ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Python. Ветвление. Многоальтернативное ветвление

Методические указания к лабораторным работам

Рязань 2017

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Python. Ветвление. Многоальтернативное ветвление

Лабораторные работы №3,4

Методические указания к лабораторным работам

Рязань 2017

УДК 004.432

Python. Ветвление. Многоальтернативное ветвление: методические указания к лабораторным работам. / Рязан. гос. радиотехн. универ.; Сост.: А.Н. Пылькин, Н.Н. Степанов, Н.А. Тярт. – Рязань, 2016 г.

Рассмотрены понятия разветвляющегося алгоритма и его разновидностей. Приведены синтаксис и описание действия операторов ветвления if-else и if-elif-else. Рассмотрены логические операции. Дан синтаксис тернарного оператора. Рассмотрены примеры применения операторов if-else и if-elif-else.

В качестве практических заданий предлагается составить две программы (по одной на работу), связанные с осуществлением различных действий в зависимости от выполнения условия.

Табл.: 2. Ил.: 14. Библиогр.: 4 назв.

Печатается по решению Научно-методического совета Рязанского государственного радиотехнического университета.

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.).

Лабораторная работа №3
Оператор условного перехода

Цель работы

Освоение технологии проектирования алгоритма и программы разветвляющейся структуры с использованием оператора условного перехода на языке Python.

Разветвляющийся алгоритм и условный оператор

Условный оператор (оператор условного перехода) реализует программирование структуры простого (дихотомического) ветвления (см. рис. 1) и позволяет выбрать и выполнить один из двух входящих в него наборов команд в зависимости от значения (выполнения или невыполнения) условия.

Общий вид оператора

```
if <условие>:
    <команда>
    ...
    <команда>
else:
    <команда>
    ...
    <команда>
```

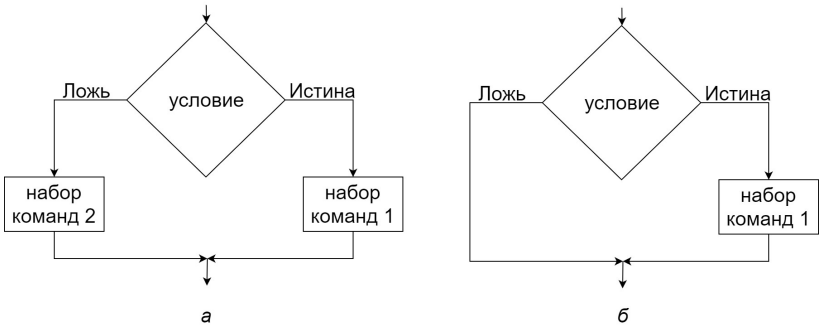


Рис. 1. Схемы условного оператора

Порядок выполнения условного оператора поясняется рис. 1, а. Если <условие> выполняется, т.е. принимает значение Истина (`True`), то выполняется <набор команд 1>, если же оно принимает значение Ложь (`False`), то выполняется <набор команд 2>. В любом случае далее выполняется оператор, стоящий в программе непосредственно за условным оператором.

Например, оператор

```
if x < y:
    x = x + 1
else:
    y = y + 1
```

позволяет увеличивать на единицу меньшее из двух значений *x* или *y* и может быть «прочитан» следующим образом: «ЕСЛИ x меньше *y*, ТО *x* увеличить на единицу, ИНАЧЕ (т.е. когда *y* меньше или равен *x*) увеличить на единицу *y*».

Условный оператор может не иметь конструкции `else`, тогда он называется сокращенным условным оператором (сокращенным ветвлением). В этом случае, если условие принимает значение Ложь (`False`), то сразу выполняется оператор, следующий за условным (рис. 1, б).

Например, оператор

```
if x < 0:
    x = -x
```

обеспечивает инвертированное значение переменной *x*, если оно отрицательно, и оставляет его без изменения в противном случае.

В общем виде сокращенный условный оператор имеет вид

```
if <условие>:
    <команда>
    ...
    <команда>
```

Набор команд, расположенных строго друг под другом, образует последовательность, которую в программировании называют блоком.

Вложенность блоков программного кода в Python достигается отступами; у набора команд одного уровня вложенности отступы должны быть одинаковой величины. Общепринятым стандартом

является отступ в 4 пробела (в среде PyCharm именно такой отступ вставляется при нажатии клавиши *Tab*).

На месте условия в операторе обычно стоит логическое выражение, которое может принимать значения `True` и `False`. Однако в реальности на этом месте может стоять любая константа, выражение (логическое или арифметическое) или объект. За истину принимается не только значение `True`, но также любое числовое значение, отличное от 0 (в том числе отрицательное), или любой непустой объект. За ложь, таким образом, принимаются значения `False`, 0 и `None` (отсутствие объекта).

Примером использования условного оператора может послужить следующая маленькая программа:

```
# Пользователь вводит значение
res = eval(input("Введите что-нибудь: "))
# Используем условный оператор для проверки
# типа введенного пользователем значения
if type(res) == int :  # Если целое число
    print("Вы ввели целое число!")
else:  # Если что-то другое
    print("Это точно не целое число!")
# После выполнения условного оператора
print("Работа завершена!")
```

Необходимо пояснить некоторые особенности работы данной программы.

Во-первых, здесь представлен один из способов ввода числовых значений. В отличие от приведения типов, использование функции `eval()` для обработки результатов ввода более универсально, т.к. результатом может быть как целое число типа `int` (если ввести, например, 10), так и вещественное типа `float` (если ввести 10.0). Более того, если пользователь введет арифметическое выражение, его результат будет посчитан и возвращен функцией `eval()`. Однако в таком случае строковые значения необходимо вводить в кавычках, иначе функцией `eval()` будет возвращена ошибка.

Во-вторых, в качестве условия здесь стоит логическое выражение `type(res)==int`. Оно будет истинным, если в переменной `res` хранится значение типа `int`. Здесь функция `type()` возвращает тип хранимого значения, а логический оператор `==` производит сравнение на равенство.

Важно: не путайте оператор логического равенства «==» с оператором присваивания «=»! Если в проверяемом условии вы по ошибке напишете оператор присваивания, то получите синтаксическую ошибку и программа не будет запущена. (А в языках вроде C++ вместо ошибки компиляции вы бы и вовсе получили неверно работающее условие с труднонаходимой ошибкой в коде).

Пример 1. Даны три числа *a*, *b* и *c*. Вычислить и вывести на экран значение *z*, равное квадрату большего из них.

Схема алгоритма решения поставленной задачи показана на рис. 2.

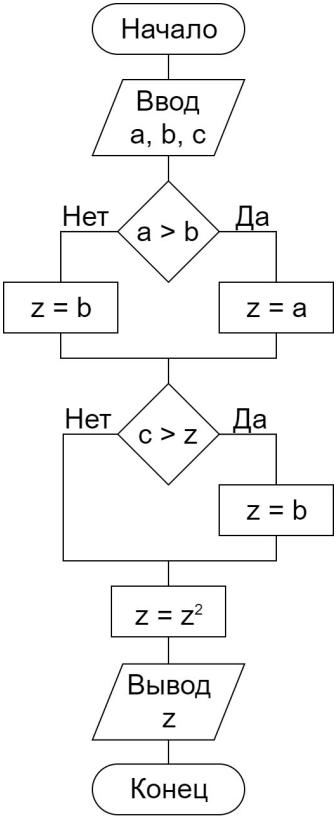


Рис. 2. Схема алгоритма нахождения квадрата максимального значения

В общем случае алгоритм состоит из трех шагов: ввод исходных данных, обработка данных, вывод полученных результатов. Обработка данных состоит из двух более простых шагов: поиска максимального из трех чисел и возведения найденного значения в квадрат. Поиск максимального значения осуществляется сравнением сначала двух значений *a* и *b* и выбора наибольшего из них, а затем сравнением полученного значения с третьим числом *c*. В итоге в качестве максимального выбирается число *c*, если оно окажется больше любого из двух первых чисел. Для реализации первого ветвления используется условный оператор в полной форме записи, а для второго – в сокращенной форме. Программа, соответствующая схеме алгоритма (рис. 2), имеет следующий вид:

```
print "Введите a, b, c"
a = input()
b = input()
c = input()
print "Вы ввели:"
print "a =", a
print "b =", b
print "c =", c

if a > b:
    z = a
else:
    z = b

if c > z:
    z = c

z = z ** 2
print "z =", z
```

Логические выражения

При решении практических задач ветвления могут иметь более сложную структуру и проверяемые условия (в том числе логические выражения) могут быть сколь угодно сложными. Чаще всего в качестве условия в операторе `if` используются операции типа сравнения

(отношения), в результате которых образуется логический тип со значениями `True` (Истина) или `False` (Ложь). Логические данные объединяются в сложные логические выражения (условия) с помощью логических операций `not` (НЕ), `or` (ИЛИ), а также `and` (И).

В таблице 1 приведены операции сравнения, доступные в языке Python.

Таблица 1. Операции сравнения

Операция сравнения	**Описание**	**Примеры**
`==`	Проверяет, равны ли оба операнда. Если да, то условие становится истинным.	`5 == 5` в результате будет `True`. `True==False` в результате будет `False`. `"hello" == "hello"` в результате будет `True`.
`!=`	Проверяет, равны ли оба операнда. Если нет, то условие становится истинным.	`12 != 5` в результате будет `True`. `False != False` в результате будет `False`. `"hi" != "Hi"` в результате будет `True`.
`<>`	Проверяет, равны ли оба операнда. Если нет, то условие становится истинным.	`12 <> 5` в результате будет `True`. Похоже на оператор `!=`
`>`	Проверяет, больше ли значение левого операнда, чем значение правого. Если да, то условие становится истинным.	`5 > 2` в результате будет `True`. `True > False` в результате будет `True`. `"A" > "B"` в результате будет `False`.
`<`	Проверяет, меньше ли значение левого операнда, чем значение правого. Если да, то условие становится истинным.	`3 < 5` в результате будет `True`. `True < False` в результате будет `False`. `"A" < "B"` в результате будет `True`.

Продолжение таблицы 1

`>=`	Проверяет, больше или равно значение левого операнда, чем значение правого. Если да, то условие становится истинным.	`1 >= 1` в результате будет `True`. `23 >= 3.2` в результате будет `True`. `"C" >= "D"` в результате будет `False`.
`<=`	Проверяет, меньше или равно значение левого операнда, чем значение правого. Если да, то условие становится истинным.	`4 <= 5` в результате будет `True`. `0 <= 0.0` в результате будет `True`. `-0.001 <= -36` в результате будет `False`.

Операции сравнения (операции отношения) применимы к данным (операндам) любого типа. В математической логике такие операции/функции называют предикатами.

Предикат – функция, которая проверяет, обладает ли объект/объекты заданным свойством. В языках программирования предикаты отношения (проверяют, состоят ли объекты в заданном отношении) реализуются как операции. Помимо отношений в языках обязательно есть и другие функции-предикаты, позволяющие проверять различные свойства и идентифицировать объекты. В программах такие операции/функции позволяют задавать и проверять простые условия. Для задания и проверки более сложных условий используют логические операции.

Логические операции применимы *только* к операндам (константам, переменным, функциям) логического типа и возвращают логическое значение. В языке определены следующие операции:

- `not` – логическое отрицание (инверсия, операция НЕ);
- `and` – логическое умножение (конъюнкция, операция И);
- `or` – логическое сложение (дизъюнкция, операция ИЛИ).

Логические операции и операции отношения нередко встречаются в одном логическом выражении, при этом *отношения имеют меньший приоритет, поэтому их необходимо заключать в скобки*. Например, если необходимо записать выражение, принимающее значение True, когда действительная переменная x

принадлежит к одному из двух отрезков [*a,b*] или [*c,d*], его можно представить в следующем виде:

(*x* >= *a*) **and** (*x* <= *b*) **or** (*x* >= *c*) **and** (*x* <= *d*)

Таблица 2. Таблица истинности логических операций

x	*y*	**not** *x*	*x* **and** *y*	*x* **or** *y*
False	False	True	False	False
False	True	True	False	True
True	False	False	False	True
True	True	False	True	True

Напоминаем, что операции сравнения обладают бóльшим приоритетом выполнения, нежели логические операции (см. таблицу приоритетов в лабораторной работе №2). Однако приоритет выполнения можно изменить правильной расстановкой скобок. Также скобки можно ставить не только для изменения приоритета выполнения, но и для повышения читаемости и наглядности кода. То есть ими можно явно обозначать, какие операции будут выполнены в первую очередь.

Структура ветвления может входить составной частью в одну из ветвей другой структуры ветвления. В этом случае имеет место сложное *вложенное ветвление*. Примером задачи, приводящей к сложному ветвлению, может служить следующее задание.

Пример 2. Даны две переменные *x* и *y*. Если *x* = *y*, то вывести на печать значения этих переменных без изменения. Если *x* > *y*, значения переменных уменьшить в два раза. Если же *x* < *y*, то значения переменных увеличить на 10.

Схема алгоритма решения поставленной задачи показана на рис. 3. В одной из ветвей внешнего ветвления содержится еще один условный оператор, то есть внутреннее ветвление. Далее приведена программа, реализующая данный алгоритм.

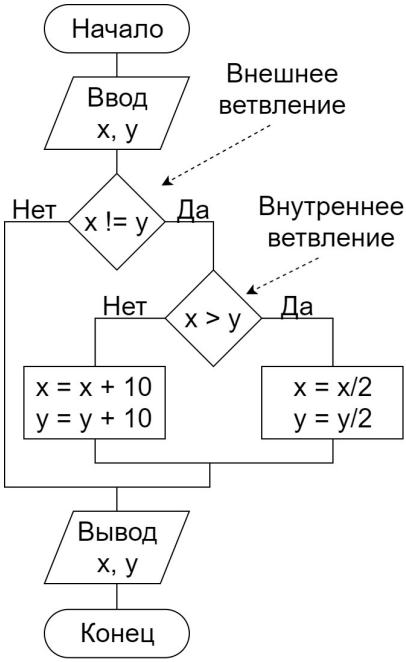


Рис. 3. Схема вложенного ветвления

```
print "Введите x, y"
x = input()
y = input()
print "Вы ввели"
print "x =", x
print "y =", y

if x != y:
    if x > y:
        x += 10
        y += 10
    else:
        x /= 2
        y /= 2

print "x =", x
print "y =", y
```

Пример 3. Вычислить значение функции, график которой представлен на рис. 4.

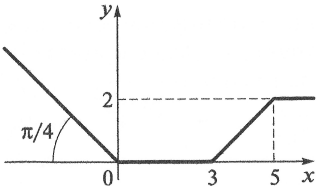


Рис. 4. График функции

Для решения данной задачи необходимо прежде всего записать аналитическое выражение функции. В соответствии с графиком область определения функции разбивается на 4 участка:

$$y = \begin{cases} -x, & x \le 0; \\ 0, & 0 < x \le 3; \\ x - 3, & 3 < x \le 5; \\ 2, & x > 5. \end{cases}$$

Для построения схемы алгоритма используем вложенную конструкцию операторов ветвления (рис. 5).

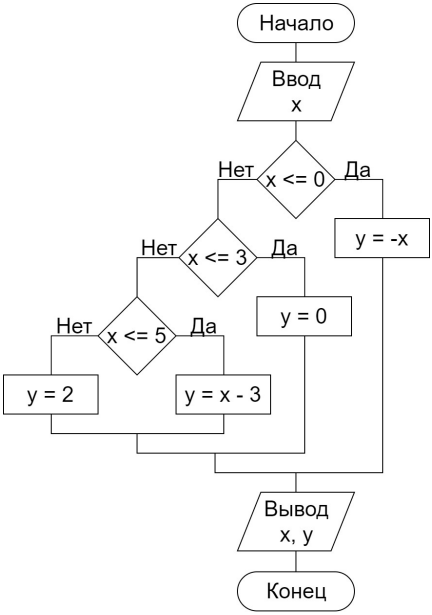


Рис. 5. Алгоритм вычисления значения функции по графику

Проверяем условия последовательно. Первым проверим условие $x <= 0$. Следующее условие $0 < x <= 3$ будет проверяться только в том случае, если первое не соблюдается и $x > 0$. Следовательно, часть второго условия $0 < x$ можно не проверять, если дело дошло до проверки этого условия, то $0 < x$. Аналогично исключается проверка $3 < x$, а также проверка последнего условия $x > 5$. Текст программы будет иметь следующий вид:

```
print "Введите x"
x = input()

if x <= 0:
    y = -x
else:
    if x <= 3:
        y = 0
    else:
        if x <= 5:
            y = x - 3
        else:
            y = 2
print "При x =", x, " y =", y
```

Пример 4. Определить, принадлежит ли точка M(x,y) закрашенной области (рис. 6).

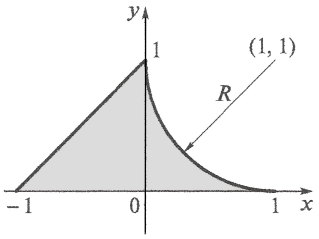


Рис. 6. Область определения

Для решения данной задачи необходимо определить, какие значения переменных x и y удовлетворяют граничным условиям заданной области. Поскольку она расположена в первом и втором квадрантах, то обязательным условием принадлежности точки с

координатами (x,y) изображенной на рисунке области является истинность отношения $y > 0$, что позволяет выбрать верхнюю полуплоскость (рис. 7, а).

Уравнение окружности радиусом R с координатами центра (a,b) задается выражением

$$(x - a)^2 + (y - b)^2 = R^2$$

Координаты точки, находящейся вне круга (рис. 7, б), ограниченного окружностью радиусом $R = 1$ с центром в точке с координатами $a = 1$ и $b = 1$, должны удовлетворять неравенству

$$(x - 1)^2 + (y - 1)^2 > 1$$

Однако при этом в выделенную область будет входить все пространство верхней полуплоскости, находящейся вне круга. Нам же необходимо ограничиться областью, прилегающей к центру координат. Поэтому следует наложить еще одно ограничение $x < 1$ (рис. 7, в). Чтобы точка попала в область, расположенную ниже прямой, определяемой уравнением $y = x + 1$, необходимо выполнение неравенства $y < x + 1$ (рис. 7, г).

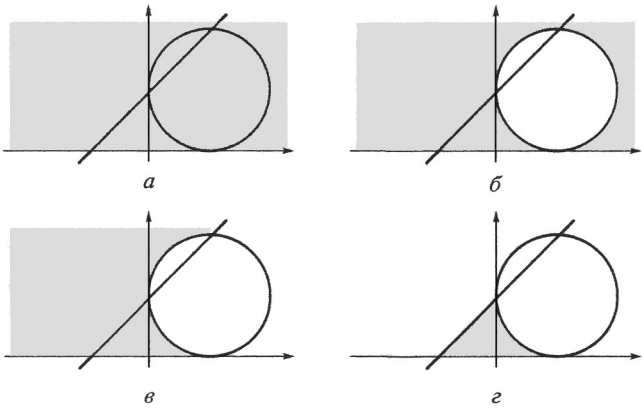


Рис. 7. Процесс отделения области

Окончательное условие попадания точки с координатами (x,y) в заданную область будет определяться одновременным выполнением совокупности следующих неравенств:

$$y > 0;\ (x - 1)^2 + (y - 1)^2 > 1;\ x < 1;\ y < x + 1.$$

Алгоритм может быть представлен простым ветвлением, в котором проверяется найденное условие попадания в область.

Программная реализация решения поставленной задачи будет иметь следующий вид:

```
print "Введите координаты точки ( x ; y )"
x = input()
y = input()

if y > 0 and (x - 1) ** 2 + (y - 1) ** 2 > 1 \
and x < 1 and y < x + 1:
    print "Точка (", x, ";", y, ") попадает в область"
else:
    print "Точка (", x, ";", y, ") не попадает в
область"
```

Тернарный оператор

Набор команд вида:

```
if X:
    A = Y
else:
    A = Z
```

достаточно прост, однако все равно занимает целых 4 строки. Для таких случаев в языке присутствует тернарный оператор в виде конструкции следующего вида:

```
A = Y if X else Z
```

Результаты выполнения двух команд, представленных выше, будут абсолютно одинаковыми. Поэтому тернарный оператор позволяет эффективно сокращать запись малых ветвлений. Однако не пытайтесь записать в виде тернарного оператора ветвление, содержащее много вложенных действий – это ухудшит читаемость кода.

Пример выполнения задания

Задание. Для вводимых размеров кирпича *a, b, c* и размеров прямоугольного отверстия *x, y* определить, пройдет ли кирпич сквозь отверстие.

Схема алгоритма программы (рис. 8):

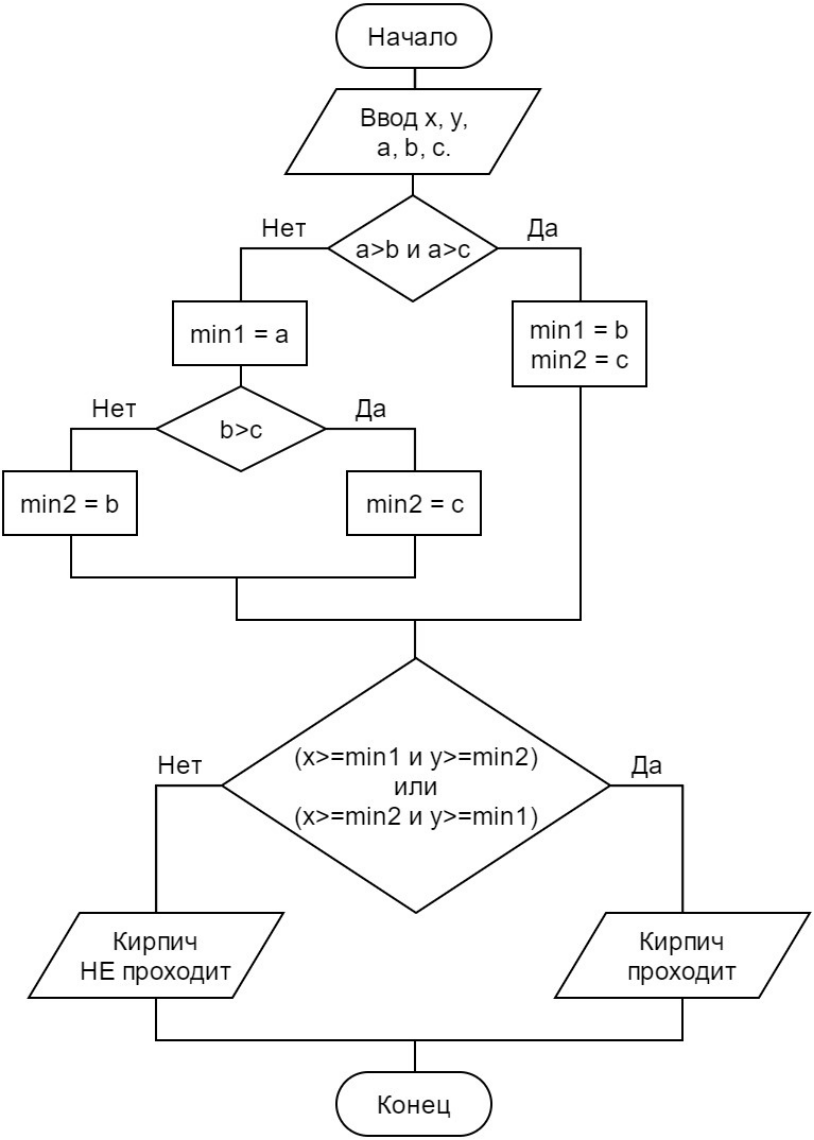


Рис. 8. Алгоритм определения соответствия размеров кирпича и прямоугольного отверстия

Сначала мы сравниваем между собой размеры кирпича *a, b* и *c* и выбираем из них два наименьших. Делается это от обратного – поиском наибольшего размера (его нам нужно отбросить, а два оставшихся - сохранить). Затем два наименьших размера кирпича сравниваются с двумя размерами отверстия *x* и *y* (возможны два варианта того, как просунуть кирпич в отверстие) и выводится результат.

Текст программы:

```
x = float(input("Введите x: "))
y = float(input("Введите y: "))
a = float(input("Введите a: "))
b = float(input("Введите b: "))
c = float(input("Введите c: "))

if (a > b) and (a > c):
    min1 = b
    min2 = c
else:
    min1 = a
    if (b > c): min2 = c
    else: min2 = b
if ((x >= min1) and (y >= min2)) \
   or ((x >= min2) and (y >= min1)):
    print("Кирпич размеров ", a, "*", b, "*", c, " \
          проходит в отверстие размеров ", x, "*", y)
else:
    print("Кирпич размеров ", a, "*", b, "*", c, " НЕ \
          проходит в отверстие размеров ", x, "*", y)
```

Результаты выполнения:

```
Введите x: 10
Введите y: 20
Введите a: 1
Введите b: 30
Введите c: 15
Кирпич размеров  1.0 * 30.0 * 15.0  проходит в
отверстие размеров  10.0 * 20.0
```

Контрольные вопросы

1. Что такое ветвление и как оно представляется на схеме?
2. В чем суть усеченного ветвления?
3. Каков синтаксис оператора if (для полного и усеченного ветвления)?
4. Каков синтаксис тернарного оператора?
5. Какие логические операции вы знаете?

Задания

Для задания в соответствии со своим вариантом составить алгоритм и написать программу, реализующую выполнение определенных действий в зависимости от выполнения заданного условия. Оформить отчет, содержащий текст варианта задания, схему алгоритма, код программы и результаты выполнения.

Варианты задания:

1. Вычислить значение функции *f(x)* для вводимого значения *x*:

$$f(x) = \begin{cases} 2x + 4, & -2 \le x \le -1; \\ 2x^2, & -1 < x \le 1; \\ -2x + 2, & 1 < x \le 2. \end{cases}$$

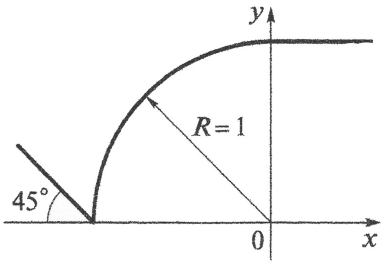
При $x < -2$ и $x > 2$ функция *f(x)* не определена.

2. Вычислить значение функции *f(x)* для вводимого значения *x*:

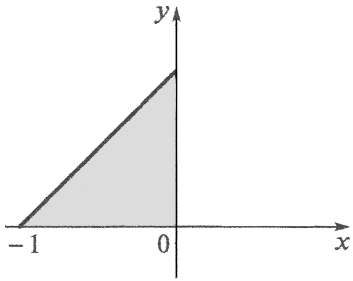
$$f(x) = \begin{cases} 1, & x \le -1; \\ 2x^2 - 1, & -1 < x \le 1; \\ x^2, & 1 < x \le 2. \end{cases}$$

При $x > 2$ функция *f(x)* не определена.

3. Определить максимальное значение из заданных x_1, x_2, x_3, x_4.
4. Заданы длины трех отрезков x_1, x_2 и x_3. Разработать алгоритм и программу, которая по результатам анализа вводимых длин отрезков выводят на экран дисплея одно из следующих сообщений: «треугольник построить нельзя», «разносторонний треугольник», «равнобедренный треугольник», «равносторонний треугольник».
5. Составить программу вычисления значения функции, заданной графиком, при произвольном значении *x*.



6. Определить, принадлежит ли точка M(*x,y*) выделенной области.



7. Заданы следующие параметры геометрических фигур: *x, y, z* – стороны треугольника; *a* – стороны квадрата; *r* – радиус круга. Вывести на экран дисплея наименование и числовое значение площади фигуры с максимальной площадью.

8. Для отрезков *a, b* и *c* определить, можно ли из них построить треугольник и является ли этот треугольник прямоугольным (*a, b, c* – целые числа).

9. Вычислить значение функции *f(x)* для вводимого значения *x*:

$$f(x) = \begin{cases} 2sin(\frac{3x}{4}) & x \le \frac{\pi}{2}; \\ \frac{x}{2}tg(\frac{x+1}{3}) & x > \frac{\pi}{2}. \end{cases}$$

В тех случаях, когда тангенс не имеет значений, вывести сообщение «функция не существует».

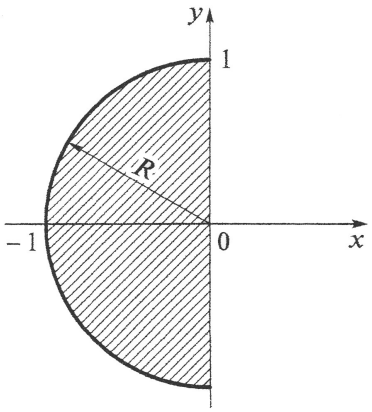
10. Дана функция $f(x) = \sqrt{x}$. Проверьте, что для любых произвольно выбранных аргументов $x_1 > 0$ и $x_2 > 0$ имеет место неравенство $f\left(\frac{x_1+x_2}{2}\right) \ge \frac{f(x_1)+f(x_2)}{2}$.

11. Дана функция $f(x) = \frac{x}{4x^2+9x}$. Найти значение функции при произвольно заданных значениях аргумента x_1, x_2, и x_3. На экран дисплея вывести минимальное значение функции.

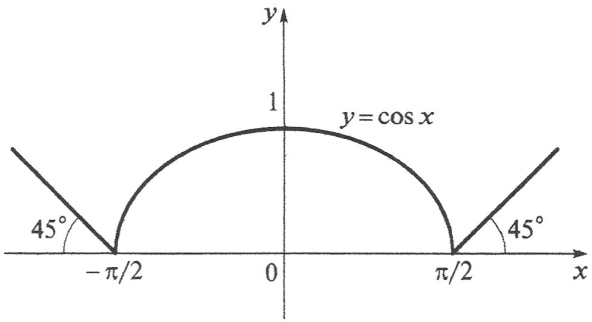
12. Вычислить значение функции *f(x)* для вводимого значения *x*:

$$f(x) = \begin{cases} \frac{8}{x}, & x \le -2; \\ x^3 + 4, & -2 < x \le 0; \\ \frac{4}{x^2+1}, & x > 0. \end{cases}$$

13. Определить, принадлежит ли точка M(*x,y*) заштрихованной области.



14. Составить программу вычисления значения функции, заданной графиком, при произвольном значении *x*.



15. Определите знак чисел *a* и *b*:

$$a = sin\left(\frac{5\pi}{6}\right) cos\left(\frac{5\pi}{7}\right) tg\left(\frac{5\pi}{8}\right) ctg\left(\frac{5\pi}{9}\right);$$

$$b = sin\left(\frac{4\pi}{7}\right) cos\left(\frac{-4\pi}{9}\right) tg\left(\frac{4\pi}{9}\right) ctg\left(\frac{-4\pi}{11}\right).$$

16. Определить минимальное значение из заданных x_1, x_2, x_3, x_4.

17. Вычислить значение функции *f(x)* для вводимого значения *x*:

$$f(x) = \begin{cases} 2sin\left(\frac{3x}{4}\right), & x \le \frac{\pi}{2}; \\ \frac{x}{2} tg\left(\frac{x+1}{3}\right), & x > \frac{\pi}{2}. \end{cases}$$

При $x < 1$ функция *f(x)* не определена.

18. Заданы стороны двух треугольников: (a_1, b_1, c_1) и (a_2, b_2, c_2). Определить треугольник с максимальной площадью.

19. Вычислить значение функции *f(x)* для вводимого значения *x*:

$$f(x) = \begin{cases} 3x + \sqrt{|x+1|}, & x < -2; \\ \dfrac{\sin(x) + \cos(x)}{2x+1}, & -2 \le x \le -1; \\ e^{-3x^2+2x-1}, & 1 < x \le 1. \end{cases}$$

При $x > 1$ функция *f(x)* не определена.

20. Составить алгоритм и программу, которая производит анализ дискриминанта квадратного трехчлена ax^2+bx+c и выводит на экран дисплея одно из следующих сообщений: «корней нет»; «корни одинаковые»; «корни разные».

21. Вычислить значение функции *f(x)* для вводимого значения *x*:

$$f(x) = \begin{cases} \dfrac{(x+5)(x-6)}{(x-2)(x+3)}, & x > 1.75; \\ \dfrac{x^2+2x+4}{x^2-2x+1}, & x \le 1.75. \end{cases}$$

Точки разрыва исключить.

22. Даны значения *a, b, c*. Найти среднее арифметическое этих значений и вывести переменную и ее значение с наименьшим отклонением от среднего арифметического.

23. Даны три числа *a, b, c*, которые являются сторонами треугольника. Определить вид треугольника: прямоугольный, остроугольный, тупоугольный.

Лабораторная работа №4
Многоальтернативное ветвление

Цель работы

Изучение оператора `if-elif-else` для организации многоальтернативного ветвления.

Многоальтернативное ветвление

Проверка некоторого количества условий по очереди (обособленно) в большинстве случаев является неэффективным и неудобным решением. Для решения проблем, связанных со сложным выбором в зависимости от нескольких условий, используют вложенное ветвление.

Ниже (рис. 1) представлена схема интересующего нас в данный момент варианта вложенного ветвления, где каждое внутреннее ветвление содержится в ложной ветви внешнего ветвления:

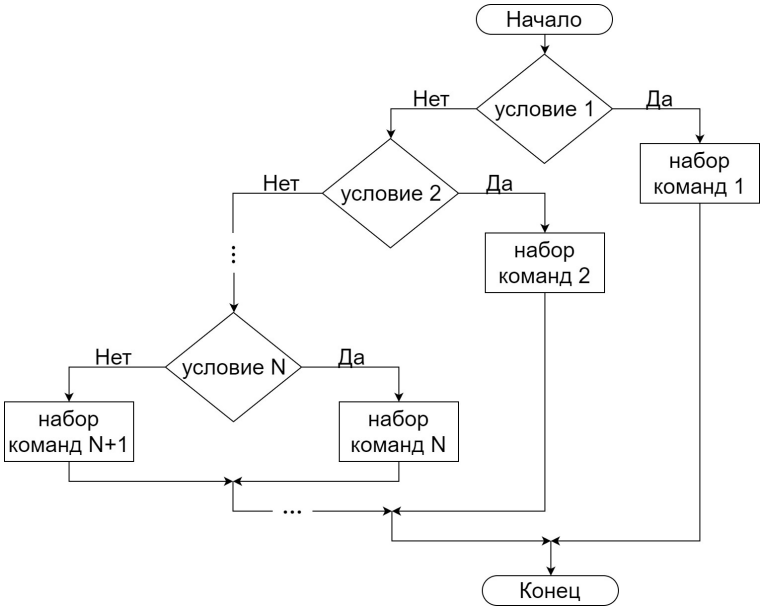


Рис. 1. Схема алгоритма многоальтернативного ветвления

Это упрощенный вариант алгоритма вложенного ветвления – здесь внутренние ветвления располагаются только в ложных ветвях внешних по отношению к ним ветвлений. Также в любой ложной ветви с вложенным ветвлением не содержится никаких дополнительных команд кроме самого ветвления.

Суть данного алгоритма проста. Сначала проверяется условие 1. Если оно истинно, то выполняется набор команд 1; остальные условия не проверяются, управление передается следующему оператору после условия (на рис. 4 – завершается работа алгоритма). Если же результат проверки условия 1 оказался ложным, то только тогда проверяется условие 2, и т.д. В крайнем случае по очереди будут проверены все условия, все они выдадут ложны результат и только тогда будет выполнен набор команд N+1.

На заметку: во многих языках программирования присутствует оператор многоальтернативного выбора (например, `case` в языке Pascal). Его суть заключается ветвлении в зависимости от конкретного значения одной переменной. На схеме он также обозначается иначе. Однако в Python подобной конструкции нет, т.к. она избыточна и легко реализуется с помощью вложенного ветвления оператором `if-elif-else`, рассмотренным ниже. При этом все условия в таком операторе будут однотипны и разница будет лишь в конкретных проверяемых значениях.

Оператор if-elif-else

Для реализации алгоритма, представленного на рис. 1, можно использовать обычный оператор if несколько раз в виде вложенного ветвления. Однако благодаря упрощенной структуре алгоритма есть возможность несколько видоизменить запись, используя предназначенный для этого оператор `if-elif-else`. Работа данного оператора абсолютно совпадает с алгоритмом, представленным на схеме (рис. 4). Его синтаксис имеет вид:

```
if <условие 1>:
    <набор команд 1>
elif <условие 2>:
    <набор команд 2>
...
```

```
elif <условие N>:
    <набор команд N>
else:
    <набор команд N+1>
```

Пример выполнения задания

Задание. Вывести меню выбора действия и для вводимого x вычислить значение выбранной пользователем функции $f_i(x)$:

1) $f_1(x) = x^2$;
2) $f_2(x) = 2x + 1$;
3) $f_3(x) = 3x^2 + 4x + 5$;
4) $f_4(x) = 6x^{0.8}$.

Схема алгоритма программы (рис. 2):

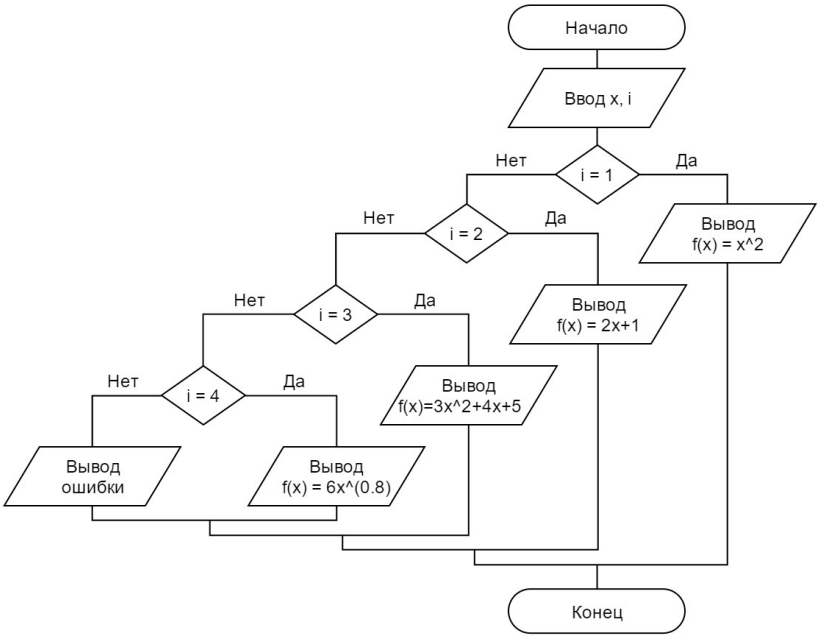


Рис. 2. Алгоритм выбора варианта действия

Текст программы:

```
x = float(input("Введите x: "))
print("Выберите номер функции: ")
print("1. f(x) = x^2")
print("2. f(x) = 2x + 1")
print("3. f(x) = 3(x^2) + 4x + 5")
print("4. f(x) = 6(x^0.8)")
i = int(input("Номер: "))

if i == 1:
    y = x**2
    print("f(x) = x^2 = ", y)
elif i == 2:
    y = 2*x + 1
    print("f(x) = 2x + 1 = ", y)
elif i == 3:
    y = 3*(x**2) + 4*x + 5
    print("f(x) = 3(x^2) + 4x + 5 = ", y)
elif i == 4:
    y = 6*(x**0.8)
    print("f(x) = 6(x^0.8) = ", y)
else:
    print("Неверный номер функции!")
```

Результаты выполнения:

```
Введите x: 2
Выберите номер функции:
1. f(x) = x^2
2. f(x) = 2x + 1
3. f(x) = 3(x^2) + 4x + 5
4. f(x) = 6(x^0.8)
Номер: 4
f(x) = 6(x^0.8) =  10.44660675955349
```

Контрольные вопросы

1. Для чего используется многоальтернативное ветвление?
2. Опишите алгоритм многоальтернативного ветвления?
3. Каков синтаксис оператора `if-elif-else`?

Задания

Для задания в соответствии со своим вариантом составить алгоритм и написать программу, реализующую выполнение определенных действий в зависимости от результатов проверки заданного набора условий. Оформить отчет, содержащий текст варианта задания, схему алгоритма, код программы и результаты выполнения.

Варианты задания:

1. Вывести меню выбора действия и для вводимого *x* вычислить значение выбранной пользователем функции $f_i(x)$:
   1) $f_1(x) = x^2$;
   2) $f_2(x) = \sqrt{x}$;
   3) $f_3(x) = e^x$;
   4) $f_4(x) = \ln(x)$.
2. Для введенного числа исполнителей вывести название ансамбля из данного числа людей:
   1) 2 человека – дуэт;
   2) 3 человека – трио;
   3) 4 человека – квартет;
   4) 5 человек – квинтет.
3. Вывести в современных единицах измерения значение выбранной старой русской меры длины:
   1) миля = 7,4676 км;
   2) верста = 1,068 км;
   3) сажень = 2,1336 м;
   4) аршин = 71,12 см;
   5) локоть = 54,7 см.

4. Вывести меню выбора действия и для вводимого *x* вычислить значение выбранной пользователем функции $f_i(x)$:

1) $f_1(x) = x^2$;
2) $f_2(x) = (x+1)^3$;
3) $f_3(x) = (x-2)^4$;
4) $f_4(x) = (x+3)^5$.

5. Вывести меню выбора действия и вычислить выбранное пользователем значение:

1) $a = \sin(\pi/2)$;
2) $b = \sin(\pi/3)$;
3) $c = \sin(\pi/4)$;
4) $d = \sin(\pi/5)$.

6. Вычислить и вывести число байт в выбранной единице измерения:

1) килобайт = 2^{10} байт;
2) мегабайт = 2^{20} байт;
3) гигабайт = 2^{30} байт;
4) терабайт = 2^{40} байт.

7. Вывести кратность выбранной приставки, используемой в метрической системе:

1) дека = $1*10^1$;
2) гекто = $1*10^2$;
3) кило = $1*10^3$;
4) мега = $1*10^6$;
5) гига = $1*10^9$.

8. Вывести меню выбора действия и вычислить выбранное пользователем значение:

1) $a = 3!$;
2) $b = 4!$;
3) $c = 5!$;
4) $d = 6!$.

9. Вывести меню выбора действия и для вводимого *x* вычислить значение выбранной пользователем функции $f_i(x)$:

1) $f_1(x) = x^3$;
2) $f_2(x) = 5x - 1$;
3) $f_3(x) = 3x^3 + x + 2$;
4) $f_4(x) = 5x^{1.7}$.

10. Согласно выбранному пользователем методу вычислить и вывести на экран приближенное значение числа π:
    1) Китай: $\pi = \sqrt{10}$;
    2) Архимед: $\pi = (3\frac{10}{71} + 3\frac{1}{7})/2$;
    3) Встроенная константа языка: $\pi = 3.141592$ ....
11. Для вводимого номера месяца вывести название времени года:
    1) 12, 1, 2 – зима;
    2) 3, 4, 5 – весна;
    3) 6, 7, 8 – лето;
    4) 9, 10, 11 – осень.
12. Для вводимого номера месяца вывести число дней в нем:
    1) 1, 3, 5, 7, 8, 10, 12 – 31 день;
    2) 4, 6, 9, 11 – 30 дней;
    3) 2 – 28 дней.
13. Для вводимой числовой оценки вывести ее текстовое представление:
    1) 1 = плохо;
    2) 2 = неудовлетворительно;
    3) 3 = удовлетворительно;
    4) 4 = хорошо;
    5) 5 = отлично.
14. Для вводимых чисел *a* и *b* вычислить результат выбранного действия:
    1) сложение;
    2) вычитание;
    3) умножение;
    4) деление.
15. По вводимому числу *k* грибов вывести фразу «Мы нашли в лесу <k> грибов», где окончание слова «грибов» согласуется со значением числа *k* (гриб, гриба, грибов).
16. Вывести меню выбора действия и для вводимого *x* вычислить значение выбранной пользователем функции $f_i(x)$:
    5) $f_1(x) = \sin(x)$;
    6) $f_2(x) = \cos(x)$;
    7) $f_3(x) = tg(x)$;
    8) $f_4(x) = ctg(x)$.

17. Для выбранной фигуры вычислить площадь, используя значение вводимого *x*:
    1) «о» = окружность, *x* – длина радиуса;
    2) «т» = равнобедренный прямоугольный треугольник, *x* – длина катета;
    3) «к» = квадрат, *x* – длина стороны.
18. Для вводимого символа *x* вывести фразу «Символ <x> - это…»:
    1) a..z – строчная буква;
    2) A..Z – прописная буква;
    3) 0..9 – цифра.
19. Для вводимого номера семестра вывести номер курса:
    1) 1, 2 – 1 курс;
    2) 3, 4 – 2 курс;
    3) 5, 6 – 3 курс;
    4) 7, 8 – 4 курс.
20. Для введенного количества углов вывести название геометрической фигуры:
    1) 3 – треугольник;
    2) 4 – четырехугольник;
    3) 5 – пятиугольник;
    4) ≥6 – многоугольник.

Литература

1. Python на примерах. Практический курс по программированию. / А.Н. Васильев. – СПб.: Наука и Техника, 2016. – 432 с.
2. http://pythonicway.com
3. https://pythonworld.ru
4. http://www.python-course.eu/python3_course.php

Содержание