Scala Tutorial Pt. 2

For the next few minutes, I won't be writing the entire Scala program. Instead, I'll show you a portion of the program (a few lines of code) to explain the syntax and elements of the language. I'll provide you with various syntax examples and information.

I don't expect you to learn and remember every syntax and detail I discuss. Just read and try to remember as much as you can, but don't worry too much because you'll eventually become familiar with all of these through practice. Now, let's begin with variables (`var`) and constants (`val`).

Variables are created using `var` keyword.

```
var amount
```

But, this is not syntactically correct because variables need to be defined with a type since Scala is a *typed* programming language.

Following are the few of may types Scala by default provides you.

```
// Integer data types.
* Byte
* Short
* Int
* Long

// Floating point number data types.
* Float
* Double

// Character data type.
* Char

// Boolean data type.
* Boolean

// String data type.
* String
```

Following is the correct way to define variable `amount` in Scala with type.

```
var amount: Float
```

The type of a variable is specified after its name, followed by a colon. This syntax is inspired by Haskell, a well-known functional programming language. Go also use the same syntax and so as TypeScript. Some language designers think that this is more easy to read like, *define a variable* ***`amount`*** *with type float.*

You can optionally assign an initial value to the variables you're creating using the assignment operator (=).

```
var amount: Float = 3.29
```

Scala is an intelligent programming language, which means that if you provide an initial value to a variable, it will automatically determine or "infer" the type of the variable based on that initial value. In such cases, you don't have to explicitly mention the type of the variable.

```
var amount = 3.29
```

In the code mentioned above, Scala will assign the type `Double` to the variable `amount`. Scala has multiple types for floating point numbers and integers, and it needs to decide on a default initial type. For floating point numbers, Scala has chosen `Double`, and for integers, it has chosen `Int`. However, for types like `Char`, `Boolean`, `String`, and others, there is no need for Scala or us to make a choice because there are no other options available. Additionally, it's worth noting that all types in Scala are capitalized.

Similar to variables, we can define constants using the same syntax, with one difference: instead of using the keyword var, we use val, which stands for "value" or constant. The rule for constants is that you cannot change their value once they are assigned.

```
val price = 49
```

Nowadays, developers use word *immutable variables* instead of constants. So, with this new trend, we just defined the immutable variable `price` with initial value of `49` using `val` keyword.

Most of the time, I'm going to use `val` keyword as that's what community prefer and suggest. Also, if you know the initial value, you don't have to worry about the types!

Let's talk about the methods as you already know the part of it as we have already defined method in previous article. Let me copy the same method here for the further discussion.

```
def hello() =
  println("hello, world")
```

Using `return` statement, we can return something from the method.

```
def hello(): String =
  return "hello, world"
```

The return keyword is often optional, especially when the method's last expression is the value you want to return.

Notice the type we've mentioned after colon(`:`) and before the assignment operator (`=`). Yes, if we're returning something we need to define the return type for the method like above.

For the method that return nothing, `Unit` should be used but optionally.

```
def hello(): Unit =
  println("hello, world")
```

Method by default return the result of last expression. In such a case, you can omit the `return` keyword.

```
def hello(): String =
  "hello, world"
```

If method accepts the parameters, those parameters also needs to defined with type.

```
def sum(x: Int, y: Int): Int =
  return x + y
```

This method accepts `x` and `y` both as the integer type and return the integer. It is mandatory to define the parameters of the method with type like we did for `x` and `y`. Calling this method is as usual as,

```
sum(20, 22)
```

This call return the same of these two numbers.

Scala supports optional parameters by allowing you to provide default values.

```
def sum(x: Int = 0, y: Int = 0): Int =
  return x + y
```

We can now call the above method as,

```
sum(20, 22)
sum(20)
sum()
```

The first call return `42`. Second call returns the `20` as value of `x` is 20 and `y` is 0 as default value. Third call returns 0 as `x` and `y` are default to `0`.