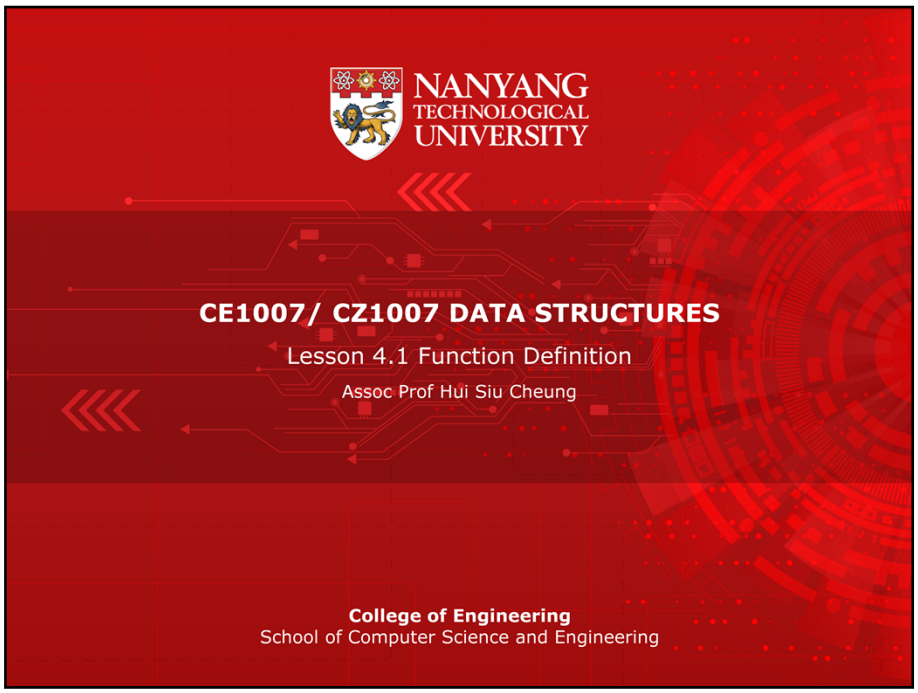
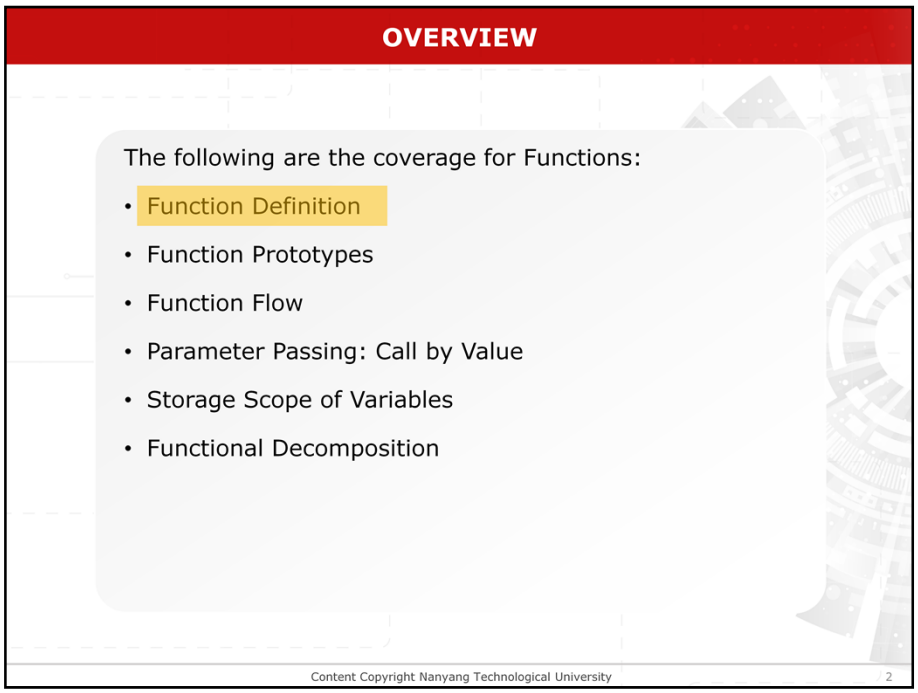
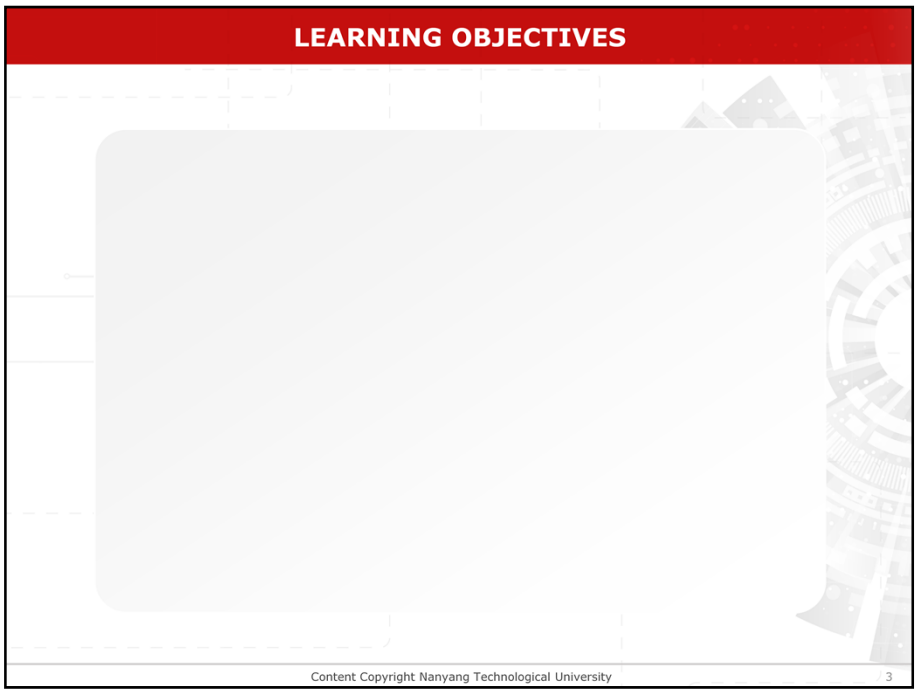


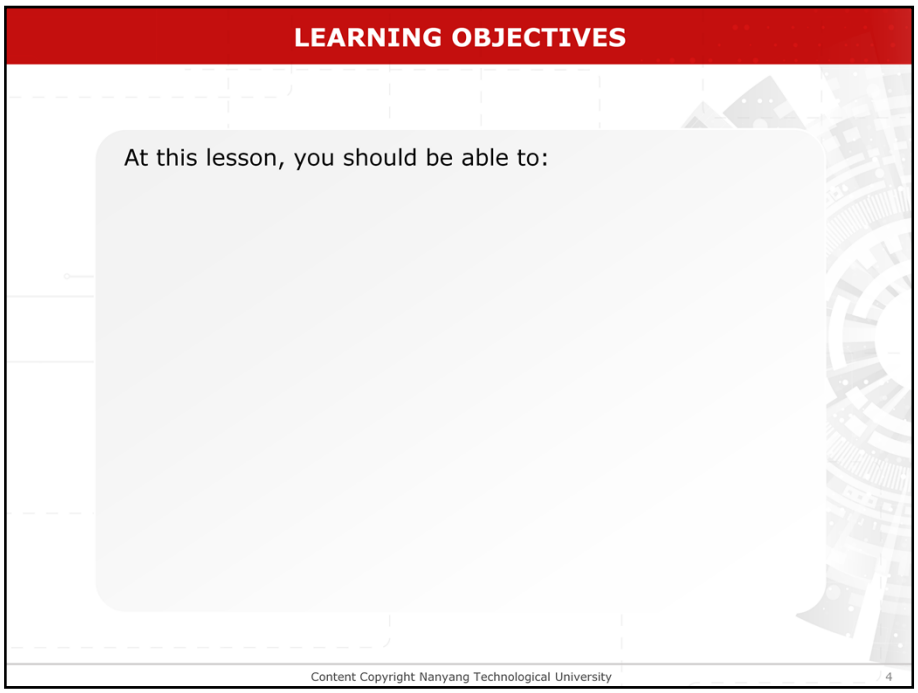
This lesson is on Functions.



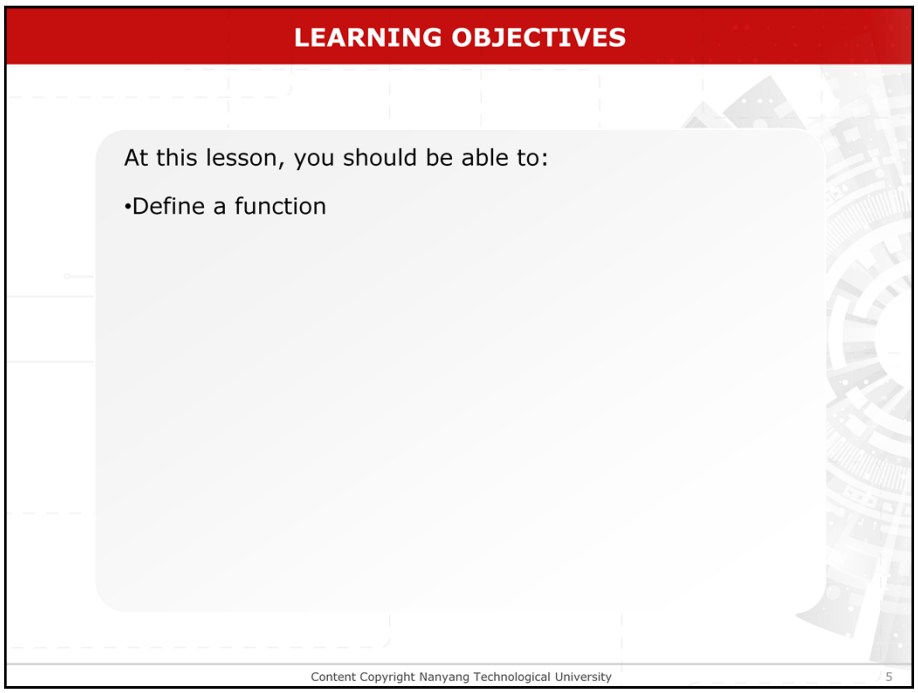
There are 6 main sections to cover for Functions. This video focused on the 1st topic: Function definition



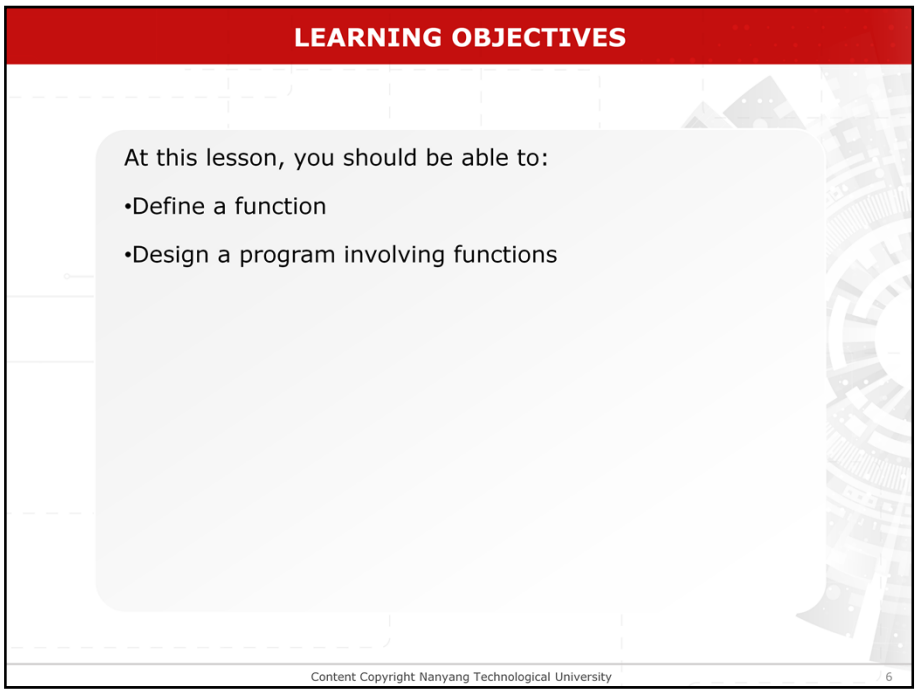
Learning objectives



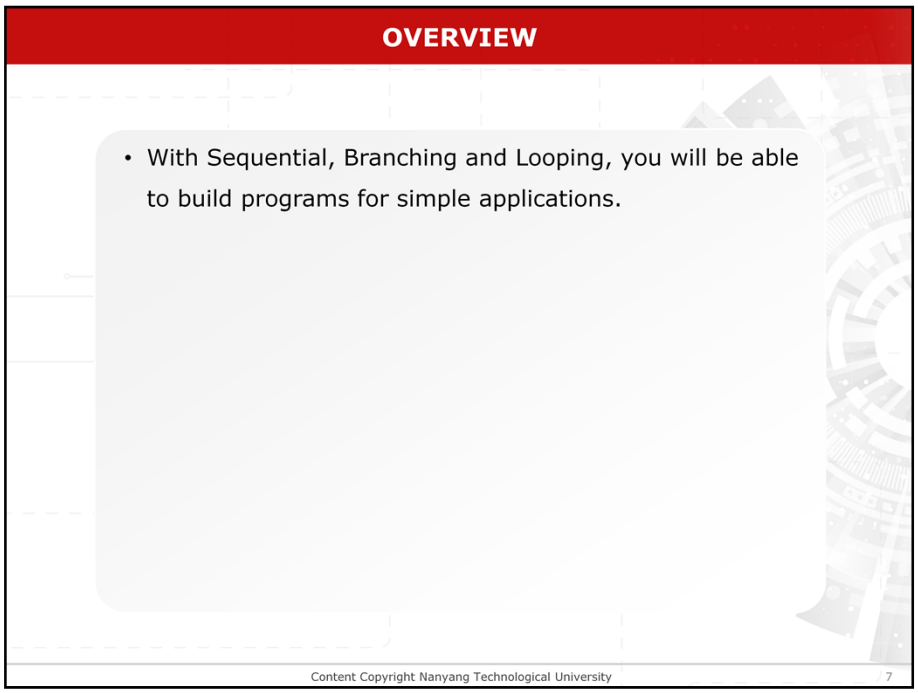
At this lesson, you should be able to:



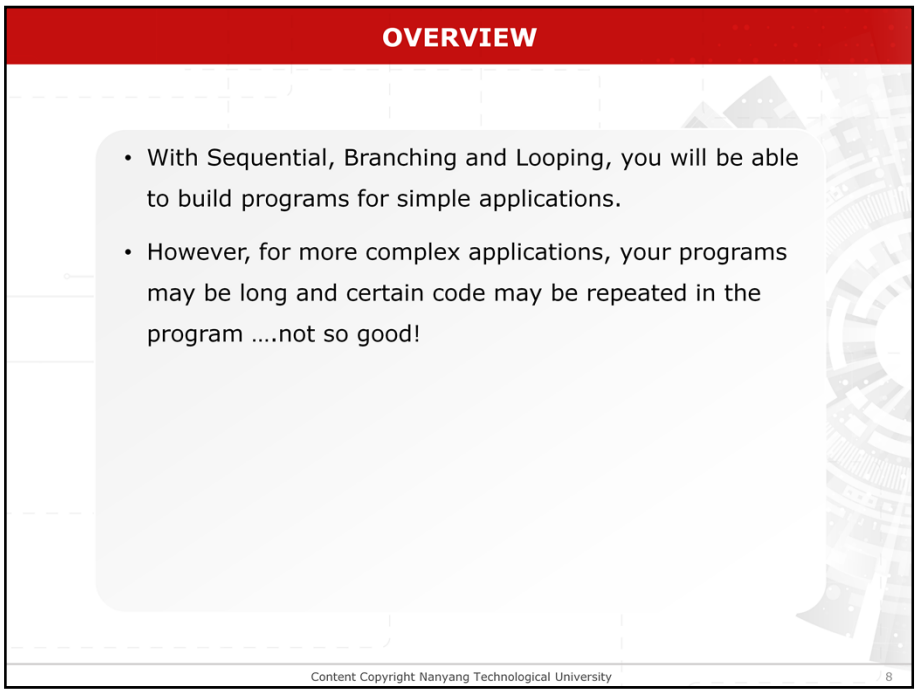
Define a function



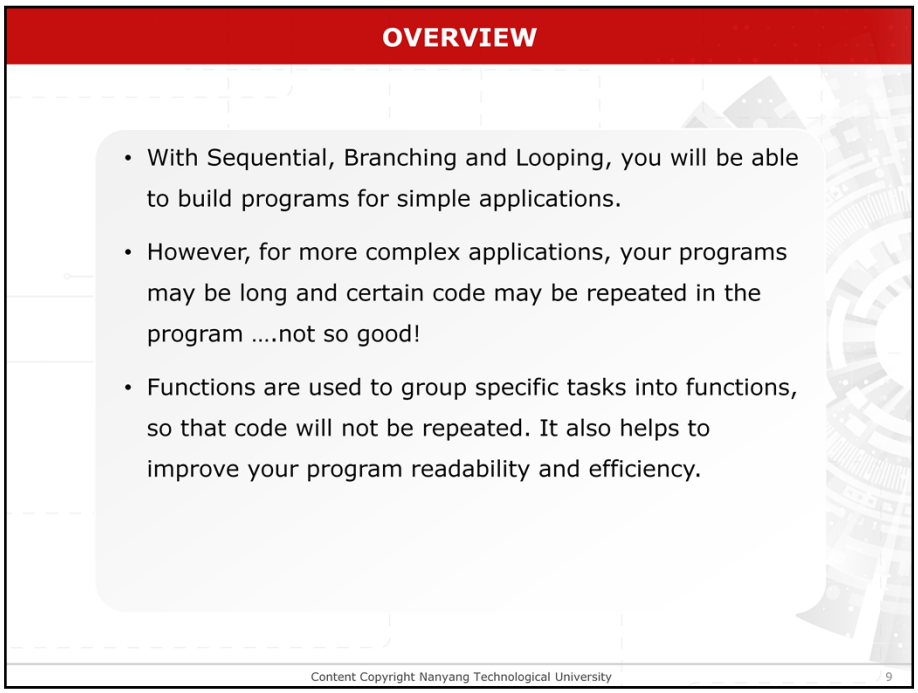
Design a program involving functions



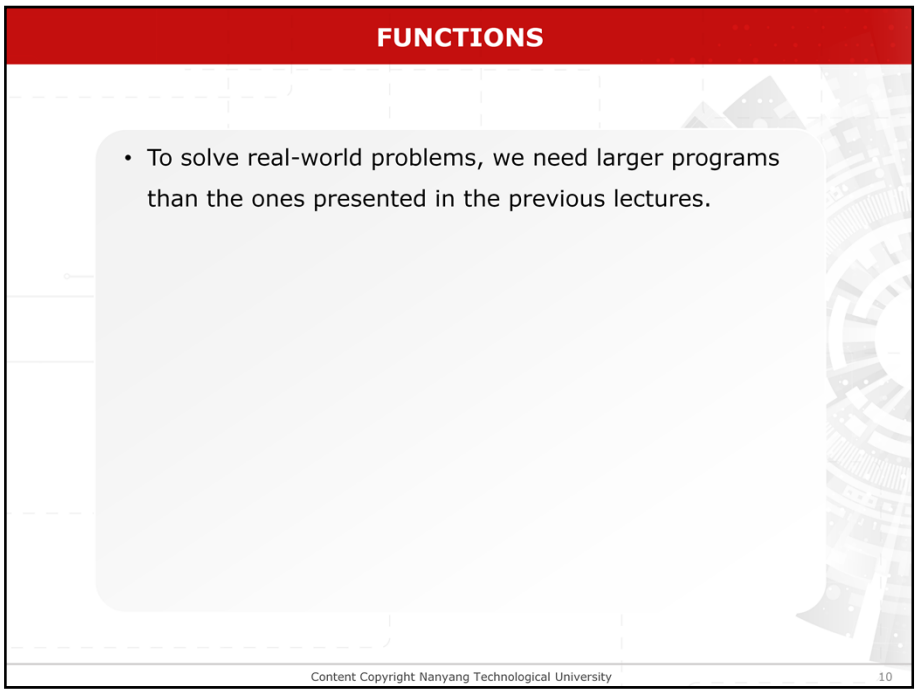
With Sequential, Branching and Looping, you will be able to build programs for simple applications.



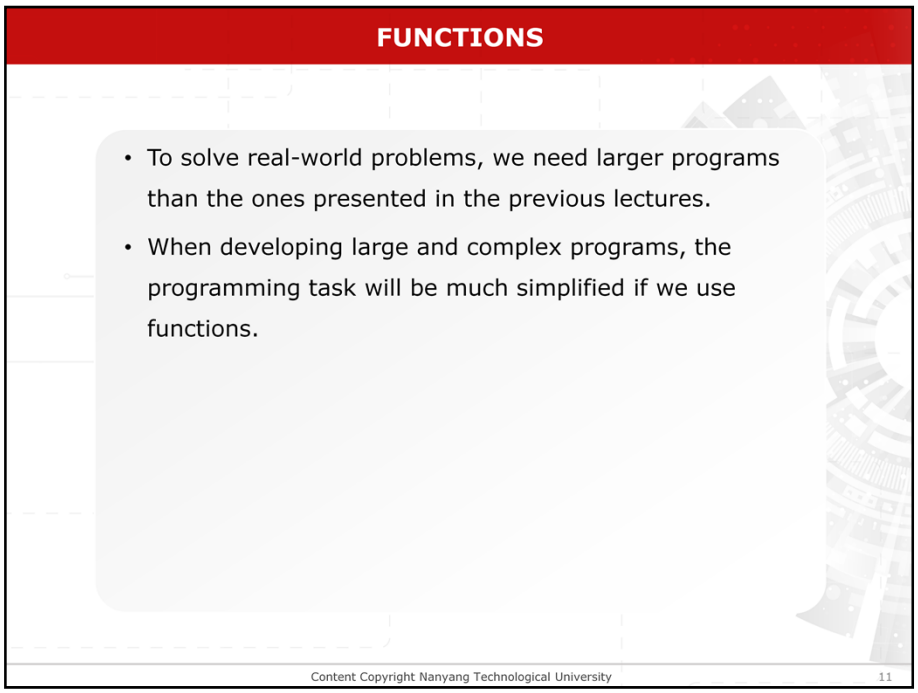
However, for more complex applications, your programs may be long and certain code may be repeated in the program ….not so good!



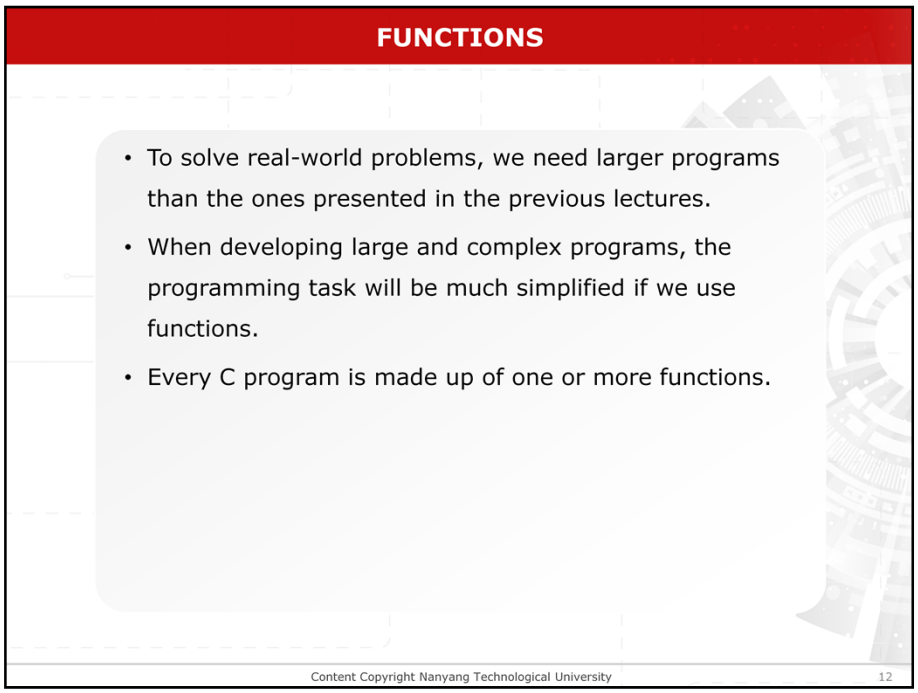
Functions are used to group specific tasks into functions, so that code will not be repeated. It also helps to improve your program readability and efficiency.



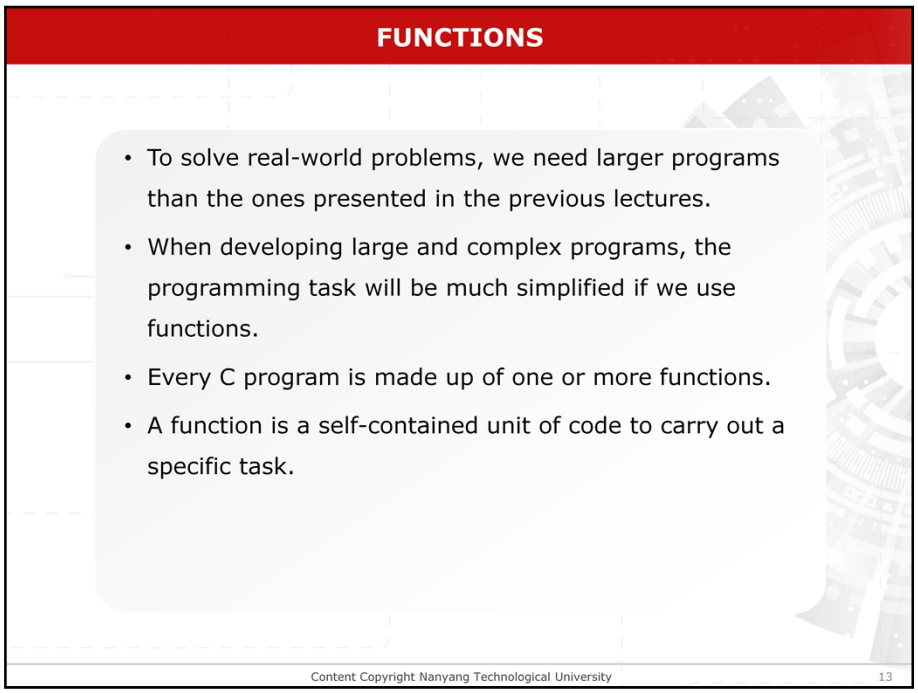
To solve real-world problems, we need larger programs than the ones presented in the previous lectures.



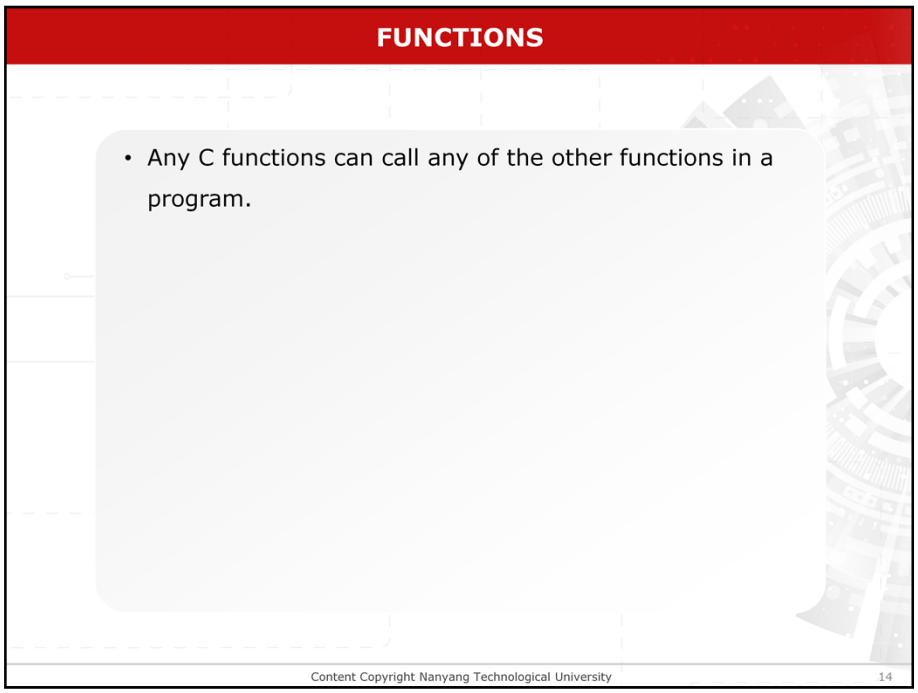
When developing large and complex programs, the programming task will be much simplified if we use functions.



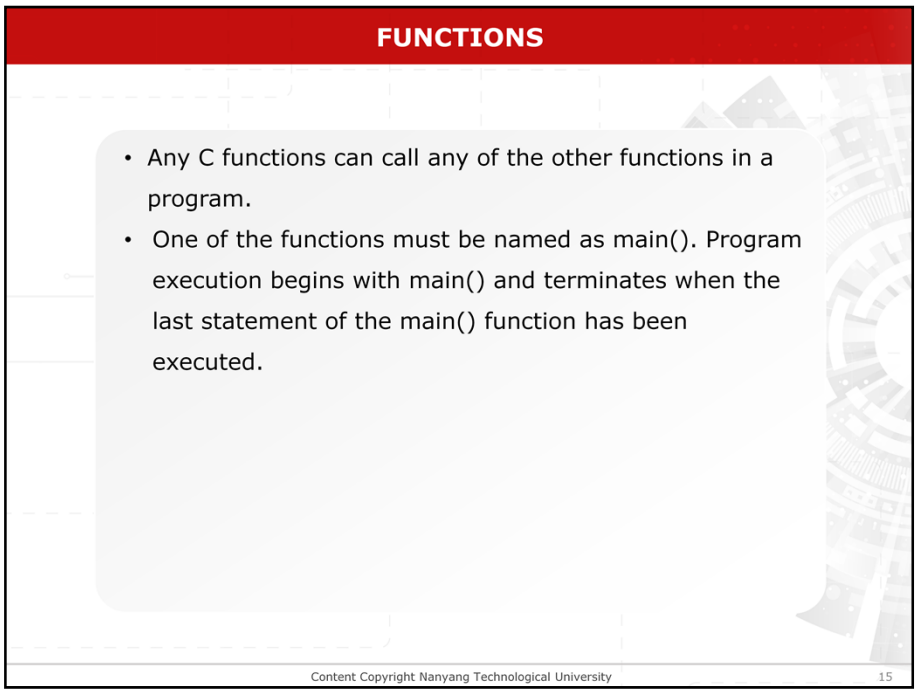
Every C program is made up of one or more functions.



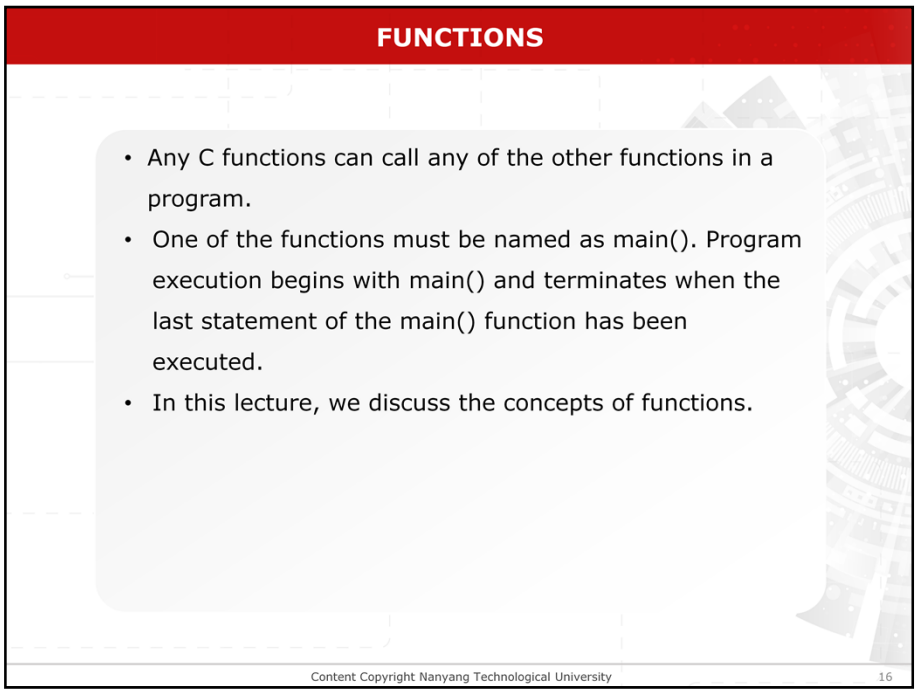
A function is a self-contained unit of code to carry out a specific task.



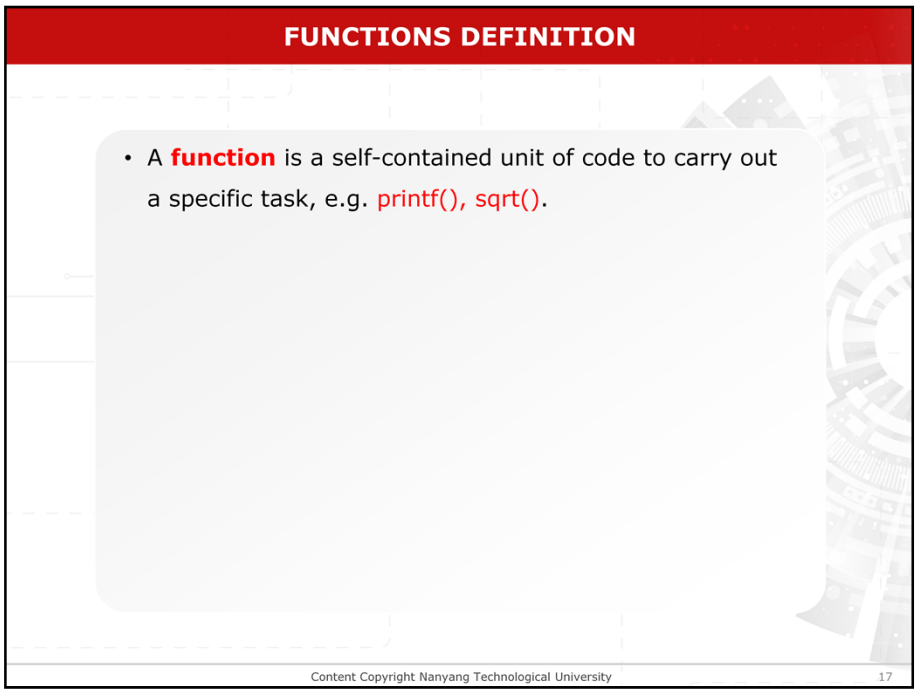
Any C functions can call any of the other functions in a program.



- One of the functions must be named as main(). Program execution begins with main() and terminates when the last statement of the main() function has been executed.

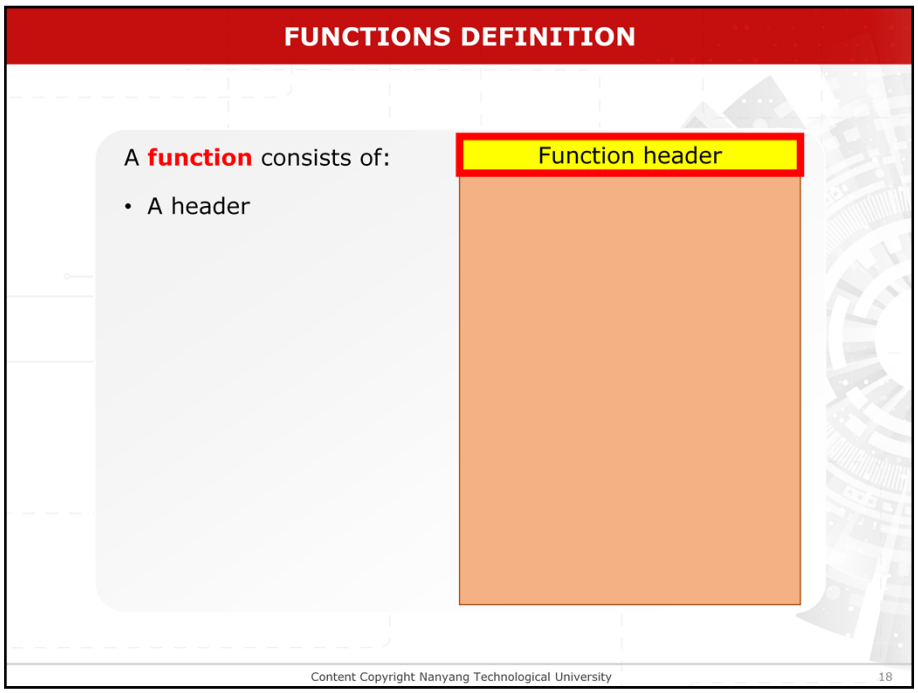


In this lecture, we discuss the concepts of functions.

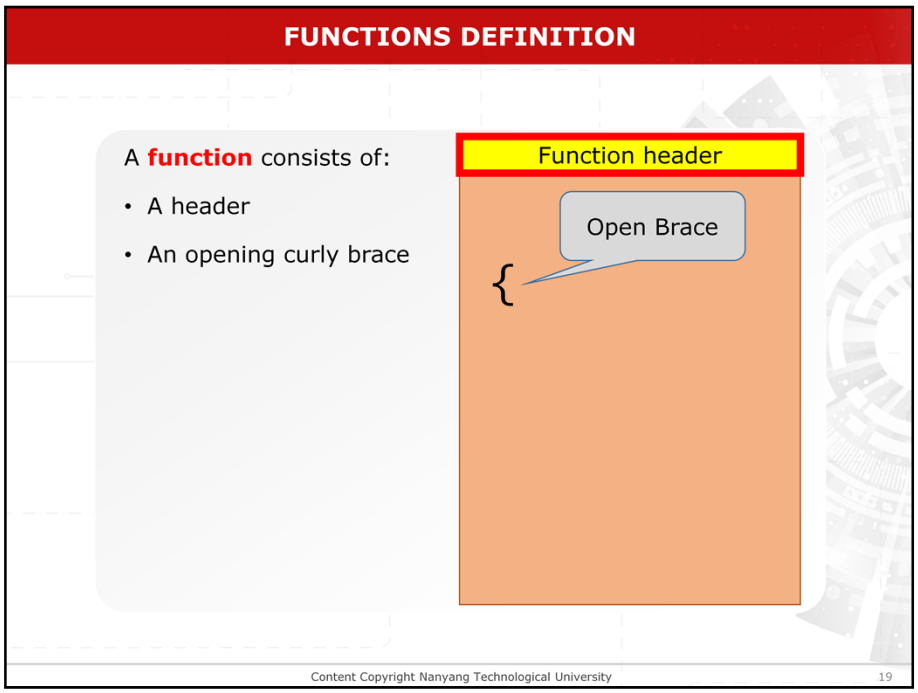


Function Definition

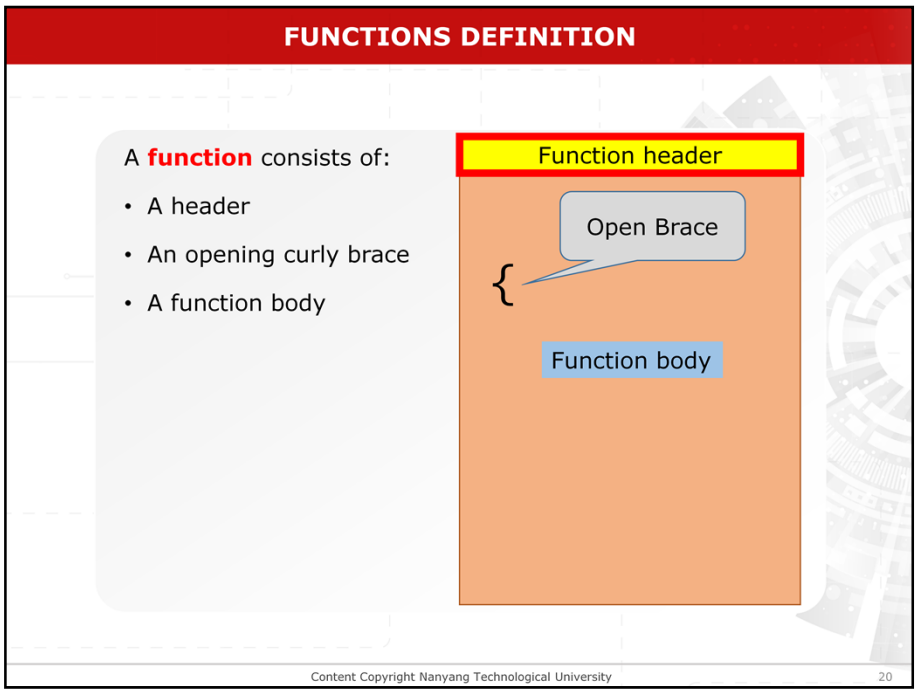
A function is a piece of code, which specifies the actions of the function, and the local data used by the function.



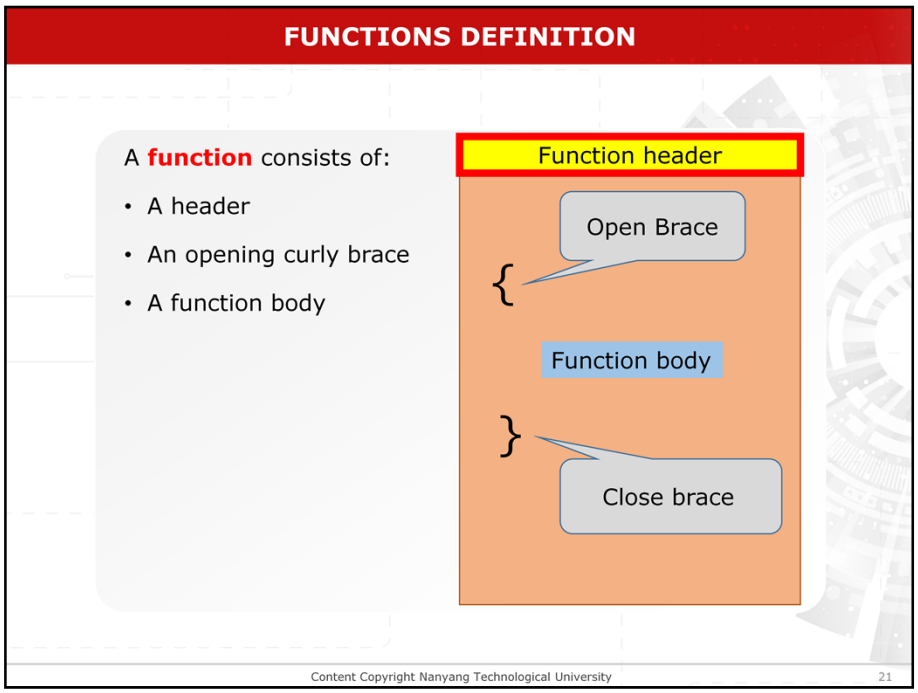
A function definition has the following structure: f**unction_header**



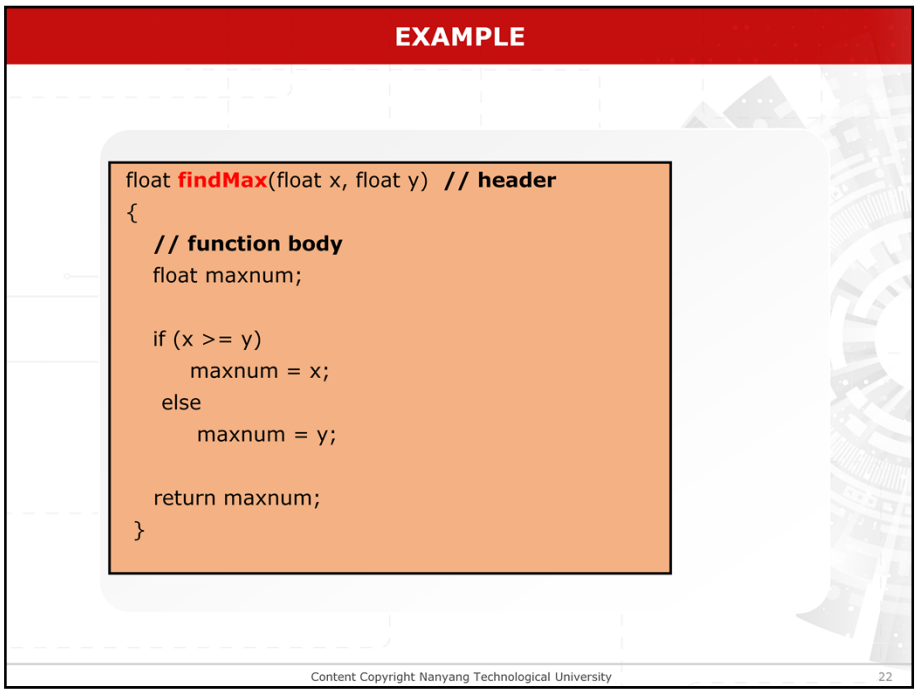
an opening curly brace



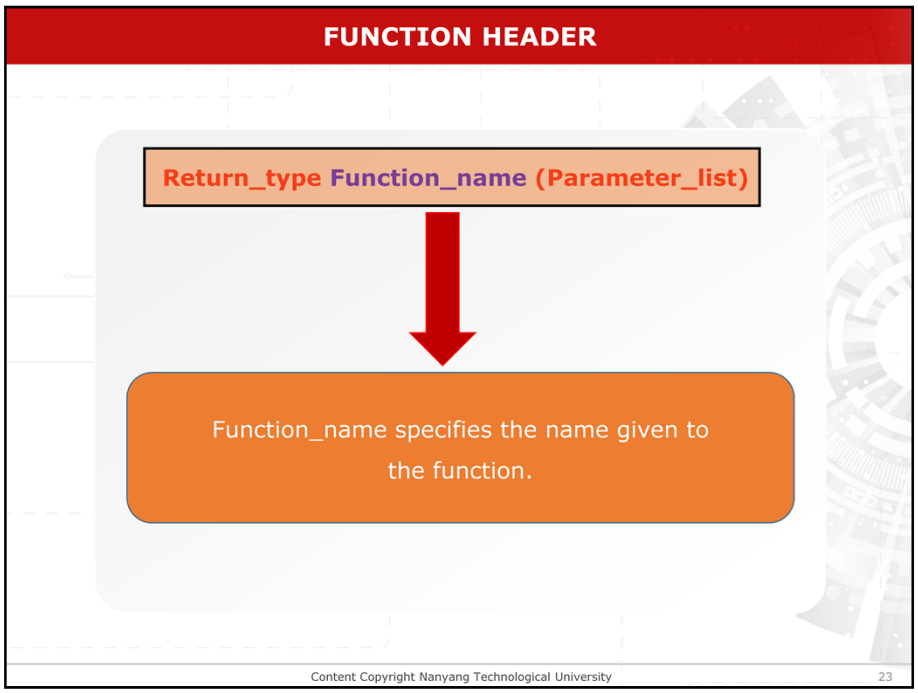
a function body



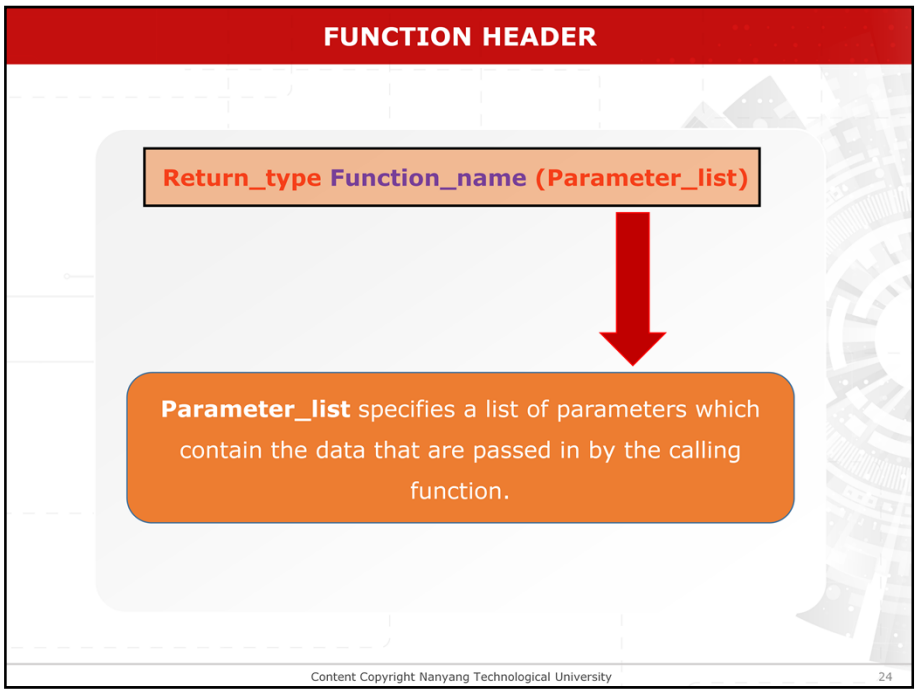
And a closely curly brace



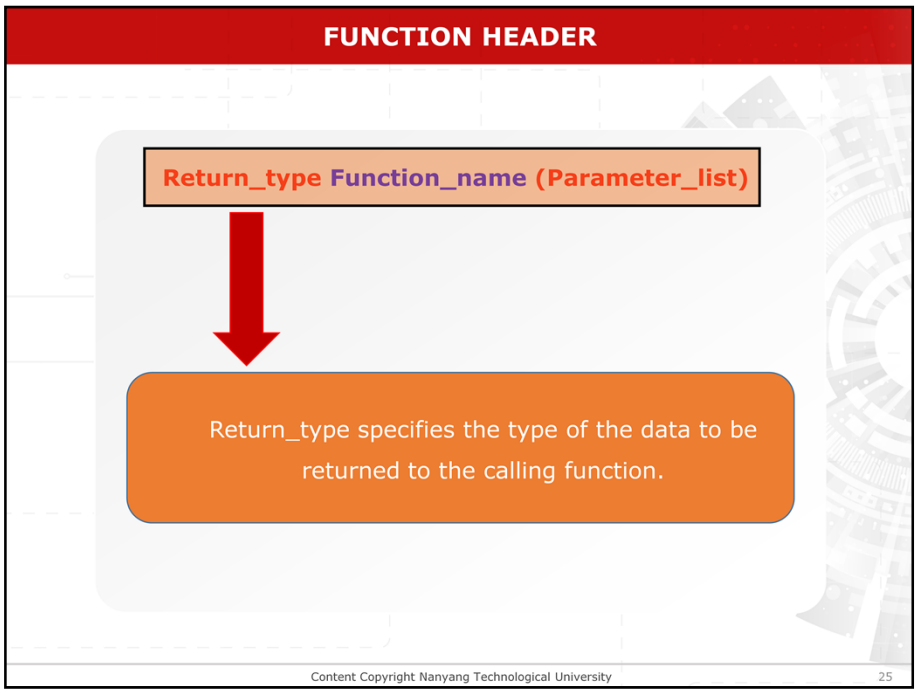
An example is illustrated in the **findMax()** function, which has the function header and function body.



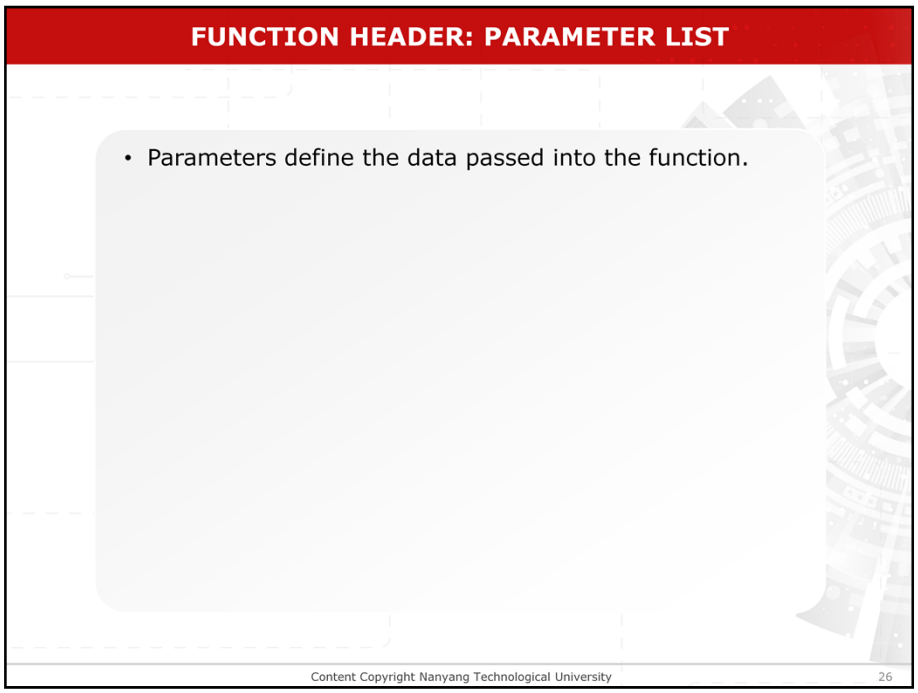
The *function header* has the format as shown. **Function name** is the name given to the function.



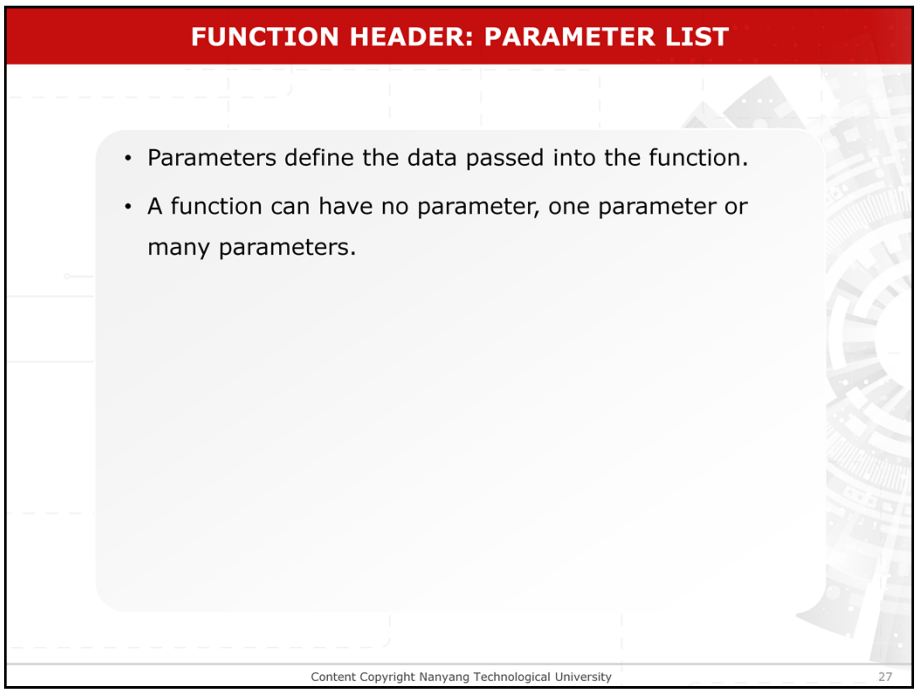
Parameter list specifies a list of parameters which contain the data that are passed in by the calling function.



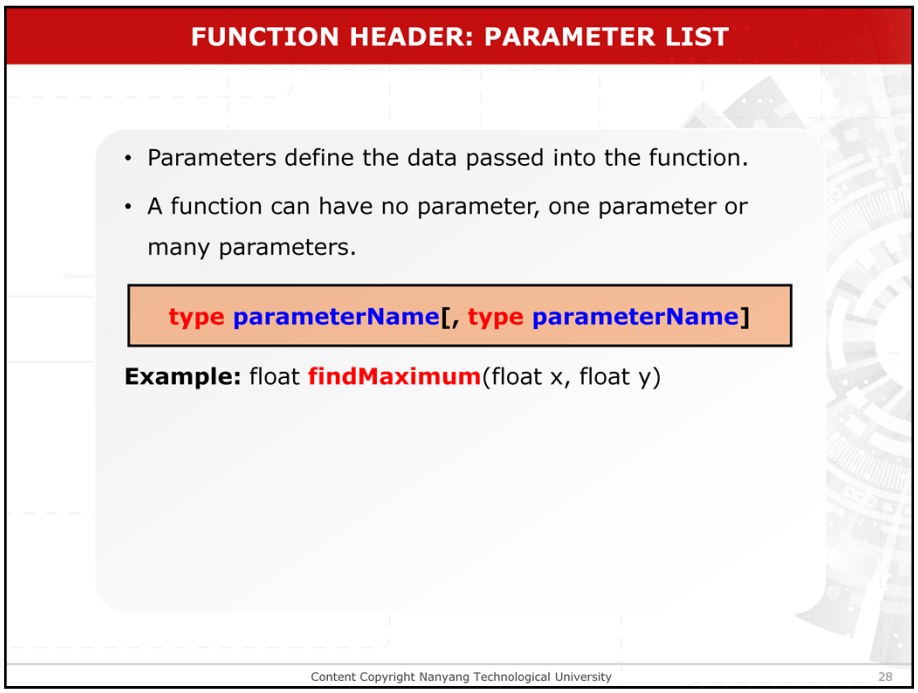
Return type specifies the type of the data to be returned to the calling function.



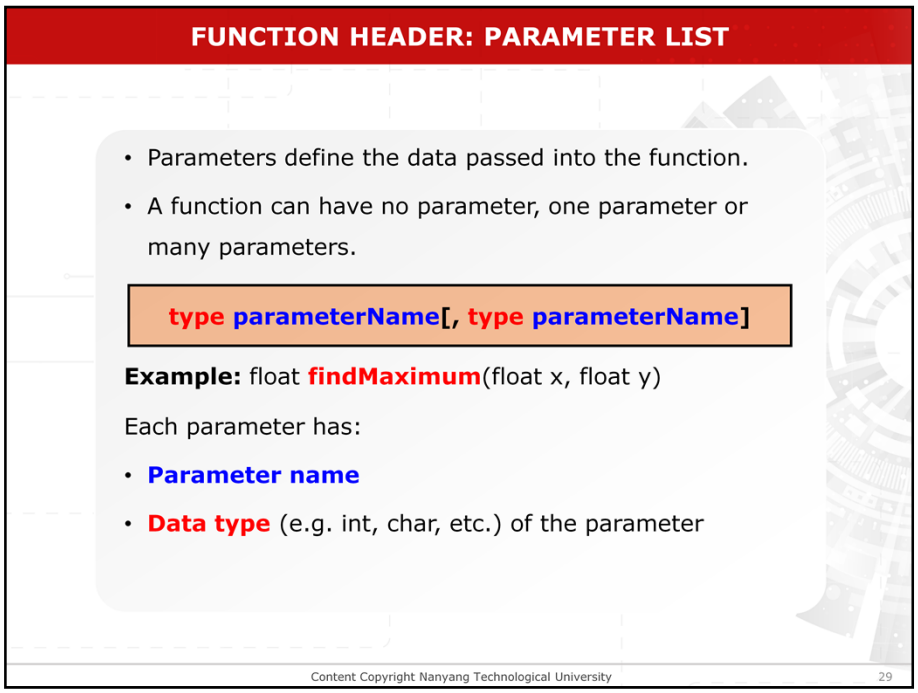
Parameters define the **data** **passed into** the function.



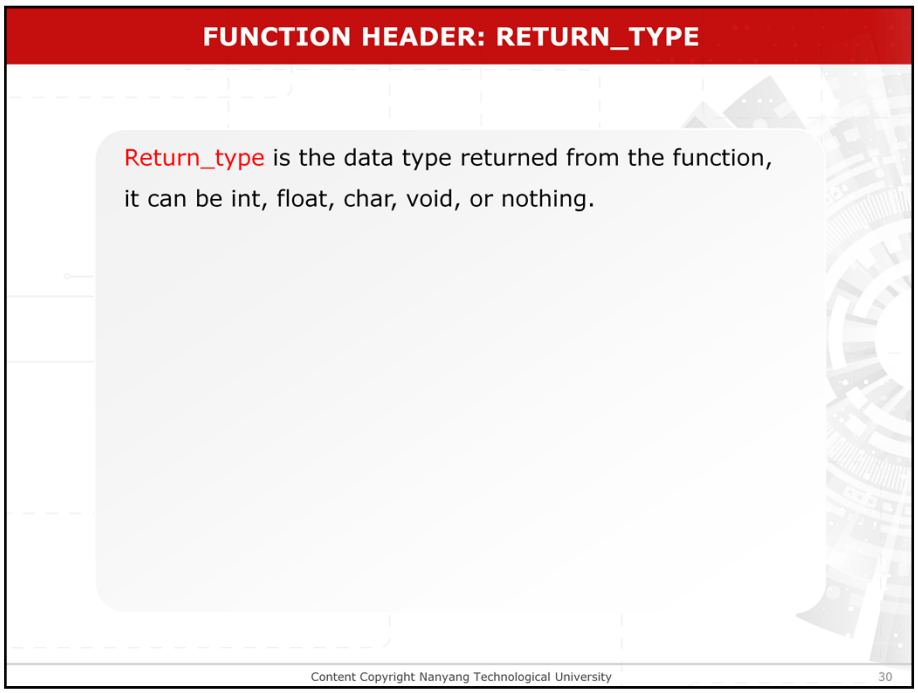
A function can have no parameter, one parameter or many parameters.



The format is shown here, where **parenthesis c**an be repeated zero or more times.

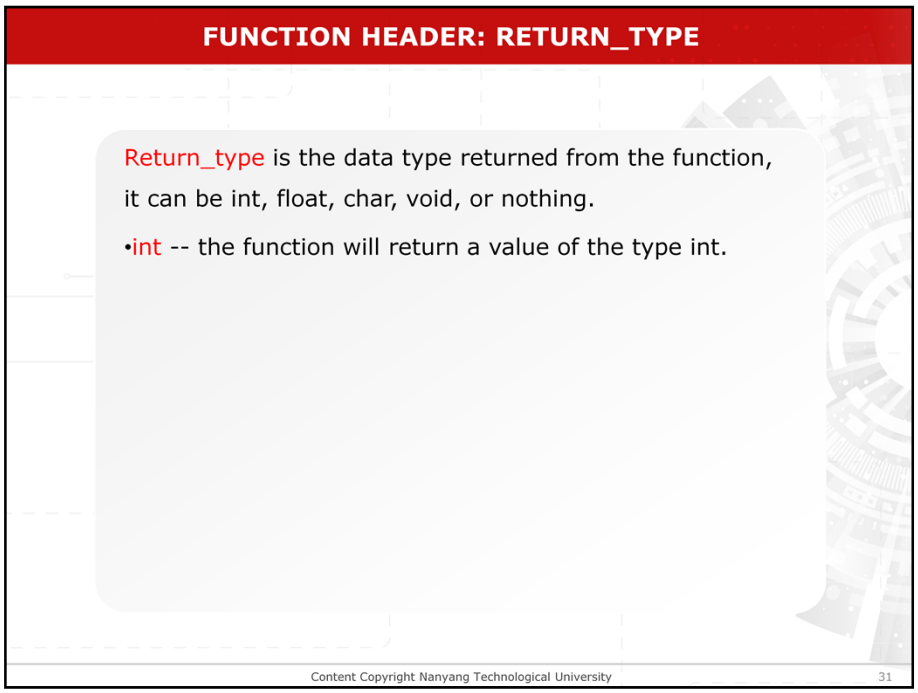


The parameters are known only inside the function body. Each parameter has: **parameter name** and **data type** (eg int, car, etc)of the parameter

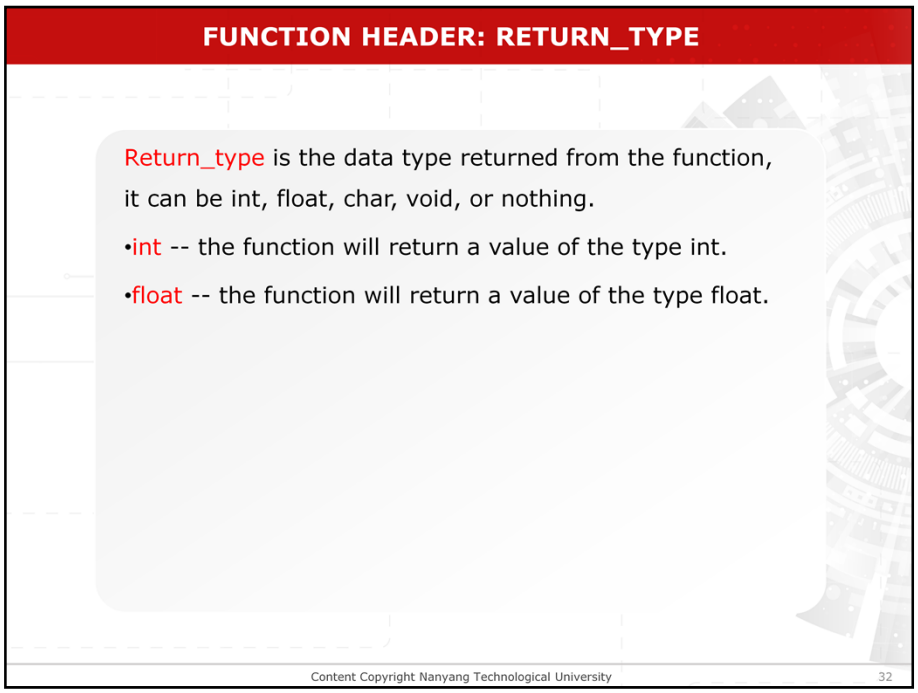


Function Header: Return_type

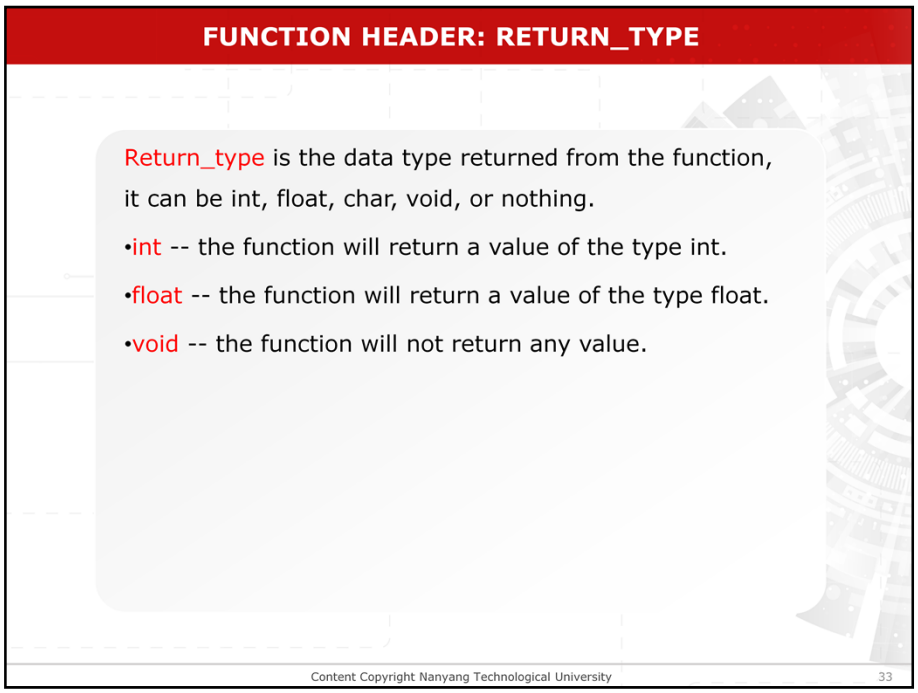
Return_type is the data type of the value returned by the function, it can be int, float, char, void, or nothing.



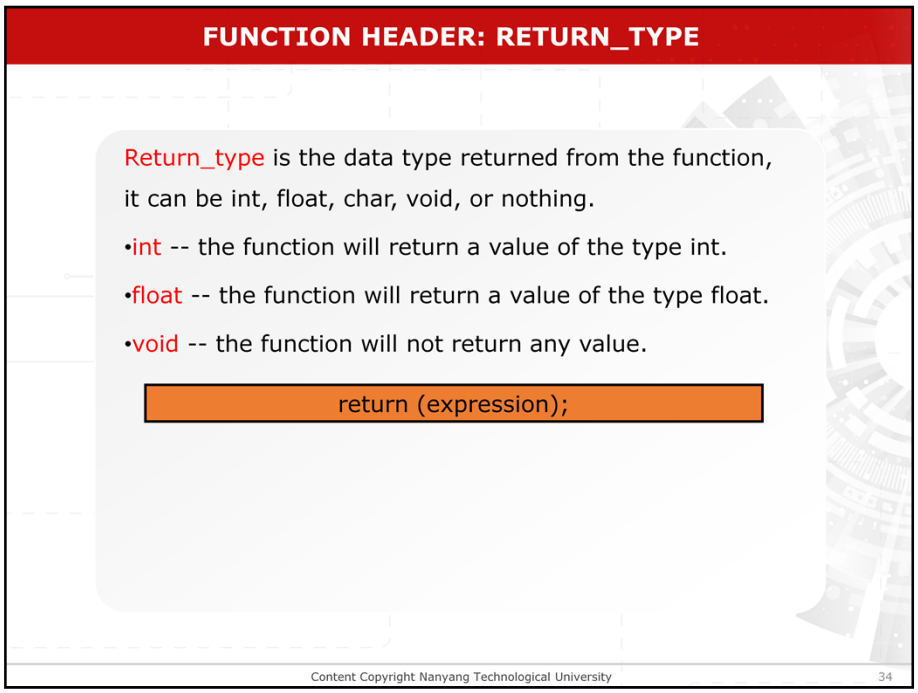
Int , the function will return a value of the type int



Float, the function will return a value of the type float

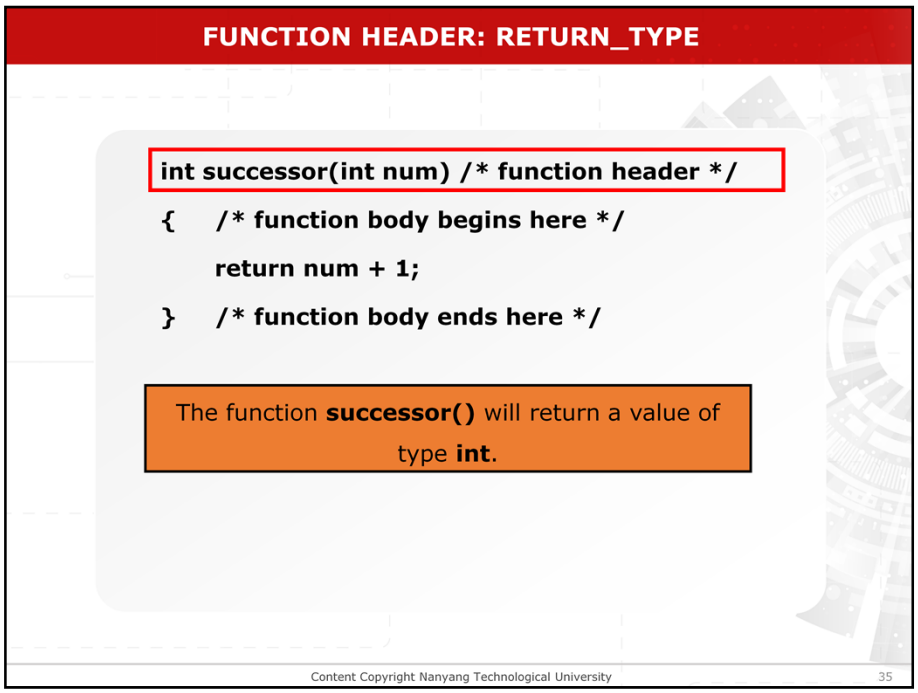


Void, the function will not return any value

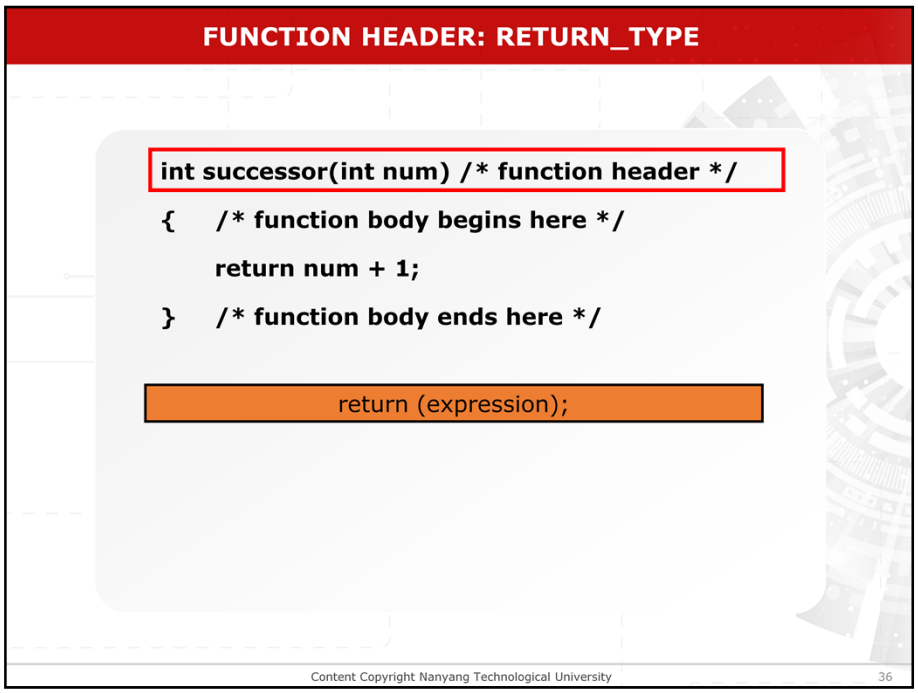


The syntax for the **return** statement is

return (expression);



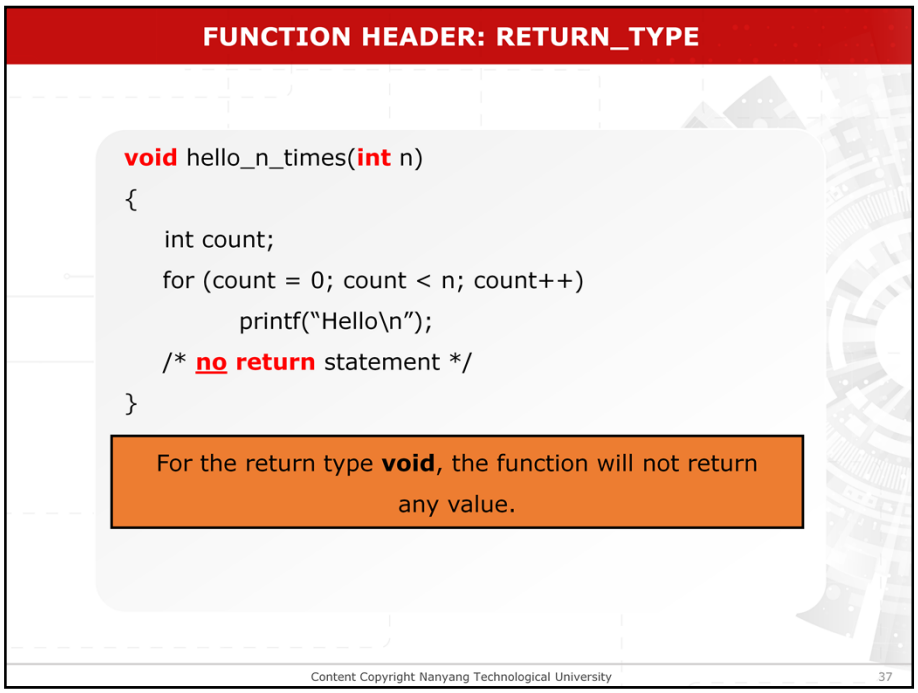
In the example shown, the function **successor()** will return a value of type **int**.



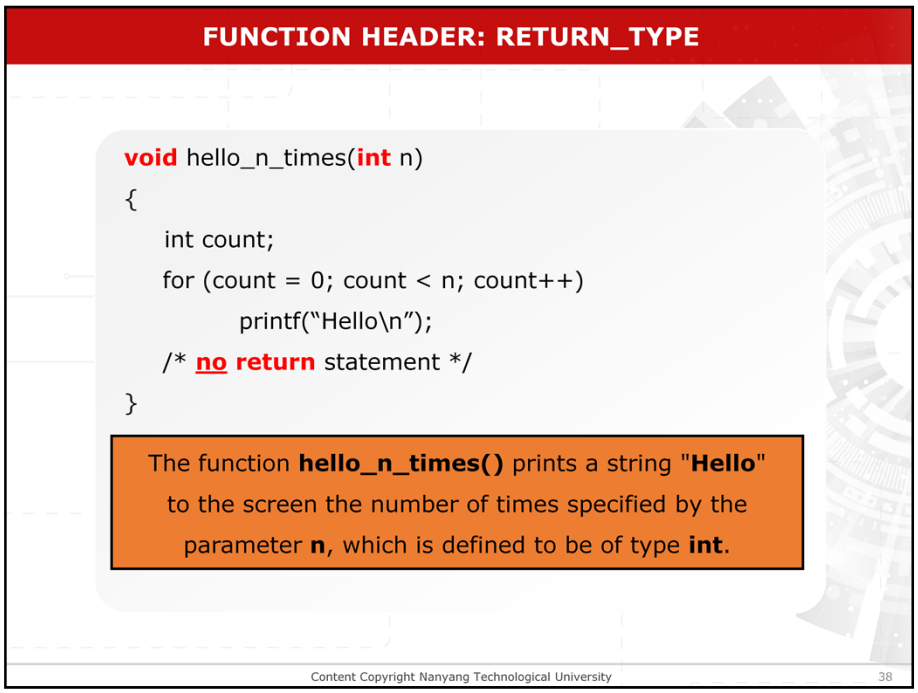
In the example shown, the function **successor()** will return a value of type **int**.
For the return type **void**, the function will not return any value. The function **hello_n_times()**

```
void hello_n_times(int n)
{
    int count;
    for (count = 0; count < n; count++)
        printf("Hello\n");
    /* no return statement here */
}
```

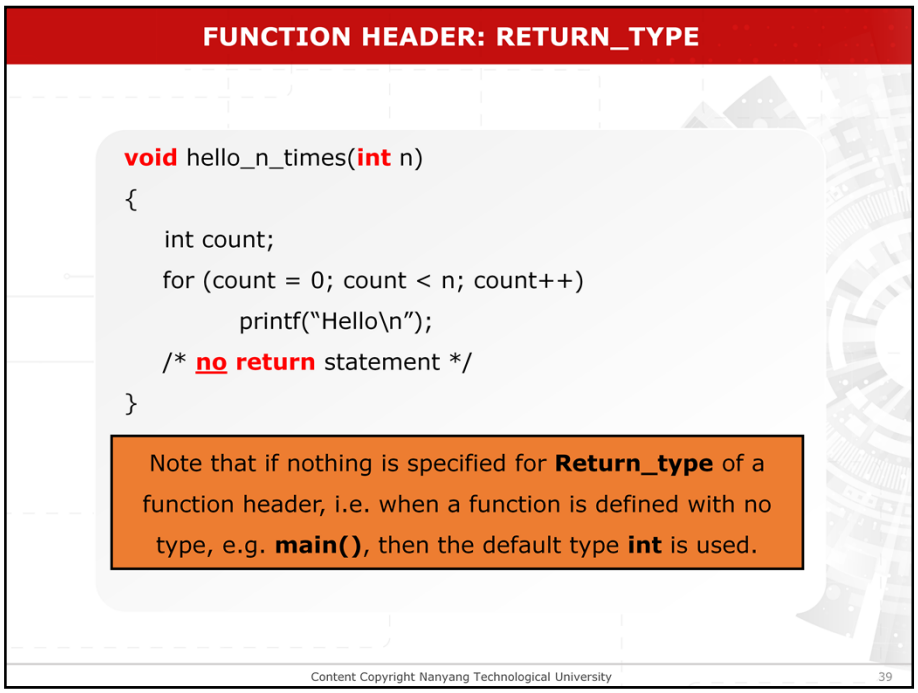
prints a string "**Hello**" to the screen the number of times specified by the parameter **n**, which is defined to be of type **int**.
If nothing is specified for **Return_type** of a function header, i.e. when a function is defined with no type, e.g. **main()**, then the default type **int** is used.



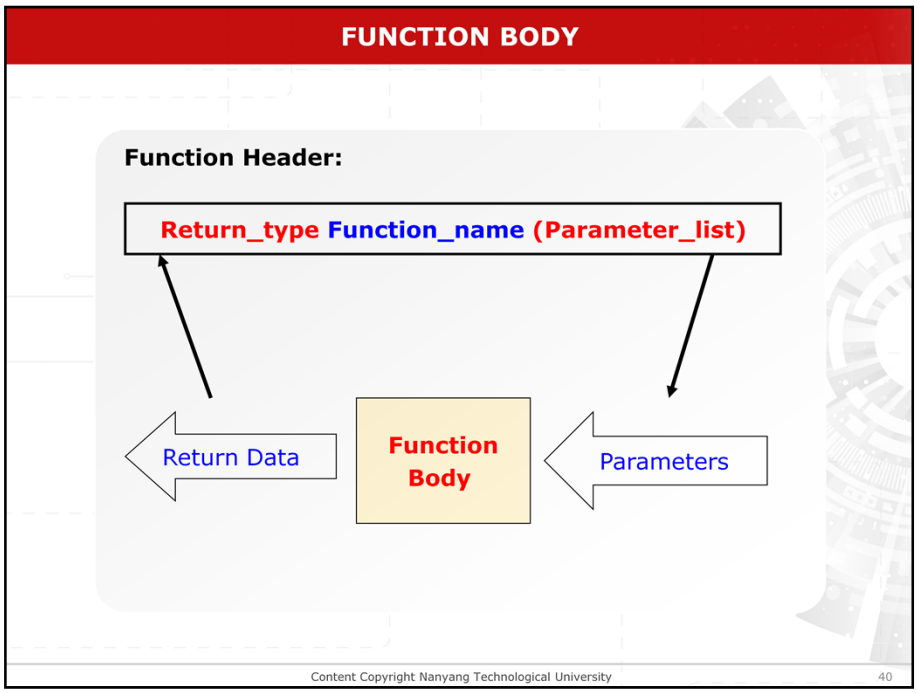
For the return type **void**, the function will not return any value.



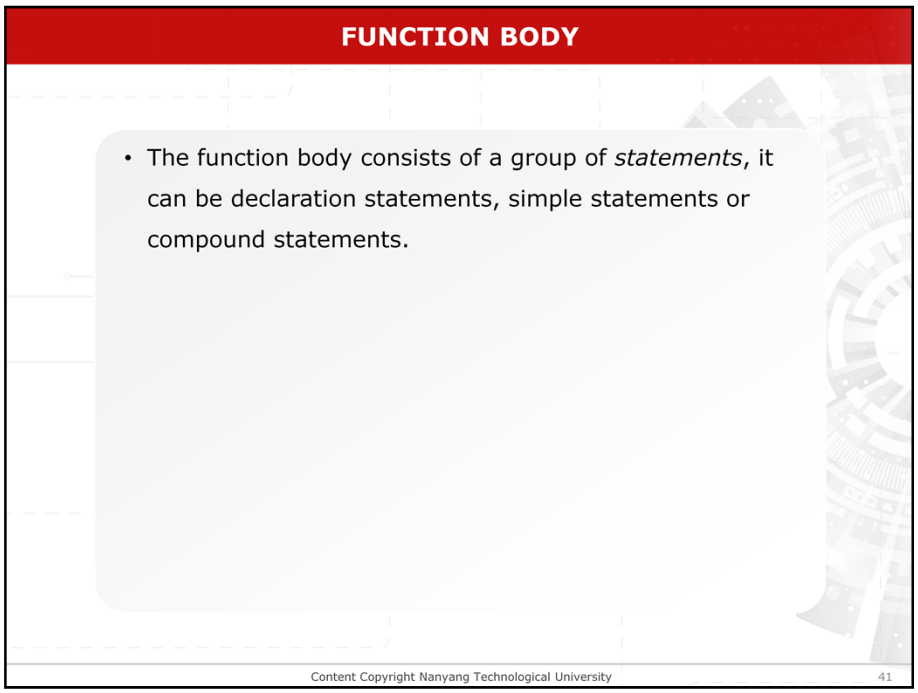
The function **hello_n_times()** prints a string "**Hello**" to the screen the number of times specified by the parameter **n**, which is defined to be of type **int**.
If nothing is specified for **Return_type** of a function header, i.e. when a function is defined with no type, e.g. **main()**, then the default type **int** is used.



If nothing is specified for **Return_type** of a function header, i.e. when a function is defined with no type, e.g. **main()**, then the default type **int** is used.

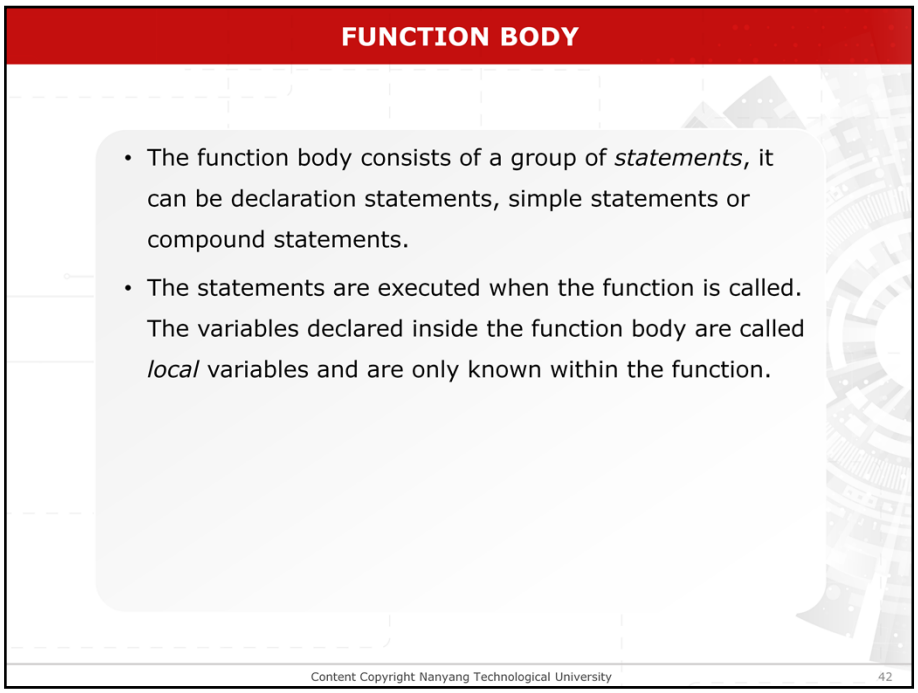


The function header and function body are shown here

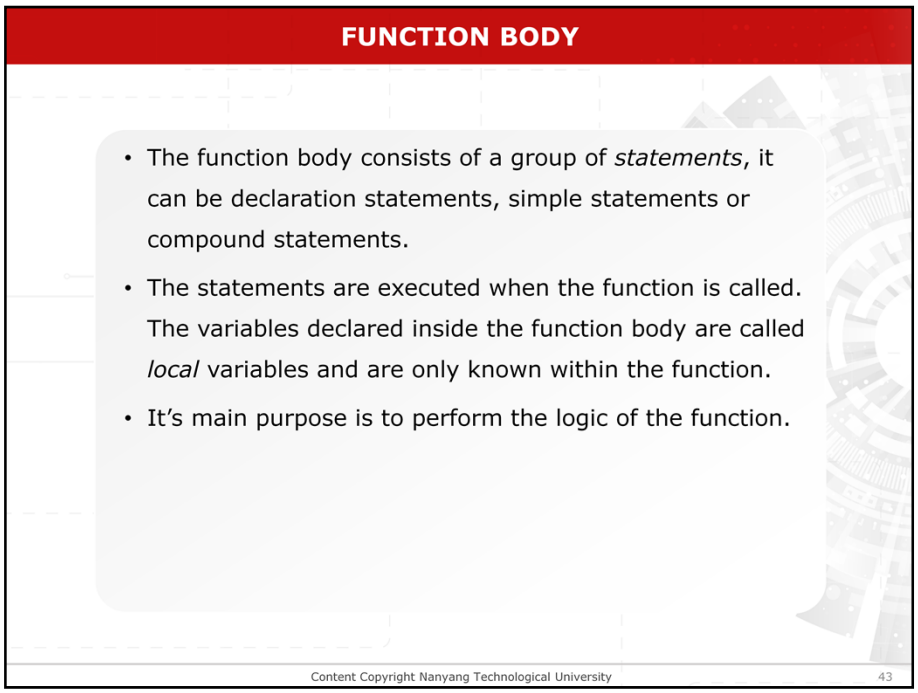


Function Body

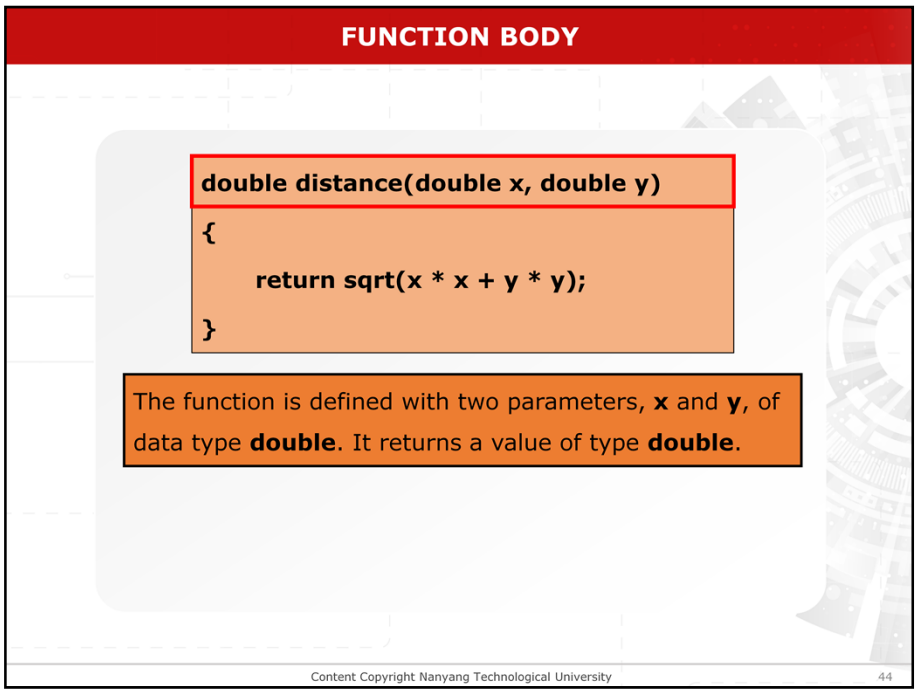
The function body consists of a group of *statements*, it can be declaration statements, simple statements or compound statements.



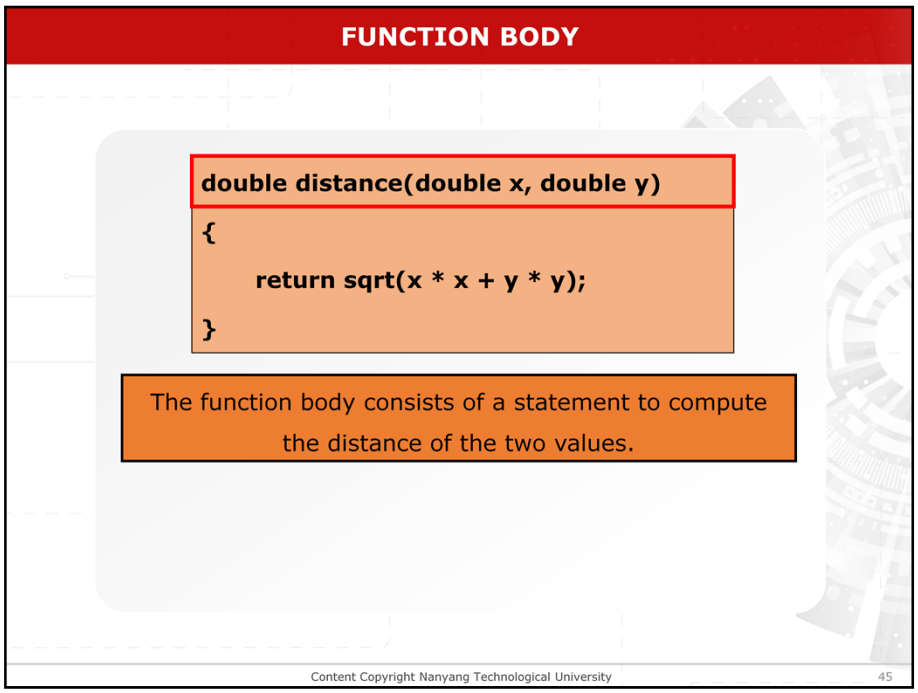
The statements are executed when the function is called. The variables declared inside the function body are called *local* variables and are only known within the function.



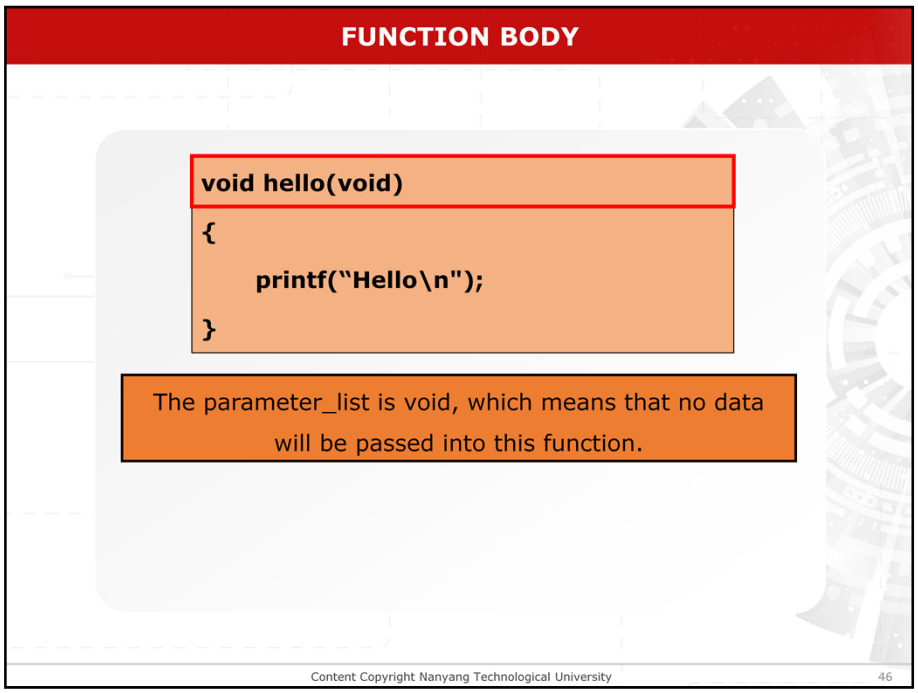
It's main purpose is to perform the logic of the function.



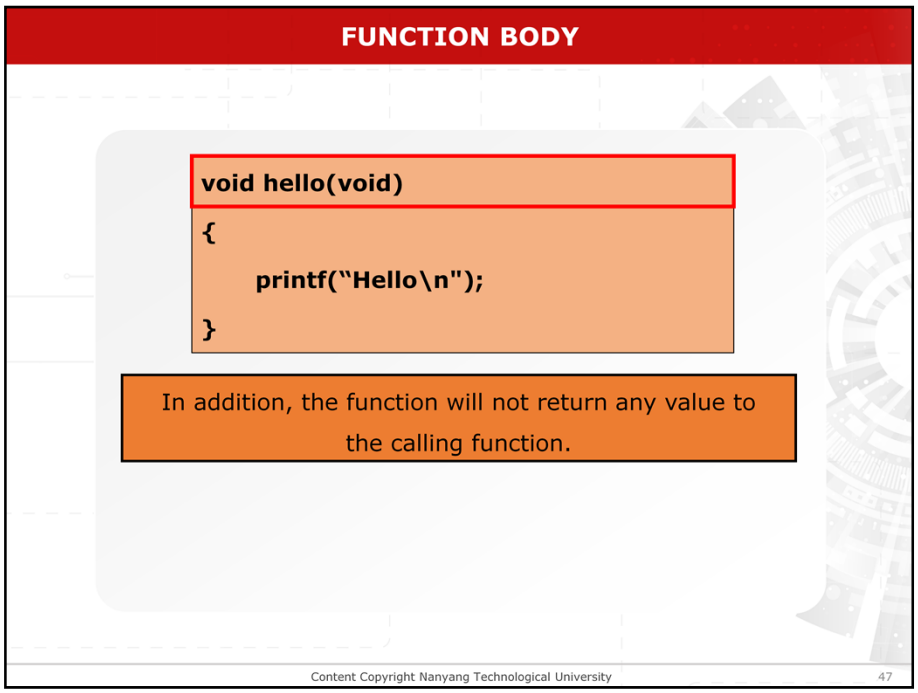
For example, in the **distance()** function shown here, the function is defined with two parameters, **x** and **y**, of data type **double**. It returns a value of type **double**.



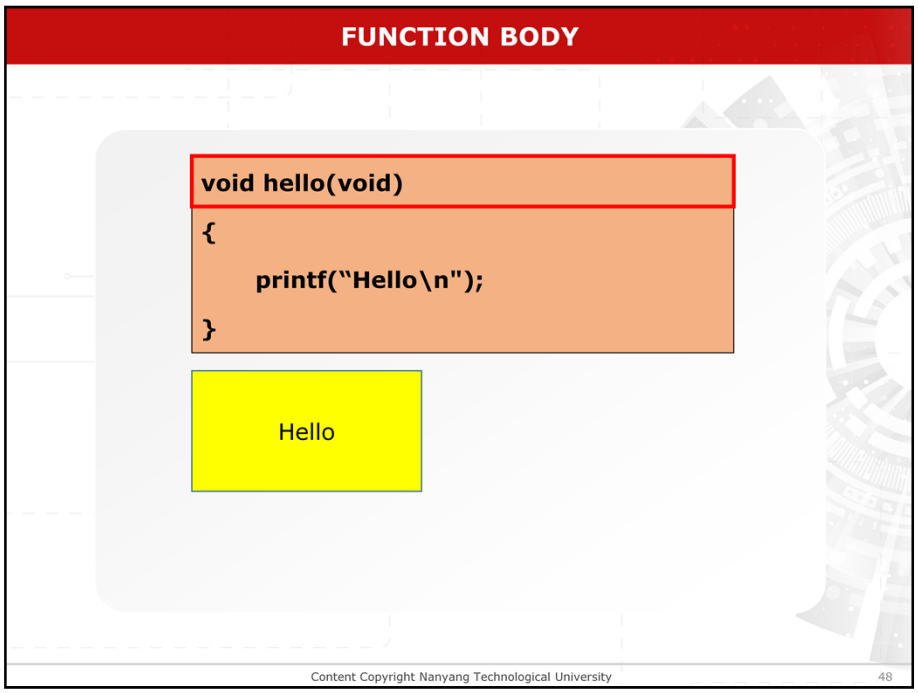
The function body consists of a statement to compute the distance of the two values.



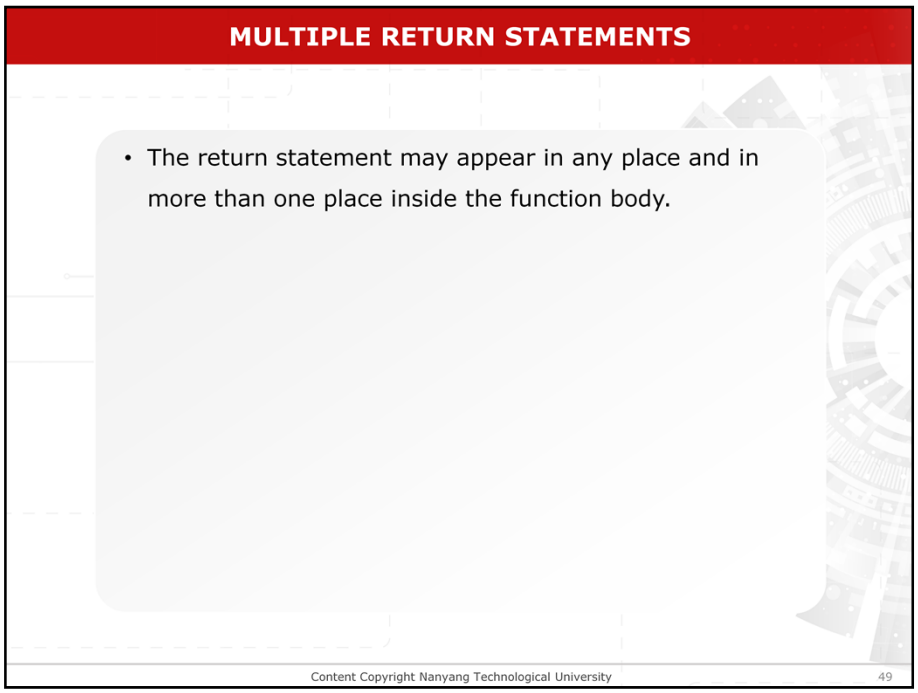
In the hello() function shown here, the **parameter_list** is **void**, which means that no data will be passed into this function.



In addition, the function will not return any value to the calling function. The function body just prints out the message **hello** on the screen.

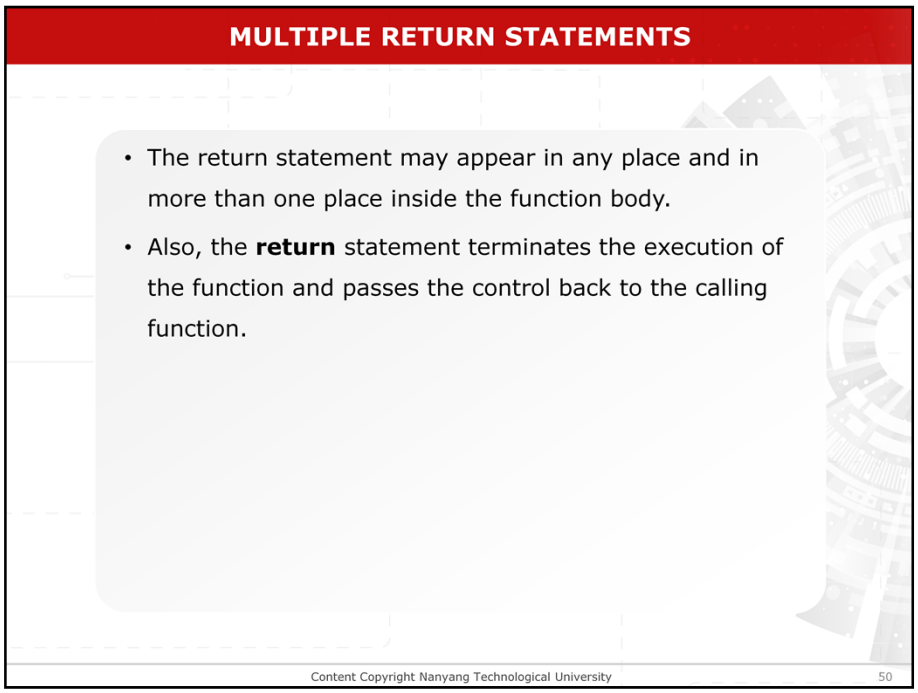


The function body just prints out the message **hello** on the screen.



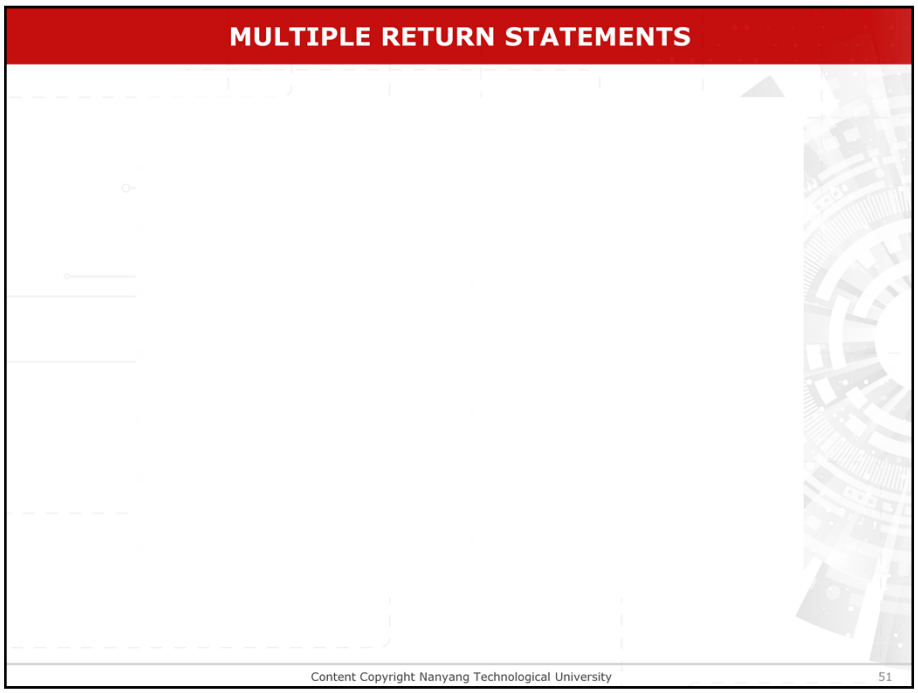
Multiple Return Statements

The **return** statement may appear in any place and in more than one place inside the function body.



Multiple Return Statements

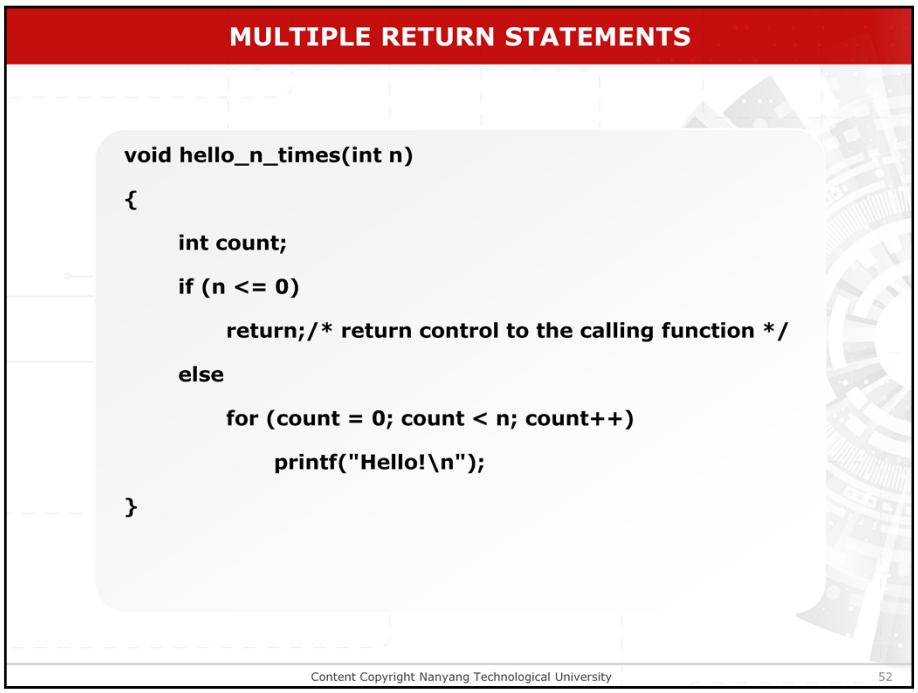
The **return** statement may appear in any place and in more than one place inside the function body.



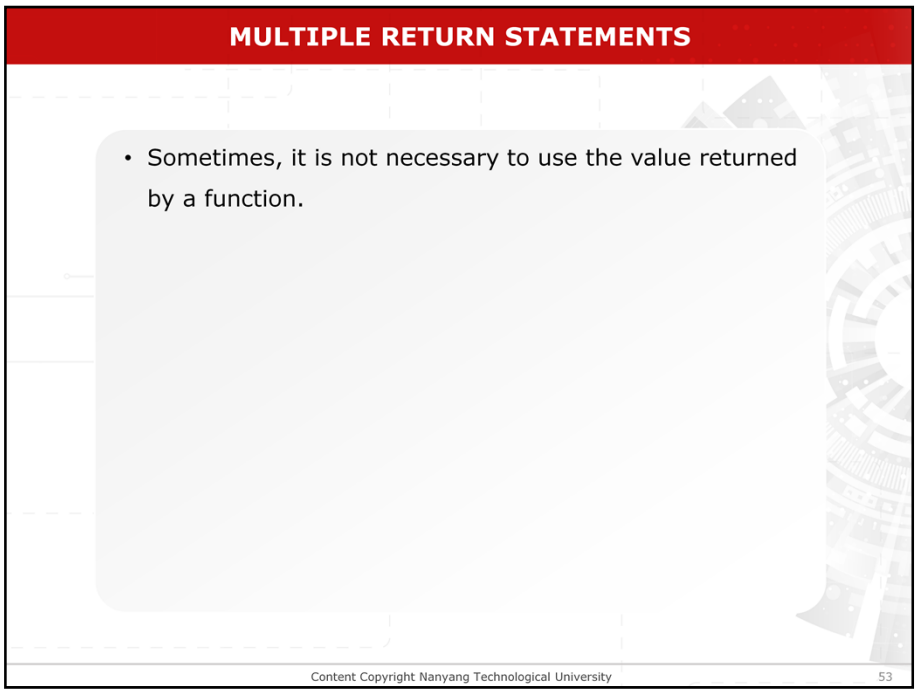
Multiple Return Statements

The **return** statement may appear in any place and in more than one place inside the function body. Also, the **return** statement terminates the execution of the function and passes the control back to the calling function. In the program, the function **fact()** has **return** statements in various locations in the function body. If **n** is less than 0, then an error message is displayed, and the function returns the value 0 to the calling function . If **n** equals to 0, then the function returns a value 1. If **n** is greater than 0, then the factorial of **n** is evaluated using a **for** loop, and the result is returned.

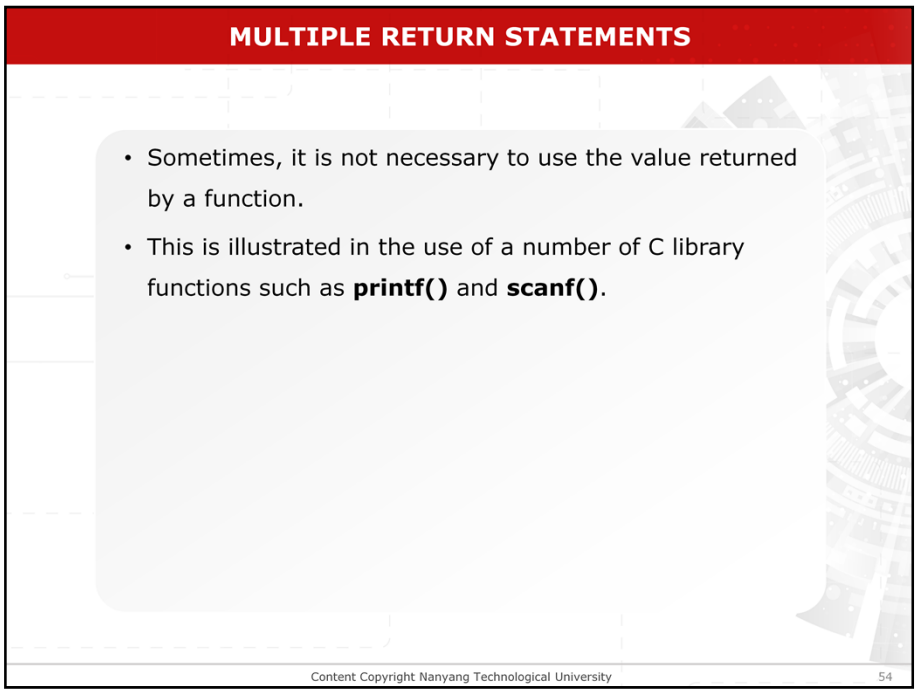
The value returned by the function can be assigned to a variable or used in other functions. In the program, the **main()** function calls the function **fact()**. The value 24 returned by the function **fact()** is then assigned to the variable **num**.



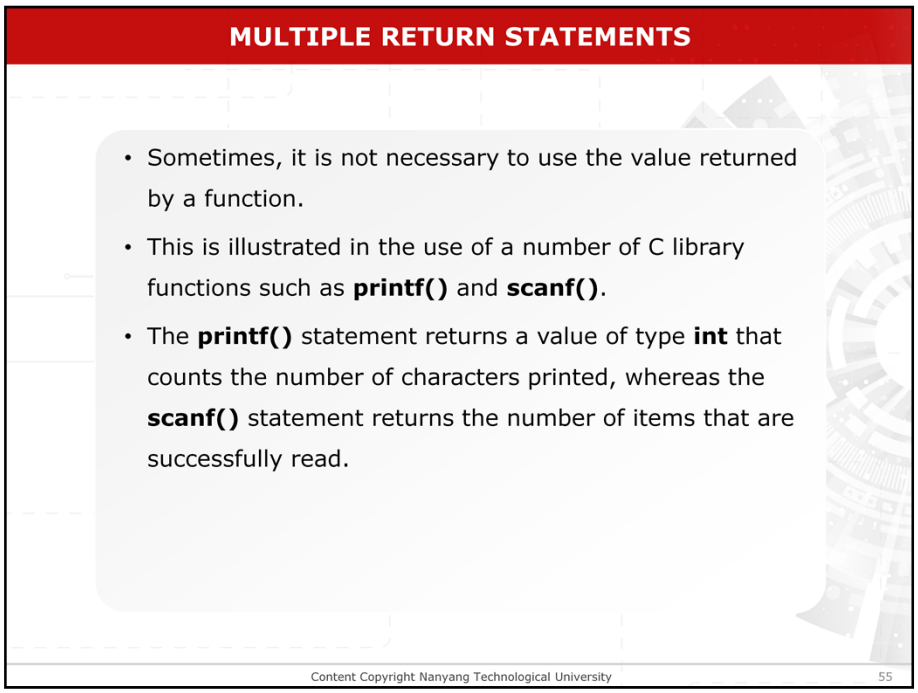
A type **void** function may also have a **return** statement to terminate the function. However, it must not have a **return** expression.
If the function does not have a **return** statement, then the control will be passed back to the calling function when the closing brace of the function body is encountered.
Sometimes, it is not necessary to use the value returned by a function. This is illustrated in the use of a number of C library functions such as **printf()** and **scanf()**. The **printf()** statement returns a value of type **int** that counts the number of characters printed, whereas the **scanf()** statement returns the number of items that are successfully read. However, if we do not require this information, we do not need to use the return value returned by these functions.



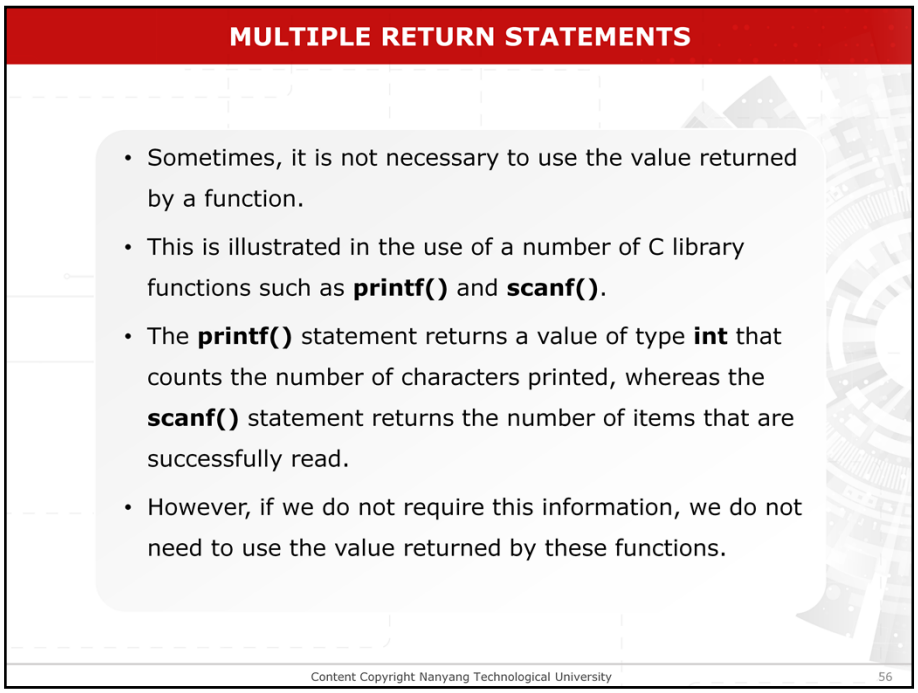
Sometimes, it is not necessary to use the value returned by a function.



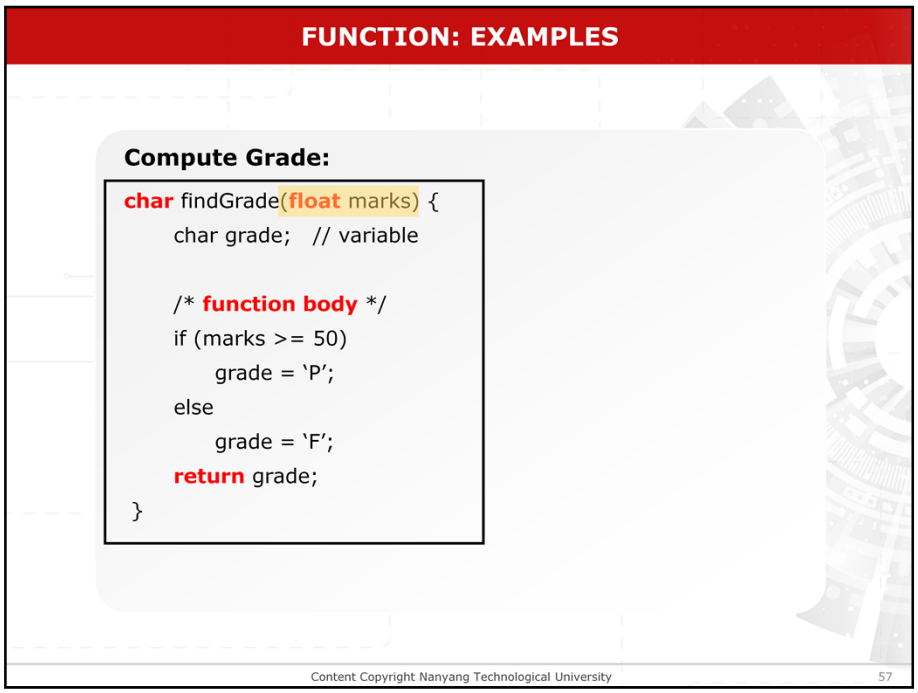
This is illustrated in the use of a number of C library functions such as **printf()** and **scanf()**.



The **printf()** statement returns a value of type **int** that counts the number of characters printed, whereas the **scanf()** statement returns the number of items that are successfully read.

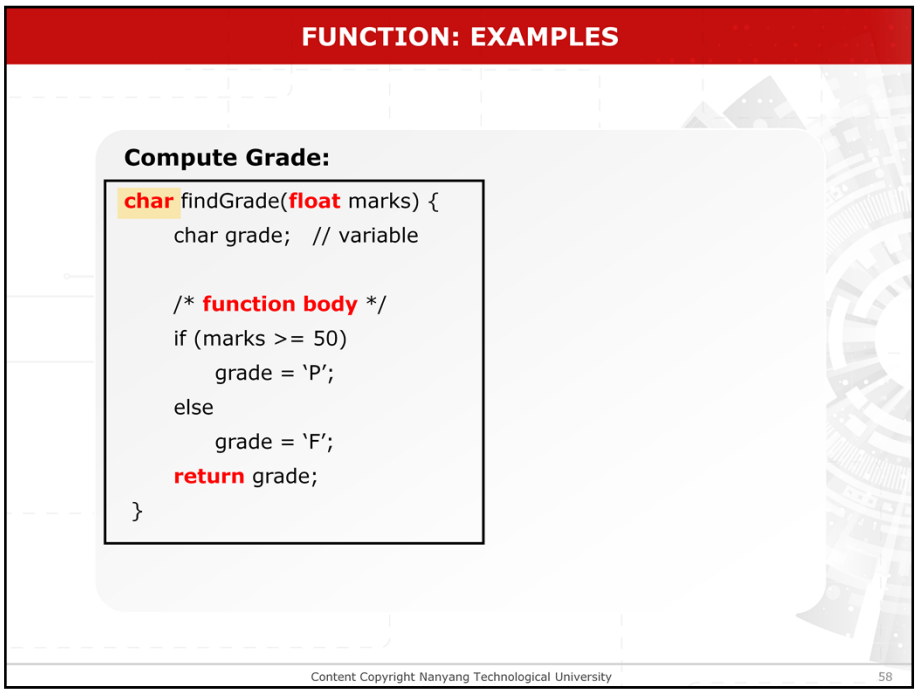


However, if we do not require this information, we do not need to use the value returned by these functions.

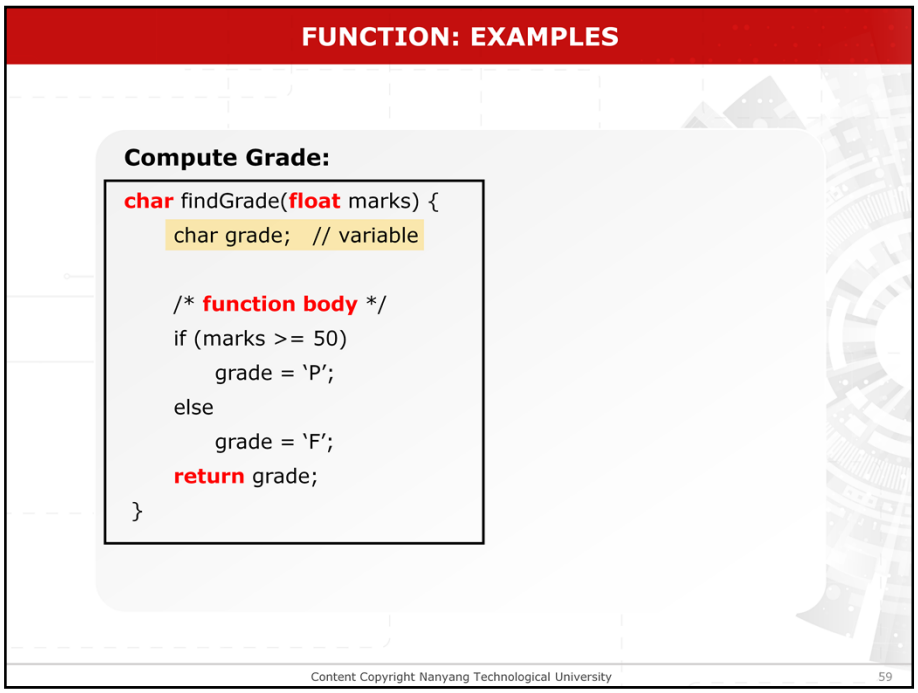


Function: findGrade()

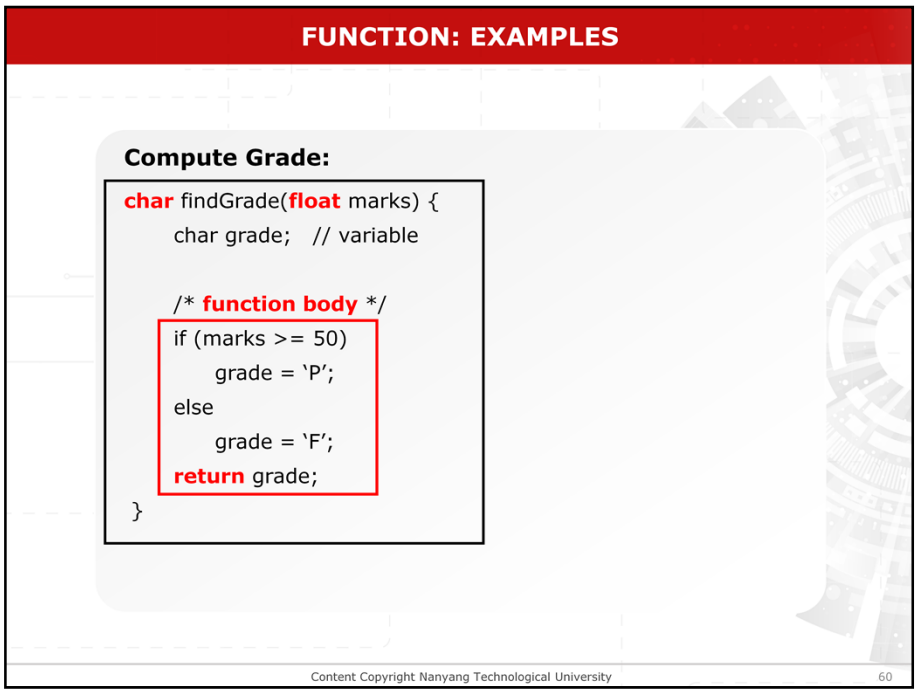
The function **findGrade()** expects a parameter of type **float**



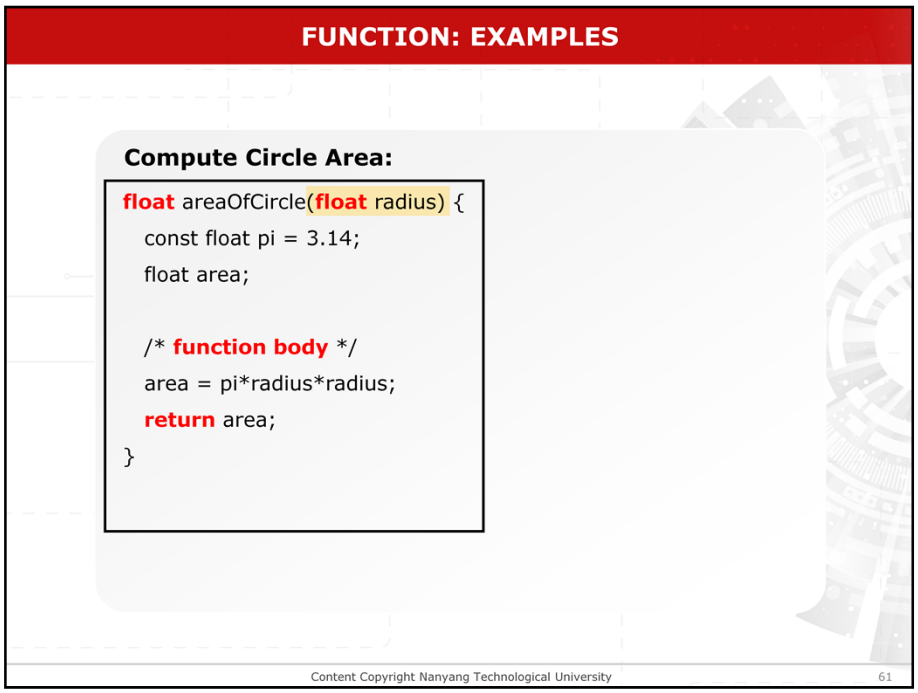
and returns a value of type **char**.



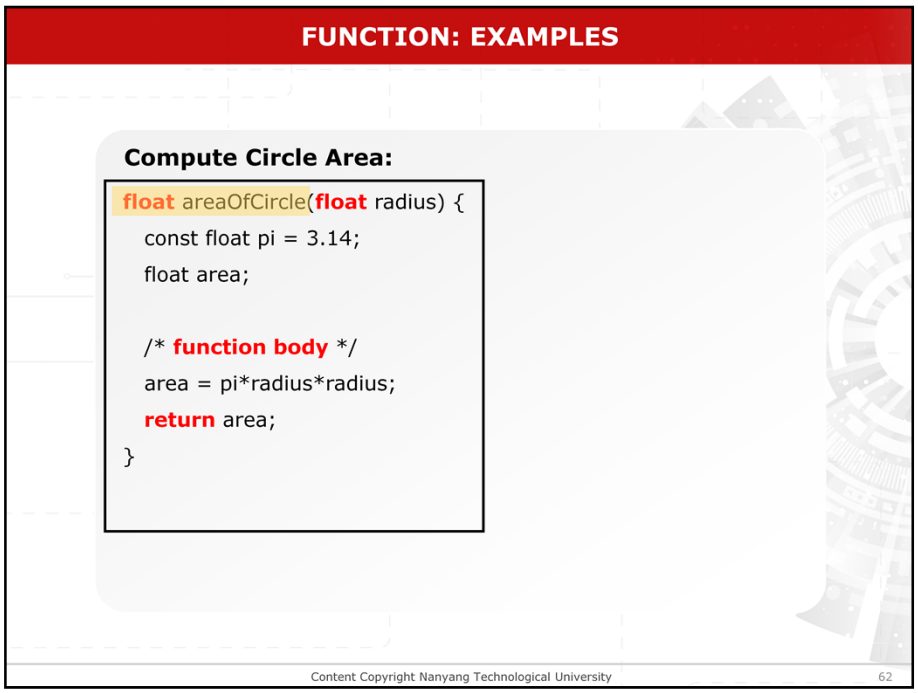
There is one variable defined in the function. The variable is a local variable and can only be accessed within the function.



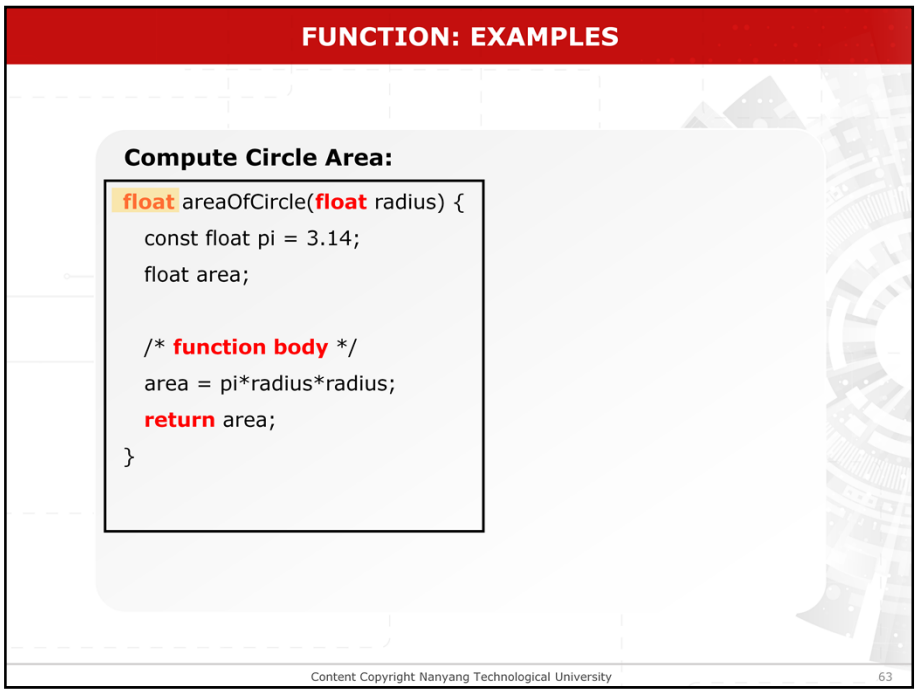
The variable is created when the function is called, and destroyed when the function ends.



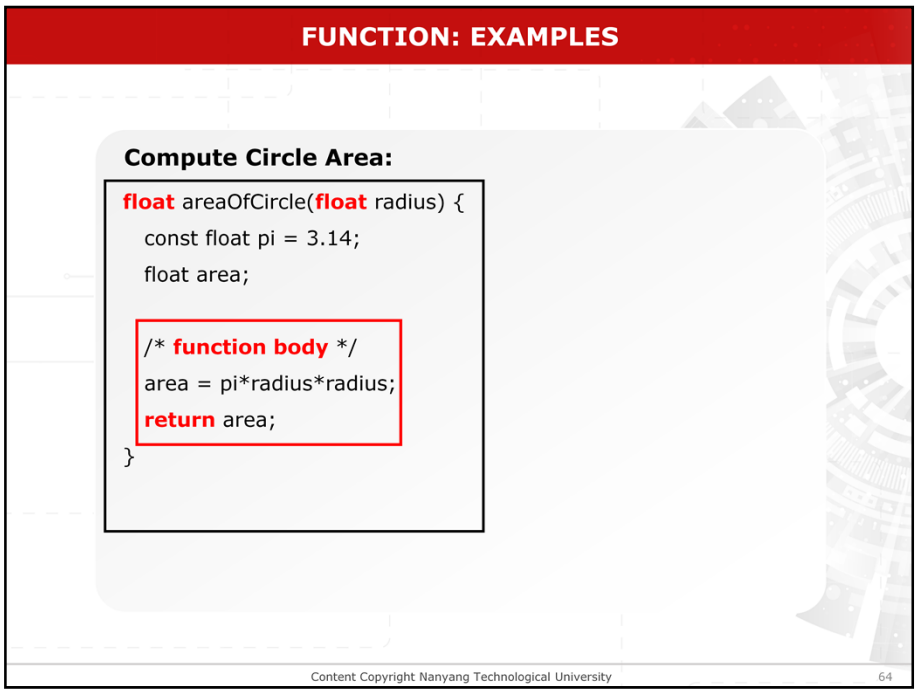
Another example to cumpute circle area is shown here. The function **areaOfCircle()** expects one parameter of type **float**.



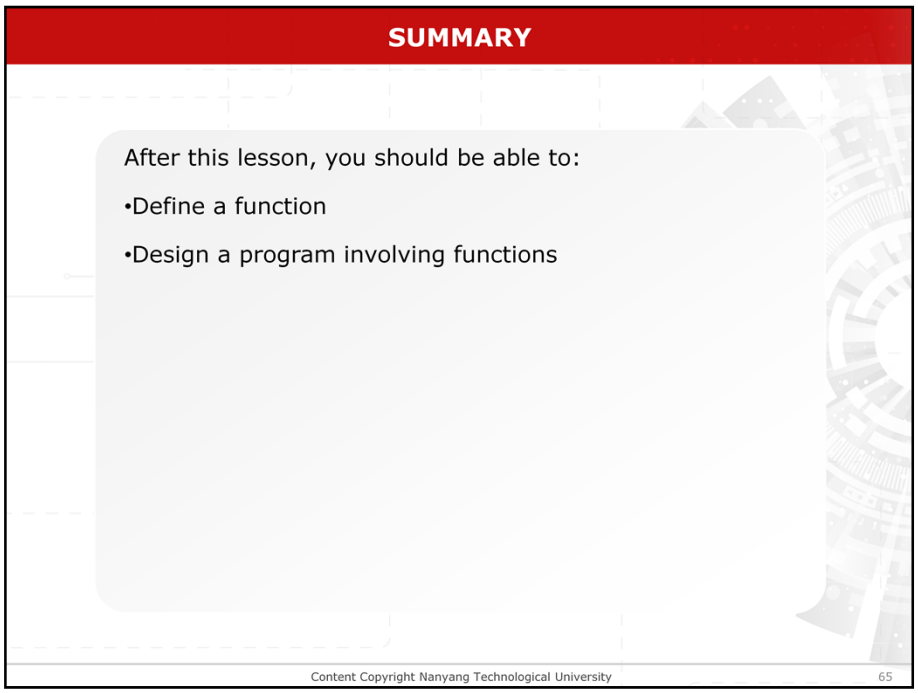
It returns a value of type **float**.



A local variable **area** is also declared in the function



This variable is only accessible within the function **areaOfCircle()**. It is also created when the function is called, and will be destroyed when the function exits.



In summary, after viewing this video lesson, your should be able to do the listed.