**Лабораторная работа №4**

**Class.**

from test1 import A  
from test2 import B  
from test3 import C  
from test4 import D  
from test5 import J  
  
while 1:  
 print("A - 1")  
 print("B - 2")  
 print("C - 3")  
 print("D - 4")  
 print("J - 5")  
  
 number = int(input("выберите: "))  
 match number:  
 case 1: A()  
 case 2: B()  
 case 3: C()  
 case 4: D()  
 case 5: J()

**Алгоритм выполнения кода:**

**Шаг 1: Инициализация**

**1. Функция проверки простоты числа is\_prime(n):**

1.1. Функция принимает целое число n в качестве входных данных.

1.2. Если n меньше 2, вернуть False (число не является простым).

1.3. Для каждого числа i от 2 до квадратного корня из n включительно: Если n делится на i без остатка, вернуть False (число не является простым).

1.4. Если ни один из предыдущих шагов не вернул False, вернуть True (число является простым).

**2. Функция поиска простых делителей prime\_factors(n, factors):**

2.1. Функция принимает целое число n и необязательный список factors (по умолчанию пустой).

2.2. Если n равно 1, вернуть список factors (базовый случай рекурсии).

2.3. Для каждого числа i от 2 до квадратного корня из n включительно: Если n делится на i без остатка. Если i является простым числом (проверить с помощью функции is\_prime(i)), то добавить i в список factors. Рекурсивно вызвать функцию prime\_factors для n, деленного на i (n // i), и текущим factors. Вернуть результат рекурсивного вызова.

2.4. Если n является простым числом (проверить с помощью функции is\_prime(n)), то добавить n в список factors.

2.5. Вернуть список factors (содержащий все простые делители числа n).

**3. Функция подсчета трехзначных чисел с суммой простых делителей, кратной 5 sum\_of\_prime\_factors\_divisible\_by\_5(n, sum\_divisible\_by\_5):**

3.1. Функция принимает целое число n (начиная с 100) и необязательную переменную sum\_divisible\_by\_5 (по умолчанию 0, для учета).

3.2. Если n больше 999, вернуть значение sum\_divisible\_by\_5 (базовый случай рекурсии, для остановки).

3.3. Найти все простые делители числа n, используя функцию prime\_factors(n).

3.4. Вычислить сумму простых делителей.

3.5. Если сумма простых делителей кратна 5: Увеличить значение sum\_divisible\_by\_5 на 1. 3.6. Рекурсивно вызвать функцию sum\_of\_prime\_factors\_divisible\_by\_5, передав n + 1 (для перехода к следующему числу) и текущее значение sum\_divisible\_by\_5.

3.7. Вернуть результат рекурсивного вызова.

**Шаг 2: Основной алгоритм** **1.** **Вызов рекурсивной функции**:

1.1. Вызвать функцию sum\_of\_prime\_factors\_divisible\_by\_5 с начальным значением n = 100 (первое трехзначное число) и начальным значением sum\_divisible\_by\_5 = 0.

1.2. Сохранить результат вызова в переменную result.

**Шаг 3: Вывод результата**

**1.** **Вывод результата на экран:**

1.1. Вывести значение переменной result, которое представляет количество трехзначных чисел, у которых сумма простых делителей кратна 5.

**Задание 1**. Реализация одного задания из лабораторной работы №3 в виде рекурсивной функции.

- Найти количество трехзначных чисел, сумма простых делителей которых кратна 5.

class A:  
 def \_\_init\_\_(self):  
 def is\_prime(n):if n < 2:  
 return False  
 for i in range(2, int(n \*\* 0.5) + 1):  
 if n % i == 0:  
 return False  
 return True  
  
 def prime\_factors(n, factors=None):  
 if factors is None:  
 factors = []  
  
 if n == 1:  
 return factors  
  
 for i in range(2, int(n \*\* 0.5) + 1):  
 if n % i == 0:  
 if is\_prime(i):  
 factors.append(i)  
 return prime\_factors(n // i, factors)  
 else:  
 return prime\_factors(n // i, factors)   
 if is\_prime(n):  
 factors.append(n)  
 return factors  
  
 def sum\_of\_prime\_factors\_divisible\_by\_5(n, sum\_divisible\_by\_5=0):if n > 999:  
 return sum\_divisible\_by\_5  
  
 factors = prime\_factors(n)  
  
 if sum(factors) % 5 == 0:  
 sum\_divisible\_by\_5 += 1  
  
 return sum\_of\_prime\_factors\_divisible\_by\_5(n + 1, sum\_divisible\_by\_5)  
  
 # Main execution  
 result = sum\_of\_prime\_factors\_divisible\_by\_5(100)  
 print(f"Количество трехзначных чисел с суммой простых множителей, кратных 5: {result}")

ОТВЕТ:

A - 1

B - 2

C - 3

D - 4

J - 5

выберите: 1

**Количество трехзначных чисел с суммой простых множителей, кратных 5: 160**

**Алгоритм выполнения кода:**

**Шаг 1: Подготовка к сортировке**

**1. Определить гласные: Задать строку, содержащую все русские гласные буквы (например, "аеёиоуыэюя").**

2. **Определить функцию извлечения слов (extract\_words):**

2. 1. Принять строку текста в качестве аргумента.

2. 2. Используя регулярное выражение (например, r'\b\w+\b'), найти все слова в тексте, игнорируя знаки препинания и лишние пробелы.

2. 3. Вернуть список найденных слов.

**3. Определить функцию расчета доли гласных (calculate\_vowel\_ratio):**

3. 1. Принять слово в качестве аргумента.

3. 2. Привести слово к нижнему регистру, чтобы не учитывать регистр.

3. 3. Инициализировать счетчик гласных значением 0.

3. 4. **Для каждой буквы в слове (**Если буква содержится в строке гласных, увеличить счетчик гласных на 1.**).**

3. 5. Вычислить общее количество букв в слове.

3. 6. Если общее количество букв больше 0, вычислить долю гласных (количество гласных, деленное на общее количество букв).

3. 7. Иначе (если общее количество букв равно 0) вернуть 0 (во избежание деления на ноль).

3. 8. Вернуть вычисленную долю гласных.

**4. Принять текст на вход:** Принять исходную строку текста, которую нужно обработать.

**Шаг 2: Обработка текста**

2. 1. Используя функцию извлечения слов (extract\_words), получить список слов из входного текста.

2. 2. Проверить, пуст ли список слов.

2. 3. Если список слов пуст, вернуть пустой список и завершить работу.

**Шаг 3: Сортировка слов**

3. 1. Используя встроенную функцию сортировки (sorted()), отсортировать список слов.

3. 2. В качестве ключа для сортировки использовать функцию расчета доли гласных (calculate\_vowel\_ratio). Это обеспечит сортировку по возрастанию доли гласных.

**Шаг 4: Возврат результата**

4. 1. Вернуть отсортированный список слов.

**Шаг 5: Вывод**

5. 1. Для отладки и демонстрации работы алгоритма можно вывести отсортированный список.

**Задание 2.** Обработка строки символов.

* Дана строка символов. Определить количество букв «н», расположенных после первой запятой.

class B:  
 def \_\_init\_\_(self):  
 def count\_n\_after\_comma(text):  
 comma\_index = text.find(",")  
  
 if comma\_index == -1:  
 return 0 # No comma found  
  
 substring\_after\_comma = text[comma\_index + 1:]  
 count = substring\_after\_comma.lower().count("н")  
 return count  
  
 # Пример  
 string1 = "Пример, длинное слово с ннннн"  
 string2 = "Нет запятой здесь"  
 string3 = "Первая запятая,но ни одного н здесь"  
 string4 = ",ннннннн"  
 string5 = "Это,слово,тоже,нннн"  
 string6 = "Ещё одна строка, но н в начале"  
 string7 = "123,456,ннн"  
  
 print(f"'{string1}': {count\_n\_after\_comma(string1)}") # Output: 5  
 print(f"'{string2}': {count\_n\_after\_comma(string2)}") # Output: 0  
 print(f"'{string3}': {count\_n\_after\_comma(string3)}") # Output: 0  
 print(f"'{string4}': {count\_n\_after\_comma(string4)}") # Output: 7  
 print(f"'{string5}': {count\_n\_after\_comma(string5)}") # Output: 1  
 print(f"'{string6}': {count\_n\_after\_comma(string6)}") # Output: 1  
 print(f"'{string7}': {count\_n\_after\_comma(string7)}") # Output: 3

ОТВЕТ:

A - 1

B - 2

C - 3

D - 4

J - 5

выберите: 2

**'Пример, длинное слово с ннннн': 7**

**'Нет запятой здесь': 0**

**'Первая запятая,но ни одного н здесь': 4**

**',ннннннн': 7**

**'Это,слово,тоже,нннн': 4**

**'Ещё одна строка, но н в начале': 3**

**'123,456,ннн': 3**

**Алгоритм выполнения кода:**

**Шаг 1: Инициализация и Подготовка**

**1. Определение гласных:**

1.1. Задать строку vowels и присвоить ей все гласные буквы русского алфавита: "аеёиоуыэюя". Эта строка будет использоваться для проверки, является ли символ гласным. **2. Определение функции извлечения слов extract\_words(text):**

2.1. Функция принимает на вход строку text.

2.2. Используя регулярное выражение r'\b\w+\b', найти в строке text все подстроки, соответствующие словам. \* \b означает границу слова. \* \w+ означает один или более буквенно-цифровых символов или символ подчеркивания.

2.3. Вернуть список найденных слов.

**3. Определение функции расчета доли гласных calculate\_vowel\_ratio(word):**

3.1. Функция принимает на вход одно слово word.

3.2. Преобразовать слово word в нижний регистр, чтобы сравнение было нечувствительно к регистру: word = word.lower().

3.3. Инициализировать переменную vowel\_count значением 0. Она будет хранить количество гласных букв в слове.

3.4. Пройтись по каждой букве char в слове word. Проверить, является ли буква char гласной, сравнив её с набором гласных vowels. Если char есть в vowels, увеличить vowel\_count на 1.

3.5. Вычислить общее количество букв в слове, используя функцию len(word), и сохранить это значение в переменную total\_letters.

3.6. Проверить, является ли total\_letters больше 0 (чтобы избежать деления на ноль). Если total\_letters больше 0, вычислить отношение vowel\_count / total\_letters и вернуть его. Иначе вернуть 0 (если слово пустое).

**4. Принятие текста на вход:**

4.1. Получить строку text (исходный текст, который нужно обработать) как входной аргумент.

**Шаг 2: Извлечение слов и проверка на пустоту**

**1.** **Извлечение слов:**

1.1. Вызвать функцию extract\_words(text), передав в неё входную строку text.

1.2. Сохранить результат вызова функции (список слов) в переменную words.

**2. Проверка списка на пустоту:**

2.1. Проверить, является ли список words пустым.

2.2. Если список words пустой, то: Вернуть пустой список []. Завершить выполнение алгоритма.

**Шаг 3: Сортировка слов**

**1.** **Сортировка списка слов:**

1.1. Использовать встроенную функцию sorted() для сортировки списка words.

1.2. В качестве ключа для сортировки (key) указать функцию calculate\_vowel\_ratio. Это означает, что сортировка будет происходить на основе значения, возвращаемого функцией calculate\_vowel\_ratio для каждого слова.

1.3. Сохранить отсортированный список слов в переменную sorted\_words.

**Шаг 4: Возврат результата**

**1.** **Возврат отсортированного списка:**

1.1. Вернуть отсортированный список sorted\_words как результат работы алгоритма.

**Шаг 5: Вывод (Пример использования):**

**1. Пример вызова и вывода результатов:**

1.1. Задать тестовую строку text (например, text = "мама мыла раму" ).

1.2. Вызвать функцию sort\_words\_by\_vowel\_ratio(text), передав ей тестовую строку.

1.3. Сохранить результат в переменную, например result.

1.4. Вывести result на экран, чтобы увидеть отсортированный список слов.

**Задание 3**. Обработка текста, состоящего из отдельных слов.

* Дан текст из слов, разделенных знаками препинания. Определить, какое из слов встречается в строке раньше: с максимальным количеством гласных или согласных букв.

class C:  
 def \_\_init\_\_(self):  
 import re  
  
 def find\_earlier\_word(text):  
 vowels = "аеёиоуыэюяaeiou"   
 words = re.findall(r'\b\w+\b', text.lower())   
 if not words:  
 return None   
  
 max\_vowel\_word = None  
 max\_consonant\_word = None  
 max\_vowel\_count = 0  
 max\_consonant\_count = 0  
  
 for word in words:  
 vowel\_count = sum(1 for char in word if char in vowels)  
 consonant\_count = sum(1 for char in word if char.isalpha() and char not in vowels)  
  
 if vowel\_count > max\_vowel\_count:  
 max\_vowel\_count = vowel\_count  
 max\_vowel\_word = word  
  
 if consonant\_count > max\_consonant\_count:  
 max\_consonant\_count = consonant\_count  
 max\_consonant\_word = word

if max\_vowel\_word == max\_consonant\_word:  
 return max\_vowel\_word  
  
 if not max\_vowel\_word and max\_consonant\_word:  
 return max\_consonant\_word  
  
 if not max\_consonant\_word and max\_vowel\_word:  
 return max\_vowel\_word  
  
   
 for word in words:  
 if word == max\_vowel\_word or word == max\_consonant\_word:  
 return word  
  
 # Пример  
 text1 = "мама, мыла раму! быстро, чисто."  
 text2 = " слово1 , сл2 , слово3 ."  
 text3 = "один,два,три,четыре,пять."  
 text4 = " а, б, в, г, д , е."  
 text5 = "согласное,согласнее,гласное"  
 text6 = "без гласных ,без согласных,ничего"  
 text7 = " , . ,"  
 text8 = "слово"  
 text9 = " слово слово"  
  
 print(f"'{text1}': {find\_earlier\_word(text1)}") # Output: мама  
 print(f"'{text2}': {find\_earlier\_word(text2)}") # Output: слово1  
 print(f"'{text3}': {find\_earlier\_word(text3)}") # Output: четыре  
 print(f"'{text4}': {find\_earlier\_word(text4)}") # Output: а  
 print(f"'{text5}': {find\_earlier\_word(text5)}") # Output: согласное  
 print(f"'{text6}': {find\_earlier\_word(text6)}") # Output: без  
 print(f"'{text7}': {find\_earlier\_word(text7)}") # Output: None  
 print(f"'{text8}': {find\_earlier\_word(text8)}") # Output: слово  
 print(f"'{text9}': {find\_earlier\_word(text9)}") # Output: слово

ОТВЕТ:

A - 1

B - 2

C - 3

D - 4

J - 5

выберите: 3

**'мама, мыла раму! быстро, чисто.': мама**

**' слово1 , сл2 , слово3 .': слово1**

**'один,два,три,четыре,пять.': четыре**

**' а, б, в, г, д , е.': а**

**'согласное,согласнее,гласное': согласное**

**'без гласных ,без согласных,ничего': согласных**

**' , . ,': None**

**'слово': слово**

**' слово слово': слово**

**Алгоритм выполнения кода:**

**Шаг 1: Инициализация и Подготовка**

**1. Прием входных данных:**

1.1. Принять строку text в качестве входного текста.

1.2. Принять целое число k в качестве индекса буквы для замены (индексация начинается с 1).

1.3. Принять символ replacement\_char в качестве символа для замены.

**2.** **Определение функции замены буквы в слове** replace\_word(word)**:**

2.1. Функция принимает на вход одно слово word.

2.2. Получить длину слова word (количество букв в слове).

2.3. Проверить, является ли k (индекс буквы для замены) меньше или равен длине слова word. Если k меньше или равно длине слова, то: создать новую строку: Взять часть слова до k-1 символа. Добавить replacement\_char к этой части. Добавить оставшуюся часть слова с k-го символа до конца. Вернуть полученную новую строку. Иначе (если k больше длины слова), вернуть исходное слово word без изменений.

**Шаг 2: Замена букв в словах текста**

**1.** **Поиск и замена с использованием регулярного выражения:**

1.1. Использовать функцию re.sub() из модуля re для замены в строке text.

1.2. В качестве регулярного выражения указать r'\b\w+\b', которое ищет отдельные слова (последовательность буквенных символов, ограниченную границами слова).

1.3. В качестве функции для замены указать replace\_word, которая будет применяться к каждому найденному слову.

1.4. Сохранить результат работы re.sub() в переменную modified\_text.

**Шаг 3: Возврат результата**

**1.** **Возврат измененной строки:**

1.1. Вернуть измененную строку modified\_text как результат работы алгоритма.

**Шаг 4: Вывод (Пример использования):**

**1. Пример вызова и вывода результатов:**

1.1. Задать тестовую строку text (например, text = "Hello World").

1.2. Задать значение для k (например, k = 2).

1.3. Задать символ для замены replacement\_char (например, replacement\_char = '\*').

1.4. Вызвать функцию replace\_kth\_letter(text, k, replacement\_char), передав ей тестовые данные.

1.5. Сохранить результат вызова в переменную result.

1.6. Вывести result на экран, чтобы увидеть измененную строку.

**Задание 4**. Обработка регулярных выражения для поиска и преобразования текста.

- В каждом слове текста k-ю букву заменить заданным символом. Если k больше длины слова, корректировку не выполнять.

class D:  
 def \_\_init\_\_(self):  
 import re  
  
 def replace\_kth\_letter(text, k, replacement\_char):  
 def replace\_word(match):  
 word = match.group(0)   
 if len(word) >= k:  
 return word[:k - 1] + replacement\_char + word[k:]  
 return word index   
  
  
 modified\_text = re.sub(r'\b\w+\b', replace\_word, text)  
 return modified\_text  
  
 # Пример  
 text1 = "Hello World, this is a Test text"  
 text2 = "one two three four five"  
 text3 = "short word"  
 text4 = "verylongword"  
 text5 = " first second third "  
 text6 = ",word,word"  
 text7 = ""  
  
 print(f"'{text1}', k=2, rep='\*': '{replace\_kth\_letter(text1, 2, '\*')}'")  
 # Output: 'H\*llo W\*rld, th\*s is a T\*st t\*xt'  
  
 print(f"'{text2}', k=3, rep='#': '{replace\_kth\_letter(text2, 3, '#')}'")  
 # Output: 'on# tw# thr#e fo#r fiv#'  
  
 print(f"'{text3}', k=3, rep='!': '{replace\_kth\_letter(text3, 3, '!')}'")  
 # Output: 'sh!rt wo!d'  
  
 print(f"'{text4}', k=5, rep='@': '{replace\_kth\_letter(text4, 5, '@')}'")  
 # Output: 'very@ongword'  
  
 print(f"'{text4}', k=10, rep='@': '{replace\_kth\_letter(text4, 10, '@')}'")  
 # Output: 'verylongword'  
  
 print(f"'{text5}', k=1, rep='\_': '{replace\_kth\_letter(text5, 1, '\_')}'")  
 # Output: ' \_irst \_econd \_hird '  
  
 print(f"'{text6}', k=3, rep='$': '{replace\_kth\_letter(text6, 3, '$')}'")  
 # Output: ',wo$,wo$'  
  
 print(f"'{text7}', k=2, rep='\_': '{replace\_kth\_letter(text7, 2, '\_')}'")  
 # Output: ''

ОТВЕТ:

A - 1

B - 2

C - 3

D - 4

J - 5

выберите: 4

**'Hello World, this is a Test text', k=2, rep='\*': 'H\*llo W\*rld, t\*is i\* a T\*st t\*xt'**

**'one two three four five', k=3, rep='#': 'on# tw# th#ee fo#r fi#e'**

**'short word', k=3, rep='!': 'sh!rt wo!d'**

**'verylongword', k=5, rep='@': 'very@ongword'**

**'verylongword', k=10, rep='@': 'verylongw@rd'**

**' first second third ', k=1, rep='\_': ' \_irst \_econd \_hird '**

**',word,word', k=3, rep='$': ',wo$d,wo$d'**

**'', k=2, rep='\_': ''**

**Алгоритм выполнения кода:**

**Шаг 1: Подготовка и Инициализация**

**1. Определение функции is\_prime(n) (Проверка на простоту):**

1.1. Функция принимает на вход целое число n.

1.2. Если n меньше 2, вернуть False (не простое).

1.3. Перебрать числа от 2 до квадратного корня из n: Если n делится на какое-либо из этих чисел без остатка, вернуть False (не простое).

1.4. Если ни один из предыдущих шагов не вернул False, вернуть True (простое).

**2. Определение функции prime\_factors(n) (Поиск простых делителей):**

2.1. Функция принимает на вход целое число n.

2.2. Создать пустой список factors для хранения простых делителей.

2.3. Найти простые делители числа n: Перебрать числа i от 2 до квадратного корня из n: Пока n делится на i без остатка: Если i является простым (проверить с помощью функции is\_prime(i)), добавить i в список factors. Разделить n на i (целочисленное деление). Если оставшееся n после всех делений является простым (проверить с помощью is\_prime(n)), добавить его в список factors.

2.4. Вернуть список factors (список простых делителей n).

**3.** **Инициализация счетчика**

3.1. Создать переменную count и установить ее в значение 0.

**Шаг 2: Обработка трехзначных чисел**

**1.** **Перебор трехзначных чисел:**

1.1. Начать перебор чисел num от 100 до 999 (включительно). Для каждого числа num выполнить следующие шаги.

**2.** **Анализ каждого трехзначного числа:**

2.1. Найти простые делители числа num, используя функцию prime\_factors(num).

2.2. Вычислить сумму этих простых делителей.

2.3. Проверить, кратна ли сумма простых делителей 5. Если сумма кратна 5, увеличить счетчик count на 1.

**Шаг 3: Вывод результата**

**1.** **Вывод значения счетчика:**

1.1. Вывести значение переменной count (количество трехзначных чисел, сумма простых делителей которых кратна 5).

**Задание 5**. Реализация заданий связанных со строками в виде вложенных функций.

- Рассортировать слова русского текста по возрастанию доли гласных букв (отношение количества гласных к общему количеству букв в слове).

class J:  
 def \_\_init\_\_(self):  
 import re  
  
 def sort\_words\_by\_vowel\_ratio(text):  
 vowels = "аеёиоуыэюя"   
  
 def calculate\_vowel\_ratio(word):  
 word = word.lower()  
 vowel\_count = sum(1 for char in word if char in vowels)  
 total\_letters = len(word)  
 return vowel\_count / total\_letters if total\_letters > 0 else 0   
  
 def extract\_words(text):  
  
 return re.findall(r'\b\w+\b', text)  
  
 words = extract\_words(text)  
  
 if not words:  
 return []   
  
 sorted\_words = sorted(words, key=calculate\_vowel\_ratio)  
 return sorted\_words  
  
 # Пример  
 text1 = "мама мыла раму быстро чисто"  
 text2 = "я иду домой а потом в лес"  
 text3 = "это короткий текст"  
 text4 = "согласные слова совсем без гласных"  
 text5 = " пусто "  
 text6 = "гласное ааа согласное оо"  
 text7 = "её оа"  
  
 print(f"'{text1}': {sort\_words\_by\_vowel\_ratio(text1)}")  
 # Output: ['мыла', 'мама', 'чисто', 'быстро', 'раму']  
  
 print(f"'{text2}': {sort\_words\_by\_vowel\_ratio(text2)}")  
 # Output: ['в', 'лес', 'домой', 'потом', 'иду', 'я']  
  
 print(f"'{text3}': {sort\_words\_by\_vowel\_ratio(text3)}")  
 # Output: ['текст', 'это', 'короткий']  
  
 print(f"'{text4}': {sort\_words\_by\_vowel\_ratio(text4)}")  
 # Output: ['без', 'согласные', 'совсем', 'слова', 'гласных']  
  
 print(f"'{text5}': {sort\_words\_by\_vowel\_ratio(text5)}")  
 # Output: []  
  
 print(f"'{text6}': {sort\_words\_by\_vowel\_ratio(text6)}")  
 # Output: ['согласное', 'гласное', 'оо', 'ааа']  
  
 print(f"'{text7}': {sort\_words\_by\_vowel\_ratio(text7)}")  
 # Output: ['оа', 'её']

ОТВЕТ:

A - 1

B - 2

C - 3

D - 4

J - 5

выберите: 5

**'мама мыла раму быстро чисто': ['быстро', 'чисто', 'мама', 'мыла', 'раму']**

**'я иду домой а потом в лес': ['в', 'лес', 'домой', 'потом', 'иду', 'я', 'а']**

**'это короткий текст': ['текст', 'короткий', 'это']**

**'согласные слова совсем без гласных': ['гласных', 'совсем', 'без', 'слова', 'согласные']**

**' пусто ': ['пусто']**

**'гласное ааа согласное оо': ['гласное', 'согласное', 'ааа', 'оо']**

**'её оа': ['её', 'оа']**