

# The Syntax

Below is the syntax we can use to access and bind tuple values with variables safely.

The *basic syntax* for **structured bindings** is as follows:

```
auto [a, b, c, ...] = expression;  
auto [a, b, c, ...] { expression };  
auto [a, b, c, ...] ( expression );
```



The compiler introduces all identifiers from the `a, b, c, ...` list as names in the surrounding scope and binds them to sub-objects or elements of the object denoted by expression.

Behind the scenes, the compiler might generate the following *pseudo-code*:

```
auto tempTuple = expression;  
using a = tempTuple.first;  
using b = tempTuple.second;  
using c = tempTuple.third;
```



Conceptually, the expression is copied into a tuple-like object ( `tempTuple` ) with member variables that are exposed through `a`, `b` and `c`. However, the variables `a`, `b` and `c` are not references; they are aliases (or bindings) to the generated object member variables. The temporary object has a unique name assigned by the compiler.

For example:

```
std::pair a(0, 1.0f);  
auto [x, y] = a;
```



`x` binds to `int` stored in the generated object that is a copy of `a`. And similarly, `y` binds to `float`.

Bindings with types such as `int` and `float` also offer modifier compatibility.  
More on this in the next lesson.