Introduction to Kotlin

1.1: Introduction to Kotlin.

▼ Your First Program in Kotlin.

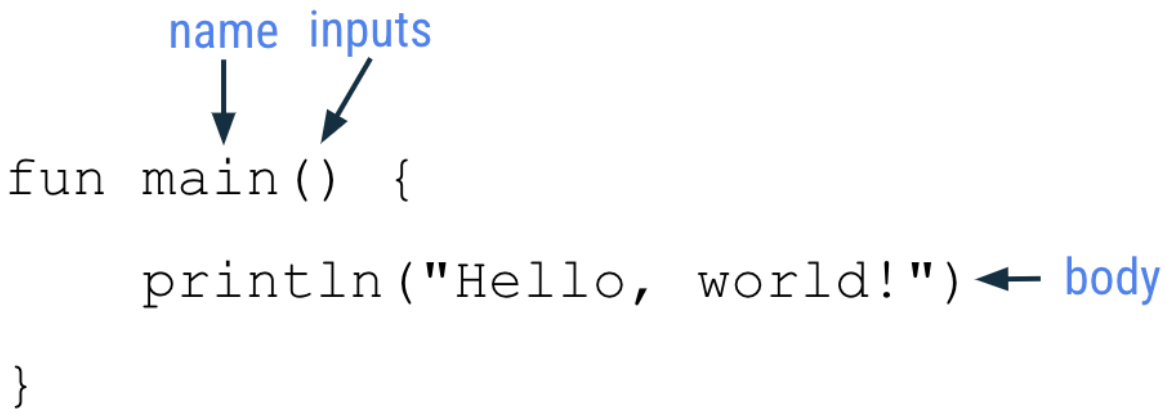
```
fun main() {
    println("Hello mum")
}
```

Kotlin Functions.

A *function* is a segment of a program that performs a specific task. Once the function is defined, then you can *call* that function, so the instructions within that function can be performed or executed.

These are the key parts needed to define a function:

- The function needs a **name**, so you can call it later.
- The function can also require some **inputs** that needs to be provided when the function is called.
- The function also has a **body** which contains the instructions to perform the task.



An input is a piece of data that a function needs to perform its purpose. When you define a function, you can require that certain inputs be passed in when the function is called. If there are no inputs required for a function, the parentheses are empty. The *function body* contains the instructions needed to achieve the purpose of the function.

Conclusion.

- A Kotlin program requires a `main` function as the entry point of the program.
- To define a function in Kotlin, use the `fun` keyword followed by the name of the function, any inputs enclosed in parentheses, followed by the function body enclosed in curly braces.
- The name of a function should follow camel case (`camelCase`) convention and start with a lowercase letter.
- Use the `println()` function call to print some text to the output.

▼ Create and Use Variables in Kotlin.

A *variable*, which is a container for a single piece of data.

Data Types.

When you decide what aspects of your app can be variable, it's important to specify *what type of data* can be stored in those variables. In Kotlin, there are some common basic data types.

Kotlin data type	What kind of data it can contain	Example literal values
`String`	Text	`"Add contact"` `"Search"` `"Sign in"`
`Int`	Integer number	`32` `1293490` `-59281`
`Double`	Decimal number	`2.0` `501.0292` `-31723.99999`
`Float`	Decimal number (that is less precise than a `Double`). Has an `f` or `F` at the end of the number.	`5.0f` `-1630.209f` `1.2940278F`
`Boolean`	`true` or `false`. Use this data type when there are only two possible values. Note that `true` and `false` are keywords in Kotlin.	`true` `false`

```
fun main() {
    val count: Int = 2
    println(count)
}
```

In the variable declaration, the variable name follows the `val` keyword. Variable names should follow the camel case convention. The first word of the variable name is all lower case. If there are multiple words in the name, there are no spaces between words, and all other words should begin with a capital letter.

String Template.

```
fun main() {
    val count: Int = 2
```

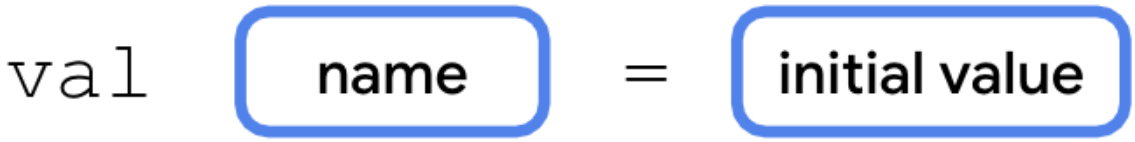
```
    println("You have $count unread messages")
}
```

This is a *string template* because it contains a *template expression*, which is a dollar sign (`$`) followed by a variable name. A template expression is evaluated and its value gets substituted into the string.

You can see how useful a string template can be. You only wrote the string template once in your code (`"You have $count unread messages."`). If you change the initial value of the `count` variable, the `println()` statement still works.

Type Inference.

Type inference is when the Kotlin compiler can infer (or determine) what data type a variable should be, without the type being explicitly written in the code. That means you can omit the data type in a variable declaration, if you provide an initial value for the variable. The Kotlin compiler looks at the data type of the initial value, and assumes that you want the variable to hold data of that type.



NB: If you don't provide an initial value when you declare a variable, you must specify the type.

Basic Math Operations with Integers.

```
fun main() {
    val unreadCount = 5
    val readCount = 100
    println("You have ${unreadCount + readCount} total messag
}
```

Updating Variables.

Variables declared with `val` cannot be reassigned. If you need to update the value of a variable, declare the variable with the Kotlin keyword `var`, instead of `val`.

- `val` keyword - Use when you expect the variable value will not change. With `val`, the variable is *read-only*, which means you can only read, or access, the value of the variable. Once the value is set, you cannot edit its value.
- `var` keyword - Use when you expect the variable value can change. With `var`, the variable is *mutable*, which means the value can be changed or modified. The value can be mutated.

```
fun main() {
    var cartTotal = 0
    println("Total: $cartTotal")

    cartTotal = 20
    println("Total: $cartTotal")
}
// Total: 0
// Total: 20
```

Increment and Decrement Operators.

If you want to increase a variable by `1`, you can use the *increment operator* (`++`). By using these symbols directly after a variable name, you tell the compiler that you want to add 1 to the current value of the variable, and then store the new value in the variable.

```
fun main() {
    var count = 10
    println("You have $count unread messages.")
    count++
    println("You have $count unread messages.")
}
```

Explore Other Data Types.

When reading code that contains strings, you may come across *escape sequences*. Escape sequences are characters that are preceded with a backslash symbol (`\`), which is also called an escaping backslash.

```
fun main() {
    println("Say \"hello\"")
}
// Output: Say "hello"
```

The `Boolean` data type is useful when your variable only has two possible values represented by `true` or `false`.

Other data types can be concatenated to `Strings`. For example, you can concatenate `Booleans` to `Strings`. Use the `+` symbol to concatenate (or append) the value of the `notificationsEnabled` boolean variable onto the end of the `"Are notifications enabled? "` string.

```
fun main() {
    val notificationsEnabled: Boolean = false
    println("Are notifications enabled? " + notificationsEnab
}
// Output: Are notifications enabled? false
```

Conclusion.

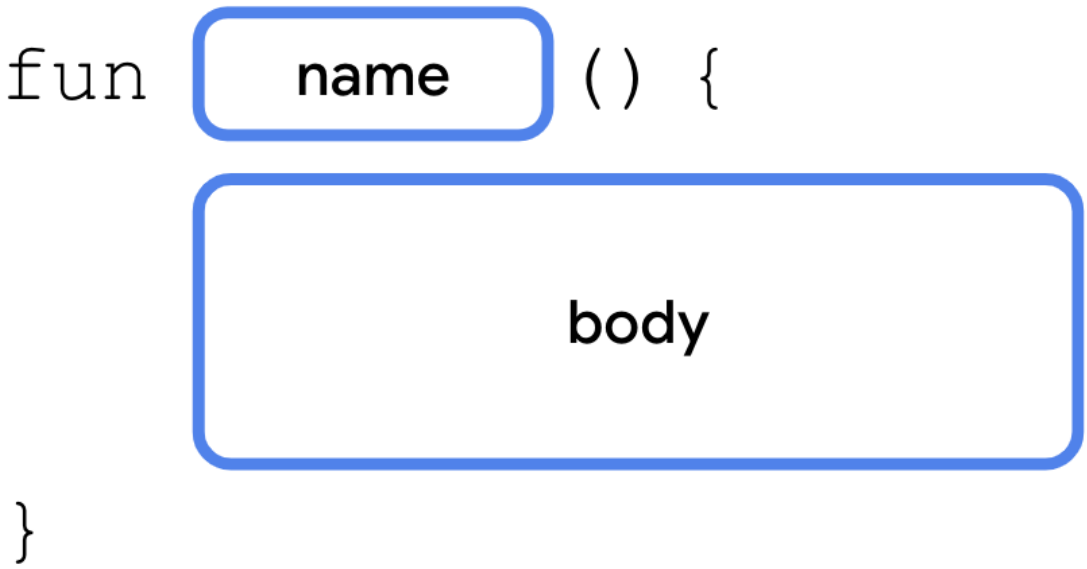
- A variable is a container for a single piece of data.
- You must declare a variable first before you use it.
- Use the `val` keyword to define a variable that is read-only where the value cannot change once it's been assigned.
- Use the `var` keyword to define a variable that is mutable or changeable.
- In Kotlin, it's preferred to use `val` over `var` when possible.
- To declare a variable, start with the `val` or `var` keyword. Then specify the variable name, data type, and initial value. For example: `val count: Int = 2`.

- With type inference, omit the data type in the variable declaration if an initial value is provided.
- Some common basic Kotlin data types include: `Int`, `String`, `Boolean`, `Float`, and `Double`.
- Use the assignment operator (`=`) to assign a value to a variable either during declaration of the variable or updating the variable.
- You can only update a variable that has been declared as a mutable variable (with `var`).
- Use the increment operator (`++`) or decrement operator (`-`) to increase or decrease the value of an integer variable by 1, respectively.
- Use the `+` symbol to concatenate strings together. You can also concatenate variables of other data types like `Int` and `Boolean` to `Strings`.

▼ Functions in Kotlin.

Define and Call a Function.

The `main()` function is intended only to include other code you want to execute. The syntax for defining a function is shown in the following diagram:

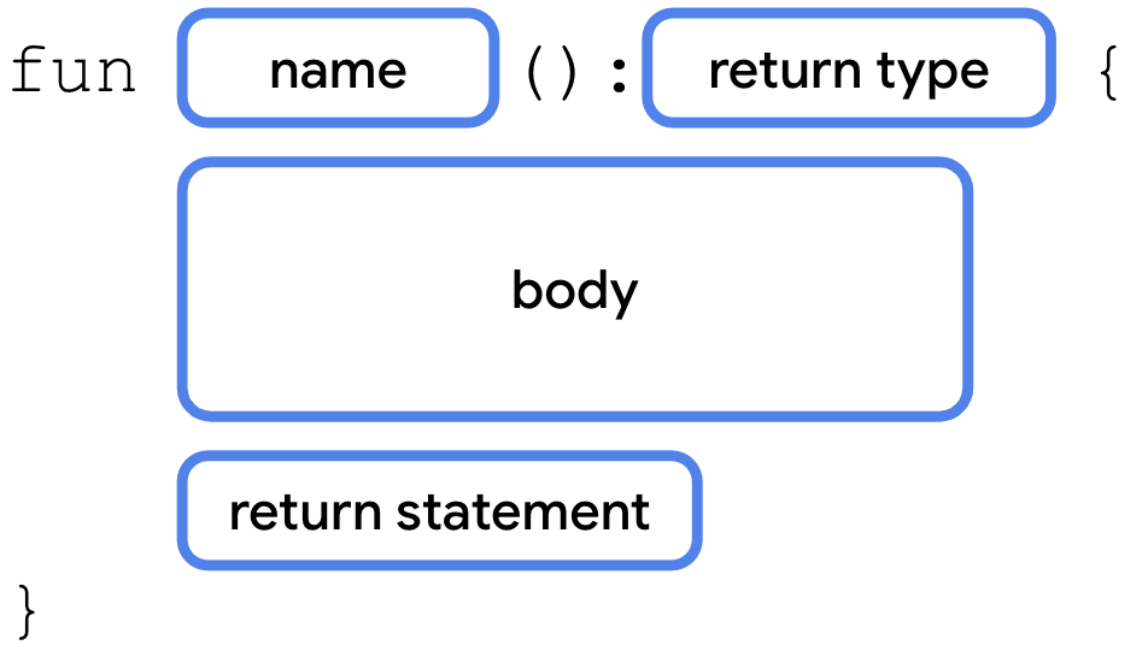


```
fun main() {
        birthdayGreeting()
}

fun birthdayGreeting() {
        println("Happy Birthday Collins")
        println("You are now 23 years old!")
}
```

Return a Value from a Function.

When defining a function, you can specify the data type of the value you want it to return. The return type is specified by placing a colon (`:`) after the parentheses and the name of a type (`Int`, `String`, etc) after the colon. A return statement consists of the `return` keyword followed by the value, such as a variable, you want the function to return as an output.



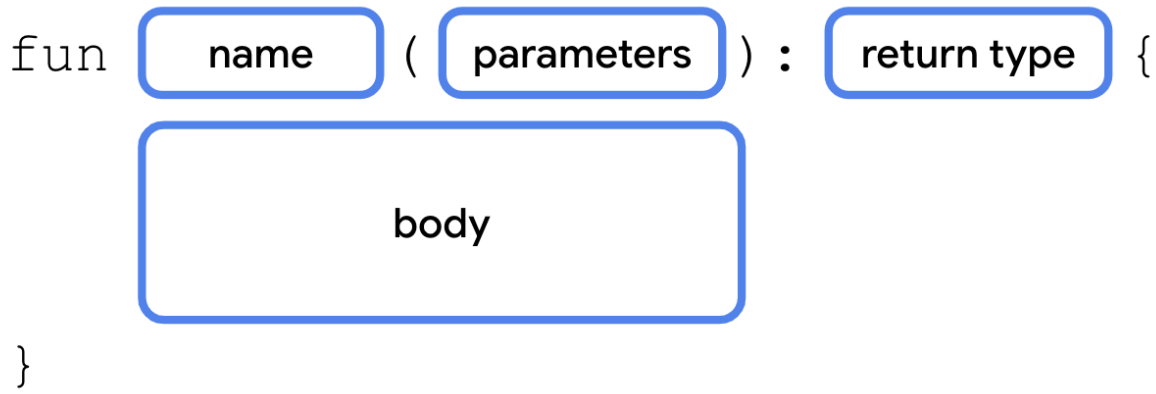
The `Unit` Type.

By default, if you don’t specify a return type, the default return type is `Unit`. `Unit` means the function doesn't return a value. `Unit` is equivalent to void return types in other languages. For functions that don't return anything, or returning `Unit`, you don't need a return statement.

```
fun birthdayGreeting(): String {
    val nameGreeting = "Happy Birthday, Rover!"
    val ageGreeting = "You are now 5 years old!"
    return "$nameGreeting\n$ageGreeting"
}
```

Parameters.

A parameter specifies the name of a variable and a data type that you can pass into a function as data to be accessed inside the function. Parameters are declared within the parentheses after the function name.



Each parameter consists of a variable name and data type, separated by a colon and a space. Multiple parameters are separated by a comma.

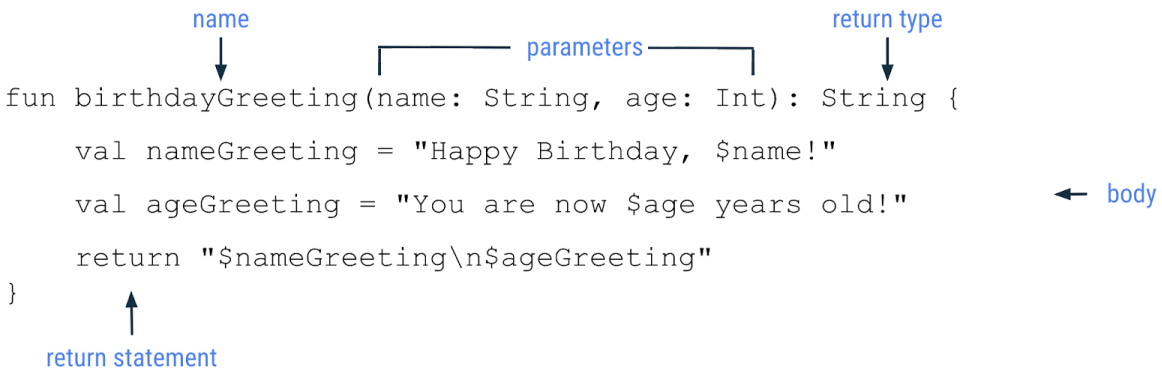
```
fun birthdayGreeting(name: String): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now 5 years old!"
    return "$nameGreeting\n$ageGreeting"
}
```

💡 Unlike in some languages, such as Java, where a function can change the value passed into a parameter, parameters in Kotlin are immutable. You cannot reassign the value of a parameter from within the function body.

For multiple parameters:

```
fun Function name ( First parameter , Second parameter , ...)
```

```
fun birthdayGreeting(name: String, age: Int): String {
    val nameGreeting = "Happy Birthday, $name!"
    val ageGreeting = "You are now $age years old!"
    return "$nameGreeting\n$ageGreeting"
}
```



Named Arguments.

You don't need to specify the parameter names, `name` or `age`, when you called a function. However, you're able to do so if you choose.

```
println(birthdayGreeting(name = "Rex", age = 2))
```

Default Arguments.

Function parameters can also specify default arguments. When you call a function, you can choose to omit arguments for which there is a default, in which case, the default is used.

```
fun birthdayGreeting(name: String = "Rover", age: Int): Strin
    return "Happy Birthday, $name! You are now $age years old
}
```

In the first call to `birthdayGreeting()` for Rover in `main()`, set the `age` named argument to `5`. Because the `age` parameter is defined after the `name`, you need to use the named argument `age`.

Without named arguments, Kotlin assumes the order of arguments is the same order in which parameters are defined. The named argument is used to ensure Kotlin is expecting an `Int` for the `age` parameter.

```
println(birthdayGreeting(age = 5))
println(birthdayGreeting("Rex", 2))
```

Conclusion.

- Functions are defined with the `fun` keyword and contain reusable pieces of code.
- Functions help make larger programs easier to maintain and prevent the unnecessary repetition of code.
- Functions can return a value that you can store in a variable for later use.
- Functions can take parameters, which are variables available inside a function body.
- Arguments are the values that you pass in when you call a function.
- You can name arguments when you call a function. When you use named arguments, you can reorder the arguments without affecting the output.
- You can specify a default argument that lets you omit the argument when you call a function.