Lazy Evaluation and Streams

Was ist strict evaluation?

-> call-by-value

- Ausdrücke werden direkt berechnet, bevor sie tatsächlich benötigt werden.
- jede Berechnung wird einmal durchgeführt und das Ergebnis gespeichert.
- effizient wenn man den Wert sicher benötigt.

```
lazy val x = {
  println("Berechnung von x")
  42
}

println("Vor der Nutzung von x")
println(x) // Erst hier wird x berechnet
println(x) // x wurde bereits berechnet und wird nur ausgegeben

Vor der Nutzung von x
Berechnung von x
42
42
```

Was ist non-strict evaluation?

-> call-by-name

- Ausdrücke werden erst berechnet, wenn sie tatsächlich benötigt werden.
- jede Berechnung wird bei jedem Zugriff durchgeführt, Wert wird nicht gespeichert.
- nicht das selbe wie lazy evaluation, da lazy evaluation den Wert speichert und nur einmal berechnet.

```
def callByName(x: => Int): Int = {
  println("Funktion aufgerufen")
  x + x
}

val result = callByName({
  println("Berechnung von x")
  42
})

println("Vor der Nutzung von x")
println(result) // result wird hier berechnet
println(result) // result wird hier erneut berechnet
```

```
Funktion aufgerufen
Berechnung von x
42
Berechnung von x
42
```

Was ist lazy evaluation?

- Ausdrücke werden erst berechnet, wenn sie tatsächlich benötigt werden.
- Wert wird gespeichert und nur einmal berechnet.

```
lazy val x = {
  println("Berechnung von x")
  42
}

println("Vor der Nutzung von x")
println(x) // Erst hier wird x berechnet
println(x) // x wurde bereits berechnet und wird nur ausgegeben


Vor der Nutzung von x
Berechnung von x
42
42
```

Vergleiche val, lazy val und def (function).