|  |  |  |
| --- | --- | --- |
| 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 7 | | |
|  | | Работу выполнил  студент группы ИТ-7-8-2025 1 курса  Батраков Никита Станиславович  «12» июня 2025 г. |
| Работу проверил  \_\_\_\_\_\_\_\_\_\_\_\_\_\_ Фамилия И.О.  «\_\_\_» \_\_\_\_\_\_\_\_\_ 2024 г. |
| Пермь 2025 | | |

СОДЕРЖАНИЕ

[Постановка задачи 3](#_Toc153130027)

[Алгоритм решения 3](#_Toc153130028)

[Тестирование 5](#_Toc153130029)

[Код программы 5](#_Toc153130030)

# Постановка задачи

Игра в «пьяницу». В игре в «пьяницу» карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт – проигрывает. Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту ("шестерка берет туза"). Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды). Напишите программу, которая моделирует игру в пьяницу и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9. Входные данные Программа получает на вход две строки: первая строка содержит 5 чисел, разделенных пробелами—номера карт первого игрока, вторая – аналогично 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой. Выходные данные. Программа должна определить, кто выигрывает при данной раздаче, и вывести слово first или second, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 10\*\*6 ходов игра не заканчивается, программа должна вывести слово botva. Примеры входные данные 1 3 5 7 9 2 4 6 8 0 выходные данные second 5

# Алгоритм решения

1. Класс Queue (Очередь)
   1. Используется для хранения карт игроков по принципу FIFO (First In, First Out).
   2. Основные методы:
      1. enqueue(item) – добавляет карту в конец очереди.
      2. dequeue() – забирает карту из начала очереди.
   3. is\_empty() – проверяет, пуста ли очередь.
2. Класс Player (Игрок)
   1. Хранит имя игрока (first или second) и его колоду карт (Queue).
   2. Основные методы:
      1. add\_cards(cards) – добавляет стартовые карты в колоду.
      2. play\_card() – разыгрывает верхнюю карту.
      3. take\_cards(card1, card2) – забирает выигранные карты (в порядке: сначала свою, потом чужую).
      4. is\_empty() – проверяет, закончились ли карты.

## Ввод данных

* 1. Программа запрашивает у пользователя две строки с картами для каждого игрока
  2. Каждая строка содержит 5 уникальных чисел от 0 до 9, разделенных пробелами

## Проверка корректности ввода для каждого игрока

* 1. Все ли элементы числа
  2. Ровно 5 карт
  3. Карты от 0 до 9
  4. Нет повторяющихся карт
  5. Нет общих карт у игроков

## Инициализация колод

* 1. Создаются две очереди для каждого игрока
     1. Почему именно очередь?
        1. Порядок карт: Карты всегда разыгрываются сверху (из начала очереди) и добавляются вниз (в конец очереди). Это соответствует принципу FIFO, который реализует очередь
        2. Операции: Взять верхнюю карту – dequeue() и добавить карту в низ колоды – enqueuer()
        3. Очередь позволяет легко и эффективно выполнять необходимые операции
  2. Карты из ввода помещаются в соответствующие очереди

## Игровой процесс

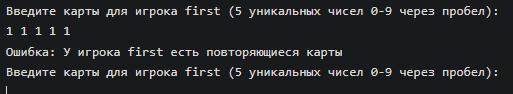
* 1. Игра продолжается пока
     1. У одного из игроков не закончатся карты ИЛИ
     2. Не будет сделано 10\*\*6 ходов
  2. Ходы
     1. Извлечение верхних карт
     2. Определение победителя
     3. Распределение карт
  3. Увеличение счетчика ходов

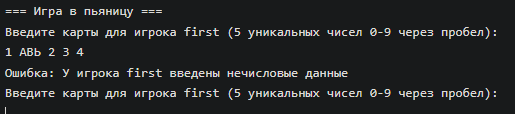
## Проверка окончания игры

* 1. Если у одного из игроков закончились карты, то победа другого
  2. Если сделано 10\*\*6 ходов, вывод на экран слова botva

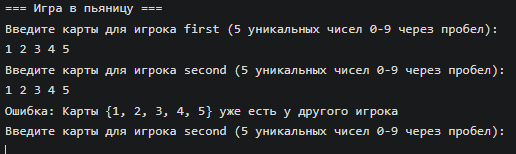
# Тестирование

# 





# 



# Код программы

# Класс узла для связанного списка, используемого в очереди

class QueueNode:

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

self.value = value # Значение узла

self.next = None # Ссылка на следующий узел

# Определяем класс Queue (очередь) для хранения карт игроков

class Queue:

# Инициализация пустой очереди

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

self.head = None # Указатель на начало очереди

self.tail = None # Указатель на конец очереди

# Метод добавления элемента в конец очереди

def enqueue(self, item):

# Создаем новый узел с заданным значением

new\_node = QueueNode(item)

# Если очередь пуста, новый узел становится и головой и хвостом

if self.tail is None:

self.head = new\_node

self.tail = new\_node

else:

# Иначе добавляем новый узел после текущего хвоста

self.tail.next = new\_node

self.tail = new\_node

# Метод извлечения элемента из начала очереди

def dequeue(self):

# Если очередь пуста, возвращаем None

if self.head is None:

return None

# Сохраняем значение головного узла

value = self.head.value

# Перемещаем указатель головы на следующий узел

self.head = self.head.next

# Если голова стала None, значит очередь опустела - обнуляем хвост

if self.head is None:

self.tail = None

# Возвращаем значение извлеченного элемента

return value

# Метод проверки очереди на пустоту

def is\_empty(self):

# Очередь пуста, если голова None

return self.head is None

# Определяем класс Player для представления игрока

class Player:

# Инициализация игрока с именем

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

self.name = name # Имя игрока (first или second)

self.deck = Queue() # Колода карт игрока (экземпляр Queue)

# Метод добавления списка карт в колоду игрока

def add\_cards(self, cards):

for card in cards: # Добавляем каждую карту по очереди

self.deck.enqueue(card)

# Метод проверки, пуста ли колода игрока

def is\_empty(self):

return self.deck.is\_empty() # Используем метод Queue

# Метод розыгрыша верхней карты

def play\_card(self):

return self.deck.dequeue() # Извлекаем карту из колоды

# Метод взятия карт после выигранного раунда

def take\_cards(self, card1, card2):

self.deck.enqueue(card1) # Сначала кладем карту первого игрока

self.deck.enqueue(card2) # Затем карту второго игрока

# Функция проверки корректности введенных данных

def validate\_input(cards\_str, player\_name):

try:

# Пробуем преобразовать строку в список чисел

cards = list(map(int, cards\_str.split()))

except ValueError:

# Если преобразование не удалось - ошибка ввода

raise ValueError(f"У игрока {player\_name} введены нечисловые данные")

# Проверяем, что карт ровно 5

if len(cards) != 5:

raise ValueError(f"У игрока {player\_name} должно быть ровно 5 карт")

# Проверяем, что все карты в диапазоне 0-9

if any(card < 0 or card > 9 for card in cards):

raise ValueError(f"У игрока {player\_name} карты должны быть числами от 0 до 9")

# Проверяем на уникальность карт

if len(set(cards)) != len(cards):

raise ValueError(f"У игрока {player\_name} есть повторяющиеся карты")

return cards # Возвращаем валидный список карт

# Функция получения корректных карт от пользователя

def get\_valid\_cards(player\_name, existing\_cards=None):

while True: # Бесконечный цикл, пока не получим правильные данные

try:

# Запрашиваем ввод карт

cards\_str = input(f"Введите 5 уникальных карт (0-9) для игрока {player\_name}: ")

# Проверяем введенные данные

cards = validate\_input(cards\_str, player\_name)

# Если переданы существующие карты, проверяем на пересечение

if existing\_cards:

common = set(cards) & set(existing\_cards)

if common:

raise ValueError(f"Карты {common} уже есть у другого игрока")

return cards # Возвращаем корректные карты

except ValueError as e:

print(f"Ошибка: {e}") # Выводим сообщение об ошибке

# Основная функция симуляции игры

def drunken\_simulator():

# Получаем карты первого игрока

first\_cards = get\_valid\_cards("first")

# Получаем карты второго игрока с дополнительной проверкой

while True:

second\_cards = get\_valid\_cards("second", first\_cards)

# Проверяем, что карты не пересекаются

common = set(first\_cards) & set(second\_cards)

if not common:

break # Выходим из цикла, если пересечений нет

# Иначе сообщаем об ошибке

print(f"Ошибка: карты {common} присутствуют у обоих игроков")

print("Пожалуйста, измените карты для второго игрока")

# Создаем игроков

first\_player = Player("first")

second\_player = Player("second")

# Добавляем карты игрокам

first\_player.add\_cards(first\_cards)

second\_player.add\_cards(second\_cards)

max\_moves = 10\*\*6 # Максимальное количество ходов

moves = 0 # Счетчик ходов

# Основной игровой цикл

while moves <= max\_moves:

# Проверяем окончание игры

if first\_player.is\_empty():

print(f"second {moves}")

return

if second\_player.is\_empty():

print(f"first {moves}")

return

# Игроки разыгрывают карты

card1 = first\_player.play\_card()

card2 = second\_player.play\_card()

# Определяем победителя раунда

if (card1 == 0 and card2 == 9):

winner = first\_player # Особый случай: 0 бьет 9

elif (card1 == 9 and card2 == 0):

winner = second\_player # Особый случай: 0 бьет 9

elif card1 > card2:

winner = first\_player # Старшая карта побеждает

else:

winner = second\_player # Иначе побеждает второй игрок

# Победитель забирает карты

winner.take\_cards(card1, card2)

moves += 1 # Увеличиваем счетчик ходов

# Если превышено максимальное число ходов

print("botva")

drunken\_simulator()