Scala | Lazy Evaluation

Last Updated : 15 Mar, 2019

Lazy evaluation or call-by-need is a evaluation strategy where an expression isn't evaluated until its first use i.e to postpone the evaluation till its demanded. Functional programming languages like Haskell use this strategy extensively. C, C++ are called strict languages who evaluate the expression as soon as it's declared. Then there are languages like Scala who are strict by default but can be lazy if specified explicitly i.e. of mixed type.

Let's see an example in Scala:

Without lazy:

```
val geeks = List(1, 2, 3, 4, 5)

val output = geeks.map(l=> l*2)

println(output)
```

The value of output is calculated as soon as the operation is applied on it.

With lazy:

```
val geeks = List(1, 2, 3, 4, 5)

lazy val output2 = geeks.map(l=> l*2)

println(output2)
```

The value isn't calculated till we use output2 that's till println(output2).

Why lazy evaluation?

In the example what if we never use the output value? We wasted our map operation (CPU computations) which can be very costly when we write more complex and bigger code. Here lazy evaluation helps us in optimizing the process by evaluating the expression only when it's needed and avoiding unnecessary overhead.

Pros:

Optimizes the computation process. Spark a big data computation engine uses this technique at it's core.

Lazy evaluation can help us to resolve circular dependencies.

Gives access to infinite data structure.

Allows modularity of code into parts.

The programmer lose control over the sequence their code is executed as some expressions are evaluated and others aren't depending on the need.

Cons:

Finding bugs can be tricky as programmer has no control over program execution.

Can increase space complexity as all the instructions(operations) have to stored.

Harder to code in contrast with conventional approach.