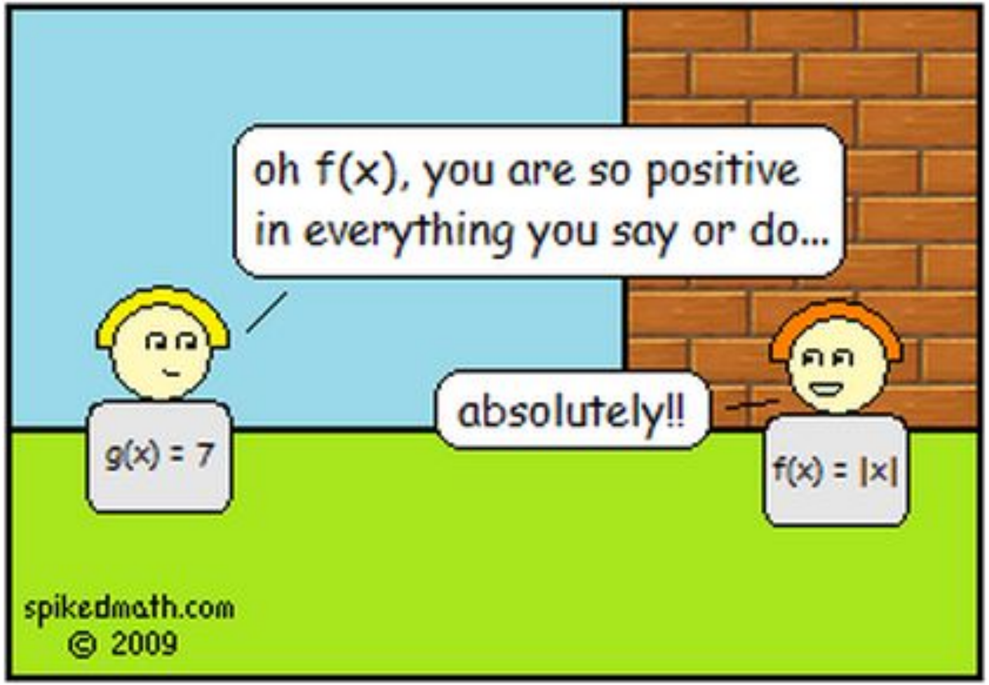
Lecture 7: Return Values, and Not Return Values



Announcements and reminders

Submissions:

- HW 3 (functions) -- due Saturday at 6 PM

Course reading to stay on track:

- 5.5 - 5.8 today
- 5.9 before Friday

Practicum 1

± 30 minutes, still finalizing...

- 5:30 - 7 PM, Wednesday 20 Feb
- Let us know (Piazza) about conflicts.
  Include some verification (covering all our tails)

Last time on *Starting Computing…*

We learned what a function is!

We learned how to implement a function!

We learned how to pass parameters into a function and send return values back out!

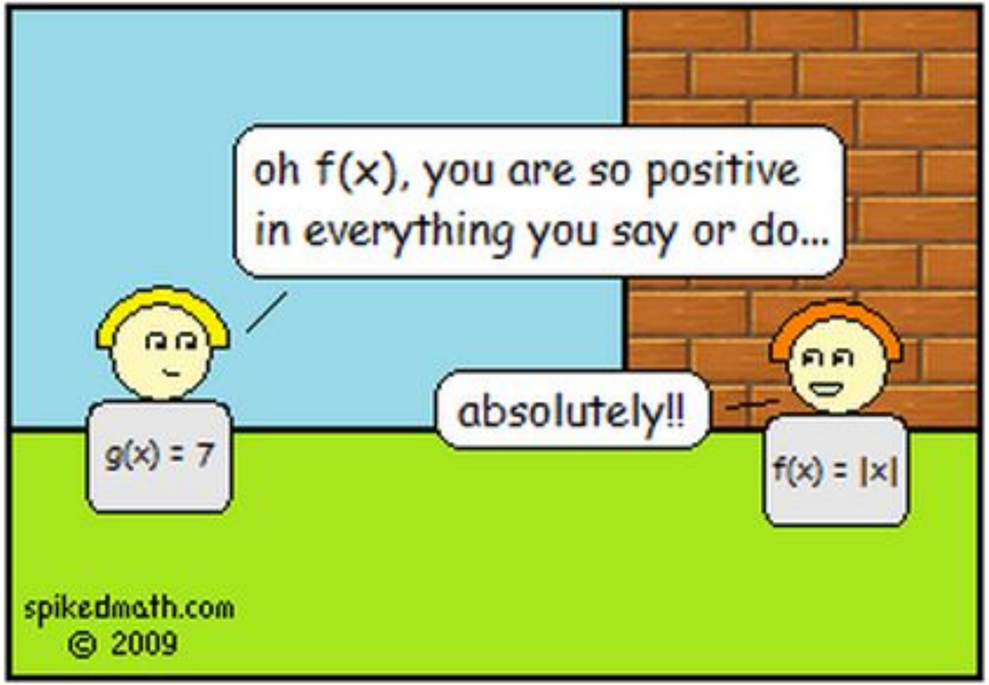
We learned a little bit about the **scope** of variables!

- Much more on this later!

Chapter 5: Functions

Chapter topics:

- Functions as black boxes
- Implementing functions
- Parameter passing
- Return values
- Not return values
- Reusable functions
- Stepwise refinement
- Variable scope and globals



Return Values

The return statement ends the execution of a function. This behavior can be used to handle unusual cases.

Example: What should we do if the side length of a cube is negative?
→ We can return a zero and not do any calculations:

```
double cube_volume(double side_length)
{




}
```

Return Values

The return statement ends the execution of a function. This behavior can be used to handle unusual cases.

Example: What should we do if the side length of a cube is negative?
→ We can return a zero and not do any calculations:

```
double cube_volume(double side_length)
{
    if (side_length < 0)
       return 0;
    double volume = side_length * side_length * side_length;
    return volume;
}
```

otherwise cont.

Fun fact: Nothing within that function is executed after the `return` statement
- execution *returns* to `main()`

Return Values

The return statement ends the execution of a function. This behavior can be used to handle unusual cases, and can be **a useful shortcut** to write more concise code

Example: Instead of saving the return value in a variable and returning the variable, we can eliminate the variable and return a more complex expression:

```
double cube_volume(double side_length)
{
    if (side_length < 0)
        return 0;
    return side_length * side_length * side_length;
}
```

Common Error -- Missing Return Value

A function **must always return something**

Example: What could go wrong with the function below?

```
double cube_volume(double side_length)
{
    if (side_length >= 0)
    {
        return side_length * side_length * side_length;
    }
}
```

if side_length < 0, then we don't have a return value!

Function Declarations (Prototype Statements)

It is a compile-time error to call a function that the compiler does not know

- Just like using an undefined variable

So define all functions before they are first used

- But sometimes that is not possible -- what if 2 functions call one another?

Function Declarations (Prototype Statements)

It is a compile-time error to call a function that the compiler does not know

- Just like using an undefined variable

So define all functions before they are first used

- But sometimes that is not possible -- what if 2 functions call one another?

Can include a “prototype” definition for each function at the top of the program, then the complete function code goes after `main() { ... }`

- A prototype statement is just the function header line followed by a semicolon:

```
double cube_volume(double side_length);
```

- The variable names are optional - what’s important is to declare the types of inputs and outputs for a function. So this would work equally well:

```
double cube_volume(double);
```

Function Declarations (Prototype Statements)

Example: Refactor the `cube_volume` testing code to declare a function prototype for `cube_volume`

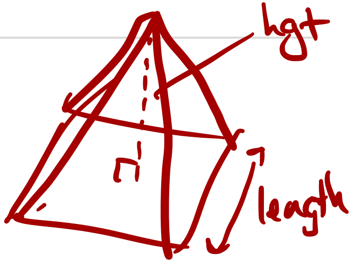
Function Declarations (Prototype Statements)

Common error: No function declared before encountering function call in main()

```
int main()
{
   volume1 = cube_volume(2.0);
   return 0;
}

double cube_volume(double side-length)
{
   return side_length * side_length * side_length;
}
```

Steps to Implementing a Function



1) Describe what the function should do

   Example: Compute the volume of a pyramid whose base is a square

2) Determine the function's inputs

   Example: inputs: base length & height

3) Determine the types of the parameters and return value

   Example: ↳ doubles ←

4) Write pseudocode for obtaining the desired result

   Example: $\text{volume} = \frac{1}{3} \times \text{height} \times (\text{length})^2$

5) Implement the function body

   Example:

6) Test your function

   Example:

Steps to Implementing a Function

1) Describe what the function should do

   Example: Compute the volume of a pyramid whose base is a square

2) Determine the function's inputs

   Example: height, base side length

3) Determine the types of the parameters and return value

   Example:

   ```
   double pyramid_volume(double height, double base_length)
   ```

4) Write pseudocode for obtaining the desired result

   Example:

   ```
   volume = ⅓ * height * (base length)$^2$
   ```

5) Implement the function body

   Example:

   ```
   double base_area = base_length * base_length;
   return height * base_area / 3;
   ```

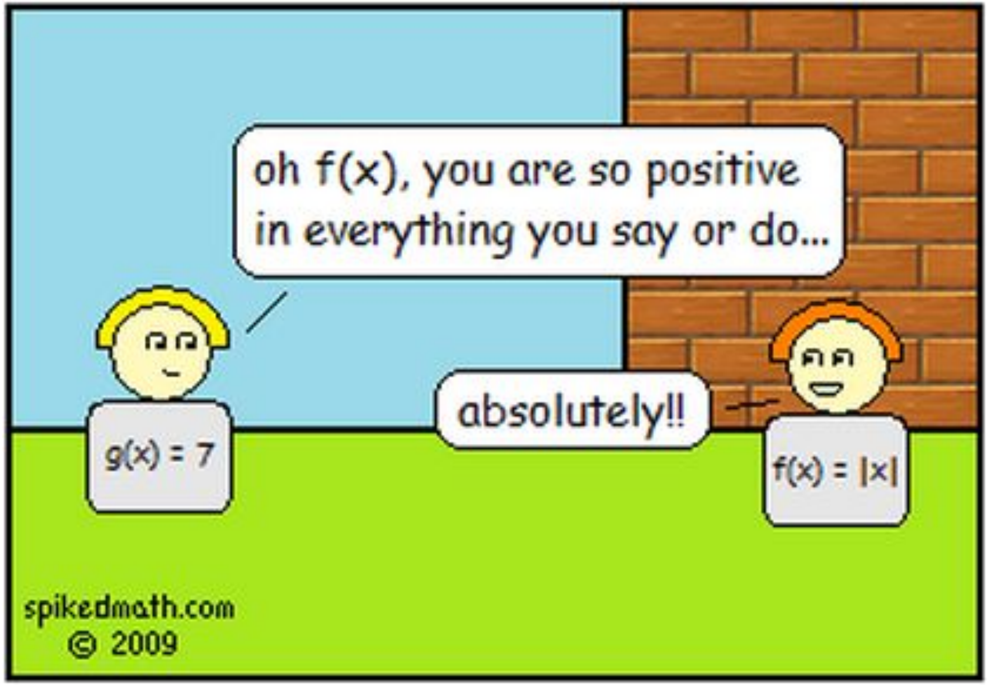
6) Test your function

   Example: Write a `main()` function to call it multiple times, including **boundary cases**

Chapter 5: Functions

Chapter topics:

- Functions as black boxes
- Implementing functions
- Parameter passing
- Return values
- Not return values
- Reusable functions
- Stepwise refinement
- Variable scope and globals



Functions without return values

Consider the task of writing/printing a string with the following format around it

Any string could be used

For example, the string “Hello” would produce:

```
--------
!Hello!
--------
```

Quick aside: Use the <string> header file to include the string data type and work with string variables (like, words, and phrases):

```
#include <string>
```

Functions without return values -- the void type

```
--------
!Hello!
--------
```

Definition: This kind of function is called a **void function**

- void is a type, just like int or double
- Use a return type of void to indicate that a function does not return a value
- void functions are used to simply perform a sequence of instructions, but not return any particular values to the caller

type for strings! characters/ words/letters

Example: `void box_string(string str)`

Functions without return values -- the void type

Example:

```
void box_string(string str)
{
    int n = str.length();
    for (int i = 0; i < n + 2; i++) { cout << “-”; }
    cout << endl;
    cout << “!” << str << “!” << endl;
    for (int i = 0; i < n + 2; i++) { cout << “-”; }
    cout << endl;
}
```

NO RETURN STATEMENT

2

we'll talk about these later!

```
-------
!Hello!
-------
```

Note that this function doesn’t compute any value.

It performs some actions and then returns to the caller **without returning a value**

→ There is no return statement

Calling void functions

A void function has no return value, so we cannot call it with assignment like this:

```
result = box_string("Hello");      // Error: box_string does not return a result
```

↑

void box_string(...)

Instead, we call it like this, without assignment:

```
box_string("Hello");
```

What just happened…?

We learned how to pass parameters into a function and send return values back out!

We learned about functions to perform sets of tasks without return values!

→ void functions