Functions in C++

remember that learning another language is just really all about syntax
- yes, there *may* be a few other tidbits to get through
- but once you know how to "code," learning another language is pretty easy!
- and you better get used to this because our field changes all the time
- you *will* have to learn new languages, concepts, ideas, etc on your own in the "real world"

automatic variables
- memory is allocated at block entry and deallocated at block exit
  - e.g. a variable local to a function

static variables
- ***HANDOUT*** static
- memory is allocated at program start and remains allocated throughout program execution

const
- ***HANDOUT*** const
- any parameter declared as const in a function parameter list cannot be changed in the function
- it also cannot be on the left-side of assignment statement (makes sense, yes?)

default parameters
- ***HANDOUT*** default_params
- usually the number of actual and formal parameters must be the same
  - e.g. void f(int, int); --> f(x, y);
- this is relaxed when we define default parameters
- default values are used when actual parameters are not specified
- but there are rules we have to follow
  - all default parameters must be on the right side of the parameter list
  - default values can only be constants, globals, or function calls
  - values of default parameters can be specified when calling the function
  - if the value of a default parameter isn't specified in a function call, omit all following parameters
  - cannot assign a constant value as a default value to a reference parameter
  - e.g.

```
string myFunction(int, float, char = 'X', bool = false);
myFunction(i, f); // legal
myFunction(i, f, 'J'); // legal
myFunction(2, 7, 'P', false); // legal
myFunction(i, 'D', true); // illegal
```

overloading
- ***HANDOUT*** overloading
- motivation: print an array of integers, floats, etc
- functions can have the same name
  - in this case, they must have completely different parameters
- we consider the function name and its parameters the "signature" of the function
  - two functions are different if they either have different names or different parameters (types)
- some problems
  - e.g. void myFunc(int); and int myFunc(int); → which one is used?
    - we can't do this...obviously!
- e.g.
  - print an array of "stuff"
  - determine the larger of two values (int, float, etc)
    - int larger(int, int); → int i = larger(3, 4);
    - float larger(float, float); → float f = larger(1.1, 2.2);

templates
- ***HANDOUT*** templates
- motivation: e.g. int larger(int, int); float larger(float, float);
  - these accomplish a similar thing (find the larger of two ints or two floats)
  - is there a way to write only one function that can handle many different data types?
  - function overloading allows us to do similar things for different parameters (types)
  - but there is still too much redundancy and too little code reuse
  - so what if we could write a generic function that could handle many types to do the same thing?
- e.g. print an array of "stuff"
- e.g. find the largest item in an array of "stuff"