Scanduc|25

Regular Expressions

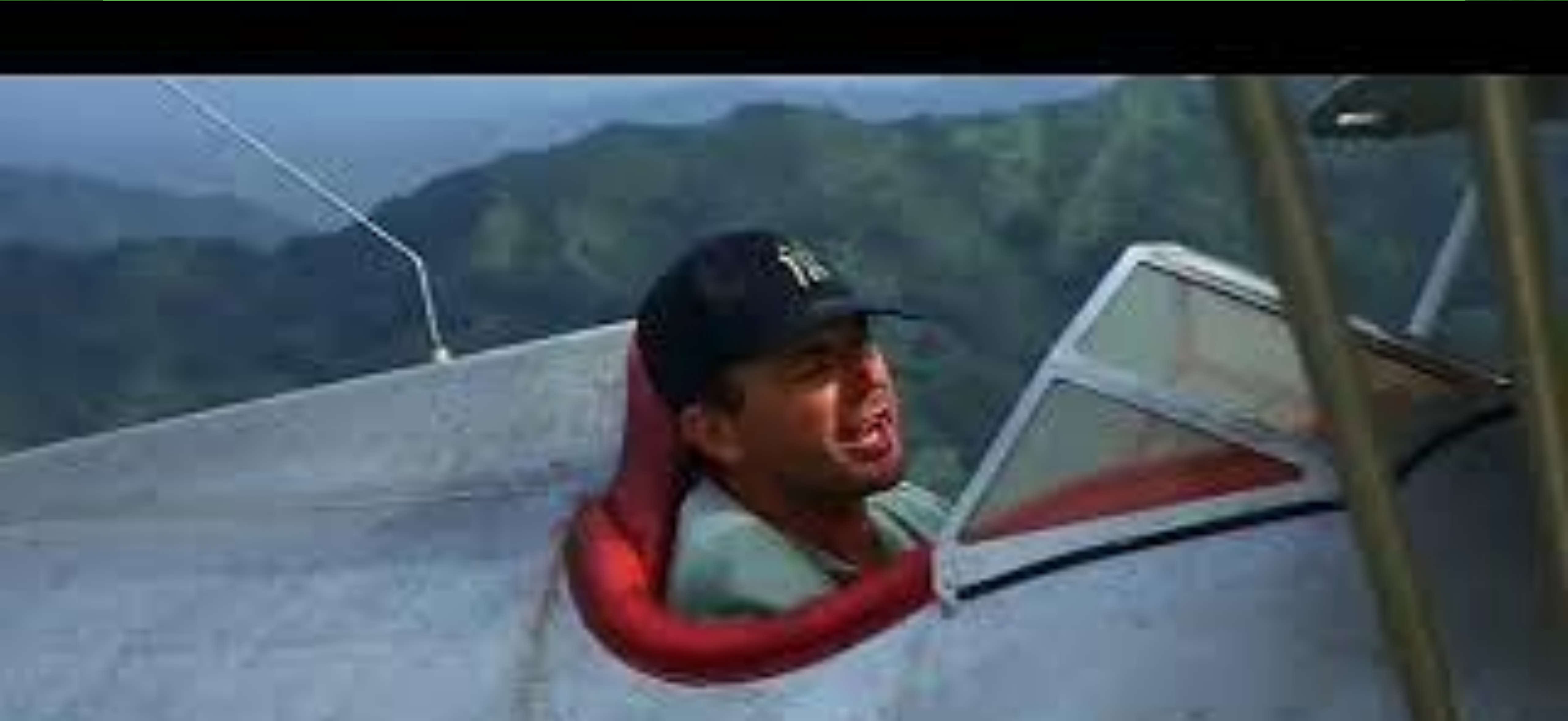
Mike Peat, Unicorn InterGlobal

So how did this happen?

- I'd promised Charlotte I'd do a presentation here at Scanduc...
- However I was short of ideas for a topic...
- So Sture suggested I give do something on the new Regular Expression facilities (RegEx) in DataFlex 25...
- But, but...

RegExs?
Why did it have to be RegExs?

I hate RegExs, Jock!
I hate 'em!



Snakes…

- In my youth I fell into a box of wriggling RegExs
- …and I've had an abiding horror of them ever since
- But as the resolute action hero you know me to be…
- I guess I'll have to overcome my fear, drop into the pit, spray them with kerosene and set them on fire!
- So here goes… 😓

Unix

- I first encountered RegEx in the mid-80s when I was learning Unix
- I've used them since, in pretty simple ways, to do find, or find-and-replace, in the **vi** text editor (or **sed** stream editor)
- And to filter the output of other commands such as **ps** with **grep** (named for the g/re/p command in the **ed** line editor which printed all matching lines)
- In spite of that, I've always considered RegEx black magic! 🤪

So what *is* RegEx?

- RegEx is essentially a string pattern-matching mechanism
- It has spawned several "*dialects*" in which to express the "pattern" you want to match
- Of these Data Access has chosen the popular Perl Compatible Regular Expressions - PCRE - library to build into the DataFlex runtime
- They have provided a class: cRegEx, with properties and methods you can use to find (and act on) matches in text

The RegEx class

- The property psExpression holds your regular expression
- Then the functions:
  - Match and Substitute
  - MatchAll and SubstituteAll
  - MatchAllCallback and SubstituteAllExCallback
  - MatchAllOffsets
  - RMatch
  - Split
  - MatchAllGroups
- Carry out those operations on a string of passed text

So let's see some of those in action:

DEMO!

Th RegEx expressions

- As you can see from that monster EMail-address-identifying-and-validating expression, writing comprehensive RegEx can be quite challenging!
- (FYI: I copied it from https://emailregex.com/index.html and it is the same one Harm uses in his Learning Center course on RegEx. It is only claimed to be 99.99% effective - it notes: *there is no perfect email regex*)
- So… now it is time to *spray those RegExs with kerosene and set them on fire…* 🔥 😉
- Let's look at writing our own
- First, the tools we have at our disposal…

RegEx matching syntax (PCRE version)

Symbol	Matches
. (dot/period)	any character (ex: \n)
ab	"ab"
a|b	"a" or "b" (logical or)
a*	zero or more "a"s
a+	one or more "a"s
a?	zero or one "a"
a{3}	3 "a"s
a{3,}	3 or more "a"s
a{3,9)	3 to 9 "a"s
\	escape special char
\d	one digit
\D	one non-digit

Symbol	Matches
\s	one whitespace
\S	one non-whitespace
\w	one "word" character
\W	one non-word character
\n	newline
\r	carriage return
\t	tab
\b	word boundary
\B	non-word boundary
[b-q]	character in set (range)
[^b-q]	char not in set
[\b]	backspace

Symbol	Matches
^	start of string
$	end of string
\<	start of word
\>	end of word
(...)	capturing group
(?: ...)	non-capturing group
(?<*xyz*>...)	named group "xyz"
(?#...)	comment
\0	null
\YYY	octal char "YYY"
\xYY	hex char "YY"
\cY	ctrl-character "Y"

So to validate an email address…

- An email address is made up of three parts:
  - the local part (mailbox name)
  - an "@" sign
  - the email domain, itself made up of:
    - 1 or more subdomains, ending in dots (".")
    - A top level domain (TLD)
- You can also have dots and other chars in the local part:
  - M.Peat or M-Peat or m_peat - the complete list is:

    .!#$%&'*+-/=?^_`{|}~

What we'll try

- We don't actually want to use the: "*Firstname Surname <email-address>*" form so we won't allow for that
- The "local part" can only be a maximum of 64 characters, so we can use the quantifier {1,64} on that
- The TLD cannot contain anything but letters and must be at least 2 characters long; the longest at present is "*travelersinsurance*", but we can accommodate any length by using a quantifier of {2,}

What we'll try

- \w will cover most of what we want to allow in local and subdomains and we can add to that as a range: [\w...] as required (\w is very useful: all alphanums plus underscore)
- (Note: you do not need to escape most special characters in a [...] range)
- We can use word-boundaries (\b) to identify the start and end of what we are looking for
- Each subdomain will be a series of 1-63 characters followed by a dot, so we can make that a group with a quantifier of {1,63} chars followed by \.
- However we have to take care with groups: (...)

What we'll try

- If we use a *capturing* group, MatchAllCallback will call its callback function for the matches but also for any groups it finds, so we need use a *non-capturing group*: (?:...)
- I've never come across an email address with more than three subdomains, so let's call the limit on those 6 (the *actual* limit is 125, but that's just silly!)
- There is always one, so we can use a quantifier of {1,6}
- I intend to use the "audience-debugger", so pay attention and shout out when you see me go wrong!

So let's have a go!

(What could *possibly* go wrong? 😈)

Clearer about RegEx?

- Our expression was: `` \b[\w.!#$%&'*+-/=?^`{|}~]{1,64}\b@(?:\w{1,63}\.){1,6}[a-zA-Z]{2,}\b ``
- I hope that has left you a little less mystified by RegEx expressions
- Personally I still consider them to mostly be a "*write-only*" form of programming
- It is worth remembering, if you are faced with some specific RegEx problem, that somebody somewhere might have already solved it and documented that…
- Just Google it
- Or failing that, ask on Stack Overflow! 😃

Thank you!

Are there any questions?