**1 Основные типы данных. Неизменяемые типы**

К основным встроенным типам относятся: Логические переменные (**True, False**), Числа(**int, float, complex**), Списки(**list, tuple, range**), Строки(**str**), Множества(**set, frozen set**), Словари(**dict**).

В Python существуют изменяемые и неизменяемые типы.

К неизменяемым типам относятся: целые числа (*int*),  числа с плавающей точкой (*float*), комплексные числа (*complex*), логические переменные (*true, false*), кортежи (*tuple*), строки (*str*) и неизменяемые множества (*frozen set*).

К изменяемым типам относятся: списки (*list*), множества (*set*), словари (*dict*).

**2 Числовые типы в Python. Библиотека random**

**Числовой тип данных в Python** предназначен для хранения числовых значений. Это *неизменяемый тип данных*, что означает, что изменение значения числового типа данных приведет к созданию нового объекта в памяти (и удалению старого)

В **Python** есть четыре вида числового типа данных:

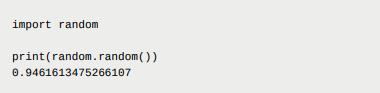
* *int* (целое число)
* *long* (длинное целое число [может быть представлено в восьмеричной или шестнадцатеричной системе исчисления])
* *float* (число с плавающей точкой: -0.2, 0.0, 3.14159265 и т.д.)
* *complex* (комплексное число)

Модуль **random** позволяет генерировать случайные числа. Прежде чем использовать модули, необходимо подключить его с помощью инструкции:

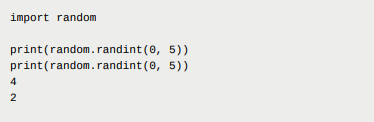


Список **методов модуля random** в Python:

random.random() — возвращает псевдослучайное число от 0.0 до 1.0



random.randint(a,b) — возвращает псевдослучайное целое число в диапазоне от a до b:



Это те что я использовал в ходе выполнения лабораторных работ, их там конечно гораздо больше, но для примера хватит и этих с другими можно ознакомится на сайте metanit.com и pythonworld.ru

**3 Библиотека math. Основные константы и операции**

Встроенный модуль **math** в Python предоставляет набор функций для выполнения математических, тригонометрических и логарифмических операций.

Некоторые из основных **операций** модуля:

* **pow(a, b):** возведение числа a в степень b
* **sqrt(a):** квадратный корень числа a
* **factorial(a):** факториал числа
* **degrees(rad):** перевод из радиан в градусы
* **radians(grad):** перевод из градусов в радианы
* **cos(rad):** косинус угла в радианах
* **sin(rad):** синус угла в радианах
* **tan(rad):** тангенс угла в радианах
* **acos(rad):** арккосинус угла в радианах
* **asin(rad):** арксинус угла в радианах
* **atan(rad):** арктангенс угла в радианах
* **log(a, base):** логарифм числа a по основанию base
* **log10(a):** десятичный логарифм числа a

Основных **константы** модуля:

* **math.pi** - число пи
* **math.e** - натуральный логарифм

**4 Строки в Python. Основные операции. Особенности**

Под **строками в Python** подразумевается набор символов между кавычками. В **Python** можно использовать пары одинарных либо двойных кавычек. Из строк можно взять подстроку используя оператор нарезки ( [ ] и [ : ] ) с индексами от нуля для первого символа строки и до последнего. Так же можно использовать обратную индексацию от -1 для последнего символа до начала.

Оператор плюс ( + ) для строк соединяет две строки в одну, звездочка ( \* ) оператор повторения

Встроенные **функции** строк:

* chr() - преобразует целое число в символ
* ord() - преобразует символ в целое число
* len() - возвращает длину строки
* str() - изменяет тип объекта на string

**5 Списки в Python. Основные операции. Вложение, виды вложений**

**Список** (list) представляет тип данных, который хранит набор или изменяемую последовательность элементов. Для создания списка в квадратных скобках ([]) через запятую перечисляются все его элементы. Например, определим список чисел: numbers = [1, 2, 3, 4, 5]. Список необязательно должен содержать только однотипные объекты. Мы можем поместить в один и тот же список одновременно строки, числа, объекты других типов данных: objects = [1, 2.6, "Hello", True]. Списки кроме стандартных данных типа строк, чисел, также могут содержать другие списки: users = [["Tom", 29], ["Alice", 33], ["Bob", 27]]

Встроенные **функции** списка:

* len(list) - возвращает длину списка
* sorted(list, [key]) - возвращает отсортированный список
* min(list) - возвращает наименьший элемент списка
* max(list) - возвращает наибольший элемент списка

Для управления элементами списки имеют целый ряд **методов**:

* append(item): добавляет элемент item в конец списка
* insert(index, item): добавляет элемент item в список по индексу index
* remove(item): удаляет элемент item. Удаляется только первое вхождение элемента. Если элемент не найден, генерирует исключение ValueError
* clear(): удаление всех элементов из списка
* index(item): возвращает индекс элемента item. Если элемент не найден, генерирует исключение ValueError
* pop([index]): удаляет и возвращает элемент по индексу index. Если индекс не передан, то просто удаляет последний элемент.
* count(item): возвращает количество вхождений элемента item в список
* sort([key]): сортирует элементы. По умолчанию сортирует по возрастанию. Но с помощью параметра key мы можем передать функцию сортировки.
* reverse(): расставляет все элементы в списке в обратном порядке

Вложенный список - это список списков. Вложенный список — это всегда один элемент вне зависимости от количества его элементов.

Виды вложений не нашел

**6 Словари в Python. Основные операции. Вложение, виды вложений**

**Словарь** — неупорядоченный набор пар ключ-значение. Они используются, когда нужно сопоставить каждому из ключей значение и иметь возможность быстро получать доступ к значению, зная ключ.

Определение словаря имеет следующий синтаксис:

* dictionary = { ключ1:значение1, ключ2:значение2, ....}

Определим пару словарей:

* users = {1: "Tom", 2: "Bob", 3: "Bill"}
* elements = {"Au": "Золото", "Fe": "Железо", "H": "Водород", "O": "Кислород"

Для управления элементами словари имеют целый ряд методов.

* dict.clear() - очищает словарь.
* dict.copy() - возвращает копию словаря.
* dict.fromkeys(seq[, value]) - создает словарь с ключами из seq и значением value (по умолчанию None).
* dict.get(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а возвращает default (по умолчанию None).
* dict.items() - возвращает пары (ключ, значение).
* dict.keys() - возвращает ключи в словаре.
* dict.pop(key[, default]) - удаляет ключ и возвращает значение. Если ключа нет, возвращает default (по умолчанию бросает исключение).
* dict.popitem() - удаляет и возвращает пару (ключ, значение). Если словарь пуст, бросает исключение KeyError. Помните, что словари неупорядочены.
* dict.setdefault(key[, default]) - возвращает значение ключа, но если его нет, не бросает исключение, а создает ключ со значением default (по умолчанию None).
* dict.update([other]) - обновляет словарь, добавляя пары (ключ, значение) из other. Существующие ключи перезаписываются. Возвращает None (не новый словарь!).
* dict.values() - возвращает значения в словаре.

**Вложенные словари** – это словари, содержащие другие словари. Мы можем создать вложенный словарь так же, как мы создаем обычный словарь, используя фигурные скобки.

Видов тоже нет, мне кажется он их по приколу добавил

**7 Кортежи в Python. Основные операции. Особенности**

**Кортеж** (tuple) представляет последовательность элементов, которая во многом похожа на список за тем исключением, что кортеж является неизменяемым типом. Поэтому мы не можем добавлять или удалять элементы в кортеже, изменять его.

Для создания кортежа используются круглые скобки, в которые помещаются его значения, разделенные запятыми. Если вдруг кортеж состоит из одного элемента, то после единственного элемента кортежа необходимо поставить запятую:

user = ("Tom", 23)

user = "Tom", 23

user = ("Tom",)

Обращение к элементам в кортеже происходит так же, как и в списке по индексу.

Преимущества **кортежей**:

* Защита от дурака. То есть кортеж защищен от изменений, как намеренных, так и случайных.
* Меньший размер

Основные операции **кортежей**:

* Операция присваивания кортежа. =
* Операция tuple() – создание кортежа из итерированного объекта
* Операция T[i]. Получить элемент по индексу
* Операция T[i][j]. Получить доступ к составному элементу в кортеже
* Операция T[i:j]. Взятие среза в кортеже
* Сложение. +
* Повторение. \*
* Операция in. Проверка вхождения элемента в кортеж

**Методы** работы с кортежами:

* index() - поиск позиции элемента в кортеже
* count() - количество вхождений элемента в кортеж

**8 Множества в Python. Основные операции. Особенности**

**Множества** — это неупорядоченный список, то есть элементы внутри него располагаются в случайном порядке. Как списки и словари, множество относится к изменяемым типам данных. Но при этом невозможен поиск по индексу, так как элементы в случайном порядке. Зато множества дают высокую скорость поиска по ним. Например, это удобно, когда порядок вам не важен, а нужно просто быстро искать нужные элементы.

Основные **операции**:

* Создание множеств
* Доступ к элементам множеств
* Добавление элементов во множество
* Удаление элементов из множеств
* Объединение множеств
* Пересечение множеств
* Разница множеств
* Сравнение множеств

**Методы:**

* copy() - этот метод возвращает копию множества
* isdisjoint() - этот метод проверяет, является ли множество пересечением или нет
* len() - этот метод возвращает длину множества

**9 Файлы в Python. Основные операции. Unicode и двоичные файлы**

**Файл** — это всего лишь набор данных, сохраненный в виде последовательности битов на компьютере. Информация хранится в куче данных (структура данных) и имеет название «имя файла» (filename).

В Python существует два типа файлов:

* Текстовые
* Бинарные

**Текстовые файлы**

Это файлы с человекочитаемым содержимым. В них хранятся последовательности символов, которые понимает человек. Блокнот и другие стандартные редакторы умеют читать и редактировать этот тип файлов.

Текст может храниться в двух форматах: ( .txt ) — простой текст и ( .rtf ) — «формат обогащенного текста»

**Бинарные файлы**

В бинарных файлах данные отображаются в закодированной форме (с использованием только нулей (0) и единиц (1) вместо простых символов). В большинстве случаев это просто последовательности битов.

Они хранятся в формате  .bin

Открытие файла производится с помощью **метода open()**

f = open(file\_name, access\_mode)

Где, file\_name  = имя открываемого файла, а access\_mode  = режим открытия файла. Он может быть: для чтения, записи и т. д. По умолчанию используется режим чтения ( r ). Например: (r-только для чтения, w-только для записи. Создаст новый файл, если не найдет с указанным именем, и так далее)

Закрытие файла производится с помощью **метода close()**

Основные **функции**:

* read() - функция чтения
* readline() - функция построчного чтения
* write() - функция записи
* rename() - функция переименовывания

Основные **методы**:

* file.write(str) - добавляет строку str в файл
* file.next() - возвращает следующую строку файла
* file.read(n) - чтение первых n символов файла

Остальное описано выше

10 Типизация в Python. Понятие динамической типизации

**Типизация** — это то, как язык распознаёт типы переменных. Типизация определяет, нужно ли вам писать тип, или язык «поймёт» его сам, и насколько свободно можно с типами работать

**Динами́ческая типиза́ция** — приём, используемый в языках программирования и языках спецификации, при котором переменная связывается с типом в момент присваивания значения, а не в момент объявления переменной. Таким образом, в различных участках программы одна и та же переменная может принимать значения разных типов.

**11 Операторы в Python. Перечислить и описать**

**Оператор в Python** — это символ, который выполняет операцию над одним или несколькими операндами. Операндом выступает переменная или значение, над которыми проводится операция.

Операторы Python бывают 7 типов:

* **Арифметические операторы**

1. Сложение (+)

2. Вычитание (-)

3. Умножение (\*)

4. Деление (/)

5. Возведение в степень (\*\*)

6. Деление без остатка (//)

7. Деление по модулю (остаток от деления) (%)

* **Операторы сравнения**

1. Меньше ()

2. Больше (>)

3. Меньше или равно (<=)

4. Больше или равно (>=)

5. Равно (==)

6. Не равно (!=)

* **Операторы присваивания**

1. Присваивание (=)

2. Сложение и присваивание (+=)

3. Вычитание и присваивание (-=)

4. Деление и присваивание (/=)

5. Умножение и присваивание (\*=)

6. Деление по модулю и присваивание (%=)

7. Возведение в степень и присваивание (\*\*=)

8. Деление с остатком и присваивание (//=)

* **Логические операторы**

1. И (and)

2. Или (or)

3. Не (not)

* **Операторы принадлежности**

1. В (in)

2. Нет в (not in)

* **Операторы тождественности**

1. Это (is)

2. Это не (is not)

* **Битовые операторы**

1. Бинарное И (&)

2. Бинарное ИЛИ (|)

3. Бинарное ИЛИ НЕТ (^)

4. Инвертирующий оператор (~)

5. Бинарный сдвиг влево (<>)

**12 Функции в Python. Назначение и применение.**

**Функции** представляют блок кода, который выполняет определенную задачу и который можно повторно использовать в других частях программы. Формальное определение функции:

def имя\_функции ([параметры]):

инструкции

Для вызова функции указывается имя функции, после которого в скобках идет передача значений для всех ее параметров. Например:

def say\_hello(name):

print("Hello,",name)

say\_hello("Tom")

Некоторые параметры функции мы можем сделать необязательными, указав для них **значения по умолчанию** при определении функции. Например:

def say\_hello(name="Tom"):

print("Hello,", name)

say\_hello()

say\_hello("Bob")

При передаче значений функция сопоставляет их с параметрами в том порядке, в котором они передаются. Использование **именованных параметров** позволяет переопределить порядок передачи:

def display\_info(name, age):

print("Name:", name, "\t", "Age:", age)

display\_info(age=22, name="Tom")

Функция может **возвращать результат**. Для этого в функции используется оператор return, после которого указывается возвращаемое значение. Поскольку функция возвращает значение, то мы можем присвоить это значение какой-либо переменной и затем использовать ее.

def exchange(usd\_rate, money):

result = round(money/usd\_rate, 2)

return result result1 = exchange(60, 30000)

print(result1)

В Python функция может возвращать сразу несколько значений:

def create\_default\_user():

name = "Tom"

age = 33

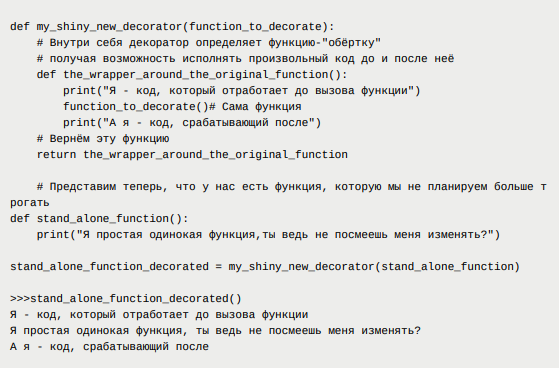
return name, age

**13 Декораторы в Python. Применение и особенности**

**Декоратор** – это функция, которая принимает другую функцию в качестве аргумента. **Декоратор** модифицирует или улучшает принятую функцию и выдает измененную. Это значит, что когда вы вызываете декорированную функцию, вы получите функцию, которая может иметь небольшие отличия, в виде дополнительных функций, совмещенных с базовым определением.

**Особенности:**

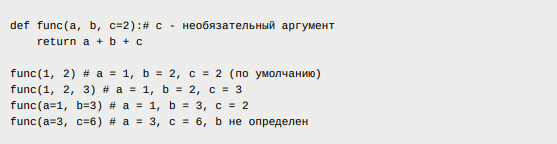
* Декораторы несколько замедляют вызов функции, не забывайте об этом.
* Вы не можете "раздекорировать" функцию. Безусловно, существуют трюки, позволяющие создать декоратор, который можно отсоединить от функции, но это плохая практика. Правильнее будет запомнить, что если функция декорирована — это не отменить.
* Декораторы оборачивают функции, что может затруднить отладку.



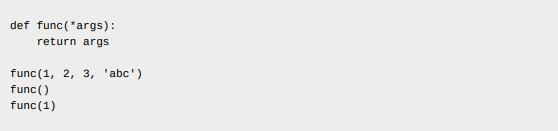
**14 Аргументы функций. Варианты передачи. Аннотирование типов**

Функция может принимать произвольное количество аргументов или не принимать их вовсе.

Варианты передачи:



Функция также может принимать переменное количество позиционных аргументов, тогда перед именем ставится \*

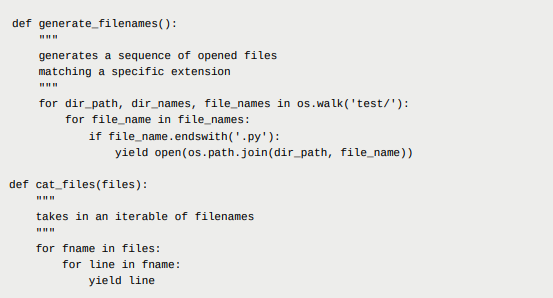


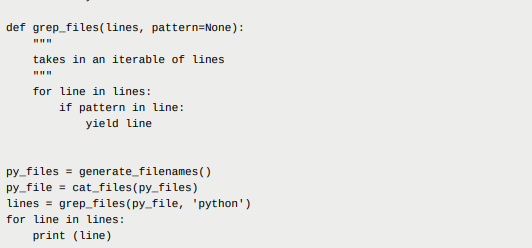
Как видно из примера, args - это кортеж из всех переданных аргументов функции, и с переменной можно работать также, как и с кортежем.

**15 Генераторы в Python. Применение и особенности**

**Генераторы** — это функции, которые можно приостанавливать и возобновлять во время их выполнения, при этом они возвращают объект, который можно итерировать.

Генераторы идеально подходят для чтения большого количества больших файлов, поскольку они выдают данные по одному фрагменту за раз, независимо от размера входного потока. Они также могут привести к более чистому коду путем разделения процесса итерации на более мелкие компоненты.





**16 Классы в Python. Стандартные методы классов.**

Класс — это пользовательский тип, состоящий из методов и атрибутов. Класс является шаблоном или формальным описанием объекта, а объект представляет экземпляр этого класса, его реальное воплощение.

Создание класса:

class название\_класса:

методы\_класса

Чтобы создать экземпляры классов, нужно вызвать класс с использованием его имени и передать аргументы, которые принимает метод **init**.

Для создания объекта (экземпляра) класса используется следующий синтаксис:

название\_объекта = название\_класса([параметры])

Каждая такая запись генерирует свой объект класса. Отличие от C++ в том, что в C++ описание класса — это лишь объявление, а в питоне — это создание объекта.

Внутри классов вы можете определить функции или методы, которые являются частью этого класса.

В классе метод определяется с помощью оператора def. Первый параметр каждого метода всегда ссылается на сам объект. В соответствии с общепринятым соглашением для этого параметра применяется имя self. Все операции, затрагивающие атрибуты объекта, должны явно ссылаться на переменную self.

Стандартные **методы**:

* \_\_init\_\_(self, [...) инициализатор класса
* \_\_del\_\_(self) деструктор
* \_\_cmp\_\_(self, other) базовый метод сравнения
* \_\_add\_\_(self, other) сложение

Про методы вообще информации не много

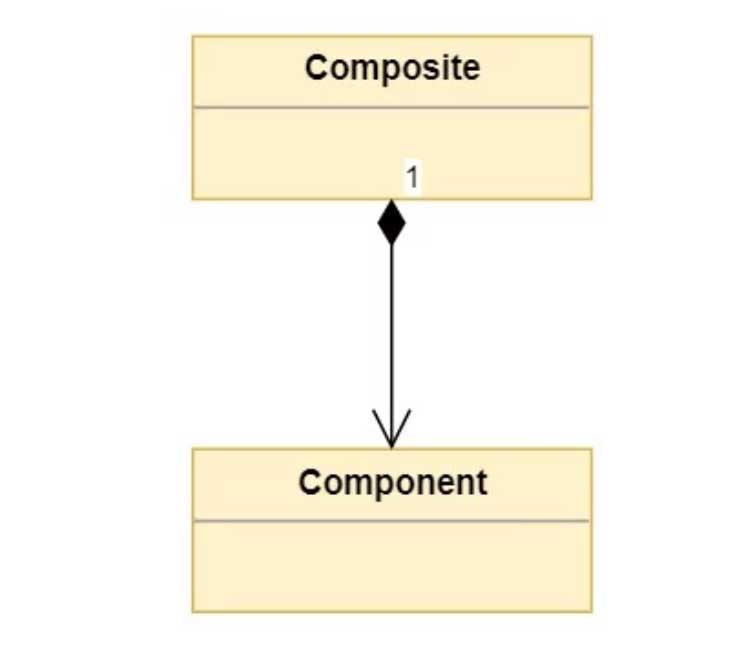
**17 Наследование, композиция, агрегация в Python**

Наследование: можно создавать специализированные классы на основе базовых. Это позволяет нам избегать написания повторного кода.

Ключевыми понятиями наследования являются подкласс и суперкласс. Производный (дочерний) наследует от базового (родительского) все публичные атрибуты и методы.

Класс называется абстрактным, если он предназначен только для наследования. Экземпляры абстрактного класса обычно не имеют большого смысла.

Композиция — это концепция, которая моделирует отношения. Она позволяет создавать сложные типы, комбинируя объекты других типов. Это означает, что класс Composite может содержать объект другого класса Component.



Отдельный класс задач, который не может обойтись без циклов, называется агрегированием данных. К таким задачам относятся поиск максимального, минимального, суммы, среднего арифметического и т.п. Их главная особенность в том, что результат зависит от всего набора данных. Для расчёта суммы нужно сложить все числа, для вычисления максимального нужно сравнить все числа.

sum\_numbers\_from\_range(5, 7) # 5 + 6 + 7 = 18

sum\_numbers\_from\_range(1, 2) # 1 + 2 = 3

# [1, 1] диапазон с одинаковым началом и концом – тоже диапазон

# он в себя включает ровно одно число – саму границу диапазона

sum\_numbers\_from\_range(1, 1) # 1

sum\_numbers\_from\_range(100, 100) # 100

1**8 Numpy. Назначение и основные возможности**

**NumPy** (сокращенно от *Numerical Python*)— библиотека с [открытым](https://ru.wikipedia.org/wiki/%D0%9E%D1%82%D0%BA%D1%80%D1%8B%D1%82%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%BD%D0%BE%D0%B5_%D0%BE%D0%B1%D0%B5%D1%81%D0%BF%D0%B5%D1%87%D0%B5%D0%BD%D0%B8%D0%B5) [исходным кодом](https://ru.wikipedia.org/wiki/%D0%98%D1%81%D1%85%D0%BE%D0%B4%D0%BD%D1%8B%D0%B9_%D0%BA%D0%BE%D0%B4) для [языка программирования](https://ru.wikipedia.org/wiki/%D0%AF%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) [Python](https://ru.wikipedia.org/wiki/Python).

Возможности:

* поддержка многомерных [массивов](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2) (включая [матрицы](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%82%D1%80%D0%B8%D1%86%D0%B0_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)));
* поддержка [высокоуровневых](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%81%D0%BE%D0%BA%D0%BE%D1%83%D1%80%D0%BE%D0%B2%D0%BD%D0%B5%D0%B2%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) математических функций, предназначенных для работы с многомерными массивами.

Назначение:

Математические алгоритмы, реализованные на [интерпретируемых языках](https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D1%82%D0%B5%D1%80%D0%BF%D1%80%D0%B5%D1%82%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) (например, Python), часто работают гораздо медленнее тех же алгоритмов, реализованных на [компилируемых языках](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D0%B8%D0%BB%D0%B8%D1%80%D1%83%D0%B5%D0%BC%D1%8B%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) (например, [Фортран](https://ru.wikipedia.org/wiki/%D0%A4%D0%BE%D1%80%D1%82%D1%80%D0%B0%D0%BD), [Си](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)), [Java](https://ru.wikipedia.org/wiki/Java)). Библиотека NumPy предоставляет реализации [вычислительных](https://ru.wikipedia.org/wiki/%D0%92%D1%8B%D1%87%D0%B8%D1%81%D0%BB%D0%B8%D1%82%D0%B5%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0) [алгоритмов](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) (в виде функций и операторов), оптимизированные для работы с многомерными массивами. В результате любой алгоритм, который может быть выражен в виде последовательности операций над массивами (матрицами) и реализованный с использованием NumPy, работает так же быстро, как эквивалентный код, выполняемый в [MATLAB](https://ru.wikipedia.org/wiki/MATLAB).

**19 Типы данных в Numpy**

В NumPy доступны следующие типы данных:

bool\_ Логический тип (истина или ложь), хранящийся в виде байта.

int\_ Целочисленный тип установленный по умолчанию (такой же, как C long, как правило это либо int64 либо int32).

intc Идентичен C int (int32 или int64).

intp Целочисленный тип, используемый для индексирования (такой же, как C ssize\_t, как правило это либо int64 либо int32).

int8 Целые числа в диапазоне от -128 по 127 (числа размером 1 байт).

int16 Целые числа в диапазоне от -32768 по 32767, (числа размером 2 байта).

int32 Целые числа в диапазоне от -2147483648 по 2147483647, (числа размером 4 байта).

int64 Целые числа в диапазоне от -9223372036854775808 по 9223372036854775807, (числа размером 8 байт).

uint8 Целые числа в диапазоне от 0 по 255 (числа размером 1 байт).

uint16 Целые числа в диапазоне от 0 по 65535 (числа размером 2 байта).

uint32 Целые числа в диапазоне от 0 по 4294967295 (числа размером 4 байта).

uint64 Целые числа в диапазоне от 0 по 18446744073709551615 (числа размером 8 байт).

float\_ То же самое что и float64.

float16 Вещественные числа половинной точности: 1 бит знака, 5 бит экспоненты, 10 бит мантисы (числа размером 2 байта).

float32 Вещественные числа одинарной точности: 1 бит знака, 8 бит экспоненты, 23 бита мантисы (числа размером 4 байта).

float64 Вещественные числа двойной точности: 1 бит знака, 11 бит экспоненты, 52 бита мантисы (числа размером 8 байт).

complex\_ То же самое что и complex128.

complex64 Комплексные числа в которых действительная и мнимая части представлены двумя вещественными числами типа float32.

complex128 Комплексные числа в которых действительная и мнимая части представлены двумя вещественными числами типа float64.

В NumPy существует 5 базовых числовых типов: булевы числа (тип bool: 0 - ложь и 1 - истина), целые числа (тип int), беззнаковые целые числа (тип uint), вещественные числа (числа с плавающей запятой, тип float) и комплексные числа (тип complex). У некоторых после указания типа следует количество бит необходимое для хранения такого числа в памяти (16, 32, 64 или 128), но некоторые, такие как int\_ или intp зависят от используемой платформы (32 или 64-разрядные машины).

**20 Основные операции над массивами и скалярами в Numpy**

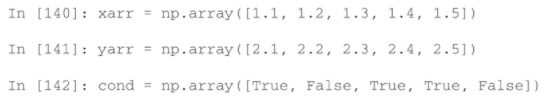
NumPy включает в себя набор пакетов для решения специализированных задач, например:

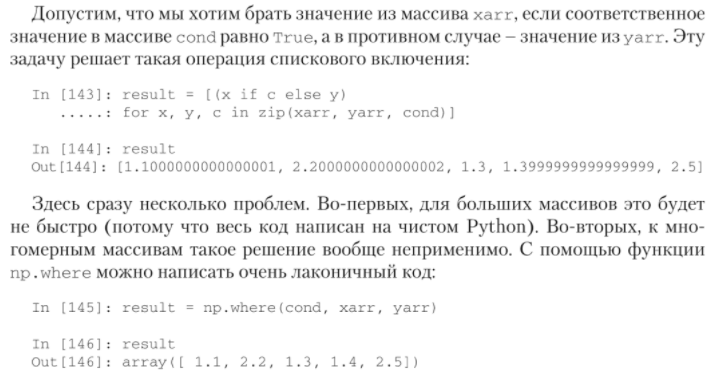
* numpy.linalg — реализует операции линейной алгебры (простое умножение векторов и матриц есть в базовом варианте);
* numpy.random — реализует функции для работы со случайными величинами;
* numpy.fft — реализует прямое и обратное преобразование Фурье.

Основные операции:

copyto(dst, src[, casting, where]) Копирует данные из одного массива в другой с выполнением транслирования если это необходимо.  
**21 Запись логических условий в виде операций с массивами**







**22 Индексирование и вырезание. Булево индексирование**

Механизм индексирования участвует в выполнении запросов; он вызывается в соответствии с планом, построенным на этапе оптимизации. Оптимизатор, перебирая и оценивая различные пути выполнения запроса, должен понимать возможности всех методов доступа, которые потенциально можно применить. Механизм индексирования позволяет PostgreSQL одинаково работать с самыми разными методами доступа, учитывая их возможности.

Основные способы сканирования:

1) Индексное сканирование

2) Сканирование по битовой карте

Индексное сканирование хорошо работает, когда речь идет всего о нескольких значениях. Однако при увеличении выборки возрастают шансы, что придется возвращаться к одной и той же табличной странице несколько раз. Поэтому в таком случае оптимизатор переключается на сканирование по битовой карте.

3) Последовательное сканирование

4) Покрывающие индексы

Как правило, основная задача метода доступа — вернуть идентификаторы подходящих строк таблицы, чтобы механизм индексирования мог прочитать из них необходимые данные. Но что, если индекс уже содержит все необходимые для запроса данные? Такой индекс называется покрывающим (covering), и в этом случае оптимизатор может применить исключительно индексное сканирование.

5) Null

Неопределенные значения играют важную роль в реляционных базах данных как удобный способ представления того факта, что значение не существует или не известно.

6) Индексы по нескольким полям

Как правило, если на первое поле не наложено условие, индекс использоваться не будет. Но в некоторых случаях оптимизатор может счесть, что это все-таки выгоднее последовательного сканирования.

7) Индексы по выражениям

Функциональный индекс создается не по полю таблицы, а по произвольному выражению; оптимизатор будет принимать во внимание такой индекс для условий вида «индексированное-выражение оператор выражение».

8) Частичные индексы

9) Сортировка

10) Параллельное построение

Обычно построение индекса требует установки блокировки типа SHARE на таблицу. Такая блокировка позволяет читать данные из таблицы, но запрещает любые изменения, пока строится индекс.

**23 Срезы в Numpy**

Срезы позволяют извлекать части массива для создания новых массивов. Когда вы используете срезы для списков Python, результирующие массивы — это копии, но в NumPy они являются представлениями одного и того же лежащего в основе буфера.

В зависимости от части массива, которую необходимо извлечь, нужно использовать синтаксис среза; это последовательность числовых значений, разделенная двоеточием (:) в квадратных скобках.

Чтобы получить, например, часть массива от второго до шестого элемента, необходимо ввести индекс первого элемента — 1 и индекса последнего — 5, разделив их :.

>>> a = np.arange(10, 16)

>>> a

array([10, 11, 12, 13, 14, 15])

>>> a[1:5]

array([11, 12, 13, 14])

Если нужно извлечь элемент из предыдущего отрезка и пропустить один или несколько элементов, можно использовать третье число, которое представляет собой интервал последовательности. Например, со значением 2 результат будет такой:

>>> a[1:5:2]

array([11, 13])

Чтобы лучше понять синтаксис среза, необходимо рассматривать и случаи, когда явные числовые значения не используются. Если не ввести первое число, NumPy неявно интерпретирует его как 0 (то есть, первый элемент массива). Если пропустить второй — он будет заменен на максимальный индекс, а если последний — представлен как 1. То есть, все элементы будут перебираться без интервалов.

>>> a[::2]

array([10, 12, 14])

>>> a[:5:2]

array([10, 12, 14])

>>> a[:5:]

array([10, 11, 12, 13, 14]

**24 Векторизованные операции в Numpy**

Функция Numpy vectorize принимает функцию python (py func) и возвращает векторизованную версию функции.

Векторизованная версия функции принимает последовательность объектов или массивов NumPy в качестве входных данных и вычисляет функцию Python над каждым элементом входной последовательности. Векторизация Numpy по существу функционирует подобно python map (), но с дополнительной функциональностью – механизмом вещания NumPy.

Необходимые параметры:

py func : Функция, которую мы хотим применить к последовательности объектов

Дополнительные параметры:

типы : Выходные типы функции могут быть указаны в виде строки или списка типов данных. Если типы не указаны, а кэш установлен в True , тип вывода определяется вызовом первого элемента ввода.

doc : Для указания строки документа созданного. Если не указано, будет использоваться исходная строка документа функции( если таковая имеется).

кэш : Если True , то кэшируйте первый вызов функции, определяющий количество выходов, если o типов не предусмотрено.

**25 Pandas. Назначение и основные возможности**

Рandas это высокоуровневая Python библиотека для анализа данных. Построена она поверх более низкоуровневой библиотеки NumPy (написана на Си), что является большим плюсом в производительности. В экосистеме Python, pandas является наиболее продвинутой и быстроразвивающейся библиотекой для обработки и анализа данных.

Чтобы эффективно работать с pandas, необходимо освоить самые главные структуры данных библиотеки: DataFrame и Series.

Основные возможности библиотеки:

* Объект *DataFrame* для манипулирования индексированными массивами двумерных данных[[4]](https://ru.wikipedia.org/wiki/Pandas#cite_note-4)
* Инструменты для обмена данными между структурами в памяти и файлами различных форматов
* Встроенные средства совмещения данных и способы обработки отсутствующей информации
* Переформатирование наборов данных, в том числе создание сводных таблиц
* Срез данных по значениям индекса, расширенные возможности индексирования, выборка из больших наборов данных
* Вставка и удаление столбцов данных
* Возможности группировки позволяют выполнять трёхэтапные операции типа «разделение, изменение, объединение» ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA) *split-apply-combine*).
* Слияние и объединение наборов данных
* Иерархическое индексирование позволяет работать с данными высокой размерности в структурах меньшей размерности
* Работа с временными рядами: формирование временных периодов и изменение интервалов и так далее

**26 Типы данных Dataframe и Series в Pandas**

Ядром pandas являются две структуры данных, в которых происходят все операции:

Series и Dataframes

Series — это структура, используемая для работы с последовательностью одномерных данных, а Dataframe — более сложная и подходит для нескольких измерений.

Пусть они и не являются универсальными для решения всех проблем, предоставляют отличный инструмент для большинства приложений. При этом их легко использовать, а множество более сложных структур можно упросить до одной из этих двух.

Однако особенности этих структур основаны на одной черте — интеграции в их структуру объектов index и labels (метки). С их помощью структурами становится очень легко манипулировать.

Series — это объект библиотеки pandas, спроектированный для представления одномерных структур данных, похожих на массивы, но с дополнительными возможностями. Его структура проста, ведь он состоит из двух связанных между собой массивов. Основной содержит данные (данные любого типа NumPy), а в дополнительном, index, хранятся метки.

Dataframe — это табличная структура данных, напоминающая таблицы из Microsoft Excel. Ее главная задача — позволить использовать многомерные Series. Dataframe состоит из упорядоченной коллекции колонок, каждая из которых содержит значение разных типов (числовое, строковое, булевое и так далее).

В отличие от Series у которого есть массив индексов с метками, ассоциированных с каждым из элементов, Dataframe имеет сразу два таких. Первый ассоциирован со строками (рядами) и напоминает таковой из Series. Каждая метка ассоциирована со всеми значениями в ряду. Второй содержит метки для каждой из колонок.

**27 Базовая функциональность Pandas. Доступ по индексу, выборка и фильтрация**

Базовые функции DataFrame

T Переставляет строки и столбцы.

axes Возвращает список с метками оси строк и метками оси столбцов в качестве единственных членов.

dtypes Возвращает типы в этом объекте.

empty Истинно, если NDFrame полностью пуст [нет элементов]; если какая-либо из осей имеет длину 0.

ndim Количество осей / размеров массива.

shape Возвращает кортеж, представляющий размерность DataFrame.

size Количество элементов в NDFrame.

values Numpy представление NDFrame.

head() Возвращает первые n строк.

tail() Возвращает последние n строк.

Выбирать отдельные элементы можно по принципу обычных массивов numpy, используя для этого индекс. Или же можно выбрать метку, соответствующую положению индекса. Таким же образом можно выбрать несколько элементов массива numpy с помощью следующей команды:

>>> s[0:2]

a 12

b -4

dtype: int64

Благодаря тому что основной библиотекой в pandas является NumPy, многие операции, применяемые к массивам NumPy, могут быть использованы и в случае с Series. Одна из таких — фильтрация значений в структуре данных с помощью условий.

Например, если нужно узнать, какие элементы в Series больше 8, то можно написать следующее:

>>> s[s > 8]

a 12

d 9

dtype: int64  
**28 Обработка отсутствующих данных в Pandas**

Pandas - это библиотека Python для анализа и обработки данных. Почти все операции в pandas вращаются вокруг DataFrame s, абстрактной структуры данных, специально созданной для обработки метрической тонны данных.

Первым шагом к правильному анализу данных является очистка и организация данных.

Если мы хотим отобразить список значений, которые должны рассматриваться как отсутствующие значения во всех столбцах, мы можем передать список кандидатов, которые мы хотим глобально рассматривать как отсутствующие значения, в параметр na\_values

После того, как мы определили все недостающие значения в DataFrame и правильно аннотировали их, есть несколько способов обработки недостающих данных.

Один из подходов - удалить все строки, содержащие пропущенные значения. Это легко сделать с помощью специально предназначенной для этого функции dropna()

inplace = True вносит все изменения в существующий DataFrame не возвращая новый. Без него вам пришлось бы переназначить DataFrame самому себе.

axis указывает, работаете ли вы со строками или столбцами: 0 - строки, а 1 - столбцы.

Вы можете контролировать , хотите ли вы удалить строки , содержащие по меньшей мере , 1 NaN или все NaN значения, установив , how параметр в dropna методе.

Заполнение недостающих значений:

1) Можно использовать функции .median() , .mode() и .mean() для столбца и указать их в качестве значения заполнения.

2) Вы также можете решить заполнить значения с пометкой NA постоянным значением. Например, вы можете ввести специальную строку или числовое значение.

3) Прямое заполнение отсутствующих значений фрейма данных.

df['Salary'].fillna(method='ffill', inplace=**True**)

4) Обратное заполнение отсутствующих значений фрейма данных

df['Salary'].fillna(method='bfill', inplace=**True**)

5) Этот метод использует математическую интерполяцию, чтобы определить, какое значение было бы на месте отсутствующего значения:

df['Salary'].interpolate(method='polynomial')

Очистка и предварительная обработка данных - очень важная часть каждого анализа данных и каждого проекта в области науки о данных.

**!!!!!29 Статистика в Pandas. Уникальные значения, счётчики значений**

Большое количество методов совместно вычисляет описательную статистику и другие связанные операции над DataFrame. Большинство из них являются агрегатами, такими как **sum (), mean (),** но некоторые из них, например **sumsum ()** , создают объект одинакового размера. Вообще говоря, эти методы принимают аргумент **оси** , как и *ndarray. {Sum, std, …},* но ось может быть указана по имени или целому числу.  
**30 Базовая функциональность Pandas.**

Объекты pandas ( Index , Series , DataFrame ) можно рассматривать как контейнеры для массивов, которые содержат фактические данные и выполняют фактические вычисления. Для многих типов базовым массивом является numpy.ndarray . Однако pandas и сторонние библиотеки могут расширять систему типов NumPy, добавляя поддержку настраиваемых массивов.

Основные функции серии

**axes** Возвращает список меток оси строк

dtype Возвращает dtype объекта.

empty Возвращает True, если серия пуста.

ndim Возвращает количество измерений базовых данных по определению 1.

size Возвращает количество элементов в базовых данных.

values Возвращает серию как ndarray.

**head()** Возвращает первые n строк.

tail() Возвращает последние n строк.

Базовые функции DataFrame

T Переставляет строки и столбцы.

axes Возвращает список с метками оси строк и метками оси столбцов в качестве единственных членов.

dtypes Возвращает типы в этом объекте.

empty Истинно, если NDFrame полностью пуст [нет элементов]; если какая-либо из осей имеет длину 0.

ndim Количество осей / размеров массива.

shape Возвращает кортеж, представляющий размерность DataFrame.

size Количество элементов в NDFrame.

values Numpy представление NDFrame.

head() Возвращает первые n строк.

tail() Возвращает последние n строк.

**31 Переиндексация и удаление элементов в Pandas**

Переиндексирование df.reindex()

После объявления в структуре данных объект Index нельзя менять. Но с помощью операции переиндексирования это можно решить.

>>> ser = pd.Series([2,5,7,4], index=['one','two','three','four']) >>> ser

one 2

two 5

three 7

four 4

dtype: int64

Для того чтобы провести переиндексирование объекта Series библиотека pandas предоставляет функцию reindex(). Она создает новый объект Series со значениями из другого Series, которые теперь переставлены в соответствии с новой последовательностью меток.

При операции переиндексирования можно поменять порядок индексов, удалить некоторые из них или добавить новые. Если метка новая, pandas добавит NaN на место соответствующего значения.

>>> ser.reindex(['three','four','five','one'])

three 7.0

four 4.0

five NaN

one 2.0

dtype: float64

Как видно по выводу, порядок меток можно поменять полностью. Значение, которое раньше соответствовало метке two, удалено, зато есть новое с меткой five.

Удаление

Удалить строку или колонку не составит труда, потому что метки используются для обозначения индексов и названий колонок.

В этом случае pandas предоставляет специальную функцию для этой операции, которая называется drop(). Метод возвращает новый объект без элементов, которые необходимо было удалить.

Например, возьмем в качестве примера случай, где из объекта нужно удалить один элемент. Для этого определим базовый объект Series из четырех элементов с 4 отдельными метками.

>>> ser = pd.Series(np.arange(4.), index=['red','blue','yellow','white'])

>>> ser

red 0.0

blue 1.0

yellow 2.0

white 3.0

dtype: float64

Теперь, предположим, необходимо удалить объект с меткой yellow. Для этого нужно всего лишь указать ее в качестве аргумента функции drop().

>>> ser.drop('yellow')

red 0.0

blue 1.0

white 3.0

dtype: float64

Для удаления большего количества элементов, передайте массив с соответствующими индексами.