Оформление спецификации к лабораторной работе

Спецификация к лабораторной работе как отчетная форма, сопровождающая компьютерную программу, сдается вместе с программой, оформляется на отдельном листе или в тетради. В заголовке указывается номер и название лабораторной работы, фамилия и инициалы выполнившего работу студента, номер варианта, название учебной группы. Спецификация должна содержать следующие пункты:

1. Четкую и однозначную **постановку задачи.**
2. **Состав входных и выходных данных.**
3. Описание **метода** решения задачи.
4. Описание **алгоритма** решения задачи в виде блок-схемы.
5. Хорошо продуманные **функциональные тесты.**

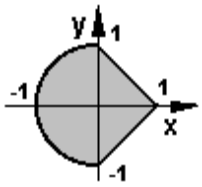
Указания к выполнению лабораторной работы №2

Рекомендуется выполнять работу поэтапно. Выполнение пунктов 1-3 не предполагают работы за компьютером.

1. Сформулировать постановку задачи и обдумать состав входных и выходных данных. Оформить пункты 1 и 2 спецификации работы.

Пример:

Постановка задачи: определить, сколько из заданных точек попали в заданную область.



Состав входных и выходных данных:

	Имя переменной	Тип переменной	Смысл переменной
ВХОД	N	цел.	Общее количество точек
	x, y	вещ.	Абсцисса и ордината очередной точки
ВЫХОД	p	цел.	Количество точек, попавших в заданную область

2. В соответствии со своим индивидуальным вариантом составить логическое выражение принадлежности точки с координатами (*x*,*y*) заданной области. Полученное

выражение записать как метод в спецификацию. Для этой работы разрешается не включать в спецификацию описание алгоритма в виде блок-схемы.

Пример:

Метод: точка с координатами (x, y) попадает в заданную область, если:

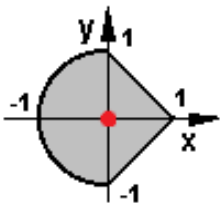
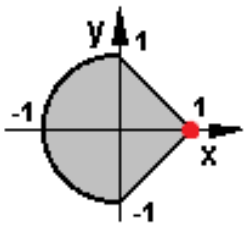
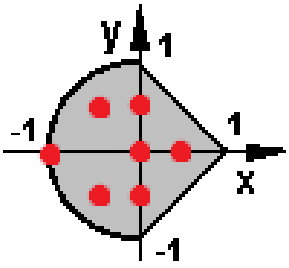
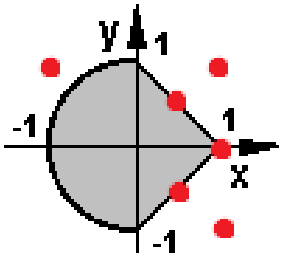
$$\left(x^2 + y^2 \le 1\right) and \left(y < 1 - x\right) and \left(y > x - 1\right)$$

(pow(x, 2) + pow(y, 2) < 1) and (y < 1 – x) and (y > x – 1)

3. Составить пять функциональных тестов. Подобрать координаты точек таким образом, чтобы проиллюстрировать корректность программы в разных ситуациях (когда точка попадает внутрь заданных областей; снаружи; на границах, входящих в заданную область; на границах, не входящих в заданную область) и изобразить каждый тест на отдельном рисунке. Ниже приводится ориентировочная схема тестов:

- задана одна точка, попавшая в область;
- задана одна точка, не попавшая в область;
- заданы 7 разных точек, все попали в область;
- заданы 6 разных точек, все не попали в область;
- заданы 5 разных точек, часть из которых попадает в область, а часть – нет.

Пример:

<table>
<tr>
<td>Тест 1.
N=1
x = 0, y = 0
S = 1</td>
<td> 
</td>
<td>Тест 2.
N = 1
X = 1, y = 0
S = 0</td>
<td> 
</td>
</tr>
<tr>
<td>Тест 3.
N = 7
x = 0, y = 0
x = 0, y = 0.5
x = 0, y = –0.5
x = –1, y = 0
x = 0.5, y = 0
x = –0.5, y = –0.5
x = –0.5, y = 0.5
S = 7</td>
<td> 
</td>
<td>Тест 4.
N = 6
x = -1, y = 1
x = 1, y = 1
x = 0.5, y = 0.5
x = 0.5, y = -0.5
x = 1, y = -1
x = 1, y = 0
S = 0</td>
<td> 
</td>
</tr>
<tr>
<td colspan="4">Тест 5.</td>
</tr>
</table>

N = 5

x = 0, y = 0

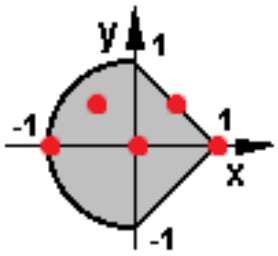
x = 1, y = 0

x = -1, y = 0

x = 0.5, y = 0.5

x = -0.5, y = 0.5

S = 3



4. Приступить к разработке программы. Прежде всего, обеспечить ввод координат (x, y) одной точки, проверку условия ее принадлежности заданной области и вывод результата этой проверки («точка принадлежит области» или «точка не принадлежит области»). Выполнить проверку работы программы на подготовленных ранее функциональных тестах.

Пример программного кода №1:

```
(01) # лабораторная работа 2
(02) # ввод координат точки
(03) print("Введите координаты точки")
(04) x = float(input())
(05) y = float(input())
(06) # проверка условия принадлежности точки (x,y) области
(07) if (pow(x,2)+pow(y,2)<=1) and (y<1-x) and (y>x-1):
(08)       # действие, выполняющееся только если условие ВЕРНО
(09)       print("Точка принадлежит области :)")
(10) else:
(11)       # действие, выполняющееся только если условие НЕ ВЕРНО
(12)       print("Точка не принадлежит области :(")
```

Отметим, что строки 01, 02, 06, 08 и 11 на выполнение программы никак не влияют, потому что начинаются со знака #. Такие строки называются ***комментариями*** и служат для пояснения фрагментов программного кода. Рекомендуется умеренное использование комментариев. Здесь и далее комментарии относятся к следующим за ними фрагментам кода.

В приведенной выше программе присутствует ***условный оператор*** (строки 07, 09, 10, 12) – базовая конструкция, позволяющая выполнять некоторый набор операций, только если выполняется определенное условие (или другой набор, если условие не выполняется). В строке 07 записан ***заголовок условного оператора***, между ключевым словом `if` и знаком двоеточия записывается ***условие***. Условием может быть только ***логическое (булево) выражение*** – выражение, о котором после подстановки в него значений переменных можно сказать, истинно оно или ложно. После заголовка условного оператора необходимо отступить от левого края и записать ***блок операторов*** (т.е. один или несколько операторов), выполняющихся, если условие истинно. В нашем случае таким блоком является один оператор `print`, записанный в строке 09. В строке 10

размещено ключевое слово `else` и знак двоеточия, которые сигнализируют о начале другого блока операторов, выполняющихся только при невыполнении условия. Само ключевое слово должно быть записано на том же уровне, что и соответствующий ему оператор `if`, а его блок операторов снова записывается с отступом от левого края, как это сделано в примере (см. строку 12).

Довольно часто условный оператор используется в усеченном варианте (в виде так называемой ***полуальтернативы***), когда необходимо выполнить какие-либо действия, если условие истинно, и ничего не делать, если условие ложно. Запись такого оператора состоит из заголовка и блока операций, выполняющихся при истинности условия. Ключевое слово `else` пропускается. Пример использование полуальтернативы будет приведен ниже.

Немного упростить запись заголовка в условном операторе можно, используя ***логические переменные***. Переменные, которые могут хранить в себе лишь два значения – истина (`True`) или ложь (`False`) – называются логическими. Их использование часто помогает проще записать громоздкие условия. Запишем наше условие попадания точки через логические переменные. Для этого предлагается рассматривать нашу область по простым фрагментам (для рассматриваемого случая, например, полукруг и треугольник).

Введем новую логическую переменную `PointInSemicircle`, которая будет принимать значение `True`, только если точка попала в полукруг.

```
PointInSemicircle = (pow(x,2)+pow(y,2)<=1) and (x<0)
```

Аналогично введем логическую переменную принадлежности точки треугольнику:

```
PointInTriangle=(y<1-x) and (y>x-1) and (x>=0)
```

Понятно, что точка попала в область, если она попала в полукруг **ИЛИ** она попала в треугольник. На основании этого можем переписать заголовок условного оператора в следующем виде:

```
if PointInSemicircle or PointInTriangle:
```

5. На этом этапе добавим программе способность проверять на принадлежность области заданное число точек и подсчитывать количество попавших в нее. Довольно очевидно, что для каждой точки следует выполнить следующие действия:

- ввод координат очередной точки (x, y);
- проверка на принадлежность точки (x, y) заданной области;
- учет факта попадания, если он имел место.

Для реализации повторяющихся действий в программировании используются ***циклы***. В настоящей работе предлагается использовать параметрический цикл `for`, так как нам заранее известно, сколько раз следует повторить вышеуказанные действия – ровно столько, сколько задано точек – N. Рассмотрим структуру цикла на примере кода №2. Оператор параметрического цикла расположен в строках 06-13 и состоит из двух частей: из ***заголовка цикла*** (строка 06) и блока повторяющихся операций – ***тела цикла*** (строки 07-13). Весь блок тела цикла необходимо размещать со сдвигом от левого края относительно заголовка цикла.

Пример программного кода №2:

```
(01) # лабораторная работа 2
(02) i=0
(03) p=0
(04) print("Введите количество точек")
(05) n=int(input())
(06) for i in range(n):
(07)      print("Введите координаты точки")
(08)      x = float(input())
(09)      y = float(input())
(10)      PointInSemicircle=(pow(x,2)+pow(y,2)<=1) and (x<0)
(11)      PointInTriangle=(y<1-x) and (y>x-1) and (x>=0)
(12)      if PointInSemicircle or PointInTriangle:
(13)           p=p+1 # добавление к значению переменной p единицы
(14) if p==0:
(15)      print("В область не попало ни одной точки")
(16) else:
(17)      print("В область попало ", p, " точек")
```

В представленном примере используется условный оператор `if` в виде полуальтернативы – пропущен блок операций, выполняющихся при невыполнении условия. Также изменено действие для случая выполнения условия: теперь вместо вывода в консоль сообщения о попадании точки производится увеличение значения переменной p на единицу.

Увеличить переменную p на единицу можно с помощью более компактной записи

```
p+=1
```

Смысл этой операции – добавление величины, стоящей справа от оператора +=, к переменной, стоящей слева. Аналогично работают операторы `-=`, `*=` и /=.

6. Организация тестирования с применением скриптов. Использование специальных файлов для тестирования позволит существенно облегчить тестирование программ. Однако такой подход потребует подготовки.

Структура файла для тестирования несложная, он состоит всего из трёх строк.

```
(1) @echo off
(2) <путь к интерпретатору> <путь к программе> [<параметры>]
(3) pause
```

Первая строка содержит команду, запрещающую выводить в консоль информацию о запуске других команд, чтобы не загромождать окно вывода посторонней информацией.

Вторая строка состоит из трех частей: первая часть – это путь к интерпретатору языка Python, далее ставится пробел, и указывается вторая часть – путь к программе. Третья часть содержит параметры программы.

Третья строка отвечает за выполнение задержки перед завершением программы и закрытием консоли.

Запустим сначала программу с помощью скрипта, чтобы убедиться, что пути к интерпретатору и программе прописаны верно. Для этого создадим новый текстовый файл и запишем в него вышеописанные строки.

Например

```
@echo off
C:\Python34\Python C:\Python34\Labs\Lab2.py
pause
```

Затем сохраним этот файл под именем test1, изменив при этом расширение на bat (test1.bat). Двойной щелчок левой кнопкой мыши по созданному файлу должен запустить нашу программу. Если в консоли будут указаны ошибки или она закроется сразу после запуска, следует проверить, что пути к интерпретатору и программе указаны верно. Также проблемы могут возникнуть из-за несоответствия кодировок в случае, если пути содержат кириллические символы.

Если программа запустилась и отработала верно, создадим четыре копии нашего файла для тестирования. Назовем копии test2.bat, test3.bat, test4.bat и test5.bat.

Добавим во вторую строку файла test1.bat параметры для запуска. Вспомним, что первый тест содержит одну точку с координатами (0.0, 0.0), как указано в разделе функциональных тестов. Для изменения файла щелкнуть по нему правой кнопкой и выбрать пункт «Изменить» контекстного меню. Таким образом, файл примет вид:

```
@echo off
C:\Python34\Python C:\Python34\Labs\Lab2.py 1 0.0 0.0
pause
```

Единица во второй строке отвечает за количество точек, подаваемых на вход, а два вещественных нуля – это этой координаты точки. У каждого из этих параметров свой порядковый номер, начинающийся с одного: у единицы порядковый номер 1, первый из нулей имеет номер 2, а последний ноль – номер 3. Все параметры запуска доступны в программе, они расположены в специальной переменной `argv`. Чтобы узнать параметр, стоящий, например, на втором месте, необходимо обратиться к переменной `argv` и указать номер параметра в квадратных скобках:

```
argv[1]
```

Теперь мы должны указать программе, какой из параметров в какую переменную класть. Для этого потребуется немного изменить программу.

Пример программного кода №3:

```
(01) # лабораторная работа 2
(02) from sys import argv
(03) i=0
(04) p=0
(05) n=int(argv[1])
(06) print("Введены координаты ", n, " точек")
(07) for i in range(n):
(08)      x=float(argv[2*(i+1)])
(09)      y=float(argv[2*(i+1)+1])
(10)      print("(", x,", ", y, ")" )
(11)      PointInSemicircle=(pow(x,2)+pow(y,2)<=1) and (x<0)
(12)      PointInTriangle=(y<1-x) and (y>x-1) and (x>=0)
(13)      if PointInSemicircle or PointInTriangle:
(14)           p+=1 # добавление к значению переменной p единицы
(15) if p==0:
(16)      print("В область не попало ни одной точки")
(17) else:
(18)      print("В область попало ", p, " точек")
```

В строке (02) подключена возможность работы со списком параметров командной строки. Приглашение к вводу количества точек и сам ввод с клавиатуры заменен простым связыванием переменной *n* с первым параметром из командной строки, это показано в строке 05. В строке 06 размещен оператор печати в консоль, который выведет значение переменной *n* для удобства.

На вводе координат точек остановимся подробнее. Приглашения к вводу и сам ввод заменен простым чтением параметров командной строки (строки 06 и 07), аналогично тому, как считано количество точек. Но если количество точек – это единственный параметр с порядковым номером 1, то с координатами точек сложнее. Потому что координаты

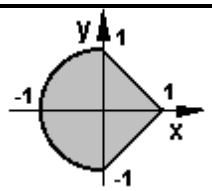
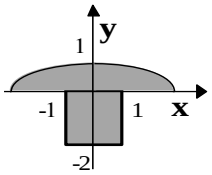
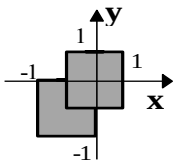
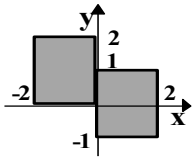
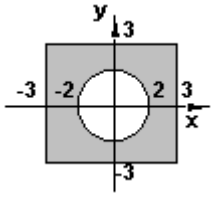
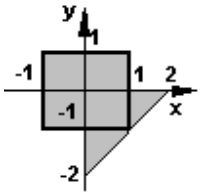
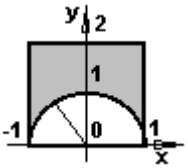
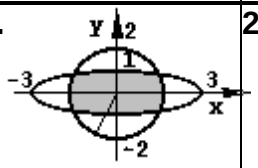
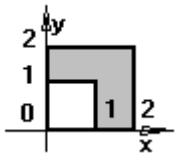
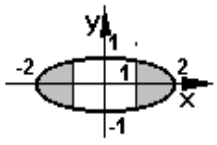
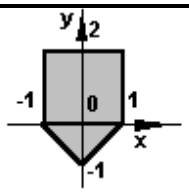
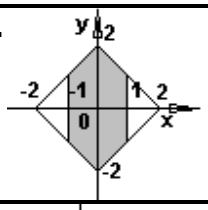
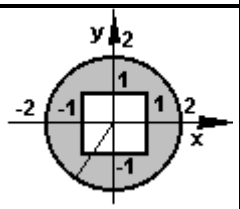
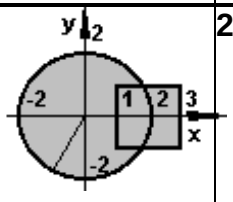
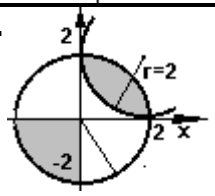
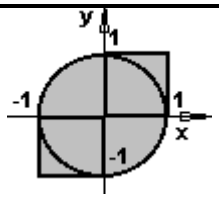
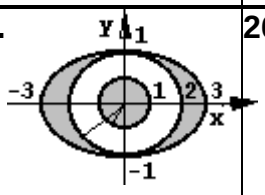
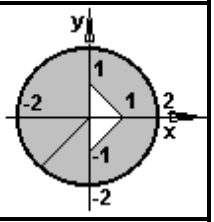
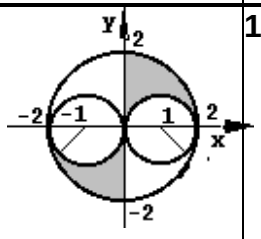
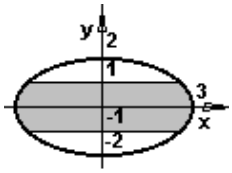
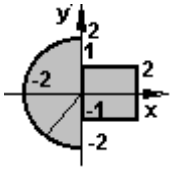
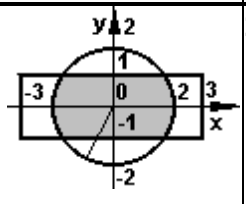
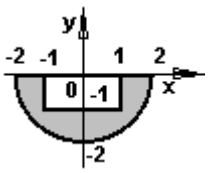
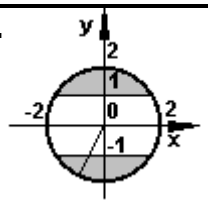
- первой точки имеют порядковые номера в командной строке 2 и 3;
- второй точки имеют порядковые номера в командной строке 4 и 5 и т.д.

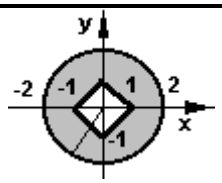
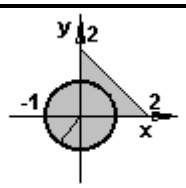
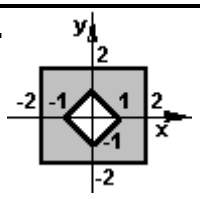
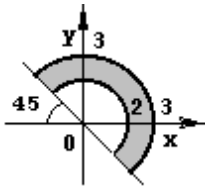
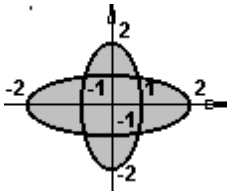
За номер точки, попадание в область которой проверяется в текущий момент, отвечает переменная *i*. В программе точки нумеруются с 0 до (N – 1). На основании этого можем составить зависимость порядковых номеров координат x и y от номера точек и записать эту зависимость (см. строки 08 и 09). Для удобства координаты каждой точки выводятся в консоль в строке 10.

Изменив таким образом программу, можем попробовать запустить первый тест, дважды щелкнув левой кнопкой мыши по файлу test1.bat. В случае корректного выполнения программы, добавить параметры запуска в остальные тестовые файлы и убедиться, что все отработали корректно.

7. Сдать лабораторную работу и полностью оформленную к ней спецификацию преподавателю, продемонстрировав программный код и корректное прохождение всех пяти тестов.

Варианты заданий к лабораторной работе 2

1. 	11. 	21. 
2. 	12. 	22. 
3. 	13. 	23. 
4. 	14. 	24. 
5. 	15. 	25. 
6. 	16. 	26. 
7. 	17. 	27. 
8. 	18. 	28. 

9. 	**19.** 	**29.** 
10 	**20.** 	**30.**