

Ammonite “Quirks”

While Ammonite is great when it comes to additional features, such as multiple main’s and dynamically loading files, and in the past it was a rock-solid piece of software written by one of my Scala-heroes (Haoyi Li), it seems it has acquired some “quirks” when it transitioned to Scala 3. Interestingly, the problems listed below only occur inside the REPL. So running Scala-scripts on the command line with for example

```
$ amm cw02.sc
```

works fine. Similarly, the original Scala REPL seems to not suffer from these quirks. Unfortunately, this REPL does not allow you to dynamically load files which might be relevant in later CWs.

Problem with Implicits

In CW2 (and possibly others), if you use the template code and develop your code inside the REPL, a problem arises in the line

```
val KEYWORD : Rexp = "while" | "if" | "then"
```

with the error

```
-- [E008] Not Found Error: cmd17.sc:1:36 -----
1 | val KEYWORD : Rexp = "while" | "if" | "then"
  | ~~~~~
  | value | is not a member of ammonite.$sess.cmd4.ALT.
  | An extension method was tried, but could not be fully constructed:
  |
  |     ammonite.$sess.cmd16.|()
  |
  |     failed with:
  |
  |         value |: <overloaded ammonite.$sess.cmd16.|> does not take
  |         parameters
  | Compilation Failed
```

Fix: The code runs absolutely fine when called on the commandline. It also runs absolutely fine inside the standard Scala REPL. If you prefer to use to use the Ammonite REPL, then you need to make the following changes: (1) comment out the implicits below and change the \$ to for example &

```

extension (s: String) {
  def & (r: Rexp) = RECD(s, r)
  //def | (r: Rexp) = ALT(s, r)
  //def | (r: String) = ALT(s, r)
  //def % = STAR(s)
  //def ~ (r: Rexp) = SEQ(s, r)
  //def ~ (r: String) = SEQ(s, r)
}

```

You need to then use `&` in the code for the `WHILE_REGS`.

Problem with Loading Dynamically Files

There seems to be cache problems with loading dynamically files in the REPL. For example in the file `thompson.sc`, the code for DFAs, NFAs etc is loaded using:

```

import $file.dfa, dfa._
import $file.nfa, nfa._
import $file.enfa, enfa._

```

The menace is that all works perfectly in the first REPL session, but if you close it and then re-open it for example the next day, you receive the following error when you try to re-evaluate the same lines:

```

-- [E008] Not Found Error: ~/cfl-material/progs/automata/nfa.sc:50:2
50 |   dfa
    |   ^^^
    |   value dfa is not a member of ammonite.$sess - did you mean $sess.nfa?
-- [E006] Not Found Error: ~/cfl-material/progs/automata/nfa.sc:6:18
6 | import $file.$ , dfa._
  |               ^^^
  |               Not found: dfa
  |
  | longer explanation available when compiling with `--explain`
Compilation Failed

```

Fix: The code again runs absolutely fine when called on the commandline. But the REPL trips over. In this case, the trick with using the Scala REPL does not work, because it does not support dynamically loading files. However, one ugly fix is to cut-and-paste all the code into a single file and then develop this one gigantic file. Or a slightly less ugly solution is to clear out the cache of ammonite. The cache on my system is located in

```
~/ammonite/cache/3.0.3/
```

I usually delete all files inside this directory and in this way force ammonite to be in a consistent state.

Warnings in CW2 (A)

If you introduce new values for example for the `NTIMES` and `PLUS` regular expressions, then do not call the values `Ntimes`, `Plus`. Rather call them `Pls` and `Ntms`. This avoids problems for filesystems that do not distinguish between uppercase and lowercase letters.

Warnings in CW2 (B)

If you get a warning about `$` being not defined infix, then define the extension method with

```
infix def $ (r: Rexp) = RECD(s, r)
```

Outdated Syntax in CW4

The templates for this task contain the lines

```
def i(args: Any*): String = "  " ++ sc.s(args:_) ++ "\n"
def l(args: Any*): String = sc.s(args:_) ++ ":\n"
```

which produce a warning about “The syntax ‘x: _’ is no longer supported for vararg splices”. This warning can be avoided by writing `args*` instead of `args:_`.

Compile_Block not Found in Compile.sc

When you evaluate in Ammonite parsers or the `eval-` and `compile-` functions, you encounter the following problem: For example in the file `compile.sc` the function `compile_stmt` function depends on `compile_block` function and vice versa. This means you need to evaluate them together when you run the code inside the REPL (this problem does not occur when you run the complete file on the command line). You can evaluate functions in a big “chunk” by enclosing it inside `{...}`.