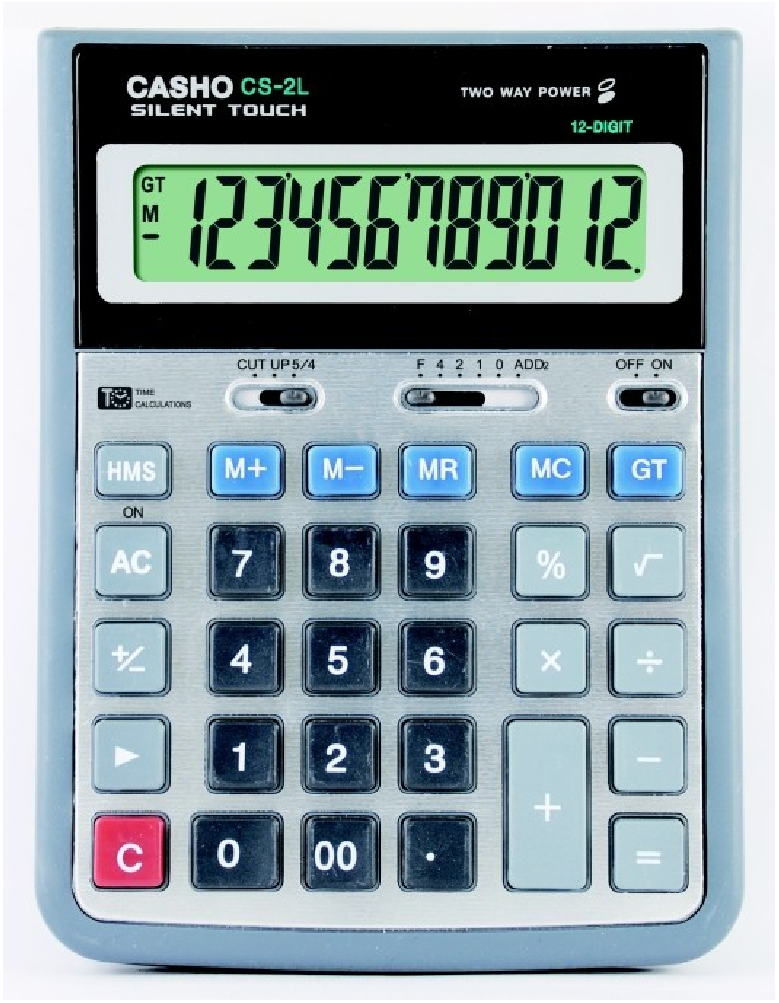
C++ and openFrameworks

Week 3

Instructor: Nina Freeman

What is a function?

- Functions are like the buttons on a calculator. They execute an action.
- What are some of the functions of a calculator?
- What are some openFrameworks functions that we have used?

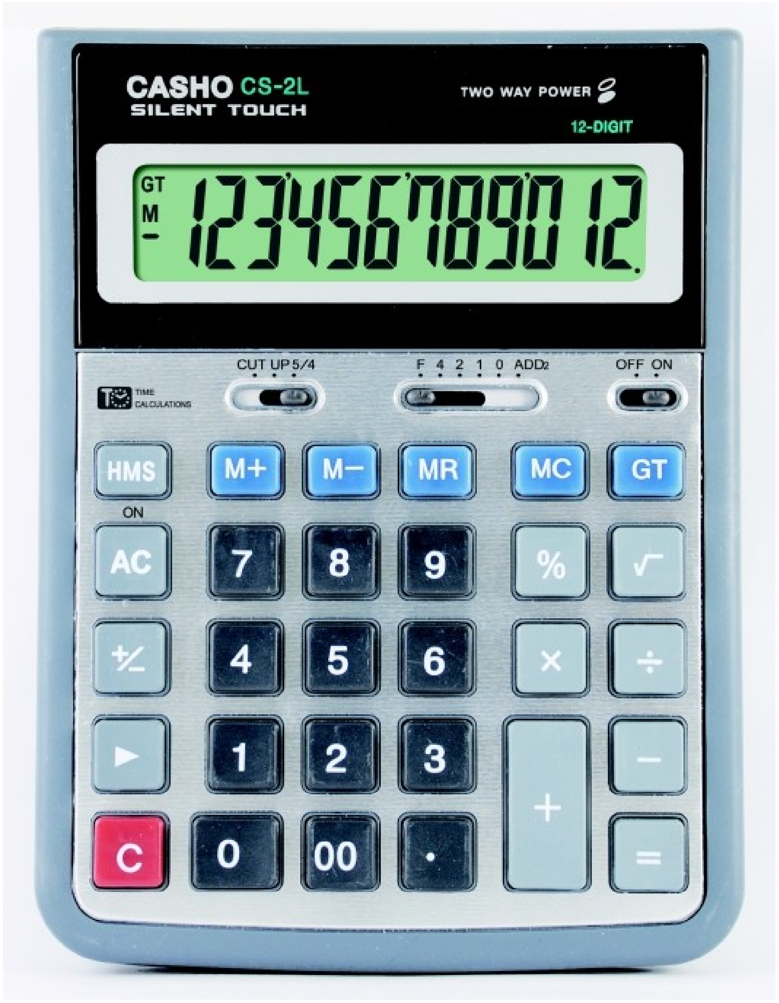


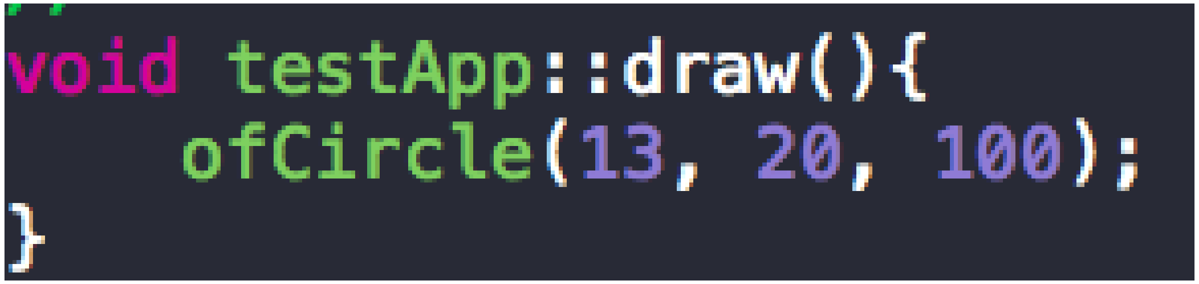
What is a function?

The actual code that defines what the function does lives in a separate place than where you actually use it.

You call it by name (kind of like a button!) in your program where you want to use it.

e.g. ofCircle()





```
void testApp::draw(){
    ofCircle(13, 20, 100);
}
```

When the function is called, the program jumps to the function definition and executes it.

Once the function is done running, the program continues from where it left off.

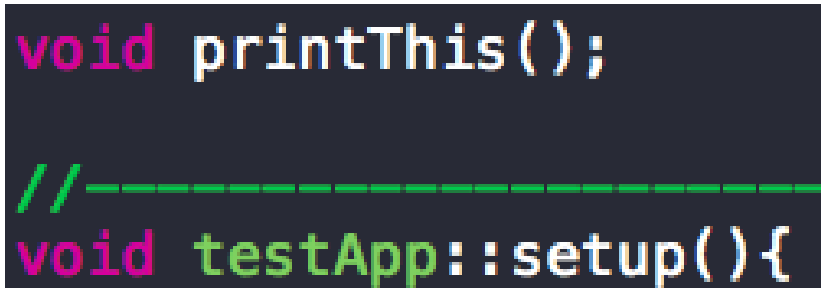
Functions are a form of *abstraction.*

Abstraction means that you don't need to understand how everything works "under the hood" in order to interact with a system.

Building a function

Declare your function in the same place you declare variables--outside of any other functions or brackets using a prototype*.
*It's actually optional, but definitely recommended.

Prototype →



```
void printThis();

//----------------------
void testApp::setup(){
```

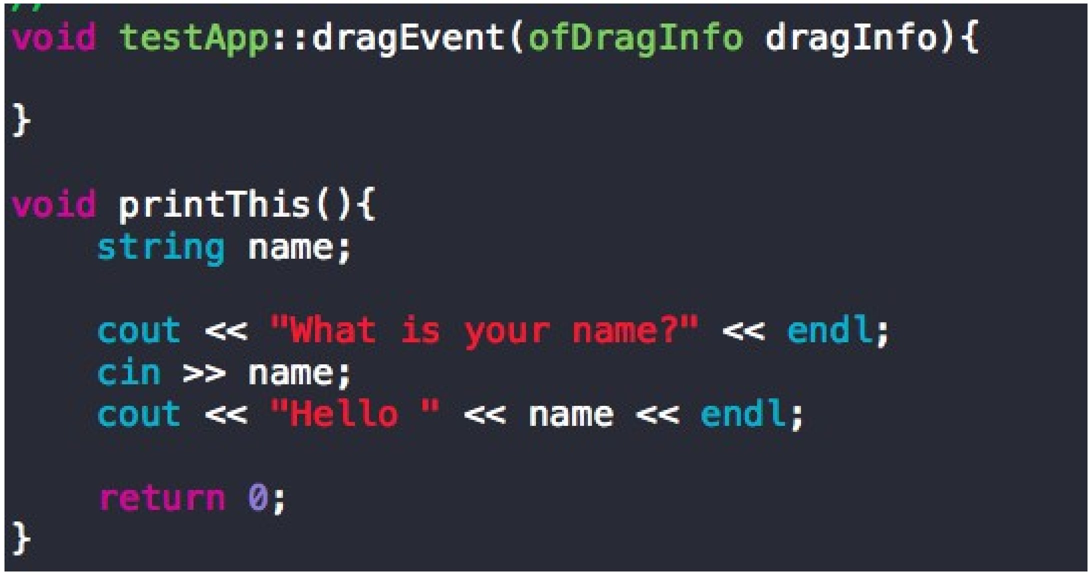
Syntax:
returnType functionName(parameters);

Define your function

- What does your function do?

Syntax:

```
returnType functionName(parameterType parameterName){
    //Block of code to execute.
    //return statement (it's like a stop sign, see later slides)
}
```

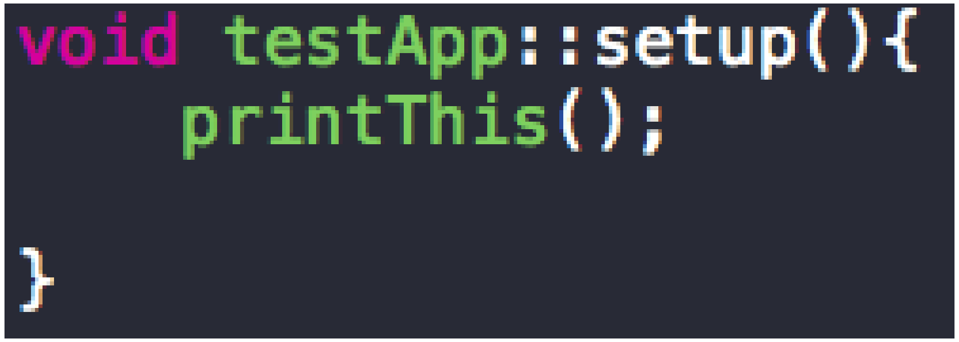


```
void testApp::dragEvent(ofDragInfo dragInfo){

}

void printThis(){
    string name;

    cout << "What is your name?" << endl;
    cin >> name;
    cout << "Hello " << name << endl;

    return 0;
}
```

Definition

Where do you want to use your function?

I want the user to enter their name at the very start of my program.



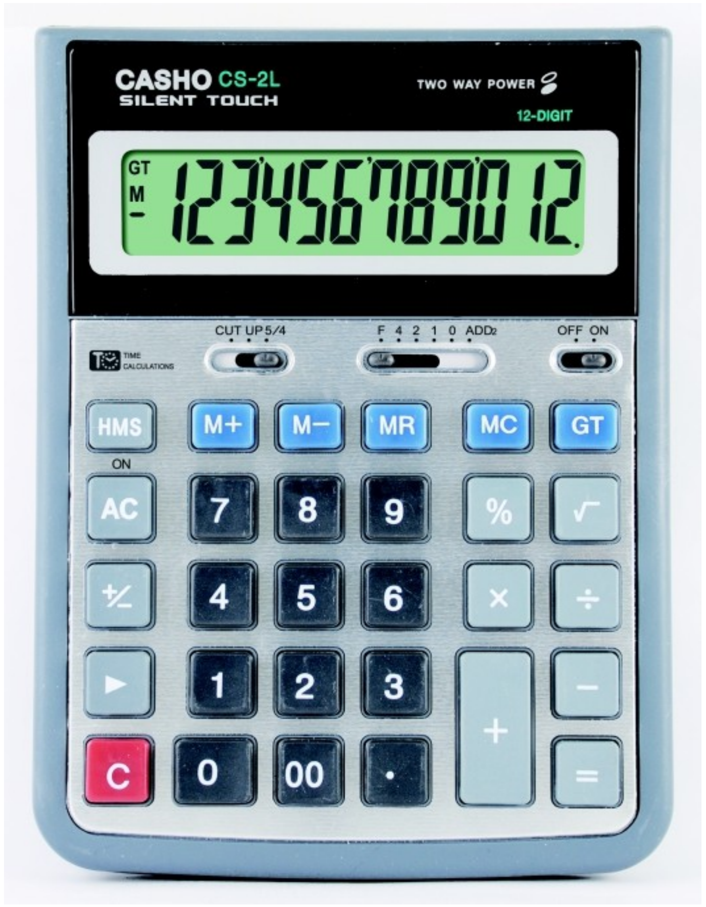
```
void testApp::setup(){
    printThis();

}
```

- When do you think functions will be useful to you?

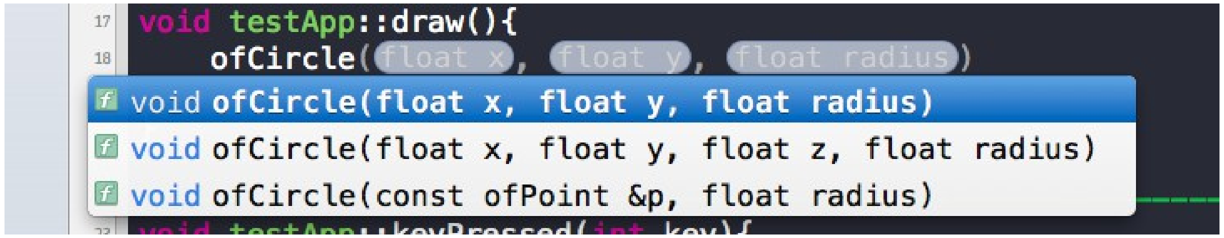
- Can you think of any functions you might want to write?

- What is the simplest way you can think of to describe what functions do?

Why should I use functions? Are they versatile? Yes! Functions are just like verbs!



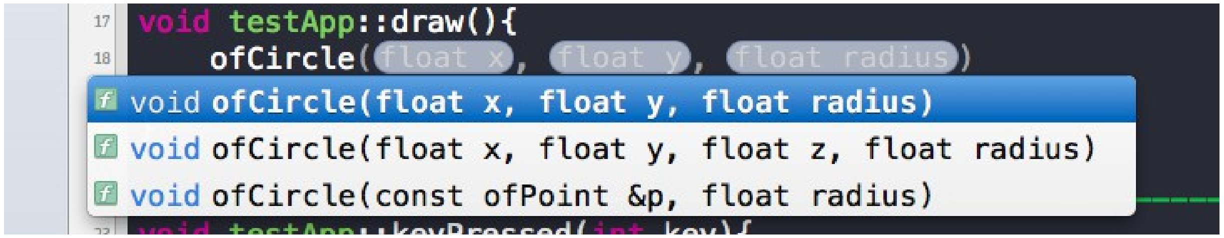
Parameters



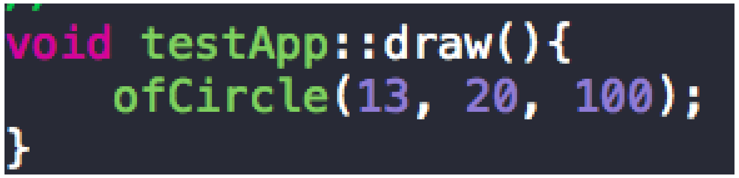
```
void testApp::draw(){
    ofCircle(float x, float y, float radius)
void ofCircle(float x, float y, float radius)
void ofCircle(float x, float y, float z, float radius)
void ofCircle(const ofPoint &p, float radius)
```

- Functions are able to use values from the outside by defining parameters.
- Parameters live in the parenthesis following a function's name.
- You need to specify the data type of these parameters in your function definition and declaration.

Parameters



```
void testApp::draw(){
    ofCircle(float x, float y, float radius)
void ofCircle(float x, float y, float radius)
void ofCircle(float x, float y, float z, float radius)
void ofCircle(const ofPoint &p, float radius)
```



```
void testApp::draw(){
    ofCircle(13, 20, 100);
}
```

You pass arguments to functions according to the parameter's data type.

The function then makes a **copy** of that value to use when it runs. This is called *passing by value.*

Newspaper content is passed to a printer, just like a function parameter!

So…what happens to all of those parameter variables that were copied and passed by value?

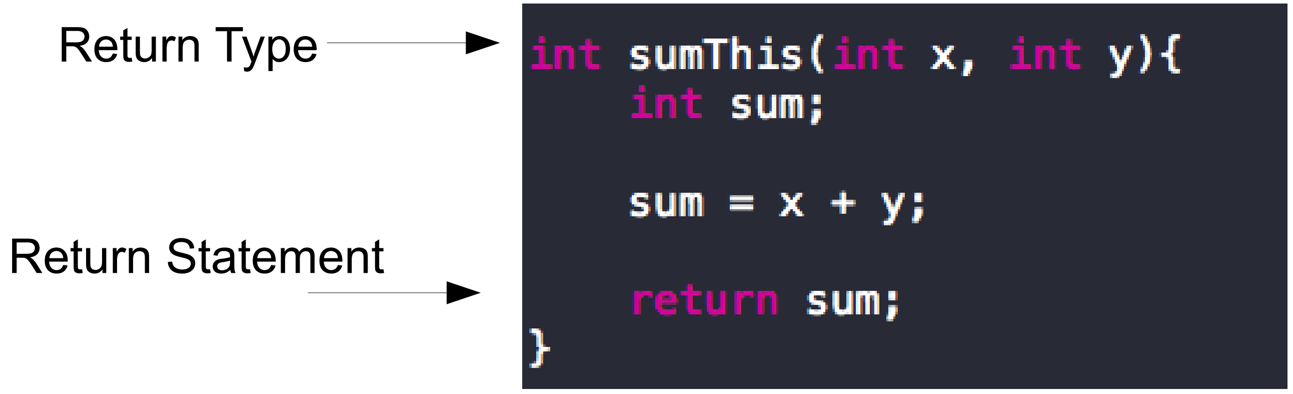
What lives in a function, dies in a function.

Don’t worry—those variables aren’t just lying all over the place.

When a function ends, all of the variables declared within it, including arguments passed by value, are cleared from memory.

Return Values

- The *return* statement allows a function to send data back when it is called.
- You need to specify your return type before the function name in both the prototype and function header.
- Of course, the actual return value's data type must also match the functions return type.



```
int sumThis(int x, int y){
    int sum;

    sum = x + y;

    return sum;
}
```

Return Values

1. The return statement is a stop sign for the function.

2. You don’t have to return any data, but you do need to have a return statement. Return type is void if the function doesn't return anything.

3. To stop a function with a void return type, you just use:
```
 return;
```

Can you think of any oF functions that we have used to return something other than void? A number, perhaps?

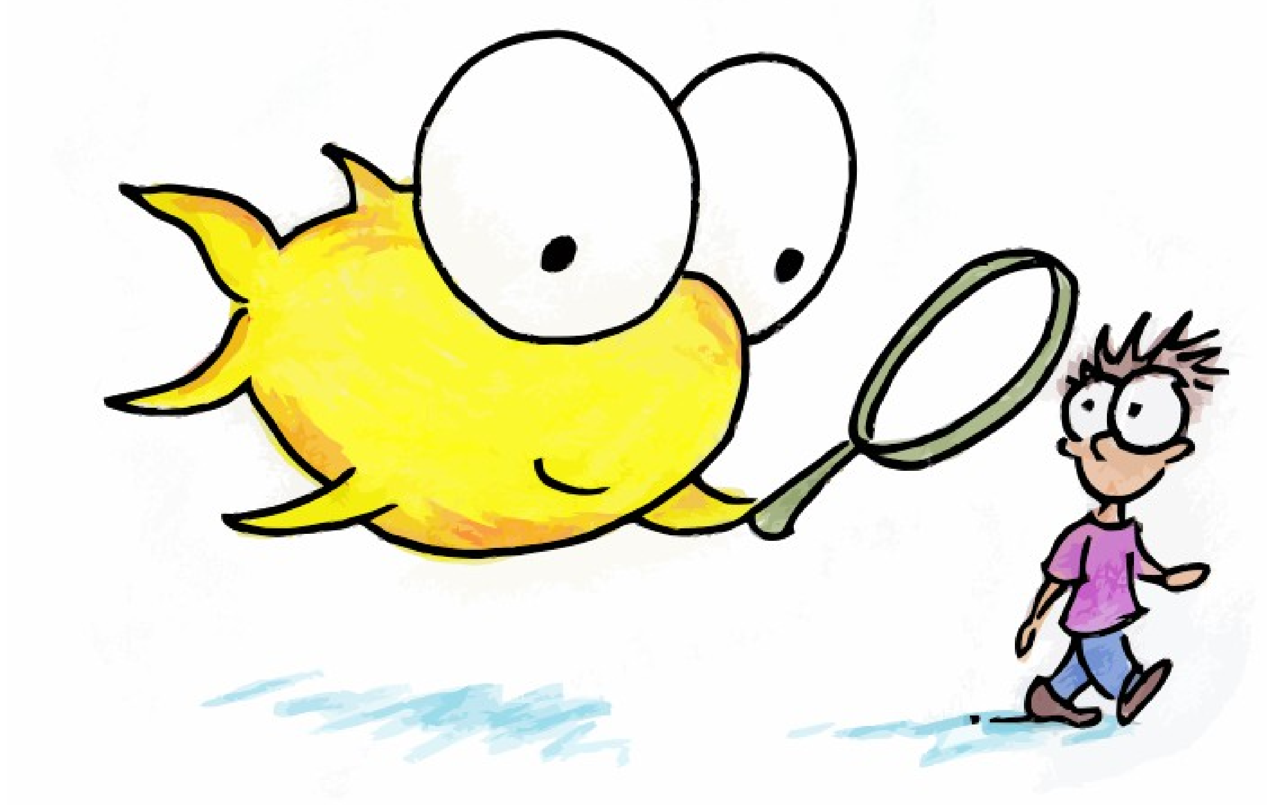
Can you think of any oF functions that we have used to return something other than void? A number, perhaps?

ofDist()

ofGetWidth()

ofGetHeight()

So what's all this fuss about parameters and return values—why can't my program just see ALL of my code?



Well… do you remember abstraction? You don't need to know what makes a laptop tick, because you can interact with it via UI, a keyboard, a screen, etc.

Abstraction makes it easier to use complex systems. Encapsulation hides the details under the hood.

- *Encapsulation* dictates that data is only visible within the *scope* (aka any set of curly brackets) that it lives in.

- Think about encapsulation in terms of bundling—each object is a separate bundle of instructions that are wrapped up and hidden from the user.

- Functions need return values and parameters so that the user can interact with a program without having to worry about each line of code required to execute a task.

Review!

What's an if statement?

How might you track a game state?

What's an array?

When is a for loop useful?

What are some openFrameworks functions?