|  |  |  |
| --- | --- | --- |
| МИНОБРНАУКИ РОССИИ | | |
| Федеральное государственное автономное  образовательное учреждение высшего образования  «Пермский государственный национальный  исследовательский университет» | | |
|  | Институт компьютерных наук и технологий | |
| **ОТЧЁТ**  по индивидуальной работе №2  по дисциплине «Язык программирования Python»  Вариант 1 | | |
|  | | Работу выполнил  студент группы ИКНТ-ИТ-7,  2024-2025 1 курса  Вавилина Анастасия Юрьевна  «\_\_\_» \_\_\_\_\_\_\_\_2024 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 2](#_Toc573149039)

[Алгоритм решения 3](#_Toc701975083)

[Тестирование 5](#_Toc1491303543)

[Код программы 10](#_Toc1988607354)

# Постановка задачи

Реализуйте с помощью двусвязного циклического списка следующую игру: Ученики школы встают в круг. Один ученик пишет программу, генерирующую случайное целое число из промежутка [–10;10]. Если выпало положительное число, то отсчет ведется «по часовой стрелке», если отрицательное, то «против часовой». Ученик, на котором остановился счет, не выбывает из круга, а делает «доброе дело» и уровень его рейтинга увеличивается на 1. Следующий отсчет начинается с ученика, стоящего в круге рядом с тем, который только что делал «доброе дело», с правого (если отсчет ведется по часовой стрелке) или с левого (против часовой). После того, как игра всем надоела, нужно вывести список учеников в порядке не возрастания рейтинга. Если имеется несколько учеников с одинаковым рейтингом, то их нужно выводить в том порядке, в котором они поступили в список. Исходный список фамилий учеников для игры находится в текстовом файле. В начале игры рейтинг каждого ученика равен 0. Во время игры нужно выводить протокол: какое случайное число выпало, какой ученик делал доброе дело и рейтинг ученика после совершения доброго дела. Количество раундов игры вводится пользователем с клавиатуры.

# Алгоритм решения

1. Создание структуры данных.

Класс StudentNode (хранение данных между узлами):

1. name(str) – фамилия участника.
2. rating(int) – текущий рейтинг(начинается с 0).
3. prev(StudentNode) – ссылка на предыдущий узел.
4. next(StudentNode) – ссылка на следующий узел.
5. Реализация двусвязного циклического списка.

Класс StudentCircle (управление кругом учеников):

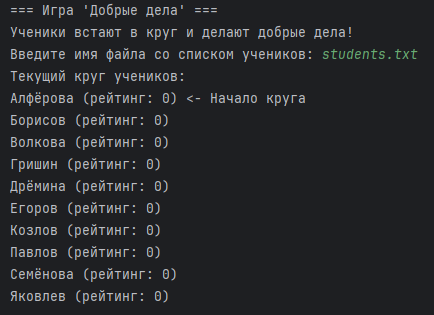
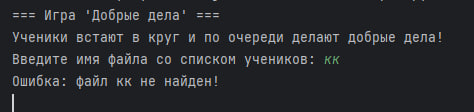
1. add\_student(self, name) – добавляет нового ученика в круг.
   1. Создаёт новый узел.
   2. Если круг пустой делает узел головой и замыкает ссылки на себя.
   3. Иначе вставляет узел перед головой, сохраняя цикличность.
2. find\_student(self, name).
   1. Ищет ученика по имени (без учёта регистра).
   2. Линейный обход круга с проверкой node.name.lower() == name.lower().
3. get\_student\_steps(self, start\_node, steps) - перемещается на steps узлов от start\_node.
   1. Определяет направление (знак steps).
   2. Делает abs(steps) % size шагов, чтобы избежать лишних оборотов.
4. print\_circle(self) - выводит текущее состояние круга.
   1. Проверяет не пуст ли круг.
   2. Начиная с головы, выводит каждого ученика.
   3. Завершает обход при возврате к голове.
5. get\_students\_list - возвращает список (name, rating, original\_index) для сортировки
6. Чтение данных и настройка игры.
   1. Функция read\_students\_from\_file(filename) - загружает список учеников из файла.
      1. Открывает файл, читает строки, пропускает пустые.
      2. Обрабатывает ошибки (например, отсутствие файла).
   2. Функция select\_starting\_student(circle)
      1. Выводит текущий круг (print\_circle())
      2. Запрашивает имя начального ученика.
      3. Если ввод пустой - выбирает голову списка.
      4. Иначе ищет ученика (find\_student), обновляет current\_marker.
7. Игровой процесс.

Функция main()

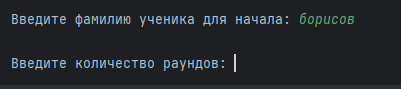
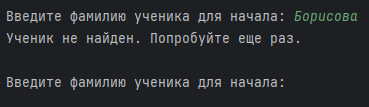
1. Инициализация
   1. Чтение файла: создание круга (StudentCircle).
   2. Выбор начального ученика (select\_starting\_student).
2. Раунды (для каждого раунда):
   1. Запрос числа (get\_number\_input).
   2. Поиск ученика (get\_student\_steps).
   3. Обновление рейтинга.
3. Сортировка и вывод.
   1. Сортировка (sort\_students\_by\_rating)
      1. Собирает список (name, rating, original\_index)
      2. Сортирует по убыванию rating, при равенстве - по original\_index.
   2. Вывод результатов
      1. Печатает рейтинговую таблицу (от 1-го места)

# Тестирование

1. Проверка нахождения файла.

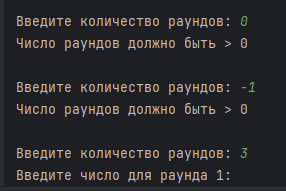


1. Проверка на нахождение ученика.

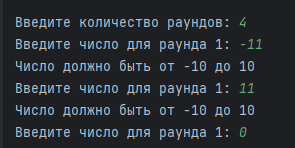


(Считывает при маленьком регистре)

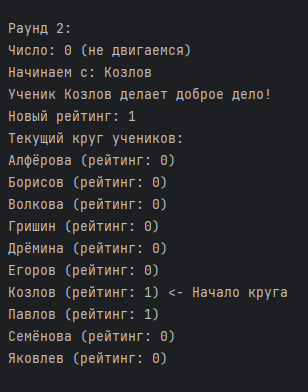
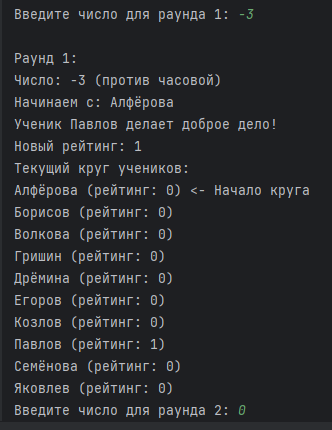
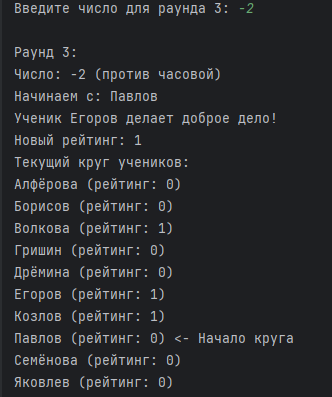
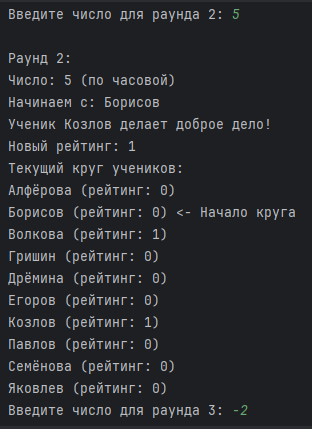
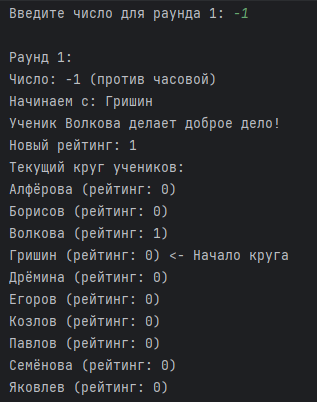
1. Проверка ввода раундов.



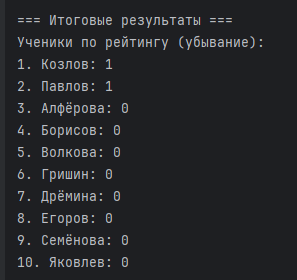
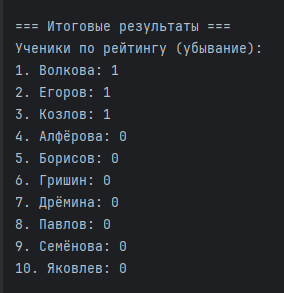
1. Проверка диапазона для числа.



1. Проверка как ставится рейтинг.



1. Вывод итога.



# Код программы

# Импорт необходимых модулей

import random # Для генерации случайных чисел

import sys # Для системных функций (выход из программы)

# Класс узла двусвязного списка

class StudentNode:

# Конструктор класса

def \_\_init\_\_(self, name):

self.name = name # Хранит фамилию ученика

self.rating = 0 # Начальный рейтинг ученика

self.prev = None # Ссылка на предыдущий узел в списке

self.next = None # Ссылка на следующий узел в списке

# Класс для управления кругом учеников

class StudentCircle:

# Инициализация круга

def \_\_init\_\_(self):

self.head = None # Первый элемент списка

self.current\_marker = None # Текущая позиция начала круга

self.size = 0 # Количество учеников в круге

# Добавление нового ученика

def add\_student(self, name):

# Создаем новый узел

new\_node = StudentNode(name)

# Если список пуст

if self.head is None:

# Делаем узел ссылающимся сам на себя

new\_node.next = new\_node

new\_node.prev = new\_node

self.head = new\_node

self.current\_marker = new\_node

else:

# Добавляем в конец списка

last = self.head.prev

last.next = new\_node

new\_node.prev = last

new\_node.next = self.head

self.head.prev = new\_node

# Увеличиваем счетчик учеников

self.size += 1

# Поиск ученика по имени

def find\_student(self, name):

# Если список пуст

if self.head is None:

return None

# Начинаем с головы списка

current = self.head

while True:

# Сравниваем имена без учета регистра

if current.name.lower() == name.lower():

return current

# Переходим к следующему ученику

current = current.next

# Если вернулись к началу - завершаем цикл

if current == self.head:

break

return None

# Перемещение по кругу на N шагов

def get\_student\_steps(self, start\_node, steps):

# Если нет учеников

if self.size == 0:

return None

current = start\_node

# Обработка отрицательных шагов

if steps < 0:

steps = self.size - (abs(steps) % self.size)

# Нормализация шагов

steps %= self.size

# Выполняем шаги

for \_ in range(steps):

current = current.next

return current

# Вывод текущего состояния круга

def print\_circle(self):

# Проверка на пустой круг

if self.head is None:

print("Круг пуст")

return

current = self.head

print("Текущий круг учеников:")

while True:

# Вывод информации об ученике

print(f"{current.name} (рейтинг: {current.rating})", end="")

# Помечаем начало круга

if current == self.current\_marker:

print(" <- Начало круга")

else:

print()

# Переход к следующему ученику

current = current.next

# Проверка завершения круга

if current == self.head:

break

# Сортировка учеников по рейтингу

def get\_sorted\_list(self):

# Если нет учеников

if self.head is None:

return []

students = []

current = self.head

index = 0

# Собираем всех учеников

while True:

students.append((current.name, current.rating, index))

index += 1

current = current.next

if current == self.head:

break

# Сортировка по убыванию рейтинга

students.sort(key=lambda x: (-x[1], x[2]))

# Возвращаем только имя и рейтинг

return [(name, rating) for name, rating, \_ in students]

# Чтение списка учеников из файла

def read\_students\_from\_file(filename):

try:

# Открываем файл на чтение

with open(filename, 'r', encoding='utf-8') as file:

# Читаем непустые строки

return [line.strip() for line in file if line.strip()]

except FileNotFoundError:

# Обработка отсутствия файла

print(f"Ошибка: файл {filename} не найден!")

sys.exit(1)

except Exception as ex:

# Обработка других ошибок

print(f"Ошибка при чтении файла: {ex}")

sys.exit(1)

# Выбор начального ученика

def select\_starting\_student(circle):

# Показываем текущий круг

circle.print\_circle()

while True:

# Запрос имени ученика

name = input("\nВведите фамилию ученика для начала: ").strip()

# Если ввод пустой - берем первого

if not name:

return circle.head

# Поиск ученика

student = circle.find\_student(name)

if student:

# Устанавливаем маркер начала

circle.current\_marker = student

return student

print("Ученик не найден. Попробуйте еще раз.")

# Ввод числа для раунда

def get\_number\_input(round\_num):

while True:

try:

# Запрос числа от пользователя

num\_input = input(f"Введите число для раунда {round\_num}: ").strip()

# Если ввод пустой - генерируем случайное

if not num\_input:

return random.randint(-10, 10)

# Преобразуем в число

num = int(num\_input)

# Проверка диапазона

if num < -10 or num > 10:

print("Число должно быть от -10 до 10")

continue

return num

except ValueError:

print("Ошибка: введите целое число!")

# Основная функция игры

def main():

# Приветствие

print("=== Игра 'Добрые дела' ===")

print("Ученики встают в круг и делают добрые дела!")

# Запрос имени файла

filename = input("Введите имя файла со списком учеников: ")

# Чтение учеников из файла

student\_names = read\_students\_from\_file(filename)

# Проверка на пустой файл

if not student\_names:

print("Файл не содержит имен учеников!")

return

# Создание круга учеников

circle = StudentCircle()

for name in student\_names:

circle.add\_student(name)

# Выбор начального ученика

current\_student = select\_starting\_student(circle)

# Запрос количества раундов

while True:

try:

rounds = int(input("\nВведите количество раундов: "))

# Проверка положительного числа

if rounds <= 0:

print("Число раундов должно быть > 0")

continue

break

except ValueError:

print("Ошибка: введите целое число!")

# Основной игровой цикл

for round\_num in range(1, rounds + 1):

# Получение числа для раунда

number = get\_number\_input(round\_num)

# Определение направления

direction = "по часовой" if number > 0 else "против часовой" if number < 0 else "не двигаемся"

# Вывод информации о раунде

print(f"\nРаунд {round\_num}:")

print(f"Число: {number} ({direction})")

print(f"Начинаем с: {current\_student.name}")

# Выбор ученика для доброго дела

selected = circle.get\_student\_steps(current\_student, number)

# Увеличение рейтинга

selected.rating += 1

# Обновление маркера начала круга

circle.current\_marker = current\_student

# Вывод информации о действии

print(f"Ученик {selected.name} делает доброе дело!")

print(f"Новый рейтинг: {selected.rating}")

# Выбор следующего ученика

current\_student = selected.next if number > 0 else selected.prev

# Вывод обновленного круга

circle.print\_circle()

# Вывод итогов

print("\n=== Итоговые результаты ===")

print("Ученики по рейтингу (убывание):")

# Получение и вывод отсортированного списка

sorted\_students = circle.get\_sorted\_list()

for i, (name, rating) in enumerate(sorted\_students, 1):

print(f"{i}. {name}: {rating}")

# Точка входа в программу

if \_\_name\_\_ == "\_\_main\_\_":

try:

main()

except KeyboardInterrupt:

# Обработка прерывания пользователем

print("\nПрограмма прервана пользователем.")

except Exception as err:

# Обработка других ошибок

print(f"Произошла ошибка: {err}")