Laboratory work 5

Author: Dávid Kromka

Part 1:

For **Task 1**, the code finds prime roots modulo n and prints a table of reflections for each value of n in the n_values list. The algorithm works as follows:

1. Find Prime Roots Modulo n:
    - Loop through all integers a from 1 to n - 1.
    - For each a, calculate residues for values of x from 1 to n - 1 using the formula (a ** x) % n.
    - Store the residues in a list residues.
    - If the number of residues equals n - 1, add the list of residues to the prime_roots list.
    - Return the list of prime roots.
2. Print Table of Reflections:
    - Iterate through the prime_roots list for each n in n_values.
    - Print the prime roots for the current n.
    - Print a table header with indices from 1 to the length of prime roots.
    - For each x from 1 to n - 1, print the corresponding residues in the table.

For **Task 2**, the code finds possible values of x for given (a, y) combinations where (a ** x) % 7 = y. It then prints the combinations of a, possible x values, and y in a table. The algorithm works as follows:

1. Find Possible x Values:
    - Loop through integers x from 0 to 6.
    - For each x, calculate (a ** x) % 7 and check if it equals y.
    - If it matches, add x to the possible_x list.
2. Find (a, x, y) Combinations:
    - Iterate through predefined combinations of (a, y) in the combinations list.
    - For each combination, find possible values of x using the find_x function.
    - If there are possible x values, store the combination of (a, x_values) as the key and y as the value in the results dictionary.
3. Print Results in a Table:

- Print the header for the table.
- For each key-value pair in the results dictionary, print a, possible x values, and y in the table.

Output Part 1:

Task 1

Prime roots modulo 5 are: {1, 2, 3, 4}

Table of reflections for n = 5:

```
  x  |  1  2  3  4
----------------------
   1 |  1  1  1  1

   2 |  2  4  3  1

   3 |  3  4  2  1

   4 |  4  1  4  1
```

Prime roots modulo 11 are: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

Table of reflections for n = 11:

```
  x  |  1  2  3  4  5  6  7  8  9 10
----------------------------------------------
   1 |  1  1  1  1  1  1  1  1  1  1

   2 |  2  4  8  5  10  9  7  3  6  1

   3 |  3  9  5  4  1  3  9  5  4  1
```

4 | 4 5 9 3 1 4 5 9 3 1

5 | 5 3 4 9 1 5 3 4 9 1

6 | 6 3 7 9 10 5 8 4 2 1

7 | 7 5 2 3 10 4 6 9 8 1

8 | 8 9 6 4 10 3 2 5 7 1

9 | 9 4 3 5 1 9 4 3 5 1

10 | 10 1 10 1 10 1 10 1 10 1

Task 2

a	x	y
1	0, 1, 2, 3, 4, 5, 6	1
3	4	4
4	2, 5	2
5	3	6
6	1, 3, 5	6

Part 2:

1. Define a function called hex_to_decimal that takes a single argument hex_string, which represents a hexadecimal number. The function returns the decimal equivalent of the input hexadecimal number. This is achieved by using the built-in int() function with base 16, which converts the hexadecimal string to a decimal integer.

2. Create two lists: direct_replacement_table and reverse_swap_table, which store hexadecimal strings representing elements in two different tables.

3. Create two empty lists: direct_replacement_decimal and reverse_swap_decimal to store the decimal equivalents of the elements in the direct_replacement_table and reverse_swap_table, respectively.

4. Iterate over each element in the direct_replacement_table and reverse_swap_table using list comprehensions. For each element, call the hex_to_decimal function to convert the hexadecimal string to its decimal equivalent, and add the result to the respective list (direct_replacement_decimal or reverse_swap_decimal).

5. Print the results of the conversion for both tables in decimal notation.

Output Part 2:
Direct Replacement Table (Decimal Notation): [1, 3, 4, 5, 6, 13, 16]

Reverse Swap Decryption Table (Decimal Notation): [241, 243, 244, 246, 248, 255, 16]