**НЕГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ ЧАСТНОЕ** **УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ** **«МОСКОВСКИЙ ФИНАНСОВО-ПРОМЫШЛЕННЫЙ УНИВЕРСИТЕТ** **“СИНЕРГИЯ”»**

|  |  |  |
| --- | --- | --- |
| **Факультет/Институт** |  | Информационных технологий |
|  |  | (наименование факультета/ Института) |
| **Направление/специальность** |  | 09.02.07 Информационные системы и программирование |
| **подготовки:** |  | (код и наименование направления /специальности подготовки) |
| **Форма обучения:** |  | очная |
|  |  | (очная, очно-заочная, заочная) |
|  |  |  |

**Отчет по лабораторной работе № 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **на тему** |  | Разработка тестовых пакетов | | |
|  |  | (наименование темы) | | |
|  |  |  | | |
| **по дисциплине** | | |  | Тестирование информационных систем |
|  | | |  | (наименование дисциплины) |

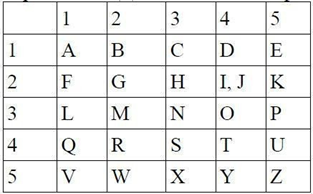
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Обучающийся** |  | Мартынов Василий Викторович |  |  |
|  |  | (ФИО) |  | (подпись) |
| **Группа** |  | ДКИП-311 |  |  |
|  |  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Преподаватель** |  | Авдеенков Владимир Александрович |  |  |
|  |  | (ФИО) |  | (подпись) |

**Москва 2024 г**

**Лабораторная работа №2.** **«Разработка тестовых пакетов»**

В Древней Греции (II в. до н.э.) был известен шифр, называемый "квадрат Полибия". Шифровальная таблица представляла собой квадрат с пятью столбцами и пятью строками, которые нумеровались цифрами от 1 до 5. В каждую клетку такого квадрата записывалась одна буква. В результате каждой букве соответствовала пара чисел, и шифрование сводилось к замене буквы парой чисел. Для латинского алфавита квадрат Полибия имеет вид:



Пользуясь вышеизложенным способом написать программу, которая:

а) зашифрует введенный текст и выведет на экран;

б) считает зашифрованный текст и расшифрует данный текст (*пункт б выполнить по желанию и возможностям*).

**Для удобства шифр записывается в файл, расшифровка (для дополнительного пункта б) также происходит из файла.**

def get\_table() -> tuple:

    table = (

        ("A", "B", "C", "D", "E"),

        ("F", "G", "H", "IJ", "K"),

        ("L", "M", "N", "O", "P"),

        ("Q", "R", "S", "T", "U"),

        ("V", "W", "X", "Y", "Z"))

    return table

def is\_all\_latin\_letters(text) -> bool:

    return all(s.isalpha() and ('A' <= s.upper() <= "Z") for s in text)

def get\_encrypt(symbol: str) -> str:

    table = get\_table()

    if symbol.upper() in ("I", "J"):

        return "24"

    for i, line in enumerate(table, start=1):

        for j, smbl in enumerate(line, start=1):

             if symbol.upper() == smbl:

                return f"{i}{j}"

def to\_encrypt(text: str) -> str:

    crypt\_text = ""

    for symbol in text:

        crypt\_text += get\_encrypt(symbol)

    return crypt\_text

def write\_encrypt(text: str) -> None:

    with open("crypt.txt", "w") as file:

        file.write(to\_encrypt(text))

def read\_encrypt() -> str:

    with open("crypt.txt") as file:

        text = file.read()

        return text

def is\_code(text:str) -> bool:

    if not len(text) == 0:

        return all(s.isdigit() and (1 <= int(s) <= 5) and (len(text)%2 == 0) for s in text)

    else:

        return False

def get\_decode(i: int, j: int) -> str:

    table = get\_table()

    symbol = table[i][j]

    return symbol

def to\_decode() -> str:

    encrypt\_text = read\_encrypt()

    if is\_code(encrypt\_text):

        text = ""

        while len(encrypt\_text) > 0:

            i = int(encrypt\_text[0])-1

            j = int(encrypt\_text[1])-1

            text += get\_decode(i, j)

            encrypt\_text = encrypt\_text[2:]

        return text

    else:

        return "Содержимое файла не корректно"

def main(your\_text: str):

    if not len(your\_text) == 0 :

        if is\_all\_latin\_letters(your\_text):

            print (f"Код {to\_encrypt(your\_text)} записан в файл")

            write\_encrypt(your\_text)

        else:

            print ('Ошибка ввода, символы не подлежат кодированию')

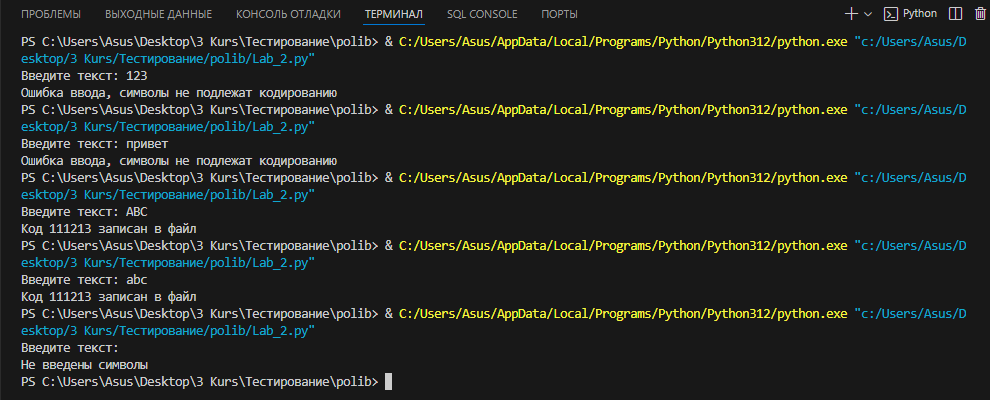
    else:

        print("Не введены символы")

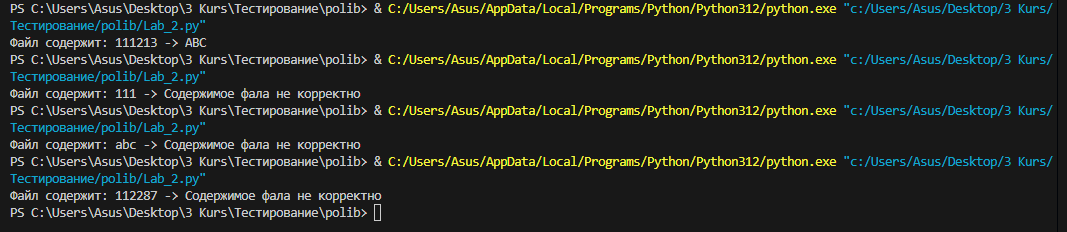
main(input("Введите текст: ")) # для задания 1.а)

#print(f"Файл содержит: {read\_encrypt()} -> {to\_decode()}") # для задания 1.б)

Примеры работы программы по заданию a):



Примеры работы программы по заданию б):



проектировать тесты по принципу «белого ящика» для программы, разработанной в задании № 1

кодирования

def is\_all\_latin\_letters(text) -> bool:

    return all(s.isalpha() and ('A' <= s.upper() <= "Z") for s in text)

для покрытия данного блока кода необходимо проверить все точки условий:

является ли каждый символ строки буквой, принадлежит ли каждая буква латинскому алфавиту (по условию).

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Ожидаемый результат | Фактический результат | Результат |
| "abcdefghijklmnopqrstuvwxyz" | True | True | успех |
| "ABCDEFGHIJKLMNOPQRSTUVWXYZ" | True | True | успех |
| “123” | False | False | успех |
| “абв” | False | False | успех |
| “АБВ” | False | False | успех |
| “ , ” | False | False | успех |
| “Hello World” | False | False | успех |

Тестовый сценарий Pytest

data\_is\_all\_latin = (

    ("abcdefghijklmnopqrstuvwxyz", True),

    ("ABCDEFGHIJKLMNOPQRSTUVWXYZ", True),

    ("абв", False),

    ("АБВ", False),

    ("123", False),

    (" , ", False),

    ("Hello Word",False),

)

@pytest.mark.parametrize('symbol, result', data\_is\_all\_latin)

def test\_is\_all\_latin\_letter(symbol, result):

    assert is\_all\_latin\_letters(symbol) == result

def get\_encrypt(symbol: str) -> str:

    table = get\_table()

    if symbol.upper() in ("I", "J"):

        return "24"

    for i, line in enumerate(table, start=1):

        for j, smbl in enumerate(line, start=1):

             if symbol.upper() == smbl:

                return f"{i}{j}"

полное покрытие данного блока возможно достичь за счет перебора всех возможных аргументов и соответствующих им значений

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Ожидаемый результат | Фактический результат | Результат |
| Символ A | 11 | 11 | успех |
| Символ B | 12 | 12 | успех |
| Символ C | 13 | 13 | успех |
| Символ D | 14 | 14 | успех |
| Символ E | 15 | 15 | успех |
| Символ F | 21 | 21 | успех |
| Символ G | 22 | 22 | успех |
| Символ H | 23 | 23 | успех |
| Символ I | 24 | 24 | успех |
| Символ J | 24 | 24 | успех |
| Символ K | 25 | 25 | успех |
| Символ L | 31 | 31 | успех |
| Символ M | 32 | 32 | успех |
| Символ N | 33 | 33 | успех |
| Символ O | 34 | 34 | успех |
| Символ P | 35 | 35 | успех |
| Символ Q | 41 | 41 | успех |
| Символ R | 42 | 42 | успех |
| Символ S | 43 | 43 | успех |
| Символ T | 44 | 44 | успех |
| Символ U | 45 | 45 | успех |
| Символ V | 51 | 51 | успех |
| Символ W | 52 | 52 | успех |
| Символ X | 53 | 53 | успех |
| Символ Y | 54 | 54 | успех |
| Символ Z | 55 | 55 | успех |

Тестовый сценарий Pytest

data\_get\_encrypt = (

    ("A", "11"),

    ("B", "12"),

    ("C", "13"),

    ("D", "14"),

    ("E", "15"),

    ("F", "21"),

    ("G", "22"),

    ("H", "23"),

    ("I", "24"),

    ("J", "24"),

    ("K", "25"),

    ("L", "31"),

    ("M", "32"),

    ("N", "33"),

    ("O", "34"),

    ("P", "35"),

    ("Q", "41"),

    ("R", "42"),

    ("S", "43"),

    ("T", "44"),

    ("U", "45"),

    ("V", "51"),

    ("W", "52"),

    ("X", "53"),

    ("Y", "54"),

    ("Z", "55"),

)

@pytest.mark.parametrize('symbol, result', data\_get\_encrypt)

def test\_data\_get\_encrypt(symbol, result):

    assert get\_encrypt(symbol) == result

def to\_encrypt(text: str) -> str:

    crypt\_text = ""

    for symbol in text:

        crypt\_text += get\_encrypt(symbol)

    return crypt\_text

Для проверки достаточно любой комбинации символов в верхнем и в нижнем регистре, например все возможные символы в одной строке (кодирование каждого символа проверено ранее)

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Ожидаемый результат | Фактический результат | Результат |
| "abcdefghijklmnopqrstuvwxyz" | "1112131415212223242425313233343541424344455152535455" | "1112131415212223242425313233343541424344455152535455" | успех |
| "ABCDEFGHIJKLMNOPQRSTUVWXYZ" | "1112131415212223242425313233343541424344455152535455" | "1112131415212223242425313233343541424344455152535455" | успех |

Сценарий Pytest

data\_to\_encrypt = (

    ("abcdefghijklmnopqrstuvwxyz", "1112131415212223242425313233343541424344455152535455"),

    ("ABCDEFGHIJKLMNOPQRSTUVWXYZ", "1112131415212223242425313233343541424344455152535455")

)

@pytest.mark.parametrize('text, result', data\_to\_encrypt)

def test\_to\_encrypt(text, result):

    assert to\_encrypt(text) == result

def write\_encrypt(text: str) -> None:

    with open("crypt.txt", "w") as file:

        file.write(to\_encrypt(text))

def read\_encrypt() -> str:

    with open("crypt.txt") as file:

        text = file.read()

        return text

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Ожидаемый результат | Фактический результат | Результат |
| Запись\Чтение | Запись в файл | Запись в файл | успех |
| Запись\Чтение | Чтение из… | Чтение из… | успех |

Сценарий Pytest

data\_read\_write = (

    ("abc", "111213"),

    ("xyz", "535455")

)

@pytest.mark.parametrize('text, result', data\_read\_write)

def test\_read\_write(text, result):

    write\_encrypt(text)

    assert read\_encrypt() == result

5)Вывод данных в терминале

def main(your\_text: str):

    if not len(your\_text) == 0 :

        if is\_all\_latin\_letters(your\_text):

            print (f"Код {to\_encrypt(your\_text)} записан в файл")

            write\_encrypt(your\_text)

        else:

            print ('Ошибка ввода, символы не подлежат кодированию')

    else:

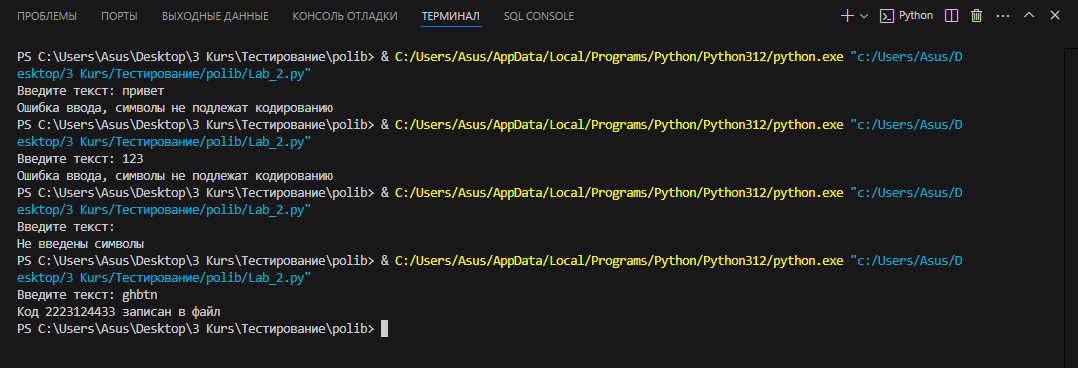
        print("Не введены символы")

для проверки необходимо покрытие всех условий:

текст нулевой длинны

все буквы латинские (функция проверена в п.1)

не все буквы латинские



def is\_code(text:str) -> bool:

    if not len(text) == 0:

        return all(s.isdigit() and (1 <= int(s) <= 5) and (len(text)%2 == 0) for s in text)

    else:

        return False

для покрытия данного блока кода необходимо проверить все точки условий тк не выполнение хотя бы одного меняет значение функции:

является ли длинна строки в файле ненулевой,

четное ли количество символов

являются ли все символы цифрами

и лежат ли цифры в промежутке от 1 до 5

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Ожидаемый результат | Фактический результат | Результат |
| “1” | False | False | успех |
| “1277” | False | False | успех |
| “” | False | False | успех |
| “11” | True | True | успех |
| “abc” | False | False | успех |

Сценарий Pytest

ata\_is\_code = (

    ("1", False),

    ("1277", False),

    ("abc", False),

    ("", False),

    ("11", True)

)

@pytest.mark.parametrize('text, result', data\_is\_code)

def test\_is\_code(text, result):

    assert is\_code(text) == result

7) Декодирование

def get\_decode(i: int, j: int) -> str:

    table = get\_table()

    symbol = table[i][j]

    return symbol

def to\_decode() -> str:

    table = get\_table()

    symbol = table[i][j]

    return symbol

encrypt\_text = read\_encrypt()

    if is\_code(encrypt\_text):

        text = ""

        while len(encrypt\_text) > 0:

            i = int(encrypt\_text[0])-1

            j = int(encrypt\_text[1])-1

            text += get\_decode(i, j)

            encrypt\_text = encrypt\_text[2:]

        return text

    else:

        return "Содержимое файла не корректно"

полное покрытие данного блока возможно достичь за счет перебора всех возможных аргументов и соответствующих им значений (для символов)

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | Ожидаемый результат | Фактический результат | Результат |
| 11 | A | A | успех |
| 12 | B | B | успех |
| 13 | C | C | успех |
| 14 | D | D | успех |
| 15 | E | E | успех |
| 21 | F | F | успех |
| 22 | G | G | успех |
| 23 | H | H | успех |
| 24 | I | I, J | Не успех |
| 24 | J | I, J | Не успех |
| 25 | K | K | успех |
| 31 | L | L | успех |
| 32 | M | M | успех |
| 33 | N | N | успех |
| 34 | O | O | успех |
| 35 | P | P | успех |
| 41 | Q | Q | успех |
| 42 | R | R | успех |
| 43 | S | S | успех |
| 44 | T | T | успех |
| 45 | U | U | успех |
| 51 | V | V | успех |
| 52 | W | W | успех |
| 53 | X | X | успех |
| 54 | Y | Y | успех |
| 55 | Z | Z | успех |

Сценарий Pytest

data\_get\_decode = (

    ("A", "A"),

    ("B", "B"),

    ("C", "C"),

    ("D", "D"),

    ("E", "E"),

    ("F", "F"),

    ("G", "G"),

    ("H", "H"),

    ("I", "I"),

    ("J", "J"),

    ("K", "K"),

    ("L", "L"),

    ("M", "M"),

    ("N", "N"),

    ("O", "O"),

    ("P", "P"),

    ("Q", "Q"),

    ("R", "R"),

    ("S", "S"),

    ("T", "T"),

    ("U", "U"),

    ("V", "V"),

    ("W", "W"),

    ("X", "X"),

    ("Y", "Y"),

    ("Z", "Z"),

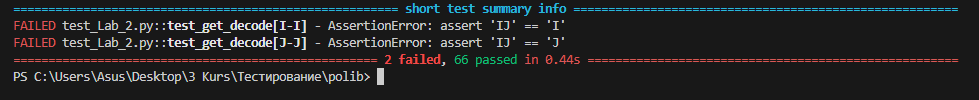
)

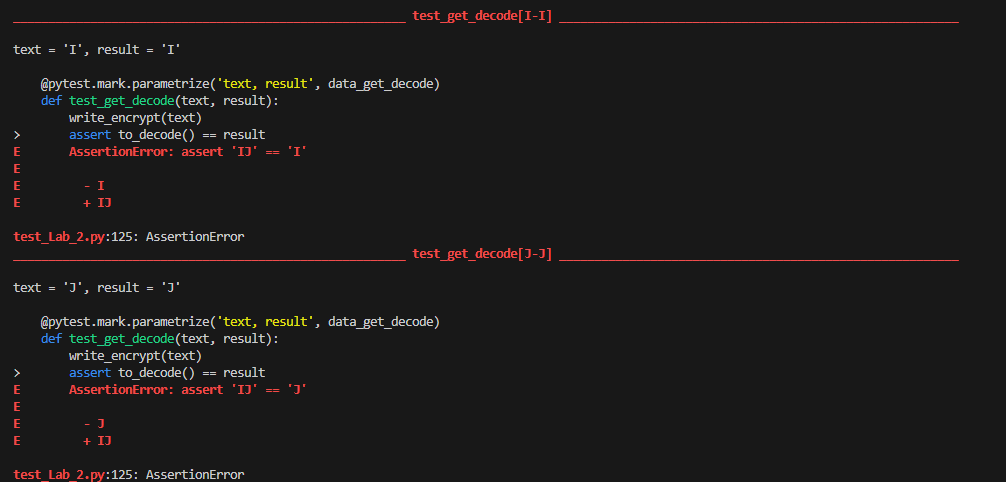
@pytest.mark.parametrize('text, result', data\_get\_decode)

def test\_get\_decode(text, result):

    write\_encrypt(text)

    assert to\_decode() == result

**Итог автоматического тестирования на тестовом пакете**

**Лог ошибок**

В ходе лабораторной работы написана программа, реализующая алгоритм шифрования «Квадрат Полибия», был разработан пакет тестовых сценариев, который полностью охватывает код программы. В ходе тестирования удалось обнаружить недостаток алгоритма при дешифровании, это связано с тем, что двум буквам (I, J) соответствует одна уникальная пара индексов таблицы (2, 4), и при обратном процессе - шифру 24 соответствуют два значения (I и J). Все остальные тесты пройдены успешно.