Tuples and Lists

CSC345: Programming Languages and Paradigms

WCU
WEST CHESTER
UNIVERSITY

Number Typeclasses

- Tuples and Lists: constructed by combining a number of pieces of data into a single object
- Tuple: *fixed* number of values of *fixed* types (can be different types)
- List: *arbitrary* number of values, all of the *same* type

Tuples

- A pair is a 2-item tuple.
- Triple is a 3-item tuple.
- Quadruple is a 4-item tuple.
- …

```
p :: (Int, Char)
p = (5, 'r')
```

`(x,y)` notation used for both the type and pair value.

Type **`(t1, t2, …, tn)`** consist of tuples of values **`(v1, v2, …, vn)`** where **`v1 :: t1, … , vn :: tn`**

Lists

[] is the empty list

- Contains no items
- is an element of every list type

```
nums :: [Int]
nums = [1,2,4,9]
```

Collection of items of the given type

```
[v1, v2, v3, v4] :: [t]
v1, v2, v3, v4 :: t
v1 :: t
v2 :: t
v3 :: t
v4 :: t
```

```
nums2 :: [Int]
nums2 = []
```

Type Error:
stuff = [1, “rich”, ‘c’]

A **`String`** is really just a list of **`Char`**'s

```
hello1 :: [Char]
hello1 = ['h','e','l','l','o']

hello2 :: String
hello2 = "hello"

helloSame :: Bool
helloSame = hello1 == hello2
```

Nested Lists

```
nested :: [[Int]]
nested = [[1], [2,4,2], [], [5..10]]
```

Range Notation w/ Lists

- **`[m..n]`** is the list **`[m, m+1, …, n]`**
- If **`m > n`**, the list is empty
- **`[m,p..n]`** is the list where the first two elements are **`m`** and **`p`**, the last is **`n`**, and the step size is **`p-m`**

```
range :: [Int]
range = [1..100]
```

```
range2 :: [Int]
range2 = [1,3..9]
```

More Range Examples

Try out:

```
['a'.. 'm']
[7,6..3]
[0.0, 0.3 .. 1.0]
[3.1 .. 7.0]
```

- Last item will be element in the sequence that is closest to **`n`**
  - Overshooting possible

Using Tuples in Functions

- *Example:* a function that returns a compound result

```
minAndMax :: Int -> Int -> (Int, Int)
```

Using Tuples in Functions

```
addPair :: (Int, Int) -> Int
```

Tuple Selector Functions for pairs only

- Built-in to Haskell: fst and snd

- But fst and snd are easy to define for ourselves anyway

Alternative ways of writing **`addPair`**

1. Using fst and snd
2. Pattern matching

Example

```
shift :: ((Int,Int), Int) -> (Int, (Int, Int))
```

Example

```
maxOccurs :: Int -> Int-> (Int, Int)
maxThreeOccurs :: Int -> Int -> Int -> (Int, Int)
```