**How to Organize Your Code with Functions**

Functions are a fundamental building block of programming. They help us organize our code into manageable and reusable pieces.

Let's explore the basics of functions by way of a burger joint.

A burger joint may seem like an odd place to learn about code organization...but let's see where this goes. And you can view the associated video here.

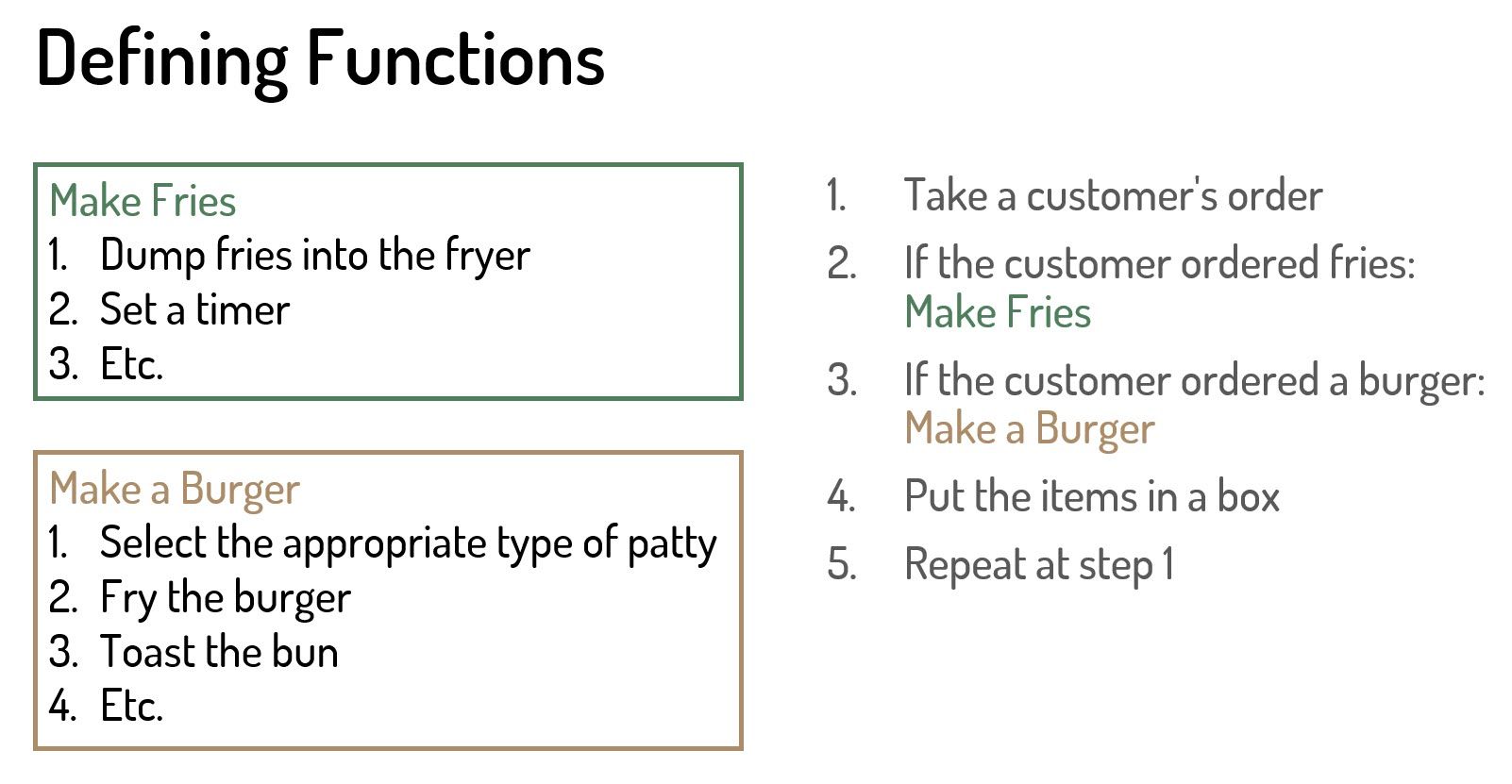
Are you hungry for some knowledge? Or maybe a burger?

Tackling any large set of tasks requires some amount of organization. Say we work at a burger restaurant. We could define a simplified view of the process as shown below:

1. Take a customer's order
2. If the customer ordered fries, make fries:
   * Dump fries into the fryer
   * Set a timer
   * Etc.
3. If the customer ordered a burger, make the burger:
   * Select the appropriate type of patty (veggie, chicken, fish, beef)
   * Fry the burger
   * Toast the bun
   * Etc.
4. Put the items in a box
5. Repeat at step 1

A worker takes a customer's order. If the customer ordered fries, they make fries. Notice the "sub-list" describing how to make the fries. And if the customer ordered a burger, they make the burger. And there's another "sub-list" describing how to make the burger.

To keep our main list of instructions straightforward and easier to follow, we can move these sub-lists to separate sets of instructions.



The left side of Figure 1 shows the list of steps for making fries, and the list for making a burger. We reference those instructions in the main flow, as shown on the right side of Figure 1.

The result is that each separate list of instructions is clearly defined. And the main flow on the right is easier to see without all of the sub-lists.

In programming, we call each of these self-contained sets of instructions a **function**.

Let's stop at this point and think about this. What are some benefits of breaking out some of the instructions into functions? Thoughts?

Separating our code into functions has several advantages:

* When building or maintaining the function, we can focus just on that function: what information it needs, what steps it performs, and what result it provides.
* We can simplify the main set of instructions, making it easier to read and maintain over time.
* It helps us separate work for a team, assigning each independent function to a member of the team. Jesse can be making fries, Chris making the burgers, and Sandhya follows the main flow, taking orders.
* And we can more easily reuse the function in multiple places in the application.

Did you think of other benefits?

## **What Is a Function?**

In programming:

* A **function** is a self-contained set of instructions to accomplish a chunk of a larger task.
* A function separates responsibilities for a specific part of a task, making the main task easier to work with and read.
* Functions add structure to our programs, and make them easier to read and modify over time.

Here is a tip: Code is often read much more often than it's written, so make your code readable.

## **Anatomy of a Function**

When writing code, functions look something like this:

### Make Fries

function makeFries(fries) {

... instructions go here ...

return cookedFries

}

### Make a Burger

function makeBurger(patty, bun, condiments) {

... instructions go here ...

return cookedBurger

}

Note that the details of the functions may look a bit different depending on the programming language you use.

A function often takes in some information, performs the set of instructions using that information, and gives back (or "returns") a result. At the burger joint we can say: "Hey Chris, here is a patty, bun, and condiments, go make the burger, and bring it back to me when it's done."

Functions are often named with the task they perform following a verbObject style naming convention: makeFries and makeBurger.

The name is followed by a list of the information that the function needs. In this example, this information is enclosed in parenthesis and separated with commas. For our makeFries, we need the fries. And for the burger, we need a patty, bun, and condiments.

The function body contains the set of instructions required for this function.

In many cases, a function performs its set of steps and returns a result. So lastly, we return that result. The result is often indicated with a return statement. In this example, when the fries are made or the burger is done, we pass them back to the main flow and they are boxed for the customer.

## **How to Create a Function**

Let's look at another example from a simple virtual pet adoption website as shown in Figure 2.



The user enters the type and number of pets, and clicks Adopt. The application then displays a message and a greeting from each of the virtually adopted pets.

When writing the code for this website, we want to simplify the main set of instructions by separating out the feature that prepares the pet greeting. We define a function for those instructions like this:

function prepareGreeting(typeOfPet, numberOfPets) {

var greeting = '';

for (let i = 0; i < numberOfPets; i++) {

if (typeOfPet === 'cat') {

greeting += 'meow' + '<br/>';

}

if (typeOfPet === 'dog') {

greeting += 'woof' + '<br/>';

}

}

return greeting;

}

This function is named "prepareGreeting" following our verbObject convention. It's best practice to give every function a meaningful name.

For a function to perform its set of instructions, it often needs some information. In this case, it needs the type of pet and the number of pets. When creating a function, we identify that needed information using parameters.

A **parameter** is a placeholder for the information the function needs. We give each placeholder a descriptive name, such as typeOfPet and numberOfPets. We add the parameters after the function name, often within parentheses and separated with commas.

The function name with its set of parameters is called a **function signature**. The function signature uniquely identifies the function.

The **function body** is where we write the code to perform the set of instructions. In this example, it's where we prepare the greeting.

In programming languages that use curly braces, the function body is defined between the first and last curly brace. In some languages, the function body is defined simply by its indentation.

In this function body, we prepare the pet's greeting. First, we initialize a greeting variable to an empty string. This ensures that we have a string (or text) variable we can use for the greeting text.

Then we loop for each pet. We use a counter represented by "i", repeat the loop while our counter is less than the total number of pets, and increment "i" at the end of each loop. Notice that in most programming languages, counting is zero-based, meaning it counts the iterations of the loop starting at 0: 0, 1, 2 for three pets.

Within the loop, if the passed-in pet type is a cat, we add a "meow" for each pet to the greeting variable. If the passed in pet type is a dog, we append "woof" for each pet. We then return that resulting greeting to the main set of instructions.

## **How to Call a Function**

Code in a function won't do anything until we call that function from some other code, such as our main set of instructions. The exact syntax for calling a function depends on the programming language you use. But it will look something like this:

greetingForDisplay = prepareGreeting("cat", 3)

We use the name of the function to identify which function we want to call. Then pass a value in for each parameter placeholder. In this example, we pass in a string (or text) using quotation marks and a number.

The result of the function's instructions are returned to the code that called it. In this example, the value is assigned to the greetingForDisplay variable. The main code could then display the contents of this variable to the user.

When working with functions, be sure to keep these two terms clear:

* **Parameter**: The placeholder in the function signature where we define what kind of information the function needs.
* **Argument**: The value(s) passed in when calling the function, giving the function the information it needs to perform its instructions

## **Wrapping Up**

We use a function to define a self-contained set of instructions for a chunk of a larger task. Using functions helps break up long code into manageable pieces. Just like building blocks, we combine functions to create simple to complex applications and websites.

For more information on general programming concepts, check out my course: ["Gentle Introduction to Programming for Beginners"](https://www.youtube.com/playlist?list=PLErOmyzRKOCrO9bwM1931IY8S3iWfhrr8). And for information on web development, GitHub, Angular, and C#, subscribe to [my YouTube channel](https://www.youtube.com/@deborah_kurata).

Now, let's go order that burger!