Unit 4 Seminar

Title: Programming Language Concepts

Please note that the Jupyter Notebook environment is available in Codio for you to carry out these activities. This week, two programming exercises will help you understand two valuable language concepts – recursion and regex.

Recursion

One of the classic programming problems often solved by recursion is the towers of Hanoi problem. Cormen & Balkcom (n.d.) provide a reasonable explanation and walkthrough - 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 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, executes the moves, and displays the number of actions performed.
- What is the (theoretical) maximum number of disks your program moves without generating an error?
- What limits the number of iterations? What is the implication for application and system security?

Regex

The second language concept we will examine is regular expressions (regex). We have already presented some studies on their use and potential problems above. The lecture cast also contains a helpful 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 several 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 ideal postcodes (2020).
- Create a Python program that implements a regex that complies with the above rules – test it against the examples provided.

Examples:

- M1 1AA
- M60 1NW
- CR2 6XH
- DN55 1PT
- W1A 1HQ
- EC1A 1BB

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

You can share your responses with the tutor for formative feedback or discuss them in this week's seminar. There will also be an opportunity to review your team's progress during the seminar.

Remember to record your results, ideas and team discussions in your e-portfolio.

Learning Outcomes

- Identify and manage security risks as part of a software development project.
- Critically analyse development problems and determine appropriate methodologies, tools and techniques (including program design and development) to solve them.
- Design and develop/adapt computer programs and produce a solution that meets the design brief and critically evaluate solutions that are produced.

Recursion

1. Here is my Python code representing the Towers of Hanoi based on the ideas of Invent with Python (N.D.) and (GeeksforGeeks, 2015):

```
"""
import module
"""
import sys

def tower_of_hanoi(num_disks, source_pole, destination_pole, aux_pole, poles):
    """
    Solve the Tower of Hanoi problem using recursion.
    """
    if num_disks == 1:
        print(f"Move disk 1 from pole {source_pole} to pole {destination_pole}.")
        return

    # Move num_disks-1 disks from source to auxiliary pole
    tower_of_hanoi(num_disks - 1, source_pole, aux_pole, destination_pole, poles)

    # Move the largest disk from the source to the destination pole
    poles[destination_pole].append(poles[source_pole].pop())
    print(f"Move disk {num_disks} from pole {source_pole} to pole {destination_pole}.")

    # Move num_disks-1 disks from auxiliary to destination pole
    tower_of_hanoi(num_disks - 1, aux_pole, destination_pole, source_pole, poles)

def print_moves(num_disks):
    """
    Print information about the Tower of Hanoi solution.
    """
    total_moves = 2 ** num_disks - 1
    print(f"Recursion limit: {sys.getrecursionlimit()}")
    print(f"Total moves required: {total_moves}")

def main():
    """
    Main function to execute the Tower of Hanoi program.
    """
    number_of_disks = int(input("Enter the number of disks: "))
    source_pole = "A"
    destination_pole = "B"
    aux_pole = "C"

    poles = {
        source_pole: [i for i in range(number_of_disks, 0, -1)],
        destination_pole: [],
        aux_pole: []
    }

    tower_of_hanoi(number_of_disks, source_pole, destination_pole, aux_pole, poles)
    print_moves(number_of_disks)


main()
```

Output:

```
C:\Users\hcham\anaconda3\envs\pythonProject-SSD\python.exe
C:\Users\hcham\PycharmProjects\pythonProject-SSD\recursion.py
Enter the number of disks: 5
Move disk 1 from pole A to pole B.
Move disk 2 from pole A to pole C.
Move disk 1 from pole B to pole C.
Move disk 3 from pole A to pole B.
Move disk 1 from pole C to pole A.
Move disk 2 from pole C to pole B.
Move disk 1 from pole A to pole B.
Move disk 4 from pole A to pole C.
Move disk 1 from pole B to pole C.
Move disk 2 from pole B to pole A.
Move disk 1 from pole C to pole A.
Move disk 3 from pole B to pole C.
Move disk 1 from pole A to pole B.
Move disk 2 from pole A to pole C.
Move disk 1 from pole B to pole C.
Move disk 5 from pole A to pole B.
Move disk 1 from pole C to pole A.
Move disk 2 from pole C to pole B.
Move disk 1 from pole A to pole B.
Move disk 3 from pole C to pole A.
Move disk 1 from pole B to pole C.
Move disk 2 from pole B to pole A.
Move disk 1 from pole C to pole A.
Move disk 4 from pole C to pole B.
Move disk 1 from pole A to pole B.
Move disk 2 from pole A to pole C.
Move disk 1 from pole B to pole C.
Move disk 3 from pole A to pole B.
Move disk 1 from pole C to pole A.
Move disk 2 from pole C to pole B.
Move disk 1 from pole A to pole B.
Recursion limit: 1000
Total moves required: 31

Process finished with exit code 0
```

2. The program uses recursion to solve the Tower of Hanoi puzzle for any number of disks, ensuring accuracy for all positive integers. Although the program has no theoretical limit on the number of disks it can handle, it is vital to note, as Pierce (2023) states, that a RecursionError in Python can arise if a function lacks a stopping condition and continues to call itself. To avoid such an error, the program has integrated a base case or an iterative (sys.getrecursionlimit) method into the code. Additionally, it is crucial to exercise caution when adjusting the recursion limit to prevent any potential system crashes, as Pierce (2023) mentioned.

3. In my code, the number of iterations (recursive calls) is determined by the value of the number of disks and follows the principles of the Tower of Hanoi problem. The number of iterations required to solve the Tower

of Hanoi problem with several disks is $2^n - 1$ (GeeksforGeeks, 2015), and this is calculated in the print_moves function as total_moves = 2 ** num_disks - 1.

When determining the optimal number of disks, it is imperative to factor in the potential impact on the system's security and applications. Employing a more significant number of disks may result in time complexity and heightened consumption of system resources, such as CPU and memory, due to the program running through multiple iterations (Study Smarter, N.D.). This could lead to performance complications or malfunction if the recursion depth surpasses the system's limitations. In Python, the default recursion limit is set to 1000 with the use of 'sys.getrecursionlimit()', but it can be customised to meet specific requirements (Jalli, 2021).

So, when working with significant inputs or recursive algorithms like this, it's essential to consider the potential resource consumption and recursion depth. Depending on the problem size and system constraints, you might need to optimise the code or use iterative approaches to avoid running into performance and security issues.

Regex

Using a regex, the UK government verifies UK postcodes. It comprises multiple letter-and-number combinations. To make geo-lookup more convenient, the regex might be simplified. The more straightforward regex may permit erroneous postcodes but limit input to the proper range (Burns, 2018). As a result, I have used the following regex '^[A-Z]{1,2}\d[A-Z\d]? ?\d[A-Z]{2}$' to validate the posts code. This is my code:

```
"""
Import regex library
"""

import re

def validate_uk_post_code(post_code):
    """
    validates the postcode pattern
    and return true or false
    """
    # UK postcode regex pattern
    pattern = r'^[A-Z]{1,2}\d[A-Z\d]? ?\d[A-Z]{2}$'
```

```
    # Use re.match to check if the post_code matches the pattern
    if re.match(pattern, post_code):
        return True
    else:
        return False


# Test cases
test_post_codes = [
    "M1 1AA",
    "M60 1NW",
    "CR2 6XH",
    "W1A 1HQ",
    "EC1A 1BB",
    "DN55 1P",  # Invalid postcode
    "12345",  # Invalid postcode
    "A1 123",  # Invalid post code
]

for post_code in test_post_codes:
    if validate_uk_post_code(post_code):
        print(f"{post_code} is a valid UK post code.")
    else:
        print(f"{post_code} is not a valid UK post code.")
```

Output:

```
C:\Users\hcham\anaconda3\envs\pythonProject-SSD\python.exe
C:\Users\hcham\PycharmProjects\pythonProject-SSD\regex.py
M1 1AA is a valid UK postcode.
M60 1NW is a valid UK postcode.
CR2 6XH is a valid UK postcode.
W1A 1HQ is a valid UK postcode.
EC1A 1BB is a valid UK postcode.
DN55 1P is not a valid UK postcode.
12345 is not a valid UK postcode.
A1 123 is not a valid UK postcode.

Process finished with exit code 0
```

I have implemented a validate_uk_post_code function within my code that uses a regex pattern to validate UK postcodes. To safeguard against potential malicious regex attacks, I have implemented the following precautions:

Firstly, I have limited the character sets to only allow valid characters in UK postcodes within the regex pattern. Specifically, I have used the following regex '/^[a-z]{1,2}\d[a-z\d]?\s*\d[a-z]{2}$/i;' from Postcodes (N.D.) to specify these valid characters.

Additionally, the regex uses ^ and $ anchors to ensure that it matches the entire input string and not just a part of it. This helps avoid any unintended matches.

Moreover, the regex has quantifiers like {1,2} to limit the repetition of characters, thus preventing excessive matching.

Lastly, I have correctly escaped any metacharacters within the regex pattern to avoid unintended behaviour.

References:

Invent With Python (N.D). *Chapter 14 - Practice Projects*. [online] Available at: https://inventwithpython.com/beyond/chapter14.html [Accessed 10 Sep. 2023].

GeeksforGeeks. (2015). *Iterative Tower of Hanoi*. [online] Available at: https://www.geeksforgeeks.org/iterative-tower-of-hanoi/.

Pierce, D. (2023). *How to Fix RecursionError in Python*. [online] Rollbar. Available at: https://rollbar.com/blog/python-recursionerror/# [Accessed 10 Sep. 2023].

Study Smarter (N.D.). *Tower of Hanoi Algorithm: Recursive Solution & Python Coding*. [online] Available at: https://www.studysmarter.co.uk/explanations/computer-science/algorithms-in-computer-science/tower-of-hanoi-algorithm/ [Accessed 10 Sep. 2023].

Jalli, A. (2021). *Python: Maximum Recursion Depth Exceeded [How to Fix It]*. [online] codingem.com. Available at: https://www.codingem.com/python-maximum-recursion-depth/.

Burns, A. (2018). *UK Postcode Validation Regex*. [online] Andy Burns' Blog. Available at: https://andrewwburns.com/2018/04/10/uk-postcode-validation-regex/ [Accessed 10 Sep. 2023].

Postcodes, U.P.L. and A.V.-I. (N.D.). *Postcode Validation*. [online] UK Postcode Lookup and Address Validation - Ideal Postcodes. Available at: https://ideal-postcodes.co.uk/guides/postcode-validation.