Tabla de contenido

[Welcome to Android Basics in Kotlin 2](#_Toc90365416)

[Build your first Android app in Kotlin 3](#_Toc90365417)

[Write your first program in Kotlin 4](#_Toc90365418)

[Before you begin 4](#_Toc90365419)

[Run your first program in Kotlin 5](#_Toc90365420)

[Modify your program 5](#_Toc90365421)

[Extend your program 5](#_Toc90365422)

[Solution code 5](#_Toc90365423)

[Summary 5](#_Toc90365424)

[Learn more 5](#_Toc90365425)

[Practice on your own 6](#_Toc90365426)

[Create a birthday message in Kotlin 6](#_Toc90365427)

[Quiz 6](#_Toc90365428)

# Welcome to Android Basics in Kotlin

https://youtu.be/OpQ3VzzgE0g

Meet the team and learn what you’ll need to begin developing Android apps in Kotlin.

# Build your first Android app in Kotlin

https://youtu.be/7VG8b7FtTo0

Learn more about the course goals and requirements in this introduction to Unit 1. To check if your computer can download Android Studio, see the system requirements at the bottom of this page: https://developer.android.com/studio.

# Write your first program in Kotlin

Create your first program in Kotlin.

## Before you begin

In this codelab, you are going to write your first program in the Kotlin language using an interactive editor that you can run from your browser.

You can think of a program as a series of instructions for the system to perform some action. For example, you could write a program that creates a birthday card. In that program, you could write an instruction to print congratulatory text or calculate someone's age from their birth year.

Just like you use human language to communicate with another person, you use a programming language to communicate with the operating system of your computer. Fortunately, programming languages are less complex than human languages and quite logical!

Android apps are written in the Kotlin programming language. Kotlin is a modern language created to help developers write code efficiently and with as few errors as possible.

Learning to create an app and learning the basics of programming at the same time will be challenging, so we are going to start you off with a bit of programming before getting into app creation. Becoming comfortable with some programming basics first is not only an important step towards creating apps, it is also going to make it easier to create your first app later in this course.

Code editors are tools that help you write code, in the same way a word processor (like Google Docs) helps you create text documents. In this codelab, you are using an interactive Kotlin editor within your browser. This means that you do not have to install any software to take your first step towards app development.

**PREREQUISITES**

Use interactive websites in your web browser.

**WHAT YOU'LL LEARN**

How to create, change, understand, and run a minimal Kotlin program that displays a message.

**WHAT YOU'LL BUILD**

A program in the Kotlin programming language that displays a message when you run it.

**WHAT YOU NEED**

A computer with a modern web browser, such as the latest version of Chrome.

Internet access on your computer.

## Run your first program in Kotlin

In this task, you will use an editor on a website to start programming in the Kotlin language right away.

**USE AN INTERACTIVE CODE EDITOR**

Instead of installing software on your computer, you can use a web-based tool to create your first program.

1. In your browser, open https://developer.android.com/training/kotlinplayground. This opens a browser-based programming tool.
2. You should see a page similar to the screenshot below, with a code editor in the middle.

Interfaz de usuario gráfica, Texto, Aplicación, Correo electrónico

Descripción generada automáticamente

This is the program code in the editor:

fun main() {

println("Hello, world!")

}

**RUN THE PROGRAM CODE**

Running a program that you created is not much different than running a program such as a word processor on your computer. The difference is that when you run a program to accomplish a task, or play a game, you primarily care about what the program can do for you, and you don't concern yourself with the code that makes it work. When you are programming, you get to see and work with the actual code that makes the magic happen.

Let's see what this program does!

1. In the editor, in the top-right corner, find the white or **green triangle** and click it to run the program.
2. Look at the pane at the bottom.

Hello, world!

Notice Hello, world! printed, like in the image above. So now you know what this program does: It prints, or outputs, a hello world message.

**Compilation** is a process that translates the Kotlin program code into a form that the system can run. If compilation completes successfully, there are no errors in the program that would keep it from running. If there are problems, they will appear in pane at the bottom.

## Modify your program

**CHANGE THE HELLO WORLD CODE**

Let's change the program to make it do something a little different.

1. Change the "Hello, world!" text to say "Happy Birthday!".
2. Run your program by clicking the blue or green run button at the top right.
3. At the bottom, you should now see Happy Birthday! printed, as shown below.

Happy Birthday!

**HOW DOES IT WORK?**

How is this done? This seems like a lot of code to just print something!

Well, if you wanted a friend to write "Hello, world!" on a piece of paper, there is a lot of implied information. If you just tell them, "Write ‘Hello world!' on this piece of paper", they are going to make assumptions about the information you left out. For example, they are going to assume they need to use a pen, and that you want them to write it using letters! The computer does not make these assumptions, so you have to give precise instructions that include every step.

Just like the English language has structure, so does a programming language. If you've ever learned another language, you know the challenge of learning the grammar, the spelling, perhaps a new alphabet of symbols, and the vocabulary. Learning to program has similar challenges, but fortunately, it is less complex and a lot more logical than learning, for example English.

**UNDERSTAND THE PARTS OF THE PROGRAM**

Now, take a look at the code. Each piece of this program serves a specific purpose, and you need all the pieces in order to be able to run the program. Let's start with the first word.

fun

* **fun** is a word in the Kotlin programming language. **fun** stands for function. A function is a section of a program that performs a specific task.

*Note: Kotlin has many special words with very specific meanings. As you learn to program in the Kotlin language, you will learn these words. They are often called keywords or reserved words.*

fun main

* **main** is the name of this function. Functions have names, so they can be distinguished from each other. This function is called main, because it is the first, or main, function that is called when you run the program. **Every Kotlin program needs a function named main**.

fun main()

* The function name is always followed by **()** two parentheses.
* Inside the parentheses, you can put information for the function to use. This input to the function is called "arguments" or **args** for short. You will learn more about arguments later.

fun main() {}

* Notice the pair of curly braces **{}** after the parentheses. Inside a function is code that accomplishes a task. These curly braces surround those lines of code.

Look at the line of code between the curly braces:

println("Happy Birthday!")

This line of code prints the Happy Birthday! text.

* **println** tells the system to print a line of text.
* Inside the parentheses you put the text to be printed.
* Notice that the text to be printed is surrounded by quotes. This tells the system that everything inside the quotation marks should be printed exactly as given.

To actually print the text, this whole **println** instruction has to be inside the **main** function.

So, there it is. The smallest Kotlin program.

fun main() {

println("Happy Birthday!")

}

## Extend your program

**PRINT MORE THAN ONE MESSAGE**

Great job! You printed one line of text using the **println() function**. However, you can put as many lines of instructions inside a function as you want or need to get a task accomplished.

1. Copy the line **println("Happy Birthday!")** and paste it two more times below it. Make sure your pasted lines are inside the curly braces of the **main** function.
2. Change one text to be printed to someone's name, say "Jhansi".
3. Change the other text to be printed to "You are 25!".

Your code should look like the code below.

fun main() {

println("Happy Birthday!")

println("Jhansi")

println("You are 25!")

}

What would you expect this code to do when it runs?

1. Run your program to see what it does.
2. Go to the output pane, and you should see 3 lines printed in the console window, as shown below.

Happy Birthday!

Jhansi

You are 25!

Nice work!

**DEALING WITH ERRORS**

Making mistakes while programming is normal, and most tools will give you feedback to help you fix mistakes. In this step, create a mistake to see what happens.

1. In your program, remove the quotes around the text Jhansi, so that line looks as shown below.

println(Jhansi)

1. Run your program. You should see **Jhansi** printed in red, and an exclamation mark next to the line of code you changed, to show you where there is an error.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Look at the output pane. It shows a message with the same exclamation mark icon and the word Error. What follows is a description of the error in your code.

Logotipo

Descripción generada automáticamente con confianza media

1. This message, **Unresolved reference: Jhansi**, tells you what the system thinks is the error in the code. Even if you don't know what the error message means, you may be able to figure out what's wrong. In this case, you know that the println() instruction prints text. You learned earlier that the text has to be between quotes. If the text is not quoted, that is an error.
2. Go ahead and add the quotes back in.
3. Run your program to make sure it works again.

Congratulations, you have run and changed your first Kotlin program!

## Solution code

This is the complete code of the program you worked on in this codelab.

fun main() {

println("Happy Birthday!")

println("Jhansi")

println("You are 25!")

}

## Summary

* *https://developer.android.com/training/kotlinplayground* is an interactive code editor on the web where you can practice writing Kotlin programs.
* All Kotlin programs need to have a **main()** function: **fun main() {}**
* Use the **println()** function to print a line of text.
* Place text you want to print between double quotes. For example **"Hello"**.
* Repeat the **println()** instruction to print multiple lines of text.
* Errors are marked red in the program. There is an error message in the output pane to help you figure out where the error is and what might be causing it.

## Learn more

* https://developer.android.com/training/kotlinplayground
* Vocabulary for Android Basics in Kotlin

## Practice on your own

*Note: Practice problems are optional. They provide an opportunity for you to practice what you've learned in this codelab.*

**DO THE FOLLOWING:**

1. Change the **println()** instructions to **print()**.
2. Run your program. What happens?

**Hint**: The **print()** instruction just prints the text without adding a line break at the end of each string.

1. Fix the text so that each part of the message is on its own line.

**Hint**: Use **\n** inside the text to add a line break. For example, **"line \n break"**. Adding a line break changes the output as shown below.

**Hint**: You can print an empty line by supplying no text: **println("")**.

**CODE:**

fun main() {

println("no line break")

println("")

println("with line \n break")

}

**OUTPUT:**

no line break

with line

break

**CHECK YOUR WORK:**

Here is one possible solution:

fun main() {

print("Happy Birthday!\n")

print("Jhansi\n")

print("You are 25!")

}

# Create a birthday message in Kotlin

Create a short program in Kotlin using functions and loops to print a happy birthday message.

## Introduction

In this codelab you will create a short Kotlin program that prints a birthday message with a cake and a birthday banner.

**PREREQUISITES**

How to open and edit code in https://developer.android.com/training/kotlinplayground, a browser-based Kotlin programming tool.

Understand the **"Hello world!"** program from the Write your first Kotlin program codelab.

How to use **println()** to write text to the console of the online Kotlin code editor.

**WHAT YOU WILL LEARN**

How to print more complex text from your program.

How to do basic math in Kotlin and store the results in variables for later use.

How to create a function to print the same string several times.

How to create a loop that prints a text snippet multiple times.

**WHAT YOU WILL BUILD**

You will create a short program that you can use to print birthday messages, a text-based picture of a cake, and a banner.

**WHAT YOU NEED**

A computer with internet access and a modern web browser, such as the latest version of Chrome.

## Create a birthday message in Kotlin

**SET UP YOUR STARTER CODE**

1. In your browser, open https://developer.android.com/training/kotlinplayground. This opens a browser-based Kotlin programming tool.
2. Inside the **fun main()** function, replace the **"Hello, world!"** text with **"Happy Birthday, Rover!"**.
3. Below that, still inside the curly braces, add two more lines to print: **"You are already 5!"** and **"5 is the very best age to celebrate!"**.

Your finished code should look like this.

fun main() {

println("Happy Birthday, Rover!")

println("You are already 5!")

println("5 is the very best age to celebrate!")

}

1. Run your code.
2. Verify that the output pane shows Happy Birthday, Rover! and below that, You are already 5! and 5 is the very best age to celebrate!

Happy Birthday, Rover!

You are already 5!

5 is the very best age to celebrate!

**ADD A BIRTHDAY CAKE**

A birthday message needs a birthday-themed picture. Like, a cake. You can add a cake to your birthday message by printing additional lines that use the letters and symbols on your keyboards and **println()**.

Continue from the solution code above.

1. In your code, between the two println() statements for Happy Birthday and You are already 5, add the following lines of print statements, as shown below. This creates a cake. The last println() statement has no text between the quotes, which prints an empty line.

println(" ,,,,, ")

println(" ||||| ")

println(" =========")

println("@@@@@@@@@@@")

println("{~@~@~@~@~}")

println("@@@@@@@@@@@")

println("")

To help others understand your code, you can add a comment before you print the cake. If you run your code, the output won't look any different, because comments are just information for you and other developers, not commands for the system. An inline comment starts with **//** followed by text, as shown below.

// This is a comment line

// This is another comment

1. Add a comment before you print the cake: **// Let's print a cake!**.
2. Add a comment before printing the empty line: **// This prints an empty line**.

Your code should look like the code below.

fun main() {

println("Happy Birthday, Rover!")

// Let's print a cake!

println(" ,,,,, ")

println(" ||||| ")

println(" =========")

println("@@@@@@@@@@@")

println("{~@~@~@~@~}")

println("@@@@@@@@@@@")

// This prints an empty line.

println("")

println("You are already 5!")

println("5 is the very best age to celebrate!")

}

*Tip: Notice that we added some whitespace (blank lines) in the code to separate the sections of the code. This makes the code more readable. You can add empty lines anywhere you find it helpful.*

1. Run your code, and the output should look like below.

Happy Birthday, Rover!

,,,,,

|||||

=========

@@@@@@@@@@@

{~@~@~@~@~}

@@@@@@@@@@@

You are already 5!

5 is the very best age to celebrate!

## Create and use variables

**STORE ROVER'S AGE IN A VARIABLE**

1. In your finished code so far, notice how you are repeating the same age number twice.

Instead of repeating this number, you could store it in one place, as a variable. It's like putting your number into a box and giving it a name. Then you can use this variable name any time you need the value. And, if the age changes, you only have to change your program in one place. By changing the variable, the correct value for the age is printed everywhere the variable is used.

1. In your program, as the first line of code inside the **main()** function, add the following code to create a variable called **age**, with a value of 5, as shown below. (You must put this line before the **println()** statements).

val age = 5

This line means:

* **val** is a special word used by Kotlin, called a keyword, indicating that what follows is the name of a variable.
* **age** is the name of the variable.
* **=** makes the value of **age** (on its left) be the same as the value on its right. In math, a single equal sign is used to assert that the values on each side are the same. In Kotlin, unlike in math, a single equal sign is used to assign the value on the right to the named variable on the left.

A developer would say it like this: This line declares a variable named **age** whose assigned value is **5**.

***Important****: A variable declared using the val keyword can only be set once. You cannot change its value later in the program.*

*You can declare a changeable variable with the var keyword, which you will do in a later codelab.*

To use a variable inside a print statement, you need to surround it with some symbols that tell the system that what comes next is not text, but a variable. Instead of printing text, the system needs to print the value of the variable. You do this by putting your variable inside curly braces preceded by a dollar sign, like in the example below.

${variable}

1. In your code, replace the number 5 in both print statements with the **age** variable as shown below.

println("You are already ${age}!")

println("${age} is the very best age to celebrate!")

1. Run your code, and both messages should show the same age.
2. Change the value of the variable to something different. For example, you could show Rover's age in days instead of years. To do this, multiply the age with 365, omitting leap years. You can do this calculation right when you create the variable, as shown below.

val age = 5 \* 365

1. Run your code again, and notice that both messages now show the age in days.

Happy Birthday, Rover!

,,,,,

|||||

=========

@@@@@@@@@@@

{~@~@~@~@~}

@@@@@@@@@@@

You are already 1825!

1825 is the very best age to celebrate!

1. **[Optional]** Change the text of the print messages to work better with days. For example, change them to read:

You are already 1825 days old!

1825 days old is the very best age to celebrate!

**PUT TEXT INTO A VARIABLE**

You can not only put numbers into variables, but also text.

1. Below the variable for **age**, add a variable called **name** for the name of the birthday person and set its value to **"Rover"**.

val name = "Rover"

1. Replace the name **Rover** in the birthday message with the variable, as shown below.

println("Happy Birthday, ${name}!")

And you can have more than one variable in a print statement.

1. Add **Rover** to the age message, using the **name** variable, as shown below.

println("You are already ${age} days old, ${name}!")

Your completed code should look similar to this.

fun main() {

val age = 5 \* 365

val name = "Rover"

println("Happy Birthday, ${name}!")

// Let's print a cake!

println(" ,,,,, ")

println(" ||||| ")

println(" =========")

println("@@@@@@@@@@@")

println("{~@~@~@~@~}")

println("@@@@@@@@@@@")

// This prints an empty line.

println("")

println("You are already ${age} days old, ${name}!")

println("${age} days old is the very best age to celebrate!")

}

Congratulations! You can now create messages with text, graphics created from symbols, use variables to store numbers and text, and print text using variables.

## Print a birthday banner with a border

In this task you will create a birthday banner, and then learn how to simplify that code using techniques for repeating and reusing code, and why that is a good thing.

**CREATE A STARTER BIRTHDAY BANNER**

1. In https://developer.android.com/training/kotlinplayground, place your cursor somewhere inside the code.
2. Right-click to open the menu and choose **Select All**.
3. Press the backspace or delete key to delete all the code.
4. Copy-paste the code below into the editor.

fun main() {

println("=======================")

println("Happy Birthday, Jhansi!")

println("=======================")

}

1. Run your program to see a banner printed in the Console.

=======================

Happy Birthday, Jhansi!

=======================

**CREATE A FUNCTION FOR PRINTING THE BORDER**

The code that you just pasted and ran is a function called **main()** that contains three print statements. When you press the **Run** button, the system executes the function and all the code inside it.

***Recap***

*In a previous codelab, you learned that:*

* *A function is a section of a program that performs a specific task.*
* *The fun keyword marks some code as a function.*
* *After the fun keyword follows the name of the function, parentheses for optional input to the function (arguments), and curly braces.*
* *Your code for printing text always went inside those curly braces.*

Your Kotlin program always has to have a **main()** function. In addition, you can create and use functions of your own. Just like variables help you avoid duplicating work, functions can help you avoid writing the same code multiple times. In your code, the print statements for the top and bottom of the banner are exactly the same. Let's create and use a function for printing those borders.

1. In the editor, below the **main()** function, insert an empty line, just to give you some room to work. The system ignores empty lines, and you can insert them wherever they are helpful for organizing your code.
2. Create a function. Start with the **fun** keyword, followed by a name, **printBorder**, a pair of parentheses **()**, and a pair of curly braces **{}**, as shown below.

fun printBorder() {}

A word about naming functions.

* Notice how the name of the function **printBorder** starts with a lower-case letter and a verb. Function names almost always start with a lowercase letter, and a verb, and the name should describe what the function does. Like: **print()** or here, **printBorder()**.
* Notice also that the second word inside the name starts with uppercase. This style is called "camel case", and makes names much easier to read. Some more name examples are **drawReallyCoolFancyBorder** and **printBirthdayMessage**.

***Note****: Naming functions like this is a "coding convention", an agreement among developers about how to format code. Formatting all the code similarly makes it easier to read and learn from code that other programmers have written. When you see code from other Android developers, it will usually be formatted using these conventions.*

*To learn more about formatting code, you can find all the conventions in an official style guide at https://developer.android.com/kotlin/style-guide. There is a lot in that guide, but if you are curious, go take a look.*

1. Put the closing brace **}** of the **printBorder** function on a new line and add an empty line in between the two curly braces, to give you room to add more code. Having the closing brace **}** on its own line makes it easier to see where the function ends.
2. Inside the **main()** function, copy the print statement for the border and paste it between the curly braces of the **printBorder()** function.

Your finished **printBorder()** function should look like this.

fun printBorder() {

println("=======================")

}

To use or call, a function, use its name with parentheses. Notice that this is how you have been using **println()**! So to use the **printBorder** function, call **printBorder()** anywhere in your code that you need to.

1. In the **main()** function, replace the lines of code that print a border line using **println()** with calls to the **printBorder()** function. Your finished code should look like this.

fun main() {

printBorder()

println("Happy Birthday, Jhansi!")

printBorder()

}

fun printBorder() {

println("=======================")

}

1. Run your code to make sure all is working as before.

Note that changing the code to make it better or easier to work with without changing the output is called "refactoring".

**REPEAT A BORDER PATTERN**

Looking at the border line, it's really the same symbol all over again. So, instead of saying:

"Print this **string** of 23 symbols"

you could say,

"Print this **1 symbol** 23 times".

In code, you do this by using a **repeat()** statement.

1. In the **printBorder()** function, use a **repeat()** statement to print the equal sign 23 times.
2. Instead of using **println()**, use **print()**, so that you do not jump to a new line after printing each "=".

Here is the code. You now have a single instruction to print the equal sign, and to repeat that instruction 23 times, you use a **repeat()** statement.

fun printBorder() {

repeat(23) {

print("=")

}

}

* The **repeat()** statement starts with the word **repeat**, followed by **()**. This kind of statement is referred to as a ‘loop' because you are repeating or looping over the same code multiple times. You will learn about other ways of creating loops later.
* Inside the parentheses **()** is the number of repetitions,
* followed by curly braces **{}**,
* and inside the curly braces **{}**, is the code to repeat.

1. Within the **printBorder()** function, after the closing curly brace } of the **repeat()** statement, which is after you are done printing the border line, add a **println()** statement to print a newline.

Your code should now be like this.

fun printBorder() {

repeat(23) {

print("=")

}

println()

}

The code in the **main()** function does not change, and your entire program should look like this.

fun main() {

printBorder()

println("Happy Birthday, Jhansi!")

printBorder()

}

fun printBorder() {

repeat(23) {

print("=")

}

println()

}

1. Run your code. The output should be the same as before, but this time, you were able to create the border by specifying the "=" symbol only once!

=======================

Happy Birthday, Jhansi!

=======================

**USE ARGUMENTS TO CHANGE THE BORDER**

What if you wanted to create borders that use different symbols, such as the ones below?

%%%%%%%%%%%%%%%%%%%%%%%

:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

You could define a separate function for each of these different symbols. However, there is a more efficient way to do this. You can reuse the function you already wrote and make it more flexible, so that it works for different kinds of symbols.

A cool thing about functions is that you can give them input using arguments. You encountered this briefly in a previous codelab, when you got acquainted with **main()**. In this step, you will add an argument to the **printBorder()** function, so that it can print any border pattern that you supply.

1. In **main()**, at the top, create a variable called **border** for the border pattern. This will hold text to repeat for the border.

val border = "%"

1. Now, pass that **border** variable into both calls to the **printBorder()** function as an argument. You do this by placing **border** inside the parentheses **()**, just like when you supplied text for **println()** to print.

Your **main()** function should look like the code below.

fun main() {

val border = "%"

printBorder(border)

println("Happy Birthday, Jhansi!")

printBorder(border)

}

The **printBorder()** function is going to take the value of this **border** as an input and figure out how to print out the full border.

1. Run your code. Your code does not execute, and instead, you see error icons next to your code.

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Look at the output panel, and there is an error message.

As before, the message indicates where the error is, and gives you a hint as to what it might be. The important part is: **Too many arguments for public fun printBorder()**. You are calling the **printBorder()** function and passing a border as input. However, the **printBorder()** function definition does not accept any input at the moment.

1. Fix this error by adding an argument for the border to the **printBorder()** function definition. See the first line of code as shown below.

fun printBorder(border: String) {

repeat(23) {

print("=")

}

println()

}

* Notice that the name of the argument is **border**.
* The name is followed by a colon **:**
* and the word **String**, which is a description of what kind, or type, of argument this is.

A **String** is a piece of text made up of characters surrounded by quotes. You can think of it as beads lined up on a string to form a necklace, like, characters lined up to form words and text. Specifying that the argument has to be a **String** helps the system enforce that your argument is text, and not, for example, a number.

1. Run your code. The **printBorder()** function now accepts a border **String** as input. And the code in **main()** calls **printBorder(border)** with **border** as the argument. Your code should run without errors.
2. Look at the output of your program in the **Console** and it still shows the same border as before?

=======================

Happy Birthday, Jhansi!

=======================

This is not the intended behavior! You tried to make a border with the "%" symbol, but the program is still printing a border with the "=" symbol. In the next steps, you will investigate why this is happening.

1. In the editor, notice a grayed exclamation mark. This icon indicates a warning. Warnings are about issues with your code that you need to pay attention to, but they don't keep your code from running.

Texto

Descripción generada automáticamente

1. Hover the mouse over the exclamation mark, and a message appears. It says **"Parameter 'border' is never used."** This warning explains the problem with the output. You are passing a new string for the border to the function, but you are not using it for printing.
2. Change the **printBorder()** function to use the passed in **border** instead of printing the "=". This works exactly the same as if **border** was a variable that you defined inside the function!

fun printBorder(border: String) {

repeat(23) {

print(border)

}

println()

}

1. Run your code again. The output should look like what's below.

%%%%%%%%%%%%%%%%%%%%%%%

Happy Birthday, Jhansi!

%%%%%%%%%%%%%%%%%%%%%%%

Great job, you fixed the problem! Here is your finished code.

fun main() {

val border = "%"

printBorder(border)

println("Happy Birthday, Jhansi!")

printBorder(border)

}

fun printBorder(border: String) {

repeat(23) {

print(border)

}

println()

}

You made the **printBorder()** function a lot more flexible, without adding much more code. Now you can print a border of different symbols with just a small change.

1. **[Optional]** By changing only a single line of code in the **main()** function, how would you print birthday banners like this?

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Happy Birthday, Jhansi!

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

:::::::::::::::::::::::

Happy Birthday, Jhansi!

:::::::::::::::::::::::

**MODIFY A FUNCTION TO HAVE TWO ARGUMENTS**

What if you wanted to use a different pattern that was longer than 1 character, say **"'-.\_,-'"**. You would not repeat this pattern 23 times as it would be way too long. You could repeat it maybe 4 times. To accomplish this, you could change the number of repetitions in the **repeat()** statement of **printBorder()**. However, you can do something better!

You can define a fancier border based on two things:

* The pattern to repeat (which you already have done)
* The number of times you want to repeat the pattern

You can create variables for each, the pattern and the number of repetitions, and then pass both pieces of information into the **printBorder()** function.

1. In **main()**, change the border to be the **"'-.\_,-'"** pattern.

val border = "`-.\_,-'"

1. Run your code, and notice that the pattern is now way too long.
2. In **main()**, below the definition of **border**, create a new variable named **timesToRepeat** for the number of repetitions. Set its value to 4.

val timesToRepeat = 4

1. In **main()**, when calling **printBorder()**, add the number of repetitions as a second argument. Separate the two arguments with a comma.

printBorder(border, timesToRepeat)

The **main()** function should now look like this:

fun main() {

val border = "`-.\_,-'"

val timesToRepeat = 4

printBorder(border, timesToRepeat)

println("Happy Birthday, Jhansi!")

printBorder(border, timesToRepeat)

}

As before, this code gives you an error, because you have more arguments calling **printBorder()** than in the definition of **printBorder()**.

1. Fix **printBorder()** to also accept the number of repetitions as input. Add a comma after the argument, followed by the additional argument: **timesToRepeat: Int**. The first line of your function definition now looks like shown below.

fun printBorder(border: String, timesToRepeat: Int) {

Notice:

* The comma separates the two arguments.
* **timesToRepeat** is the name of the argument,
* followed by a colon : symbol,
* and the type: **Int. timesToRepeat** is a number, so instead of making it of type **String**, you need to make it of type **Int**, which is short for an integer, a whole number.

1. Inside **printBorder()**, change **repeat** to use the **timesToRepeat** argument (instead of the number 23). Your **printBorder()** code should look like this.

fun printBorder(border: String, timesToRepeat: Int) {

repeat(timesToRepeat) {

print(border)

}

println()

}

1. Run your code. And the output looks as shown below.

`-.\_,-'`-.\_,-'`-.\_,-'`-.\_,-'

Happy Birthday, Jhansi!

`-.\_,-'`-.\_,-'`-.\_,-'`-.\_,-'

To make this output perfect, insert two spaces at the beginning of the Happy Birthday message. Then your output will be as shown below.

`-.\_,-'`-.\_,-'`-.\_,-'`-.\_,-'

Happy Birthday, Jhansi!

`-.\_,-'`-.\_,-'`-.\_,-'`-.\_,-'

Here is the final code for your banner:

fun main() {

val border = "`-.\_,-'"

val timesToRepeat = 4

printBorder(border, timesToRepeat)

println(" Happy Birthday, Jhansi!")

printBorder(border, timesToRepeat)

}

fun printBorder(border: String, timesToRepeat: Int) {

repeat(timesToRepeat) {

print(border)

}

println()

}

Congratulations! With functions, arguments, variables, and a repeat loop, you have learned fundamental building blocks that are used in almost all programs.

Take a break, and then tackle the next task below, where you will create more functions and loops, and you will gain the power to build a gigantic cake with the right number of candles with just a few lines of programming.

## Create a cake with the layers and candles

In this task, you are going to upgrade the birthday cake code to always be the right size with the right number of candles for any age.

* You will create a total of three functions for drawing a layered cake with candles.
* You will use a **repeat()** inside another **repeat()**, creating what's called a "nested loop".
* The way you will build up this code is how you can build up any program, starting with the big picture and adding detail. This is called "top-down development".
* The instructions are not as detailed for this practice, and you can refer to the finished code if you get stuck.

Here is a picture of the cake you will be baking:

,,,,,,,,,,,,,,,,,,,,,,,,

||||||||||||||||||||||||

==========================

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

And here are the instructions.

**CREATE THE MAIN() FUNCTION**

1. Reset your code in the editor to the **Hello, world!** program.
2. You can remove the argument to **main()**, because you won't be using it.
3. In **main()**, create a variable **age** and set it to 24.
4. In **main()**, create a second variable **layers** and set it to 5.
5. In **main()**, call a function **printCakeCandles()** and pass in the **age**. This will leave you with an error, because you have not created that function yet.
6. Samewise, call a function **printCakeTop()** and also pass in the **age**.
7. Finally, call a function **printCakeBottom()** and pass in the **age** and also the number of **layers**.
8. To get rid of the errors, comment out the three function calls by adding **//** at the beginning of each line, as shown below. This technique allows you to draft your code without triggering errors.
9. Run your program, and it should have no errors and do nothing.

Your **main()** function should look like the code below.

fun main() {

val age = 24

val layers = 5

// printCakeCandles(age)

// printCakeTop(age)

// printCakeBottom(age, layers)

}

**CREATE printCakeTop()**

The **printCakeTop()** function to print the top of the cake, a line of equal signs, is almost the same as the **printBorder()** function you created earlier in this codelab.

==========================

1. Below the **main()** function, add a blank line, and then create a function, **printCakeTop()** that takes one argument, **age**, of type **Int**.
2. Inside, use a **repeat()** statement to print one equal sign **age** times plus 2. The extra two equals signs are so that the candles won't fall off the side of the cake.
3. At the end, when the **repeat()** is done, print an empty line.
4. In **main()**, remove the two **//** symbols from the beginning of the line of code for **printCakeTop()**, because the function now exists.

printCakeTop(age)

Here is your finished function.

fun printCakeTop(age: Int) {

repeat(age + 2) {

print("=")

}

println()

}

1. Run your code to see the top of the cake.

**CREATE printCakeCandles()**

Each candle is made up of two symbols: a comma (,) for the flame, and a vertical line (|) for the candle body.

,,,,,,,,,,,,,,,,,,,,,,,,

||||||||||||||||||||||||

To accomplish this in one function, put two **repeat()** statements, one for the flames and one for the bodies, into your function.

1. Below the **main()** function and the **printCakeTop()** function, create a new function, **printCakeCandles()** that takes one argument, **age**, of type **Int**.
2. Inside, use a **repeat()** statement to print one comma **,** for the flame.
3. Repeat this **age** times.
4. At the end, print an empty line.
5. Add a print statement to print one space for insetting the candles.
6. Below, repeat the steps to create a second **repeat()** statement to print the candle bodies with a vertical line |.
7. At the end, print a new line, using **println()**.
8. In **main()**, remove the two **//** symbols from the beginning of the line of code for printCakeCandles().

printCakeCandles(age)

1. Run your code to see the top of the cake and the candles

Solution:

fun printCakeCandles(age: Int) {

print(" ")

repeat(age) {

print(",")

}

println() // Print an empty line

print(" ") // Print the inset of the candles on the cake

repeat(age) {

print("|")

}

println()

}

**CREATE printCakeBottom()**

In this function, you are drawing a cake bottom that is as wide as age + 2, and you draw it for a height of a given number of layers.

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

* This means your function needs two arguments, one for the width (**age**) and one for the height (**layers**).
* To print the bottom of the cake, you first repeat the ‘at' @ symbol **age + 2** times to print one layer. Then, you repeat printing one layer **layers** times.

**DRAW THE AT SYMBOL AGE+2 TIMES TO CREATE A LAYER**

1. Below the existing functions, create a function **printCakeBottom()** with two arguments, age and **layers**, both of type **Int**.
2. Inside the function, use a **repeat()** statement to print one layer of ‘at' **@** symbols **age + 2** times. Finish by printing an empty line, as shown below.

fun printCakeBottom(age: Int, layers: Int) {

repeat(age + 2) {

print("@")

}

println()

}

1. Run your code to verify that it prints one line of cake bottom.

,,,,,,,,,,,,,,,,,,,,,,,,

||||||||||||||||||||||||

==========================

@@@@@@@@@@@@@@@@@@@@@@@@@@

**NESTED REPEAT() STATEMENTS**

To print multiple identical layers of cake bottom, you could say:

For layer 1 repeat the symbol 12 times: @@@@@@@@@@@@

For layer 2 repeat the symbol 12 times: @@@@@@@@@@@@

For layer 3 repeat the symbol 12 times: @@@@@@@@@@@@

Or you can say this much more concisely as:

Repeat for three layers:

Repeat the symbol 12 times.

@@@@@@@@@@@@

@@@@@@@@@@@@

@@@@@@@@@@@@

Now, this is something neat you can do with **repeat()** statements. You can put one **repeat()** statement inside another **repeat()** statement. So you could create a **repeat()** statement within a repeat() statement to print the symbol a certain number of times for a certain number of layers.

**USE A NESTED REPEAT() TO PRINT CAKE LAYERS**

1. Put a second **repeat()** statement around all of the code inside the function. Repeat this loop **layers** times.
2. In **main()**, remove only the two **//** from the line of code for **printCakeBottom()**.

printCakeBottom(age, layers)

1. Run your code to see the whole cake.

Solution for **printCakeBottom()**.

fun printCakeBottom(age: Int, layers: Int) {

repeat(layers) {

repeat(age + 2) {

print("@")

}

println()

}

}

Congratulations! You've just finished a pretty complex program with several functions and a nested **repeat** statement. And your cake will always have the right number of candles!

The final output of your program should be:

,,,,,,,,,,,,,,,,,,,,,,,,

||||||||||||||||||||||||

==========================

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

@@@@@@@@@@@@@@@@@@@@@@@@@@

## Solution code

fun main() {

val age = 24

val layers = 5

printCakeCandles(age)

printCakeTop(age)

printCakeBottom(age, layers)

}

fun printCakeCandles(age: Int) {

print (" ")

repeat(age) {

print(",")

}

println() // Print an empty line

print(" ") // Print the inset of the candles on the cake

repeat(age) {

print("|")

}

println()

}

fun printCakeTop(age: Int) {

repeat(age + 2) {

print("=")

}

println()

}

fun printCakeBottom(age: Int, layers: Int) {

repeat(layers) {

repeat(age + 2) {

print("@")

}

println()

}

}

## Troubleshooting

If the browser-based Kotlin programming tool is not executing your code or gives you an unexpected error not related to your code, you can try the following:

* Reload the page using **Shift+Reload**.
* Wait a minute, then try again.

## Summary

* Use **${}** to surround variables and calculations in the text of print statements. For example: **${age}** where **age** is a variable.
* Create a variable using the **val** keyword and a name. Once set, this value cannot be changed. Assign a value to a variable using the equal sign. Examples of values are text and numbers.
* A **String** is text surrounded by quotes, such as **"Hello"**.
* An **Int** is a whole positive or negative number, such as 0, 23, or -1024.
* You can pass one or more arguments into a function for the function to use, for example: **fun** **printCakeBottom(age:Int, layers:Int) {}**
* Use a **repeat() {}** statement to repeat a set of instructions several times. For example: **repeat (23) { print("%") }** or **repeat (layers) { print("@@@@@@@@@@") }**
* A loop is an instruction to repeat instructions multiple times. A **repeat()** statement is an example of a loop.
* You can nest loops, that is, put loops within loops. For example, you can create a **repeat()** statement within a **repeat()** statement to print a symbol a number of times for a number of rows, like you did for the cake layers.

**Summary of using function arguments**: To use arguments with a function, you need to do three things:

* Add the argument and type to the function definition: **printBorder(border: String)**
* Use the argument inside the function: **println(border)**
* Supply the argument when you call the function: **printBorder(border)**

## Learn more

* https://developer.android.com/training/kotlinplayground
* Vocabulary for Android Basics in Kotlin

Here is the official documentation for the Kotlin concepts you learned in this codelab.

* Defining variables
* Comments
* Defining functions
* repeat statement

# Quiz

## What is a program?

* A specific task
* A tool that helps you write Kotlin code
* A series of instructions that a computer system executes to accomplish some action
* A defined set of instructions that tells your computer to print “Happy Birthday!”

## Which keyword do you use to define a function in Kotlin?

* fun
* func
* function
* newFunction
* main

## Which of the following do you need to create a Kotlin program that prints a line of text?

*Choose as many answers as you see fit.*

* comment describing what your program does
* a main() function
* curly braces {} around the instructions to the system
* a call to print() or println()
* a piece of text surrounded by quotation marks

## What do you expect this Kotlin code to do?

fun main(args: Array<String>) {

println("Hello, world!")

println("It's a sunny and warm day!")

}

* Print one line of text
* Print two lines of text
* Print three lines of text
* Print two lines of text separated by a blank line

## How would you modify this main() function so that it prints a 6-layer cake for someone's 4th birthday?

fun main() {

val age = 24

val layers = 5

printCakeCandles(age)

printCakeTop(age)

printCakeBottom(age, layers)

}

* Set val age to 6, set val layers to 4
* Set val age to "4", set val layers to "6"
* Set val age to 4, set val layers to 6
* Leave the code as-is

## Which of these options correctly calls the function, below, and passes it valid input arguments?

fun createMessage(name: String, location: String, age: Int) {

println("My name is ${name}. I am from ${location}, and I am ${age} years old.")

}

* createMessage("Amy", "Australia", 20)
* createMessage("Evan", England, 9)
* createmessage("Tom", "Thailand", “40”)
* createMessage(Heather, “Haiti”, 7)