This Scala notebook uses *BeakerX*, a Two Sigma Open Source project that enhances Jupyter.

http://beakerx.com/ (http://beakerx.com/)

In [1]:
```
scala.util.Properties.versionMsg
```

Out[1]:
```
Scala library version 2.11.12 -- Copyright 2002-2017, LAMP/EPFL
```

We'll be using the JFiglet library in our example, so we'll add it via Maven:

In [2]:
```
%classpath add mvn com.github.lalyos jfiglet 0.0.8
```

Defining a MessagePrinter

We'll create a simple class that takes a message and then prints it to the console.

In [3]:
```
class MessagePrinter {
  def print(msg: String): Unit = println(msg)
}
```

Out[3]:
```
defined class MessagePrinter
```

In [4]:
```
(new MessagePrinter).print("hello world")
```

```
hello world
```

Out[4]:
```
null
```

Our `MessagePrinter` isn't very interesting. Let's spice it up with some mixins.

Defining some Mixins for MessagePrinter

In [5]:
```
trait Reverse extends MessagePrinter {
  override def print(msg: String): Unit =
    super.print(msg.reverse)
}
```

Out[5]:
```
defined trait Reverse
```

In [6]:
```
(new MessagePrinter with Reverse).print("hello world")
```

```
dlrow olleh
```

Out[6]: null

In [7]:
```
trait TitleCase extends MessagePrinter {
  override def print(msg: String): Unit =
    super.print(msg.split(' ').map(_.capitalize).mkString(" "))
}
```

Out[7]: defined trait TitleCase

In [8]:
```
(new MessagePrinter with TitleCase).print("hello world")
```

```
Hello World
```

Out[8]: null

In [9]:
```
trait Banner extends MessagePrinter {
  import com.github.lalyos.jfiglet.FigletFont

  override def print(msg: String): Unit =
    super.print(FigletFont.convertOneLine(msg))
}
```

Out[9]: defined trait Banner

In [10]:
```
(new MessagePrinter with Banner).print("hello world")
```

```
 _               _   _                                          _       _
| |__     ___   | | | |   ___       __      __   ___    _ __   | |   __| |
| '_ \   / _ \  | | | |  / _ \      \ \ /\ / /  / _ \  | '__|  | |  / _` |
| | | | |  __/  | | | | | (_) |      \ V  V /  | (_) | | |     | | | (_| |
|_| |_|  \___|  |_| |_|  \___/        \_/\_/    \___/  |_|     |_|  \__,_|
```

Out[10]: null

Stacking Mixins on MessagePrinter

In [11]:
```
(new MessagePrinter with TitleCase with Reverse).print("hello world")
```

```
Dlrow Olleh
```

Out[11]: null

In [12]:
```
(new MessagePrinter with Reverse with TitleCase).print("hello world")
```

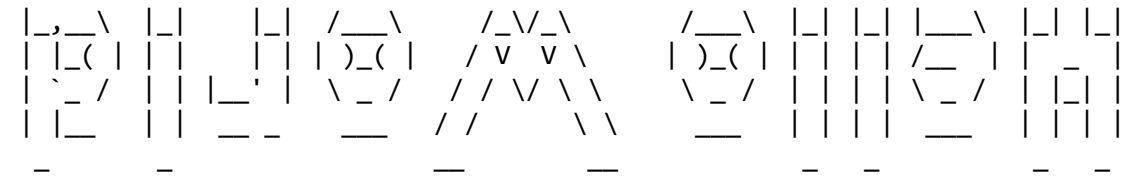
```
dlroW olleH
```

Out[12]: null

In [13]:
```
(new MessagePrinter with Banner with Reverse with TitleCase).print("hello worl
d")
```

Out[13]: null

In [14]:
```
(new MessagePrinter with Reverse with Banner with TitleCase).print("hello worl
d")
```



Out[14]: null

In [15]:
```
class BannerTitleMessagePrinter extends MessagePrinter with Banner with TitleC
ase
```

Out[15]: defined class BannerTitleMessagePrinter

In [16]:
```
(new BannerTitleMessagePrinter).print("hello world")
```

Out[16]: null

Using Abstract Overrides

In [17]:
```
abstract class MessageWriter {
    def write(msg: String): Unit
}
```

Out[17]: defined class MessageWriter

If we try declaring a trait that overrides an abstract method, it doesn't work:

In [18]:
```
trait BadReverse extends MessageWriter {
    override def write(msg: String): Unit =
        super.write(msg.reverse)
}
```

```
<console>:14: error: method write in class MessageWriter is accessed from sup
er. It may not be abstract unless it is overridden by a member declared `abst
ract' and `override'
       super.write(msg.reverse)
             ^
```

Instead, we need to use `abstract override` to tell the compiler that we really want to do this:

In [19]:
```
trait GoodReverse extends MessageWriter {
    abstract override def write(msg: String): Unit =
        super.write(msg.reverse)
}
```

Out[19]: defined trait GoodReverse

Now we can create a concrete implementation of our abstract class and use it with our mixin:

In [20]:
```
class ConsoleMessageWriter extends MessageWriter {
    override def write(msg: String): Unit =
        println(msg)
}
```

Out[20]: defined class ConsoleMessageWriter

In [21]:
```
(new ConsoleMessageWriter).write("hello world")
```

```
hello world
```

Out[21]: null

In [22]:
```
(new ConsoleMessageWriter with GoodReverse).write("hello world")
```

```
dlrow olleh
```

Out[22]: null