## Задание 1.

Исходные данные представляют собой словарь. Необходимо поместить их в файл, используя сериализатор.  Организовать считывание данных, поиск, сортировку в соответствии с индивидуальным заданием. Обязательно использовать классы. Реализуйте два варианта: 1)формат файлов CSV; 2)модуль pickle

Реализуйте товар (наименование товара, старая цена, новая цена). Составьте программу, определяющую, на какие товары повысятся цены и на сколько процентов. Выведите информацию о товаре, введенном с клавиатуры

```
import csv
import os.path
import pickle

from CheckDir import CheckDirMixin


class PSuperClass:
    """Просто пример класса для наследования"""
    def __init__(self):
        print("Super Inited")


class Task1(PSuperClass, CheckDirMixin):
    folder = 'Task_1/files/'
    file_csv = folder + "Task_1.csv"
    file_bin = folder + "Task_1.bin"

    class WorkWithCsv:
        """Запись и чтение из csv"""
        def __init__(self, path):
            self.path = path

        def read_all(self):
            with open(self.path, "r+") as f:
                return list(csv.DictReader(f))

        def write_all(self, data):
            with open(self.path, "w+") as f:
                columns = ["name", "old_price", "new_price"]
                writer = csv.DictWriter(f, fieldnames=columns)
                writer.writeheader()
                writer.writerows(data)

        def __str__(self):
            return "csv"

    class WorkWithPickle:
        """Запись и чтение из бинарника"""
        def __init__(self, path):
            self.path = path
```

```
    def read_all(self):
        with open(self.path, "rb+") as f:
            return pickle.load(f)

    def write_all(self, data):
        with open(self.path, 'wb+') as f:
            pickle.dump(data, f)

    def __str__(self):
        return "pickle"

dict = [
    {"name": "Apple", "old_price": 7, "new_price": 8},
    {"name": "Orange", "old_price": 15, "new_price": 14},
    {"name": "Banana", "old_price": 13, "new_price": 16},
    {"name": "Mango", "old_price": 20, "new_price": 21},
    {"name": "Kiwi", "old_price": 16, "new_price": 15},
    {"name": "Lemon", "old_price": 9, "new_price": 10},
    {"name": "Pineapple", "old_price": 25, "new_price": 29},
    {"name": "Peach", "old_price": 14, "new_price": 13},
    {"name": "Pear", "old_price": 8, "new_price": 9},
    {"name": "Nectarine", "old_price": 11, "new_price": 12},
    {"name": "Lime", "old_price": 999, "new_price": 998},
    {"name": "Fig", "old_price": 123, "new_price": 321},
]

def __init__(self):
    super().__init__()
    print("Task 1. CSV and Pickle")
    super().check_dir(self.folder)
    self.csv_ = self.WorkWithCsv(self.file_csv)
    self.pickle_ = self.WorkWithPickle(self.file_bin)

    self.read_write_dict(self.csv_)
    print()
    self.read_write_dict(self.pickle_)

    self.sort_by_key()

    self.individual_task()

    self.get_info()

def read_write_dict(self, cls):
    """Вызов функций для чтения \ записи"""
    print()
    print('-' * 100)
    print("Write using", cls.__str__())
    cls.write_all(self.dict)
    print("Read using", cls.__str__())
    self.print_info(cls.read_all())

def individual_task(self):
    """Найти какие продукты подорожали и насколько"""
    print()
```

```
        print('-' * 100)
        new_dict = {i['name']: (i['new_price'] - i['old_price']) / i['old_price'] for i
in self.dict if
                    i['old_price'] < i['new_price']}
        for i in new_dict.keys():
            print(f' ↑ {i:<13} - {(new_dict[i] * 100):>7.2f}%')

    @staticmethod
    def print_info(elements):
        """Вывод инфы о продуктах фул"""
        for i in elements:
            print("Name: ", i['name'])
            print("Old price", i['old_price'])
            print("New price", i['new_price'])
            print("Diff: ", float(i['new_price']) - float(i['old_price']))
            print("Diff%: ", (float(i['new_price']) - float(i['old_price'])) /
float(i['old_price']) * 100)
            print()
        print('~' * 10)

    def get_info(self):
        """Инфа по названию"""
        while True:
            inp = input("Enter product name or 0 to exit back:  ")
            if inp == "0":
                return
            new_dict = [i for i in self.dict if i['name'].lower() == inp.lower()]
            if len(new_dict) > 0:
                self.print_info(new_dict)
            else:
                print("Product not found")

    def sort_by_key(self):
        """Сортировка по всему что есть только в этой таске"""
        print()
        print('-' * 100)
        print("Unsorted")
        self.print_info(self.dict)

        sorted_by_name = sorted(self.dict, key=lambda x: x['name'])
        print("Sorted by name")
        self.print_info(sorted_by_name)

        sorted_by_old_price = sorted(self.dict, key=lambda x: x['old_price'])
        print("Sorted by old price")
        self.print_info(sorted_by_old_price)

        sorted_by_new_price = sorted(self.dict, key=lambda x: x['new_price'])
        print("Sorted by new price")
        self.print_info(sorted_by_new_price)

        sorted_by_diff = sorted(self.dict, key=lambda x: x['new_price'] -
x['old_price'])
        print("Sorted by price difference")
        self.print_info(sorted_by_diff)
```

## Задание 2.

В соответствии с заданием своего варианта составить программу для анализа текста. Считать из исходного файла текст. Используя регулярные выражения получить искомую информацию (см. условие), вывести ее на экран и сохранить в другой файл. Заархивировать файл с результатом с помощью модуля zipfile и обеспечить получение информации о файле в архиве.
Также выполнить общее задание – определить и сохранить в файл с результатами:

- количество предложений в тексте;
- количество предложений в тексте каждого вида отдельно (повествовательные, вопросительные и побудительные);
- среднюю длину предложения в символах (считаются только слова);
- среднюю длину слова в тексте в символах;
- количество смайликов в заданном тексте. Смайликом будем считать последовательность символов, удовлетворяющую условиям:

первым символом является либо «;» (точка с запятой) либо «:» (двоеточие) ровно один раз;
далее может идти символ «-» (минус) сколько угодно раз (в том числе символ минус может идти ноль раз);
в конце обязательно идет некоторое количество (не меньше одной) одинаковых скобок из следующего набора: «(», «)», «[», «]»;
внутри смайлика не может встречаться никаких других символов. Например, эта последовательность является смайликом: «;---------[[[[[[[[». Эти последовательности смайликами не являются: «]», «;--»,«:»,«)».

Вывести все слова, включающие сочетание букв верхнего регистра и цифр.
Проверить, надежно ли составлен пароль. Пароль считается надежным, если он состоит из 8 или более символов, где символом может быть английская буква, цифра или знак подчеркивания. Пароль должен содержать хотя бы одну заглавную букву, одну маленькую букву и одну цифру.

```
import re
import zipfile

import CheckDir


class Task2(CheckDir.CheckDirMixin):
    folder = 'Task_2/files'
    file_src = folder + '/src.txt'
    file_ans = folder + '/ans.txt'
```

```
    file_zip = folder + '/ans.zip'
    text = ''

    def __init__(self):
        super().check_dir(self.folder)
        print("Task 2. Text")
        self.read_file()
        print("Text\n", self.text)
        self.general_task()
        if not self.read_file():
            return
        self.my_task()

    def read_file(self):
        try:
            with open(self.file_src, 'r') as f:
                self.text = f.read()
                return True
        except Exception as e:
            print("Error:", e)
            return False

    def general_task(self):
        print("Sentence count:", self.find_sentence_count())
        print("Sentence count by type [. ! ?]:",
self.find_sentence_type_count())
        print("Average sentence len:",
self.find_average_sentence_len())
        print("Average word len:", self.find_average_word_len())
        print("Smile count:", self.find_smiles_count())
        with open(self.file_ans, 'w+') as f:
            f.write(f"Sentence count:
{self.find_sentence_count()}\n")
            f.write(f"Sentence count by type [. ! ?]:
{self.find_sentence_type_count()}\n")
            f.write(f"Average sentence len:
{self.find_average_sentence_len()}\n")
            f.write(f"Average word len:
{self.find_average_word_len()}\n")
            f.write(f"Smile count: {self.find_smiles_count()}\n")
```

```
    def my_task(self):
        a = self.find_word_with_capital_digit()
        with open(self.file_ans, 'a+') as f:
            print("Print all words that include combination of
uppercase letters and numbers:", a)
            f.write(f"Print all words that include combination of
uppercase letters and numbers: {a}")
            for i in a:
                print("Is good password " + i,
self.check_password_good(i))
                f.write(f"Is good password {i}
{self.check_password_good(i)}\n")
            print("Not capital letters words",
self.find_not_capital_word_cnt())
            f.write(f"Not capital letters words
{self.find_not_capital_word_cnt()}\n")
        with zipfile.ZipFile(self.file_zip, 'w') as zip_file:
            zip_file.write(self.file_ans, arcname=self.file_ans)

    def find_sentence_count(self):
        """Кол-во предложений"""
        return len(re.findall(r'[\.!?]', self.text))

    def find_sentence_type_count(self):
        """Кол-во предложений по типу"""
        sentence_type_list = re.findall(r'[\.!?]', self.text)
        return sentence_type_list.count('.'),
sentence_type_list.count('!'), sentence_type_list.count('?')

    def find_average_sentence_len(self):
        """Средняя длина"""
        return sum(len(elem) for elem in re.findall(r'\w+',
self.text)) / self.find_sentence_count()

    def find_average_word_len(self):
        """Среднее кол-во букв"""
        lst = re.findall(r'\w+', self.text)
        return sum(len(elem) for elem in lst) / len(lst)

    def find_smiles_count(self):
        """Смайлы"""
```

```
        return len(re.findall(r'[:;]-*(\)+|\(+|]+|\[+)',
self.text))

    def find_word_with_capital_digit(self):
        """1 большая, 1 цифра"""
        return
re.findall(r'\b(?=[A-Za-z0-9]*[A-Z])(?=[A-Za-z0-9]*[0-9])(?=.*)[
A-Za-z0-9]{1,}\b', self.text)

    def check_password_good(self, text):
        """Хороший ли пароль"""
        return
len(re.findall(r'^(?=[A-Za-z0-9_]*[A-Z])(?=[A-Za-z0-9_]*[0-9])(?
=[A-Za-z0-9_]*[a-z])[A-Za-z0-9_]{8,}$',
                                text)) == 1

    def find_not_capital_word_cnt(self):
        """Маленькие слова"""
        return len(re.findall(r'\b(?=[a-z])[a-z]+\b', self.text))
```

## Задание 3.

В соответствии с заданием своего варианта доработать программу из ЛР3, использовав класс и обеспечить:
а) определение дополнительных параметров среднее арифметическое элементов последовательности, медиана, мода, дисперсия, СКО последовательности;
б) с помощью библиотеки matplotlib нарисовать графики разных цветов в одной координатной оси

$$\arcsin x = \sum_{n=0}^{\infty} \frac{(2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} = x + \frac{x^3}{6} + \frac{3x^5}{40} + \ldots, |x| \le 1$$

```
import math
import statistics

import matplotlib.pyplot as plt

import CheckDir


class Task3(CheckDir.CheckDirMixin):
```

```
    def __init__(self):
        super().check_dir("Task_3/img/")
        print("Task 3. Plot")
        try:
            x = float(input("Введите от -1 до 1: "))
        except ValueError as e:
            print("Не. Не то чтото...")
            return

        ans, n, lst = self.my_arcsin(x, 0.0001)
        if ans is None:
            return
        print('Посчитанный вручную результат: ', ans)
        print('Посчитанный результат с помощью модуля math: ',
math.acos(x))
        print('Количество итераций: ', n)
        print(f'Среднее арифметическое: {statistics.mean(lst)}')
        print(f'Медиана: {statistics.median(lst)}')
        print(f'Мода: {statistics.mode(lst)}')
        print(f'Дисперсия: {statistics.variance(lst)}')
        print(f'СКО: {statistics.stdev(lst)}')
        self.draw(1, 1000)

    def my_arcsin(self, x: float, eps: float):
        """Мой арксин"""
        if abs(x) > 1:
            print("\nX should be in range -1..1")
            return None, None, None

        n = 0
        n_fact = 1
        n_2_fact = 1
        ans = 0
        x_2n1 = x

        new_sum = lambda: n_2_fact / (4 ** n * (n_fact * n_fact)
* (2 * n + 1)) * x_2n1
        arr = []

        while new_sum() > eps:
```

```
            arr.append(ans)
            ans += new_sum()
            n += 1
            if n == 500:
                break
            n_fact *= n
            n_2_fact *= 2 * n * (2 * n - 1)
            x_2n1 *= x * x

        return ans, n, arr

    def draw(self, h, rng):
        """Функция, выполняющая построение графика с функцией варианта."""
        my_sin = [self.my_arcsin(elem, 0.001)[0] for elem in [i / rng for i in range(-rng, rng, h)]]
        math_sin = [math.asin(elem) for elem in [i / rng for i in range(-rng, rng, h)]]
        fig, ax = plt.subplots()
        ax.plot([i / rng for i in range(-rng, rng, h)], my_sin, 'red', linewidth=2, label='My')
        ax.plot([i / rng for i in range(-rng, rng, h)], math_sin, 'blue', linewidth=2, label='Math')
        ax.legend(loc='lower left')
        ax.set_xlabel('X')
        ax.set_ylabel('Y')
        plt.savefig('Task_3/img/task3.png')
        plt.show()
```

## Задание 4.

В соответствии с заданием своего варианта разработать базовые классы и классы наследники.
Требования по использованию классов:
Абстрактный класс «Геометрическая фигура» содержит абстрактный метод для вычисления площади фигуры (https://docs.python.org/3/library/abc.html )
Класс «Цвет фигуры» содержит свойство для описания цвета геометрической фигуры (https://docs.python.org/3/library/functions.html#property )
Класс «Прямоугольник» (Круг, Ромб, Квадрат, Треугольник и т.д.) наследуется от класса «Геометрическая фигура». Класс должен содержать конструктор по параметрам

«ширина», «высота» (для другого типа фигуры соответствующие параметры, например, для круга задаем «радиус») и «цвет». В конструкторе создается объект класса «Цвет фигуры» для хранения цвета. Класс должен переопределять метод, вычисляющий площадь фигуры https://docs.python.org/3/library/math.html .
Для класса «Прямоугольник»(тип фигуры в инд. задании)
определить метод, который возвращает в виде строки основные параметры фигуры, ее цвет и площадь. Использовать метод format (https://pyformat.info/ )
название фигуры должно задаваться в виде поля данных класса и возвращаться методом класса.
В корневом каталоге проекта создайте файл main.py для тестирования классов.
Используйте конструкцию, описанную в https://docs.python.org/3/library/__main__.html
Пример объекта: Прямоугольник синего цвета шириной 5 и высотой 8.
Программа должна содержать следующие базовые функции:
1) ввод значений параметров пользователем;
2) проверка корректности вводимых данных;
3) построение, закрашивание фигуры в выбранный цвет, введенный с клавиатуры, и подпись фигуры текстом, введенным с клавиатуры;
4) вывод фигуры на экран и в файл.

Построить правильный пятиугольник со стороной a.

```python
import math
from abc import ABC, abstractmethod


from matplotlib import pyplot as plt


import CheckDir




class Color:
    def __init__(self, color):
        """Функция, инициализирующая объект класса."""
        self.color = color


    @property
    def color_init(self):
        """Функция-геттер для переменной color."""
        return self.color


    @color_init.setter
    def color_init(self, new_color):
        """Функция-сеттер для переменной color."""
```

```
        self.color = new_color


    @color_init.deleter
    def color_init(self):
        """Функция-делетер для переменной color."""
        del self.color


    def __str__(self):
        """Магический метод, to_string."""
        return self.color



class GeometricFigure(ABC):


    @abstractmethod
    def square(self):
        """Функция, вычисляющая площадь фигуры."""



class Pentagon(GeometricFigure):
```

```
    def __init__(self, name, a, color):
        self.a_ = a
        self.name = name
        self.color = Color(color)


    @property
    def a(self):
        """Функция-геттер для переменной"""
        return self.a_


    @a.setter
    def a(self, a):
        """Функция-сеттер для переменной."""
        self.a_ = a


    @property
    def name(self):
        """Функция-геттер для переменной name."""
        return self.name_


    @name.setter
```

```
    def name(self, name):
        """Функция-сеттер для переменной name."""
        self.name_ = name

    def square(self):
        """Тут считаем площадь"""
        return self.a_ ** 2 * math.sqrt(25 + 10 * math.sqrt(5)) /
4

    def draw(self):
        """Тут рисуем пятиугольник"""
        try:
            dots_x = []
            dots_y = []
            r = self.a_ / 2 / math.sin(math.pi * 36 / 180)
            for i in range(0, 361, 72):
                dots_x.append(r * math.sin(math.pi * i / 180))
                dots_y.append(r * math.cos(math.pi * i / 180))

            plt.legend('', frameon=False)
```

```
            plt.plot(dots_x, dots_y)

            plt.fill(dots_x, dots_y, alpha=0.5,
facecolor=self.color.color_init)



            plt.title(str(self.name_) + " площадь " +
str(self.square()))

            plt.grid(True)

            plt.savefig('Task_4/img/task4.png')

            plt.show()

        except Exception as e:

            print("Чтото пошло не так...", e)




class Task4(CheckDir.CheckDirMixin):

    def __init__(self):

        super().check_dir('Task_4/img/')

        print("Task 4. Pentagon")

        a = int(input("Введите длину стороны: "))

        p = Pentagon("Pentagon", a, 'black')

        p.draw()
```

## Задание 5.

В соответствии с заданием своего варианта исследовать возможности библиотека NumPy при работе с массивами и математическими и статическими операциями. Сформировать целочисленную матрицу A[n,m] с помощью генератора случайных чисел (random). Отсортировать матрицу по убыванию элементов последнего столбца.
Вычислить среднее значение элементов последнего столбца. Ответ округлите до сотых.
Вычисление среднего значения выполнить двумя способами: через стандартную функцию и через программирование формулы

```
import random

import numpy as np


class Task5:
    def __init__(self):
        print("Task 5. Numpy")
        n, m = (int(input('Введите длину матрицы: \n')),
                int(input('Введите ширину матрицы: \n')))
        self.arr = np.array([[random.randint(-100, 100) for _ in
range(n)] for _ in range(m)])
        print('Матрица:')
        print(self.arr)
        self.sort_it()
        print('Отсортированная матрица:')
        print(self.arr)
        print('Среднее np: ')
        print(f'{np.mean(self.arr[:, -1]):<.2f}')
        print('Среднее my: ')
        print(f'{self.mean():.2f}')

    def sort_it(self):
        """Сортим матрицу"""
        self.arr = np.array(sorted(self.arr, key=lambda x: x[-1],
reverse=True))

    def mean(self):
        """Среднее в ласт столбце"""
        ans = 0
        for i in self.arr:
```

```
            ans += i[-1]
        return ans / len(self.arr)
```