Introduction

This lesson will teach you about the importance of functions and how they are implemented in D.

WE'LL COVER THE FOLLOWING

Functions

Similar to how fundamental types are building blocks of program data, functions are building blocks of program behavior.

Functions are also closely related to the craft aspect of programming. Functions that are written by experienced programmers are succinct, simple and clear. This goes both ways: the mere act of trying to identify and write smaller building blocks of a program makes for a better programmer.

We have covered basic statements and expressions in previous chapters. Although there will be many more statements in later chapters, what we have seen so far are commonly-used features of D. Still, they are not sufficient on their own to write large programs. The programs that we have written so far have all been very short, each demonstrating just a simple feature of the language. Trying to write a program with any level of complexity without functions would be very difficult and prone to bugs.
This chapter covers only the basic features of functions. We will see more about functions in a later chapter.

Function: A real-life analogy

Functions are features that group statements and expressions together to form a unit of program execution. Together these statements and expressions

are given a *name* that describes what they collectively achieve. This name can then be used to execute this unit of code.

This idea of giving names to a sequence of steps to do a task is common in our daily lives. For example, the act of cooking an omelet can be described in some level of detail as follows:

- get a pan
- get butter
- get an egg
- turn on the stove
- put the pan on the fire
- put butter into the pan when it is hot
- put the egg into butter when it is melted
- remove the pan from the fire when the egg is cooked
- turn off the stove

Since that much detail is obviously excessive, steps that are related together would be combined under a single name:

- **make preparations** (get the pan, butter and the egg)
- turn on the stove
- **cook the egg** (put the pan on the fire, etc.)
- turn off the stove

Going further, there can be a single name for all of the steps:

- **make a one-egg omelet**(all of the steps)

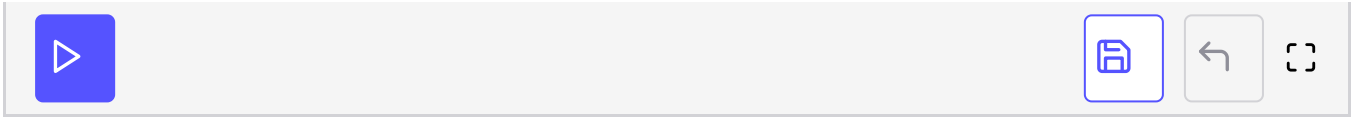
Functions are based on the same concept: steps that can collectively be named as a whole are put together to form a function. As an example, let's start with the following lines of code that achieve the task of displaying a menu:

```
import std.stdio;
```

```
void main() {


    writeln(" 0 Exit");
    writeln(" 1 Add");
    writeln(" 2 Subtract");
    writeln(" 3 Multiply");
    writeln(" 4 Divide");

}
```



Code to print a menu

Since it would make sense to name those combined lines as `printMenu`, they can be put together to form a function by using the following syntax:

```
void printMenu() {
    writeln(" 0 Exit");
    writeln(" 1 Add");
    writeln(" 2 Subtract");
    writeln(" 3 Multiply");
    writeln(" 4 Divide");
}
```

The contents of that function can now be executed from within `main()` simply by using its name:

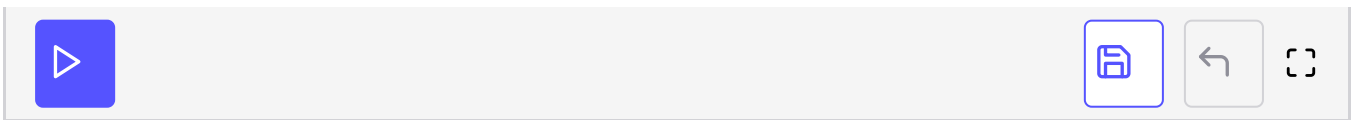
```
import std.stdio;

void printMenu() {
    writeln(" 0 Exit");
    writeln(" 1 Add");
    writeln(" 2 Subtract");
    writeln(" 3 Multiply");
    writeln(" 4 Divide");
}

void main() {

    printMenu();

}
```



It may be obvious from the similarities between the definitions of `printMenu()`

and `main()` that `main()` is a function as well. The execution of a D program starts with the function named `main()` and branches out to other functions from there.

Parameters

Some of the powers of functions come from the fact that their behaviors are adjustable through parameters.
Let's continue with the omelet example by modifying it to make an omelet of five eggs instead of one. The steps would exactly be the same, the only difference being the number of eggs to use. We can change the more general description above accordingly:

- make preparations (get the pan, butter and five eggs)
- turn on the stove
- cook the eggs (put the pan on the fire, etc.)
- turn off the stove

Likewise, the most general single step would become the following:

- make a five-egg omelet (all of the steps)

This time there is additional information that concerns some of the steps: "get five eggs," "cook the eggs," and "make a five-egg omelet."

Behaviors of functions can be adjusted similar to the omelet example. The information that functions use to adjust their behavior is passed using **parameters**. Parameters are specified in a comma-separated, function parameter list. The parameter list rests inside of the parentheses that comes after the name of the function.

The `printMenu()` function above was defined with an empty parameter list because that function always printed the same menu. Let's assume that sometimes the menu will need to be printed differently in different contexts. For example, it may make more sense to print the first entry as `Return` instead of `Exit` depending on the part of the program that is being executed at that time.

In such a case, the first entry of the menu can be parameterized by being defined in the parameter list. The function then uses the value of that parameter instead of the literal `"Exit"`:

```
void printMenu(string firstEntry) {
    writeln(" 0 ", firstEntry);
    writeln(" 1 Add");
    writeln(" 2 Subtract");
    writeln(" 3 Multiply");
    writeln(" 4 Divide");
}
```

Notice that since the information conveyed in the `firstEntry` parameter is a piece of text, its type has been specified as `string` in the parameter list. This function can now be called with different parameter values to print menus having different first entries. All that needs to be done is using the appropriate `string` values, depending on where the function is being called from:

```
import std.stdio;

void printMenu(string firstEntry) {
    writeln(" 0 ", firstEntry);
    writeln(" 1 Add");
    writeln(" 2 Subtract");
    writeln(" 3 Multiply");
    writeln(" 4 Divide");
}

void main() {

    // At some place in the program:
    printMenu("Exit");
    // ...
    writeln ("\n");
    // At some other place in the program:
    printMenu("Return");
}
```

Reusability of code using functions

> **Note:** When you write and use your own functions with parameters of type `string`, you may encounter compilation errors. As written above, `printMenu()` cannot be called with parameter values of type `char[]`. For

printMenu() cannot be called with parameter values of type char[]. For example, the following code would cause a compilation error:

```
char[] anEntry;
anEntry ~= "Take square root";
printMenu(anEntry); // ← compilation ERROR
```

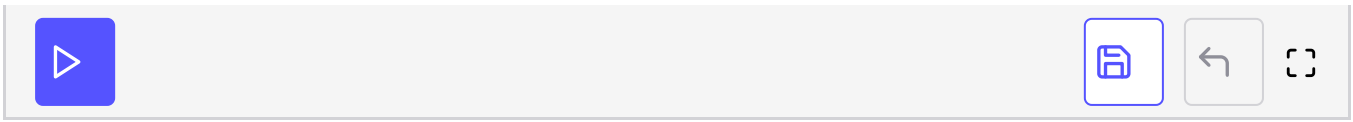
```
import std.stdio;

void printMenu(string firstEntry) {
    writeln(" 0 ", firstEntry);
    writeln(" 1 Add");
    writeln(" 2 Subtract");
    writeln(" 3 Multiply");
    writeln(" 4 Divide");
}

void main() {

    char[] anEntry;
    anEntry ~= "Take square root";
    printMenu(anEntry); // ← compilation ERROR

}
```



On the other hand, if `printMenu()` was defined to take its parameter as `char[]`, then it could not be called with strings like `Exit`. This is related to the concept of immutability and the `immutable` keyword, both of which will be covered in a later chapter.

Let's continue with the menu function and assume that it is not appropriate to always start the menu selection numbers with zero. In that case, the starting number can also be passed to the function as its second parameter. The parameters of the function must be separated by commas:

```
void printMenu(string firstEntry, int firstNumber) {
    writeln(' ', firstNumber + 0, ' ', firstEntry);
    writeln(' ', firstNumber + 1, " Add");
    writeln(' ', firstNumber + 2, " Subtract");
    writeln(' ', firstNumber + 3, " Multiply");
    writeln(' ', firstNumber + 4, " Divide");
}
```

It is now possible to tell the function what number to start at:

```
import std.stdio;

void printMenu(string firstEntry, int firstNumber) {
    writeln(' ', firstNumber + 0, ' ', firstEntry);
    writeln(' ', firstNumber + 1, " Add");
    writeln(' ', firstNumber + 2, " Subtract");
    writeln(' ', firstNumber + 3, " Multiply");
    writeln(' ', firstNumber + 4, " Divide");
}

void main() {

    printMenu("Return", 1);

}
```

printMenu() function with different first number

In the next lesson, you will learn how a function can be called, how it performs the task and finally how a function sends back the result.