Removing auto_ptr

The lesson introduces how the pointers have been revamped for the better.

C++98 added `auto_ptr` as a way to support basic RAII features for **raw pointers**. However, due to the lack of move semantics in the language, this smart pointer could be easily misused and cause runtime errors.

Here's an example where `auto_ptr` might cause a crash:

```
#include <iostream>
#include <memory>

void doSomething(std::auto_ptr<int> myPtr)
{
  *myPtr = 11;
  std::cout << *myPtr;
}
void AutoPtrTest() {
  std::auto_ptr<int> myTest(new int(10));
  std::cout << *myTest;
  doSomething(myTest);
  *myTest = 12;
  std::cout << *myTest;
}

int main(){
  AutoPtrTest();
}
```

The code will produce an error!

`doSomething()` takes `auto_ptr` by value, but since it's not a shared pointer, it gets the unique ownership of the managed object. Later, when the function is completed, the copy of the pointer goes out of scope, and the object is deleted.

In `AutoPtrTest()` when `doSomething()` is finished the pointer is already cleaned up, and you'll get undefined behaviour when calling `*myTest = 12`.

In C++11 we got smart pointers: `unique_ptr`, `shared_ptr` and `weak_ptr`. With the move semantics, the language could finally support proper unique resource transfers. Also, new smart pointers can be stored in standard containers, which was not possible with `auto_ptr`. You should replace

`auto_ptr` with `unique_ptr` as it's the direct and the best equivalent for `auto_ptr`.

We can rewrite the example so it uses `unique_ptr`:

```
void doSomething(std::unique_ptr<int> myPtr) {
    *myPtr = 11;
}

void AutoPtrTest() {
    auto myTest = std::make_unique<int>(10);
    doSomething(myTest); // won't compile!
    *myTest = 12; // use after move ??
}
```

Now, the code won't compile as you need to move `unique_ptr` into `doSomething()`. Since the move is explicit it requires to possibly rethink the solution. For example, in this case, maybe `doSomething()` doesn't need the ownership of the pointer? Perhaps it's better to pass a raw pointer, without the ownership?

Alternatively, you can use `shared_ptr` and then, the pointer won't be deleted after `doSomething()` is finished, as `shared_ptr` uses reference counting.

New smart pointers are much more powerful and safer than `auto_ptr`, so it has been deprecated since C++11. Compilers should report a warning:

```
    warning: 'template<class> class std::auto_ptr' is deprecated
```

Now, when you compile with a conformant C++17 compiler, you'll get an error.

Here's the error from MSVC 2017 when using `/std:c++latest`:

```
    error C2039: 'auto_ptr': is not a member of 'std'
```

If you need help with the conversion from `auto_ptr` to `unique_ptr` you can check Clang Tidy, as it provides auto conversion: Clang Tidy: modernize-replace-auto-ptr.

> *Extra Info:* The change was proposed in: N4190.

Next up on our list of deprecated utilities, we have `std::random_shuffle`.