Unit 4

Seminar 2 – Programming Language Concepts

Recursion

One of the classic programming problems that is often solved by recursion is the towers of Hanoi problem. A good explanation and walkthrough are provided by Cormen & Balkcom (n.d.) - the link is in the reading list. (the code they used for their visual example is provided on their website as well).

- Read the explanation, study the code and then create your own version using Python (if you want to make it more interesting you can use asterisks to represent the disks). Create a version that asks for the number of disks and then executes the moves, and then finally displays the number of moves executed.

```
left = []
middle = []
right = []

def disk_mover(disks, source_rod, target_rod, spare_rod):
    if (disks == 1):
        value = source_rod.pop()
        target_rod.append(value)

        print(f"L: {str(left).ljust(max_padding)} M:
{str(middle).ljust(max_padding)} R:
{str(right).ljust(max_padding)}")

        return 1

    # This is the recursive portion of the function
    moves = disk_mover(disks - 1, source_rod, spare_rod,
target_rod)

    value = source_rod.pop()
    target_rod.append(value)
```

```
    print(f"L: {str(left).ljust(max_padding)} M:
{str(middle).ljust(max_padding)} R:
{str(right).ljust(max_padding)}")

    moves += 1
    moves += disk_mover(disks - 1, spare_rod, target_rod,
source_rod)

    return moves


disks_to_move = int(input("Enter number of disks to move: "))

temp = disks_to_move
while temp > 0:
    left.append(''.join([char*temp for char in '*']))
    temp -=1

max_padding = len(str(left))

total_moves = disk_mover(disks_to_move, left, right, middle)
print (f"Steps executed: {total_moves}")
```

- What is the (theoretical) maximum number of disks that your program can move without generating an error?

  Since the Towers of Hanoi recursion problem takes at most 2^n-1 steps to complete, theoretically the maximum number of disks that can be moved is (providing stack overflows are no issue) would be infinite, however the larger the number of disks to move, the greater the amount of time required to complete the puzzle.

  The maximum number of disks that can be moved *without* generating an error seems to be around 15 disks, that results in a total number of steps executed, as 32767.

- What limits the number of iterations? What is the implication for application and system security?

- Number of iterations is limited by the **recursion** depth allowed by the Python IDE, which is set by default to 1000. However this can be changed in Python via the following statements

```
import sys
sys.setrecursionlimit(100000)
```

  According to the Python documentation, “The highest possible limit is platform-dependent”. (Python, 2021).

- Another cause for limited iterations is whether or not developers incorporate safe **condition checks** in their algorithm such as

```
    if (disks <= 0):
        return 0
```

The limiting of recursion depth impacts system security in that it ensures rogue code is never permitted to endlessly consume all CPU resources for a hack-style algorithm. Such limitation should be seen as a *benefit* to system security, however, it is important that the recursion limit is set to a reasonable depth level for operational software, as setting it too low may raise multiple RecursionErrors.

RegEx

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).

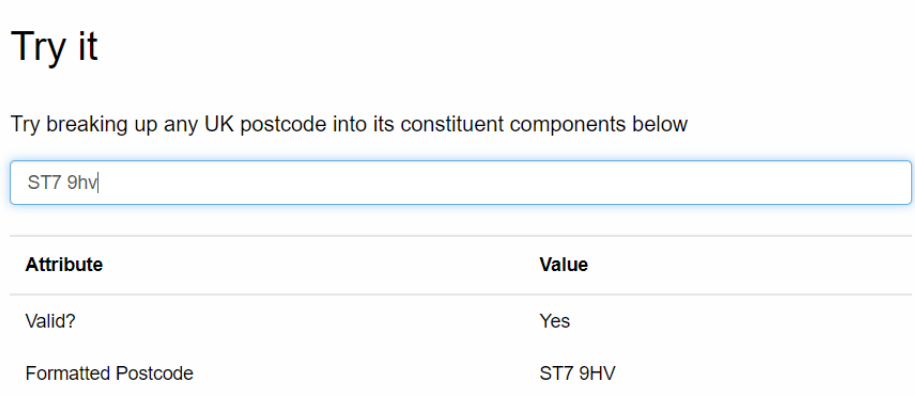
- Create a python program that implements a regex that complies with the rules provided above – test it against the examples provided. Examples:

  - M1 1AA
  - M60 1NW
  - CR2 6XH
  - DN55 1PT

- W1A 1HQ
- EC1A 1BB

```
import re

def postcodes(codes):
    if type(codes) is not list:
        print("Please provide a list of UK-style postcodes.")
        return

    rex = re.compile('[A-Za-z]{1,2}\d[A-Za-z\d]?\s*\d[A-Za-z]{2}$')

    for code in codes:
        match = rex.match(code)
        print(f"{code} is valid? ".ljust(22) + ("Yes" if match != None else
"No"))

postcodes(['ST7 9HV', 'M1 1AA', 'M60 1NW', 'CR2 6XH', 'DN55 1PT', 'W1A
1HQ', 'EC1A 1BB'])
```

Note: Even though ST7 9HV was stated as "not valid" earlier, testing this postcode on the referenced site (Ideal Postcodes) reveals that it does work. Therefore, the Python code was updated using `[A-Za-z\d]?` *to ensure the "9" is correctly picked up according to the post code rules.*



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

To minimize the chances of a ReDOS attack, one can consider

- alternative llibraries such as pyre2 from Facebook (pyre2, 2021) that compiles regular expressions to **deterministic finite automata**, guaranteeing linear-time behaviour.
- Craft regular expressions that do not rely on **backtracking.**
- Limit the length of input to perform regular expression matching on.
- Utilisize **atomic groupings** that automatically throw away backtracking information.
- Automatically terminate long-running regular expressions and log the termination reason.

References

Python (2021) Python Runtime Services. Available from https://docs.python.org/3/library/sys.html [Accessed 3 Sep 2021]

Pyre2 (2021) pyre2 0.3.6. Available from https://pypi.org/project/pyre2/ [Accessed 3 Sep 2021]

Bibliography

Cimpanu, C. (2018) JavaScript Web Apps and Servers Vulnerable to ReDOS Attacks. Available from https://www.bleepingcomputer.com/news/security/javascript-web-apps-and-servers-vulnerable-to-redos-attacks/ [Accessed 3 Sep 2021]

DelftStack (2020) Get and Increase the Maximum Recursion Depth in Python. Available from https://www.delftstack.com/howto/python/how-to-get-and-increase-the-maximum-recursion-depth-in-python/ [Accessed 3 Sep. 2021]