Regex

The second language concept we will look at is regular expressions (regex). We have already presented some studies on their use, and potential problems, above. The lecturecast also contains a useful link to a tutorial on creating regex. Re-read the provided links and tutorial (Jaiswal, 2020) and then attempt the problem presented below:

- The UK postcode system consists of a string that contains a number of characters and numbers – a typical example is ST7 9HV (this is not valid – see below for why). The rules for the pattern are available from idealpostcodes (2020).

- Create a python program that implements a regex that complies with the rules provided above – test it against the examples provided.

- Examples:

- M1 1AA

- M60 1NW

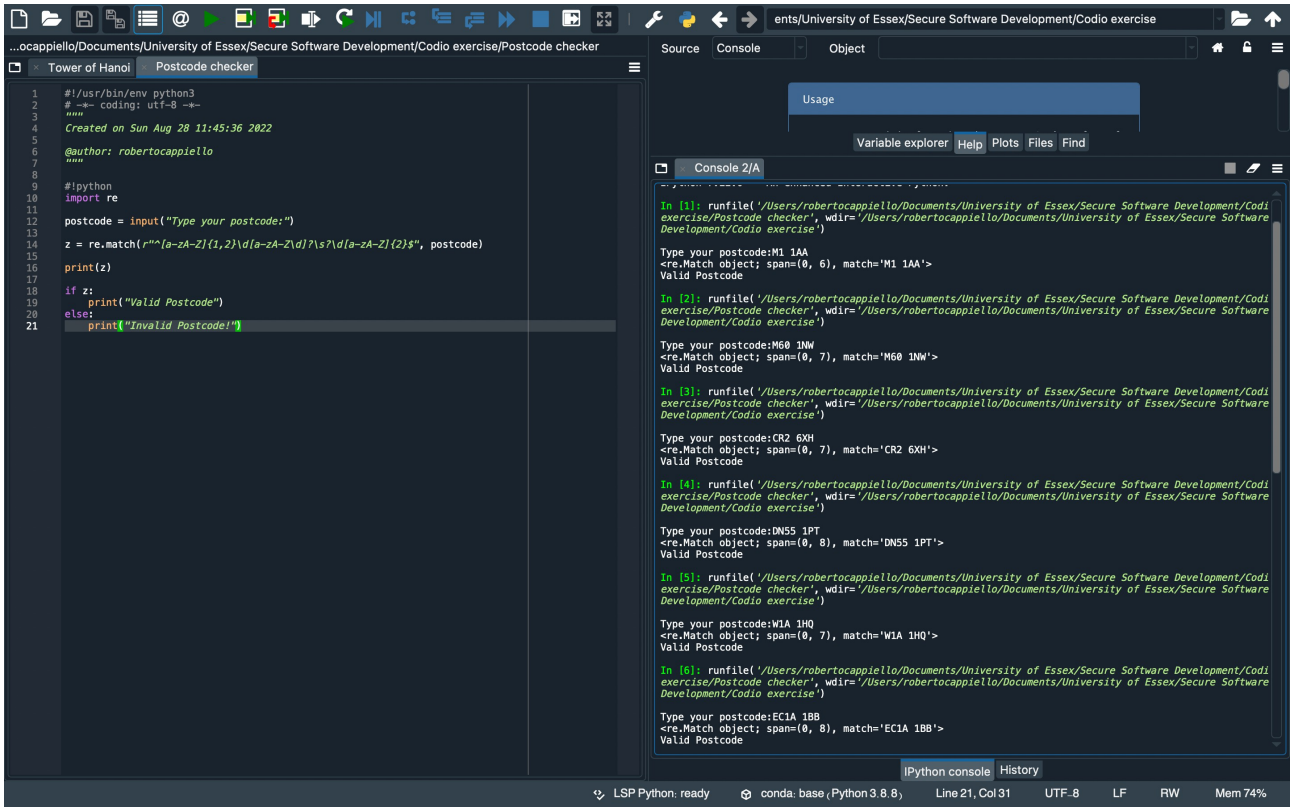
- CR2 6XH

- DN55 1PT

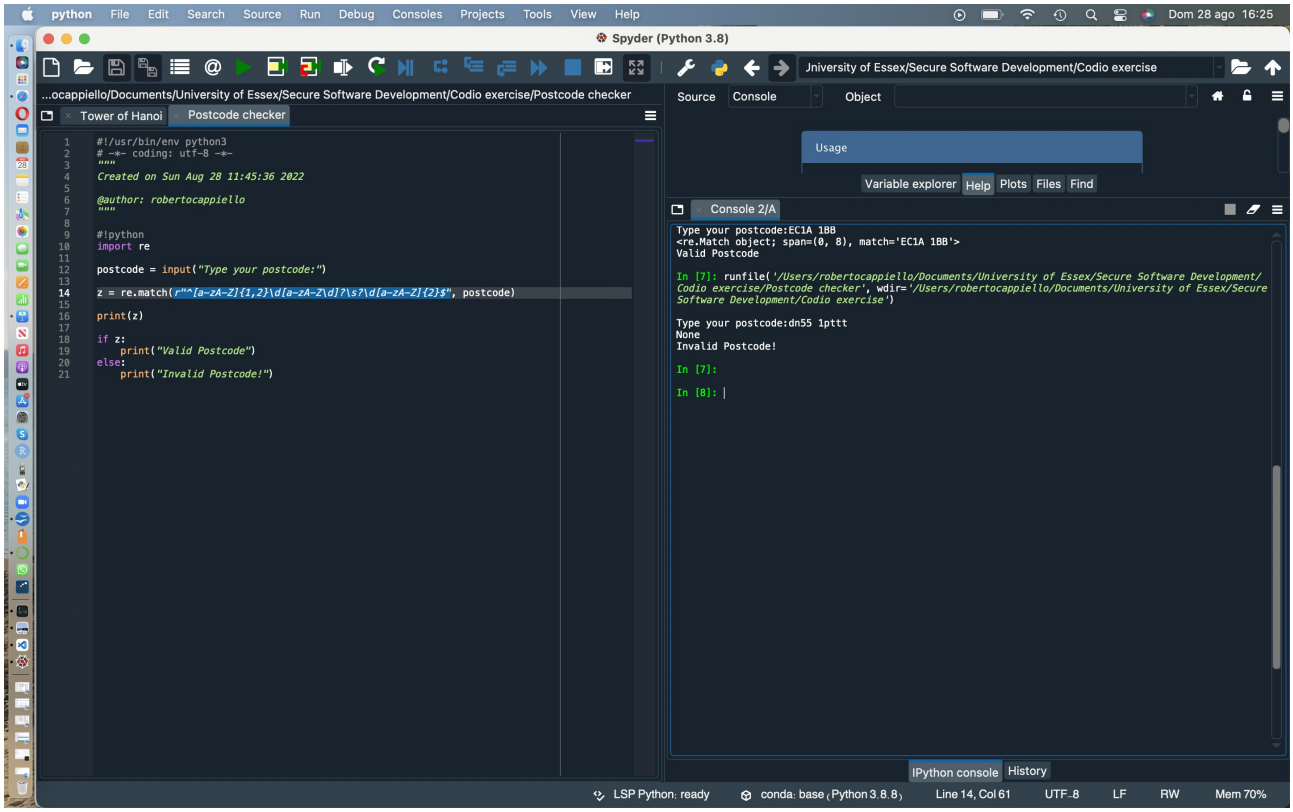
- W1A 1HQ

- EC1A 1BB

Here is a screenshot proving that the code is working and the above postcodes are being tested and valid UK postcodes



here is an example of recognising an invalid postocde



•How do you ensure your solution is not subject to an evil regex attack?

A according to OWASP it is possible to avoid Evil Regex pattern by avoiding:

- •Grouping with repetition
- •Inside the repeated group:
- •Repetition
- •Alternation with overlapping

Examples of Evil Regex:

- •`(a+)+`
- •`([a-zA-Z]+)*`
- •`(a|aa)+`
- •`(a|a?)+`
- •`(.*a){x} for x \> 10`

All the above are susceptible to the input `aaaaaaaaaaaaaaaaaaaaaaaa!` (The minimum input length might change slightly, when using faster or slower machines).

References

owasp.org. (n.d.). *Regular expression Denial of Service - ReDoS | OWASP*. [online] Available from https://owasp.org/www-community/attacks/Regular_expression_Denial_of_Service_-_ReDoS