Jupyla Progress report:

First attempt:

- Plan to use Ammonite for scala REPL to interpret code
- Try to use Scalatra as a web server
    - Scalatra version is not compatible with scala-binding-Zeromq dependencies
- Successfully creating fake frontend with Jupyter (no kernel code and no connection), basically creating connection file in .ipython/.

New attempt:

- We try to follow Simple Kernel in python language as a guide for connection and communication with Jupyter.
- **JupylaKernel.scala**
    - First step, we write a kernel generator file in bash to register kernel for jupyter notebook. This kernel contains our kernel information such as display name, argument list that contains path to kernel file and kernel file name, and lastly, language that kernel used.
    - Once we get the kernel registered, we mostly stuggled and confused of what we should do. After read document from several places, we still didn't figure out the path to success.
    - Eventually, we gets connection_file path from kernel that we register earlier, and parse connection information out from json file from connection_file.
    - After we got all connection details, to make something that can connect to jupyter notebook
    - To connect to jupyter notebook which use ZeroMQ to communicate with kernel, we try to find several ZeroMQ library from several places. However, we not really know which one is better than which. So we try several ZeroMQ library (jzmq, scala-zeromq, jeromq, zeromq-scala-binding).
    - zeromq-scala-binding is a library that we choose because it is shown on zeromq official website (still not working ka Aj. TwT).
    - After several try, we finally gets heartbeat partially working. After we try to refractor the code and try to make heartbeat fully working by using zmq protocal we gets error with zmq.device (use to echo the message that receive from jupyter). : (

- **Interpreter.scala**
    - Representation :
        - Each **Array**[String] = Each **cell** , “String” is the code part.
        - **List**[Array[String]] = **Queue** contains cells.
    - Before Interpreter.handler is called, we will keep a code for each cell as an Array[String] and keep it in a List in an ascending cell order, eg. List[Array[String]]. So the code will run from top to bottom as the “Run All” behavior should be.
        - We use scala built-in Interpreter to interpret code. We looked at Iscala as a guide
            - We try to use both PrintWriter and StringWriter to get results.
            - We can print out the results to the terminal but still cannot retrieved the result as a string in order to sent it back to jupyter
            - We asumed that the code Interpreter.handler get is already parsed from Json file which is a List[Array[String]].
            - For many cells (run all), we loop over the list and interpret each array.
            - If it has only one cell, the list will contains only one array, eg. List(Array(“var a = 0”))