Records

This lesson will highlight the basic features of the record data structure.

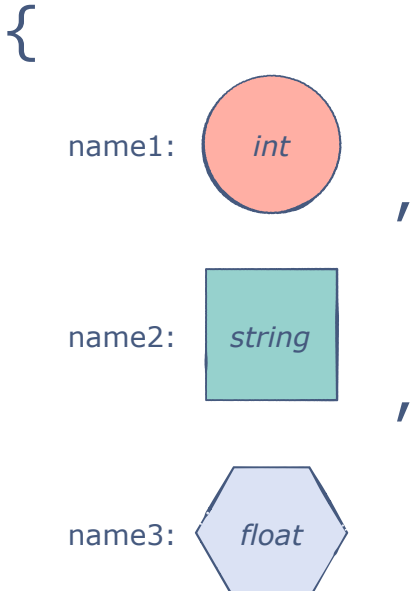
WE'LL COVER THE FOLLOWING

The Structure

Records are similar to tuples in the sense that they can store components of different types and are also immutable by default.

The beauty of a record lies in the fact that **each component has a name or label to identify it** within the record.

A component in a record is known as a field. Each field of a record is enclosed in curly braces, `{}`.



```
};
```

Creating a Record

Before we can create the actual record, we must define the types of all the names/identifiers that we will use in it.

The structure of the type definition is identical to that of the record itself. Let's define a `wizardInfo` type which we'll use to create a record:

```
type wizardInfo = {
  name: string,
  age: int,
  school: string,
  house: string
};
```

Our `wizardInfo` type contains 4 fields specifying the details of a record.

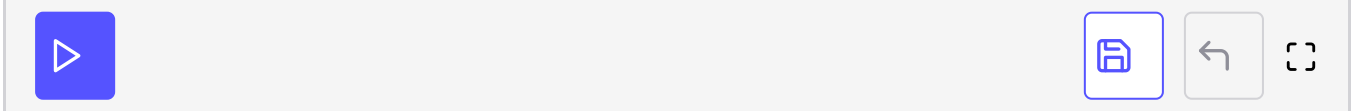
Below, we can find the syntax for creating a simple `wizard` record:

```
type wizardInfo = {
  name: string,
  age: int,
  school: string,
  house: string
};

/* Without type annotation */
let wizard = {
  name: "Harry",
  school: "Hogwarts",
  house: "Gryffindor",
  age: 14
};

/* With type annotation */
let wizard2: wizardInfo = {
  name: "Draco",
  school: "Hogwarts",
  house: "Slytherin",
  age: 14
};

Js.log(wizard);
Js.log(wizard2);
```



Voila! We've created our first record without any problems.

Voila! We've created our first record without any problems.

A cool thing to notice is that the order in which we set the values of our fields does not matter.

The record only uses field names to figure out values. Hence, **order does not matter in records**.

Accessing Record Fields

In Reason, we can access a particular field of a record using the `.` operator:

```
type wizardInfo = {
  name: string,
  age: int,
  school: string,
  house: string
};

let wizard = {
  name: "Harry",
  school: "Hogwarts",
  house: "Gryffindor",
  age: 14
};

/* Name */
Js.log(wizard.name);

/* House */
Js.log(wizard.school);
```

Pattern Matching in Records

Another convenient way of extracting a record's field values is to specify the desired fields within a pattern.

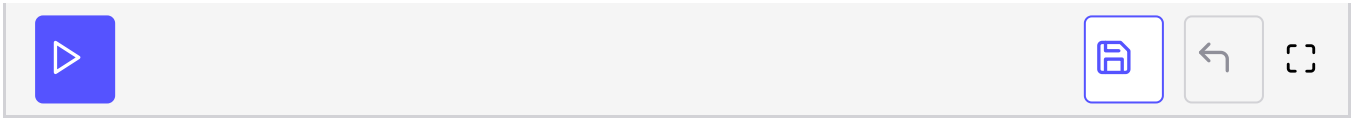
In records, the names in the pattern **must** match the ones in the record. Furthermore, a record's pattern does not need to mention all the field names:

```
type wizardInfo = {
  name: string,
  age: int,
  school: string,
  house: string
};
```

```
let wizard = {
  name: "Harry",
  school: "Hogwarts",
  house: "Gryffindor",
  age: 14
};

let {school, age} = wizard;

Js.log(school);
Js.log(age);
```



Things to Remember

- The names of all the fields in a record should be different.
- Records are immutable by default.
- When creating a record, all of the fields must be given some value.

In the next lesson, we'll further explore the behavior of records.