**Лабораторная работа №3**

**Class.**

from test1 import A  
from test3 import B  
from test4 import C  
  
while 1:  
 print("1. A")  
 print("3. B")  
 print("4. C")  
  
 number = int(input("выберите: "))  
 match number:  
 case 1: A()  
 case 3: B()  
 case 4: C()

**Словесный алгоритм для кода:**  
  
1. **Функция prostik(n):** Проверяет, является ли число n простым.  
- Шаг 1: Если n меньше или равно 1, возвращает False (число 1 не является простым).  
- Шаг 2:  Перебирает все числа i от 2 до квадратного корня из n включительно.  
 Шаг 2.1: Для каждого i проверяет, делится ли n на i без остатка (n % i == 0).  
 Шаг 2.2: Если n делится на i, значит n не является простым, возвращает False.  
- Шаг 3: Все проверки пройдены, то n является простым, возвращает True.  
2. **Функция summa\_prost\_delitel(n):** Вычисляет сумму простых делителей числа n.  
- Шаг 1: Инициализирует переменную summa\_delitel значением 0 для хранения суммы.  
- Шаг 2: Перебирает все числа i от 2 до квадратного корня из n включительно.  
 - Шаг 2.1: Для каждого i проверяет, является ли i простым числом с помощью функции prostik(i) и делится ли n на i.  
 - Шаг2.2: Если i - простое число и делитель n, то добавляет i к summa\_delitel.  
 - Шаг2.3: Если n само является простым числом, то добавляет n к summa\_delitel.  
- Шаг 3: Возвращает значение summa\_delitel.  
  
3**. Основная часть кода:**Инициализирует счетчик count значением 0.  
значением i равным 100

-Шаг0: Если i < 1000, перейти на шаг1, иначе перейти на шаг6.  
- Шаг 1: Для каждого i вычисляет сумму простых делителей i с помощью функции summa\_prost\_delitel(i).  
- Шаг 2: Проверяет, делится ли сумма простых делителей i на 5 без остатка.  
- Шаг 3: Если делится, то увеличивает счетчик count на 1.  
- Шаг 4: Увеличивает i на 1.

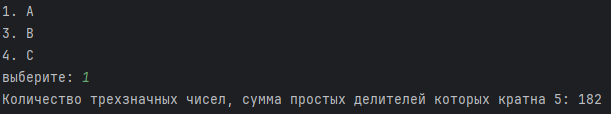
-Шаг5: Переходим на шаг0, иначе идем на шаг6, если выражение становится ложным.  
- Шаг 6: Выводит результат: "Количество трехзначных чисел, сумма простых делителей которых кратна 5:", count.

**Задание 6.**

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

class A:  
 def \_\_init\_\_(self):  
 def prostik(n):  
 # проверка на простое число  
 if n <= 1:  
 return False  
 for i in range(2, int(n \*\* 0.5) + 1):  
 if n % i == 0:  
 return False  
 return True  
  
 def summa\_prost\_delitel(n):  
 # сумма простых делителей числа  
 summa\_delitel = 0  
 for i in range(2, int(n \*\* 0.5) + 1):  
 if prostik(i) and n % i == 0:  
 summa\_delitel += i  
 if prostik(n):  
 summa\_delitel += n  
 return summa\_delitel  
  
 count = 0  
 i = 100 # Начало цикла  
 while i < 1000: # Условие цикла  
 if summa\_prost\_delitel(i) % 5 == 0:  
 count += 1  
 i += 1 # Шаг цикла  
  
 print("Количество трехзначных чисел, сумма простых делителей которых кратна 5:", count)

**Вывод:**

****

**Словесный алгоритм для кода:**  
**Шаг1. Определение функции nod(a, b):**  
  
1.1 Эта функция реализует алгоритм Евклида для нахождения наибольшего общего делителя (НОД) двух чисел a и b.

**Шаг2. Определение функции prom(M, N):**  
  
 2.1 Вычисляем НОД чисел M и N с помощью функции nod(M, N).  
 2.2 Делим числитель M и знаменатель N дроби на НОД. Результаты деления сохраняются в переменные new\_M и new\_N соответственно.   
 2.3 Возвращаем (new\_M, new\_N), представляющий сокращенную дробь.

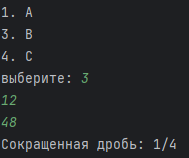
**Шаг3. Основная часть кода:**  
  
 3.1 Считываем числитель дроби M и знаменатель N с помощью input().  
 3.2 Вызываем функцию prom(M, N) для сокращения дроби.  
 3.3 Выводим сокращенную дробь в формате "Сокращенная дробь: {new\_M}/{new\_N}".

**Задание 21.**

Даны два натуральных числа M и N – числитель и знаменатель дроби M/N. Требуется сократить дробь, насколько это возможно.

class B:  
 def \_\_init\_\_(self):  
 def prom(M, N):  
 def nod(a, b):  
 while b:  
 a, b = b, a % b  
 return a  
  
 i = nod(M, N)  
 new\_M = M // i  
 new\_N = N // i  
 return (new\_M, new\_N)  
  
 M = int(input())  
 N = int(input())  
 prom1 = prom(M, N)  
 print(f"Сокращенная дробь: {prom1[0]}/{prom1[1]}")

**Вывод:**



**Словесный алгоритм для кода:**

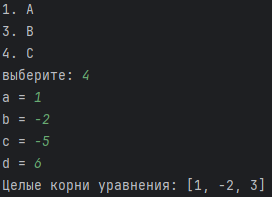
1. **Функция find\_integer\_roots:**  
   Шаг 1: Создать пустое множество под названием roots для хранения найденных целых корней уравнения.  
   Шаг 2: Перебор делителей свободного члена d  
    2.1 Для каждого целого числа i от 1 до абсолютного значения d (назовем его abs(d)):  
       2.2 Проверить, делится ли d на i без остатка.  
     
   Шаг 3: Проверка на корень (для положительного делителя i)  
   Если d делится на i, то:  
       3.1 Подставить i в уравнение a \* i\*3 + b \* i\*2 + c \* i + d.  
       3.2 Если результат равен 0, то i является корнем уравнения, добавить его в множество roots.  
     
   Шаг 4: Проверка на корень (для отрицательного делителя -i)  
   Если d делится на i, то:  
       4.1 Подставить -i в уравнение a \* (-i)\*3 + b \* (-i)\*2 + c \* (-i) + d.  
       4.2 Если результат равен 0, то -i является корнем уравнения, добавить его в множество roots.  
     
   Шаг 5: Вернуть множество roots, содержащее все найденные целые корни уравнения.  
     
   **2. Пример работы алгоритма:**  
     
   Пусть у нас есть уравнение x\*3 - 2x\*2 - 5x + 6 = 0 с коэффициентами a = 1, b = -2, c = -5 и d = 6.  
     
   Шаг 1: Создается пустое множество roots.  
     
   Шаг 2: Проверяются делители d = 6: 1, 2, 3, -1, -2, -3.  
     
   Шаг 3:  
    3.1 i = 1: d делится на 1, подставляем 1 в уравнение и получаем 0, значит 1 - корень, добавляем в roots.  
    3.2 i = 2: d делится на 2, подставляем 2 в уравнение и получаем 0, значит 2 - корень, добавляем в roots.  
    3.3 i = 3: d делится на 3, подставляем 3 в уравнение и получаем 0, значит 3 - корень, добавляем в roots.  
     
   Шаг 4: Поскольку d делится на 1, 2 и 3, мы также проверяем -1, -2 и -3 соответственно, но ни один из них не является корнем.  
     
   Шаг 5: Функция возвращает множество roots со значениями {1, 2, 3}.

**Задание 36.**

Найти все целые корни уравнения a\*x\*\* 3 + b\*x \*\* 2 + c \*x + d, где a,b,c и d – заданные целые числа, причем . (Замечание: целыми корнями могут быть только положительные и отрицательные делители коэффициента **d!!!**).

class C:  
 def \_\_init\_\_(self):  
 def find\_integer\_roots(a, b, c, d):  
  
 roots = set() # Множество для хранения найденных корней  
  
 # Проверяем все делители d, включая положительные и отрицательные  
 for i in range(1, abs(d) + 1):  
 if d % i == 0:  
 if a \* i \*\* 3 + b \* i \*\* 2 + c \* i + d == 0:  
 roots.add(i)  
 if a \* (-i) \*\* 3 + b \* (-i) \*\* 2 + c \* (-i) + d == 0:  
 roots.add(-i)  
  
 return roots  
  
 # Пример использования  
 a = 1  
 b = -2  
 c = -5  
 d = 6  
  
 roots = find\_integer\_roots(a, b, c, d)  
 print("Целые корни уравнения:", roots)

**Вывод:**



**Словесный алгоритм для кода:**

**Функция prom(M, N):**  
**Шаг1. Определение функции nod(a, b):**  
  
1.1 Эта функция реализует алгоритм Евклида для нахождения наибольшего общего делителя (НОД) двух чисел a и b.  
  
**Шаг2. Определение функции prom(M, N):**  
  
 2.1 Вычисляем НОД чисел M и N с помощью функции nod(M, N).  
 2.2 Делим числитель M и знаменатель N дроби на НОД. Результаты деления сохраняются в переменные new\_M и new\_N соответственно.   
 2.3 Возвращаем (new\_M, new\_N), представляющий сокращенную дробь.  
  
**Шаг3. Основная часть кода:**  
  
 3.1 Считываем числитель дроби M и знаменатель N с помощью input().  
 3.2 Вызываем функцию prom(M, N) для сокращения дроби.  
 3.3 Выводим сокращенную дробь в формате "Сокращенная дробь: {new\_M}/{new\_N}".