Rust structures.

In Rust we can group together basic types into *structures* which can contain data of different types.

There are three kinds of structure, depending on how verbose we want to be:

C-Style structures

In the C language, structures have names and each field has a name and a type

```
struct MyStruct {
    alice: u8,
    bob: usize,
    claire: f32,
}
```

In C, this structure will have these items in order and will take 24 bytes of memory because of *alignment*. In Rust, the compiler may re-order the items to use less memory - 16 bytes in this case.

We build and acces the structure like this:

```
let ms = MyStruct { alice: 1, bob: 2, claire: 2.0 }
println!("{}", ms.alice);
println!("{}", ms.bob);
println!("{}", ms.claire);
```

Tuple structs

If we are very lazy, we can omit the field names, but keep the struct name. We use parentheses instead of curly braces.

```
struct MyStruct2(u32);
struct MyStruct3(u64, f32);
```

We build and access the structure like this:

```
let ms2 = MyStruct2(1);
let ms3 = MyStruct3(2, 3.0);
println!("{}", ms2.0);
println!("{}", ms3.0);
println!("{}", ms3.1);
```

The case where there is only one parameter is know as a *new type* and is used instead of *typedef* to provide strong type control.

Tuples

If we are even lazier, we can omit the struct name too.

```
let ms4 = (1, 1.0, "hello");
println!("{}", ms4.2);
```

Wars have been fought over the pronunciation of *tuple* vacillating between *toople* to *tupple*

Conclusion

All these structs are the same as far as the computer is concerned. It is happy for us meatbags to name things how we want.

If you want to learn more about Rust, Blockchains, Physics and other subjects, drop me, Amy, a line.