## Не хочу использовать сложные выражения

Ларри Уолл, автор первого языка Perl, однажды сказал, что у великих программистов есть три добродетели: лень, капризность и тщеславие. На первый взгляд, ни одно из этих трех слов не является положительным словом, но в мире программистов эти три слова имеют разные значения. Прежде всего, лень будет мотивировать программистов писать программы, спасающие от проблем, чтобы помочь себе или другим лучше выполнять свою работу, чтобы нам не приходилось выполнять эту повторяющуюся и утомительную работу; Точно так же мы никогда не будем писать 10 строк кода для вещей, которые можно решить с помощью 3 строк кода. Во-вторых, капризность заставит программистов проявить инициативу по выполнению какой-то работы, которую вы еще не предложили, оптимизировать свой код, чтобы сделать его более эффективным, могут выполнять задачи за 3 секунды, мы не должны терпеть 1 минуту ожидания. Наконец, тщеславие побуждает программистов писать надежный и безошибочный код, не для того, чтобы получать критику и обвинения, а для того, чтобы ими восхищались другие.

Итак, есть интересный вопрос, который стоит изучить: нам нужна программа, чтобы найти наибольшее число из трех введенных чисел. Эта программа - кусок пирога для всех, кто умеет программировать, даже для тех, кто не умеет программировать после 10 минут обучения. Ниже приведен код Python для решения этой проблемы.

a = int(input('a = '))b = int(input('b = '))c = int(input  
('c = ')  
)if a > b:the\_max = a else:  
 the\_max   
  
=  
 b   
если c > the\_max: the\_max =  
 cprint('  
Максимальное значение:', the\_max)

Но, как мы только что сказали, программисты ленивы, и многие программисты будут использовать троичные условные операторы для переписывания приведенного выше кода.

a = int(input('a = '))b = int(input('b = '))c = int(input  
('c = '))the\_max = a  
, если a > b else b  
the\_max   
= c если c > the\_max еще the\_max  
print('Максимальное значение:', the\_max)

Следует отметить, что в Python не было троичных условных операторов, используемых в строках 4 и 5 приведенного выше кода до версии 2.5, и причиной этого является Гвидо ван Россум ( Отец Python) считал, что тернарные условные операторы не помогают Python стать более лаконичным, поэтому те, кто привык использовать тернарные условные операторы в C/C++ или Java (в этих языках тернарные условные операторы также называются «Оператор Элвиса», потому что ?: собрал много похожего на большую спину известного рок-певца Элвиса Элвиса) программисты пытались использовать и и Природа короткого замыкания оператора OR имитирует тернарный оператор, поэтому в те времена приведенный выше код был написан так.

a = int(input('a = '))b = int(input('b = '))c = int(input  
('c = '))the\_max = a   
> b и a или b   
 the\_max   
= c > the\_max и c или the\_max  
print('Максимальное значение:', the\_max)

Однако этот подход недопустим в некоторых сценариях, и см. следующий код.

a = 0 b =  
 -100# Ожидалось, что следующий код выведет значение a, но вместо этого получит   
значение b #, потому что  
 a Значение 0 обрабатывается как False при выполнении логических операций для обработки   
print(True и a или б)   
# print(a, если True, иначе b)

Таким образом, после Python 2.5 были введены тернарные условные операторы, чтобы избежать вышеуказанных рисков (последнее предложение приведенного выше кода закомментировано). Итак, снова возникает вопрос, можно ли написать приведенный выше код более кратко? Ответ положительный.

a = int(input('a = '))b = int(input('b = '))c = int(  
input('c = '))print('Максимальное значение:',   
 (a if a > b else  
 b) if ( a, если a > b else b) > c else c)

Но действительно ли это хорошо? Делает ли такое сложное выражение код намного более неясным? Мы обнаружили, что в реальной разработке многие разработчики любят злоупотреблять функциями или синтаксическим сахаром определенного языка, так что простой многострочный код становится сложными однострочными выражениями. Я не раз задавал себе этот вопрос, и теперь ответ, который я могу дать, это следующий код, использующий вспомогательные функции.

def the\_max(x, y):  
 return x, if x > y else y  
  
  
 a = int(input('a = '))b = int(input('b = '))c = int(  
input('  
c = '))print  
 («Максимальное значение:», the\_max(the\_max(a, b), c))

В приведенном выше коде я определил вспомогательную функцию, the\_max используется для нахождения большего из двух значений, передаваемых параметром, поэтому следующий выходной оператор может найти максимальное значение трех чисел, вызвав функцию the\_max дважды, и теперь читаемость кода намного лучше. Замена сложных выражений вспомогательными функциями - действительно хороший выбор, ключ в том, что после того, как логика сравнения размера перенесена на эту вспомогательную функцию, ее можно не только вызывать повторно, но и выполнять каскадные операции.

Конечно, во многих языках нет необходимости реализовывать сами более крупные функции (обычно встроенные функции), и то же самое верно для Python. Встроенная функция max в Python использует преимущества поддержки Python для вариативных параметров, позволяя передавать сразу несколько значений или итератор и находить максимальное значение, поэтому проблемы, рассмотренные выше, в Python состоят из одного предложения. Но идея упрощения сложных выражений от сложных выражений до использования вспомогательных функций очень интересна, поэтому поделитесь ею и общайтесь с вами.

a = int(input('a = '))b = int(input('b = '))c = int(  
input('c = '))print('Максимальное значение:',   
max  
(a, b, c))