Lecture 35:

- Haskell Functions

Announcements:

- HW-7 out
**Boolean Expressions**

- Boolean values are `True` or `False` (instead of 1 and 0)

  ```haskell
  Prelude> True && False
  False

  Prelude> False || True
  True

  Prelude> True && 1
  <interactive>:1:8:
    • No instance for (Num Bool) arising from the literal `1'
    • In the second argument of `(&&)', namely `1'
      In the expression: True && 1
      In an equation for `it': it = True && 1
  ```

- Dissecting the error message:

  ```
  No instance for (Num Bool)
  - `ghci` tries to treat the numeric value 1 as a `Bool` (which fails)
  - Here it is saying `Bool` is not a member of the `Num` typeclass
  ```
Boolean comparisons are similar to C derivatives (C++, Java, etc.)

Prelude> 1 == 1
True

Prelude> 2 < 3
True

Prelude> 4 >= 3.99
True

Prelude> 2 /= 3 -- instead of !=
True

Prelude> not True -- instead of !
False

Prelude> not (2 > 3 || 3 < 2)
True
List Operations

List concatenation (++)

- Returns an entirely new list
- Values in the second list are appended to the values of the first list

\[
Prelude \> [3, 1, 3] \> (+) \> [3, 7]\]
\[
[3, 1, 3, 3, 7]
\]

\[
Prelude \> [True, False] \> (+) \> []\]
\[
[True, False]
\]

List construction (:

- List construction is also often called “cons”
- Creates entirely new list with 1 prepended to values of given list
- Entire list can be written as a sequence of cons operations

\[
Prelude \> 1 : [2, 3]\]
\[
[1, 2, 3]
\]

\[
Prelude \> 1 : 2 : 3 : []\]
\[
[1, 2, 3]
\]

- Is cons (:) right or left associative?
  - Right associative!
  - \(1 : (2 : (3 : []))\)
  - \(1 : 2\) is a type error since second operand is not a list!
List head (aka “car”) gives first value of a list

```
Prelude> head [4, 1, 5, 3]
4

Prelude> head []
*** Exception: Prelude.head: empty list
```

List tail (aka “cdr”) gives list minus head value

```
Prelude> tail [4, 1, 5, 3]
[1, 5, 3]

Prelude> tail [1]
[]

Prelude> tail []
*** Exception: Prelude.tail: empty list
```

List null checks for the empty list

```
Prelude> null [4, 1, 5, 3]
False

Prelude> null []
True
```
Haskell list “enumeration” syntax

Prelude> [1..4]
[1,2,3,4]

• Works for enumerable values (e.g., ["bar"..'foo"] won’t work)

• More examples:

  Prelude> [1, 4 .. 10]
  [1,4,7,10]
  Prelude> [1.0, 1.25 .. 2.0]
  [1.0,1.25,1.5,1.75,2.0]
  Prelude> [0.1, 0.3 .. 1.0] -- look out for precision!
  [0.1,0.3,0.5,0.7,0.8999999999999999,1.0999999999999999]
  Prelude> [10, 9 .. 1]
  [10,9,8,7,6,5,4,3,2,1]
  Prelude> [10, 6 .. 1]
  [10,6,2]
  Prelude> [1, 5 .. 4]
  [1]
  Prelude> ['a' .. 'd']
  "abcd"
  Prelude> ['a', 'c' .. 'g']
  "aceg"

Prelude> [1, 2, 4 .. 20]
<interactive>:1:9: parse error in input `..'
Haskell Tuples

A “tuple” is a fixed size collection of values

- Each tuple value can have a **different** type
  - Whereas each value of a list is of the same type
- Tuple values and types are denoted using parentheses ...

  Prelude> (10, "abc")
  (10, "abc")

  Prelude> :type (True, "yes")
  (True, "yes") :: (Bool, [Char])

The special “unit” tuple

  Prelude> ()
  ()

  Prelude> :type ()
  () :: ()

- Both a value and a type

There are **no** one-valued tuples in Haskell

  Prelude> :type ("hello")
  ("hello") :: [Char]
Lists and tuples can be nested to any depth

(555, ("Alice", "Smith"), ["manager", "developer"])

[[1, 2], [5, 4, 3], [2, 2]]

[('a', 'b'), ('c', 'd'), ('e', 'a')]

“Pairs” (2-tuples)

- Can access elements using \texttt{fst} and \texttt{snd} functions

\begin{verbatim}
Prelude> let p = (42, 67)
Prelude> fst p
42
Prelude> snd p
67
\end{verbatim}

- Note this only works with pairs (2-tuples)!

Better approach: “Pattern Matching” (first look)

\begin{verbatim}
pairAdd1 p = ((fst p) + 1,(snd p) + 1)
\end{verbatim}

... vs ...

\begin{verbatim}
pairAdd1 (x,y) = (x+1,y+1)
tripleAdd1 (x,y,z) = (x+1,y+1,z+1)
\end{verbatim}

- we are “matching” on subcomponents of the structure
- \textit{note}: this is \textbf{not} the same as a function parameter list!