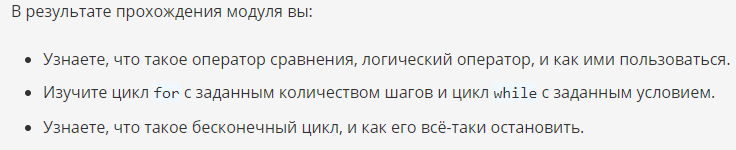
**B3**

С переменными пока норм. Теперь функции, циклы, условия.

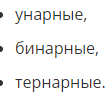


**Операторы сравнения и логические операторы**

Да, помню мат. логика, помню.

**Операторы** — это некий функционал, производящий какие-либо действия, который может быть представлен в виде символов как, например, «+» или специальных зарезервированных слов. **Операторы** могут производить некоторые действия над данными, и эти данные называются **операндами**. В общем смысле все операторы делятся на несколько типов.

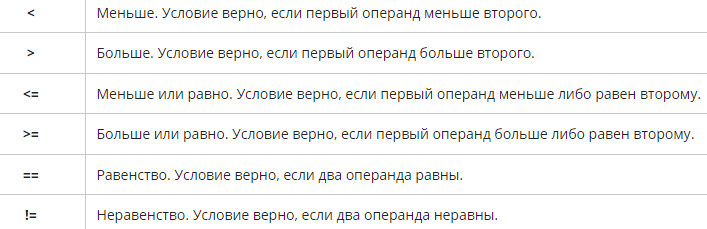
Операторы бывают:



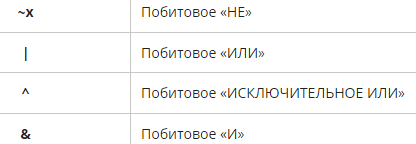
Унарные и бинарные операторы — это такие операторы, которые для своей работы требуют одно и два значения соответственно. Например, оператор сложения является бинарным оператором.

**А тернарные — это такие, которые возвращают свой второй или третий операнд в зависимости от значения логического выражения, заданного первым операндом.**

Чуть забыл.

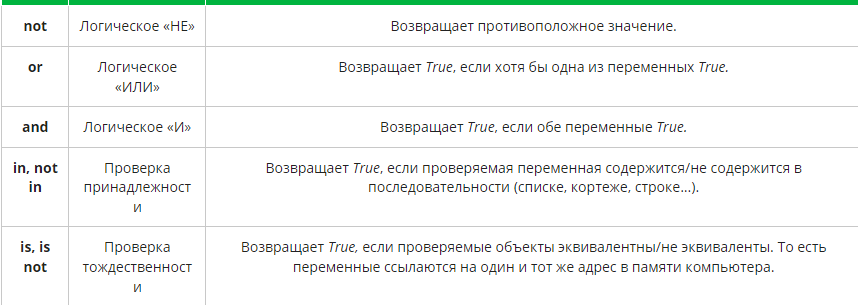


Среди логических операторов выделяют так называемые побитовые логические операторы. Мы подробно на них останавливаться не будем, но для ознакомления приведем.

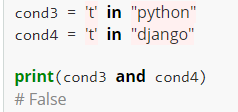
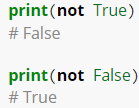


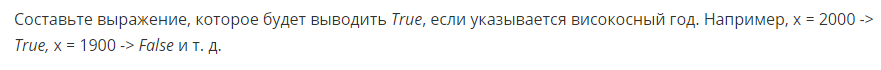
Да… с адресами памяти бывает полезно. Может позже прочитаю об этом: <https://tproger.ru/translations/bitwise-operations/>

2 и 4 если не ошибаюсь работают с операторами сравнения, только вместо одного их пара.



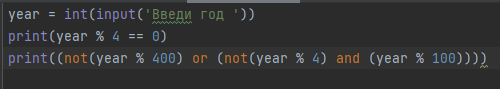
Просто кратко:



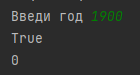




Вообще не понял, у них решение ниже и оно…



Странное. Во первых на 1900 оно не выводит False, оно выводит ноль.



А во вторых, 1900 високосный год, он делится на 4 без остатка. Короче, что за дичь вы написали? В задании и вообще.

**Практические примеры с различными операторами**

**Проверка вхождение элемента в последовательность**

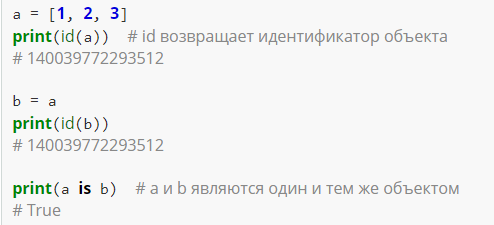


Да, можно сделать кучу операций, обратить число в строку, а потом в список. Но можно сделать так:



**Проверка эквивалентности**

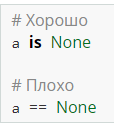
Помню упоминал насчет одинаковости списков. Точнее когда один список присваивают другому, то они оба начинают ссылаться на одну и ту же область памяти. Так вот, проверить это можно с помощью оператора **is.** Он возвращает **True** если оба объекта ссылаются на одну область памяти, то есть если они эквивалентны.



Эквивалентные объекты всегда равны, но равные объекты не обязательно эквивалентны.

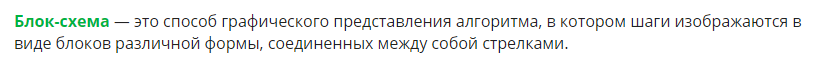
Есть еще один тип **None.** Он используется если значение переменной неизвестно заранее, но ее нужно объявить.

Если вы хотите проверить, является ли значение переменной **None** или нет, следует использовать **не оператор сравнения** **==,** а проверку **эквивалентности,** потому что идентификатор объекта **None** **не меняется на протяжении всей программы.**

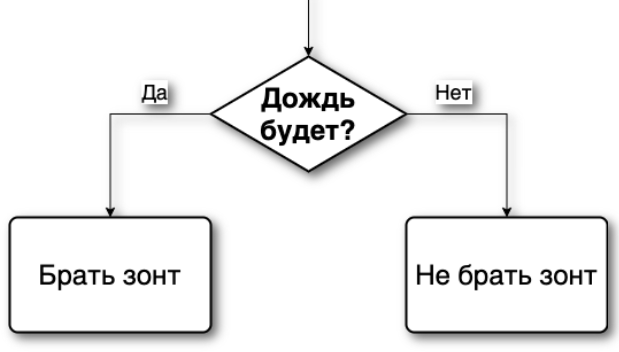


**Условный оператор**

Хм.. да тут надо вспомнить. Ведь в конце как у циклов надо ставить двоеточие.



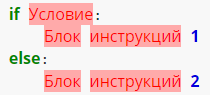
Было бы забавно, если бы я что-то узнал про блок-схемы.



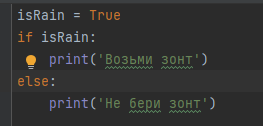
Более подробно о блок-схемах: <http://videoege.ru/informatika/elementy-blok-shem>

А тут можно их составлять: <https://www.diagrams.net/>

Ладно. Вот как это работает на Python

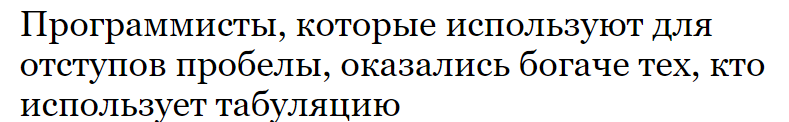


**Напомню:** двоеточие конец условия, отступ после условия. PEP8 рекомендует использовать для отступов пробелы а не табуляцию. А сам интерпретатор python3 категорически не поддерживает смешивание пробелов и табов. Сейчас проверю же.

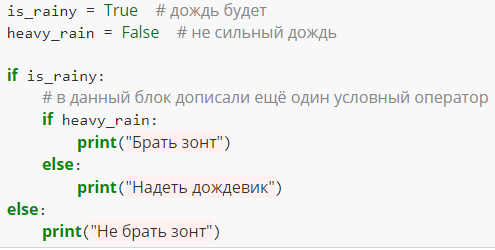


Не знай. Ошибок я не поймал.



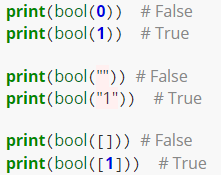


Условие в условии.

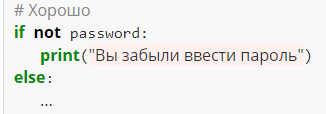
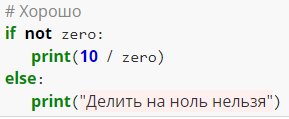


Если внутри условного оператора (**Внутри конструкции if \_\_ :)** Не использовать никаких логических операторов, то интерпретатор **Python** сам осуществит приведение типов к **bool.** Вот по такой схеме он это будет делать:

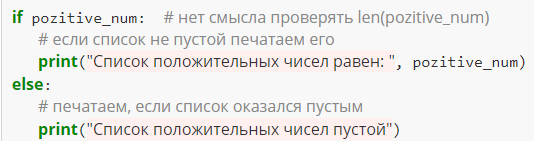


Прувы.

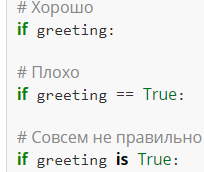
Не много правильных условий:



**Практические примеры с условным оператором**

****

**Не сравнивайте логические типы с True и False с помощью ==, иначе получается «масло масляное».**

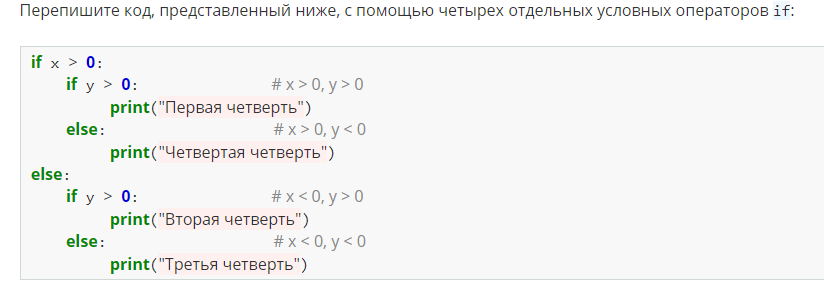
****

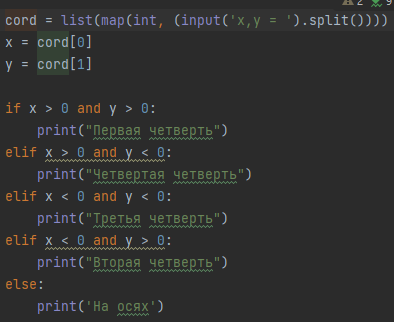
****

****

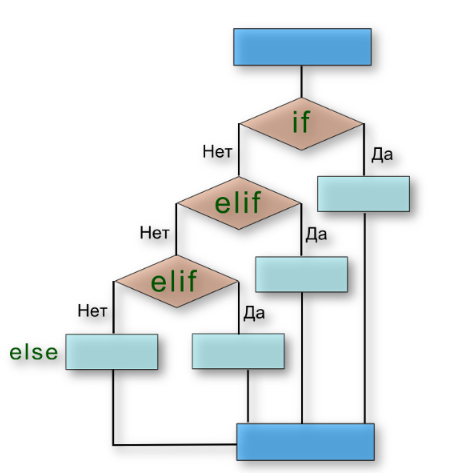
Сами же учили, что чем проще тем лучше. Мой вариант лучше.



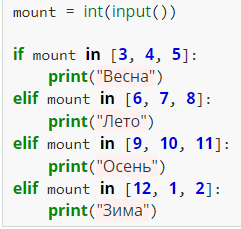


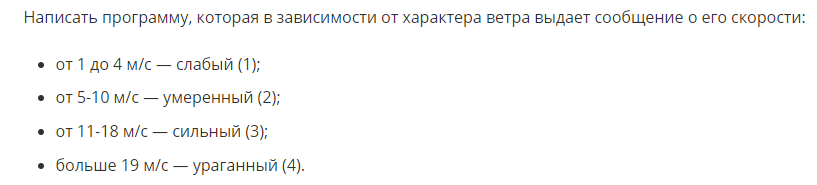


**Конструкция if-elif-else**

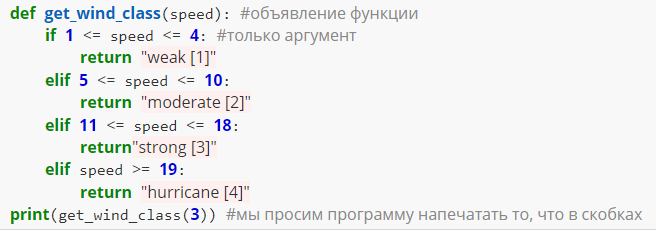


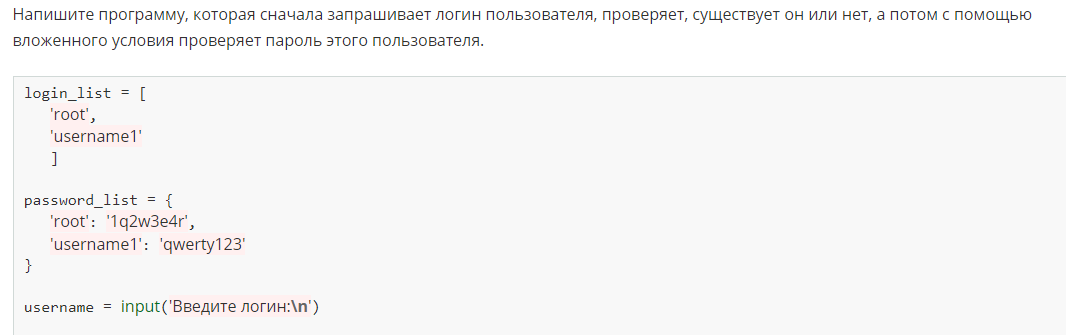
И этого достаточно.



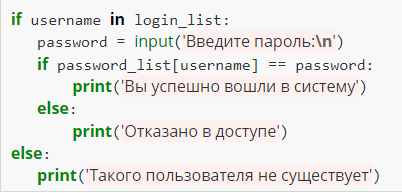
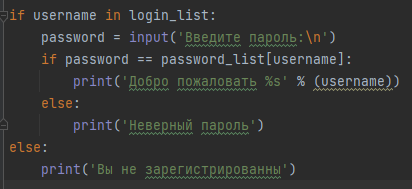


Вот тут даже функция используется:





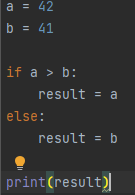
Мой вариант.



Практически без отличий.

**Тернарный условный оператор**

Допустим есть такой код



Во первых меня удивило, что переменная, объявленная в внутри условия доступна и за его пределами это важно понимать.

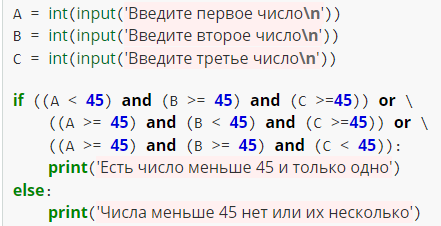
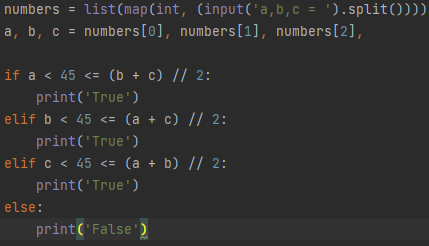
А теперь как написать это в условие с помощью тернарного оператора.



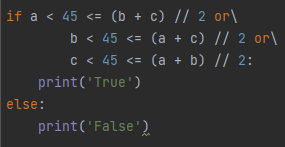
Тернарный условный оператор или инлайновый if, записывается всегда в одну строку.

**Записать условие, которое является истинным, когда только одно из чисел А, В и С меньше 45. Иногда проще записать все условия и не пытаться упростить их.**

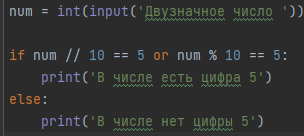
А я так не думаю:

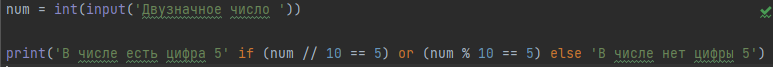


Ну единственное могу свой код переписать, чтобы все было под одним условием так:



**Дано двузначное число. Определить: входит ли в него цифра 5. Попробуйте решить её с использованием целочисленного деления и деления с остатком.**

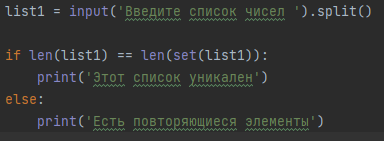
** А если переписать через тернарный?**

****

Не знай… больно длинная строка все же.

**Проверить, все ли элементы в списке являются уникальными.**

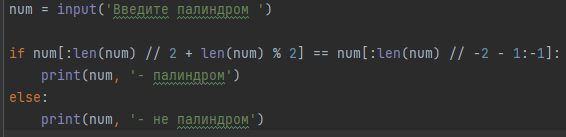
**(Подсказка: используйте возможности set)**



**Дано натуральное восьмизначное число. Выясните, является ли оно палиндромом (читается одинаково слева направо и справа налево).**

**(Подсказка: использовать целочисленное деление и деление с остатком не нужно. Попробуйте преобразовать число к строке, а потом перевернуть эту строку. См. материал прошлого модуля).**

Стало скучно и я написал программу для определния не конкретно 8-значного числа, а любого слова.





Стоп… я проиграл… перемудрил. Я сравнивал первую половину слова с перевернутой второй.

А можно было вообще просто сравнить оригинальное слово с перевернутым. Тут спорить не буду их решение в миллион раз лучше.

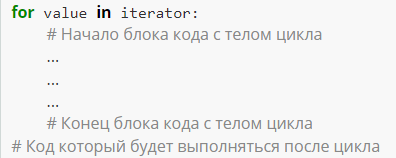


Тут даже длину учитывать не нужно.

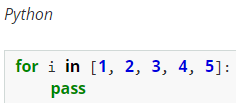
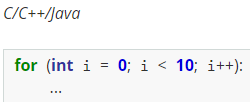
Но мою логику все же можно куда-то применить. Но не для этой задачи.

**Циклы**

Операторы: **for, while, break, continue.**

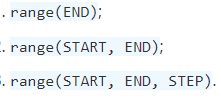


**В отличие от многих других языков программирования, где для цикла for нужно использовать условие для остановки, в Python число шагов цикла ограничено длиной последовательности.**

****

И это не значит что python лучше. Он тут скорее работает как цикл for each возвращая сразу сам элемент. Также нет возможности увеличить шаг или на каком-то этапе не управлять итератором.

В циклах часто бывает полезна созданная последовательность. Чтобы создать последовательность чисел есть функция **range()**. Работает она с такими параметрами:



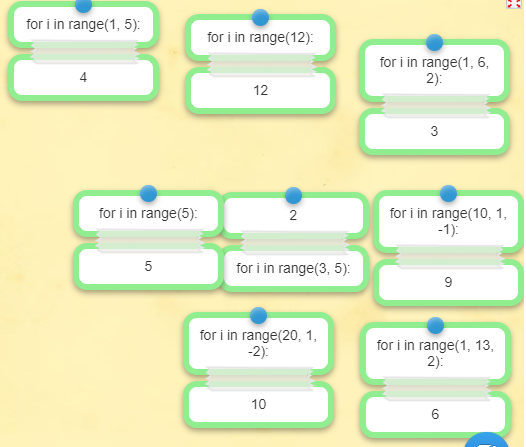


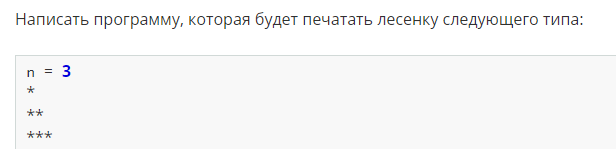


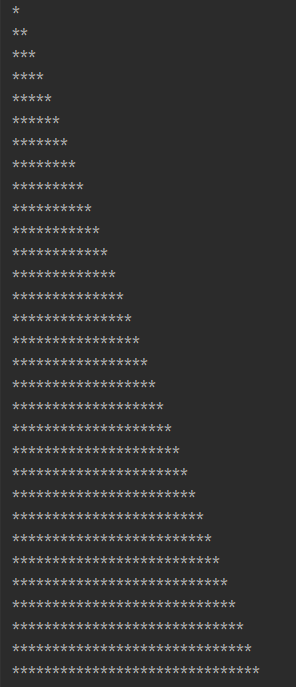
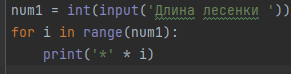
Она возвращает свой класс и уникальный кортеж.

**функция range представляет собой итератор и работает по принципу «ленивых» вычислений. Range не хранит все эти числа сразу в памяти компьютера, а вычисляет новые значения на каждом новом шаге, когда к ней обращается, например, цикл for или функция list.**

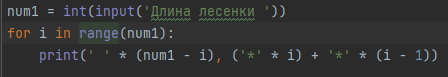
Какой цикл сколько раз отработает.

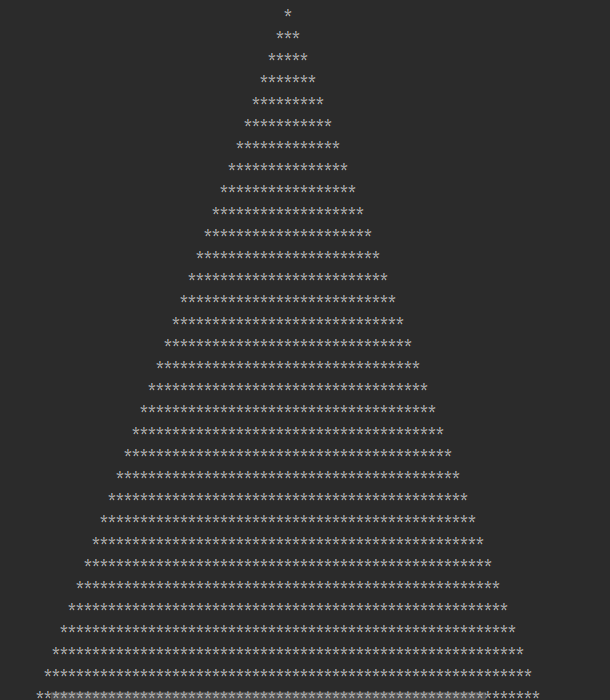




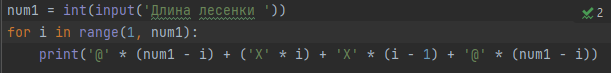


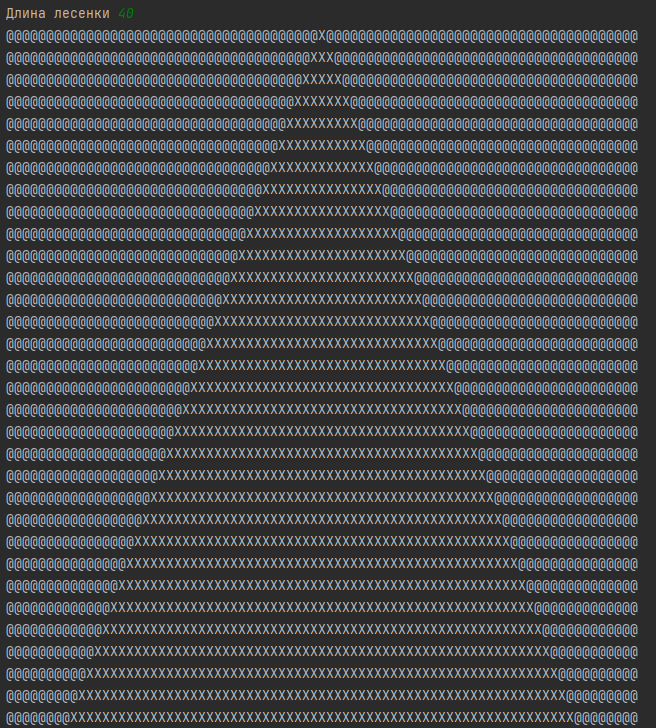
Блин, жопа как захотелось написать такой же, но симметричный.



 Если заменить это числами то…

 То начинаю понимать, что треугольник пифагора нарисовать будет на так просто.



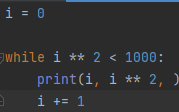


**Цикл while(циклом с предусловием)**

В этом языки им нужно будет пользоваться чаще, так как он дает более гибкие возможности над взаимодействие с ним.

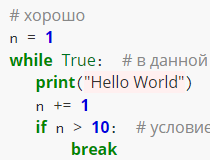
Ну сразу приходит в голову что можно поставить два условия. Первое которое должно выполнится, а второе через или достижение какого-то порога, чтобы избежать ухода в бесконечный цикл.

**Напишите цикл while, который находит максимальное натуральное число, квадрат которого меньше 1000.**

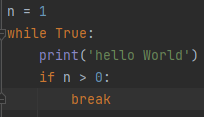


В терминале бесконечный цикл можно остановить через Ctrl + C.

Но цикл можно прервать и внутри тела цикла через **break.**



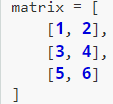
Код выше ведь абсолютно такой же, как и **do while.** Сейчас проверю.

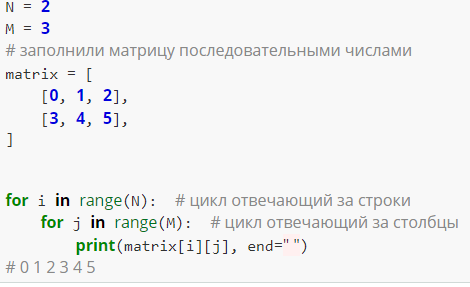


Проверил. Это и есть **do while цикл.** Другого в этом языке нет. Он всегда выполнится минимум 1 раз. **Это цикл с пост условием.**

**Работа с вложенными циклами**

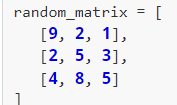
Обработка более сложных структур.

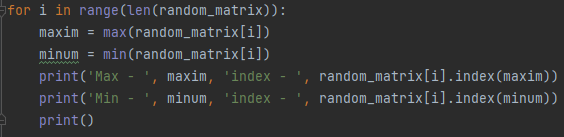


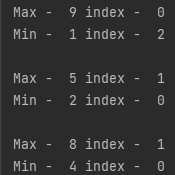


Классика.

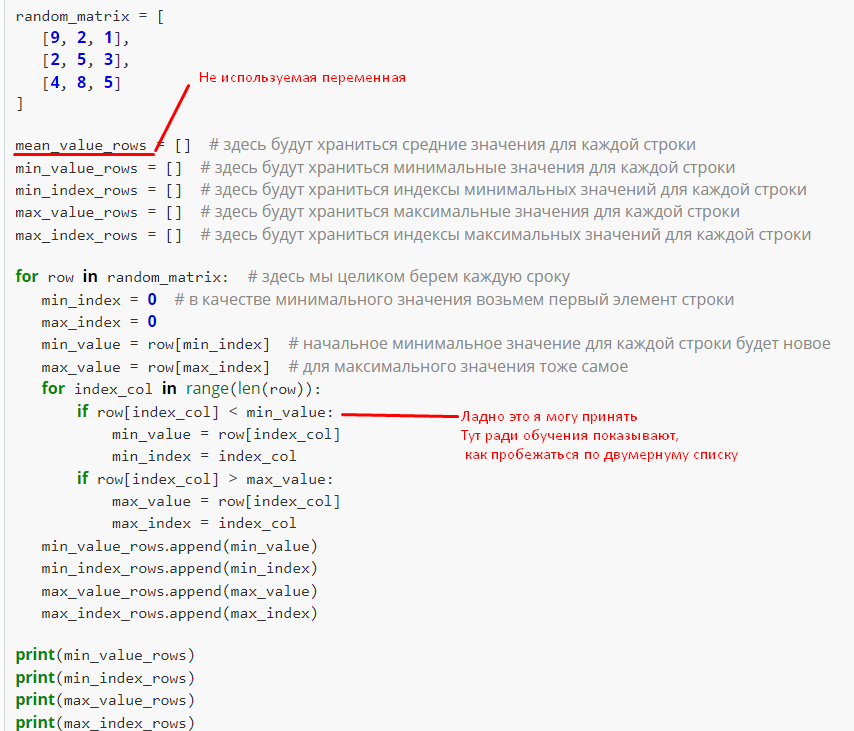
**Условие задачи: Дана двумерная матрица 3x3. Определить максимум и минимум каждой строки, а также их индексы.**

****

****

****

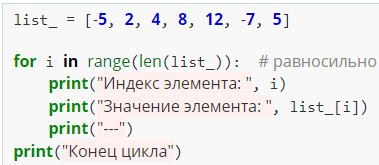
**Пару уроков назад говорил что чем проще тем лучше.. и что я вижу в их коде?**

****

**Практические примеры**

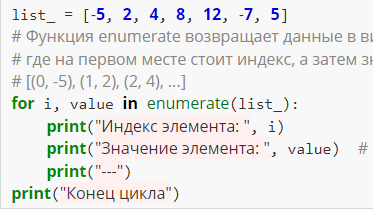
**Enumerate –** Это что такое…

Это типо такой обход списка?



А переводится хоть как… **перечислить**

**А стоп. Я тупой.** Это отдельная функция.

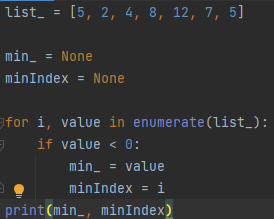
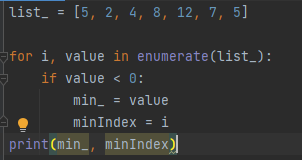


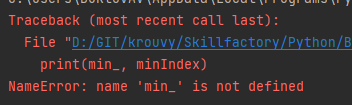
Причем обратить внимание, что в for обращаемся сразу к двум элементам присваивая значения кортежей. Да, через цикл **for** правда удобно рассматривать кортежи. Важно отметить, что первым идет именно индекс, а вторым значение. В JS **for** и **map** наоборот.

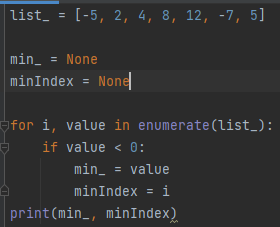
**Начинающий программист написал программу, которая находит индекс последнего отрицательного элемента в списке.** **Но он не знал, что есть функция enumerate. Ваша задача — подправить код так, чтобы он работал с помощью функции enumerate.**

Хорошо, но перед этим объясню почему не стоит объявлять переменные внутри условий. Дело в том, что если условие если они объявлены внутри условия, которое не выполнится, то если в коде ниже не них будут ссылки, то будет ошибка.

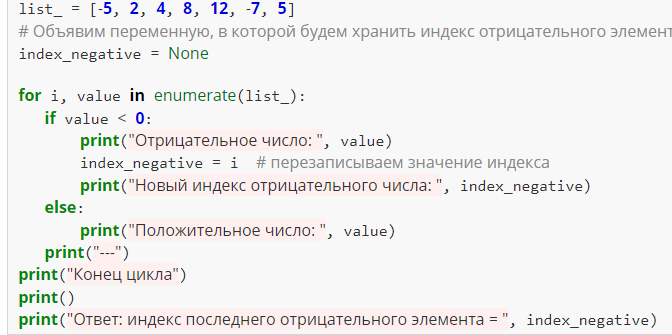
Плохо Хорошо





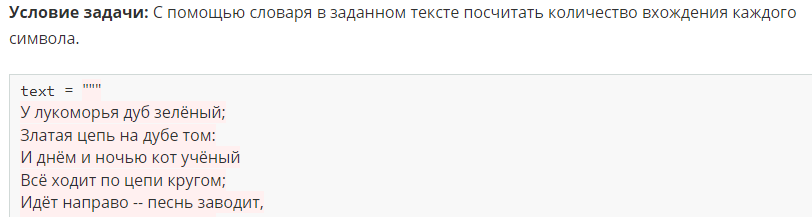


У них программа такая:

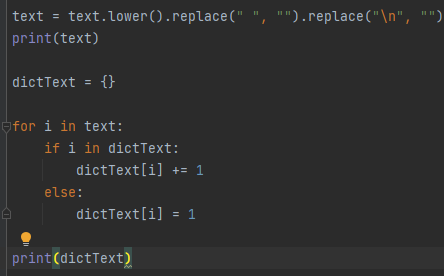


В принципе ничего плохого, тоже самое только больше выводов.

**Цикл for со строками и словарями**



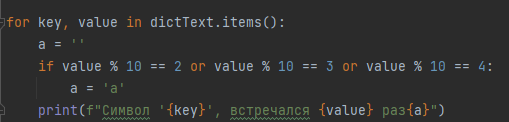
**Хм.. интересно.**

****

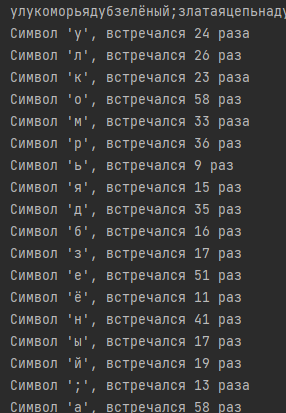
****

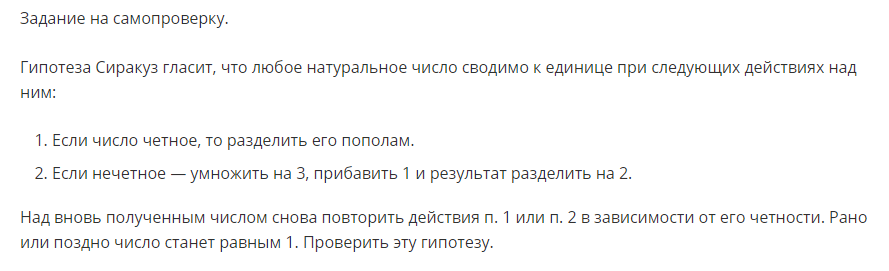
**Опа, есть новая форма строки без %%%** Хотя они тоже могут быть и удобны местами.

Так вот, чуть доработал программу.

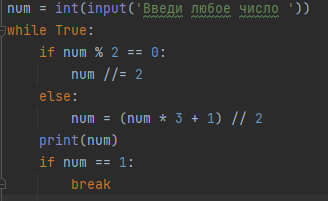


F строка действительно удобна, она как `такие кавычки`, только у нее нужно именно символ использовать, а затем можно прямо в ней описывать переменные. **F** строки работают начиная с **Python 3.6**





Да, гипотеза правдива, но я не понимаю ее суть. По моему второе условие и так бы работало.





**Continue**

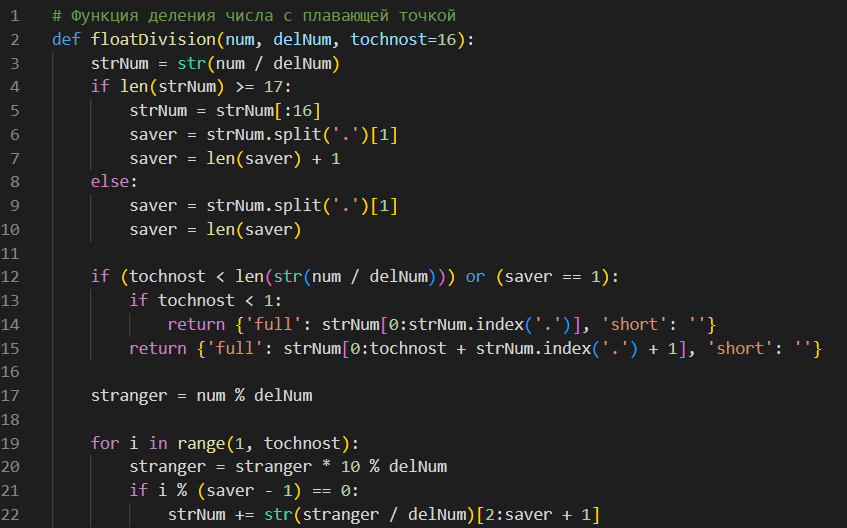
Но перед этим. Я тут по приколу математикой занимался, вспомнил деление. 178/63 = 126/63 + 52/63 = 2.825396(825396)(...) ... = 2 целых 52-е 63-их. И нашел в делении эту последовательность:

[520, 160, 340, 250, 610, 430] все эти числа при делении на 63 дают последовательность 825396

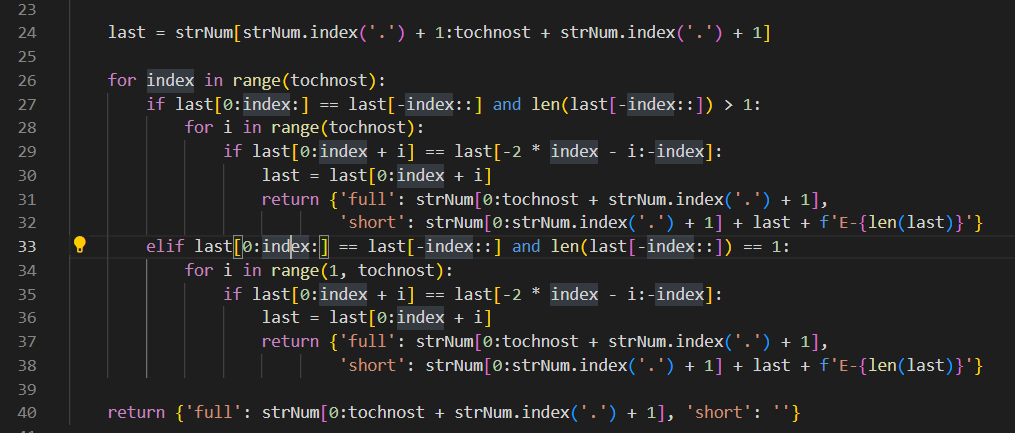
Хочу с этим чуть поигратся.

20 часов спустя… Боже, это было ужасно, как же все не работало. Сначала работало, потом переставала, происходило смещение и тд. Для разных чисел могло работать и нет. Но вот, я все отладил и готов представить этот код:

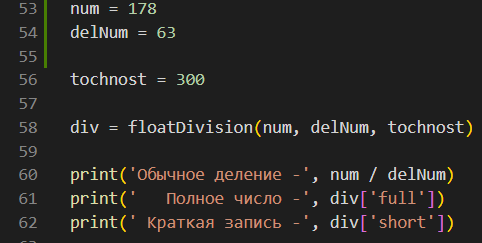
**Формирование полного числа:**

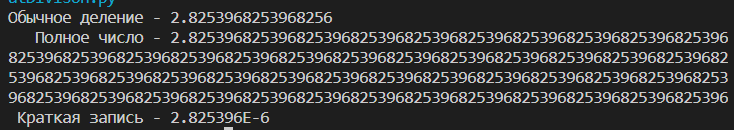


Нахождение краткой записи числа без повторения последовательности



**Вызов функции и вывод:**

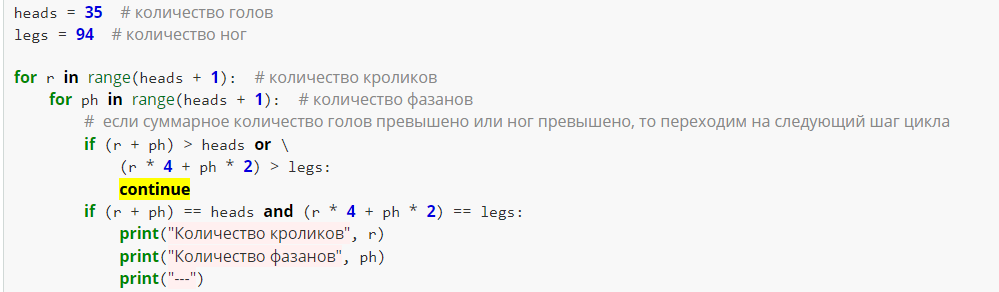


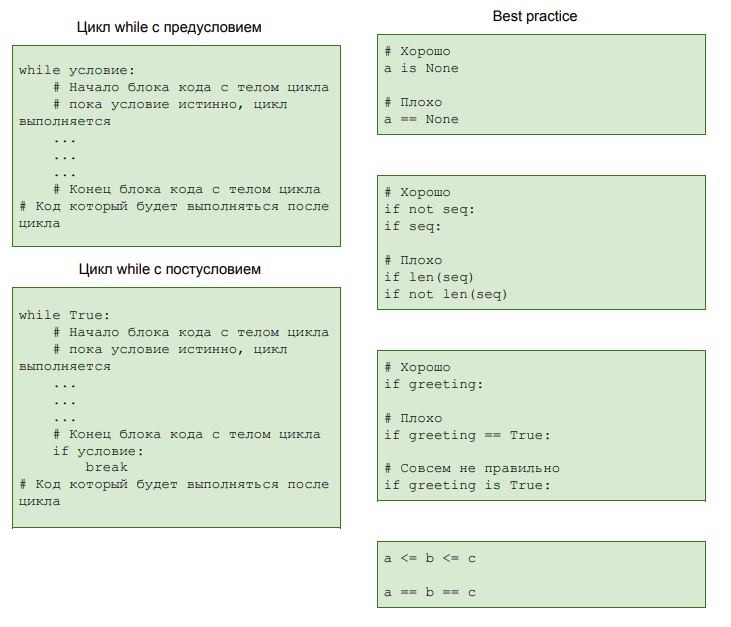
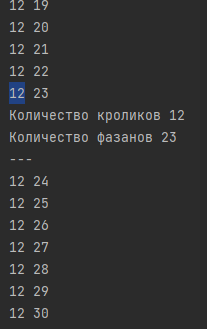


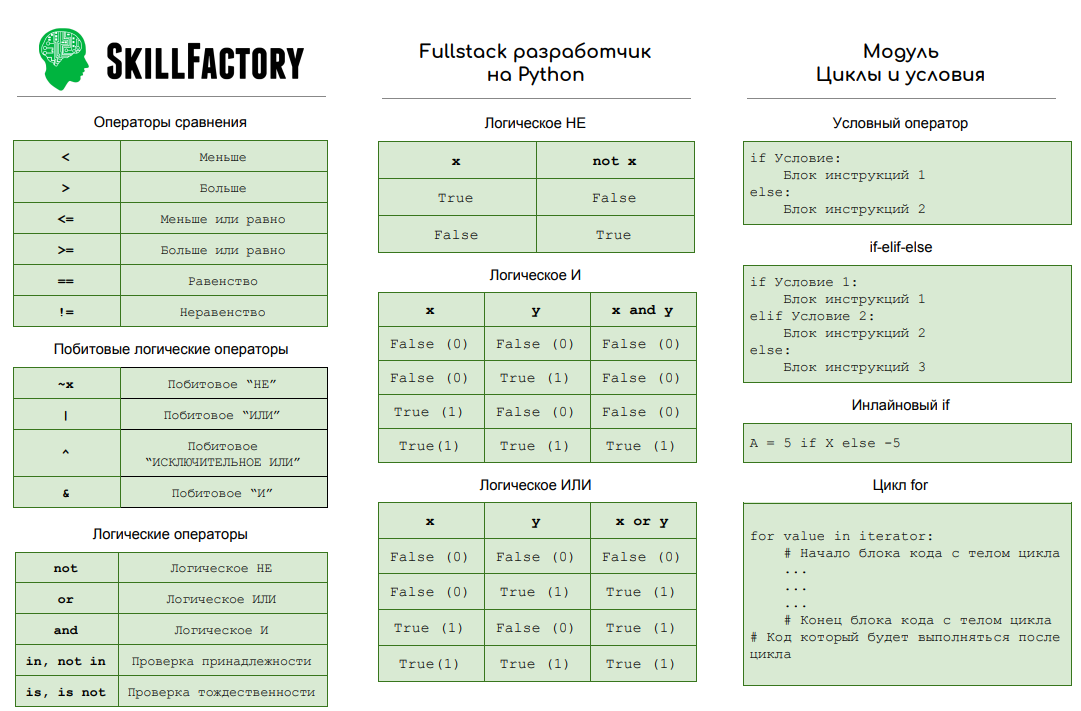
**Так что как-то так, а на чем я там остановился…**

**А continue…**

Ну это прыжок к следующей итерации.



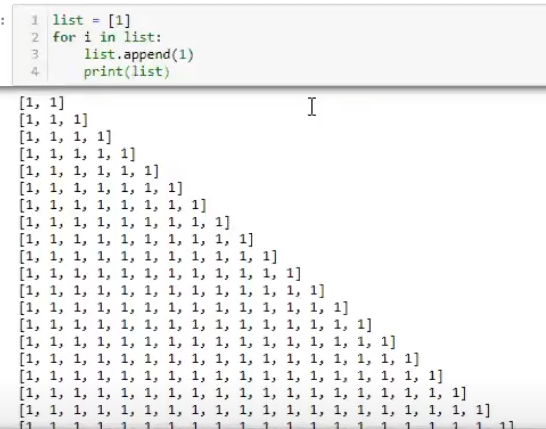




Вебинар еще…

Но может что-то важное пойму.

Цикл **for** можно сделать бесконечным.



Функция **ord()** Возвращает числа ascii для символов.