Programming Languages: Functional Programming Worksheet for 1 & 2. Introduction to Haskell

Shin-Cheng Mu

Autumn 2025

- Everything in this worksheet are mentioned in the handouts already. This worksheet is a way to guide you through the materials so that you can learn mostly by yourself.
- To start with, read through the handout "1. Introduction to Haskell: Value, Functions, And Types". Do not worry if you cannot understand everything — that is what this worksheet is for.
- Read the first two pages of the handout "2. Introduction to Haskell: Simple Datatypes & Functions on Lists" (up to Section 2.1 "List Generation").
- Start `ghci` and try the following tasks.

Functions Definitions and Types

Be sure that you have downloaded `MiniPrelude.hs`. Create a new file in your editor, starting with the two lines:

```
import Prelude ()
import MiniPrelude
```

And try questions 3 - 10 of Practicals 1, if you have not done so.

Recall that once you `import MiniPrelude`, the operators for addition and multiplication for $Float$ are renamed to `(+.)` and `(*.)`.

Some of the questions are simply designed to get you familiar with the syntax (of function definitions, **let**, **where**, guardeds, etc). Pay attention to function composition $(.)$ and be sure that you understand it. The latter part of the questions are about types and could be confusing. Feel free to ask the instructor, if needed.

List Deconstruction

The following can be done in `ghci`.

1. (a) What is the type of the function $head$? Use the command `:t` to find out the type of a value.

   (b) Since the input type of $head$ is a list ($List\ a$), let us try it on some input.

      i. $head\ [1,2,3] =$
      ii. $head$ "abcde" $=$
      iii. $head\ [\,] =$

   (c) Recall that the type $String$ in Haskell is the same as $List\ Char$. Notice that $[1,2,3]$ and "abcde" have different types. Why can we apply the same function $head$ to them? Read the part about polymorphic type in the handouts if you are not sure.

   (d) In words, what does the function $head$ do?

2. (a) What is the type of the function $tail$?

   (b) Try $tail$ on some input.

      i. $tail\ [1,2,3] =$
      ii. $tail$ "abcde" $=$
      iii. $tail\ [\,] =$

   (c) In words, what does the function $tail$ do?

   (d) Is it true that $head\ xs : tail\ xs = xs$, for *all* xs? Can you think of an xs for which the property does not hold?

3. (a) The function $(:)$ should have type $a \to List\ a \to List\ a$. (If you try to find out its type in `ghci`, you will currently see $a \to [a] \to [a]$. Well... in fact, were it not for `MiniPrelude`, $List\ a$ should actually be written $[a]$ in Haskell. I prefer $List\ a$, however). Try $tail$ on some input.

      i. $1 : [2,3] =$
      ii. $'a' :$ "bcde" $=$
      iii. $[1] : [2,3] =$
      iv. $[1] : [[2],[3,4]] =$
      v. $[[1]] : [[2],[3,4]] =$
      vi. $[1,2] : 3 =$
      vii. $[1,2] : [3] =$

Hmm... many of the attempts above fail. Think about why.

(b) In words, what does the function $(:)$ do?

Note: $(:)$ and $[\,]$ are in fact the primitive constructors of the datatype $List$. All Haskell lists, in essence are constructed by $(:)$ and $[\,]$, while $[1,2,3]$ is just a convenient notation for $1:2:3:[\,]$. Read page 2 of Handout 2 again, if you haven't.

4. (a) What is the type of the function $(++)$? (In ASCII one types ++.)

(b) Try $(++)$ on some input.

i. $[1,2,3]++[4,5] =$
ii. $[]++[4,5] =$
iii. $[1,2]++[] =$
iv. $[1]++[2,3] =$
v. $[1,2]++[3] =$
vi. $[1]++[[2],[3,4]] =$
vii. $[[1]]++[[2],[3,4]] =$
viii. $[\,]++[\,] =$
ix. $[1]++[\,]++[2,3] =$
x. $1++[2,3] =$
xi. $[1,2]++3 =$

Some of the attempts above fail. Think about why.

(c) In words, what does the function $(++)$ do?

(d) Both $(:)$ and $(++)$ seem to concatenate things into lists. How are they different?

In fact, one of them is defined in terms of the other. We will see later in this course.

5. (a) What is the type of the function $last$?

(b) Try $last$ on some input. Think about some input yourself.

i. $last \quad =$
ii. $last \quad =$
iii. $last \quad =$

(c) In words, what does the function $last$ do?

6. (a) What is the type of the function $init$?

(b) Try $init$ on some input. Think about some input yourself. Do not just try inputs that are "safe". Try whether you can cause the function to fail.

i. $init \qquad =$

ii. $init \qquad =$

iii. $init \qquad =$

(c) In words, what does the function $init$ do?

(d) The functions $init$ and $last$ should be somehow related. How do you state their relationship formally? In other words, what property does $init$ and $last$ (and perhaps $(+\!+)$) jointly satisfy?

7. (a) What is the type of the function $null$?

(b) Try $init$ on some input. Think about some input yourself.

i. $null \qquad =$

ii. $null \qquad =$

iii. $null \qquad =$

(c) Can you write down a definition of $null$, by pattern matching?

List Generation

1. What are the results of the following expressions?

(a) $[0..25] \; =$

(b) $[0, 2..25] \; =$

(c) $[25..0] \; =$

(d) $['a'..'z'] \; =$

(e) $[1..] \; =$

2. What are the results of the following expressions?

(a) $[x \mid x \leftarrow [1..10]] \; =$

(b) $[x \times x \mid x \leftarrow [1..10]] \; =$

(c) $[(x, y) \mid x \leftarrow [0..2], y \leftarrow \texttt{"abc"}] \; =$

(d) What is the type of the expression above?

(e) $[x \times x \mid x \leftarrow [1..10], \mathit{odd}\ x] \; =$

3. What are the results of the following expressions?

(a) $[(a, b) \mid a \leftarrow [1..3], b \leftarrow [1..2]] =$

(b) $[(a, b) \mid b \leftarrow [1..2], a \leftarrow [1..3]] =$

(c) $[(i, j) \mid i \leftarrow [1..4], j \leftarrow [(i + 1)..4]] =$

(d) $[(i, j) \mid i \leftarrow [1..4], \mathit{even}\ i,\ j \leftarrow [(i + 1)..4], \mathit{odd}\ j] =$

(e) $['a'|i \leftarrow [0..10]] \; =$

Combinators on Lists

1. (a) What is the type of the function $(!!)$ (two exclamation marks)?

   (b) Try $(!!)$ on some input. Think about some input yourself. Note that $(!!)$ is an infix operator. Try whether you can cause the function to fail.

   i. $[1,2,3] \,!!\, 1 \;=$

   ii. $\qquad\qquad !! \quad =$

   iii. $\qquad\qquad !! \quad =$

   (c) In words, what does the function $(!!)$ do?

2. (a) What is the type of the function $length$?

   (b) Try $length$ on some input.

   i. $length \qquad\qquad =$

   ii. $length \qquad\qquad =$

   (c) In words, what does the function $length$ do?

3. (a) What is the type of the function $concat$?

   (b) Try $concat$ on some input.

   i. $concat \qquad\qquad =$

   ii. $concat \qquad\qquad =$

   (c) In words, what does the function $concat$ do?

   (d) Again, $(:)$, $(++)$, and $concat$ all seem to concatenate things into lists. How are they different?

4. (a) What is the type of the function $take$?

   (b) Try $take$ on some input. Since $take$ expects an integer and list, try it on some extreme cases. For example, when the integer is zero, negative, or larger than the length of the list.

i. $take \quad =$
ii. $take \quad =$
iii. $take \quad =$

(c) In words, what does the function $take$ do?

5. (a) What is the type of the function $drop$?

(b) Try $drop$ on some input. Like $take$, try it on some extreme cases.

i. $drop \quad =$
ii. $drop \quad =$
iii. $drop \quad =$

(c) In words, what does the function $drop$ do?

(d) Does $take$, $drop$, and $(++)$ together satisfy some properties?

6. (a) What is the type of the function map?

(b) Try map on some input. It is a little bit harder, since map expects a functional argument.

i. $map\ square\ [1,2,3,4] \ =$
ii. $map\ (1+)\ [1,2,3,4] \ =$
iii. $map\ (const\ 'a')\ [1..10] \ =$

(c) In words, what does the function map do?

(d) Is $(1+)$ a function? Try it.

i. $(1+)\ 2 \ =$
ii. $((1+) \cdot (1+) \cdot (1+))\ 0 \ =$
where $(\cdot)$ is function composition.

Sectioning

- Infix operators are *curried* too. The operator $(+)$ may have type $Int \to Int \to Int$.

- Infix operator can be partially applied too.

$$(x \oplus)\ y = x \oplus y$$
$$(\oplus\ y)\ x = x \oplus y$$

  - $(1+) :: Int \to Int$ increments its argument by one.
  - $(1.0\ /) :: Float \to Float$ is the "reciprocal" function.
  - $(/\ 2.0) :: Float \to Float$ is the "halving" function.

1. Define a function $doubleAll :: List\ Int \to List\ Int$ that doubles each number of the input list. E.g.
   - $doubleAll\ [1,2,3] = [2,4,6]$.

   - How do you define a new function? You have to do that in a file, not in `ghci`. [1] Define the file you created in the beginning of this exercise. If you have not done so yet, you should
     (a) create a new text file (using your favourite editor) in your current working directory (the directory you executed `ghci`). The file should have extension `.hs`.
     (b) Type your definitions in the file.
     (c) Load the file into `ghci` by the command `:l <filename>`.
2. Define a function $quadAll :: List\ Int \to List\ Int$ that multiplies each number of the input list by 4. Of course, it's cool only if you define $quadAll$ using $doubleAll$.

λ Abstraction

- Every once in a while you may need a small function which you do not want to give a name to. At such moments you can use the λ notation:
  - $map\ (\lambda x \to x \times x)\ [1,2,3,4] = [1,4,9,16]$
  - In ASCII λ is written $\backslash$.

1. What is the type of $(\lambda x \to x + 1)$?
2. $(\lambda x \to x + 1)\ 2\ =$

[1] Well, you *can* define new functions in `ghci` but let's not go there...

3. What is the type of $(\lambda x \to \lambda y \to x + 2 \times y)$?
4. What is the type of $(\lambda x \to \lambda y \to x + 2 \times y)\ 1$?
5. $(\lambda x \to \lambda y \to x + 2 \times y)\ 1\ 2\ =$
6. What is the type of $(\lambda x\ y \to x + 2 \times y)$?
7. What is the type of $(\lambda x\ y \to x + 2 \times y)\ 1$?
8. $(\lambda x\ y \to x + 2 \times y)\ 1\ 2\ =$
9. Define $doubleAll :: List\ Int \to List\ Int$ again. This time using a λ expression.

10. **Pattern matching in** λ. To extract, for example, the two components of a pair
    (a) What is the type of $(\lambda(x, y) \to (y, x))$?
    (b) $(\lambda(x, y) \to (y, x))\ (1, \text{'}a\text{'})\ =$
    (c) Alternatively, try
    $(\lambda p \to (snd\ p, fst\ p))\ (1, \text{'}a\text{'})\ =$

Back to Lists

1. (a) What is the type of the function $filter$?
   (b) Try $filter$ on some input.
      i. $filter\ even\ [1..10]\ =$
      ii. $filter\ (> 10)\ [1..20]\ =$
      iii. $filter\ (\lambda x \to x\ `mod`\ 3 == 1)\ [1..20]\ =$
   (c) In words, what does the function $filter$ do?

2. (a) What is the type of the function $takeWhile$?
   (b) Try $takeWhile$ on some input.
      i. $takeWhile\ even\ [1..10]\ =$

ii. $takeWhile\ (<10)\ [1..20]\ =$

iii. $takeWhile\ (\lambda x \to x\ `mod`\ 3 == 1)\ [1..20]\ =$

(c) In words, what does the function $takeWhile$ do? How does it differ from *filter*?

(d) Define a function $squaresUpto :: Int \to List\ Int$ such that $squaresUpto\ n$ is the list of all positive square numbers that are at most n. For some examples,

- $squaresUpto\ 10 = [1, 4, 9]$.
- $squaresUpto\ (-1) = [\,]$

3. (a) What is the type of the function $dropWhile$?

(b) Try $dropWhile$ on some input.

i. $dropWhile\ even\ [1..10]\ =$

ii. $dropWhile\ (<10)\ [1..20]\ =$

iii. $dropWhile\ (\lambda x \to x\ `mod`\ 3 == 1)\ [1..20]\ =$

(c) In words, what does the function $dropWhile$ do?

4. (a) What is the type of the function zip?

(b) Try zip on some input.

i. $zip\ [1..10]$ "abcde" $=$

ii. zip "abcde" $[0..]\ =$

iii. $zip \qquad\qquad\qquad =$

(c) In words, what does the function zip do?

(d) Define $positions :: Char \to String \to List\ Int$, such that $positions\ x\ xs$ returns the positions of occurrences of x in xs. E.g.

- $positions$ 'o' "roodo" $= [1, 2, 4]$.

Check the handouts if you just cannot figure out how.

(e) What if you want only the position of the *first* occurrence of x? Define $pos :: Char \rightarrow String \rightarrow Int$, by reusing $positions$.

Morals of the Story

- Lazy evaluation helps to improve modularity.
  - List combinators can be conveniently re-used. Only the relevant parts are computed.
- The combinator style encourages "wholemeal programming".
  - Think of aggregate data as a whole, and process them as a whole!