pontus nagy

@p1xelHer0

an introduction to algebraic data types

what we'll do

- 👮 data types
- 😼 creating your own types
- 🐑 parametric polymorphism
- 🤔 what makes a data type algebraic?
- 🤦 why?

current data types in reason
how to create our own types
a concept to generalise types
why algebraic
why even use them?

👮 recap

- integers, floating-point numbers
- boolean, character, string
- tuple, array, list, records, unit

tuple is a parenthesis, can contain different types
list is an immutable array
records typed objects

👸 creating our player

```
type constructor    data constructor
type hero         = Agnes;

let player: hero = Agnes;
```

type keyword, type constructor.

a type needs data => data constructor

0 or more parameters

❤️ hit points

```
type constructor    data constructor
type health       = Health(int);

let hp: health = Health(3);
```

data constructor parameter, type

🐏 parametric polymorphism

```
type constructor    data constructor
type health('a)   = Health('a);

let hp: health(int) = Health(3);
let hp: health(float) = Health(3.5);
```

generalise types

alpha parameter type constructor

type needs a parameter to be "complete"

🤔 what makes a data type algebraic?

*“an algebraic data type is a type formed by combining other types” **

- me, right now 🤡

** I don't know the exact exact exact definition, but it's not super relevant to us right now!*

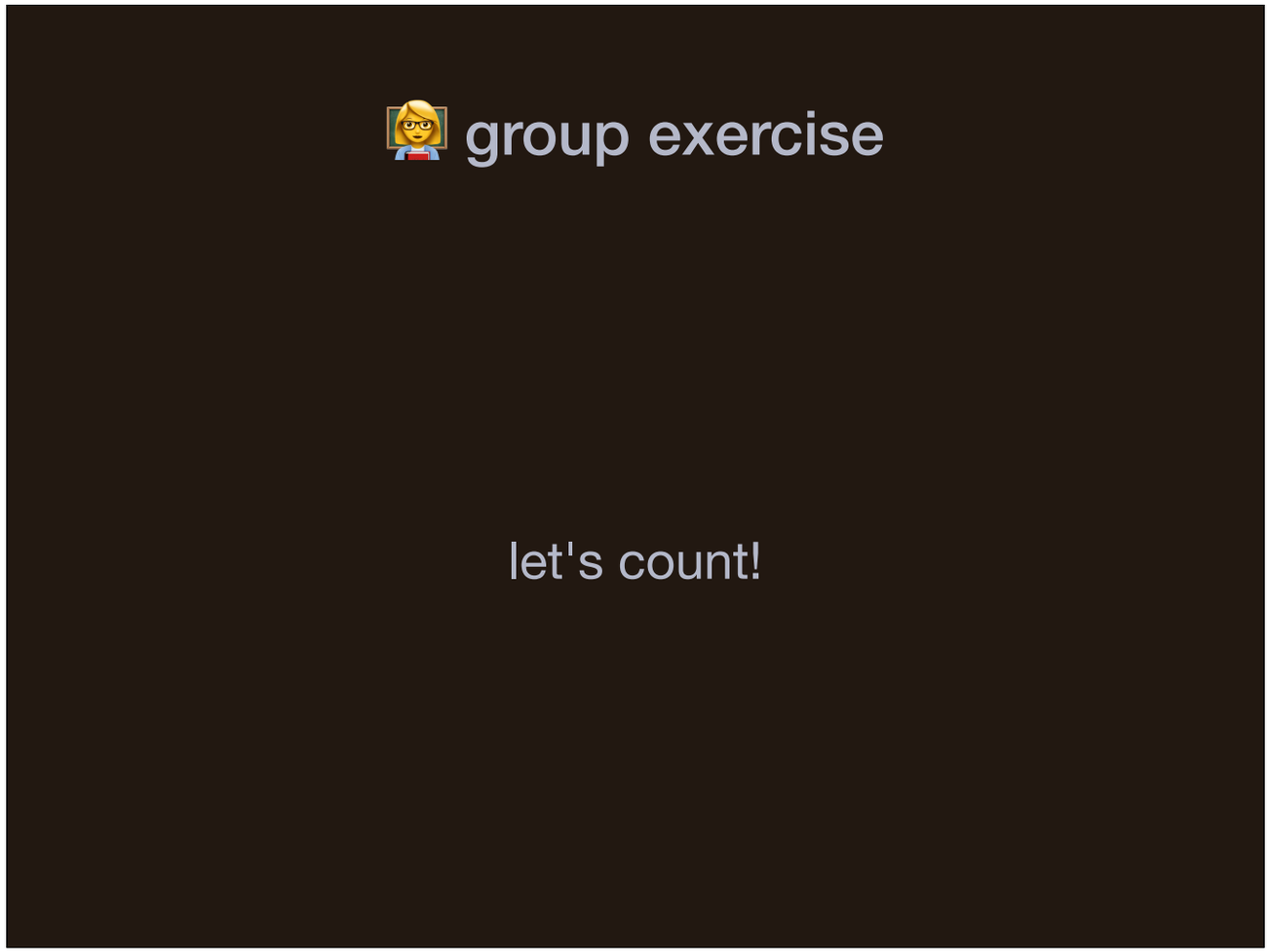
we are going to jump right into an algebraic data type!

💎 materials

```
type constructor     data constructor
type material    = | Bronze
                   | Iron
                   | Steel;
```

logical disjunction, "or"

only value at the time



group exercise!

count the possible data constructors

⏱️ let's count

```
type constructor    data constructor
type hero         = Agnes;
```

⏱️ let's count, pt 2

```
type constructor     data constructor
type material    = | Bronze
                   | Iron
                   | Steel;
```

1 + 1 + 1

⏱️ let's count, pt 3

```
type constructor   data constructor
type boolean     = True | False;
```

1 + 1, you have probably seen this before

⏱️ let's count, pt 4

```
type constructor   data constructor
type health('a)  = Dead | Health('a);
```

improved health

alpha changes count?

no, we are counting data constructors!

🌓 sum type

- set-theoretic sum
- A but not B
- "sum type"
- many names

sum, result of addition
logical disjunction
variant, union type, tagged union, many names

🤔 is there more?

are there more algebraic data types?

throw another example in their face

⚔️ our hero needs a weapon

```
type material =
  | Bronze
  | Iron
  | Steel;

type blade =
  | Axe
  | Sword;

type weapon = Weapon(blade, material);
```

```
type weapon = {
  blade: blade,
  material: material,
};

type weapon = (blade, material)
```

dual of sum, product

well known concept, without knowing the definition

"you probably know this" (pt2)

 replacing booleans

```
type attack = {
  damage: int,
  miss: boolean,
};
```

```
type attack =
  | Hit(int)
  | Miss;
```

replace booleans, make "illegal states unrepresentable"

we cant miss and have a damage above 0

🤦‍♀️ pattern matching

- defined by the `switch` keyword
- used to safely access data

i love pattern matching

👩‍🎤 example

```
let matchConstructors = typeConstructor =>
  switch (typeConstructor) {
  | dataConstructor1(value) => 1
  | dataConstructor2(value) => 2
  | dataConstructor3(value) => 3
  };
```

value for each data constructor on the type constructor

👩‍🌾 merchant

```
let rateMaterial = material =>
  switch (material) {
  | Steel => 3
  | Iron => 2
  | Bronze => 1
  };
```

exhaustiveness!

👩‍🌾 a needy merchant

```
let rateMaterial = material =>
  switch (material) {
  | Steel => 3
  | _ => 0
  };
```

default case, "for every other type"

👩‍🌾 merchant, slightly better

```
let rateMaterial = material =>
  switch (material) {
  | Steel => 3
  | Bronze
  | Iron => 0
  };
```

exhaustiveness helps us! adding new material => error

🙈 taking damage

```
type attack =
  | Hit(int)
  | Miss;

let takeDamage = (attack, playerArmour: material) =>
  switch(attack) {
  | Hit(damage) => calculateDamage(damage, playerArmour)
  | Miss => 0
  };

let calculateDamage = (damage, material) =>
  switch(material) {
  | Bronze => damage
  | Iron => damage - 1
  | Steel => damage - 2
  };
```

example of taking damage

🤓 more?

some examples, using all the things we are trying to learn!

🙅‍♀️ null? no thanks

```
type option('a) =
  | None
  | Some('a);
```

🔗 linked list

```
type list('a) =
  | Empty
  | Cons('a, list('a));
```

resursive type

🔗 linked list

```
type list('a) =
  | Empty
  | Cons('a, list('a));

let rec length = list =>
  switch(list) {
  | Empty => 0
  | Cons(_head, tail) => 1 + length(tail)
  };
};
```

example of recursion

🔗 linked list

```
type list('a) =
  | Empty
  | Cons('a, list('a));

let rec head = list =>
  switch (list) {
  | Empty => None
  | Cons(head, _tail) => Some(head)
  };

let rec tail = list =>
  switch (list) {
  | Empty => None
  | Cons(_head, tail) => Some(tail)
  };
```

exhaustiveness => option is needed

head: 'a
tail: list('a)

Empty: ???

🌺 almost done

questions? 🤷‍♂️

🏁 the end

thanks for listening ❤️