Лабораторная работа № 12

Знакомство с Python. Основные структуры данных.

Цель работы: приобретение практических навыков создания приложений на языке Python

Задание: Для выполнения лабораторной работы требуется установить интерпретатор Python версии 3.5+. Выполнить написание программы-сценария в соответствии с вариантом задания. Провести тестирование. Оформить отчет.

Вариант 3.

В текстовом файле записана матрица состоящая из 0 и 1. Эта матрица описывает контур фигуры найденной на изображении. Причем фигура, получаемая путем соединения всех единиц, является замкнутая. Определить наибольшее число точек принадлежащих одной окружности и для них найти центр.

```python
import itertools


class Point:
    """Класс для представления точки в двумерном пространстве."""
    def __init__(self, x: float, y: float):
        self.x = x
        self.y = y


    def __eq__(self, other: 'Point') -> bool:
        """Проверка равенства точек с учетом погрешности."""
        epsilon = 1e-6
        return (abs(self.x - other.x) < epsilon and
                abs(self.y - other.y) < epsilon)


    def __repr__(self) -> str:
        return f"Point({self.x:.2f}, {self.y:.2f})"
```

```
class Circle:
    """Класс для представления окружности."""
    def __init__(self, center: Point, radius: float):
        self.center = center
        self.radius = radius

    def contains_point(self, point: Point, epsilon: float = 1e-6) -> bool:
        """Проверяет, лежит ли точка на окружности с учетом погрешности."""
        dx = point.x - self.center.x
        dy = point.y - self.center.y
        distance_sq = dx**2 + dy**2

        return abs(distance_sq - self.radius**2) < epsilon

    def __repr__(self) -> str:
        return f"Circle(center={self.center}, radius={self.radius:.2f})"

class ContourAnalyzer:
    """Анализирует контур фигуры для поиска окружности с наибольшим числом точек."""
    def __init__(self):
        self.points = []

    def load_matrix(self, file_path: str) -> None:
        """Загружает матрицу из файла и извлекает точки контура."""
        self.points = []
        with open(file_path, 'r') as file:
```

```
        for y, line in enumerate(file):
            row = line.strip().split()
            for x, value in enumerate(row):
                if value == '1':
                    self.points.append(Point(x, y))

def find_max_circle(self) -> tuple[int, Point | None]:
    """
    Находит окружность, содержащую наибольшее число точек контура.
    Возвращает кортеж (максимальное количество точек, центр окружности).
    """
    points = self.points

    if not points:
        return (0, None)

    n = len(points)

    if n == 1:
        return (1, points[0])
    elif n == 2:
        center = Point((points[0].x + points[1].x)/2,
                       (points[0].y + points[1].y)/2)
        return (2, center)

    max_count = 0
    best_circle = None
```

```
        for triplet in itertools.combinations(points, 3):
            circle = self._circle_from_three_points(*triplet)

            if not circle:
                continue

                            current_count = sum(1 for p in points if
circle.contains_point(p))
             if current_count > max_count or (current_count == max_count and
not best_circle):
                max_count = current_count
                best_circle = circle

              return (max_count, best_circle.center) if best_circle else
(max_count, None)

    def _circle_from_three_points(self, p1: Point, p2: Point, p3: Point) ->
Circle | None:
          """Создает окружность по трем точкам. Возвращает None, если точки
коллинеарны."""
        if self._are_collinear(p1, p2, p3):
            return None
        x1, y1 = p1.x, p1.y
        x2, y2 = p2.x, p2.y
        x3, y3 = p3.x, p3.y

        a1 = 2 * (x2 - x1)
        b1 = 2 * (y2 - y1)
        c1 = x2**2 + y2**2 - x1**2 - y1**2
```

```
        a2 = 2 * (x3 - x1)
        b2 = 2 * (y3 - y1)
        c2 = x3**2 + y3**2 - x1**2 - y1**2

        determinant = a1 * b2 - a2 * b1

        if abs(determinant) < 1e-6:
            return None

        h = (c1 * b2 - c2 * b1) / determinant
        k = (a1 * c2 - a2 * c1) / determinant
        radius = ((h - x1)**2 + (k - y1)**2)**0.5

        return Circle(Point(h, k), radius)

    def _are_collinear(self, p1: Point, p2: Point, p3: Point, epsilon: float
= 1e-6) -> bool:
        """Проверяет, коллинеарны ли три точки."""
        area = (p2.x - p1.x) * (p3.y - p1.y) - (p2.y - p1.y) * (p3.x - p1.x)
        return abs(area) < epsilon
```

```
import unittest
from contour_analyzer import ContourAnalyzer, Point

class TestContourAnalyzer(unittest.TestCase):
    def setUp(self):
        self.analyzer = ContourAnalyzer()
```

```
def test_single_point(self):
    with open('single_point.txt', 'w') as f:
        f.write('1')

    self.analyzer.load_matrix('single_point.txt')
    count, center = self.analyzer.find_max_circle()

    self.assertEqual(count, 1)
    self.assertEqual(center, Point(0, 0))

def test_two_points(self):
    with open('two_points.txt', 'w') as f:
        f.write('1 0\n')
        f.write('0 1\n')

    self.analyzer.load_matrix('two_points.txt')
    count, center = self.analyzer.find_max_circle()

    self.assertEqual(count, 2)

    expected_center = Point(0.5, 0.5)

    self.assertEqual(center, expected_center)

def test_four_points_on_circle(self):
    matrix = [
        ['0', '0', '1', '0', '0'],
        ['0', '0', '0', '0', '0'],
```

```
            ['1', '0', '0', '0', '1'],
            ['0', '0', '0', '0', '0'],
            ['0', '0', '1', '0', '0']
        ]

        with open('circle.txt', 'w') as f:
            for row in matrix:
                f.write(' '.join(row) + '\n')

        self.analyzer.load_matrix('circle.txt')

        count, center = self.analyzer.find_max_circle()

        self.assertEqual(count, 4)
        self.assertEqual(center, Point(2.0, 2.0))



    def test_collinear_points(self):
        matrix = [['1', '1', '1']]
        with open('collinear.txt', 'w') as f:
            for row in matrix:
                f.write(' '.join(row) + '\n')

        self.analyzer.load_matrix('collinear.txt')
        count, center = self.analyzer.find_max_circle()

        self.assertEqual(count, 0)
        self.assertIsNone(center)
```

```python
    def test_empty_file(self):
        with open('empty.txt', 'w') as f:
            f.write('')

        self.analyzer.load_matrix('empty.txt')
        count, center = self.analyzer.find_max_circle()

        self.assertEqual(count, 0)
        self.assertIsNone(center)


if __name__ == '__main__':
    unittest.main()
```

```
> python3 test_contour_analyzer.py
.....
----------------------------------------------------------------------
Ran 5 tests in 0.001s

OK
```

Вывод: приобрели практические навыки создания приложений на языке Python