The Tuple Version

The lesson elaborates how to return multiple values using tuple.

The first step is to convert the output parameters into a tuple and return it from the function.

According to F.21: To return multiple "out" values, prefer returning a tuple or struct:

> **Do you know?**
>
> *A return value is self-documenting as an "output-only" value. Note that* `C++` *does have multiple return values, by the convention of using a tuple (including pair), possibly with the extra convenience of a tie at the call site.*

After the change the code might look like this:

```
std::tuple<bool, bool, bool, int>
CheckSelectionVer2(const ObjSelection &objList)
{
  if (!objList.IsValid())
    return {false, false, false, 0};

  // local variables:
  int numCivilUnits = 0;
  int numCombat = 0;
  int numAnimating = 0;

  // scan...
  return {true, numCivilUnits > 0, numCombat > 0, numAnimating };
}
```

A bit better… isn't it? The tuple version has the following advantages:

- There's no need to check raw pointers.
- Code is more expressive. We return everything in a single object.

What's more on the caller site, you can use Structured Bindings to wrap the returned tuple:

```
int main(){
  ObjSelection sel;

  bool anyCivilUnits = false;
  bool anyCombatUnits = false;
  int numAnimating = 0;

  if (auto [ok, anyCivilVer2, anyCombatVer2, numAnimatingVer2] = CheckSelectionVer2(sel); ok)
    std::cout << "ok...\n";
}
```

Unfortunately, this version might not be the best one. For example, there's a risk of forgetting the order of outputs from the tuple.

The problem of function extensions is also still present. So when you'd like to add another output value, you have to extend this tuple and the caller site.

We can fix this with one further step: instead of a tuple use a structure (as also suggested by the Core Guidelines).

Let's see how you will use a separate `struct` to resolve this issue in the next lesson.