Data Types

In the following lesson, you will be introduced to Scala's type hierarchy.

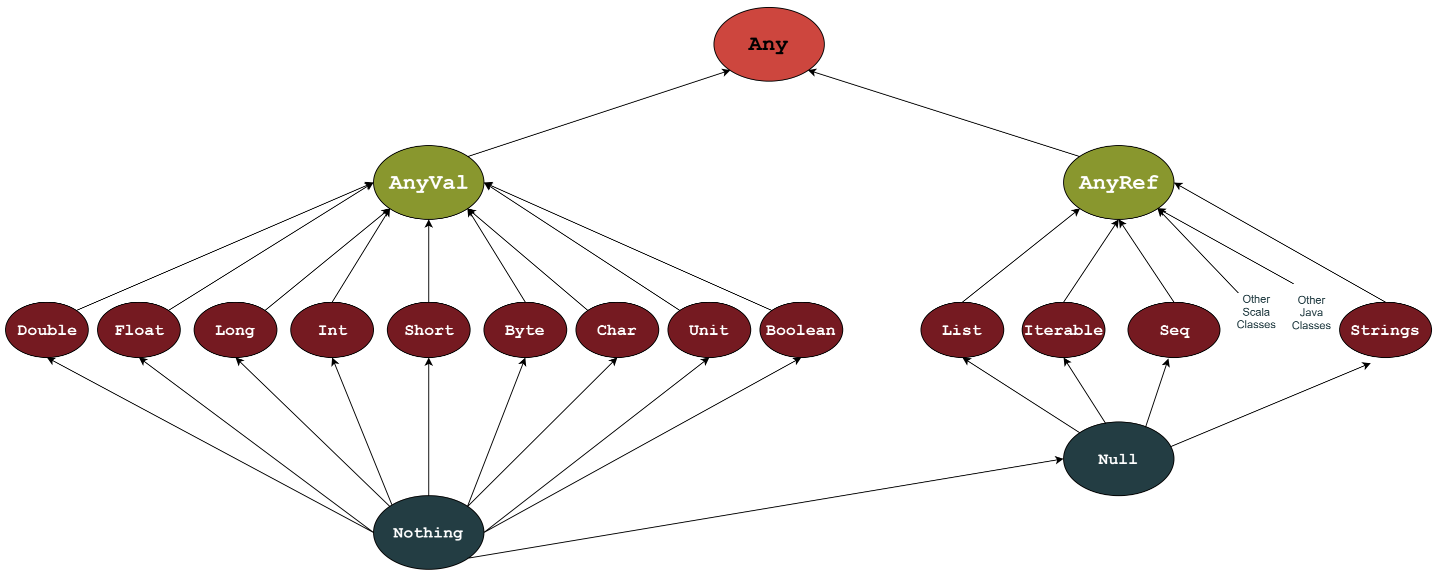
We'll cover the following

Introduction

In Scala, every value has a type and each type is part of a type hierarchy. In a type hierarchy, the more generic types are at the top and as you go lower in the hierarchy, the types get more specific, i.e., have more requirements. Each type has a requirement of its own along with the requirements of all the types above it in the hierarchy.

Scala Type Hierarchy

At the top of the type hierarchy in Scala, we have type `Any`. It is the super-type and defines certain universal methods such as `equals`, `hashCode`, and `toString`.



Scala Type Hierarchy

`Any` has two subclasses, namely `AnyVal` and `AnyRef`. `AnyVal` represents value types. There are nine main value types supported in Scala:

- Double
- Float
- Long
- Int
- Short
- Byte
- Char
- Unit
- Boolean

`AnyRef` represents reference types. Any type that is not a value type is a reference type, including types defined by users not predefined in Scala.

Value Type vs. Reference Type

To remain in the scope of this course, we won't be getting into the nitty-gritty of the difference between value types and reference types. All you need to know is that for a value type, the information it provides is the value itself. For a reference type, the information it provides is a reference to some object, i.e., the memory address of an object. To make this clearer, let's look at this using physical objects.

Imagine you have a piece of paper. You want the paper to hold some information, such as your name. You can write your name on the piece of paper; therefore the

value of the paper is the same as the information it provides. This is an example of *value type*. If someone wants to know your name, all they have to do is read the paper.

Now imagine, you want the paper to hold your house. That's not physically possible, so you write the address to your house, hence the value of the paper is a reference to the required information. This is an example of *reference type*. If someone wants to go to your house, they will first have to read the directions to your house.

In the same way, a reference type holds the memory address location of the value, while value types hold the value themselves.

> The focus of this chapter is value types.

`Any` in Action

Let's look at an example to better understand what it means to be at the top of the type hierarchy.

In the example below, we have a variable `anyInAction` which is of type `Any`. We can assign `anyInAction` a value of any type.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```
var anyInAction: Any = "A String" //String
println(anyInAction)

anyInAction = 5 //Int
println(anyInAction)

anyInAction = '☺' //Char
println(anyInAction)

anyInAction = 1.985 //Float
println(anyInAction)

anyInAction = true //Boolean
println(anyInAction)
```

The code snippet above assigns five different types of values to `anyInAction` and prints the value after each new assignment.

Using Value Types

In the previous lessons, we discussed the syntax for declaring variables. Let’s use what we have learned to declare variables of different types. Remember, in Scala, the first letter of the data type is capitalized.

This code requires the following environment variables to execute:

LANG	C.UTF-8

```scala
val myDoubleVariable: Double = 2.75
val myFloatVariable: Float = 2.75f
val myLongVariable: Long = 2750000000000L
val myIntVariable: Int = 275
val myShortVariable: Short = 1
val myByteVariable: Byte = 0xa
val myCharVariable: Char = '☺'
val myUnitVariable: Unit = ()
val myBooleanVariable: Boolean = true
```

To print any of the variables declared in the code above, simply use the `print` or `println` method we learned in one of our previous lessons and pass the variable name of the variable whose value you want to print.

From all the data types, `Unit` might appear to be somewhat unfamiliar. Just remember that `Unit` is analogous to `void` in other programming languages like Java; put simply, it is nothing.

Now that we have covered all the requirements to create a variable, in the next lesson, you will be challenged to create your own variable.