# Tuples

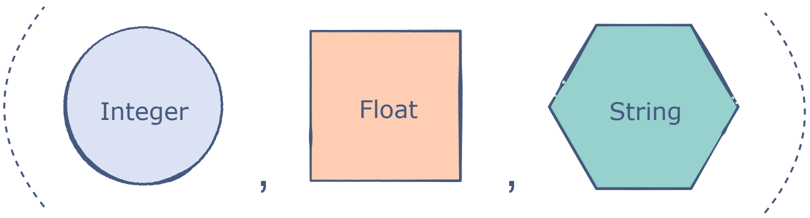
This lesson will highlight the key features of the tuple data structure.

WE'LL COVER THE FOLLOWING

## The Structure

A tuple is an **ordered** group of data elements. These elements can have different types.

The size of the tuple can be anything. However, once it has been created, the tuple is **immutable**.



The elements of a tuple are known as its **components**.

The tuple is one of the most basic data structures in Reason. We'll see how it forms the basis for more complex structures.

Let's take a look at the syntax of a tuple:

```
Js.log(("abc", 123, true)); /* A tuple containing a string, an integer, and a bool */
let t = (50, 20.5);
Js.log(t); /* [50, 20.5] */
```

The components of a tuple are always enclosed in parentheses. In the code above, the contents of tuple `t` cannot be changed after it has been declared.

Similar to the data types we explored earlier, each element of a tuple can have a **type annotation** to explicitly specify its type.

## Nested Tuples

A powerful feature of the tuple is that it can contain another tuple!

Below we can find the syntax for achieving nested tuples:

```
let t1 = ("ABC", 84.97, (12, 23)); /* The inner tuple contains 12 and 23 */
Js.log(t1);

let t2 = (t1, true);
Js.log(t2);
```

## Type Definitions in Tuples

We can predefine a type which can be used for the structure of a tuple. Keep in mind that this the order of types in this definition will have to be followed in the tuple's values as well:

```
type superhero = (string, string, int);

let batman : superhero = ("Batman", "Gotham", 30);

Js.log(batman);

/* Erroneous code */
/* let spiderman : superhero = ("Spiderman", 24, "New York"); */
```

### Polymorphic Type Definitions

An entity is said to be polymorphic if it works with multiple data types.

The `type` keyword allows us to create a named, structured data type which

The `type` keyword allows us to create a named, structured data type which can have multiple members of any primitive data type.

The type created with the `type` keyword can have any name. Along with its name, we have to give a name to the types of members in the structure as well. But the names have to be preceded with a ‘. For example,

```
type myType(‘a, ’b);
```

Here, `‘a` and `‘b` are names of two different data types that will be members of myType. In general, Reason interprets type variables as `'a`, `'b`, `'c` etc., which stand for Alpha, Beta, Charlie, etc.

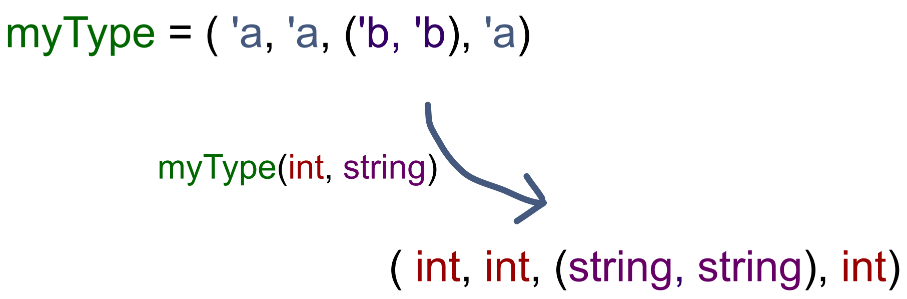
We can use type variables to create polymorphic type definitions for our tuples.

Let’s look at an example:

```
type myType('a, 'b) = ('a, 'a, ('b, 'b), 'a);

let t1: myType(int, string) = (1, 2, ("Hello", "World"), 3);

Js.log(t1);

let t2: myType(float, bool) = (1.5, 2.7, (false, false), 3.1);

Js.log(t2);
```

In the code above, we have specified that our `myType` can have two different data types for `'a` and `'b`.

Afterward, whenever we use `myType`, we must state the two types we are assigning to `'a` and `'b`. They can be the same type as well.


myType = ( 'a, 'a, ('b, 'b), 'a)
myType(int, string)
( int, int, (string, string), int)

*int is treated as 'a and*
*string is treated as 'b*

So just like that, we've created a polymorphic tuple type! Do not worry if this looks complicated right now. We'll explore polymorphism further in the later chapters of this course.

---

Now, the question arises, "how can we access the components of a tuple?"

This question will be answered in the next lesson, where we'll learn about **pattern matching**.