

Министерство образования Республики Беларусь  
Учреждение образования «Белорусский государственный университет  
информатики и радиоэлектроники»

Факультет компьютерных систем и сетей  
Кафедра Информатики  
Дисциплина «Избранные главы информатики»

**ОТЧЕТ**  
к лабораторной работе №4  
на тему:  
**«РАБОТА С ФАЙЛАМИ, КЛАССАМИ, СЕРИАЛИЗАТОРАМИ,  
РЕГУЛЯРНЫМИ ВЫРАЖЕНИЯМИ И СТАНДАРТНЫМИ  
БИБЛИОТЕКАМИ»**  
БГУИР 6-05-0612-02

Выполнил студент группы 353504  
ГОРДАШУК Владислав Сергеевич

---

(дата, подпись студента)

Проверил  
Жвакина Анна Васильевна

---

(дата, подпись преподавателя)

Минск 2024

Функции для загрузки данных в файл и чтения из файла с использованием pickle и csv. Реализован пример использования mixin.

```
class PickleSerializerMixin:
    @staticmethod
    def write_pickle(data: list[dict[str, any]], filename: str):
        with open(filename, "wb") as fh:
            pickle.dump(data, fh)

    @staticmethod
    def read_pickle(filename: str):
        with open(filename, "rb") as fh:
            return pickle.load(fh)

class CsvSerializerMixin:
    @staticmethod
    def write_csv(data: list[dict[str, any]], filename: str):
        """Save to CSV"""
        with open(filename, 'w', newline='') as file:
            writer = csv.DictWriter(file, fieldnames=data[0].keys())
            writer.writeheader()
            writer.writerows(data)

    @staticmethod
    def read_csv(filename: str) -> list[dict[str, any]]:
        """Load from CSV"""
        with open(filename, 'r') as file:
            reader = csv.DictReader(file)
            return list(reader)

class Serializer(CsvSerializerMixin, PickleSerializerMixin):
    pass
```

Класс с методами для анализа данных.

```
class TRPNormAnalyze:
    def __init__(self, students: list[Student], run_standard: float, jump_standard: float):
        self.students = students
        self.run_standard = run_standard
        self.jump_standard = jump_standard

    def not_pass(self) -> list[Student]:
        """
        Checks who hasn't passed the norm
        Returns:
        | list[Student]: not passed
        """
        return [student for student in self.students if student.run > self.run_standard or student.jump < self.jump_standard]

    def passed(self) -> int:
        """
        Count how many passed the norm
        Returns:
        | int
        """
        return len([student for student in self.students if student.run <= self.run_standard or student.jump >= self.jump_standard])
```

```

def top3_jump(self) -> list[Student]:
    """
    Find best students in jump
    Returns:
    | list[Student]: top 3
    """
    return sorted(self.students, key=lambda x: x.jump)[:3]

def top3_run(self) -> list[Student]:
    """
    Find best students in run
    Returns:
    | list[Student]: top 3
    """
    return sorted(self.students, key=lambda x: x.run)[:3]

def get_student(self, name: str):
    """
    Find student for name
    Args:
    | name (str): student name

    Returns:
    | Student
    """
    for s in self.students:
        if s.name.lower() == name.lower():
            return s
    return None

```

Класс с статическими функциями для анализа текста.

```

class FileAnalazer:
    @staticmethod
    def get_char_with_num(content):
        """
        Search for word with num and upper case char
        Args:
            content (str): text

        Returns:
            int: num of words
        """
        pattern = r'\b\w*[A-Z][0-9]|\w*[0-9][A-Z]\w*\b'
        matches = re.findall(pattern, content)
        return matches

    @staticmethod
    def is_ip(string):
        """
        Checking whether it is an IP
        Args:
            content (str): string

        Returns:
            bool
        """
        pattern = r'^((25[0-5]|2[0-4]\d|[01]?\d\d?)\.){3}(25[0-5]|2[0-4]\d|[01]?\d\d?)$'
        return bool(re.fullmatch(pattern, string))

```

```

@staticmethod
def shortest_with_w(content):
    """
    Find shortest word with 'w'
    Args:
        content (str): text

    Returns:
        str: shorter word
    """
    pattern = r'\b\w*w\b'
    matches = re.findall(pattern, content)
    return min(matches, key=len)

```

```

@staticmethod
def in_len_increase(content: str):
    """
    Sort words in length increasing
    Args:
        content (str): text

    Returns:
        list[string]
    """
    words = re.findall(r"\b[a-zA-Zа-яА-Я]+\b", content)
    return sorted(words, key=len)

```

```

@staticmethod
def sent_num(content: str):
    """
    Count num of sentences
    Args:
        content (str): text

    Returns:
        int: num of sentences
    """
    words = re.findall(r'[.!?]', content)
    if len(words) < 1:
        return 1
    return len(words)

```

```

@staticmethod
def num_of_narrative(content: str):
    """
    Count num of narrative sentences
    Args:
        content (str): text

    Returns:
        int: num of narrative sentences
    """
    words = re.findall('\.', content)
    return len(words)

```

```

@staticmethod
def num_of_interrogative(content: str):
    """
    Count num of interrogative sentences
    Args:
        content (str): text

    Returns:
        int: num of interrogative sentences
    """
    words = re.findall('\?', content)
    return len(words)

```

```

@staticmethod
def num_of_incentive(content: str):
    """
    Count num of incentive sentences
    Args:
        content (str): text

    Returns:
        int: num of incentive sentences
    """
    words = re.findall('!', content)
    return len(words)

```

```

@staticmethod
def avr_sent_len(content: str):
    """
    Calculate avarage length of sentences
    Args:
        content (str): text

    Returns:
        float
    """
    words = content.split()
    return len(words) / FileAnalazer.sent_num(content)

```

```

@staticmethod
def avr_word_len(content: str):
    """
    Calculate avarage length of word
    Args:
        content (str): text

    Returns:
        float
    """
    words = content.split()
    num = 0
    for word in words:
        num += len(word)
    return num / len(words)

```

```

@staticmethod
def num_of_smiles(content: str):
    """
    Count num of smiles
    Args:
        content (str): text

    Returns:
        int
    """
    pattern = r'[:;]-*([\[\]\]])\1*'
    matches = re.findall(pattern, content)
    return len(matches)

```

Класс для подсчета косинуса, получения членов ряда Тейлора. Определены дополнительные параметры: среднее арифметическое элементов последовательности, медиана, мода, дисперсия, СКО последовательности.

```
class Cosine:
    def __init__(self):
        self.eps = 0.0001
        self.res = 1.0
        self.term = 1.0
        self.terms = [self.term]

    def calculate(self, x: float):
        """
        Calculate cosine

        Args:
            x (float)

        Returns:
            float: cos(x)
        """
        x = x % (2 * math.pi)
        for inum in range(1, 500):
            self.term *= (-1) * x * x / ((2 * inum - 1) * (2 * inum))
            self.res += self.term
            self.terms.append(self.res)
            if abs(self.term) < self.eps:
                break
        return self.res
```

```

def get_median(self):
    """
    Calculate median
    Returns:
    |     float
    """
    return statistics.median(self.terms)

def get_mode(self):
    """
    Calculate mode
    Returns:
    |     float
    """
    return statistics.mode(self.terms)

def get_mean(self):
    """
    Calculate mean
    Returns:
    |     float
    """
    return statistics.mean(self.terms)

def get_variance(self):
    """
    Calculate variance
    Returns:
    |     float
    """
    return statistics.variance(self.terms)

def get_standart_deviation(self):
    """
    Calculate deviation
    Returns:
    |     float
    """
    return statistics.stdev(self.terms)

```

Класс фигуры правильный шестиугольник.

```

class Hexagon(Shape):
    shape_type = "Hexagon"
    def __init__(self, a: float, color: str):
        self.a = a
        self.color = Color(color)

        self.x_points = [a,
                          a / 2,
                          -a / 2,
                          -a,
                          -a / 2,
                          a / 2,
                          a]

        self.y_points = [0,
                          a * math.sqrt(3) / 2,
                          a * math.sqrt(3) / 2,
                          0,
                          -a * math.sqrt(3) / 2,
                          -a * math.sqrt(3) / 2,
                          0]

    def draw(self, color="", title=""):
        """
        Draw hexagon with params

        Args:
            color (str, optional): _description_. Defaults to "".
            title (str, optional): _description_. Defaults to "".
        """
        if color:
            self.color.color(color)
        if title:
            plt.title(title)
        plt.fill(self.x_points, self.y_points, color=self.color.color)
        plt.show()
        plt.savefig("shape.png")

    def area(self) -> float:
        return self.a * self.a * 3 * math.sqrt(3) / 2

    def __str__(self):
        return "Shape: {}, Color: {}, a: {}".format(
            self.shape_type,
            self.color.color,
            self.a
        )

```

Класс матрицы и унаследованный от него класс целочисленной матрицы.

```
class Matrix:
    def __init__(self, rows, colms):
        self.rows = rows
        self.colms = colms
        self.arr = np.empty((rows, colms))

    def fill(self):
        """Fill matrix with random nums"""
        self.arr = np.random.uniform(0, 100, size=(self.rows, self.colms))
        return self

    def __str__(self):
        return str(self.arr)

class IntMatrix(Matrix):
    def __init__(self, rows, colms):
        super().__init__(rows, colms)

    def fill(self):
        self.arr = np.random.randint(0, 100, size=(self.rows, self.colms))

def get_min_in_diag(self):
    """
    Return min value in side diag

    Arg: -

    Return: int
    """
    if self.colms != self.rows:
        return None
    return min([self.arr[i][self.rows - 1 - i] for i in range(self.rows)])

def get_var_in_diag(self):
    """
    Return variance of side diag

    Arg: -

    Return: float
    """
    if self.colms != self.rows:
        return None
    return round(np.var([self.arr[i][self.rows - 1 - i] for i in range(self.rows)]), 2)
```

```

def my_get_var_in_diag(self):
    """
    Return variance of side diag without using 'var'

    Arg: -

    Return: float
    """
    if self.colms != self.rows:
        return None
    diag = [self.arr[i][self.rows - 1 - i] for i in range(self.rows)]
    mean = np.mean(diag)
    variance = np.mean((diag - mean) ** 2)
    return round(variance, 2)

def __str__(self):
    return super().__str__()

```