```
package learning.concurrency
package promises
package myfuture

// Prokopec - 2014 - Learning Concurrent Programming in Scala, page 121

import scala.concurrent._
import scala.util.{ Try, Success, Failure }
import scala.util.control.NonFatal
import ExecutionContext.Implicits.global

object Main extends App {

  // `b` is the asynchronous computation
  // cf. scala.concurrent.impl.Future#apply (2.11.x)
  def myFuture[T](b: =>T): Future[T] = {
    val p = Promise[T]
    global.execute(new Runnable {
      def run() = try p.success(b) catch { case NonFatal(e) => p.failure(e) }
      // def run() = p complete ( try Success(b)
      //                          catch { case NonFatal(e) => Failure(e) } )
    })
    p.future
  }

  val f = myFuture { "naa" + "na" * 8 + " Katamari Damacy!" }
  f foreach { case text => println(text) }

  Thread.sleep(2000)

}
/*
package scala.concurrent.impl



import scala.concurrent.ExecutionContext
import scala.util.control.NonFatal
import scala.util.{ Success, Failure }


private[concurrent] object Future {
  class PromiseCompletingRunnable[T](body: => T) extends Runnable {
    val promise = new Promise.DefaultPromise[T]()

    override def run() = {
      promise complete {
        try Success(body) catch { case NonFatal(e) => Failure(e) }
      }
    }
  }

  def apply[T](body: =>T)(implicit executor: ExecutionContext): scala.concurrent.Fut
ure[T] = {
    val runnable = new PromiseCompletingRunnable(body)
    executor.prepare.execute(runnable)
    runnable.promise.future
  }
}
*/
```