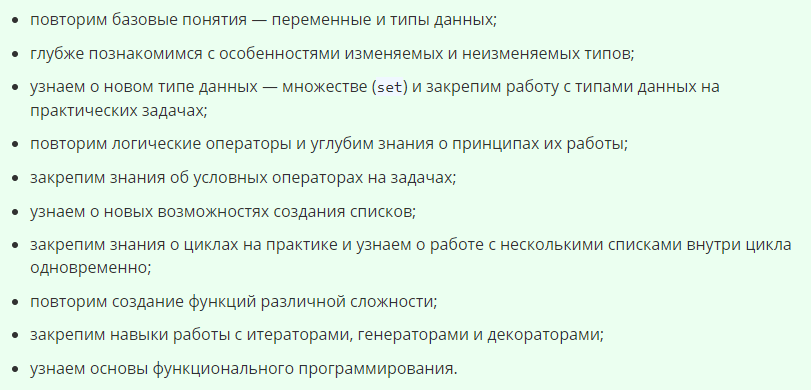
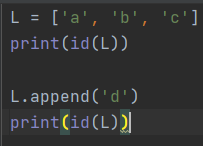
**B5**

В этом модуле будут практические задачи по пройденным темам. Посмотрим.



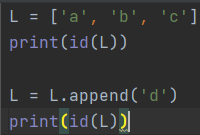
**B5.2. Переменные и типы данных**

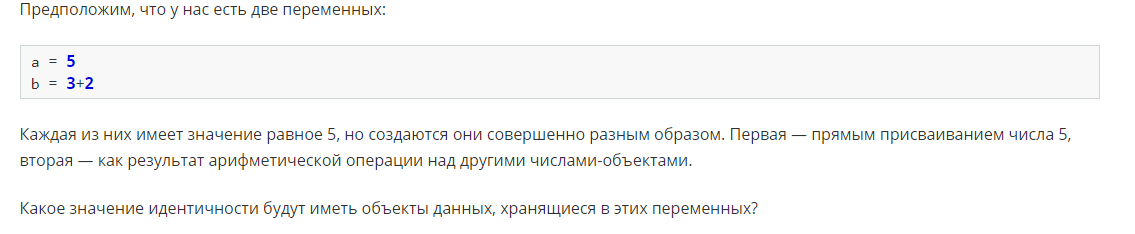


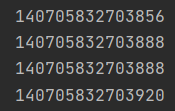
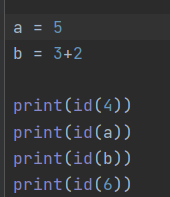
Напоминаю, что есть изменяемые и не изменяемые типы данных. Вот пример списка который можно изменить, но его id не изменится.



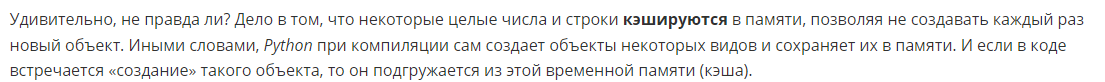
Но если произвести перезапись, то и ID изменится.





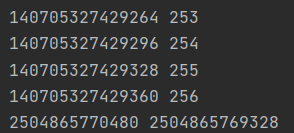
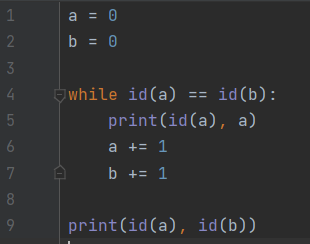


Я был уверен, что id будут разными, но нет. Не важно как сформировалась переменная. Важно что в ней хранится. Кстати, id целых чисел меняется через 2 и какой-то десяток...

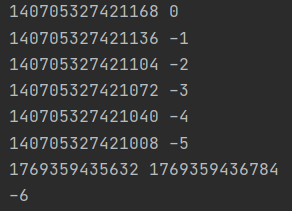
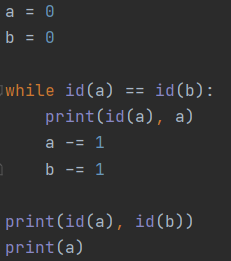


Понятно. Я этого не знал.

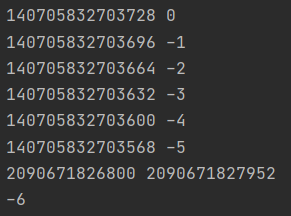
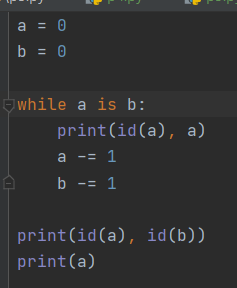


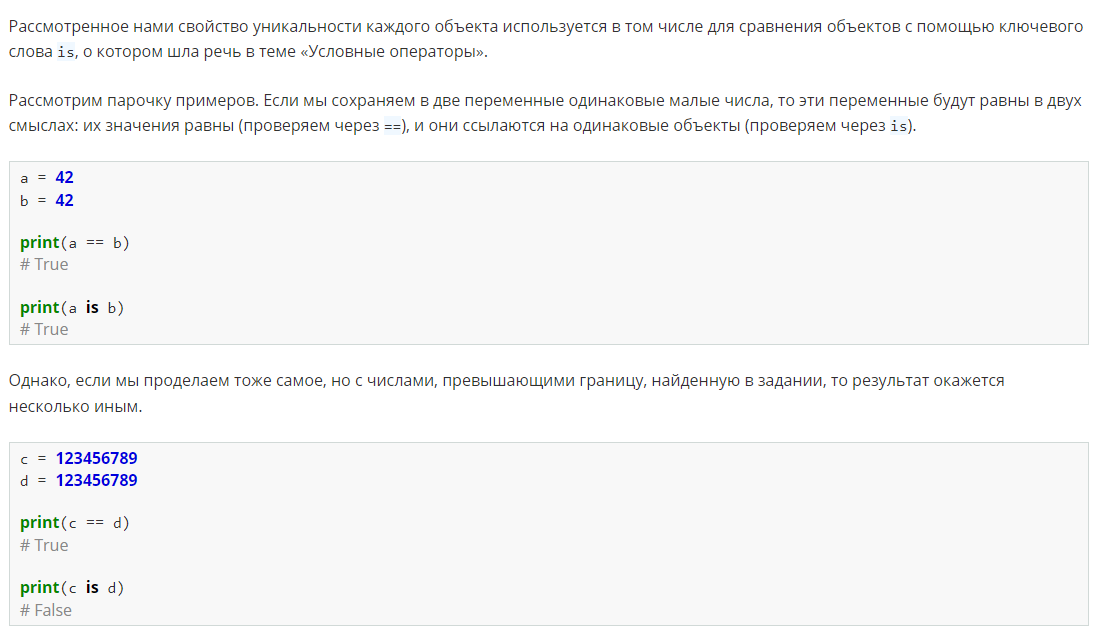


Таким образом Python может закэшировать до 257 переменных. Отрицательных чисел python хранит намного меньше.



К этому относится и сравнение чисел через **is**

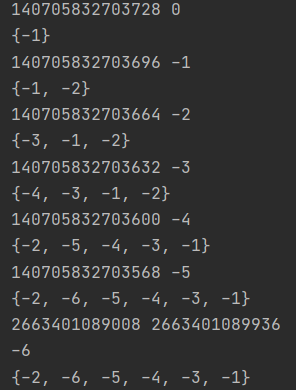
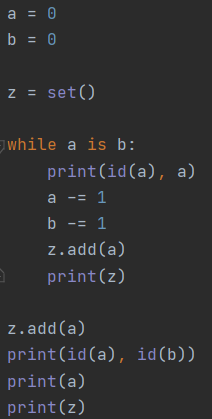




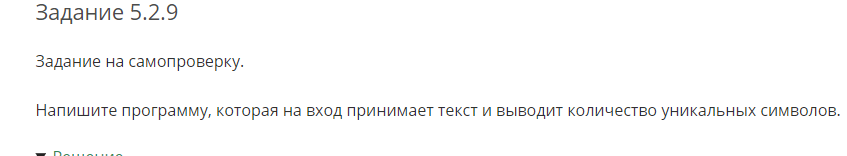
**Про неизменяемость кортежей.** Да, кортеж после создания изменить нельзя. Но если внутри кортежа есть переменные с изменяемым типом данных, то эти переменные можно свободно изменять, так как от этого их уникальный id изменится.

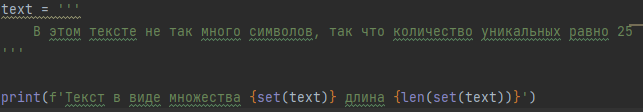
Просто напомню про еще один изменяемы тип данных, множества – **set.**

Суть в том, что во множестве не может быть элементов с одним и тем же **ID.** Что значит во множестве могут хранится только уникальные значения. Я тут подумал, а можно ли это попробовать обойти. Наверно все же речь не об **id.** А именно в значениях. Иначе условные -7, -8, -7 можно было бы туда засунуть, попробую.



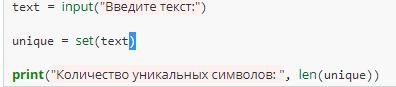
Да! Множества не зависят от **id.** Для них важны именно значения. Думаю два одинаковых списка туда тоже не добавить. Поправка. Один тоже не добавить. Множества вообще не поддерживают изменяемые типы данных, похоже.







Их вариант.



Тоже самое, только я забыл про ввод текста.

Хм…







Первый вариант это текст в программе через ‘’’ ‘’’. А второй вариант, это скопированный в консоль текст. И я кажется уже понял в чем дело. Секунду.

Да.. в консоль скопировалась только первая строка.



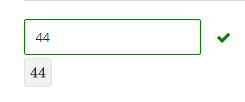
Чтобы вставить в консоль текст без пробелов, я удалил все переносы в ручную в notepade



Но тем не менее ответ я все равно получил 45



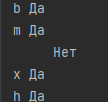
Хотя **skillfactory считают что ответ 44.**

****

И сейчас я выясню, в чем моя ошибка.



Кажется проблема в символе:



И этот символ был скрыт в самом конце.



/t это кстати, табуляция.

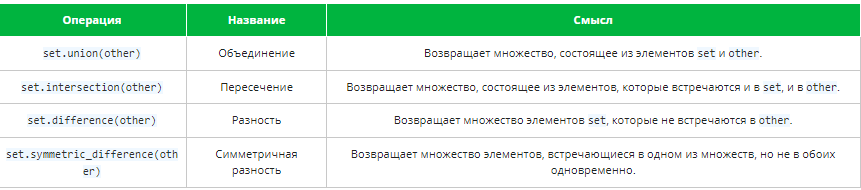
Блин, из-за проблемы с авто, думается не очень. Ладно дальше. У множества есть еще и свои методы.

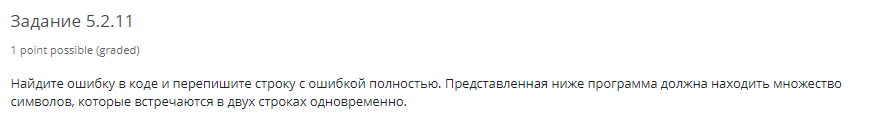
**Set.uninon(other) –** объединение двух множеств.

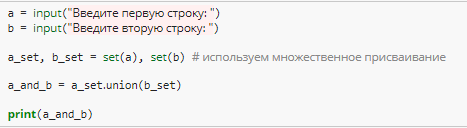
**Set.intersection(other) –** Возвращает пересечение двух множеств.

**Set.difference(other) –** возвращает разность двух множеств

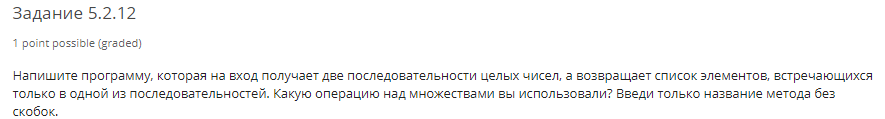
**Set.symmetric\_difference(other) –** Возварщает множество элементов, встречающиеся в одном из множеств, но не в обоих одновременно.

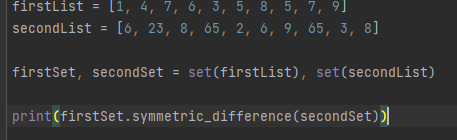






**a\_and\_b = a\_set.intersection(b\_set)**

****

****

****

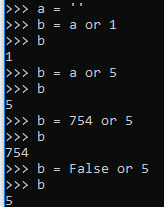
**B5.3. Условия и циклы**

Тип данных NoneType соответствует только одному виду переменных. **None.**

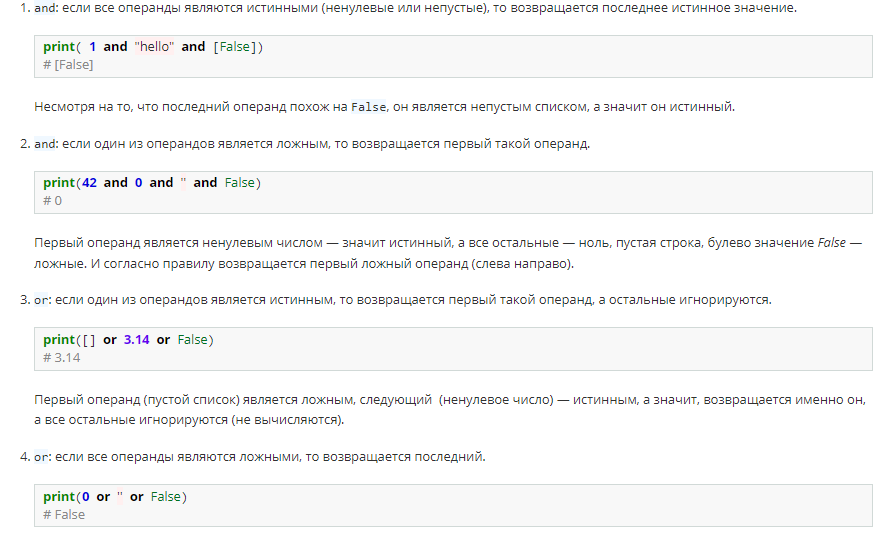
Его полезно использовать при объявлении переменных без значения. Напомню при проверки типов данных и некоторых других условий целесообразнее будет вместо **==** применять **is.**

**  
Так, честно меня поймала эта строка. B = 1. Ну причина…**

Так, как a можно считать за False, это выбирается 1. Но я думал 1 соответствует **True.** Но нет, если заменить **1** на другое число то оно и случится.

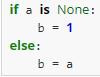


**Дело в том, что логические операторы возвращают не обязательно булевы значения (True или False), а значение одного из операндов. Это происходит согласно следующим правилам:**

****

**Поэтому варианты с присваиванием сработают,** а для условий if else **нет большой разницы**

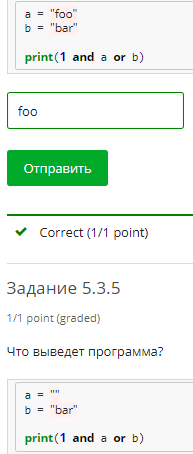
Можно тремя разными способами произвести присваивание переменной зависимости от условия. Ну… тремя только для такого условия, когда присваиваемое число может быть None или не быть.

** Обычный**

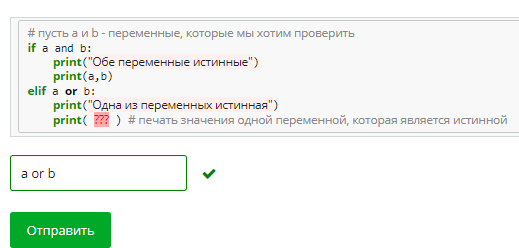
** Тернарный**

** Простой**

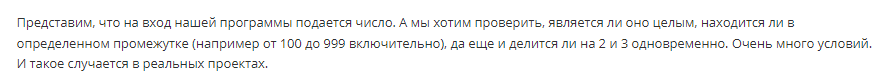
**Если честно.**

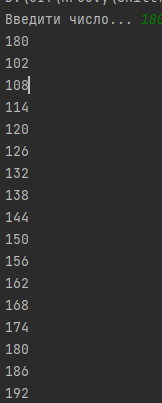
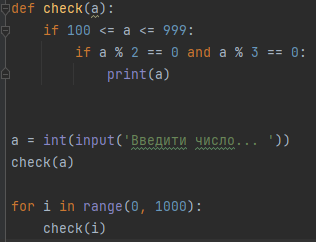


**Это вообще не интуитивно!!!**

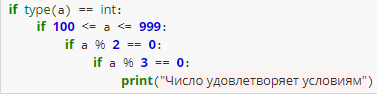


**Ну вот тут похвалю. Если одна из переменных была истинной, не важно a or b, нарисуется имена она.**

****

****

Ой.. имелось ввиду каждое отдельное условие?



Да и проверка на int тут бессмысленна, так как ввод все равно преобразован.



А если ввести тут не число, то до условий не дойдет еще там ошибка возникнет.

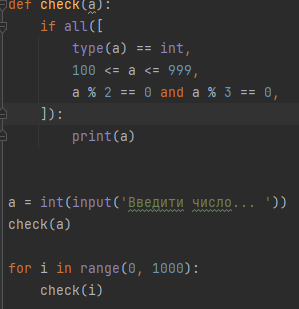
**Ну допустим я написал это через один вложенные цикл. Ну посмотрим на способ получше.**

Аа.. имелось ввиду просто все в одну строку…



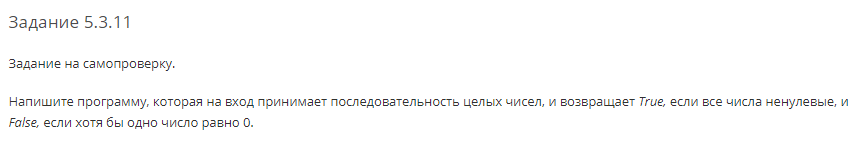
Ну.. с другой стороны это не так уж и плохо.

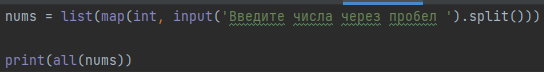
О.. и сейчас как раз будет решение. В **Python** есть функция условия. Я о ней вообще впервые слышу. Вот, как она работает. Функция называется **all()** и в качестве аргумента принимает список условий.

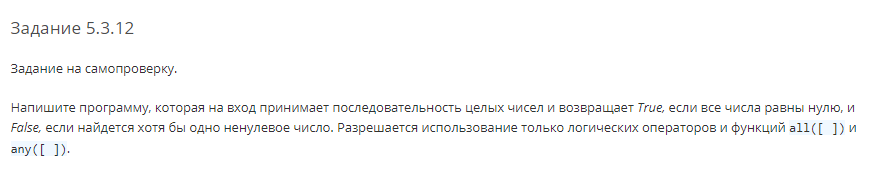
 Ну.. не плохо, читать довольно удобно. И никаких else.

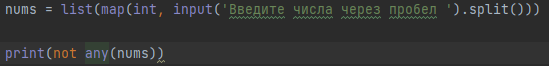
Сама функция возвращает **True**, если все условия списка истинны.

Если же достаточно хотя бы одно истинного условия, то для этого есть функция **any([]).**







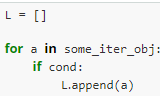


Писать большие списки вручную неудобно. Как оказалось большие списки можно составлять на лету и буквально в одну строку.

**List comprehension**

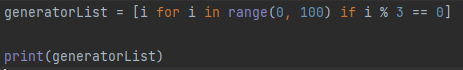
Можно назвать это чем-то вроде генератора списков.



 По факту эта запись аналогична этой.

Список со значениями a в промежутке **some\_iter\_obj** с условием.

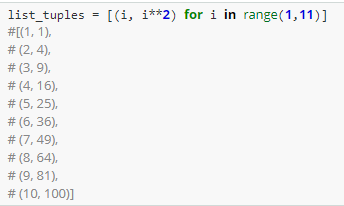
Попробуем.



Круто. Причем можно вернуть не просто i’ый элемент, а скажем его квадрат.

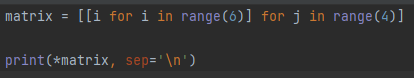
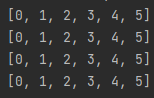


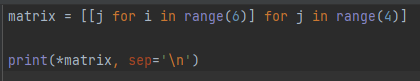
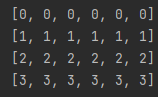
И возвращаемые типы данных тоже могут быть различны.

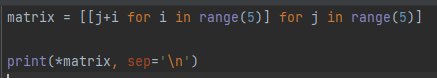
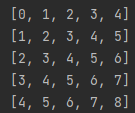


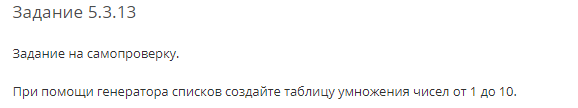
В идеале можно ускорить программу, если пробегаться через range() с определенным шагом, а не по всем элементам.

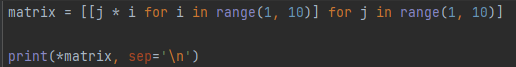
И отличный способ для создания матрицы определенной длины.

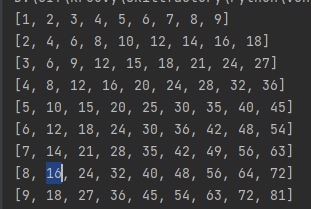
 

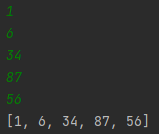
 



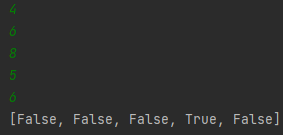




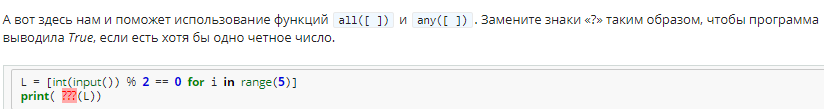
Таким способом можно совершить ручной в ввод в массив определенной длины.



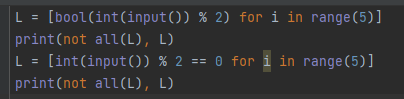
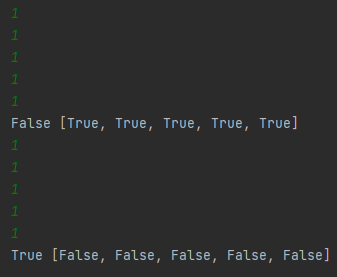
 

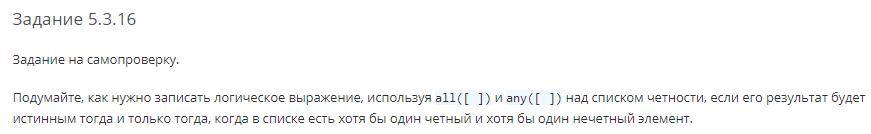


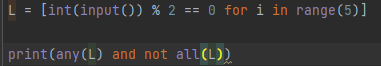
Да, упустил знак == этот способ лучше.



В моем коде была замечена ошибка. На самом деле такой записью я создаю инвертированный вариант того же списка.

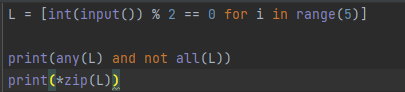
 

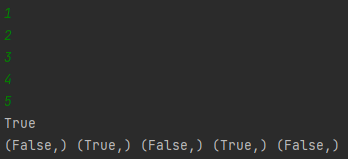




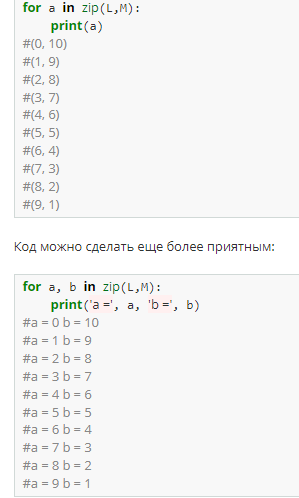
**zip()**

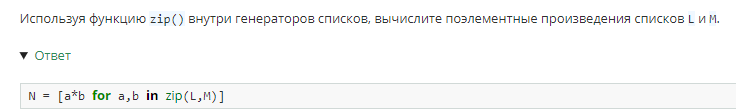
Если не ошибаюсь, это просто запаковка списка в кортеж. Происходит это так. Каждый отдельный элемент списка запаковывается в свой кортеж.



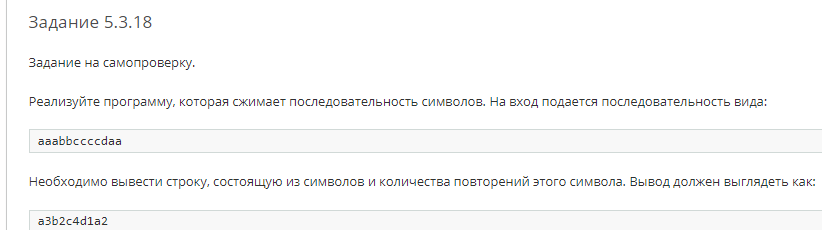


Таким образом можно запаковать сразу два списка, тем самым, получая доступ к одинаковым индексам разных списков.

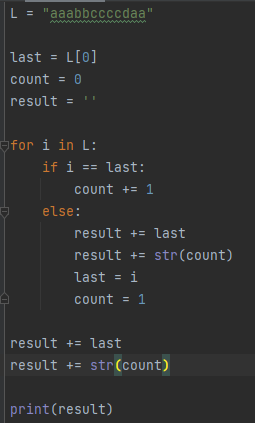




**Циклы**

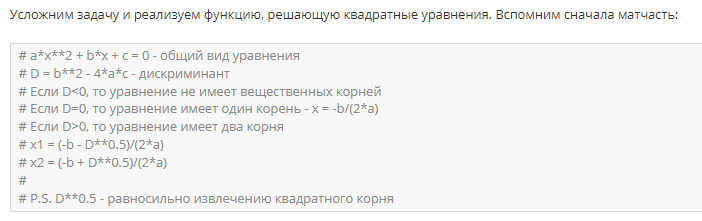
****

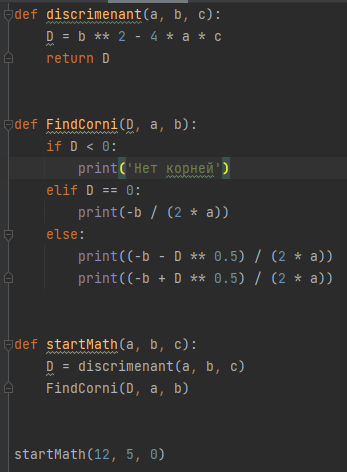
Признаться, башка немного не варила. Думал, как бы сделать это в рамках одного цикла. Но так как я пробегался через in array, то я не имел доступа к индексам. Поэтому в итоге я подсмотрел решение, и оно не представляло ничего особенного, думаю это единственный адекватный способ написать это решение на Python.



Просто без индекса я даже не могу без сторонних переменных определить, когда элемент является последним в цикле.

**Функции**

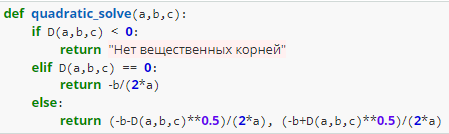




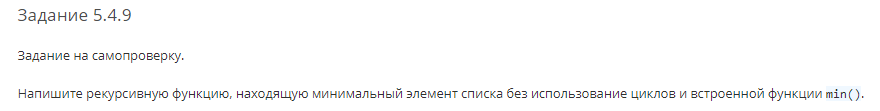
Если надо, всегда можно заменить **принты** на **return.**

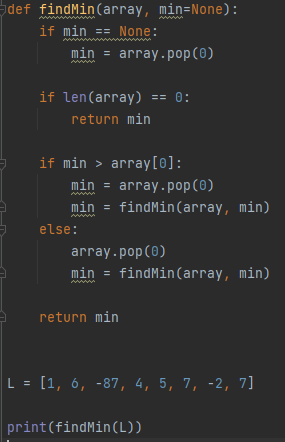
У них же код такой:



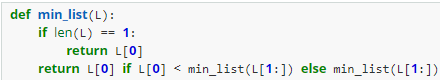


В целом разницы нет. Но я не вижу смысла в том, чтобы каждый раз заново считать дискриминант. По хорошему его нужно вынести в переменную выше до условий. Ладно, между if’ми, но в последнем else вызывается одна и та же функция дважды. Результат не изменится, но отработает больше функций.

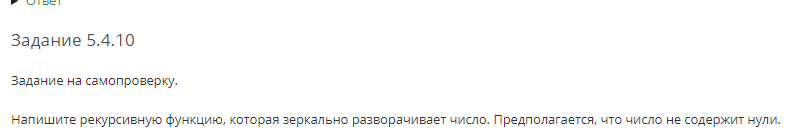


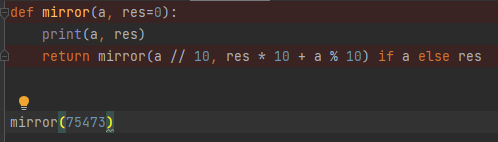


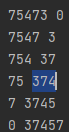




Я проиграл… Это отлично написанная функция.







Голова особо не варит, думал смогу ограничится одной переменной. В итоге просто лень. Это не мой алгоритм. Но код я понял. До тех пор, пока число не достигнет нуля будет вызываться функция,