Functional Programming
Symbolic AI is based on a representation of the world in terms of symbols and rules. Here, the AI comes in the form of algorithms that manipulate the symbols in order to solve problems or derive new knowledge. In order to explore such algorithms and experiment with them, we will use a functional programming language. Functional programming languages are well-suited for implementing complex algorithms in a concise and readable way and offer great flexibility in modifying, reusing, and testing code.

The Haskell Language
Traditionally, AI researchers and educators used the language LISP. LISP is a powerful language that is still being used; for example, emacs is programmed in LISP. However, in my opinion, LISP programs are not very readable or at least require some time to get used to. Therefore, we will use Haskell, which has a rather straightforward and clear syntax. It is similar to many other functional languages such as F#, ML, Miranda, or OCaml.

The Haskell Language
You can download the Haskell Platform and find tons of information about the language here: http://www.haskell.org
In particular, please study Chapters 1 to 6 of the free online Haskell Tutorial with the beautiful title: “Learn you a Haskell for Great Good!”
http://learnyouahaskell.com/
In the next lecture, we will review the Haskell basics, discuss your questions and look at some more advanced Haskell topics.

The Programs We Wrote Today
fact 0 = 1
fact n = n * fact (n - 1)
mymap f [] = []
mymap f (x:xs) = (f x):(mymap f xs)
 quicksort [] = []
quicksort (x:xs) = (quicksort smaller) ++ [x] ++ (quicksort greater)
   where smaller = filter (< x) xs
          greater = filter (>= x) xs
Today's Haskell Session (2)

*Main> 2^1000000
... a few hours later...
*Main> 888403162747109376
it :: Integer

*Main> 'a'
'a'
it :: Char

*Main> 'b'
'b'
it :: Char

*Main> 3 == 5
False
it :: Bool

Today's Haskell Session (3)

*Main> 5 == 5
True
it :: Bool

*Main> let double x = 2*x
double :: Num a => a -> a
*Main> double 4
8
it :: Integer

*Main> double (2.2)
4.4
it :: Double

*Main> let mult x y = x*y
mult :: Num a => a -> a -> a

Today's Haskell Session (4)

*Main> mult 4 7
28
it :: Integer

*Main> let mult 4.4 7.1
31.240000000000002
it :: Double

*Main> let timesthree = mult 3
timesthree :: Integer -> Integer

*Main> timesthree 4
12
it :: Integer

*Main> (5, 3)
(5,3)
it :: (Integer, Integer)

Today's Haskell Session (5)

*Main> 3:
[3,5]
it :: [Integer]

*Main> [1..10]
[1,2,3,4,5,6,7,8,9,10]
it :: [Integer]

*Main> ['a'..'z']
"abcdefghijklmnopqrstuvwxyz"
it :: [Char]

*Main> even 5
False
it :: Bool

*Main> mod 5 2
1
it :: Integer

Today's Haskell Session (6)

*Main> 3\[5\]
[3,5]
it :: [Integer]

*Main> [1..10]
[1,2,3,4,5,6,7,8,9,10]
it :: [Integer]

*Main> ['a'..'z']
"abcdefghijklmnopqrstuvwxyz"
it :: [Char]

*Main> even 5
False
it :: Bool

*Main> mod 5 2
1
it :: Integer

Today's Haskell Session (7)

*Main> let prime n = n > 1 && [x | x <- [2..(div n 2)], mod n x == 0] == []
prime :: Integral t => t -> Bool
Today's Haskell Session (8)

*Main> prime 2
True
it :: Bool

*Main> prime 3
True
it :: Bool

*Main> prime 4
False
it :: Bool

*Main> filter even [1..100]
[2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,40,42,44,46,48,50,52,54,56,58,60,62,64,66,68,70,72,74,76,78,80,82,84,86,88,90,92,94,96,98,100]
it :: [Integer]

Today's Haskell Session (9)

*Main> :t filter
filter :: (a -> Bool) -> [a] -> [a]

*Main> filter prime [1..100]
it :: [Integer]

*Main> :t filter prime [1..10000]
[2,3,5,7,11,13,17,19,23,29,31,37,41,43, and so on and on, ...
9871,9883,9887,9901,9907,9923,9929,9931,9941,9949,9967,9973]
it :: [Integer]

*Main> :t mult 5 6
50
it :: Integer

Today's Haskell Session (10)

*Main> :t map
map :: (a -> b) -> [a] -> [b]

*Main> map (mult 3) [1..10]
[3,6,9,12,15,18,21,24,27,30]
it :: [Integer]

*Main> map (* 3) [1..10]
[3,6,9,12,15,18,21,24,27,30]
it :: [Integer]

*Main> (+) 4 7
11
it :: Integer

Today's Haskell Session (11)

*Main> :t quicksort
quicksort :: Ord a => [a] -> [a]

*Main> quicksort "hello world!"
" !dehllloorw"
it :: [Char]

*Main> quicksort [1..100]
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40]
it :: [Integer]

Today's Haskell Session (12)

*Main> fact 0
1
it :: Integer

*Main> fact 1
1
it :: Integer

*Main> map fact [1..10]
[1,2,6,24,120,720,5040,40320,362880,3628800]
it :: [Integer]

*Main> :reload
[1 of 1] Compiling Main   ( test.hs, interpreted )
Ok, modules loaded: Main.

Today's Haskell Session (13)

*Main> mymap even [1..100]
[False,True,False,True,False,True,… and so on…, False, True]  
it :: [Bool]

*Main> mymap even [1..100]
[False,True,False,True,… and so on…, False, True]  
it :: [Bool]

*Main> :reload
[1 of 1] Compiling Main   ( test.hs, interpreted )
Ok, modules loaded: Main.

*Main> quicksort "hello world!"
" !dehllloorw"
it :: [Char]

*Main> quicksort :: Ord a => [a] -> [a]