

ZIO

ZIO

- >> ZIO[R,E,A]
- >> R = EnvironmentType
- >> E = Error Type
- >> A = Value type
- >> Mental model – $R \Rightarrow \text{Either}[E,A]$

Ammonite

<https://ammonite.io/>

Hello world example

```
import $ivy.`dev.zio::zio:1.0.0`  
import zio.ZIO
```

```
import $ivy.`dev.zio::zio:1.0.0`  
import zio.ZIO  
ZIO("hello world")  
ZIO(println("hello world"))
```

ZIO – Description of a program

Running an effect

```
import $ivy.`dev.zio::zio:1.0.0`  
import zio.ZIO  
val runtime = zio.Runtime.default  
runtime.unsafeRun(ZIO(println("hello world")))
```

Chaining effects

```
import $ivy.`dev.zio::zio:1.0.0`  
import zio.ZIO  
val input = zio.console.getStrLn  
val print = zio.console.putStrLn(input)
```


A detour

```
import scala.concurrent.ExecutionContext.global
import scala.concurrent.Future

implicit val ec: scala.concurrent.ExecutionContext = scala.concurrent.ExecutionContext.global
def getUsername: Future[String] = Future("dor")
def getUserImage(name:String): Future[Array[Byte]] = Future(Array())

getUserImage(getUsername)
```

Map for the help

```
import scala.concurrent.Future  
def getUserName: Future[String] = ???  
def getUserImage(name:String): Future[Array[Byte]] = ???  
def map[B](a: A=> B): Future[B]
```

Does it work

```
getName.map(getImage)
```

```
// res11: Future[Future[Array[Byte]]] = Future(Success(Future(Success([B@460f2b6c])))
```

Flatmap for the help

```
def flatMap[B](a: A=> Future[B]): Future[B]
```

```
@ getUsername.flatMap(getUserImage)
res12: Future[Array[Byte]] = Future(Success([B@7f31e2fb))
```

For comprehensions

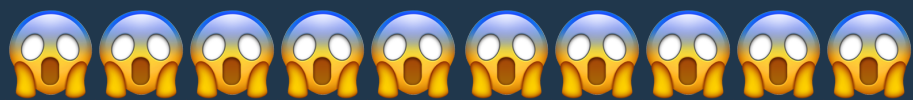
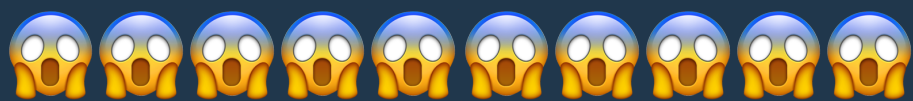
```
import scala.concurrent.Future
```

```
def getName: Future[String]= Future("string")
```

```
def getImage: Future[String] = Future("image")
```

```
def downloadImage : Future[String] = Future("image")
```

```
getName.flatMap(name => getImage.flatMap(image=> downloadImage))
```



For comprehensions syntax

```
for {  
  name <- getName  
  image <- getImage  
  contents <- downloadImage  
} yield contents
```



Chaining effects solved

```
import $ivy.`dev.zio::zio:1.0.0`  
import zio.ZIO  
val runtime = zio.Runtime.default  
val input = zio.console.getStrLn  
val print = input.flatMap(x => zio.console.putStrLn(x))  
runtime.unsafeRun(print)
```


Summary

- >> Code is a description of a program
- >> Execution happens at the end of the world (`main`, runtime)
- >> sequencing is done by `map\flatMap`

Hello world demo

Sbt file

```
import scala.util.Try

scalaVersion in ThisBuild := "2.12.11"

val zio = "dev.zio" % "zio" % "1.0.0"

resolvers += Resolver.sonatypeRepo("snapshots")


lazy val root = Project("hello-world", file("."))
  .settings(Seq(
    organization := "io.bigpanda",
    name := "example",
    libraryDependencies += "dev.zio" %% "zio" % "1.0.1",
    testFrameworks += new TestFramework("zio.test.sbt.ZTestFramework")
  ))
```

```
import zio._  
import zio.console._  
  
object MyApp extends zio.App {  
  
  def run(args: List[String]) =  
    myAppLogic.exitCode  
  
  val myAppLogic =  
    ???  
}
```

Hello world working

```
import zio._
import zio.console._

object MyApp extends zio.App {

  def run(args: List[String]) =
    myAppLogic.exitCode

  val myAppLogic =
    for {
      _    <- putStrLn("Hello! What is your name?")
      name <- getStrLn
      _    <- putStrLn(s"Hello, ${name}, welcome to ZIO!")
    } yield ()
}
```

Demo

Word Count – part #1

>> Ask for path

>> Read file

>> Count words

>> Print length

Demo

```
import zio._
import zio.console._
import scala.io.Source
object wordCount extends zio.App {
  def run(args: List[String]) =
    myAppLogic.exitCode

  val myAppLogic =
    for {
      _ <- putStrLn(
        "Hello! What path do you want to word count? please enter full path")
      fullPath <- getStrLn
    } yield fullPath

  def countWords(str: String):Int = ???
  def readFileAsString(path: String): String = Source.fromFile(path).getLines.mkString(" ")
}
```


Full code

```
object wordCount extends zio.App {

  def run(args: List[String]) =
    myAppLogic.exitCode

  def readFileAsString(path: String): Task[String] =
    ZIO.effect(Source.fromFile(path).getLines.mkString(" "))

  def countWords(str: String): UIO[Int] = UIO(str.split(" ").length)

  val myAppLogic =
    for {
      _ <- putStrLn(
        "Hello! What path do you want to word count? please enter full path")
      fullPath <- getStrLn
      contents <- readFileAsString(fullPath)
      count <- countWords(contents)
      _ <- putStrLn(count.toString)
    } yield count

}
```

Summary

- >> Unpure code is wrapped in `ZIO.effect`
- >> Pure values are wrapped in `UIO`

Scheduling & retrying

- >> error handling
- >> Scheduling & retrys

```
ZIO.effectTotal(println(1)).orDie
ZIO.effectTotal(zio.console.putStrLn("hello world").repeatN(10))
zio.console.putStrLn("heloo world").repeat(Schedule.forever)
zio.console.putStrLn("heloo world").repeat(Schedule.recurs(10) andThen Schedule.spaced(1.second))
```

