cmd> is always a builtin command)
   :edit <file>                edit file
   :edit                       edit last module
   :etags [<file>]             create tags file <file> for Emacs (default: "TAGS")
   :help, :?                   display this list of commands
   :info[!] [<name> ...]       display information about the given names
                               (!: do not filter instances)
   :issafe [<mod>]             display safe haskell information of module <mod>
   :kind[!] <type>             show the kind of <type>
                               (!: also print the normalised type)
   :load[!] [*]<module> ...    load module(s) and their dependents
                               (!: defer type errors)
   :main [<arguments> ...]     run the main function with the given arguments
   :module [+/-] [*]<mod> ...  set the context for expression evaluation
   :quit                       exit GHCi
   :reload[!]                  reload the current module set
                               (!: defer type errors)
   :run function [<arguments> ...] run the function with the given arguments
   :script <file>              run the script <file>
   :type <expr>                show the type of <expr>
   :undef <cmd>                undefine user-defined command :<cmd>
   :!<command>                 run the shell command <command>

 -- Commands for debugging:

   :abandon                    at a breakpoint, abandon current computation
   :back [<n>]                 go back in the history N steps (after :trace)
   :break [<mod>] <l> [<col>]  set a breakpoint at the specified location
   :break <name>               set a breakpoint on the specified function
   :continue                   resume after a breakpoint
   :delete <number>            delete the specified breakpoint
```

```
  :delete *                   delete all breakpoints
  :force <expr>               print <expr>, forcing unevaluated parts
  :forward [<n>]              go forward in the history N step s(after :back)
  :history [<n>]              after :trace, show the execution history
  :list                       show the source code around current breakpoint
  :list <identifier>          show the source code for <identifier>
  :list [<module>] <line>     show the source code around line number <line>
  :print [<name> ...]         show a value without forcing its computation
  :sprint [<name> ...]        simplified version of :print
  :step                       single-step after stopping at a breakpoint
  :step <expr>                single-step into <expr>
  :steplocal                  single-step within the current top-level binding
  :stepmodule                 single-step restricted to the current module
  :trace                      trace after stopping at a breakpoint
  :trace <expr>               evaluate <expr> with tracing on (see :history)

-- Commands for changing settings:

  :set <option> ...           set options
  :seti <option> ...          set options for interactive evaluation only
  :set args <arg> ...         set the arguments returned by System.getArgs
  :set prog <progname>        set the value returned by System.getProgName
  :set prompt <prompt>        set the prompt used in GHCi
  :set prompt2 <prompt>       set the continuation prompt used in GHCi
  :set editor <cmd>           set the command used for :edit
  :set stop [<n>] <cmd>       set the command to run when a breakpoint is hit
  :unset <option> ...         unset options

 Options for ':set' and ':unset':

   +m            allow multiline commands
   +r            revert top-level expressions after each evaluation
   +s            print timing/memory stats after each evaluation
   +t            print type after evaluation
   +c            collect type/location info after loading modules
   -<flags>      most GHC command line flags can also be set here
                       (eg. -v2, -XFlexibleInstances, etc.)
                 for GHCi-specific flags, see User's Guide,
                 Flag reference, Interactive-mode options

-- Commands for displaying information:

  :show bindings              show the current bindings made at the prompt
  :show breaks                show the active breakpoints
  :show context               show the breakpoint context
  :show imports               show the current imports
```

```
   :show linker               show current linker state
   :show modules              show the currently loaded modules
   :show packages             show the currently active package flags
   :show paths                show the currently active search paths
   :show language             show the currently active language flags
   :show <setting>            show value of <setting>, which is one of
                                 [args, prog, prompt, editor, stop]
   :showi language            show language flags for interactive evaluation
```

```
Prelude> let x = 1
Prelude> :show bindings
x :: Num t => t = _
```

I see that `x` is in fact an expression that return a numeric value, hence not an affectation.