Programming for Data Science (31/10/2023)

0% of the points are assigned to quality of documentation and/or comments to solutions. Solutions must include tests of executions of the developed functions.

Name files as "<your matricola>_<firstname>_<lastname>_ex1.py" for Exercise 1, and "<your matricola>_<firstname>_<lastname>_ex2.c" for the second exercise.

Upload the TWO files in a folder (named with your student number and your last name) at the following URL: Upload here (access GDrive using your university credentials)

Exercise 1. (Math, on paper)

Consider the following sets:

\[ R = \{ p \in \mathbb{Z} \mid -100 \leq p \leq 100 \} \]
\[ A = \{ m \in R \mid m \text{ is a multiple of 5} \} \]
\[ B = \{ n \in \mathbb{Z} \mid n^2 < 100 \} \]
\[ C = \{ 2x + 2 \mid x \in A \} \]

a) Which is the cardinality of the sets: \( A \cap B; B \cap C; \ A \cap B \cap C ? \)

b) List the elements of the set:
\[ D = \{ (x, y) \in (A \cap B) \times (B \cap C) \mid x \cdot y \leq 0 \} \]

c) Let's consider the function \( f: C \to \mathbb{Z} \) such that \( f(c) = c + 1 \) for every \( c \in C \). Determine if this function is injective, surjective, or bijective.

\[
\begin{array}{|c|c|c|}
\hline
A \cap B & B \cap C & D \\
\hline
-5 & 2 & (-5, 2) \\
0 & -8 & (-0, 2) \\
5 & & (0, -8) \\
\hline
\end{array}
\]

\[
\begin{array}{c|c|c}
|A \cap B| = 3 & |B \cap C| = 2 \\
|A \cap B \cap C| = 0 \\
\end{array}
\]

c) \( f \) is injective since \( f(m)=f(n) \) implies \( m+1=n+1 \) that implies \( m=n \), not surjective since not all \( \mathbb{Z} \) is image of \( f \), and hence \( f \) is not bijective since it is not (injective and surjective).

Exercise 2. (Python)

Implement the Exercise 1 in Python, according with the definition given in the previous exercise:

1. Define the three sets \( A, B \) and \( C \)
2. Create the new set $D$ made up of all tuples $(x, y)$, with $x \in (A \cap B)$ and $y \in (B \cap C)$, such that $x \cdot y \leq 0$

3. Create a function $\text{product}(s, n)$, taking a set $s$ of tuples $(x, y)$ and a number $n$ in input, and producing in output a new set resulting from the multiplication of $x$, $y$ and $n$. Test this function on the $D$ set and a number $n$ to be read from the user (only once, before the invocation of the function).

Solution:

def compute_D(A, B, C):
    inter_ab = A.intersection(B)
    inter_bc = B.intersection(C)

    D = []
    for a in inter_ab:
        for b in inter_bc:
            if a * b <= 0:
                D.append((a, b))
    return D

def product(s, n):
    r = set()
    for item in s:
        r.add(item[0] * item[1] * n)
    return r

R = set()
for i in range(-100, 100):
    R.add(i)

A = set()
for i in R:
    if i % 5 == 0:
        A.add(i)
B = set()
for i in range (-10, 10):
    if i**2 < 100:
        B.add(i)

C = set()
for i in A:
    C.add(2 * i + 2)

print("A: {}".format(A))
print("B: {}".format(B))
print("C: {}".format(C))
D = compute_D(A, B, C)
print("D: {}".format(D))
n = int(input("Insert a number: "))
r = product(D, n)
print("resulting set =", r)

Exercise 3. (C)

Write a C program that performs basic string manipulation on a user-entered string. The program should provide the implementation for each of the following operations:
1. Calculate the length of the string (without termination character \0)
2. Reverse the string.
3. Convert the string to uppercase.
4. Check if the string is a palindrome (reads the same forwards and backward).
Prompt the user to input a string and then display the result of each operation. The aforementioned operations should be implemented without exploiting the c string functions.

Solution:
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
#include <stdbool.h>
int length_string(char* str) {
    int count = 0;
    while(str[count] != '\0')
        count++;
    return count;
}

char* reverse_string(char* str, int lun) {
    char* new_str = (char *) malloc(lun * sizeof(char));
    for (int i=0; i<lun; i++)
        new_str[i] = str[lun-i-1];
    new_str[lun] = '\0';

    return new_str;
}

char* uppercase_string(char* str, int lun) {
    char* new_str = (char *) malloc(lun * sizeof(char));
    for (int i=0; i<=lun; i++) {
        if (str[i] >= 'a' && str[i] <= 'z')
            new_str[i] = str[i] - ('a' - 'A');
        else
            new_str[i] = str[i];
    }
    return new_str;
}

bool check_palindrome(char* str, int lun) {
for (int i=0; i<=ceil(lun/2); i++) {
    if (str[i] != str[lun-i-1])
        return false;
}
return true;

int main() {
    char* str = (char *) malloc(100 * sizeof(char));
    int length;
    char* reversed, *uppercase;

    printf("Insert a string:");
    scanf("%s", str);

    length = length_string(str);
    printf("The length of the string is: %d\n", length);

    reversed = reverse_string(str, length);
    printf("The reversed string is: %s\n", reversed);

    uppercase = uppercase_string(str, length);
    printf("The uppercase string is: %s\n", uppercase);

    printf("The string is palindrome: %d\n", check_palindrome(str, length));
}