**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

Факультет **Инфокоммуникационных технологий**

Образовательная программа **Мобильные и сетевые технологии**

Направление подготовки(специальность) **09.03.03 Прикладная информатика**

**ЛАБОРАТОРНАЯ РАБОТА №1.1 и 1.2**

**По дисциплине «Программирование»**

**Тема: Настройка проекта и окружения. Алгоритмы шифрования**

**Выполнил** Говоров П.И.; К3140

**Проверил** Терещенко В.В.

**Дата** 06.10.2024

**Санкт-Петербург 2024**

**Цель работы**

После настройки окружения, установки PyCharm, Git, синхронизации с Github начинаем выполнение задания: создание и тестирования калькулятора, а также создание и тестирования различных алгоритмов шифрования

**Ход работы**

**Создание калькулятора**

Калькулятор будет реализован на двух стэках: с числами, а также операциями и скобками.

На вход подается строка математического примера, например: 

Дальше строка разбивается на массив, который содержит каждый элемент (число, скобка, операция) в отдельной ячейке.

Код:

def get\_split\_arr(data\_st: str):  
 *'''split input string to arr of numbers and operations'''* data\_st = (data\_st.  
 replace('(',' ( ').  
 replace(')',' ) ').  
 replace('\*',' \* ').  
 replace('+',' + ').  
 replace('-',' - ').  
 replace('/',' / ').  
 replace('^',' ^ ').  
 strip())  
 while ' ' in data\_st:  
 data\_st = data\_st.replace(' ',' ')  
  
 example = data\_st.split()  
  
 return example

Дальше был написана функция, которая решает математический пример по следующему алгоритму:

1. Сначала создадим словарь приоритетов математический операций операций
2. Затем будем перебирать элементы списка и выполнять следующие действия
3. Если элемент “(” – открывающая скобка, то просто добавим его в стэк операций
4. Если элемент - число, то просто добавим его в стэк чисел
5. Если элемент – операция:
   1. Если приоритет нашей операции выше последней операции в стэке, или в стэке отсутствуют элементы, или последний элемент в стэке открывающаяся скобка, то просто добавим операцию в соответствующий стэк
   2. Иначе выполнять операции из стэка для последней операции и двум последним числам в стэке с числами по очереди, пока не выполнится какое-либо условие выше, а потом добавить нашу операция в стэк
6. Если элемент закрывающаяся скобка, нужно выполнять операции из стэка для последней операции и двум последним числам в стэке с числами по очереди, пока встретим открывающую скобку
7. После перебора всех элементов, выполним оставшиеся в стэке операции и получим в стэке чисел наш ответ на пример.

Код:

def to\_calc(calc\_example\_str):  
 *'''calc some math example in string format'''* try:  
 num\_stack = op\_stack = []  
 prior = {'+': 1,'-': 1, '\*': 2, '/': 2,'^':3}  
 calc\_example\_arr = get\_split\_arr(calc\_example\_str)  
  
 for i in calc\_example\_arr:  
 if i == '(':  
 op\_stack.append('(')  
 elif (i.isdigit()  
 or i.replace('.','',1).isdigit()  
 or i.replace('-','',1).isdigit()):  
 num\_stack.append(float(i))  
 elif i in '+-\*/^':  
 if (len(op\_stack) == 0 or op\_stack[-1] == '('  
 or prior[i] > prior[op\_stack[-1]]):  
 op\_stack.append(i)  
 elif prior[i] <= prior[op\_stack[-1]]:  
 while (len(op\_stack) > 0 and op\_stack[-1] != '('  
 and prior[i] <= prior[op\_stack[-1]]):  
 second\_arg = num\_stack.pop()  
 first\_arg = num\_stack.pop()  
 operation = op\_stack.pop()  
 num\_stack.append(do\_operation(first\_arg,second\_arg,  
 operation))  
 op\_stack.append(i)  
 elif i == ')':  
 while op\_stack[-1] != '(':  
 second\_arg = num\_stack.pop()  
 first\_arg = num\_stack.pop()  
 operation = op\_stack.pop()  
 num\_stack.append(do\_operation(first\_arg, second\_arg,  
 operation))  
 del\_br = op\_stack.pop()  
  
 while len(op\_stack) > 0:  
 second\_arg = num\_stack.pop()  
 first\_arg = num\_stack.pop()  
 operation = op\_stack.pop()  
 num\_stack.append(do\_operation(first\_arg, second\_arg,  
 operation))  
  
 return num\_stack[0]  
 except ZeroDivisionError:  
 return 'Error: Division by zero'  
 except:  
 return 'Input Error'

Все операции выполняет следующая функция:

def do\_operation(first\_arg: float,second\_arg: float,operation: str):  
 *'''do some operation to first and second arguments'''* match operation:  
 case '+': return round(first\_arg + second\_arg,7)  
 case '-': return round(first\_arg - second\_arg,7)  
 case '\*': return round(first\_arg \* second\_arg,7)  
 case '/': return round(first\_arg / second\_arg,7)  
 case '^': return round(first\_arg \*\* second\_arg,7)

Далее напишем тесты для калькулятора, проверяющие различные случаи, ошибки ввода и т.д.

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

**Создание алгоритмов шифрования**

**Алгоритм Цезаря**

Алгоритм основан на сдвиге букв на определенное значение по алфавиту, последние буквы циклично изменяются на буквы начала алфавита.

Будем проходить по каждому символу строки и если – это буква, то получаем код символа сдвигаем на введенное значение, берем по модулю 26, затем получаем букву обратно, если это иной символ, то просто добавляем

Для расшифровки делаем то же самое, но сдвигаем на введенное значение в обратную сторону.

def encrypt\_caesar(plaintext: str, shift: int = 3) -> str:  
 ciphertext = ""  
 offset\_ascii\_upper = 65  
 offset\_ascii\_lower = 97  
  
 for i in plaintext:  
 if i.isalpha():  
 if i.islower():  
 ciphertext += chr((ord(i) - offset\_ascii\_lower + shift)  
 % 26 + offset\_ascii\_lower)  
 else:  
 ciphertext += chr((ord(i) - offset\_ascii\_upper + shift)  
 % 26 + offset\_ascii\_upper)  
 else:  
 ciphertext += i  
 return ciphertext  
  
  
def decrypt\_caesar(ciphertext: str, shift: int = 3) -> str:  
 plaintext = ""  
 offset\_ascii\_upper = 65  
 offset\_ascii\_lower = 97  
 for i in ciphertext:  
 if i.isalpha():  
 if i.islower():  
 plaintext += chr((ord(i) - offset\_ascii\_lower - shift)  
 % 26 + offset\_ascii\_lower)  
 else:  
 plaintext += chr((ord(i) - offset\_ascii\_upper - shift)  
 % 26 + offset\_ascii\_upper)  
 else:  
 plaintext += i  
 return plaintext

**Алгоритм Виженера**

Шифр похож на алгоритм Цезаря, то за сдвиг выступают слова – где каждая буква обозначает сдвиг для соответствующей ей буквы слова равный индексу буквы в алфавите, если кодовое слова меньше шифруемого, то кодовое слово циклично используется несколько раз

Для расшифровки аналогично шифру Цезаря нужно сдвинуть в другую сторону.

Код:

from string import ascii\_uppercase  
  
def encrypt\_vigenere(plaintext: str, keyword: str) -> str:  
 ciphertext = ""  
 shift\_code\_arr = {let:ind for ind, let in enumerate(ascii\_uppercase)}  
 offset\_ascii\_upper = 65  
 offset\_ascii\_lower = 97  
 keyword\_ind\_current = 0  
  
 for i in plaintext:  
 shift = shift\_code\_arr[keyword[keyword\_ind\_current].upper()]  
 if i.isalpha():  
 if i.islower():  
 ciphertext += chr((ord(i) - offset\_ascii\_lower + shift)  
 % 26 + offset\_ascii\_lower)  
 else:  
 ciphertext += chr((ord(i) - offset\_ascii\_upper + shift)  
 % 26 + offset\_ascii\_upper)  
 else:  
 ciphertext += i  
 keyword\_ind\_current += 1  
 if keyword\_ind\_current == len(keyword):  
 keyword\_ind\_current = 0  
 return ciphertext  
  
  
def decrypt\_vigenere(ciphertext: str, keyword: str) -> str:  
 plaintext = ""  
 shift\_code\_arr = {let: ind for ind, let in enumerate(ascii\_uppercase)}  
 offset\_ascii\_upper = 65  
 offset\_ascii\_lower = 97  
 keyword\_ind\_current = 0  
  
 for i in ciphertext:  
 shift = shift\_code\_arr[keyword[keyword\_ind\_current].upper()]  
 if i.isalpha():  
 if i.islower():  
 plaintext += chr((ord(i) - offset\_ascii\_lower - shift)  
 % 26 + offset\_ascii\_lower)  
 else:  
 plaintext += chr((ord(i) - offset\_ascii\_upper - shift)  
 % 26 + offset\_ascii\_upper)  
 else:  
 plaintext += i  
 keyword\_ind\_current += 1  
 if keyword\_ind\_current == len(keyword):  
 keyword\_ind\_current = 0  
 return plaintext

**Алгоритм RSA**

Для это алгоритма было предложено написать некоторые функции:

1. Проверка числа на простоту. Если у числа не существует делителей от двух до корня этого числа, то оно простое

def is\_prime(n: int) -> bool:  
 return all(n % i for i in range(2, int(n \*\* 0.5) + 1))

1. Нахождение НОД через алгоритм Евклида. Пока остаток деления большего числа на меньшее не равен нулю, повторяем эту операцию для меньшего числа и остатка большего числа на меньшее

def gcd(a: int, b: int) -> int:  
 if b == 0:  
 return a  
 return gcd(b, a % b)

1. “Обратное умножение”. Нужно найти такое число **d** для **e** и **phi**, что выполняется следующее условие **d \* e mod phi = 1**, для этого используется алгоритм Евклида.

def multiplicative\_inverse(e: int, phi: int) -> int:  
 copy\_e = e  
 copy\_phi = phi  
  
 gcd\_hist = [(copy\_e,copy\_phi)]  
 while (copy\_e % copy\_phi != 0):  
 copy\_e, copy\_phi = copy\_phi, copy\_e % copy\_phi  
 gcd\_hist.append((copy\_e,copy\_phi))  
  
 gcd\_hist\_index = -1  
 x\_par = 0  
 y\_par = 1  
 a\_par = gcd\_hist[gcd\_hist\_index][0]  
 b\_par = gcd\_hist[gcd\_hist\_index][1]  
 while e \* x\_par + phi \* y\_par != 1:  
 gcd\_hist\_index -= 1  
 if abs(gcd\_hist\_index) == len(gcd\_hist) + 1:  
 return -1  
 a\_par = gcd\_hist[gcd\_hist\_index][0]  
 b\_par = gcd\_hist[gcd\_hist\_index][1]  
 x\_par, y\_par = y\_par, x\_par - y\_par \* (a\_par // b\_par)  
  
 return x\_par % b\_par

Затем для всех алгоритмов были написаны следующие тесты

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание**Вывод**

В результате проделанной работы я научился писать стэковый калькулятор, узнал о трех способах шифрования и научился их писать, в ходе работы над алгоритмами шифрования я узнал, как создавать функции нахождения простоты числа, НОД-а двух чисел, “обратного умножения”, а также написал тесты для калькулятора и алгоритмов шифрования