## **Лекция 1. Продолжение знакомства с jupyter notebook**

## На этой лекции мы узнаем:

a) Узнать, как работать с jupyter notebook эффективней

b) Научиться красиво и доступно оформлять jupyter notebook

c) Посмотреть на функционал модуля Random

d) Более детально изучить словари, функции в Python

e) Разобраться с генераторами

f) Познакомиться с list, set, dict comprehensions

## **Термины лекции**

Markdown - облегчённый язык разметки, созданный с целью обозначения форматирования в простом тексте, с максимальным сохранением его читаемости человеком, и пригодный для машинного преобразования в языки для продвинутых публикаций.

Тэг HTML - элемент языка разметки. Текст, содержащийся между начальным и конечным тегом, отображается и размещается в соответствии со свойствами, указанными в начальном теге.

Словарь - неупорядоченная структура данных, которая позволяет хранить пары «ключ — значение».

Вложенный словарь – это словарь, содержащий другие словари.

Аргумент функции — значение, которое передается в функцию при её вызове.

Позиционный аргумент - это аргумент, передаваемый в функцию в определенной последовательности (на определенных позициях), без указания их имен.

Именованный аргумент - это аргумент, передаваемые в функцию при помощи имени.

Генератор - это объект, который сразу при создании не вычисляет значения всех своих элементов.

## **Краткий план лекции**

1. Jupyter notebook
   1. Сочетания клавиш в режиме редактирования
   2. Сочетания клавиш в режиме ввода команд
   3. Markdown
2. Словари
3. Модель Random
4. Функции
   1. Аргументы по позиции и по имени
   2. Аргументы по умолчанию
   3. Аргумент \*args
   4. Аргумент \*kwargs
   5. Аннотирование типов
5. Генератор
6. comprehensions
   1. list comprehensions
   2. использование условий
   3. set и dict comprehensions

## **Введение**

Всем привет!

Меня зовут Юля Пономарева, я инженер машинного обучения, работаю чаще с задачами в области компьютерного зрения - это всё, что связано с распознаванием визуальных образов на картинках или видео потоках. Получается, что я дарую зрение машинам.

Но еще очень часто приходят задачи из области аналитики, где есть большой массив данных, который нужно проанализировать, построить графики, которые наглядно показывают проблему и предложить решение.

Из наиболее запомнившихся проектов:

1. Прогнозирование выручки в новой торговой точке по местоположению
2. Прогнозирование стоимости недвижимости по характеристикам квартиры

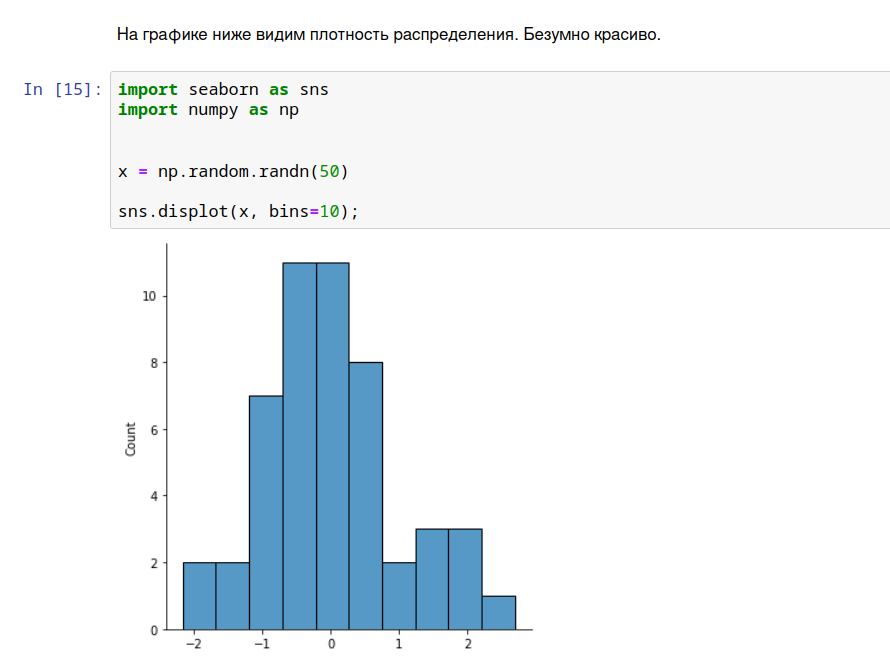
А также рассказываю про машинное обучение и инструменты для аналитиков на [youtube](https://www.youtube.com/c/machinelearrrning) канале.

## **Продолжение знакомства с jupyter notebook**

Сегодня будем дальше погружаться в jupyter notebook, узнаем, как эффективно работать с этим инструментом, рассмотрим более детальную работу со структурой данных словарь, обсудим аргументы в функциях и познакомимся с генераторами на Python.

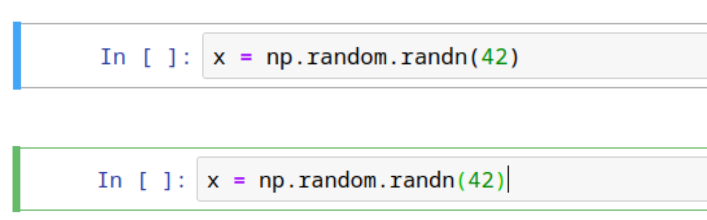
### **Jupyter notebook**

И начнем с работы в интерактивной среде разработки. В наших ноутбуках есть два вида ячеек. Ниже график показывает два типа ячеек.



К примеру, для проведения анализа набора данных по успешности рекламной кампании стоит пользоваться jupyter notebook, ведь мы можем писать код, а можем писать текст, что очень удобно для записей интерпретаций графиков - визуализировали данные и сразу же их проинтерпретировали в текстовой ячейке ниже. У нас есть возможность вести разработку и параллельно комментировать наши действия.

Также в ноутбуках есть два режима: режим ввода команд и режим редактирования. Обратите внимание, чем они отличаются. Режим ввода команд имеет выделенную синюю область, а режим редактирования имеет зеленую обводку.



В режиме ввода команд мы можем перемещаться по ячейкам, удалять их, добавлять новые, останавливать их работу. А во втором режиме можем редактировать содержимое ячейки.

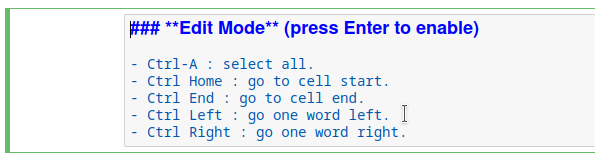
#### **Сочетания клавиш в режиме редактирования**

А теперь пройдемся по горячим сочетаниям клавиш.

Горячие сочетание клавиш (шорткат [англ. shortcut]) — это нажатие [кнопки](https://ru.wikipedia.org/wiki/%D0%9A%D0%BD%D0%BE%D0%BF%D0%BA%D0%B0_(%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0))/[клавиши](https://ru.wikipedia.org/wiki/%D0%9A%D0%BB%D0%B0%D0%B2%D0%B8%D1%88%D0%B0) (или сочетания клавиш) на клавиатуре, которому назначено некое действие — команды, исполняемые данной системой.

Примером шортката является знакомое и любимое Ctrl + C для копирования текста и Ctrl + V для вставки скопированного.

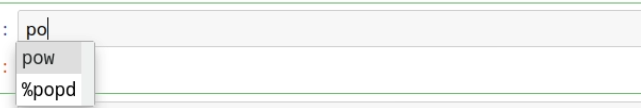
И начнем свое обсуждение с режима ввода редактирования (edit mode), чтобы в нем оказаться нужно нажать на Enter, тогда вы сможете перемещаться в рамках одной ячейки и писать код или текст.



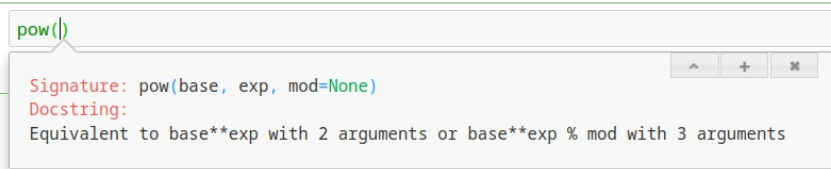
Из стандартных команд по работе с текстом - это

1. Ctrl+A - выделить всё
2. Ctrl+Home - подняться на начало ячейки
3. Ctrl+End - опуститься на конец ячейки
4. Ctrl+Left - перемещаться на одно слово влево
5. Ctrl+Right - перемещаться на одно слово вправо

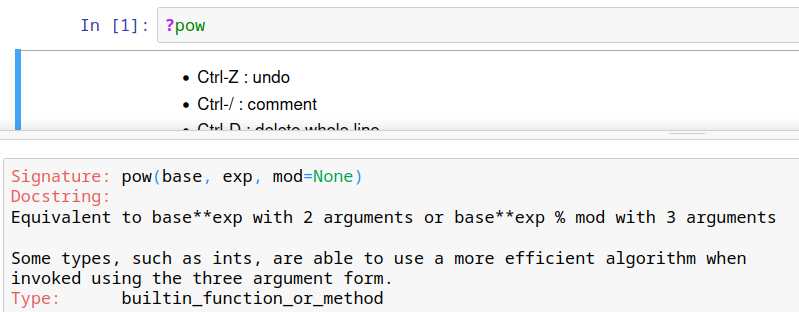
Если вы пишите код и хотите сэкономить свое время, то можете пользоваться автодополнением, нажатием на Tab, тогда у вас высветятся подсказки.



А если вдруг забыли, в каком порядке идут аргументы в функции, то для того, чтобы вспомнить можно нажать на Shift+Tab и выскакивает небольшая документация, посвященная этой функции.



Или же подобную документацию можно получить, если перед функцией написать знак вопроса (?):



И еще несколько приятных сочетаний клавиш:

1. Ctrl+Z - откатить только что сделанное действие
2. Ctrl+/ - закомментировать или раскомментировать
3. Ctrl+D - удалить целую строку

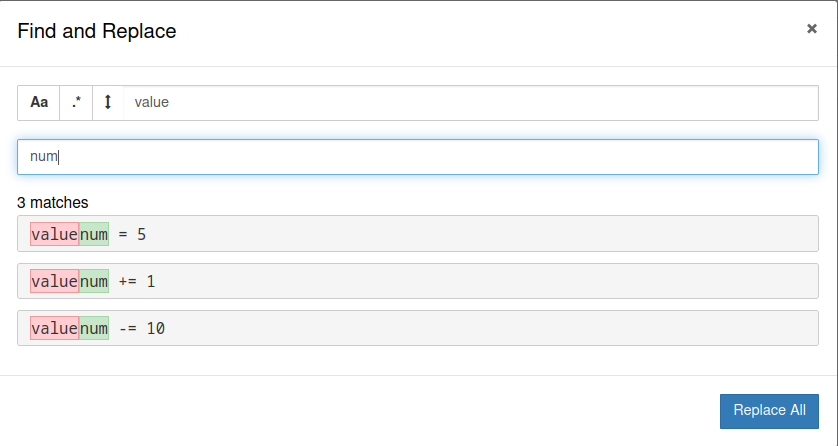
#### Сочетания клавиш в режиме ввода команд

С режимом редактирования разобрались, а теперь движемся к режиму команд (command mode), чтобы в нём очутиться нужно нажать на Escape, тогда вы сможете перемещаться по ячейкам вверх и вниз с помощью клавиш “стрелка вверх” и “стрелка вниз”.

1. А - создать новую ячейку сверху
2. B - создать ячейка снизу текущей
3. X - вырезать ячейку
4. C - копировать ячейку
5. V - вставить ячейку

Из интересного еще есть клавиша F, которая находит нужный набор символов и меняет на другое.

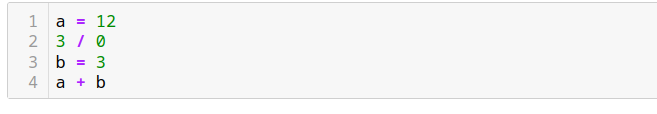
Для этого во-первых, нужно находится в выбранной ячейке, во-вторых нужно находиться в режиме ввода команд, чтобы была синяя обводка у рамки, ну а в-третьих, нужно нажать на F.



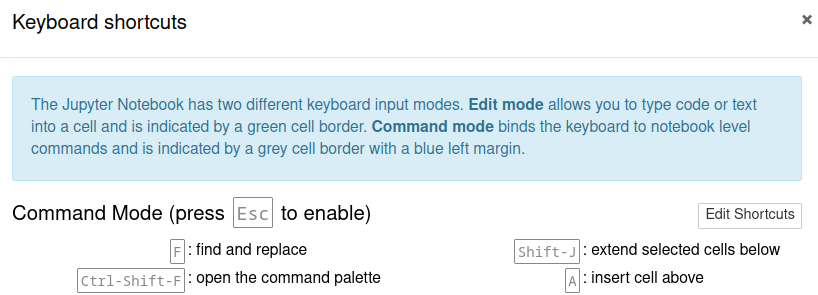
Движемся дальше и обсудим, как поменять тип ячейки:

1. если хочется из ячейки сделать код, то можно нажать на Y
2. а если хочется наоборот поменять на текст, то нажимайте на M.

Если вы где-то в коде поймали ошибку, но вам лень считать, в какой строке это произошло, то можно нажать на L и тогда юпитер покажет нумерацию строк. Если вам надоест смотреть на эти цифры можно нажать снова на L.

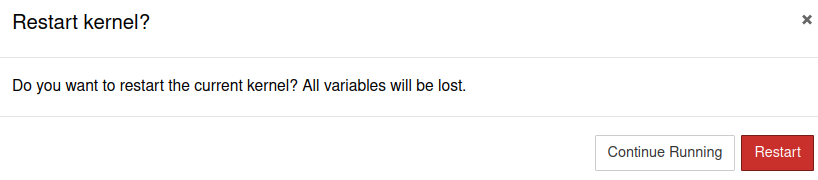


На H можно посмотреть на сочетания клавиш, некоторые из которых мы с вами уже посмотрели.



Если у вас ячейка долго выполняется, а у вас нет столько времени, то можно прекратить её выполнение, нажав на I, I.

И чтобы перезапустить весь ноутбук можно нажать на 0,0, но будьте аккуратней, ведь так все ваши переменные сотрутся и всё начнется с чистого листа. Если к этому готовы, то нажимайте на рестарт.



#### **Markdown**

Markdown - облегчённый язык разметки, созданный с целью обозначения форматирования в простом тексте, с максимальным сохранением его читаемости человеком, и пригодный для машинного преобразования в языки для продвинутых публикаций.

Этот язык разметки нужен для того, чтобы создавать красиво оформленные тексты. Вам не нужно использовать Microsoft Word или подобные программы, чтобы создавать документы с жирным или курсивным начертанием, цитатами, ссылками и таблицами.

Плавно переходим к ячейке с текстом и обсудим, как можно её красиво оформить. Можно из ячейки с текстом делать заголовок, путем нажатия на цифры от 1 до 6. 1 - это самый большой заголовок, а 6 - это самый маленький заголовок.



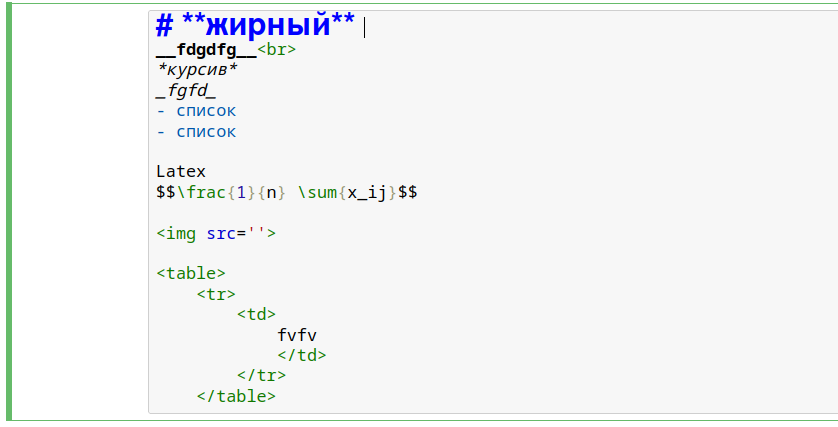


При этом можно менять заголовок через решетку (#), принцип здесь такой же, 1 решетка - самый большой заголовок, а 6 решеток - самый маленький.





Что в самой ячейке с текстом происходит? А там как ни странно можем писать текст, при этом его по-разному оформлять:



Для оформления и выделения мы используем символы, которые должны располагаться перед и после фразы или слова, которое нужно оформить.

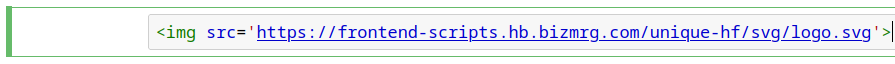
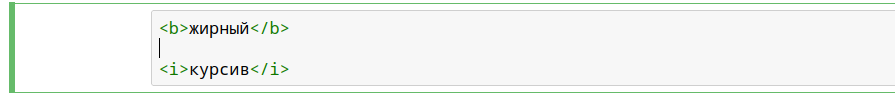
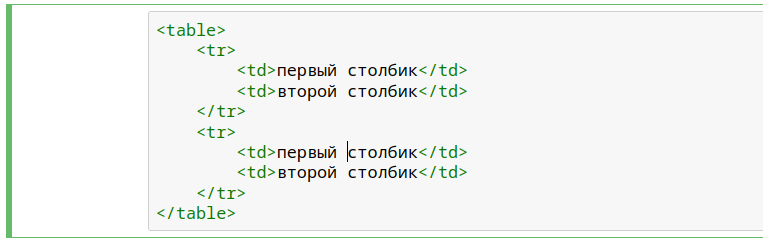
* \*\* или \_\_ - выделение текста жирным
* \* или \_ - выделение текста курсивом
* - - добавление списка в текст
* можно писать формулы на языке LaTex
* плюс можно пользоваться тегами из языка разметки html, чтобы добавлять изображения, ссылки, таблицы и многое другое

##### Теги HTML

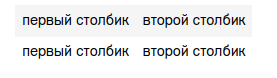
Тэг HTML - элемент языка разметки. Текст, содержащийся между начальным и конечным тегом, отображается и размещается в соответствии со свойствами, указанными в начальном теге.

Теги нужны для того, чтобы при отображении язык разметки понимал, что имеет дело не с простым текстом, а с элементом форматирования. И нужно применить какое-то действие. К примеру, любой сайт, на который вы можете перейти, содержит огромное количество тегов. Без них не было бы такого красивого форматирования страниц с текстами разных начертаний, шрифтов, с таблицами, картинками и гиперссылками.

Рассмотрим пару самых полезных тегов.

1. img, сюда в src можно передать адрес любой картинки с интернета
2. <b> оформляем наш текст в этот тег, чтобы он стал жирным
3. <i> окружаем текст этим тегом, чтобы он был написан курсивом 
4. <br> чтобы текст писался с новой строки
5. и давайте обсудим <table>, с помощью него можно создавать таблицы

Сначала пишем тег <table> и </table>, чтобы указать границы таблицы, затем, чтобы создать строку пишем <tr>, а чтобы написать значение в столбце, пишем тег <td>.



Для более комфортной и эффективной работы в интерактивной среде разработки jupyter notebook есть горячие сочетания клавиш, которые мы рассмотрели. А чтобы в ноутбуке было все читаемо и доступно мы познакомились с markdown, который позволяет писать форматируемый текст для комментирования кода и фиксирования своих мыслей, выводов и идей.

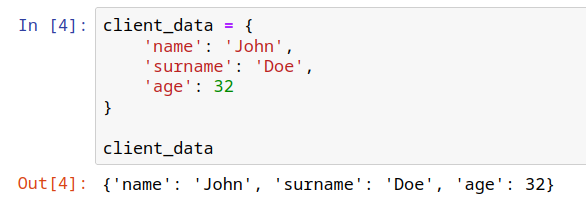
### Словари

Отлично, с ноутбуком и markdown’ом разобрались, дальше движемся к расширению наших знаний по структуре данных *dict* и будем с ним работать в юпитер ноутбуке.

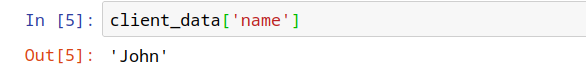
Словарь - неупорядоченная структура данных, которая позволяет хранить пары «ключ — значение».

Данный словарь использует строки в качестве ключей, однако ключом может являться в принципе любой неизменяемый тип данных. Значением же конкретного ключа может быть что угодно.

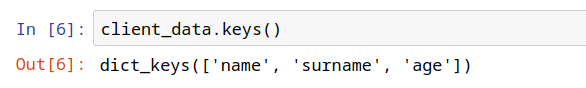
Чтобы вспомнить про эту структуру данных и узнать новые подходы для работы с ней, возьмем данные про John Doe, которые лежат в словаре.



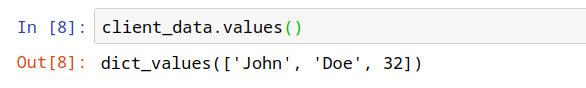
Чтобы достать нужный объект по ключу, к примеру ‘name’, нужно после имени словаря указать квадратные скобки и в них передать имя желаемого ключа.



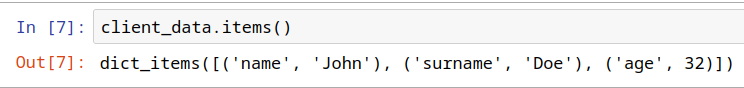
Или можем получить все ключи, для этого вызываем метод keys(), возвращается список из ключей.



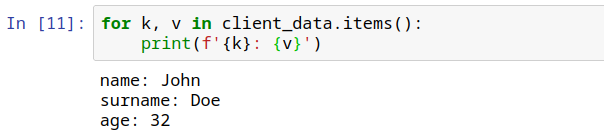
По аналогии со ключами словаря, для получения всех значений можем вызвать метод values().



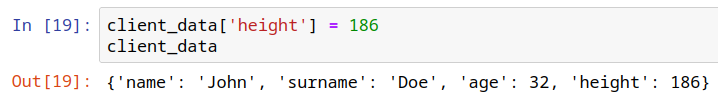
А если хочется получить и ключи, и значения сразу, то тут используем items(), возвращается кортеж, где в первом элементе будет стоять ключ, а во втором элементе - значение.



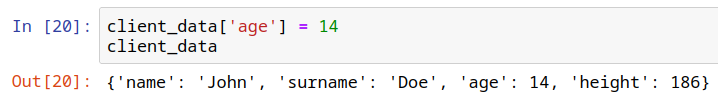
Вот по этой структуре данных можно итерироваться, получая сразу ключи и значения за одну итерацию.



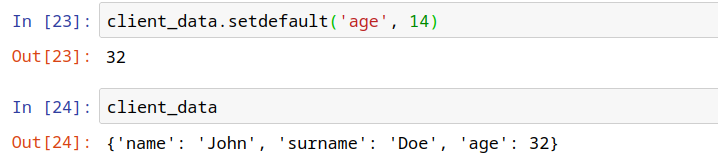
Так же можем добавить новые сведения про John Doe в наши данные. Хотим добавить информацию о росте, для этого пишем новый ключ ‘height’ и через знак равно значение роста.



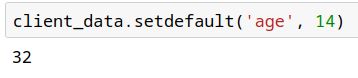
При этом если такой ключ уже был, то мы его перезапишем на новое значение, даже если мы этого не хотели.



К примеру, у нас уже записан возраст John’а, он у нас 32-летний, а мы вдруг ошибочно хотим снова записать ключ ‘age’. И всё перезаписывается. Это поведение кода для нас может быть неприемлемым.

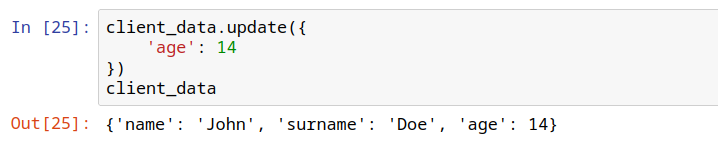


Если боитесь перезаписать существующие ключи, то можете пользоваться методом setdefault(), тогда Python сначала посмотрит, а есть ли такой ключ в словаре и если он есть, то вернет его значение, а если такого ключа нет, то запишется на то, которое указано в методе. Видим, что у нас ничего не поменялось в ключе возраста, как там был возраст 32, так он и остался:

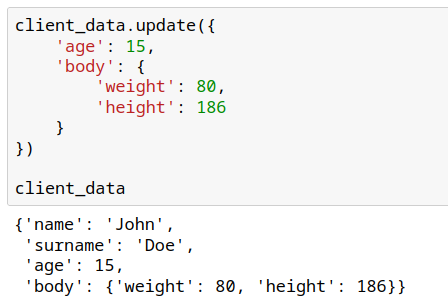


Есть еще один интересный способ, как можно менять значения в словаре, через метод update(). Метод update() обновляет словарь элементами из другого объекта словаря или из итерируемых пар ключ-значение. Функция добавляет элемент(ы) в словарь, если ключ отсутствует в словаре. Если ключ находится в словаре, он обновляет ключ новым значением.

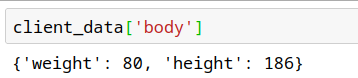
Попытаемся поменять возраст John на 14 через метод update() и видим, что здесь нет проверки: а есть ли уже такой ключ, как было в setdefault(), здесь просто происходит перезапись:



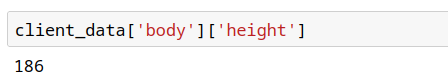
И перезаписывает он не только существующие ключи, но и записывает новые, при этом здесь записывается вложенный словарь в ключ ‘body’. Вложенный словарь – это словарь, содержащий другие словари.



Получить элемент из вложенного словаря можно через обращение к первому ключу ‘body’, где получим словарь:



А затем обращаемся к ключу ‘height’, чтобы получить рост из словаря:

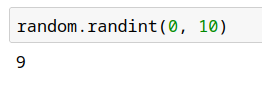


### Модуль Random

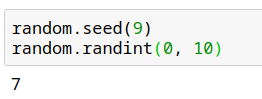
Модуль random предоставляет функции для генерации случайных чисел или к примеру, для случайного выбора элементов последовательности. Посмотрим на самые полезные функции.

При разработке программ довольно часто возникает необходимость получить некоторое случайно выбранное из множества значение, к примеру, чтобы только на этом объекте показать проведенную работу аналитика, а не чтобы выводить весь массив данных.

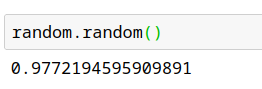
random.randint(A, B) возвращает случайное целое число из диапазона A до B.



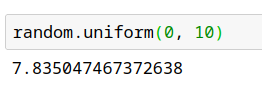
При этом значение на выходе всегда будет разное, если будем перезапускать данную ячейку, а чтобы значение случайности всегда было одинаковое можем зафиксировать random seed. И теперь сколько бы мы раз не перезапускали эту ячейку, генерация всегда будет одинаковая:



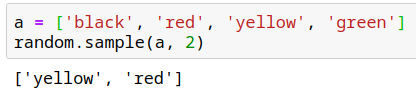
random.random() - возвращает случайное число от 0 до 1:



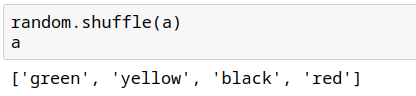
random.uniform(A, B) - возвращает случайное число с плавающей точкой в диапазоне от A до B.



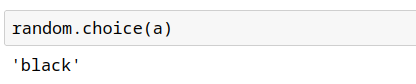
random.sample(data, k) - возвращает список длиной k из последовательности data.



random.shuffle(data) - перемешивает последовательность, при этом изменяется сама последовательность, ничего не возвращается.



random.choice(data) - возвращает случайный элемент из последовательности.



### Функции

Функция в python - объект, который принимает аргументы, производит с ними желаемые действия и возвращает значение. Также функция — это фрагмент программного кода, который решает какую-либо задачу. Его можно вызывать в любом месте основной программы. Функции помогают избегать дублирования кода при многократном его использовании.

Аргумент функции — значение, которое передается в функцию при её вызове.

#### Аргументы по позиции и по имени

Аргументы бывают:

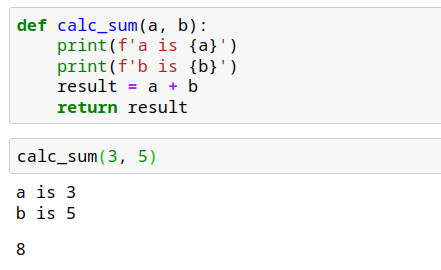
1. Позиционными

Позиционный аргумент - это аргумент, передаваемый в функцию в определенной последовательности (на определенных позициях), без указания их имен

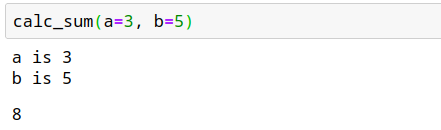
1. Именованными

Именованный аргумент - это аргумент, передаваемые в функцию при помощи имени.

В функцию подсчета суммы передаем два аргумента по позиции:

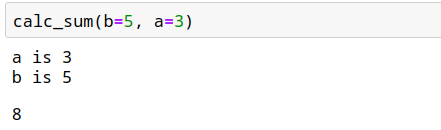


Можно передать их по имени, первый аргумент называется a, второй называется b:

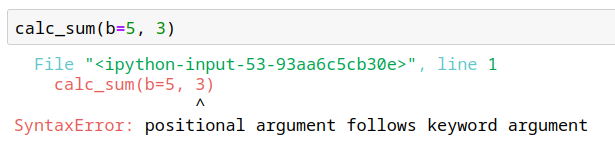


Если мы сначала попытаемся передать аргумент b, а затем аргумент a по порядку, то ничего не получится, ведь по порядку сначала надо принять a, а только потом b.

Но передать их по именам можно:



А вот если сначала указать b по имени, а аргумент a по порядку, то будет ошибка, что позиционный аргумент идет после аргумента по имени, так делать в Python нельзя:



Зато можно сначала указать по порядку аргументы, а затем по имени и никаких ошибок:

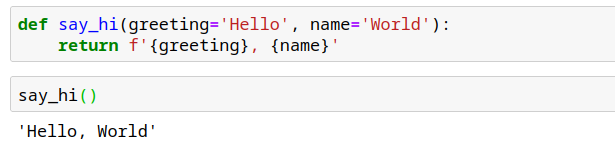


Так что стратегия такая: сначала все порядковые аргументы, а затем все именованные.

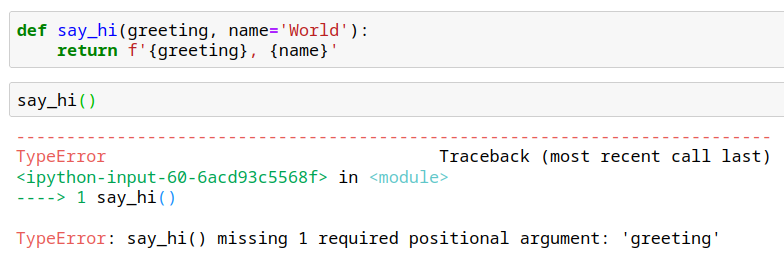
#### Аргументы по умолчанию

В функцию можно передавать аргументы по умолчанию, это те аргументы, значения которых будут использованы, если не передали им явное значение при вызове.

Такую функцию можно вызвать без аргументов и всё отработает без ошибок, ведь два аргумента имеют значения, хотя мы их не указывали при вызове:



А вот если не указать обычный рядовой аргумент, то будет ошибка, что функция ожидает обязательный позиционный аргумент, но он не пришел на вход:



Менять аргументы по умолчанию, конечно же, можно:

#### 

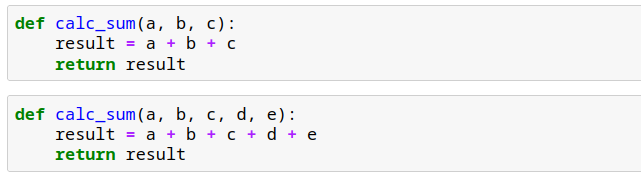
#### Аргумент \*args

\*args — это сокращение от «arguments» (аргументы).

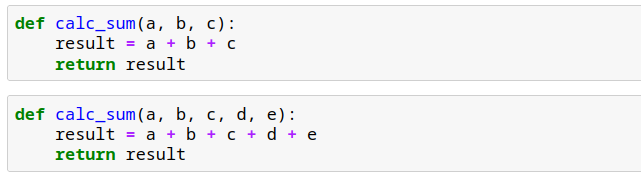
Оператор \* чаще всего ассоциируется у людей с операцией умножения, но в Python он имеет и другой смысл. Этот оператор позволяет «распаковывать» объекты, внутри которых хранятся некие элементы.

Дело в том, что «args» — это всего лишь набор символов, которым принято обозначать аргументы. Самое главное тут — это оператор \*. А то, что именно идёт после него, особой роли не играет. Благодаря использованию \* мы создали список позиционных аргументов на основе того, что было передано функции при вызове.

Давайте вернемся к функции суммирования двух элементов, а что если элементов будет не два, а три? Нужно будет переписывать функцию.



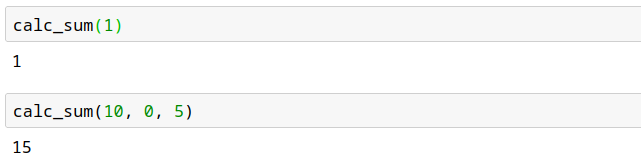
А если аргументов будет не три, а пять, снова изменять функцию.



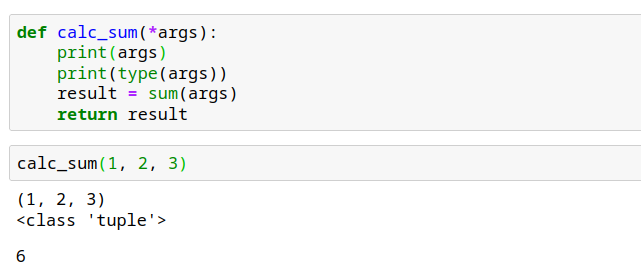
Чтобы не менять постоянно реализацию функции можно воспользоваться аргументом \*args, он помогает, когда в вашу функцию может поступать разное количество входных значений.



И такая реализация будет работать с разным количеством параметров, хоть с одним, хоть с тремя и так далее:



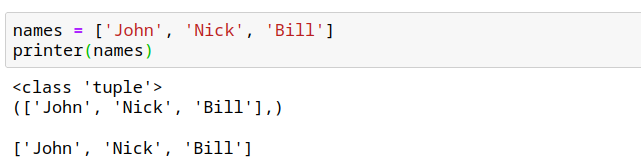
Аргумент \*args из себя представляет кортеж, который может быть любой длины, по которому можно итерироваться при желание, или же можно доставать объекты, или же можно просуммировать:



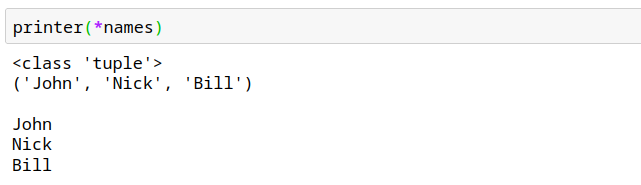
Так же в args можно передавать элементы разных типов данных:



Но аккуратней с списками, потому что список - это один элемент:



Чтобы каждый элемент списка, считался самостоятельным аргументом нужно его распаковать, используя символ звездочки:

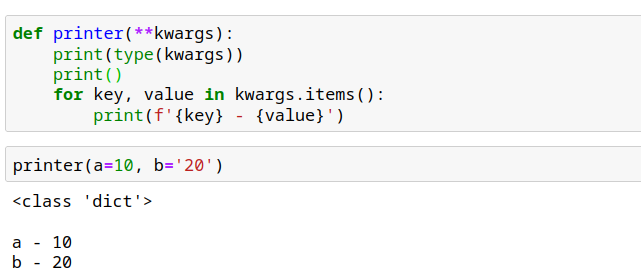


#### Аргумент \*\*kwargs

\*\*kwargs — сокращение от «keyword arguments» (именованные аргументы).

Главное — это два символа \*\*. Благодаря им создается словарь, в котором содержатся именованные аргументы, переданные функции при ее вызове.

Еще один интересный атрибут, у которого похожая логика на args, только здесь атрибуты будут именованными. Аргумент \*kwargs из себя представляет словарь, который может содержать любое количество элементов:



К этим ключам при необходимости можно обращаться, а какие-то ключи могут быть проигнорированы:

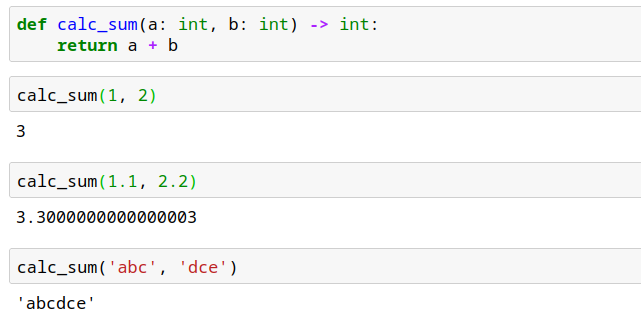


Так же в функцию с атрибутом \*\*kwargs можно передавать словарь, но его тоже нужно распаковать, используя две звездочки:

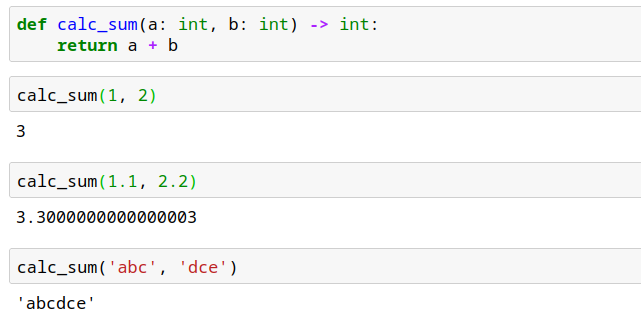


#### Аннотирование типов

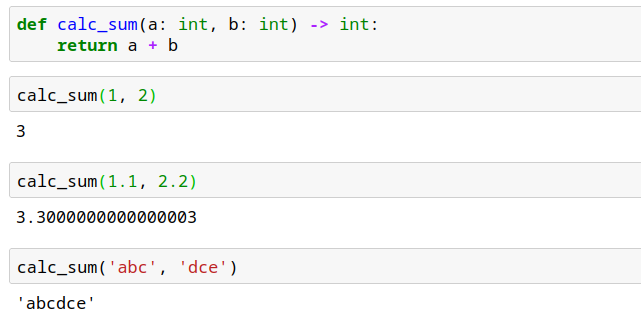
Когда работаете с Python функциями, то возникает неочевидная проблема. Порой забываешь, а какого типа должны быть входные параметры и чего вообще ждать на выходе. Здесь может помочь аннотирование типов. Можно через двоеточие указывать, какой тип ждет функция для каждого аргумента. А для вывода через -> можно указать ожидаемый тип на выходе функции:



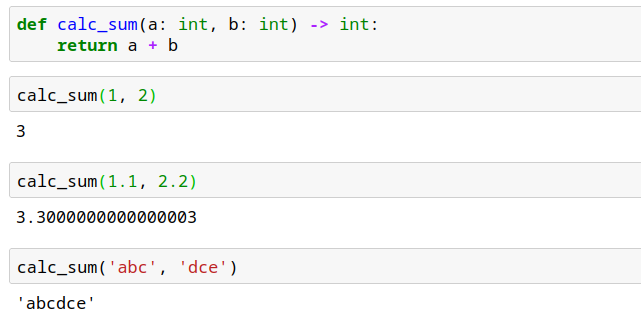
Функция ждет int - целые числа, передаем их в функцию, всё работает и на выходе получаем int:



Но если передадим в эту функцию float значения, то снова всё работает, а на выходе тоже float:



Еще забавней, когда строки можно передать в функцию для подсчета суммы и всё равно не поймаем ошибку:



А всё это связано с тем, что аннотирование типов нужно только для разработчика, чтобы он сам не запутался в своем коде, Python никак не согласуется с этими аннотациями. Конечно IDE может подсвечивать данный кусок кода и мягко ругаться, но не более, ошибок здесь не будет.

### Генератор

Допустим, у вас есть файл, который весит десяток гигабайт. Из него нужно выбрать и обработать строки, подходящие под какое-то условие. Что в такой ситуации делать? А можно обрабатывать такие объемы данных небольшими порциями, чтобы не вызывать переполнения памяти. В Python на этот случай есть специальный инструмент — генераторы.

Генератор - это объект, который сразу при создании не вычисляет значения всех своих элементов.

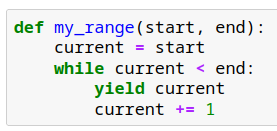
Генератор хранит в памяти:

1. последний вычисленный элемент
2. правило перехода к следующему
3. условие, при котором выполнение прерывается

Вычисление следующего значения происходит лишь при выполнении метода next(). Предыдущее значение при этом теряется.

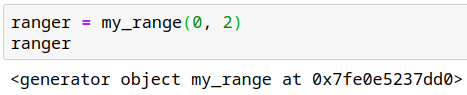
Этим генераторы отличаются от списков, ведь списки хранят в памяти все свои элементы, и удалить их можно только программно. Вычисления с помощью генераторов экономят память.

Чтобы создать генератор, необходимо определить функцию, как обычно, но использовать yield вместо return:



Оператор yield приостанавливает функцию и сохраняет локальное состояние, чтобы его можно было возобновить с того места, где оно было остановлено.

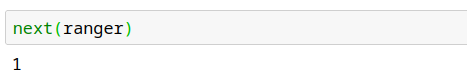
Вызов функции не выполняет ее, а функция только возвращает объект-генератор:



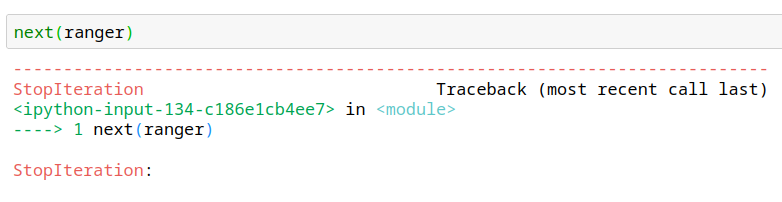
При первом вызове next() выполнение начинается с начала функции и продолжается до оператора yield, где возвращается значение справа от оператора:



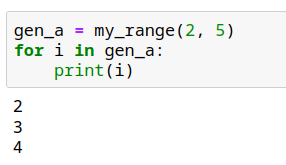
Последующие вызовы next() продолжаются с оператора yield до конца функции, затем новый обход, который идет в начало функции и выполняется снова до оператора yeild:



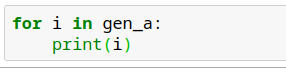
Потом еще раз тоже самое, но если yield больше не вызывается (в нашем случае означает, что условие while не выполняется, потому что current >= 0), возникает исключение StopIteration:



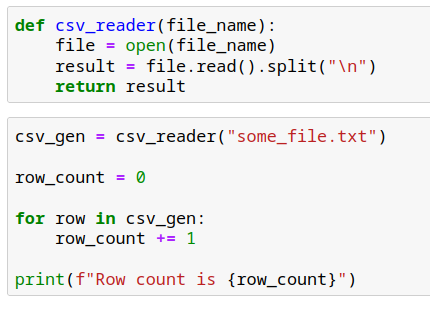
По генератору можем итерироваться через цикл for. К примеру, создадим объект генератора, который реализовали выше и запишем в переменную gen\_a. Теперь по нему можем пройтись в цикле, используя for, печатая каждый элемент на экран:



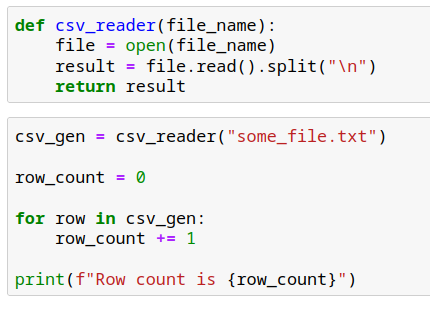
А если захотим еще раз пройтись по этой же переменной, то ничего не вернется, это снова подтверждение тому, что генератор хранит только один элемент за раз, а не весь массив данных, как это делает список:



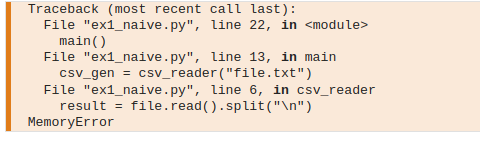
Работа с потоками данных и большими файлами, такими например как [CSV](https://realpython.com/courses/reading-and-writing-csv-files/), являются наиболее распространенными вариантами использования генераторов. Предположим, что хотите посчитать количество имеющихся в нем рядов:



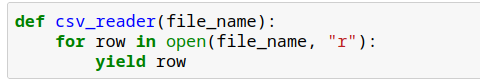
Можно предположить что csv\_gen - список. А чтобы его заполнить, csv\_reader() открывает файл и загружает его содержимое в csv\_gen. Затем итерируемся по списку, увеличивая значение row\_count для каждого нового ряда.



Всё кажется довольно логичным, но что, если файл будет слишком большим и не будет помещаться в память? В этом случае при выполнении кода выпадет ошибка по памяти:



Можем переписать функцию csv\_reader через генератор. В этой версии открываем файл, проходим его по строкам и извлекает для чтения только отдельный ряд, вместо того, чтобы возвращать весь файл целиком:



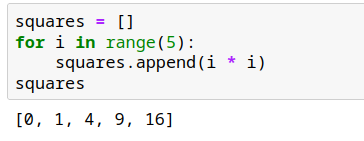
Генераторы позволяют запрашивать значения по мере необходимости, делая наш код более эффективным в использовании памяти.

### comprehensions

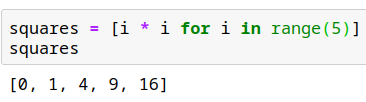
#### list comprehensions

Давайте вспомним про цикл for, если мы пользуемся им, чтобы из одного списка сделать новый список, то следуем этим пунктам:

1. Создаем пустой список
2. Итерируемся по объекту (или range)
3. Добавляем каждый элемент в список



Или же можем пользоваться List comprehensions - это чудесный способ составления списков. Можем переписать цикл for всего в одну строку кода:



Вместо того, чтобы создавать пустой список и добавлять каждый элемент в конец, мы просто определяем список и его содержимое одновременно, следуя этому формату:

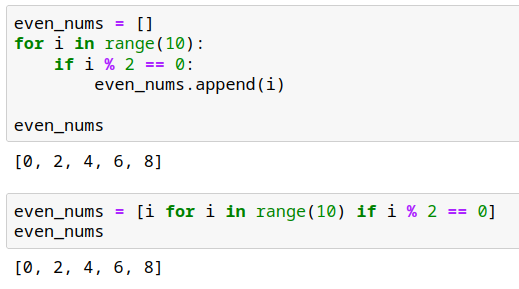


где

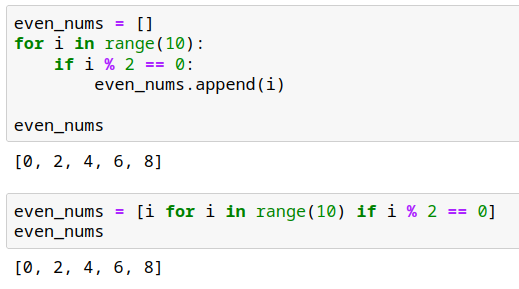
* expression - какое-то вычисление, вызов функции или любое другое допустимое выражение, которое возвращает значение.
* member - объект или значение в списке или итерируемым объекте.
* iterable - объект, по которому можно итерироваться (список, множество, последовательность, генератор)

#### Использование условий

Условные выражения позволяют отфильтровывать нежелательные значения. Снова сначала рассмотрим реализацию через цикл:

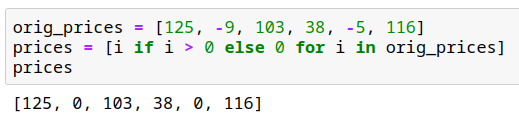


А теперь с помощью list comprehension:

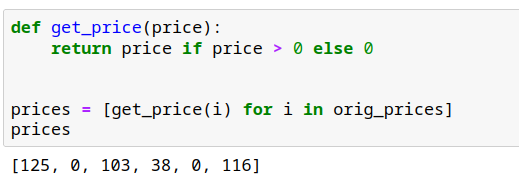


Гораздо более наглядно.

При этом условия могут быть сложнее, можем возвращать объект из итерируемого объекта или же заменять его на что-то, используя конструкцию if else:



Если условие хочется вынести отдельно, то можно реализовать дополнительную функцию для этого:



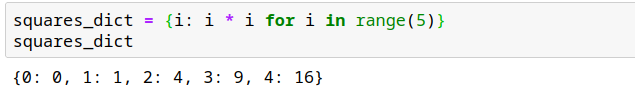
#### set и dict comprehensions

Помимо list comprehension в Python еще возможно создавать множественные и словарные представления (set comprehensions и dictionary comprehensions).

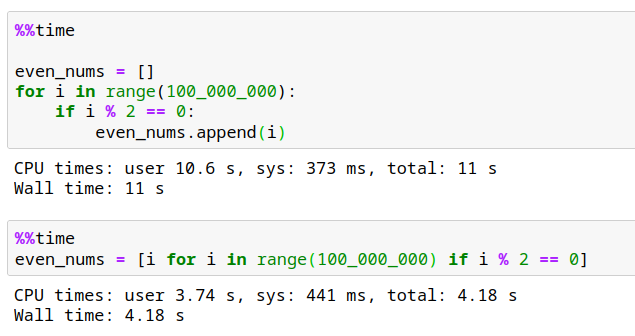
set comprehension почти точно такое же, как и list comprehensions. Разница лишь в том, что заданные значения обеспечивают, чтобы выходные данные не содержали дубликатов. Создать set comprehension можно, используя фигурные скобки вместо квадратных:



Dict comprehensions создаются аналогично, только еще добавляется определение ключа:



И напоследок, можем замерить производительность подходов, list comprehensions побеждают обычные циклы:



На следующей лекции будем знакомиться с анализом данных с помощью библиотеки Pandas. Она позволяет быстро считать статистики, фильтроваться по данным и сортировать их.

## Рекомендуемая дополнительная литература или материалы

1. Модуль Random <https://pythonworld.ru/moduli/modul-random.html>
2. Jupyter Notebook для начинающих <https://webdevblog.ru/jupyter-notebook-dlya-nachinajushhih-uchebnik/>
3. Добавляем в Jupyter Notebook красоту и интерактивность <https://habr.com/ru/post/485318/>
4. Введение в генераторы Python <https://webdevblog.ru/vvedenie-v-generatory-python/>
5. Что такое \*args и \*\*kwargs в Python? <https://habr.com/ru/company/ruvds/blog/482464/>
6. Что такое list comprehension? Зачем оно? Какие ещё бывают? <https://dvmn.org/encyclopedia/qna/5/chto-takoe-list-comprehension-zachem-ono-kakie-esche-byvajut/>

## Используемая литература или материалы

1. Генераторы Python <https://pythonist.ru/generatory-python-ih-sozdanie-i-ispolzovanie/>
2. Когда использовать List Comprehension в Python <https://webdevblog.ru/kogda-ispolzovat-list-comprehension-v-python/>