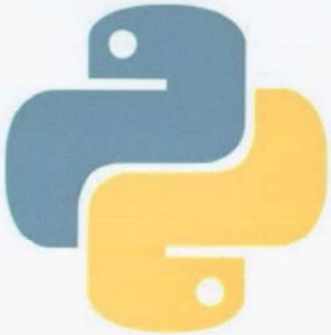
МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
имени В.Ф. УТКИНА

Python. Процедуры и функции.

Методические указания к лабораторной работе

17



Рязань 2022

УДК 004.432

Python. Процедуры и функции: методические указания к лабораторной работе/ Рязан. гос. радиотехн. ун-т.; сост.: А.В.Климухина, А.Н.Пылькин, Ю.С.Соколова, Е.С. Щенёв, М.Г. Щетинин. Рязань, 2022 / Рязань: ИП Коняхин А.В. (Book Jet), 2022. – 31 с.

Рассмотрены основные правила объявления и использования подпрограмм (в форме процедур и функций). Приведены правила оформления функций с помощью оператора def и с помощью lambda – выражения. Определен механизм передачи параметров в функцию и обратно, а также рассмотрены правила определения области действия переменных (области видимости переменных) в зависимости от их статуса и контекста использования переменных в технологии функционального программирования.

В качестве практических заданий предлагается составить программу с использованием подпрограмм. Предназначены для студентов направлений 09.03.03 – Прикладная информатика и 09.03.04 – Программная инженерия по дисциплине «Алгоритмические языки и программирование», а также для студентов очной и заочной формы обучения всех направлений подготовки и специальностей.

Рис.6.

Процедуры. Функции. Возврат значений. Механизм передачи параметров в функцию. Область видимости. Локальная переменная. Глобальная переменная. Структура программы.

Печатается по решению Научно-методического совета Рязанского государственного радиотехнического университета имени В.Ф. Уткина.

Рецензент: кафедра информатики, информационных технологий и защиты информации ФГБОУ ВО «Липецкий государственный педагогический университет им. П.П. Семенова-Тян-Шанского» (зав. каф., к.т.н., доц. Скуднев Д.М.).

Python. Процедуры и функции

Составители: Климухина Анастасия Витальевна
Пылькин Александр Николаевич
Соколова Юлия Сергеевна
Щенёв Евгений Сергеевич
Щетинин Максим Геннадьевич

Процедуры и функции

В практике программирования достаточно часто встречаются алгоритмы, в которых повторяются фрагменты, одинаковые по выполняемым действиям и различающиеся только значениями обрабатываемых данных. При составлении программы по такому алгоритму приходится задавать одну и ту же группу операторов соответственно для каждого из повторяющихся фрагментов. Для более эффективного программирования подобных алгоритмов в языках вводится понятие подпрограммы. Повторяющаяся группа операторов оформляется в виде самостоятельной программной единицы со своими входными и выходными данными – подпрограммы. Она записывается однократно, а в соответствующих местах программы обеспечиваются лишь обращения к ней (ссылки) и передача данных.

Повторяющиеся операторы оформляют в виде процедуры или функции. Во многих версиях языка Python дается определение функции, а процедура рассматривается в качестве частного случая описания и использования функции. Однако механизмы использования процедур и функций различны.

Процедуры

Рассмотрим механизм использования процедуры. Процедура может быть с параметрами или без параметров. В качестве примера процедуры без параметров ниже показана процедура вывода сообщения об ошибке, если три введенных числа не являются сторонами треугольника (подобная процедура часто используется для контроля вводимых данных).

```
# процедур без параметров
def Err():  # определение процедуры
    print("ОШИБКА: неверные данные")
a = int( input('введите первое число = '))
b = int( input('введите второе число = '))
c = int( input('введите третье число = '))
key =
(a>0)and(b>0)and(c>0)and(a+b>c)and(a+c>b)and(b+c>a)
if not key:
    Err()   # вызов процедуры
```

Вызов процедуры осуществляется с помощью отдельной строки путем указания ее имени, за которым следуют круглые скобки: в приведенном примере это `Err()`

В большинстве практических случаев используют процедуры (и функции) с параметрами. Процедура, которая выводит сумму, разность, произведение и частное двух чисел имеет вид:

```
# процедура с параметрами
def PROC(a, b):  # a, b - формальные параметры
    print('сумма = ', a + b)
    print('разность = ', a - b)
    print('произведение = ', a * b)
    print('частное = ', a / b)
x = 2.5
PROC(x, 0.5)   # x, 0.5 - фактические параметры
```

Таким образом, описание процедуры в общем виде реализуется с помощью схемы:

```
def <имя_процедуры> (<аргумент>, …, <аргумент>):
        # <строки_документации>
        <команда>
        <команда>
        …
        <команда>
```

Процедура в отличие от функции не имеет средств для возврата значений.

Определение функции

Функции (а также процедур) являются абстракциями, в которых совокупность некоторых действий скрывается под отдельным именем. Разветвленный набор функций обеспечивает их эффективное использование много раз. Библиотеки языка Python содержит множество типовых и отлаженных функций, имеющих широкое применение. Например, встроенные функции, математические функции модуля math, функции обработки строковых данных, функции для работы с регулярными выражениями. Все эти функции определены и показаны правила их использования в ранних темах. В тех случаях, когда некоторый участок программы не используется несколько раз, но относительно выходных и входных данных он достаточно автономен, его так же целесообразно выделить в отдельную функцию. Такой подход реализует концепцию

структурного программирования и делает программу более понятной и удобочитаемой.

С другой стороны, подобный подход обеспечивает реализацию технологий функционального программирования. **Функциональное программирование** — это стиль программирования, использующий только композиции функций. Другими словами, это программирование в выражениях, а не в императивных командах.

Функция – это просто блок кода, который можно переиспользовать несколько раз в разных местах программы. Мы можем вызывать функцию с какими-то аргументами и получать значения обратно.

Определить функцию в Python можно двумя способами:

1) с помощью оператора def;
2) с помощью lambda-выражения.

Первый способ позволяет использовать операторы. При втором – определение функции может быть выполнено в форме выражения.

Задание функции с помощью оператора def

Определение функции начинается ключевым словом def и именем функции, далее в круглых скобках указываются формальные параметры (аргументы). С помощью отступов задается блок кода функции, в котором указываются выполняемые команды. Возврат значения (результата выполнения функции) или нескольких значений осуществляется отдельной строкой return.

Описание функции в общем виде:

```
def <имя_функции> (<аргумент>, …, <аргумент>):
        # <строки_документации>
        <команда>
        <команда>
        …
        <команда>
        return <возвращаемые_параметры>
```

Список аргументов определяет множество формальных параметров. Все аргументы являются локальными идентификаторами, которые могут использоваться только внутри блока кода функции. При вызове функции формальные параметры заменяются на фактические

параметры, которые определены в программе и которые определяют выполнение вызываемой функции. Строки документации описывают процесс, который реализуется внутри функции. Служебное слово return показывают те параметры (идентификаторы), значение которых возвращаются в основную программу.

Функции в Python являются объектами (также строки, списки или классы. Функцию можно передавать в другую функцию, ее можно возвращать из функций, создавать внутри других функций. Функции являются объектами первого класса, т.е. объектами, которые могут быть динамически созданы, уничтожены, переданы другим функциям, возвращены как значения и имеют статус и права, как другие переменные. Это означает, что нет ограничений на ее использование, например, смотрите следующий код:

```
    def f(x):  # a, b - формальные параметры
    return x+10
r = f
print(r(8))# напечатается значение 18
print(r)   # появится сообщение об ошибке
```

Определение функции с помощью lambda-выражения

Функцию, определённую с помощью lambda-выражения, называют анонимной функцией. Например, анонимная функция может быть объявлена следующим образом:

func() = lambda x, y:x +y

Подобным образом обеспечивается безымянный объект-функция. Этот объект можно использовать для того, чтобы связать его с некоторым идентификатором (именем). На практике в большинстве случаев определяемые lambda-выражением функции используются в качестве параметров в других функциях.

Количество аргументов вариантов lambda-функции неограниченно, однако lambda-выражение должно быть единственным, поэтому возвращается также единственное значение. Опытные программисты стремятся обойтись без использования lambda-функции, которые по своей сути является обычными функциями и возвращают только одно значение. С другой стороны в некоторых случаях

применение lambda-функций упрощает написание и восприятие программы на Python.

Есть смысл сравнивать механизм использования процедуры и функции. С этой целью ниже приведены две простейшие программы.

Процедура:

```
# определение процедуры
def print_sum (a = 2, b = 2):
    sum = a + b
    print(sum)
# вызов процедуры
print_sum(5,26) # выведется значение 32
```

Функция:

```
# определение функции
def sum (a = 2, b = 2):
    sum = a + b
    return sum
# вызов функции как операнда выражения
C = sum(4, 3) # переменная C возвращает сумму 7
print(sum(5,26)) # возвращаемое значение -
                 # аргумент другой функции
                 # выведется значение 21
```

Следует иметь ввиду, что вызов функции реализуется в компьютере медленнее, чем обычный программный код, поэтому при реализации цикла, когда требуется обеспечить максимальное быстродействие, нужно просто использовать внутри тела цикла обычный программный код.

Необязательные параметры

При описании функции аргументы, которые указываются в круглых скобках после имени функции, могут быть обязательными и необязательными. Аргументы, для которых заданы начальные значения, являются необязательными. Обязательные аргументы

задаются указанием имени. При описании (задании) функции вначале указываются обязательные параметры и затем необязательные. При вызове функции допускается не указывать необязательные параметры. В этом случае значения аргументов не изменяются и определяются начальными значениями.

Программа, реализующая обязательные (arg1) и необязательные (arg2, arg3) параметры, имеет вид:

```
def func(arg1, arg2=5, arg3=99)
    print(arg1)
    print(arg2)
    print(arg3)
func('обязательный параметр', arg3=77)
```

Результат выполнения программы:

```
обязательный параметр
5
77
```

Функция с переменным числом аргументов

В ряде случаев в Python можно создать и использовать функцию, в которой предусматривается переменное число аргументов. Это реализуется с помощью переменных в качестве аргумента списка (или массива). Рассмотрим один из возможных способов задания функции с переменным числом аргументов, что обеспечивается тем, что перед именем аргумента ставится символ звёздочки "*". При передаче аргументов в этом случае они помещаются в кортеж, имя которого соответствует имени аргумента:

```
def variable_len(*args):
    for x in args:
        print(x)
variable_len(1,2,3) # выведет в столбик '1 2 3'
```


В том случае, когда вместо символа звёздочки указывается два символа звёздочки, соответствующие аргументы помещается не в список, а в словарь:

```
def variable_len(**args):
    print(type(args))
    for x, value in args.items():
        print(x, value)
variable_len(apple = "яблоко" , bread = "хлеб")
# выведет 'apple яблоко' и 'bread хлеб'
```

Возврат значений

Возврат требуемого значения фиксируется в процессе описания функции с помощью оператора *return*. Из функции удается вернуть одно или несколько значений. В Python предусмотрено, что возвращаемым обьектом может быть число, строка или None.

Для возврата нескольких значений требуется указать соответствующие элементы через запятую. При возвращении нескольких значений необходимо использовать некоторый контейнер. Требуется после ключевого слова *return* указать имя контейнера. В следующем примере в качестве контейнеры используется список:

```
def x(n)
    a = [1, 2, 3]
    a = a * n
    return a
print(x(2)) # выведет [1, 2, 3, 1, 2, 3]
```

Возвращаемые переменные можно перечислять через запятую, тогда их перечисление создаст некоторый список. Перезадаваемые значения присваиваются соответствующим переменным из списка. Подробный возврат значений из функции демонстрируется следующим программным кодом:

```
def f()
    z1 = "python"
    z2 = 44
    z3 = True
    return z1, z2, z3
```

```
z1, z2, z3 = f()
print(z1, z2, z3) # выведет 'python 44 True'
```

Передача параметров в функцию и обратно

Рассмотрим простейший пример, поясняющий синтаксис определения функции и её параметров. Определим функцию F, которая преобразует исходную строку str в список (через запятую) отдельных строк-тегов:

```
def F(str)
    spisok = []
    for x in str.split(' '):
        spisok.append(x.strip())
    return spisok
print(F('РГРТУ является ведущим вузом Рязани'))
# выведет список
#['РГРТУ', 'является', 'ведущим', 'вузом',
'Рязани']
```

В приведённой программе функция F возвращает в программу список. Если попытаться вызвать эту функцию без параметров, то появится сообщение об ошибке TypeError, так как функция ожидает параметр, а он отсутствует.

В рассмотренном примере в явном виде не указаны типы параметров функции, однако это не обязательно и соответствует динамическим свойством языка Python.

Аннотирование типов

Последние версии языка Python предусматривают возможность явного указания типа и параметров функции. Реализуется это с помощью двоеточия для входных параметров (аргументов) и стрелки для указания возвращаемого значения функции. В примере демонстрируется аннотирование типов для входных и выходных параметров:


```
def add(x: int, y: int) -> int:
    return x + y
print(add(10, 11))
print(add('знание',add('-', 'сила')))
# выведет '21' и 'знание - сила'
```

Аннотация типов помогает программисту или используемой среде разработки (IDE - сокращение, встречающиеся в описании Python), выявить ошибки различного характера. Программный код при этом будет исполняться и при передаче параметров других типов, отличающихся от указанных явно.

Механизм передачи параметров в функцию

В статических языках программирования существует чёткое различие передачи параметров по ссылке и по значению (например, в Паскале это параметры-переменные и параметры значения). В языке Python каждая переменная имеет определённую связь идентификатора (и имени переменной) с объектом в памяти компьютера. Собственно имя (ссылка на объект) передается в вызываемую функцию. Рассмотрим параметры изменяемого объекта:

```
def FF(lst1, lst2):
    lst1.extend(lst2)
val = [1, 2, 3]
FF(val, [4, 5, 6])
print(val)
# выведет [1, 2, 3, 4, 5, 6]
```

В примере определена функция FF, в которой значение (типа list) рассматривается за счёт присоединения значение lst2, то есть все его элементы добавляются в конец значение lst1. Значение val определяет список [1, 2, 3] и расширяется значением [4 , 5, 6]. Значение val изменится. Ссылки на объект в памяти попала в FF так как list является измененным объектом, то его можно изменить и объект в памяти уменьшится, то есть список val поменяет свое значение глобально

Область видимости

Область видимости переменных (в других языках программирования, например, Паскаль, это называется областью действия переменных) представляет собой некоторое пространство программного кода, аргументы которого используют объекты доступа и имеет определённый смысл. Это позволяет избежать контактов между одинаковыми идентификаторами, которые в рамках отдельных частей программы имеют различный смысл. С точки зрения области видимости перемены в Python могут быть локальными и глобальными. Тот или иной статус переменных определяется местом их первичной идентификации в программе.

Локальные переменные

Чтобы переменная имела локальную область видимости, достаточно поместить её в отдельный блок, который изолирован от остальной программы:

```
def F():
    x = 1 # x - локальная переменная в пределах
          #     определения функции F()
    print(x) # выведет значение 1
F()
```

В программном коде переменная x имеет локальную область видимости и доступна только в пределах функции F. Если попытаться вывести значение переменной с помощью метода print за приделом функции F, то компилятор сразу же выдаст ошибку:

```
def F():
    x = 1 # x - локальная переменная в пределах
          #     определения функции F()
    print(x) # выведет значение 1
F()
print(x)   # попытка вывода не существующей
           # переменной - ОШИБКА !!!
```

Причина подобной реакции компилятора заключается в том, что внешняя часть программного кода ничего не знает о переменной x,

поэтому использовать локальную переменную можно только в пределах блока, в котором она идентифицирована.

Несколько сложнее обстоит дело в том случае, если в программе используются две переменных с одинаковыми именами:

```
x = 0          # x - глобальная переменная
def F():
    x = 1      # x - локальная переменная
    print(x) # выведет '1'
F()
print(x)       # x - глобальная переменная
               # выведет '0'
```

В программе используются две различные перемены с одним и тем же именем x. Одна из переменных x является локальной областью видимости в пределах объявлении функции F. Другая переменная с именем x является объектом программы и область видимости включает весь текст программы за исключением функции F. Таким образом области двух переменных с одинаковым именем x не пересекаются: одна переменная является локальной, а вторая глобальной.

Глобальные переменные

Глобальная переменная доступна в любой части программы. В этом случае необходимо создать вне пределов области программного кода, ограниченного некоторым блоком (например, функцией). В нижеприведённом примере целочисленная переменная x идентифицируются как глобальная переменная:

```
x = 0          # x - глобальная переменная
def F():
    print(x) # x - глобальная переменная
F()            # выведет '0'
print(x)       # x - глобальная переменная
               # выведет '0'
```

Локальную переменную можно переименовать в глобальную. С этой целью используется ключевое слово global. Пометив переменную x как global, получаем возможность обращаться к её изначальному

значению, которое было идентифицировано вне зоны определения функции F, что демонстрирует следующая программа:

```
x = 0           # x - глобальная переменная
def F():
    global x # далее - работа с глобальной пере-
             # менной x
    x = 1
    print(x) # x - глобальная переменная
F()          # выведет '1'
print(x)     # x - глобальная переменная
             # выведет '1'
```

Вызов метода print приводит к ожидаемым результатам.

Приведём примеры, которые демонстрируют более сложные ситуации по формированию области видимости. В приведённой ниже программе содержится описание функции, внутри которой содержится другая функция F. Справа от программы показаны различные области видимости переменных с одним и тем же значением x:

```
x = 0
def F1():
    x = 1
    def F2():
        x = 2
        print(x) # напечатает 2
    F2()
    print(x)     # напечатает 1
F1()
print(x)         # напечатает 0
```

Чтобы переменной x стало глобальный для всего текста программы требуется дважды применить ключевое слово global:

```
x = 0
def F1():
    global x
    x = 1
    def F2():
```



```
        global x
        x = 2
        print(x) # напечатает 2
    F2()
    print(x)     # напечатает 2
F1()
print(x)         # напечатает 2
```

Переименование локальной переменной в глобальную с помощью global ограничивается той областью видимости, в которой эта переменная вызывается, что демонстрируется следующим примером:

```
x = 0
def F1():
    global x
    x = 1
    def F2():
        x = 2
        print(x) # напечатает 2
    F2()
    print(x)     # напечатает 1
F1()
print(x)         # напечатает 1
```

Для изменения области видимости некоторые перемены можно использовать ключевое слово nonlocal, которая объявляет перемену не локальной. Использование ключевого слово nonlocal демонстрирует следующая программа:

```
x = 0
def F1():
    x = 1
    def F2():
        nonlocal x
        x = 2
        print(x) # напечатает 2
    F2()
    print(x)     # напечатает 2
F1()
print(x)         # напечатает 0
```

Приведённые примеры демонстрируют сложность проблемы отслеживания области видимости переменных. Эта проблема становится ещё более актуальной, если аннулировать видимости из загружаемого модуля, а также использовать глобальные перемены в классе. Эти вопросы входят за рамки рассмотрения материала в настоящих методических указаний. Опыт программирования на языке Python позволяет рекомендовать на начальных этапах освоения приемов программирования придерживаться следующих простых правил:

1. Не злоупотребляйте использованием глобальных переменных.
2. Не обозначайте разные объекты программного кода одним и тем же именем.
3. Избегайте подключение библиотек с помощью команды import, а подключайте только необходимые функции в виде: from <>import <>.

В заключение данной темы покажем общую структуру программы на языке Python, которая приведена на рисунке 1.

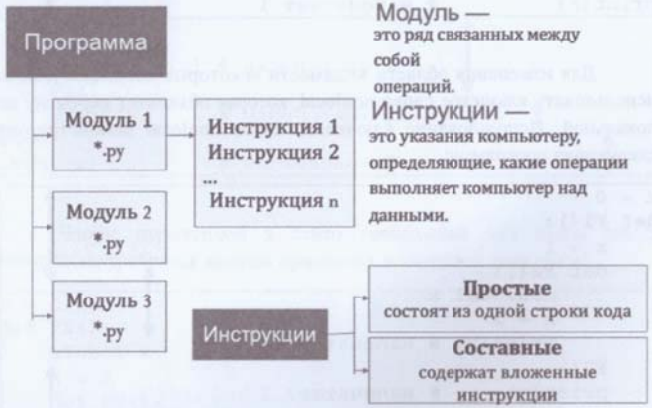


Рис.1. Структура программы на языке Python.

Пример выполнения задания с применением функций

Рассмотрим на конкретном примере процесс составления и оформления решения задачи, в процессе решения которого используются подпрограммы.

Задание: три треугольника на плоскости заданы координатами своих вершин. Определить для треугольника с минимальной площадью: располагается ли этот треугольник в некоторой области, изображенной на рис.2.

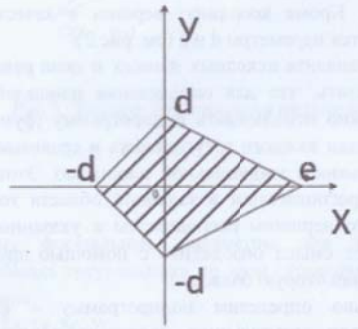


Рис.2. Заданная область на плоскости

Сформулированное задание предопределяет необходимость более подробной математической формулировки и определения используемых обозначений. Каждый из трех заданных треугольников $A_1B_1C_1$ (A_1, B_1, C_1 – вершины первого треугольника), $A_2B_2C_2$ и $A_3B_3C_3$ определены координатами своих вершин.

Координаты вершин первого треугольника:
$A_1(x_{A1}, y_{A1})$; $B_1(x_{B1}, y_{B1})$; $C_1(x_{C1}, y_{C1})$.
Координаты вершин второго треугольника:
$A_2(x_{A2}, y_{A2})$; $B_2(x_{B2}, y_{B2})$; $C_2(x_{C2}, y_{C2})$.
Координаты вершин третьего треугольника:
$A_3(x_{A3}, y_{A3})$; $B_3(x_{B3}, y_{B3})$; $C_3(x_{C3}, y_{C3})$.

Координаты вершин треугольников являются исходными данными, после задания (ввода) которых целесообразно проверить условия существования того или иного треугольника. Наиболее просто проверить корректность исходных данных можно с помощью определения для каждого из треугольников условия того, что этот треугольник не существует. Например, для треугольника $A_1B_1C_1$ это условие в виде логического выражения имеет вид:

$((x_{A1} = x_{B1})$ И $(y_{A1} = y_{B1}))$ ИЛИ $((x_{A1} = x_{C1})$ И $(y_{A1} = y_{C1}))$ ИЛИ $((x_{B1} = x_{C1})$ И $(y_{B1} = y_{C1}))$,

где И, ИЛИ – логические операции.

Аналогичные выражения можно записать для треугольников $A_2B_2C_2$ и $A_3B_3C_3$. Кроме координат вершин в качестве исходных данных используются параметры d и e (см. рис.2).

В процессе анализа исходных данных и цели решаемой задачи не трудно определить, что для определения площадей каждого из треугольников можно использовать подпрограмму (функцию). После определения площади каждого треугольника и сравнения их величин находится треугольник с минимальной площадью. Этот треугольник будет полностью располагаться в заданной области тогда и только тогда, если все его вершины расположены в указанной области. В связи с этим имеет смысл определить с помощью другой функции попадание точки в некоторую область.

Первоначально определим подпрограмму – функцию для вычисления площади треугольника, заданного координатами трех вершин. При описании функции используют формальные параметры подпрограммы, вместо которых в основной программе указываются фактические параметры.

На рис.3. показан некоторый формальный треугольник с вершинами A, B, и C, каждая из которых определяется координатами (x_A, y_A), (x_B, y_B) и (x_C, y_C).

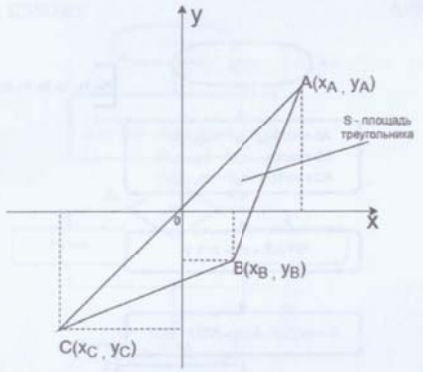


Рис.3. Задание треугольника на плоскости

Входные формальные параметры для функции, которая вычисляет площадь треугольника по трем сторонам с использованием формулы Герона:

x_A, y_A, x_B, y_B, x_C, y_C.

Выходным формальным параметром в рассматриваемом случае будет имя функции, например, AREA. Алгоритм функции AREA показан на рис.4. и имеет простейшую линейную структуру. Алгоритм подпрограммы начинается блоком «вход» и заканчивается блоком «выход» (в отличие от блоков «начало» и «конец»). Использование символов «комментарии» позволяет на схеме алгоритма фиксировать формальные входные и выходные параметры функции.

Функция AREA

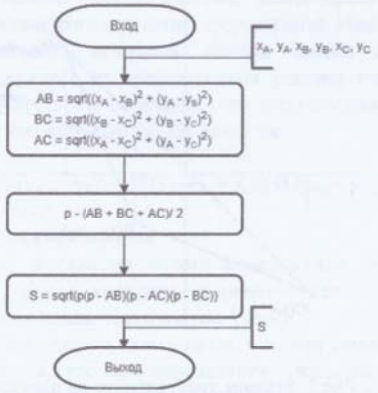


Рис.4. Алгоритм подпрограммы AREA для вычисления площади треугольника по формуле Герона

Вторая подпрограмма позволяет определить принадлежность точки с координатами (x, y) заданной области (рис.2). входными формальными параметрами этой функции будут координаты x и y, а также параметры d и e, задающие конкретный вид области, в которой может находиться точка (x, y). Схема алгоритма подпрограммы, определяющей принадлежность точки заданной области, показана на рис.5. Пусть эта подпрограмма будет иметь название ACCESSORY (англ., принадлежность). Условие принадлежности точки с координатами x и y заданной области может быть представлено в виде одновременного выполнения четырех неравенств:

$$\begin{cases} y \le x + d; \\ y \ge -x - d; \\ y \le -\frac{d}{e}x + d; \\ y \ge \frac{d}{e}x - d. \end{cases}$$

В качестве выходного параметра выберем переменную F, которая принимает значение true, если точка принадлежит заданной области.

Функция ACCESSORY

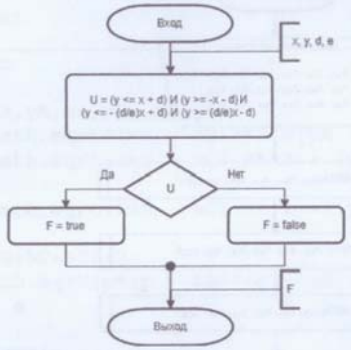


Рис.5. Схема алгоритма функции ACCESSORY определения принадлежности точки заданной области

Функция AREA и ACCESSORY реализуются не в виде отдельно транслируемых модулей, поэтому их программный код будет приведен совместно с текстом основной программы. На рис.6. показана схема алгоритма основной программы (условия существования треугольников не рассматриваются). Схема алгоритма решения поставленной задачи может иметь другую структуру. Однако приведенный алгоритм является достаточно наглядным и понятным. Кроме того, при составлении схемы алгоритма показан один из возможных недостатков, который определяется рекомендациями PEP&, а именно: неудобство использования «длинных» идентификаторов.

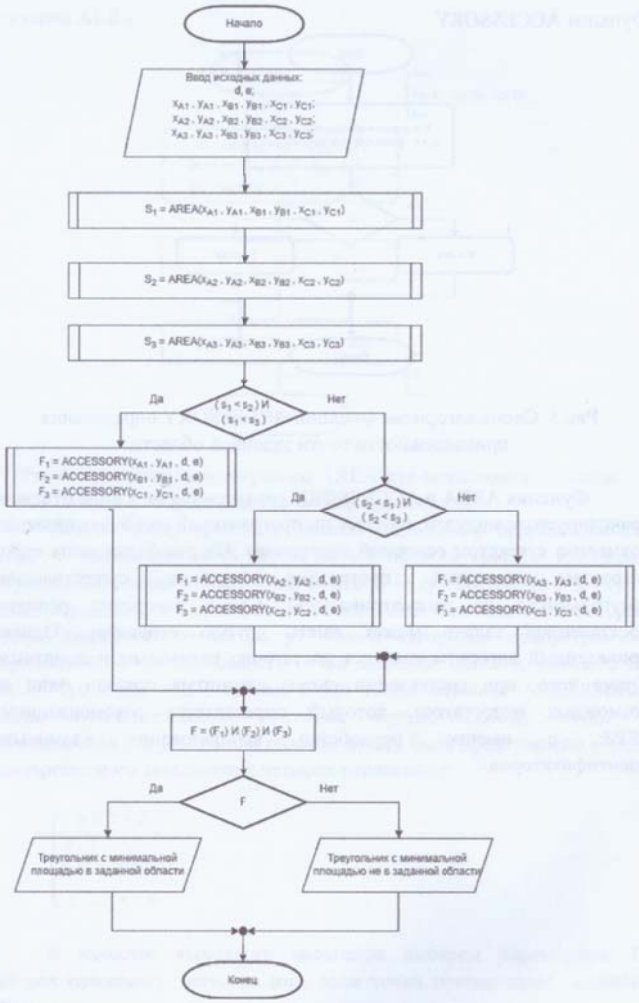


Рис.6. Схема алгоритма решения задачи

Программа на языке Python, соответствующая алгоритму на рис.6, имеет вид:

```
import math

def AREA(xA,yA,xB,yB, xC,yC):
    AB = math.sqrt((xA - xB)**2 +(yA - yB)**2)
    BC = math.sqrt((xB - xC) ** 2 + (yB - yC) **
2)
    AC = math.sqrt((xA - xC) ** 2 + (yA - yC) **
2)
    p = (AB+BC+AC)/2
    s = math.sqrt(p*(p - AB)*(p - AC)*(p - BC))
    return s

def ACCESSORY(x, y, d, e):
    if (y <= x + d) and (y>= -x - d) and (y <= -
(d/e)*x + d) and (y >= (d/e)*x -d):
        f = True
    else:
        f = False
    return f

print("Введите координату вершины A1 треугольника
A1B1C1: ")
xA1, yA1 = map(float, input().split())
print("xA1 = ",xA1,"yA1 = ", yA1)
print("Введите координату вершины B1 треугольника
A1B1C1: ")
xB1, yB1 = map(float, input().split())
print("xB1 = ",xB1,"yB1 = ", yB1)
print("Введите координату вершины C1 треугольника
A1B1C1: ")
xC1, yC1 = map(float, input().split())
print("xC1 = ",xC1,"yC1 = ", yC1)

print("Введите координату вершины A2 треугольника
A2B2C2: ")
xA2, yA2 = map(float, input().split())
print("xA2 = ",xA2,"yA2 = ", yA2)
print("Введите координату вершины B2 треугольника
A2B2C2: ")
xB2, yB2 = map(float, input().split())
```

```
print("xB2 = ",xB2,"yB2 = ", yB2)
print("Введите координату вершины C2 треугольника
A2B2C2: ")
xC2, yC2 = map(float, input().split())
print("xC2 = ",xC2,"yC2 = ", yC2)

print("Введите координату вершины A3 треугольника
A3B3C3: ")
xA3, yA3 = map(float, input().split())
print("xA3 = ",xA3,"yA3 = ", yA3)
print("Введите координату вершины B3 треугольника
A3B3C3: ")
xB3, yB3 = map(float, input().split())
print("xB3 = ",xB3,"yB3 = ", yB3)
print("Введите координату вершины C3 треугольника
A3B3C3: ")
xC3, yC3 = map(float, input().split())
print("xC3 = ",xC3,"yC3 = ", yC3)

print("Ведите e и d")
e, d = map(float, input().split())
print("e = ",e,"d = ", d)

if ((xA1 == xB1) and (yA1 == yB1)) or ((xA1 ==
xC1) and (yA1 == yC1)) or ((xC1 == xB1) and (yC1
== yB1)):
    print("Треугольник A1B1C1 не существует")
    exit(1)
if ((xA2 == xB2) and (yA2 == yB2)) or ((xA2 ==
xC2) and (yA2 == yC2)) or ((xC2 == xB2) and (yC2
== yB2)):
    print("Треугольник A2B2C2 не существует")
    exit(1)
if ((xA3 == xB3) and (yA3 == yB3)) or ((xA3 ==
xC3) and (yA3 == yC3)) or ((xC3 == xB3) and (yC3
== yB3)):
    print("Треугольник A3B3C3 не существует")
    exit(1)

s1 =  AREA(xA1, yA1, xB1, yB1, xC1, yC1)
print("Площадь треугольника A1B1C1 = ",s1)
s2 = AREA(xA2, yA2, xB2, yB2, xC2, yC2)
print("Площадь треугольника A2B2C2 = ", s2)
```



```
s3 = AREA(xA3, yA3, xB3, yB3, xC3, yC3)
print("Площадь треугольника A3B3C3 = ", s3)
if (s1 < s2) and (s1 < s3):
    f1 = ACCESSORY(xA1, yA1, d, e)
    f2 = ACCESSORY(xB1, yB1, d, e)
    f3 = ACCESSORY(xC1, yC1, d, e)
elif (s2 < s1) and (s2 < s3):
    f1 = ACCESSORY(xA2, yA2, d, e)
    f2 = ACCESSORY(xB2, yB2, d, e)
    f3 = ACCESSORY(xC2, yC2, d, e)
else:
    f1 = ACCESSORY(xA3, yA3, d, e)
    f2 = ACCESSORY(xB3, yB3, d, e)
    f3 = ACCESSORY(xC3, yC3, d, e)
if (f1 and f2 and f3):
    print("Треугольник с минимальной площадью в
области")
else:
    print("Треугольник с минимальной площадью не
в области")
```

Контрольные вопросы

1. Что такое подпрограмма?
2. Чем функция отличается от процедуры?
3. Каким образом определяется функция?
4. В чем различие функции и процедуры?
5. Что такое lambda-выражение?
6. Каким образом определяются функции с переменным числом аргументов?
7. Какие параметры функции считаются необязательными?
8. Что такое область видимости переменной?
9. Какая переменная является локальный?
10. Какая перемена является глобальный?
11. Как выглядит структуру программы на языке Python?

Задания

Вариант 1. Составить подпрограмму для определения расстояние между точками A и B в n-мерном пространстве по формуле

$$\propto = \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$$

где a, b – координаты точек A и B. Используя её, найдите минимальное расстояние между точками X, Y, Z.

Вариант 2. Составьте подпрограмму вычисления среднего арифметического элементов вектора. Используя её, преобразуйте квадратную матрицу следующим образом: диагональные элементы матрицы заменить средними арифметическими значениями элементов соответствующих строк.

Вариант 3. Составьте подпрограмму умножение двух матриц произвольный размерности. Используя её, вычислить k-ю степень квадратной матрицы.

Вариант 4. Составьте подпрограммы определение максимального и минимального элементов в одномерном массиве. Используя их, найдите минимум среди максимальных элементов строк матрицы.

Вариант 5. Составьте подпрограмму определения баланса открывающих и закрывающих скобок в выражении. Используя её, составьте программу контроля правильности записи вводимых выражений.

Вариант 6. Составить подпрограмму вычисления нормы матрицы $A=[a_{ij}]$, i,j=1,n по формуле

$$P = \max_{1 \le i \le n}\left\{\sum_{j=1}^{n}|a_{ij}|\right\}$$

Используя её, определить матрицу из X, Y, Z с минимальной нормой.

Вариант 7. Составьте требуемые подпрограммы работы с комплексными числами (сложение, вычитание, умножение и деление комплексных чисел). Используя эти подпрограммы, определите действительную и мнимую части числа.

$$\omega = \frac{z_1^3 + 1}{z_1 z_2^2 - 1}$$

Вариант 8. Составьте подпрограммы определение максимального и минимального элемента в одномерном массиве. Используя их, найдите седловую точку матрицы $A=[a_{ij}]$, i,j=1,n. Матрица имеет седловую точку если элемент a_{ij} является минимальным в k-й строке и максимальным в l-м столбце, и наоборот.

Вариант 9. Составьте подпрограмму определение суммы элементов одномерного массива. Используя её, вычислите сумму элементов матрицы.

Вариант 10. Составьте подпрограмму, определяющие количество и номер позиции, в которых встречается в тексте заданный символ. Используя её, определите эти характеристики для символов ‘Т’,’О’,’У’, текста: ‘по ту сторону добра и зла’.

Вариант 11. Составьте подпрограмму вычисления значения полинома n-го порядка по схеме Горнера. Используя её, вычислите значение функции

$$z = \frac{12x^5 - 3x^3 + 2x^2 - 1}{6x^6 - 5x^4 + x - 8}$$

для значений x, вводимых с терминала.

Вариант 12. Составьте подпрограмму, определяющую число заданных сдвоенных символов в тексте. Используя её, почитайте количество с сдвоенных символов ‘сс’,’ее’,’нн’,’лл’, текста: ‘класс, рассеянность, естественность, веер, аллегро’.

Вариант 13. Составьте подпрограмму сортировки по убыванию значений элементов одномерного массива. Используя её, отсортируйте элементы в каждом столбце матрицы.

Вариант 14. Найдите сумму элементов вспомогательных диагоналей квадратных матриц: A, B = λA, C = λB + μA где, λ,μ – скалярные переменные (используйте подпрограмму).

Вариант 15. Найдите сумму элементов, расположенных выше главной диагонали, для матрицы: X, Y, Z =X^2 + Y^2 (используйте подпрограмму).

Вариант 16. Составьте подпрограмму сортировки по возрастанию значений элементов одномерного массива. Используя её, отсортируйте элементы в каждой строке матрицы.

Вариант 17. Используя подпрограммы ввода и вывода массива, вычислите и выведите значение следующих матриц: $Z = A^T + B$, $Y = B^T + A$, где A, B – исходные матрицы. Все матрицы квадратные и одинаковой размерности.

Вариант 18. Опишите подпрограмму идентификации символа: буква или цифра. Используя её, составьте программу определения, является ли вводимая последовательность символов идентификатором.

Вариант 19. Опишите подпрограмму вычисления следа матрицы – суммы диагональных элементов. Определите матрицу X, Y, Z с максимальным следом.

Вариант 20. Опишите подпрограмму определения суммы цифр заданного целого числа. Определите указанные суммы для всех трёхзначных целых чисел, лежащих в заданном интервале.

Вариант 21. Составьте подпрограмму определения расстояние между двумя точками, заданными своими координатами. Используя подпрограмму, найдите площадь треугольника (по формуле Герона), заданного координатами своих вершин.

Вариант 22. Составьте подпрограмму определения среднего значения двухмерного массива. Используя подпрограмму, определите для какой из матрицы A, B, C (с произвольным числом строк и столбцов) среднее значение наибольшее.

Вариант 23. Составьте подпрограмму вычисления определённого интеграла по формуле трапеций. Используя под

программу найдите значение определённого интеграла с подынтегральной функцией

$$y = \frac{\sin x + \cos x}{2 + \sin^2 x} \text{ и } y = \frac{e^x + e^{-x}}{3 + \cos x}$$

Вариант 24. Составьте подпрограмму уточнения корня алгебраического уравнения методом половинного деления. Используя подпрограмму, уточните корень уравнения (формулу уравнения взять из предыдущих заданий).

Вариант 25. Составьте подпрограмму определения расстояние между двумя точками, которые заданными своими координатами. Используя подпрограмму, оцените длину кривой, заданную в некоторых точках, первая координата которых изменяются от значений x_0 до x_n с постоянным шагом h_x.

Вариант 26. Уравнения прямых заданы в форме:

$a_1 x - b_1 y + c_1 = 0$;
$a_2 x - b_2 y + c_2 = 0$;
Координаты точки пересечения этих прямых определяется формулами:

$$\begin{cases} x = (b_1 c_2 - b_2 c_1)/(b_2 a_1 - b_1 a_2) \\ y = (a_1 c_1 - a_1 c_2)/(a_1 b_2 - a_1 b_1) \end{cases}$$

Условие параллельности прямых:

$a_1 b_2 = a_2 b_1$

Составить подпрограмму определение точки пересечения двух прямых. Используя подпрограмму, определить точки пересечения трёх произвольно заданных прямых.

Вариант 27. Вариант 26 определяет подпрограмму для нахождения точки пересечения двух прямых. Прямые будут перпендикулярными, если выполняться условие:

$a_1 a_2 = b_1 b_2$ или $a_1 a_2 + b_1 b_2 = 0$

Используя подпрограмму (из варианта 26) и условие перпендикулярности, определить площадь прямоугольного

треугольника. Если треугольник не прямоугольный, то вывести соответствующее сообщение.

Вариант 28. Составьте подпрограмму вычисления площади круга. Используя подпрограмму, определить площадь кольца, если центры внутренней и внешней окружности кольца находятся в одной точке.

Вариант 29. Составьте программу вычисления двух корней квадратного уравнения (корни различны, корни равны, нет корней). Используя под программу найдите решения 3-х квадратных уравнений.

Вариант 30. Кубическое уравнение $ax^3 + bx^2 + cx + d = 0$ после деления на a имеет конический вид:

$x^3 + rx^2 + sx + t = 0$, где $r = b/a$; $s = c/a$; $t = d/a$.

Приведённое кубическое уравнение имеет вид (после замены $x = y - r/3$):

$y^3 + py + q = 0$, где $p = (3s - r^2)/3$; $1 = 2r^3/27 - rs/3 + t$.

Приведённое уравнение решается по формуле Кардано. Составьте подпрограмму, в которой определяются коэффициенты приведённого уравнения используйте подпрограмму для приведения 3-х произвольных кубических уравнений.

Оглавление