**Министерство науки и высшего образования Российской Федерации**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»**

**(Университет ИТМО)**

Факультет **Инфокоммуникационных технологий**

Образовательная программа **Мобильные и сетевые технологии**

Направление подготовки(специальность) **09.03.03 Прикладная информатика**

**ПРАКТИЧЕСКАЯ РАБОТА №4**

**По дисциплине «Программирование»**

**Тема: Задачки на ООП.**

**Выполнил** Проскуряков Р. В. K3139

**Проверил** Терещенко В. В.

**Дата** 30.11.2023

**Санкт-Петербург 2023**

**Цель работы**

Познакомиться с ООП на примере языка Python.

**Ход работы**

В ходе выполнения работы были решены две задачки:

1. “Кинотеатр "Сокол"”

Реализован класс HistoryDatabase который:

* 1. Считывает данные о просмотрах из файлов
  2. Рассчитывает список рекомендованных фильмов к просмотру

class HistoryDatabase:

films = dict() # ключ-id фильма; значение-название фильма

history = [] # все фильмы которые посмотрел пользовател i

films\_persons\_assoc = dict() # ключ-фильм; значение-все person, которые его посмотрели

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

self.read\_films("films.txt")

self.read\_history("history.txt")

def read\_films(self, path: str) -> None:

self.films.clear()

with open(path, 'r') as f:

for line in f:

key, film\_name = line.split(',', 1)

self.films[int(key)] = film\_name[:-1] if film\_name[-1] == '\n' else film\_name

def read\_history(self, path: str) -> None:

self.history.clear()

self.films\_persons\_assoc.clear()

with open(path, 'r') as f:

for line in f:

self.history.append(frozenset(map(int, line.split(','))))

for id\_film in self.history[-1]:

self.films\_persons\_assoc[id\_film] = self.films\_persons\_assoc.setdefault(id\_film, []) + [len(self.history) - 1]

def get\_advice(self, views: frozenset) -> str:

matches = dict() # ключ-persona; значение-число совпадений

for views\_film in views:

for id\_person in self.films\_persons\_assoc[views\_film]:

matches[id\_person] = matches.setdefault(id\_person, 0) + 1

len\_views = len(views)

priorities = dict()

for id\_person, number\_matches in matches.items():

k = number\_matches / len\_views

for id\_film in self.history[id\_person]:

priorities[id\_film] = priorities.setdefault(id\_film, 0) + k

result = -1

for id\_film in priorities:

if(id\_film not in views):

result = id\_film

break

if(result == -1):

return "Error! Все фильмы уже просмотрены"

for id\_film, prior in priorities.items():

if(id\_film not in views and prior > priorities[result]):

result = id\_film

return self.films[result]

1. “Разбивка респондентов по возрастным группам”

Реализован класс Human который:

* 1. Хранит в себе имя и возраст человека
  2. Умеет красиво выводить информацию по человеке
  3. Способен сравнивать двух людей по их возрасту и имени

class Human:

\_age = 0

\_name = ""

def \_\_init\_\_(self, \_age: int, \_name: str) -> None:

self.\_age = \_age

self.\_name = \_name

def get\_print(self) -> str:

return self.\_name + ' (' + str(self.\_age) + ')'

def \_\_lt\_\_(self, other) -> bool:

if self.\_age == other.\_age:

return self.\_name > other.\_name

return self.\_age < other.\_age

Реализован класс Group который:

1. Хранит в себе диапазон возрастной группы и всех людей, которые в неё попадают
2. Умеет красиво выводить информацию по группе в отсортированном порядке

class Group:

\_min = 0

\_max = 0

\_Peoples = []

def \_\_init\_\_(self, \_min: int, \_max: int) -> None:

self.\_min = \_min

self.\_max = \_max

self.\_Peoples = list()

def add(self, hum: Human) -> Human:

self.\_Peoples.append(hum)

return hum

def get\_print(self) -> str:

res = ""

if len(self.\_Peoples):

if self.\_max >= 124:

res += str(self.\_min) + '+: '

else:

res += str(self.\_min) + '-' + str(self.\_max) + ': '

res += ', '.join(map(lambda x: x.get\_print(), reversed(sorted(self.\_Peoples)))) + '\n'

return res

**Вывод**

В результате проделанной работы мы познакомились с ООП на примере python и решили две задачки на эту тему.