# TutorTube: Functions in C++

Spring 2021

## Introduction

Hello and welcome to TutorTube, where the Learning Center's lead tutors help you understand challenging course concepts with easy-to-understand videos. My name is Matt Curtin, lead tutor for computer science. In today's video, we will be exploring using functions in C++. Let's get started.

## Agenda

In today's video we will be discussing what is a function, how to make a function, coding a function example, and then at the end we will do a quick review and wrapping up.

## Why Functions?

So, why do we use functions? Have you ever been coding and realize that your code is kind of becoming spaghetti code, where everything is kind of a mess, you don't really know what's going on, and your code is lacking organization? Well, functions allow us to accomplish three things: reusability, so we can use a function to reuse the same code over and over again without having to rewrite it. We can accomplish simplicity. A function allows us to make our program look nicer so that we can read it easier, and that's the third bullet point is readability. We can use functions to simplify parts of our code so that we understand what our program is doing.

## How to Think About Functions

An analogy that helped me understand functions was to think of them as errands. Functions are just errands in your code. Your code tells a function to either do something, get something, or change something. You can kind of think of it as a parent telling a child what to do. Go do this errand and then the child comes back. Functions are very, very similar as you'll see.

## Function Flow Line

Here's what happens when you use functions in your programs. The first thing is that your code runs until it needs something from your function. So just like normal it goes and goes and goes but wait! What happens when we need bananas in our main method? Well, we need to go get bananas. So, what do we do? The next part: the main method pauses what it's doing, it stops in its tracks, and it goes over to your function and starts buying bananas. Your function runs and completes, so if our buy bananas function is going, it's going to run and complete. After it completes, we're going to jump all back to where we left off in the main method. After we jump back, we bring our bananas with us. And now we continue our program and we have the bananas that we got from our **getBanana()** or **buyBanana()** function. This is a simplistic example but it just kind of shows the flow chart of how a function interacts with your program.

## Functions That Do Things

Here are a few applications of functions. For functions that do things, you can have a function print something out, initialize something - so you know setting a value to variables, you can have functions that prompt the user for information, or open files, or perform algorithms, and so on and so on.

## Functions That Get Something

Functions that get something: these are functions that give you a value in return, so they bring you back something. You can either get the sum of two numbers, get how many bananas are in an array, get the name of a classroom, or you know get how many vowels are in word. These functions can bring you back something that you can work with.

## Functions That Change Something

Functions that change things: if you want to change a variable or change the state of something. You can have a function that multiplies things by 2, a function that makes a word all lower-case letters, remove all spaces from a word, take this tax on a sale, or take out all duplicates in an array. So, your function can change the value or state of things as well.

## The Parts of a Function

We are now going to look at the different parts of a function. A function is made up of three things: the first one being the return type, the second being the name, and the third being the parameters or arguments. If we look at a function such as **int main()**, our return type would be **int**, our name would be main, and we would have no parameters or arguments. Another example would be **bool isPositive(int x)**. Our return type would be bool our name would be is positive and our parameters and arguments would be one variable that is an integer **x**.

## Return Type

We're now going to look at what return type is specifically. Return type is essentially a promise. For example, if we're looking at **int main()**, **main()** is promising it will return an **int**. So, in terms of our errand, you're promising your parent that you're going to come back with the bananas that you were tasked with buying. So, if a function had a value, as if you were to set something to the function, its type would be the return type. A return type would be asking "what is my function going to give me?" So, for **bool isPositive(int x)**, our return type is **bool**. We put the return type in front of the function when we make it, so you can only have one return type and you can only return one thing. In this case, **isPositive()** is going to return a **bool**.

## Name

Now we're going to look at name, and naming our functions is really important too. How we name functions can tell us a lot, whether we're working by ourselves or working in a group. So in terms of our function **getBanana()**, the name implies it'll get us a banana. It's very straightforward: get banana will get you a banana. **calculateTax()** implies that this function will calculate something involving tax. My philosophy is that the more specific your function name the better. That way everything is clear when you see it written in your code. Another thing is **setBananaWeight()**. This name implies it will set the value of a bananas weight. Again, it's very specific. You know exactly what it's doing. It's setting the weight of a banana. When you see this written in your code you know what it's talking about.

## Verb Noun

A lot of times the pattern that I like to stick to is verb-noun for functions, so what is the function doing and was it doing it too. Some examples of verbs can be set, get, calculate, perform, remove, add, schedule, or parse. And some examples of nouns could be number, time, weight, characters, punctuation, size, date, bananas. One function that we could do would be called **addBanana()** or **removeDate()** or **calculateTime()**, **parseNumber()**. Basically, the function name needs to describe what it's doing and what it's doing it to. That way when you're writing, when you see it written in your code, you know what it's doing.

## Arguments

The next thing that we're going to look at is arguments. So, this is like if your parent gives you five dollars to go get you bananas. The arguments are the five dollars and maybe another argument would be the directions to the store. The parent would be the main method; it's a thing calling the function. Getting the bananas would be the function, so that is the errand. The five dollars is the inputs or the arguments, so like we said before five dollars is what the parent gives you and in return, you're going to give it back bananas. So something that this function would look like would be **bananas getBananas(int dollarAmount)**. Passing in a certain dollar amount you're going to get a certain amount of bananas.

## Arguments Continued

Arguments are often also called parameters, and there are two modes that you can call parameters: by pass by value which is you're just passing a copy of the variable to the function, and pass by reference is you passing the address of a variable.

## Pass by Value vs. Pass by Reference

Here are some key differences between pass by value and pass by reference. To pass a variable in by value, you would just mark it **int x** like if you were declaring it. This parameter is only a copy, meaning that changes made to this variable inside of the function only affect the variable while you are inside the function. Changes made to the variable inside of the function do not affect the variable outside of the function. To pass a variable in by reference you would have to add this ampersand next to the return type. This denotes that you are passing in a location in memory, or you're just passing in an address. What this does is that changes made to this variable inside of your function will affect the variable in all parts of your program.

## Making a Function

When making a function, I like to go through this process. Let's say that we are making a function that calculates the area of a rectangle. The first thing that we would need to do is to identify our inputs. What do we need to know to calculate the area? Well we need to know the length and the width. The next thing we would need to do is to identify our outputs. This gives us our return type. So, we need to know the area of a rectangle, so we're going to get an **int**. The next thing we need to do is to make an informative name, so if we are making a function to calculate area let's just name it **calculateArea()**. Another name you could do would be **calculateAreaOfRectangle()** if you want to be really really specific. Here is a chart that I can do to say "here's my inputs, here's my outputs, and the name of my function," and then using that I can figure out the return type, the name of my function, and the parameters that I need to list, and then I can define the function with the behavior that I need.

## Creating a Function Example

Here I have a program where it would be a nice application of using a function. Here I want to calculate the area of a circle given a radius. I'm going to assume that in this program I want to use this calculation more than once or multiple times and I don't want to repeat code. Here's a nice place to use a function. Let's identify what we want our function to be. So, what are our inputs? Well, we're going to input the radius as a **float**, so "**float radius.**" What are our outputs? What do we want to be returned to us? Well, we want to it to return a **float** that represents the area of a circle. And what is the name of our function going to be? How about **calculateAreaOfCircle()**? Be really specific with it. So, given this information we need to figure out how to declare our function. Let's start with the return type. The return type is going to be whatever our output is. Here we decided that we want to return a **float**, so we type "**float**." Next, we need to type our name **calculateAreaOfCircle()**. Next, we need to place our parameters between the parentheses, so our parameters are just our inputs. Here we decided that we want to input a **float** of name **radius**. We can name this whatever we want! In fact, let's name it **r**, so that we can fit with the standard of pi * r^2. Here, I'm going to declare a variable for **area**, and I'm going to return **area** at the end, and between declaring **area** and returning it I want to calculate what **area** is going to be. Well, **area** is going to be equal to pi * r^2, so let's approximate pi - 3.14 times **r** times **r**. This is an approximation of the area of a circle pi r squared or pi times r times r.

What this function is doing is that if I give the function a radius, it is going to calculate an area and give me back that value. I'm going to give it a **float** and it's going to give me back a **float**. Let's call this function here, and inside our program let's set **area = calculateAreaOfCircle(radius)** and let's pass in our radius.

Nice! Let's compile and see if it runs. Area is 140.955. Perfect! That's what we want.

To reiterate, what this program is doing is that given a radius, to calculate area we call this function **calculateAreaOfCircle()** and we pass it in the value of **radius**. The program then jumps up to this function that declares its own **area**, sets it equal to 3.14 times the value that we passed in times the value that we passed in, and it's going to return the **area** that we calculated. So, we jump back down here, and we assign **area** equal to whatever was returned. Now we can **cout** it and deal with it in our main function and do whatever we want with it.

## Pass by Value vs. Pass by Reference Example

We are now going to be going over an example between pass by value and pass by reference. Right here we have a program. Here's a function that takes in a number and it doubles it and then outputs that number. Inside our main method, we initialize the variable **x = 20**, we print what that value is before we call the function, we then call the function, and then we print what the value is after the function. Let's run the program and see what happens. So, we set **x** to 20 here, we call the function, inside of the function the number we pass in is doubled, but then look at this. After the function, **x** is still 20. So, what's going on here?

If we look, this is a pass by value function, meaning that when we call it down here, all we're doing is copying this - the value of whatever is inside of **x** - into **number**. Here, all we're doing is copying the number 20 into the variable

**number**, so any changes made to **number** is just a copy of **x**. It doesn't affect **x** at all. That's why when we exit the function and come back down here to after function, **x** is still 20 when **number** was 40. How do we change this? How do we change it so that whenever we pass in **x**, **x** is doubled so then after the function it's 40? What we can do is pass in number by reference.

All we have to do is put an ampersand right here next to the type. What this does is that it marks it as a pass by reference instead of passing in a copy of **x**. Down here what we're doing is that we're passing in the address of **x**, so the address of **x** is copied to **number**. That means that anytime **number** is changed, **x** is changed as well because it's located in the same address in memory.

Let's compile this and see what happens. Now you can see that before we call the function **x** is 20, inside of the function when we change **x** to **number** the **number** is 40, and then after the function when we leave the function **x** is 40 like it was inside the function. That is the difference between pass by value and pass by reference.

## Outro

That's going to wrap up our video today. It's not everything about functions, but it should get you started and give you an idea of what you can do. Thank you for watching TutorTube. I hope you enjoyed this video. Please subscribe to our channel for more exciting videos. Check out the links in the description below for more information about the learning center and follow us on social media. See ya!

## Sample Code: Area of Circle

```
#include <iostream>

using namespace std;


//Inputs: float radius

//Output: float that represents the area of a circle

//Name:   calculateAreaOfCircle

float calculateAreaOfCircle(float r){
	float area;
	area = 3.14 * r * r;
	return area;
}

int main(){
	//Info about circle
	float radius = 6.7;

	//I want a function to calculate area of a circle
	float area;

	//INSERT FUNCTION CALL HERE
	area = calculateAreaOfCircle(radius);

	//Display area
	cout << "Area: " << area << endl;

	return 0;
}
```

## Sample Code: Pass by Value vs Pass by Reference

```
#include <iostream>
using namespace std;

//This function doubles the number that is passed in
void doubleNumber(int& number){
	number = number * 2;
```

```
    cout << "Inside function: " << number << endl;
}

int main(){
    //Init x
    int x = 20;
    cout << "Before function: " << x << endl;

    //Call function to double x
    doubleNumber(x);

    cout << "After function: " << x << endl;

}
```