## 1. Welcome

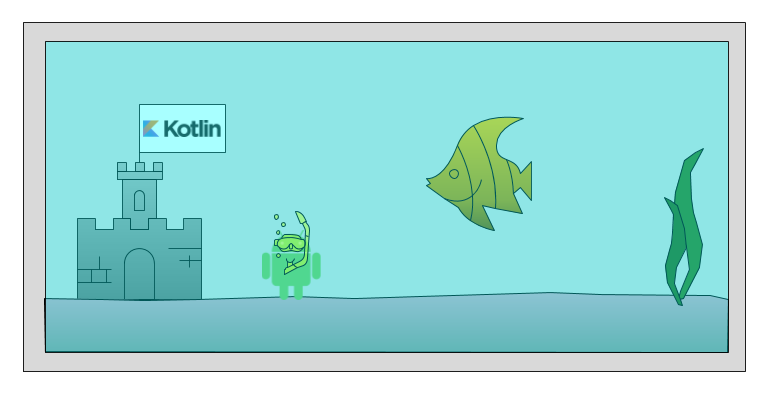
This codelab is part of the [Kotlin Bootcamp for Programmers course](https://developer.android.com/courses/kotlin-bootcamp/overview). You'll get the most value out of this course if you work through the codelabs in sequence. Depending on your knowledge, you may be able to skim some sections. This course is geared towards programmers who know an object-oriented language, and want to learn [Kotlin](https://kotlinlang.org/).



## Introduction

In this codelab, you learn the basics of the [Kotlin](https://kotlinlang.org/) programming language: data types, operators, variables, control structures, and nullable versus non-nullable variables. This course is geared toward programmers who know an object-oriented language, and want to learn Kotlin.

Rather than build a single sample app, the lessons in this course are designed to build your knowledge, but be semi-independent of each other so you can skim sections you're familiar with. To tie them together, many of the examples use an aquarium theme. And if you want to see the full aquarium story, check out the [Kotlin Bootcamp for Programmers](https://classroom.udacity.com/courses/ud9011) Udacity course.



## What you should already know

* How to create a project in IntelliJ IDEA
* How to open and execute code in Kotlin's REPL (Read-Eval-Print Loop) in IntelliJ IDEA (**Tools > Kotlin > Kotlin REPL**)

## What you'll learn

* How to use Kotlin data types, operators, and variables
* How to work with booleans and conditions
* The difference between nullable and non-nullable variables
* How arrays, lists, and loops work in Kotlin

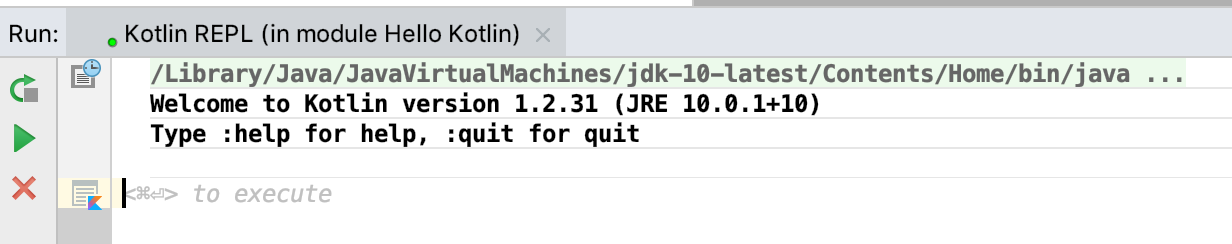
## What you'll do

* Work with the Kotlin REPL to learn the basics of Kotlin

2. Task: Learn about operators and types

In this task, you learn about operators and types in the Kotlin programming language.

Step 1: Explore numeric operators

1. Open IntelliJ IDEA, if it's not already open.
2. To open the Kotlin REPL, select **Tools** > **Kotlin** > **Kotlin REPL**.  
   

As with other languages, Kotlin uses +, -, \* and / for plus, minus, times and division. Kotlin also supports different number types, such as Int, Long, Double, and Float.

1. Enter the following expressions in the REPL. To see the result, press Control+Enter (Command+Enter on a Mac) after each one.

**Note:**In this codelab, **⇒** indicates output from your code. In the latest version of the REPL, the output includes the result number and the type of the result.

1+1

⇒ res8: kotlin.Int = 2

53-3

⇒ res9: kotlin.Int = 50

50/10

⇒ res10: kotlin.Int = 5

1.0/2.0

⇒ res11: kotlin.Double = 0.5

2.0\*3.5

⇒ res12: kotlin.Double = 7.0

Note that results of operations keep the types of the operands, so 1/2 = 0, but 1.0/2.0 = 0.5.

1. Try some expressions with different combinations of integer and decimal numbers.

6\*50

⇒ res13: kotlin.Int = 300

6.0\*50.0

⇒ res14: kotlin.Double = 300.0

6.0\*50

⇒ res15: kotlin.Double = 300.0

1. Call some methods on numbers. Kotlin keeps numbers as primitives, but it lets you call methods on numbers as if they were objects.

2.times(3)

⇒ res5: kotlin.Int = 6

3.5.plus(4)

⇒ res8: kotlin.Double = 7.5

2.4.div(2)

⇒ res9: kotlin.Double = 1.2

**Note:**It is possible to create actual object wrappers around numbers, which is known as *boxing*. Boxing happens automatically, such as for collections, where numbers are boxed and unboxed as needed.

**Warning:** Using object wrappers takes more memory than storing just a number primitive. Do not use boxing unless it is needed, such as in a collection, which is covered later.

Step 2: Practice using types

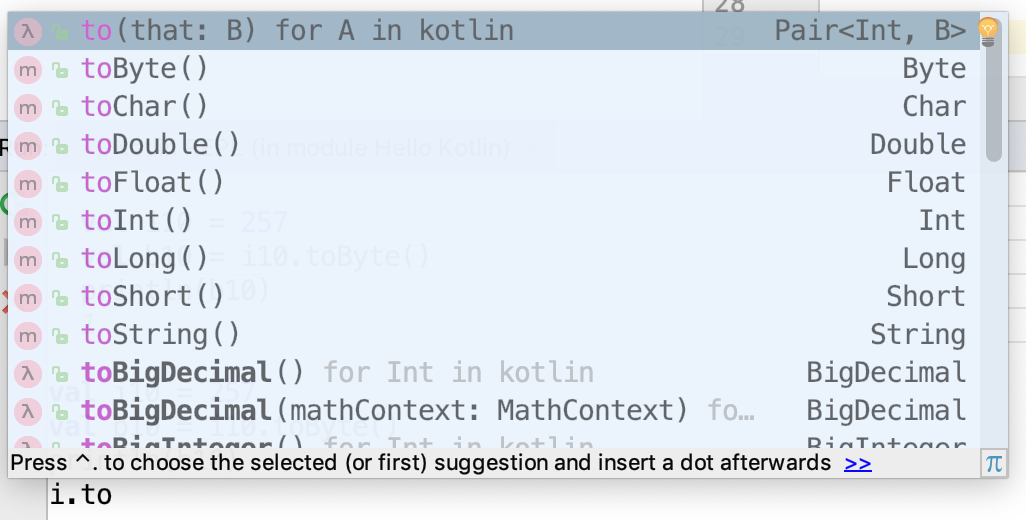
Kotlin does not implicitly convert between number types, so you can't assign a short value directly to a long variable, or a Byte to an Int. This is because implicit number conversion is a common source of errors in programs. You can always assign values of different types by casting.

1. To see some of the casts that are possible, define a variable of type Int in the REPL.

val i: Int = 6

1. Create a new variable, then enter the variable name shown above, followed by .to.

val b1 = i.to

IntelliJ IDEA displays a list of possible completions. This auto-completion works for variables and objects of any type.  


1. Select toByte() from the list, then print the variable.

val b1 = i.toByte()

println(b1)

⇒ 6

1. Assign a Byte value to variables of different types.

val b2: Byte = 1 *// OK, literals are checked statically*

println(b2)

⇒ 1

val i1: Int = b2

⇒ error: type mismatch: inferred type is Byte but Int was expected

val i2: String = b2

⇒ error: type mismatch: inferred type is Byte but String was expected

val i3: Double = b2

⇒ error: type mismatch: inferred type is Byte but Double was expected

1. For the assignments that returned errors, try casting them instead.

val i4: Int = b2.toInt() *// OK!*

println(i4)

⇒ 1

val i5: String = b2.toString()

println(i5)

⇒ 1

val i6: Double = b2.toDouble()

println(i6)

⇒ 1.0

1. To make long numeric constants more readable, Kotlin allows you to place underscores in the numbers, where it makes sense to you. Try entering different numeric constants.

val oneMillion = 1\_000\_000

val socialSecurityNumber = 999\_99\_9999L

val hexBytes = 0xFF\_EC\_DE\_5E

val bytes = 0b11010010\_01101001\_10010100\_10010010

**Note:** Because Kotlin is strongly typed, the compiler can usually infer the type for variables, so you don't need to explicitly declare it.

Step 3: Learn the value of variable types

Kotlin supports two types of variables: changeable and unchangeable. With val, you can assign a value once. If you try to assign something again, you get an error. With var, you can assign a value, then change the value later in the program.

1. Define variables using val and var and then assign new values to them.

var fish = 1

fish = 2

val aquarium = 1

aquarium = 2

⇒ error: val cannot be reassigned

You can assign fish a value, then assign it a new value, because it is defined with var. Trying to assign a new value to aquarium gives an error because it is defined with val.

The type you store in a variable is inferred when the compiler can figure it out from context. If you want, you can always specify the type of a variable explicitly, using the colon notation.

1. Define some variables and specify the type explicitly.

var fish: Int = 12

var lakes: Double = 2.5

Once a type has been assigned by you or the compiler, you can't change the type, or you get an error.

Step 4: Learn about strings

Strings in Kotlin work pretty much like strings in any other programming language using " for strings and ' for single characters, and you can concatenate strings with the + operator. You can create string templates by combining them with values; the $*variable* name is replaced with the text representing the value. This is called *variable interpolation*.

1. Create a string template.

val numberOfFish = 5

val numberOfPlants = 12

"I have $numberOfFish fish" + " and $numberOfPlants plants"

⇒ res20: kotlin.String = I have 5 fish and 12 plants

1. Create a string template with an expression in it. As in other languages, the value can be the result of an expression. Use curly braces {} to define the expression.

"I have ${numberOfFish + numberOfPlants} fish and plants"

⇒ res21: kotlin.String = I have 17 fish and plants

3. Task: Compare conditions and booleans

In this task, you learn about booleans and checking conditions in the Kotlin programming language. Like other languages, Kotlin has booleans and boolean operators such as less than, equal to, greater than, and so on (<, ==, >, !=, <=, >=).

1. Write an if/else statement.

val numberOfFish = 50

val numberOfPlants = 23

if (numberOfFish > numberOfPlants) {

println("Good ratio!")

} else {

println("Unhealthy ratio")

}

⇒ Good ratio!

1. Try a range in an if statement. In Kotlin, the condition you test can use ranges, too.

val fish = 50

if (fish in 1..100) {

println(fish)

}

⇒ 50

1. Write an if with multiple cases. For more complicated conditions, use logical and && and logical or ||. As in other languages, you can have multiple cases by using else if.

if (numberOfFish == 0) {

println("Empty tank")

} else if (numberOfFish < 40) {

println("Got fish!")

} else {

println("That's a lot of fish!")

}

⇒ That's a lot of fish!

1. Try out a when statement. There's a nicer way to write that series of if/else if/else statements in Kotlin, using the when statement, which is like the switch statement in other languages. Conditions in a when statement can use ranges, too.

when (numberOfFish) {

0 -> println("Empty tank")

in 1..39 -> println("Got fish!")

else -> println("That's a lot of fish!")

}

⇒ That's a lot of fish!

## 4. Task: Learn about nullability

In this task, you learn about nullable versus non-nullable variables. Programming errors involving nulls have been the source of countless bugs. Kotlin seeks to reduce bugs by introducing non-nullable variables.

## Step 1: Learn about nullability

By default, variables cannot be null.

1. Declare an Int and assign null to it.

var rocks: Int = null

⇒ error: null can not be a value of a non-null type Int

1. Use the question mark operator, ?, after the type to indicate that a variable can be null. Declare an Int? and assign null to it.

var marbles: Int? = null

When you have complex data types, such as a list:

* You can allow the elements of the list to be null.
* You can allow for the list to be null, but if it's not null its elements cannot be null.
* You can allow both the list or the elements to be null.

Lists and some other complex data types are covered in a later task.

## Step 2: Learn about the ? and ?: operators

You can test for null with the ? operator, saving you the pain of writing many if/else statements.

1. Write some code the longer way to check whether the fishFoodTreats variable is not null. Then decrement that variable.

var fishFoodTreats = 6

if (fishFoodTreats != null) {

fishFoodTreats = fishFoodTreats.dec()

}

1. Now look at the Kotlin way of writing it, using the ? operator.

var fishFoodTreats = 6

fishFoodTreats = fishFoodTreats?.dec()

1. You can also chain null tests with the ?: operator. Look at this example:

fishFoodTreats = fishFoodTreats?.dec() ?: 0

It's shorthand for "if fishFoodTreats is not null, decrement and use it; otherwise use the value after the ?:, which is 0." If fishFoodTreats is null, evaluation is stopped, and the dec() method is not called.

**Note:**The ?: operator is sometimes called the "Elvis operator," because it's like a smiley on its side with a pompadour hairstyle, the way Elvis Presley styled his hair. Read more about [the Elvis operator](https://kotlinlang.org/docs/reference/null-safety.html#elvis-operator) in the Kotlin documentation.

### **A point about null pointers**

If you really love NullPointerExceptions, Kotlin lets you keep them. The not-null assertion operator, !! (double-bang), converts any value to a non-null type and throws an exception if the value is null.

val len = s!!.length *// throws NullPointerException if s is null*

**Note:**In programming slang, the exclamation mark is often called a "[bang](https://en.wikipedia.org/wiki/Exclamation_mark#Computers)," so the not-null assertion operator is sometimes called the "double-bang" or "bang bang" operator.

**Note:**It's generally a bad idea to use the double-bang operator. That's why the language creators made you enter two exclamation marks instead of one. However, sometimes you need the double-bang when dealing with legacy Java code.

## 5. Task: Explore arrays, lists, and loops

In this task, you learn about arrays and lists, and you learn different ways to create loops in the Kotlin programming language.

## Step 1: Make lists

Lists are a fundamental type in Kotlin, and are similar to lists in other languages.

1. Declare a list using listOf and print it out. This list cannot be changed.

val school = listOf("mackerel", "trout", "halibut")

println(school)

⇒ [mackerel, trout, halibut]

1. Declare a list that can be changed using mutableListOf. Remove an item.

val myList = mutableListOf("tuna", "salmon", "shark")

myList.remove("shark")

⇒ res36: kotlin.Boolean = true

The remove() method returns true when it successfully removes the item passed.

**Note:** With a list defined with val, you can't change which list the variable refers to, but you can still change the contents of the list.

## Step 2: Create arrays

Like other languages, Kotlin has [arrays](https://kotlinlang.org/docs/reference/basic-types.html#arrays). Unlike lists in Kotlin, which have mutable and immutable versions, there is no mutable version of an Array. Once you create an array, the size is fixed. You can't add or remove elements, except by copying to a new array.

The rules about using val and var are the same with arrays as with lists.

**Note:** With an array defined with val, you can't change which array the variable refers to, but you can still change the contents of the array.

1. Declare an array of strings using arrayOf. Use the java.util.Arrays.toString() array utility to print it out.

val school = arrayOf("shark", "salmon", "minnow")

println(java.util.Arrays.toString(school))

⇒ [shark, salmon, minnow]

1. An array declared with arrayOf doesn't have a type associated with the elements, so you can mix types, which is helpful. Declare an array with different types.

val mix = arrayOf("fish", 2)

1. You can also declare arrays with one type for all the elements. Declare an array of integers using intArrayOf(). There are corresponding builders, or instantiation functions, for arrays of other types.

val numbers = intArrayOf(1,2,3)

**Note:** Using an array of a primitive type such as Int or Byte avoids the overhead of boxing.

1. Combine two arrays with the + operator.

val numbers = intArrayOf(1,2,3)

val numbers3 = intArrayOf(4,5,6)

val foo2 = numbers3 + numbers

println(foo2[5])

=> 3

1. Try out different combinations of nested arrays and lists. As in other languages, you can nest arrays and lists. That is, when you put an array within an array, you have an array of arrays—not a flattened array of the contents of the two. The elements of an array can also be lists, and the elements of lists can be arrays.

val numbers = intArrayOf(1, 2, 3)

val oceans = listOf("Atlantic", "Pacific")

val oddList = listOf(numbers, oceans, "salmon")

println(oddList)

⇒ [[I@89178b4, [Atlantic, Pacific], salmon]

The first element, numbers, is an Array. When you don't use the array utility to print it, Kotlin prints the address instead of the contents of the array.

1. One nice feature of Kotlin is that you can initialize arrays with code instead of initializing them to 0. Try this example:

val array = Array (5) { it \* 2 }

println(java.util.Arrays.toString(array))

⇒ [0, 2, 4, 6, 8]

The initialization code is between the curly braces, {}. In the code, it refers to the array index, starting with 0.

## Step 3: Make loops

Now that you have lists and arrays, looping through the elements works as you might expect.

1. Create an array. Use a for loop to iterate through the array and print the elements.

val school = arrayOf("shark", "salmon", "minnow")

for (element in school) {

print(element + " ")

}

⇒ shark salmon minnow

1. In Kotlin, you can loop through the elements and the indexes at the same time. Try this example:

for ((index, element) in school.withIndex()) {

println("Item at $index is $element\n")

}

⇒ Item at 0 is shark

Item at 1 is salmon

Item at 2 is minnow

1. Try different step sizes and ranges. You can specify ranges of numbers or of characters, alphabetically. And as in other languages, you don't have to step forward by 1. You can step backward using downTo.

for (i in 1..5) print(i)

⇒ 12345

for (i in 5 downTo 1) print(i)

⇒ 54321

for (i in 3..6 step 2) print(i)

⇒ 35

for (i in 'd'..'g') print (i)

⇒ defg

1. Try out some loops. Like other languages, Kotlin has while loops, do...while loops, and ++ and -- operators. Kotlin also has repeat loops.

var bubbles = 0

while (bubbles < 50) {

bubbles++

}

println("$bubbles bubbles in the water\n")

do {

bubbles--

} while (bubbles > 50)

println("$bubbles bubbles in the water\n")

repeat(2) {

println("A fish is swimming")

}

⇒ 50 bubbles in the water

49 bubbles in the water

A fish is swimmingA fish is swimming

6. Summary

Kotlin is very similar to other languages when it comes to the basics like operators, lists, and loops, but there are some important differences.

The following features may be different in Kotlin than what you're used to in other languages:

* Kotlin types can't be implicitly converted—use casting.
* Variables declared with val can only be assigned once.
* Kotlin variables are not nullable by default. Use ? to make variables nullable.
* With Kotlin, you can loop through the index and elements of an array at the same time in a for loop.

The following Kotlin programming constructs are similar to those in other languages:

* Arrays and lists can have a single type or mixed types.
* Arrays and lists can be nested.
* You can create loops with for, while, do/while, and repeat.
* The when statement is Kotlin's version of the switch statement, but when is more flexible.

## . Learn more

## Kotlin documentation

If you want more information on any topic in this course, or if you get stuck, [https://kotlinlang.org](https://kotlinlang.org/) is your best starting point.

* [Kotlin coding conventions](https://kotlinlang.org/docs/reference/coding-conventions.html)
* [Kotlin idioms](https://kotlinlang.org/docs/reference/idioms.html)
* [Explicit type conversion](https://kotlinlang.org/docs/reference/basic-types.html#explicit-conversions)
* [Defining variables](https://kotlinlang.org/docs/reference/basic-syntax.html#defining-variables)
* [String templates](https://kotlinlang.org/docs/reference/basic-types.html#string-templates)
* [Nullable values](https://kotlinlang.org/docs/reference/basic-syntax.html#using-nullable-values-and-checking-for-null)
* [Lists](https://kotlinlang.org/docs/reference/collections-overview.html#list)
* [Arrays](https://kotlinlang.org/docs/reference/basic-types.html#arrays)
* [if, when, for, while](https://kotlinlang.org/docs/reference/control-flow.html#control-flow-if-when-for-while)
* [?: (Elvis) operator](https://kotlinlang.org/docs/reference/null-safety.html#elvis-operator)
* [!! operator](https://kotlinlang.org/docs/reference/null-safety.html#the--operator)

## Kotlin tutorials

The [https://try.kotlinlang.org](https://try.kotlinlang.org/) website includes rich tutorials called Kotlin Koans, a [web-based interpreter](https://try.kotlinlang.org/#/Examples/Hello,%20world!/Simplest%20version/Simplest%20version.kt), and a complete set of reference documentation with examples.

## Udacity course

To view the Udacity course on this topic, see [Kotlin Bootcamp for Programmers](https://classroom.udacity.com/courses/ud9011).

## IntelliJ IDEA

[Documentation for the IntelliJ IDEA](https://www.jetbrains.com/help/idea/) can be found on the JetBrains website.

## 8. Homework

This section lists possible homework assignments for students who are working through this codelab as part of a course led by an instructor. It's up to the instructor to do the following:

* Assign homework if required.
* Communicate to students how to submit homework assignments.
* Grade the homework assignments.

Instructors can use these suggestions as little or as much as they want, and should feel free to assign any other homework they feel is appropriate.

If you're working through this codelab on your own, feel free to use these homework assignments to test your knowledge.

## ****Answer these questions****

### **Question 1**

Which of the following declares an unchangeable list of strings?

▢ val school = arrayOf("shark", "salmon", "minnow")

▢ var school = arrayOf("shark", "salmon", "minnow")

▢ val school = listOf("shark", "salmon", "minnow")

▢ val school = mutableListOf("shark", "salmon", "minnow")

### **Question 2**

What will be the output of the following code?  
for (i in 3..8 step 2) print(i)

▢ 345678

▢ 468

▢ 38

▢ 357

### **Question 3**

What is the purpose of the question mark in this code?  
var rocks: Int? = 3

▢ The type of the variable rocks isn't fixed.

▢ The variable rocks can be set to null.

▢ The variable rocks cannot be set to null.

▢ The variable rocks shouldn't be initialized right away.