13. Ветвление в языке Python

Логические переменные

Как известно, величины логического типа принимают всего два значения. В Python это False и True. Эти константы определены так, что False < True. Логические переменные относятся к типу bool, названному в честь английского математика *Джорджа Буля* - создателя алгебры логики. Логические значения получаются в результате выполнения операций сравнения числовых, строковых и логических выражений. Поэтому в Python логической переменной можно присваивать результат операции сравнения.

Напишем программу, определяющую истинность высказывания «Число *N* является чётным» для произвольного целого числа *N*. Пусть *z* - логическая переменная, *N* - переменная целого типа. Тогда в результате выполнения оператора присваивания *z = N % 2 == 0* переменной z будет присвоено значение *True* при любом чётном *N* и *False* в противном случае.

Пример 1.

```
print (' Определение истинности высказывания о чётности числа ')
N = int(input( ' Введите исходное число: ' ))
z = N % 2 == 0
print ( ' Число ', N, ' является чётным - ', z )
```

Логические выражения – условия

Логическое условие - это выражение, относительно которого можно сказать истинно оно (то есть выполняется) или ложно (то есть не выполняется).

В качестве условий в операторах ветвления используются логические выражения:

• простые записанные с помощью операций отношения
(<, >, >=, <=, != (не равно), == (равно));

• составные (сложные) - записанные с помощью логических операций and - и, or - или, not - не.

Например, на улице холодно и идет дождь. Два простых условия (на улице холодно), (на улице идет дождь) здесь связаны связкой И. Таким образом, **сложное условие** состоит из двух или нескольких простых отношений (условий), которые объединяются с помощью логических операций - связок

Логическая связка	Перевод	Пример	Пояснение	Значение
and &	и	(x > 0) and (x % 10 == 3) (x > 0) & (x % 10 == 3)	Положительное и оканчивается на 3	Истинно, когда обе части истинны
or	или	(x <= -10) or (x >= 10)	x меньше или равен -10 или больше или равен 10	Ложно, когда обе части ложны
not	не	not (x % 10 == 0)	число не оканчивается нулем	Замена на противоположное

Операция **И** - требует одновременного выполнения двух условий. **Условие 1 and условие 2** - будет принимать истинное значение, только если оба простых условия истинны одновременно, *причем, если условие 1 ложно, то условие 2 проверяться не будет*.

Операция **ИЛИ** - требует выполнения хотя бы одного из условий. **Условие 1 or условие 2** - будет принимать ложное значение, только если оба простых условия ложны одновременно, *причем, если условие 1 истинно, то условие 2 проверяться не будет*.

Операция **НЕ not условие 1** - будет принимать ложное значение, если условие 1 истинно и наоборот.

Например, условия *a = =b* и *not(a != b)* истинны для одних и тех же значений a и b, поэтому одно из них можно заменить на другое. Такие условия называются *равносильными*.

Приведём примеры равносильных условий.

Равносильными будут условия: *A > B* и *not (A <= B)*.

Условие *not (x >= 0 and x <= 10)* означает «x не находится внутри отрезка [0; 10]». Это значит, что значение x на числовой оси расположено левее нуля или правее, чем 10. Поэтому его можно записать без использования операции «НЕ»: *x < 0 or x > 10* . Обратите внимание, что в исходном выражении простые условия были связаны с помощью операции «И», а в равносильном обратные условия связаны с помощью «ИЛИ».

Условие *not (x == 2 or x == 5)* означает, что значение x не равно ни двум, ни пяти, то есть истинно условие *x != 2 and x != 5* . Здесь при переходе к равносильному условию без НЕ логическая операция «ИЛИ» была заменена на «И».

Используя операцию «НЕ», можно записывать условия по-разному, как нам удобнее в каждом случае.

Приоритет выполнения логических операций и отношений

1. Операции в скобках
2. Операция НЕ
3. Логические отношения >, <, >=, <=, ==, !=
4. Операция И
5. Операция ИЛИ

Для изменения порядка действий используются круглые скобки.

Напишем программу, определяющую истинность высказывания «Треугольник с длинами сторон a, b, c является равнобедренным» для произвольных целых чисел a, b, c.

Пример 2.

```
print( ' Определение истинности высказывания о  равнобедренном треугольнике ' )
a =  int(input (' Введите значение   a:    '))
b =  int(input (' Введите значение  b:    '))
c =  int(input (' Введите значение  c:    '))
z = (a  ==  b )  or  ( a  ==  c )  or  ( b  ==  c )
print( ' Треугольник является равнобедренным- ',  z)
```

В языке Python разрешены двойные неравенства, например, $A < B < C$.
Так, вместо *x>=0 and x<10* можно записать *0<=x<10*. Но в других языках и программах, позволяющих производить сравнение, такая запись приведёт к ошибке.

Условный оператор - оператор ветвления

При записи на языке Python разветвляющихся алгоритмов используют условный оператор.

Общий вид оператора полного ветвления:

```
if <условие>:
    <блок операторов 1>
else:
    <блок операторов 2>
```

Слово ***if*** переводится с английского языка как "**если**", а слово ***else*** - как "**иначе**". Если условие после слова *if* верно (истинно), то выполняются все команды (операторы), стоящие после двоеточия с новой строки и расположенные до слова *else* (блок операторов 1). Если условие неверно (ложно), то выполняются команды, стоящие после слова *else*: (блок операторов 2).

В отличие от других языков программирования, в Python важны отступы (сдвиги) операторов относительно левой границы. Эти **сдвиги влияют на работу программы**. Если посмотреть внимательно, то слово *if* и слово *else* начинаются на одном уровне, а команды, которые выполняются внутри ветвления, сдвинуты относительно этого уровня вправо на одно и тоже расстояние (рекомендуется использовать 4 пробела или символы табуляции, которые вставляются при нажатии на клавишу *Tab*). После окончания блока операторов 2, продолжаем писать программу без отступа, то есть на одном уровне с if.

Если при истинности (ложности) какого-то условия предполагается выполнить по одному действию, то условный оператор может быть записан в две строки:

```
if <условие>:  <оператор 1>
else:<оператор 2>
```

Если в блоке «иначе» не надо ничего делать (например: «если в продаже есть мороженое, купи мороженое», а если нет ...), то весь блок «иначе» можно опустить и использовать сокращенную (неполную) форму условного оператора.

Общий вид оператора неполного ветвления:

```
if <условие>:              # после условия обязательно ставится двоеточие
    <действия 1>           # отступ должен быть 4 пробела или один TAB
    <неск. команд>         # все внутренние инструкции делаются с одним отступом
<действия>                 # после окончания условия, продолжаем писать программу без отступа
```

Если при истинности условия предполагается выполнить одно действие, то условный оператор неполного ветвления может быть записан в одну строку:

```
if <условие> : <оператор>
```

Надо запомнить!

1. *if - else* - **ЭТО ОДИН ОПЕРАТОР!**

2. После слова *else* **НИКОГДА УСЛОВИЕ НЕ СТАВИТСЯ** (условие ставиться только после слова *if*).
3. Блок "**иначе**" выполняется тогда, когда основное условие, указанное после слова *if* - ложно, т.е. не выполняется.
4. Операторы, которые необходимо выполнить в каждой ветке (*if* или *else*), записываются с одинаковым сдвигом в 4 пробела. **СДВИГИ ОБЯЗАТЕЛЬНЫ!**

Рассмотрим несколько примеров программ с полным ветвлением.
Пример 3:

```
# Вводится два различных целых числа. Найти большее из них.
a = int (input ( ' Введите a: '))
b = int (input ( ' Введите b: '))
if  a > b:
   M  =  a
else:
   M  =  b
print ( ' большее ' ,  M)
```

Эту же программу можно написать и короче. Запишем её для сравнения вещественных чисел.

```
a, b  = map(float, input( ' Введите два числа ' ).split())
if a > b: print( ' большее ' , a )
else: print( ' большее ' , b )
```

Если выбирается максимальное из двух чисел, можно использовать особую форму условного оператора в Python: M = a if a > b else b

Операция выбора максимального из двух значений используется очень часто, поэтому в Python есть встроенная функция *max*, которую можно вызвать таким образом: *M = max(A, B)*. Есть и аналогичная функция для поиска минимального значения из двух или нескольких значений - *min()*.

Пример 4:

```
# Вводится целое число. Определить, четное оно или нет.
x = int(input(‘ введите целое число ’))
if  x % 2 == 0:  print( ' Число четное ' )
else: print( ' Число нечетное ' )
```

Рассмотрим несколько примеров программ с неполным ветвлением.

Пример 5.

```
# Нахождение наибольшей из трёх величин
a = int(input( ' Введите a :  ' ))
b = int(input( ' Введите b :  ' ))
c = int(input( ' Введите  c :  ' ))
y = a
if  b  >  y :   y  =  b
if  c  >  y :   y  =  c
print ( ' наибольшее  ' ,  y)
```

Пример 6.

```
# Вводится три целых числа. Определить, сколько среди них отрицательных.
x, y, z = map(int, input(‘ Введите три целых числа ‘ ).split())
k = 0
if  x < 0:    k+=1
if  y < 0:    k+=1
if  z < 0:    k+=1
print( ‘ отрицательных из них ‘, k)
```

Часто при выполнении какого-то условия нужно выполнить сразу несколько действий. Например, в задаче сортировки значений переменных a и b по возрастанию, нужно поменять местами значения этих переменных, если a > b.

Пример 7.

```
# Вводится два числа. Вывести их в порядке возрастания
a = int(input( ' Введите a :  ' ))
b = int(input( ' Введите b :  ' ))
if a > b:
   temp = a
   a = b
   b = temp
print ( a , b)
```

Здесь *temp* – это временная (вспомогательная) переменная (от англ. temporary – временный). Все операторы, входящие в блок ветвления и выполняемые, когда условие истинное, сдвинуты на одинаковое расстояние (4 пробела) от левого края. Начало и конец блока, который выполняется при истинном условии, определяется именно этими сдвигами.

Заметим, что в Python, в отличие от многих других языков программирования, есть множественное присваивание, которое позволяет выполнить такой обмен значительно проще: a, b = b, a

Пример 8.

```
# Вводится целое число не равное нулю. Если оно отрицательное, то возвести его в квадрат и вывести на
экран, а если положительное, то возвести в куб и вывести на экран.
x = int(input('введите целое число '))
if x < 0:
   y = x*x
   print(y)
if x > 0:
   y = x**3
   print(y)
```

Рассмотрим примеры программ с ветвлением и составным (сложным) условием.

Предположим, что ООО «Кнут и Пряник» набирает сотрудников, возраст которых от 25 до 40 лет включительно. Нужно написать программу, которая запрашивает возраст претендента и выдает ответ: «подходит» он или «не подходит» по этому признаку. Какое же условие должно быть истинно для того, чтобы человека приняли на работу? Одного условия «возраст >= 25» не хватает, это условие соблюдается и для людей старше 40 лет. С другой стороны, условия «возраст <= 40» тоже не хватает, так как оно выполняется и для школьников. В этой задаче нужно, чтобы два условия выполнялись одновременно: «возраст >= 25» и «возраст <= 40».

Эту задачу можно решить с помощью вложенного условного оператора, но один и тот же ответ «не подходит» придётся выводить в двух местах программы. Почти во всех языках программирования в условном операторе можно использовать такое условие:

```
if age >= 25 and age <= 40:  print(  ' подходит '  ) else:  print(  ' не подходит '  )
```

В условном операторе мы записали сложное условие, составленное из двух простых с помощью логической операции *and*.

В программах на языке Python можно сразу проверить выполнение двойного неравенства и записать

```
if 25 <= age <= 40:
```

но во многих популярных языках программирования такая запись приводит к ошибке.

Пример 9.

```
#  Определение принадлежности точки X отрезку [a,  b].
a =  int(input( ' Введите  a:  ' ))
b =  int (input ( ' Введите b :  ' ))
x =  int (input ( ' Введите x :  ' ))
if  (x >= a)  and  (x <= b) :
   print( ' Точка принадлежит отрезку ')
else:
   print( ' Точка не  принадлежит отрезку ')
```

Пример 10.

```
#Решить линейное уравнение вида Ax + B = 0
a = float(input( ' Введите коэффициент a:   '))
b = float(input( ' Введите коэффициент b:   '))
```

```
if a  !=  0 :
   x  =  - b  /  a
   print  ( ' Корень  уравнения  x = ' ,  x)
if  (a== 0)  and  (b  != 0) :
   print  ('Корней нет ')
if  (a== 0)  and  (b  == 0) :
   print ( ' x  - любое число ')
```

Предположим, что нам надо убедиться, что значение целой переменной A – трёхзначное число, которое делится на 7. Для этого нужно, чтобы одновременно выполнились три условия:
1) число не меньше 100;
2) число меньше 1000;
3) число делится на 7, то есть остаток от его деления на 7 равен нулю.
В условном операторе эти три простых условия должны быть связаны с помощью двух операций «И»:

```
if 100 <= a  and  a < 1000  and  a % 7 == 0:  print( ‘Да! ‘ )
else:  print( ‘ Нет. ‘ )
```

Рассмотрим ещё одну задачу. Самолёт из Санкт-Петербурга в Барнаул летает только по понедельникам и четвергам. В переменную *day* вводится номер дня недели (от 1 – понедельник до 7 – воскресенье). Программа должна определить, полетит ли самолёт в этот день.

Если мы напишем условие «*day = 1 and day = 4*», то это будет неверно, потому что мы потребовали, чтобы значение переменной *day* было одновременно равно и 1, и 4. Такого быть не может, поэтому это условие всегда будет ложно. Значит, операция «И» не подходит. Вместо неё нужно применить другую операцию – «ИЛИ», которая требует выполнения хотя бы одного из связанных условий.
Фрагмент решения нашей задачи выглядит так:

```
if day == 1 or day == 4: print( ‘ Полетит! ‘ )
else: print( ‘ Нет рейса. ‘ )
```

Ещё одна операция, которую можно использовать в сложных условиях – «not» («НЕ»), она означает обратное условие (противоположное исходному). Если исходное условие истинно, то обратное (противоположное) ему – ложно, и наоборот. Фрагмент решения задачи с самолётом можно записать так:

```
if not( day == 1 or day == 4 ):  print( "Нет рейса." )
else:  print( "Полетит!" )
```

Вложенный условный оператор

В блоки «если» и «иначе» могут входить любые другие операторы, в том числе и другие **вложенные условные операторы**, при этом слово ***else*** относится к ближайшему предыдущему ***if***.
Пример 11.

```
A= float(input( ' Сколько у вас денег?   '))
if A > 100:
   if A > 10000:
      print('У вас много денег. ')
   else:
      print('У вас достаточно денег. ')
else:
   print('У вас мало денег. ')
```

Жирным шрифтом выделен условный оператор, который находится внутри другого оператора ***if***, поэтому он называется **вложенный условный оператор**. Предыдущую задачу можно решить более коротким способом, используя сложные условия.

С помощью вложенного условного оператора можно реализовать выбор из нескольких вариантов, а не только из двух. Можно вложенный оператор использовать и после слова ***else*.** Например, пусть возраст Андрея записан в переменной ageA, а возраст Бориса – в переменной ageB. Нужно определить, кто из них старше. Одним условным оператором тут не обойтись, потому что есть три возможных результата: старше Андрей, старше Борис или оба одного возраста. Решение задачи можно записать так:
Пример 12.

```
ageA = float(input( ' Введите возраст Андрея:   '))
ageB  =  float(input( ' Введите возраст Бориса:   ') )
```

```
if ageA > ageB:
   print( ' Андрей старше ' )
else:
   if ageA == ageB:
       print( ' Андрей и Борис ровесники ' )
   else:
       print( ' Борис старше ' )
```

Если после *else* необходимо проверить еще одно условие, то вместо оператора ***if***, можно использовать "**каскадное**" ветвление с ключевым словом *elif* (сокращение от *else - if*).

```
ageA = float(input( ' Введите возраст Андрея:   '))
ageB = float(input( ' Введите возраст Бориса:   '))
if ageA > ageB:
   print( ' Андрей старше ' )
elif ageA == ageB:
   print( ' Андрей и Борис ровесники ' )
else:
   print( ' Борис старше ' )
```

Обратите внимание на отступы во всех примерах. При использовании каскадного условия, все ключевые слова if-elif-else находятся на одном уровне. При большом числе проверок, записанных с помощью каскадного условия, например, в цепочке if-elif-elif-... срабатывает первое истинное условие.

Пример 13.

```
# определить размер скидки на значение переменной cost
cost = 1500
if cost < 1000:
   print( "Скидок нет." )
elif cost < 2000:
   print( "Скидка 2%." )
elif cost < 5000:
   print( "Скидка 5%." )
else:
   print( "Скидка 10%." )
```

Эта программа выводит «Скидка 2%.», хотя условие cost < 5000 тоже выполняется.

Запишем программу для решения квадратного уравнения вида $Ax^2 + Bx + C = 0$, где x - переменная, A, B и C - некоторые числа (коэффициенты), причём A не равно 0.

Пример 14.

```
#  Решение квадратного уравнения
from math import*
print(' Введите коэффициенты A,  B,  C: ')
a = float(input(' A = '))
b = float(input(' B = '))
c = float(input(' C = '))
d = b * b - 4 * a * c
if a == 0: print(' Уравнение не квадратное ')
elif d < 0: print(' Корней нет ')
elif d == 0:
    x = - b / (2 * a)
    print (' Корень уравнения x= ', "{:6.4f}".format(x))
else:
   x1 = (-b + sqrt(d)) / (2 * a)
   x2 = (-b - sqrt(d)) / (2 * a)
   print(' Корни уравнения: ')
   print(' x1 = ', "{: 6.4f}". format (x1))
   print(' x2 = ', "{: 6.4f}". format (x2))
```

Ветвления в других языках программирования

Знание хотя бы одного языка программирования позволяет понимать запись программы на многих других языках. Вот фрагменты программы, которая меняет местами значения двух переменных, на языках Паскаль и C++:

Паскаль	C++

if a > b then
begin
  temp := a;
  a := b;
  b := temp
end;
	if (a > b)
	{
	  temp = a;
	  a = b;
	  b = temp;
	}

В языке Паскаль оператор присваивания записывается в виде последовательности символов «:=», а в языке C++ – так же, как и в Python – с помощью одного знака «=». Оформление в языке Паскаль более сложное: после условия пишут слово then (по-английски – «тогда»), а составной оператор «охвачен» логическими скобками – служебными словами begin и end. В языке C++ условие после if обязательно заключается в круглые скобки, а составной оператор ограничивается фигурными скобками. { }

Контрольные вопросы и задания

1. Как обозначается логический тип данных? Какие значения принимают логические величины?
2. Что такое логическое условие?
3. Какие операции отношения вы знаете? Как они обозначаются?
4. Чем отличаются операторы «=» и «==»?
5. Какие логические условия называют простыми, а какие сложными (составными)?
6. Какие логические операции – связки используются в Python? Когда истинны условия, содержащие эти связки?
7. Как определяется порядок вычислений в сложном условии? Как его изменить?
8. Какие условия считают равносильными?
9. Запишите равносильные условия, не используя операцию «*not*»:
   а) not (a < 6)
   б) not (c != 15)
   в) not (7 < a and a < 12)
   г) not (b != c or d < 5)
10. Как переводятся служебные слова *if* и *else*?
11. Каков общий вид оператора полного ветвления?
12. Каков общий вид оператора неполного ветвления?
13. Может ли быть использован *if* без *else*? А *else* без *if*?
14. Для чего нужны сдвиги вправо в строках текста программы? Объясните.
15. Когда можно записать ветвление в одну строку? В две строки? Приведите пример.
16. Что будет выведено на экран после выполнения следующей программы?
    a = int(input( ))
    if a == 1 and a == 2: print("Да!") else: print("Нет.")
17. Запишите программу поиска меньшего из двух чисел тремя способами (с полным ветвлением в три или четыре строки, с особой формой условного оператора, с использованием функции *min()*).
18. Как поменять местами значения двух переменных в Python? Запишите команду.
19. Запишите программу с полным ветвлением и составным условием, которая запрашивает возраст претендента на работу и выдает ответ «подходит» для тех, кому от 25 до 40 лет и «не подходит» для всех остальных.
20. Запишите программу с полным ветвлением и составным условием, которая определяет, является ли введенное целое число трехзначным и кратным 7.
21. Запишите программу с полным ветвлением и составным условием, содержащим *not*, которая по введенному номеру дня недели (от 1 до 7) выводит сообщение «Есть рейс» или «Нет рейса» для самолёта, который летает только по понедельникам, средам и субботам.
22. Что означает слово *elif*? Когда оно применяется?
23. Перепишите программу для решения квадратного уравнения с каскадным ветвлением. Прокомментируйте каждую строку программы подробно.