Modern "range-based" for loops

In recent versions of C++, there is a version of the **for** loop that automatically iterates over all of the things in a container. This is very useful when used with a standard library container, because you don't have to worry about trying to access memory outside of a safe range, for example: the loop will automatically begin and end in the right place.

for (*temporary variable declaration* : *container*) { *loop body* }

There's an important detail about the temporary variable. If you declare an ordinary temporary variable in the loop, it just gets a copy of the current loop item by value. Changes you make to that temporary copy won't affect the actual container!

```
#include <iostream>
#include <vector>
int main() {

  // In the standard library, a std::vector is an array with automatic size.
  // Let's make a vector of ints and loop over the contents.
  // The syntax for std::vector<> is discussed further in the lecture on
    template types.

  std::vector<int> int_list;
  int_list.push_back(1);
  int_list.push_back(2);
  int_list.push_back(3);

  // Automatically loop over each item, one at a time:
  for (int x : int_list) {
    // This version of the loop makes a temporary copy of each
    // list item by value. Since x is a temporary copy,
    // any changes to x do not modify the actual container.
    x = 99;
  }

  for (int x : int_list) {
    std::cout << "This item has value: " << x << std::endl;
  }

  std::cout << "If that worked correctly, you never saw 99!" << std::endl;

  return 0;
}
```

Run

Reset

Expected output:

```
This item has value: 1
This item has value: 2
This item has value: 3
If that worked correctly, you never saw 99!
```