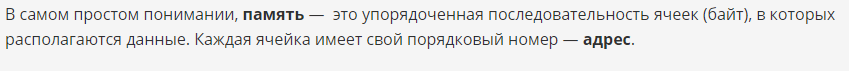
**B2**

В python существуют изменяемые и не изменяемые типы данных. Что будет если попытаться его изменить. Объекты типа словарь и список.

Ладно. Заводим старую шарманку. Переменные…



Хотя так интереснее.. память.



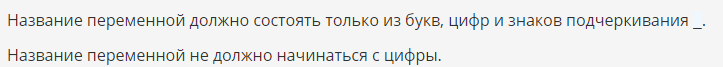


Раньше вот не было переменных, и приходилось запоминать в какой участок памяти было сохранена информация. Переменные появились именно для решения этой проблемы. Хотя в низкоуровневом программировании по прежнему встречается такая работа с памятью.

Закреплю. Переменная – именованная ячейка памяти.

**Да.. да.. да. Оператор присваивания,**

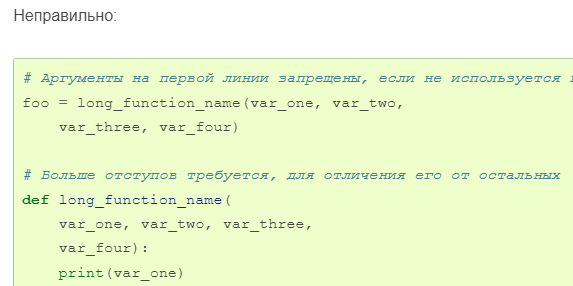
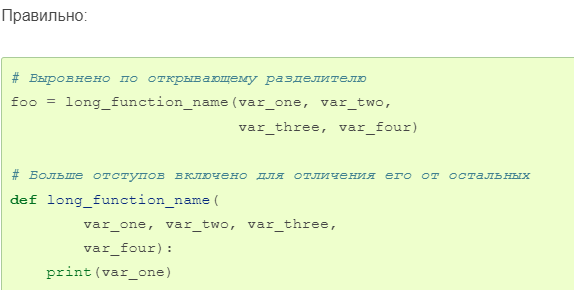
****

****

**Ладно прочитаю это чертово PEP8**

[**https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id17**](https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html#id17)

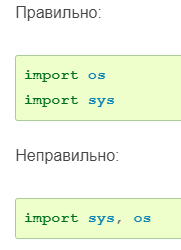
Все же с правилами написания именно в **Python** я не знаком.

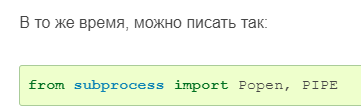


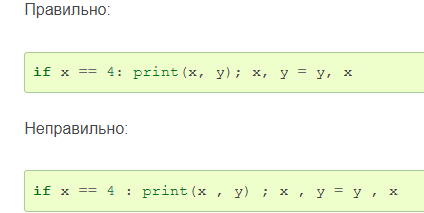
**Ограничьте длину строки максимум 79 символами.**

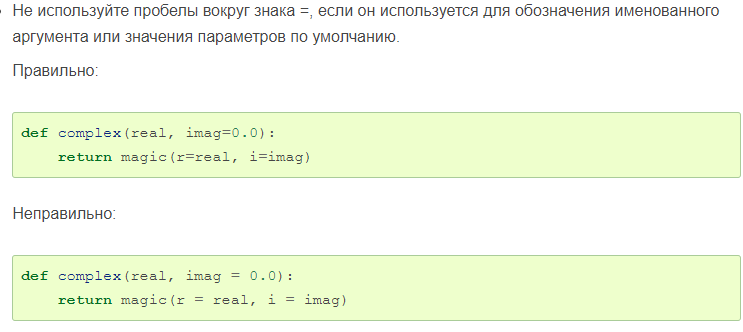
**Отделяйте функции верхнего уровня и определения классов двумя пустыми строками.**

**Определения методов внутри класса разделяются одной пустой строкой.**

****

****

****

****

**Ладно, продолжим.** В Python есть множественное присваивание.



Также можно и поменять значения переменных местами.

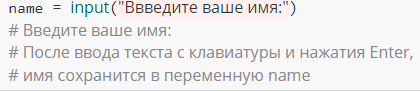


**Функции…**

**Аргументы — это данные, которые необходимы функции для работы.**

**Ввод с консоли, input()**

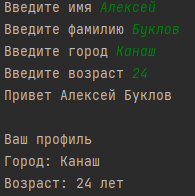
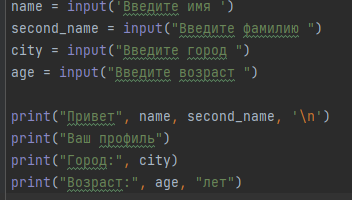
Эта функция для ввода данных. В качестве параметра может принимать текст с подсказкой. После выполнения ожидает нажатия на enter от пользователя.



Возвращает всегда строку, так что если ожидаются числа, то их нужно перед работой преобразовать.

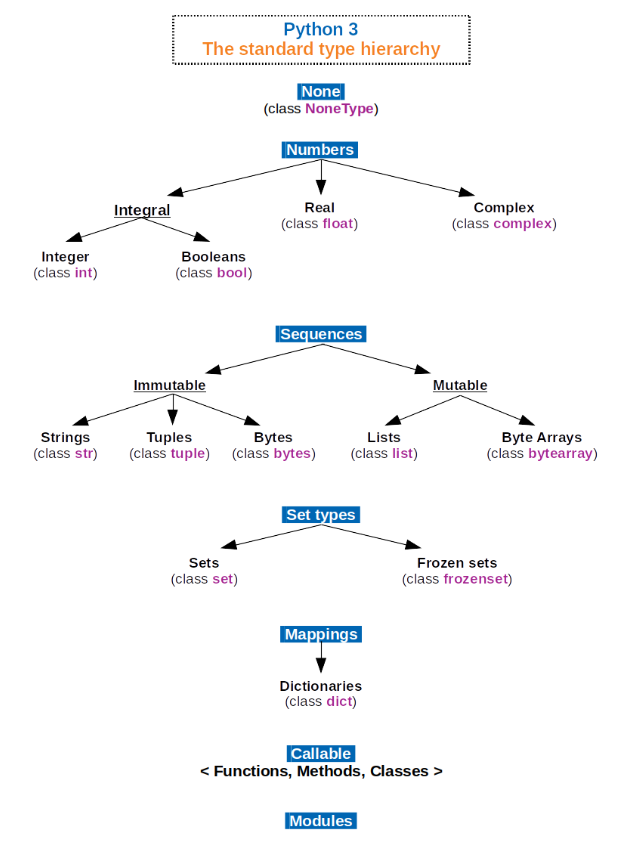
**Вывод в консоль, print()**

Ну тут все очевидно. Можно без комментариев. Ладно… напишу одну программу просто потому что положено.

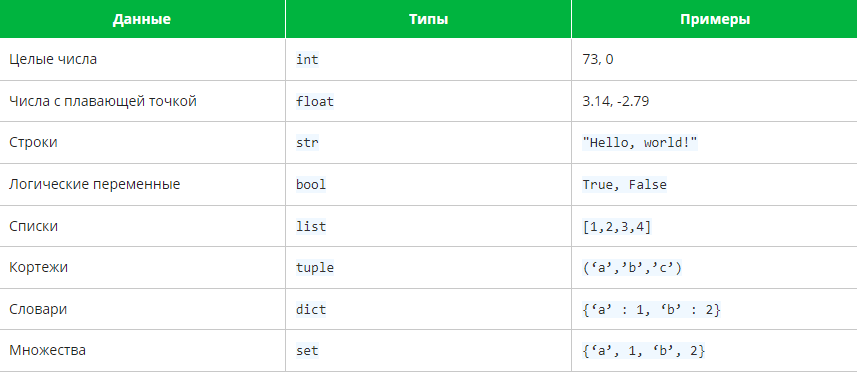


**Неизменяемые типы данных**

Вот типы данных



Каждая единица данных в Python является объектом какого-то типа. Типы данных могут быть не только встроенными в язык, но и загруженными из библиотек или созданными самостоятельно.



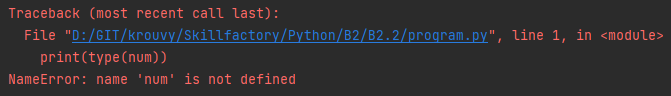
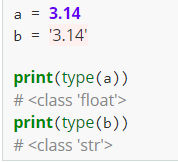
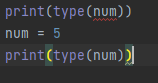
**Преобразование типов.** Ну допустим преобразовать число в строку или обратно, если строка является числом.

**Python — язык с неявной сильной динамической типизацией. Разберем это определение по частям.**

Динамическая типизация свойственна интерпретированным языкам. В них тип переменных определяется в момент присваивания значения, а не во время компиляции. Поэтому и не нужно перед именем переменной писать ее тип данных (**неявная типизация**).

**Сильная типизация нам говорит о том, что мы не можем совершать операции над объектами разного типа без приведения их к одному типу. Например, Python вызовет ошибку, если мы попытаемся сложить число и строку.**

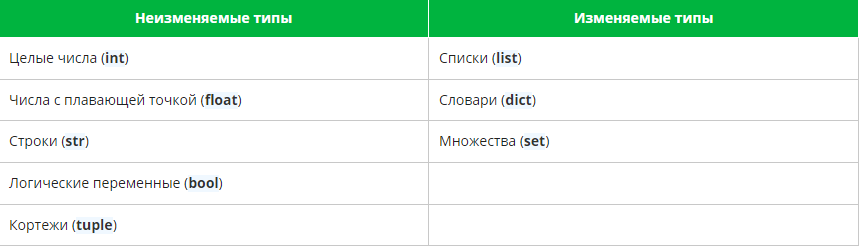
Чтобы узнать тип переменной можно воспользоваться функцией **type()** Интересно, а что если в нее засунуть еще не объявленную переменную. Короче.. Будет ошибка.



А без первой строки



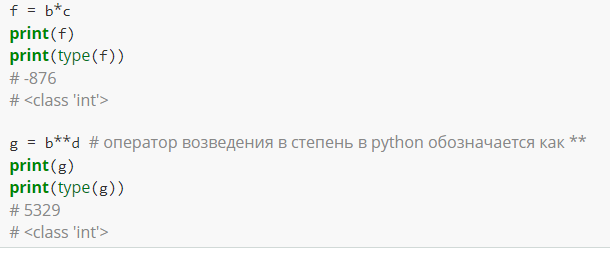
Так вот. Изменяемые и неизменяемые типы данных.



Неизменяемые типы не могут быть изменены на лету. Например если захочется изменить строку, то будет создана новая строка с внесенными изменениями.

А изменяемые типы можно менять как хочешь, через пуши, поп или изменять значения по индексам.

При работе с **целыми числами** можно использовать все математический операции и тип данных переменной не изменится(**о делении чуть позже**).

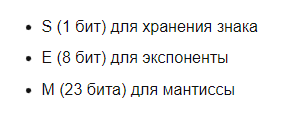
****

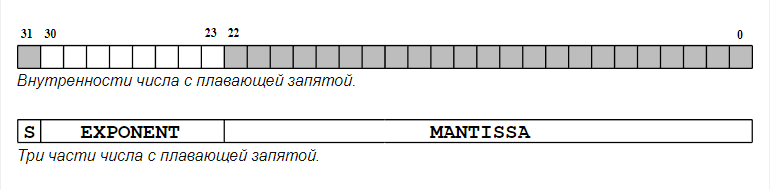
**термин «плавающая точка» или «плавающая запятая» происходит от особенностей хранения таких чисел в памяти компьютера.**

<https://habr.com/ru/post/337260/> Наглядное объяснение чисел с плавающей запятой

Боже…  экспонент с мантиссами.

32 бита разделены на 3 части.

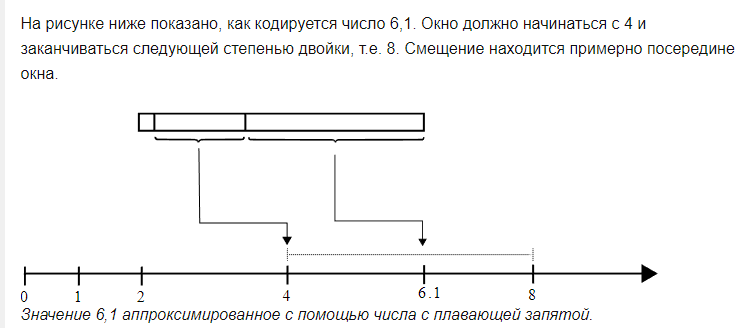


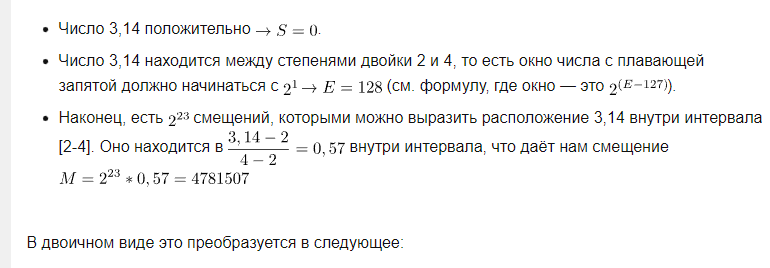


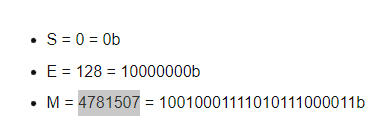
Воспринимайте экспоненту как окно (Window) или интервал между двумя соседними целыми степенями двойки. Мантиссу воспринимайте как смещение (Offset) в этом окне.

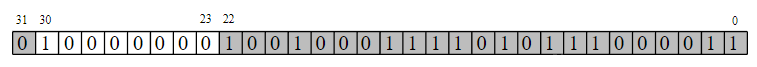


Окно сообщает нам, между какими двумя последовательными степенями двойки будет число: [0,1], [1,2], [2,4], [4,8] и так далее (вплоть до 





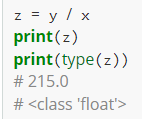


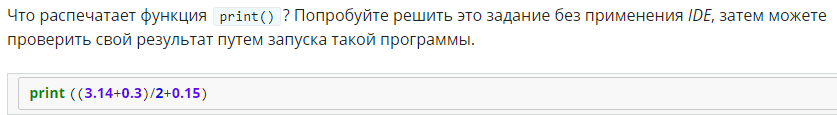


Хорошая статья, но сложная.

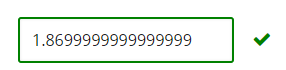
В **python** десятичный разделитель для чисел с плавающей точкой это точка.







Ха. Хрен угадать что должно быть в ответе. Потому что если посчитать руками, ответ будет **1.87**. Но так как тут работают хз если честно почему. Но ответ от python получается такой.



**Обмен значениями**

**Наличие арифметических операций для численных типов данных (как float, так и int) позволяет совершить обмен данных между переменными без создания временной переменной и не используя множественное присваивание, с которым мы знакомились в прошлом юните.**

Не понял что тут именно имелось ввиду.

**Строки**

Для записи в переменные типа **str** используются кавычки, **‘’ “”**

****

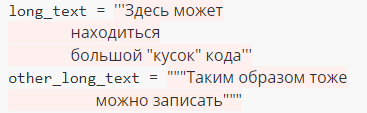
Их удобно совмещать если в самой строке имеются кавычки



Так можно, а так нельзя:

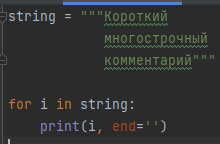
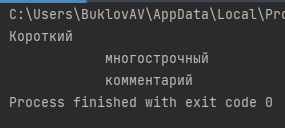


Есть еще вариант с тройными кавычками. В них можно вообще не беспокоится для кавычки.



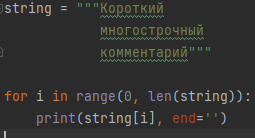
Такую форму записи рекомендуют чтобы оставлять многострочные комментарии. Но **PEP8** не рекомендует.

Как уже говорилось у строк можно пробегаться по элементам, получая их значение по индексам. Интересно что из себя представляет такая строка, которая содержит переносы на другую строку. Напишу код:

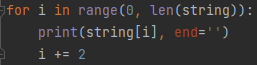
 

Ожидаемо. 

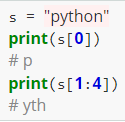
Ну можно воспользоваться и более классическим циклом.



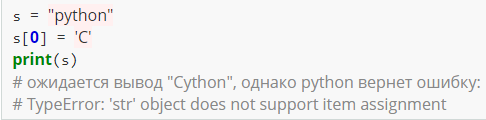
Напомню, даже так, внутри цикла **i** изменить не возможно, **оно не изменится.**



Кстати, тут также доступны срезы:



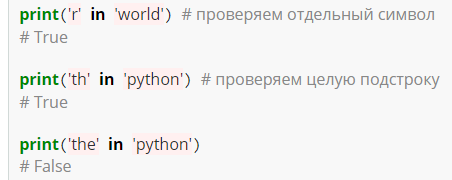
**И разумеется изменить строку по индексу мы не можем**

****

**Логические переменные:**

**True, False.** Все.

**содержится ли какой-то символ в строке:**

****

**Кортеж**

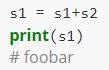
Ну тут не помню. Поэтому подробнее. Чтобы создать **кортеж** нужны круглые скобки. В нем мы можем хранить разные типы данных.



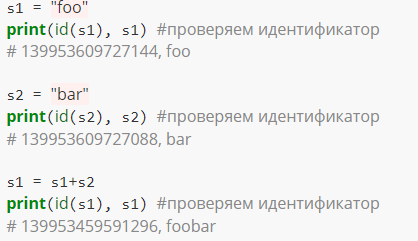
Но кортеж все равно является **неизменяемым типом.**

****

**Так вот, почему строки неизменяемый тип:**

** **

Кажется что переменная просто изменилась, но на самом деле это уже совсем другая переменная. Это можно отследить с помощью функции **id(),** которая возвращает уникальный идентификатор любого объекта.

****

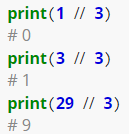
**Типы данных: число, число с плавающей точкой**

Операция деления для целых чисел имеет свои особенности.

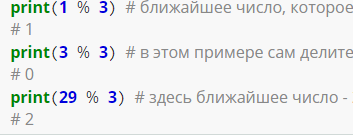
**Обычное деление:**



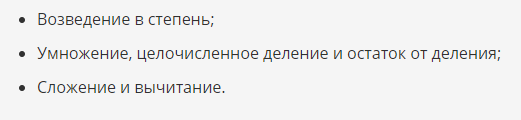
**Целочисленное деление**

****

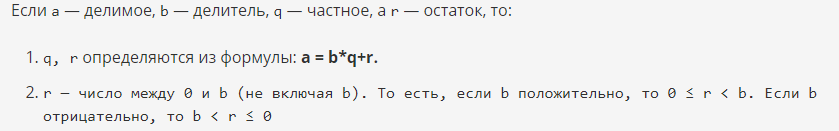
**Остаток от деления**

****

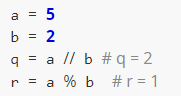
**Порядок выполнения операций.**

****

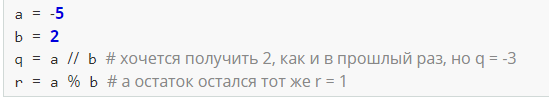
**Отдельно отметим особенность операций целочисленного деления и нахождения остатка от деления в случае отрицательных чисел.**

****

**Нихера так не понятно.**

****

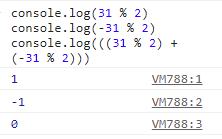
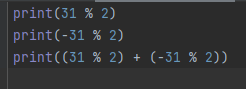
**5 = 2\*2+1**

****

**-5 = 2 \* (-3) + 1**

Просто идет сокращение в другую сторону.

Блет. Это работает иначе в этом языке. Вот один и тот же код на JS и на Python

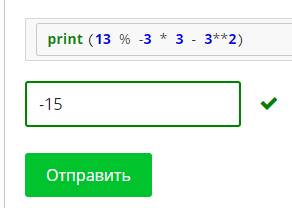
И вот еще



Путаница жопная. Вот по формуле

**13 = -3 \* (-5) + (-2)**

****

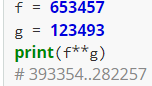
**Вот это пиздец. Ладно 13 % -3 это -2. Хорошо. Но -3^2 не поменяло знак, это пиздец.** Да сказали, что эта операция выполняется раньше, но то что она не учитывает знак мега тупо.

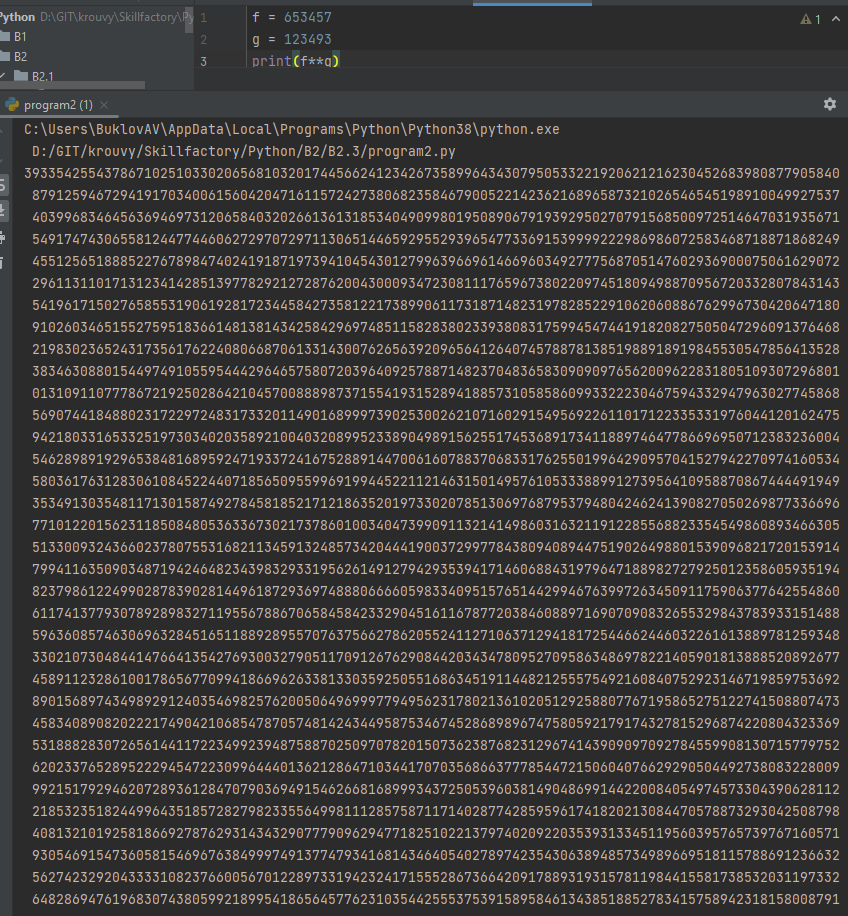
Это пиздец.

**Длинная арифметика**

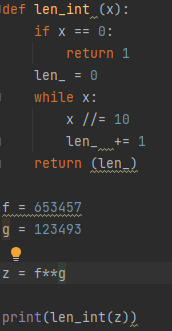
Плюс целых чисел в их бесконечной точности. Записаны могут быть сколь угодно большие числа.



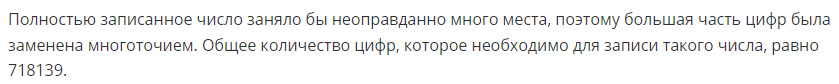
Да, это круто.



Для интереса решил узнать какой длины это число.



**718139**



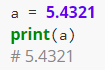
Клянусь. Я решил узнать эту длину еще до того, как прочитал эту запись.

Да выполнялся этот код долго, но в принципе он выполнился. То что **python** вообще поддерживает такую арифметику, эту круто.

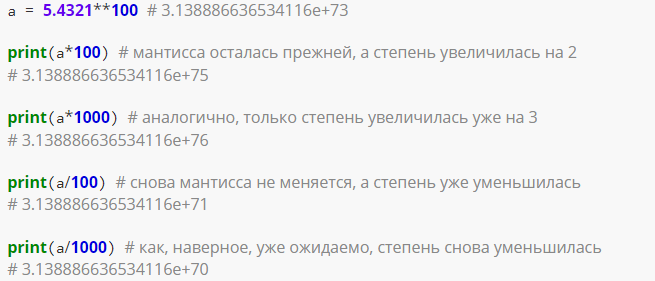
**Например, в языках C и С++ существует несколько встроенных типов для представления целых чисел. Эти типы различаются количеством памяти, которое выделяется на переменные такого типа. В зависимости от этого, переменная имеет ограничение на величину числа, которое можно записать. При попытке записать число, выходящее за границу типов, возможны непредсказуемые последствия.**

**Числа с плавающей точкой**

**Десятичная форма.**

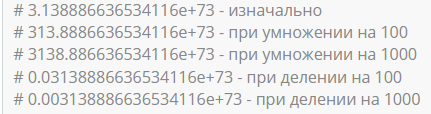


**экспоненциальная форма**

****

**Дело в том, что в памяти компьютера мантисса («само» число, его значащие цифры) хранятся отдельно, а показатель степени числа — отдельно. Благодаря этому, умножение на степени числа 10 никак не влияет на мантиссу, прибавляя (или вычитая, в случае деления) соответствующую степень десятки к показателю степени множителя.**

Не понятная дичь, читал про это. Но вот объяснение.



**Точка перемещается влево или вправо по разрядам числа, и поэтому её называют плавающей. А если подходить к этому ещё более формально, можно сказать, что числа с плавающей точкой обладают точной относительной точностью, но изменяющейся абсолютной. Иначе говоря, точность числа содержится в мантиссе (значащих цифрах), а порядок числа становится не важен.**

Числа с плавающей точкой не обладают бесконечной точностью из-за способа хранения.

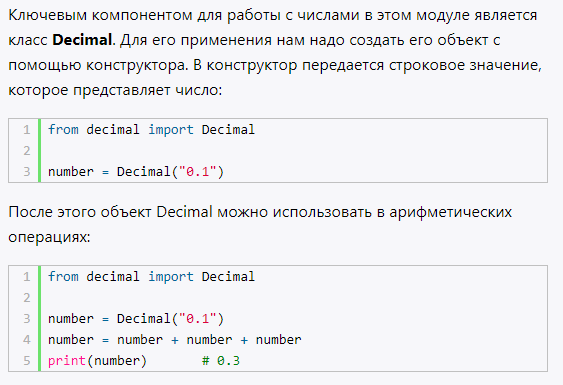
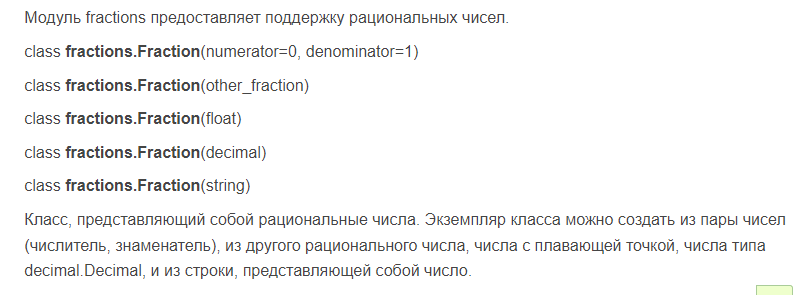
Хм… Что за бред. Почему не 3.6?

**после 16 знака после запятой вычисления дают сбой. Хранения числа используется ограниченное количество памяти (в байтах), которых хватает только на то, чтобы корректно вычислить 16 знаков.**

Если необходимы точные вычисления, то можно воспользоваться классами**. Fraction и Decimal.**

<https://pythonworld.ru/moduli/modul-fractions.html>

<https://metanit.com/python/tutorial/6.4.php>



**Преобразование типов**

Когда целые числа делятся, на выходе получается число с плавающей точкой. Или если дробное число умножить на целое на выходе все равно будет дробное, даже если результат будет целым:

** **

Операции с числами, по крайней мере, проходят.

Но по мимо неявного преобразования, можно еще сделать явное.

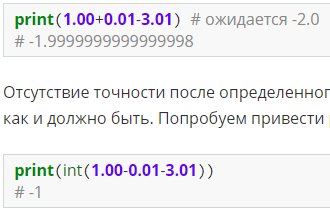
****

****

**Но отмечу, преобразование не округление. Для этого есть функция.**

****

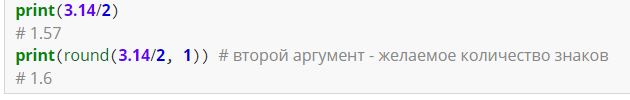
**Да, из этого может пойти проблема.**

****

Кстати, вот функция для округления.



В качестве аргументов принимает число **и количество знаков.**



**Типы данных: строки**

Однако необходимость обрабатывать строки, и в том числе их изменять, иногда появляется.

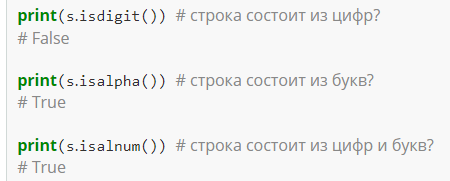
У срезов есть такой шаблон **[начало:конец:шаг],**

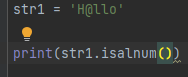


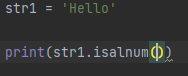
Полезные функции

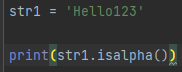
Встроенная функция **len()** позволяет узнать длину строки

Метод **find(substr),** определенный для строк, позволяет находить символы. Возвращает индекс первого подходящего







.

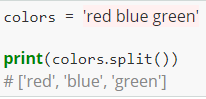
Методы для работы с регистрами.



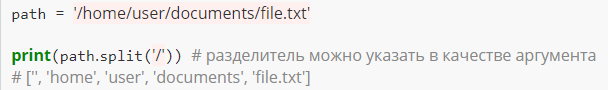
Эти методы не влияют на саму строку. А возвращают измененный вариант с другим id.

Все встроенные функции строк: <https://pythonworld.ru/tipy-dannyx-v-python/stroki-funkcii-i-metody-strok.html>

**Разбить строку на список.**



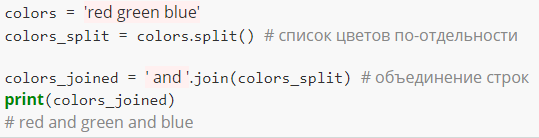
Разделитель по умолчанию это пробел, но это может быть что угодно.



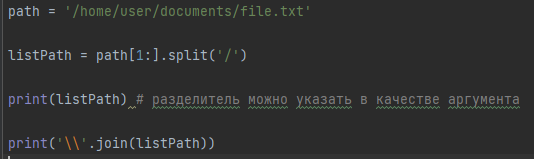
Тут в начале списка первый элемент пустой, потому что в строке разделитель самый первый символ. Чтобы это обойти можно, например обходить не всю строку, а срез после первого элемента.



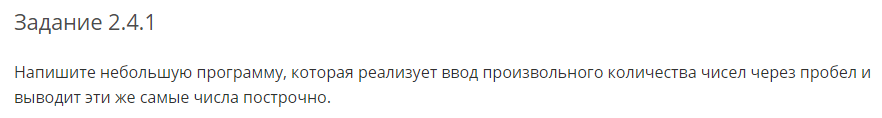
Обратный этому методу это **join.**

****

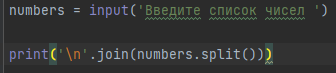
Хотя он слегка странный. Обычно разделитель кладут в скобки, а уже к самому списку применяют этот метод на да ладно.

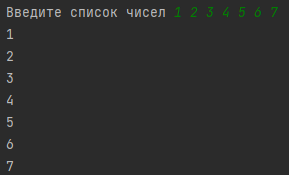




В конце используется два \\, так как он является спец. Символом и один не работает.

Забавное.



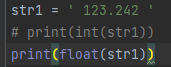


**Преобразование строк**

Если строка состоит только из чисел, то ее можно преобразовать в число.

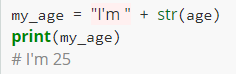
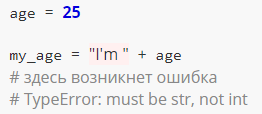


Пробелы проигнорировались.

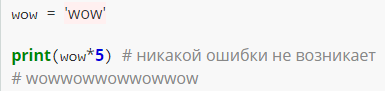


**Int()** может игнорировать пробелы, но не точки.

А чтобы преобразовать число в строку, есть метод **str().** Без преобразования будет ошибка.

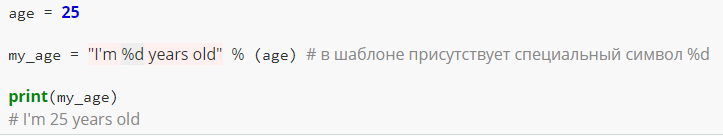
****

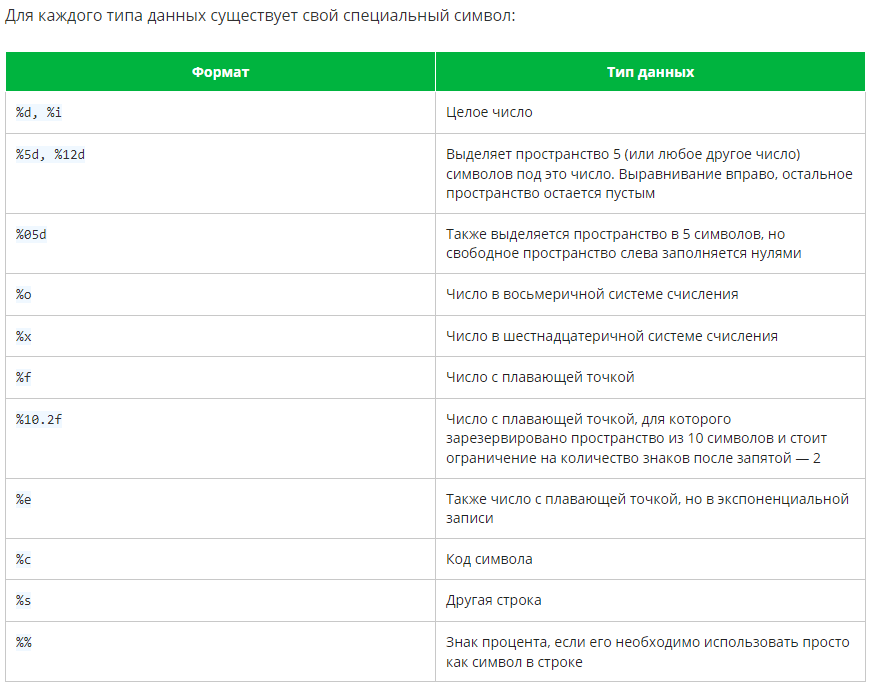
Но ошибки типа не будет возникать есть строку просто умножать.



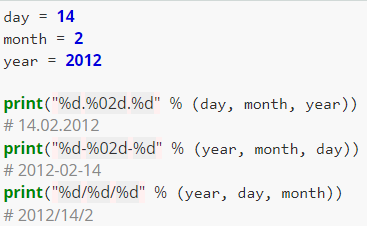
**Форматирование строк**

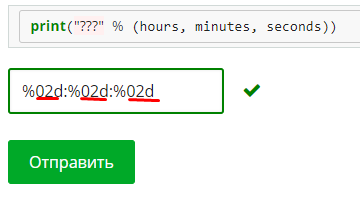
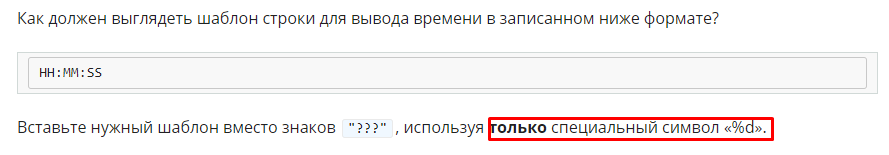
****

****%d шаблон для целого числа

****

Действительно удобно для вывода дат





Иди нахуй.

**Изменяемые типы данных**

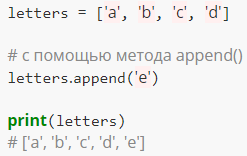
Три основных изменяемых элемента. Списки, словари и множества. Эти типы данных могут изменять свою структуру не порождая новые объекты.

**Списки**

Cписок — это упорядоченный набор объектов произвольного типа. По факту похож на Кортеж. Но уже относится к изменяемому типу данных.



Чтобы добавить к списку новый элемент можно воспользоваться методом **append()**



Список упорядоченная структура, поэтому по его элементам можно пробегаться с помощью индексов.

Если попытаться получить элемент списка, за рамками его длины будет ошибка.



А так есть 2 способа получить последний элемент списка:

**Letters[len(letters)-1]**

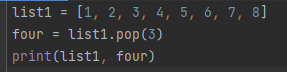
**Letters[-1]**

И в **python** предпочтительнее второй вариант, так как чем проще, тем лучше.

Помимо добавления элементов в список их можно и вытаскивать.

**Pop() remove()** Разнится только в том, что **remove** ничего не возвращает.

Если использовать этот метод без аргументов, то удалится последний элемент, а если использовать аргумент, то он выступит в качестве индекса удаляемого элемента. Также метод **pop()** возвращает то, что он удаляет.





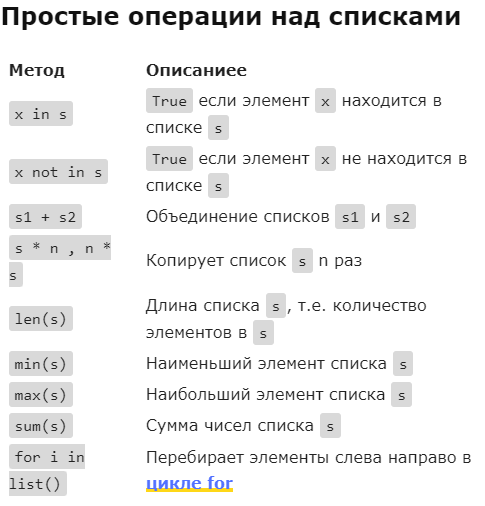
Здесь подробнее о списках.

<https://pythonru.com/osnovy/spiski-python-funkcii-i-metody-python-list>

Пару методов:

Insert() вставляет элемент в указанный индекс со сдвигом.

Extend() помогает сложить 2 списка. Хотя их можно складывать и так [1,2] + [3,4] = [1,2,3,4]



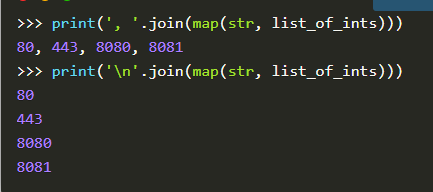
Интересный факт. Когда преобразовываешь список в строку, он преобразовывается вместе со скобками.

Но у строк есть метод для удаления символов.

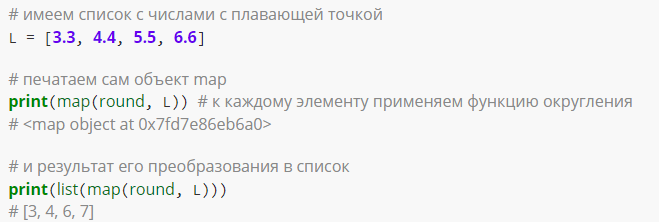


Напомню себе про функцию map, с помощью которой можно применить функцию ко всем элементам объекта.

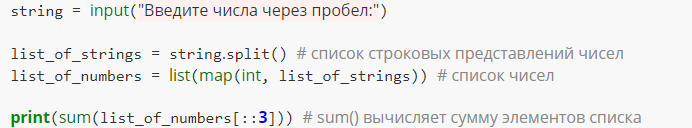


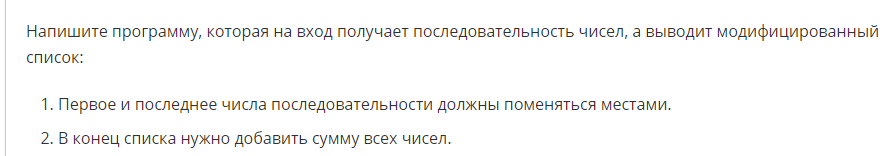
**Заметка, когда пробегаешься по срезу с отрицательными индексами нужно указать отрицательный шаг. Обязательно!**

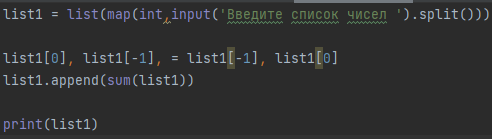
Еще раз про **map.**



**Классическая конструкция ввода.**





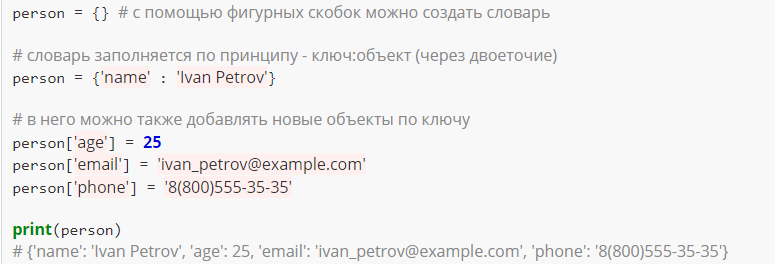


**Словари**

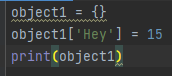
Для этой цели в Python предусмотрены словари (dict) — упорядоченные наборы объектов, доступных по ключу. Иными словами, словарь — это совокупность пар ключ-объект.

Сам объект, который хранится в словаре, может быть любым. Даже другим словарём. Ключ может быть только объектом неизменяемых типов данных, т.е. числом, строкой или кортежем.

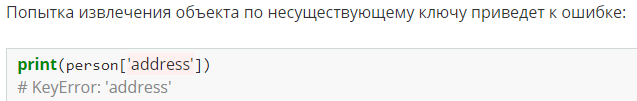
Ну пока типичный объект.



Только без методов. Интересно можно ли добавить сюда новое поле без методов? Наверно нельзя.



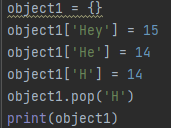
О… можно. Шикарно. А стоп… наверху в коде он же сделал тоже самое…



Методом **keys()** можно получить список ключей. А методом **values()** значения.

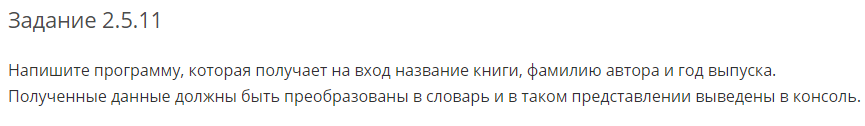


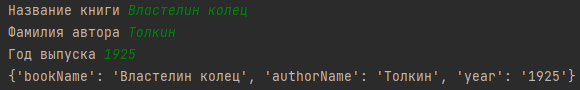
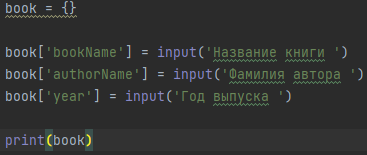
Метод **pop()** вместо индекса требует ключ.



К словарю в принципе нельзя обращаться по индексам.

У словаря нет метода **remove,** только **pop()**

****

****

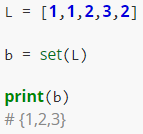
У меня есть любимая задачка, создать список, который будет заполнен случайными значениями в заданном диапазоне, но без повторов. Кажется можно упростить себе решение этой задачи через **множества.**

**Множество** — это неупорядоченный набор уникальных элементов. Иными словами, во множествах не могут повторяться элементы, а хранятся они в памяти компьютера в произвольном порядке.

Множество выглядит точно, как словарь. Но у него нет значений, будто только ключи.

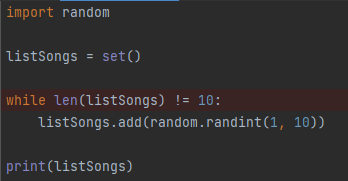


Можно создать множество из списка. Причем если в списке были повторы, они добавляться не будут.



Пустые скобки {} создают пустой словарь. Чтобы создать пустое множество нужно воспользоваться методом **set()**

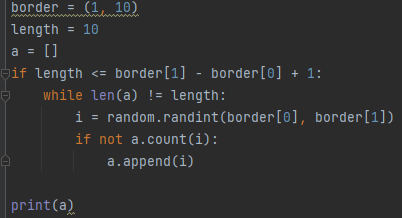
Попытался написать свою любимую прогу этим методом.



Но я не учел, что множества всегда сортируются. Поэтому, ответ всегда был такой:



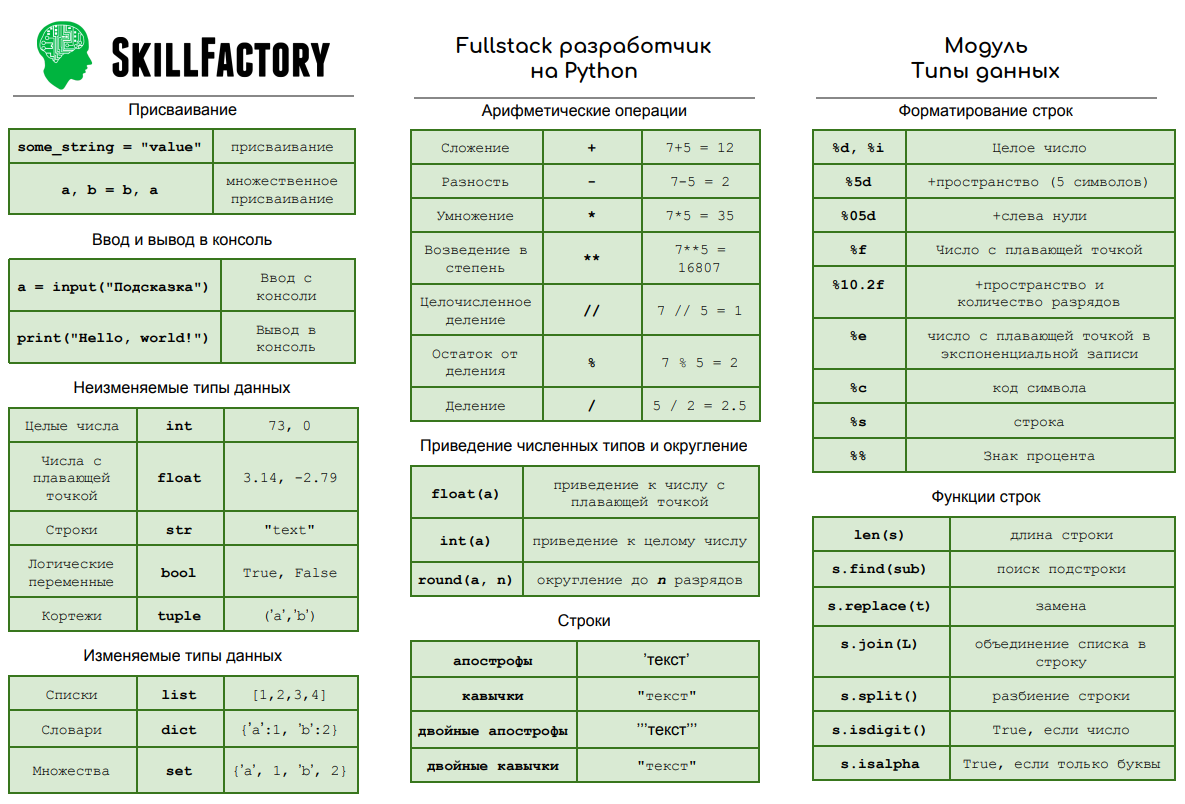
Короче для этой задачи лучше воспользоваться любым другим способом. Например вот в интернете такой увидел:

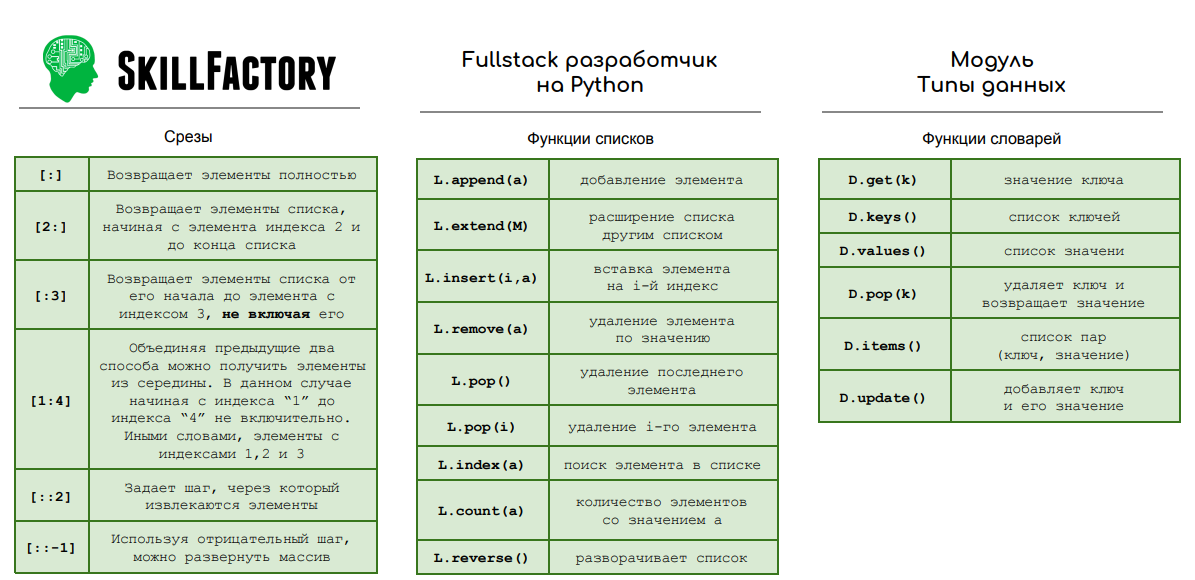


Множества в Python аналогичны математическим множествам. Поэтому для них существует несколько операций.

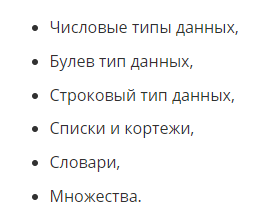


Вот маленький конспект под итог.

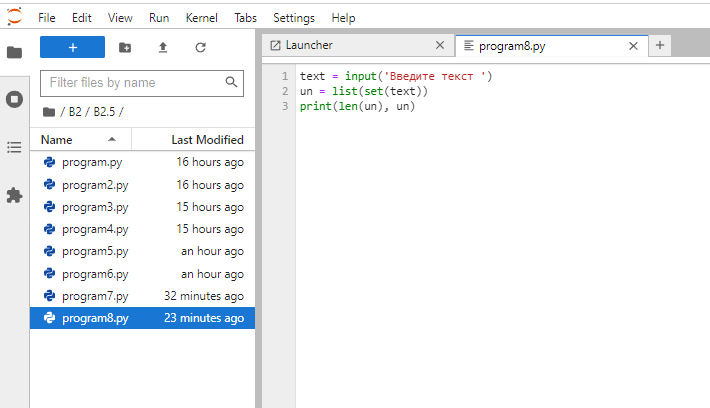




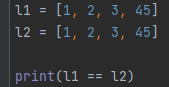
Эх… чертов вебинар на 2 часа…



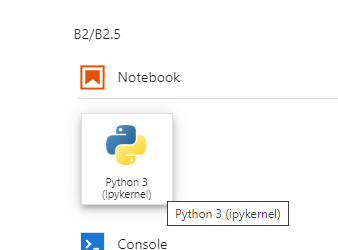
Вебинар чел сразу советует установить **jupyterlab.** Ладно. Установил в интерпретатор. Если често установилась вместе с них дохера всего.



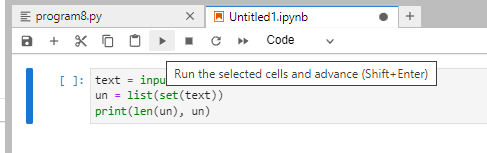
О… в **Python** можно спокойно сравнивать списки.



А насчет **Юпитера.** Можно создать файл специального формата.

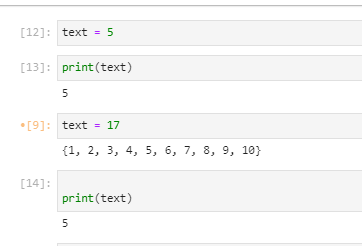


Вставлять сюда участки кода и отлаживать именно их.

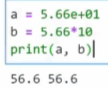


Но важно заметить что все объявленные переменные связаны и будут переписывать в случаи чего.

Но можно пропускать любые такие короткие участки кода, поэтому можно и не перезаписывать.



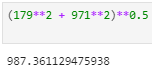
Напоминалка по работе е-шки



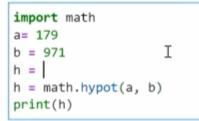
Да… чтобы сравнить дробные числа предлагают сравнивать их разность, чтобы она была меньше чем число 1е-7.



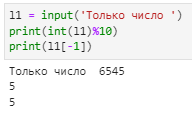
Тут a сравнивают с 0.6. Иначе если просто сравнить их через == то будет False

Пример как можно вычислить гипотенузу по 2-м сторонам. В конце число возводится в квадратный корень.

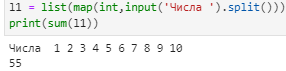
Или можно воспользоваться для этого специальной функцией.



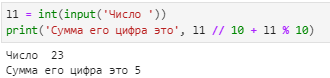




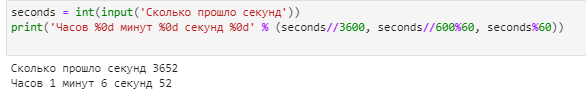


Ой не так понял задачу. Ща.

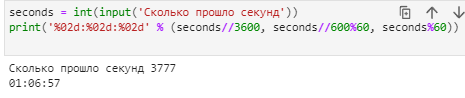
Требовалось это.





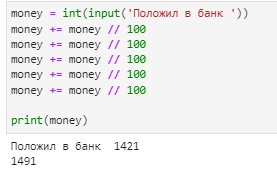
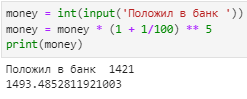


Лучше даже так:





Вот метод лучше моего, да и мой даже не верно считает кажется.



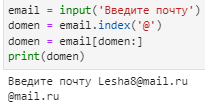
Это пропустил. Но метод **bool()** преобразует переменную в **True** или **False**

Еще можно с помощью **спец. символы** помещать одни и те же кавычки 

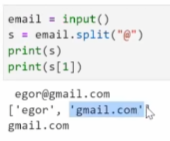


**Startswith()** это метод строки проверяет начинается ли строка с выбранной строки.

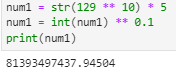




Еще можно воспользоваться методом **split()**

****

****

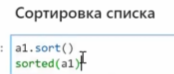
****

**Еще важная информация по копированию списков. Нельзя делать так:**

****

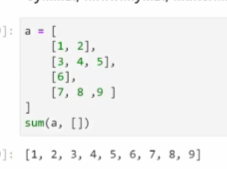
Потому что тогда b просто будет ссылаться на ту же область памяти, что и a. И когда будем изменять b изменится и a. Чтобы нормально копировать списки для этого собственно и существует метод **copy()**

**B = a.copy()**

****

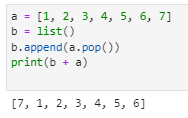
**Разница между функциями.** Метод меняет список. А функция просто возвращает измененный вариант.

Как можно список со списками рассортировать в один список.

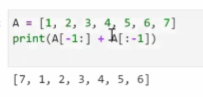




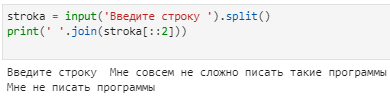
Не самый лучший вариант, но придумал первый что придумал.



Вот как надо было, но у меня что-то не вышло

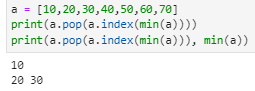




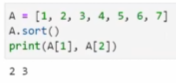




Я написал странный код.



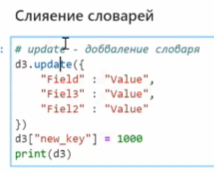
Есть еще такой способ.



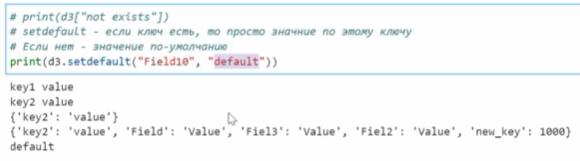
Но он из-за сортировки выполняться будет дольше.

У словаря еще есть метод, который возвращает список из кортежей пар, ключ – значение.

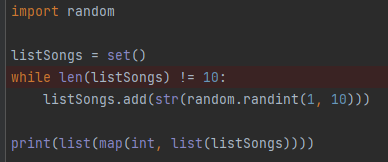




И еще. Если обратится к словарю по ключу, которого нет, то будет ошибка. Чтобы ее избежать существует метод **setdefault()**

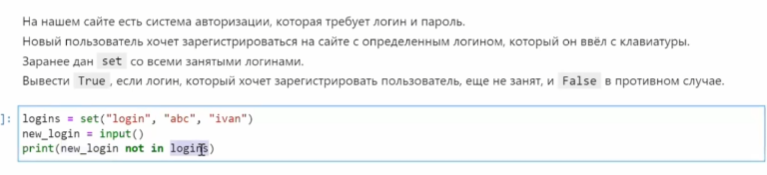
****

**Нашел решение проблемы со множествами и автоматической сортировкой. Сортировка не работает если тип данных строчный.**

****

****

**А так… множества быстро работают, так что их выгодно использовать примерно для таких задач:**

****

**Конец вебинара…**