Loops

- Proposition: Variables are enough to model all imperative programs. But what about control statements like loops? We can model them using functions.
- **Example**: Here is a Scala program that uses a while loop:

```
def power(x: Double, exp: Int): Double = {
  var r = 1.0
  var i = exp
  while (i > 0) { r = r * x; i = i – 1 }
  r
}
```

- In Scala, while is a keyword representing the While-Loop and it has the syntax:

```
while (condition) {
  command
}
```

Definition of While

- The function WHILE can be defined as follows:

```
def WHILE(condition: => Boolean)(command: => Unit): Unit = {
  if (condition) {
    command
    WHILE(condition)(command)
  }
  else
    ()
}
```

- **Note**: The condition and the command must be passed by name so that they're reevaluated in each iteration
- **Note**: WHILE is tail recursive, so it can operate with a constant stack size

Exercise: Write a function implementing a repeat loop that is used as follows:

```
REPEAT {
  command
} (condition)
```

It should execute command one or more times, until condition is true.
The implementation for this Repeat-Loop is:

```
def REPEAT(command: => Unit)(condition: => Boolean) =
  command
  if (!condition)
    REPEAT(command)(condition)
  else ()
```

Is it also possible to obtain the following syntax?

```
REPEAT {
  command
} UNITL (condition)
```

The implementation for this Repeat-Until-Loop is:

```
def REPEAT_UNTIL(command: => Unit)(condition: => Boolean) = {
  def loop(command: => Unit)(condition: => Boolean): Unit =
    if (!condition) {
      command
      loop(command)(condition)
    }

  command
  if (!condition)
    loop(command)(condition)
}
```

Scala uses the following syntax for this Repeat-Until-Loops:

```
do {
  command
} while(condition)
```

For-Loops

- The classical for loop in Java cannot be modeled simply by a higher-order function
- The reason is that in a Java program like:
  for (int i = 1; i < 3; i = i + 1) System.out.print(i + " ");
  the arguments of for contain the **declaration** of the variable i, which is visible in other arguments and in the body
- However, in Scala there is a kind of for loop similar to Java's extended for loop:
  for (i <- 1 until 3) System.out.print(i + " ")
  This displays: ***1 2***

Translation of For-Loops

- For-loops translate similarly to for-expressions, but using the foreach combinator instead of map and flatMap
- **foreach** is defined on collections with elements of type T as follows:

```
def foreach(f: T => Unit): Unit =
  // apply 'f' to each element of the collection
```

- **Example:**
  for (i <- 1 until 3; j <- "abc") println(i + " " + j)
  translates to:
  (1 until 3).foreach(i => "abc".foreach(j => println(i + " " + j)))
  and prints the following:

```
1 a
1 b
1 c
2 a
2 b
2 c
```