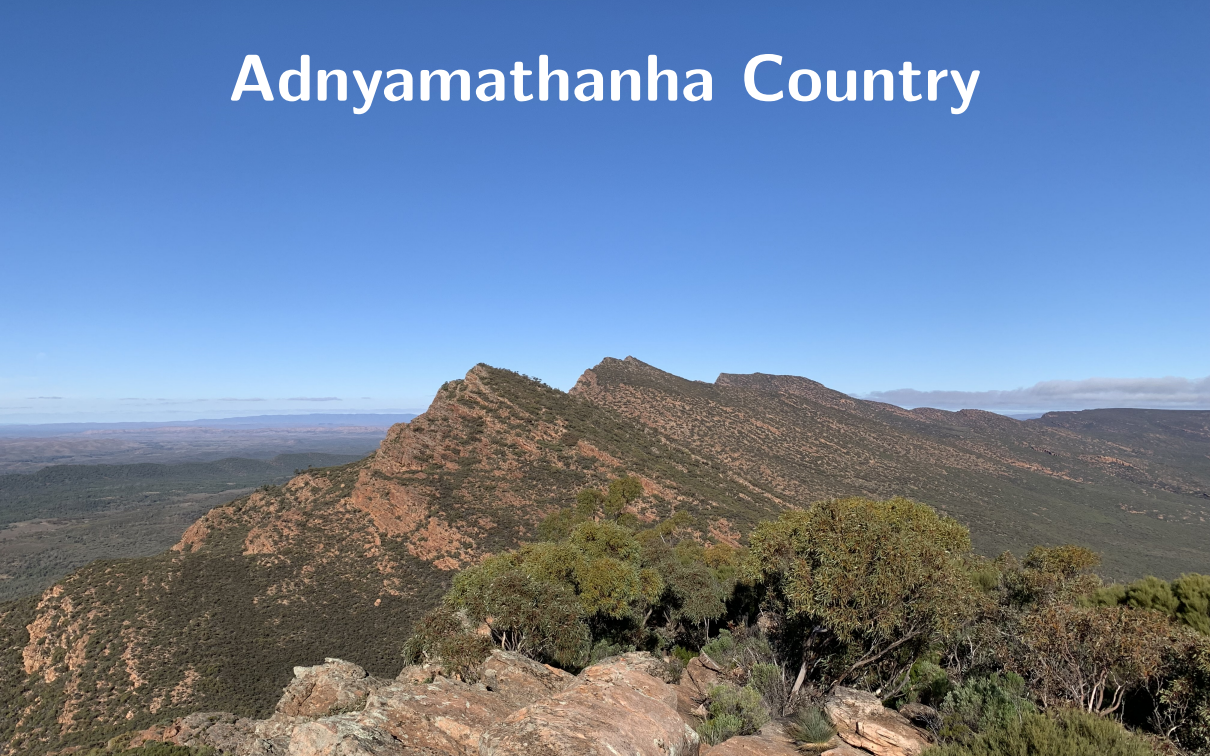
Adnyamathanha Country



We acknowledge, celebrate and pay our respects to the Ngunnawal and Ngambri people of the Canberra region and to all First Nations Australians on whose traditional lands we meet and work, and whose cultures are among the oldest continuing cultures in human history.

Learn more about Acknowledgement of Country here

In the photo: Adnyamathanha Country in South Australia

Find out more about Canberra's Aboriginal history here

Introduction to lists

Contents

Last week

(Thompson: §5.3, §14.1)

Recall how we can define **types** (constructors can have zero or more arguments)

```
1  data IntPlusBool = First Int | Second Bool  deriving Show
```

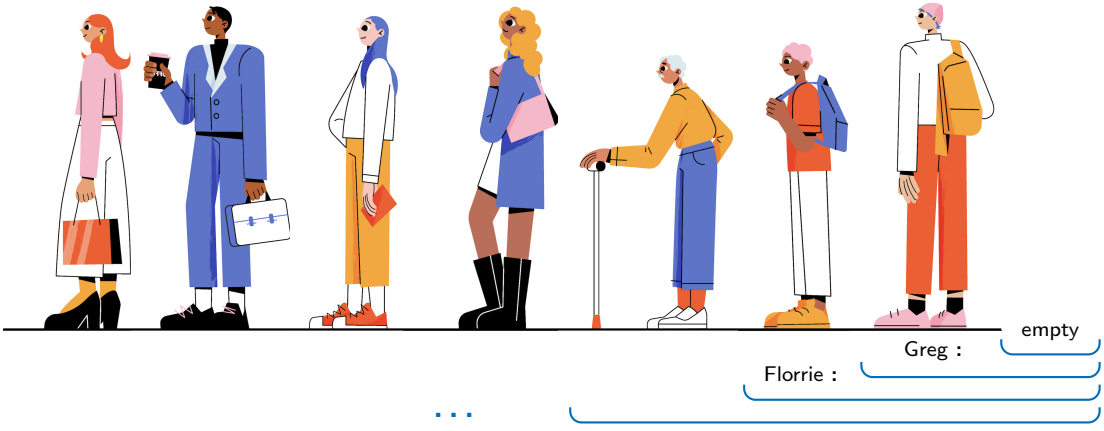
Recursive types

- A **recursive type** is one whose definition makes mention of the type itself
- This sounds circular, but it can be very useful!
- In this lecture we first consider one special recursive type: **lists**

See also: the Droste effect

Lists

(Thompson: §5.4 and §5.5)



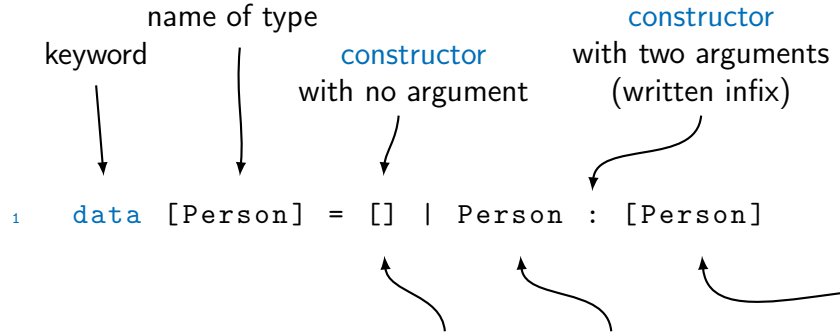
A list of persons is either **empty** or a **person in front of an existing list of persons**

Image by upklyak on Freepik

List of people

(THOMPSON: §5.4 and §5.5)

Suppose `Person` is a type, then Haskell has a built-in type of **lists** of people



So a **list** of `Persons` is either empty or a `Person` followed by a list of `Persons`

Remark It is new to see the name of the type we define on both sides of the "=" !

Special notation

(Thompson: §5.4 and §5.5)

A `[Person]` of length one can be written as

- `Alice : []`
- `[Alice]` (syntactic sugar)

A `[Person]` of length three can be written as

- `Alice : (Bob : (Charlie : []))`
- `Alice : Bob : Charlie : []` (right associativity)
- `[Alice, Bob, Charlie]` (syntactic sugar)

Lists of other types

(Thompson: §5.5)

Instead of `Person` we can also make lists of other datatypes

Example

```
1  data [Int] = [] | Int : [Int]
2  data [Char] = [] | Char : [Char]
3  data [Int -> Bool] = [] | (Int -> Bool) : [Int -> Bool]
```

In general, for any datatype `a` we have lists of elements in `a`

```
4  data [a] = [] | a : [a]
```

We say that **lists are polymorphic**

Strings

(Thompson: §5.5)

`String` is a type synonym for `[Char]`

```
1  type String = [Char]
```

We can use shorthand

```
2  "hello" == ['h','e','l','l','o']
3          == 'h':'e':'l':'l':'o':[]
```

We can use all standard list operations on `String`

Next lecture

- Tomorrow: **recursion**
- Next week: more recursion