Оглавление

[1. Многопроцессорные архитектуры с общей и разделяемой памятью – специфика и сравнение 2](#_Toc124083170)

[2. Подходы к декомпозиции крупных вычислительных задач на подзадачи для параллельного исполнения 2](#_Toc124083171)

[3. Модели параллельного программирования и их сочетаемость с архитектурами параллельных вычислительных систем 3](#_Toc124083172)

[4. Профилирование реализации алгоритмов на Python, принципы решения задачи оптимизации производительности алгоритма 3](#_Toc124083173)

[5. Проблема Global Interpreter Lock в Python и способы обхода ее ограничений 3](#_Toc124083174)

[6. Технологический стек Python для обработки и анализа данных, Python как glue language, специфика библиотеки NumPy и ее роль в экосистеме Python 3](#_Toc124083175)

[7. Организация массивов в NumPy: хранение данных, создание массивов, принципы реализации операций с едиными исходными данными 4](#_Toc124083176)

[8. Универсальные функции и применение функций по осям в NumPy 4](#_Toc124083177)

[9. Принцип распространения значений при выполнении операций в NumPy: общий алгоритм и примеры 6](#_Toc124083178)

[10. Маскирование и прихотливое индексирование в NumPy 7](#_Toc124083179)

[11. Векторизация в numpy: ключевые параметры функции, примеры применения, использование обобщенной сигнатуры функции 7](#_Toc124083180)

[12. Numba: принципы работы, базовые примеры использования 8](#_Toc124083181)

[13. Организация Pandas DataFrame и организация индексации для DataFrame и Series 8](#_Toc124083182)

[14. Применение универсальных функций и работа с пустыми значениями в Pandas 9](#_Toc124083183)

[15. Объединение данных из нескольких Pandas DataFrame: общая логика и примеры 10](#_Toc124083184)

[16. Операция GroupBy в Pandas DataFrame и реализация в ней подхода «разбиение, применение и объединение» 10](#_Toc124083185)

[17. Специфика текстовых и бинарных файлов, форматы файлов CSV и Pickle, представление данных в этих форматах и взаимодействие с ними в Python 10](#_Toc124083186)

[18. Задача сериализации и десериализации, описание формата файла JSON и пример описания данных в этом формате и взаимодействия с ним в Python 12](#_Toc124083187)

[19. Формат XML и модель DOM: общая характеристика, пример описания данных в XML и DOM, работа с ними с помощью библиотеки BeautifulSoup 12](#_Toc124083188)

[20. Форматы файлов NPY и HDF общая характеристика, пример взаимодействие с данными этих форматов в Python 13](#_Toc124083189)

[21. Взаимодействие с Excel из Python с помощью XLWings: принципы работы и примеры использования 14](#_Toc124083190)

[22. Основы работы с регулярными выражениями: базовый синтаксис, примеры использования модуля re в Python 15](#_Toc124083191)

[23. Сегментация и токенезация текста на естественном языке, стеммминг и лемматизация, примеры на Python 15](#_Toc124083192)

[24. Расстояние Левеншнтейна: определение, алгоритм эффективного поиска оптимального редакционного предписания, пример поиска на Python 15](#_Toc124083193)

[25. Векторное представление текста на естественном языке: общий алгоритм подходов TF; TF-IDF 16](#_Toc124083194)

[26. Модуль multiprocessing – назначение и основные возможности, API multiprocessing.Pool 16](#_Toc124083195)

[27. Различия между потоками и процессами, различие между различными планировщиками в Dask 17](#_Toc124083196)

[28. Граф зависимостей задач – суть структуры данных, ее построение и использование в Dask 18](#_Toc124083197)

[29. Dask.Array – структура данных, специфика реализации и применения, процедура создания 18](#_Toc124083198)

[30. Dask.Array – поддерживаемые операции и отличия от NumPy ndarray 19](#_Toc124083199)

[31. Dask.Bag - структура данных, специфика реализации и применения, процедура создания DaskBag 20](#_Toc124083200)

[32. Организация вычислений с помощью Map / Filter / Reduce : общий принцип и специфика параллельной реализации обработки данных в Dask.Bag 20](#_Toc124083201)

[33. API Dask.Bag – функции мэппинга, фильтрации и преобразования 21](#_Toc124083202)

[34. API Dask.Bag – функции группировки и свертки 22](#_Toc124083203)

**Теоретические вопросы к экзамену «Технологии обработки больших данных» (2022 г.)**

1. Многопроцессорные архитектуры с общей и разделяемой памятью – специфика и сравнение

Принципы многопроцессорной архитектуры с общей памятью (shared memory):

* несколько процессоров работают независимо, но совместно используют общую память
* изменения в памяти, осуществляемые одним процессором, видны всем другим процессорам

Преимущества и недостатки:

* Привычная модель программирования за счет единого адресного пространства
* Высокая скорость и низкая латентность обмена данными между параллельными задачами
* Низкая масштабируемость (обычно до 16 процессоров) из-за геометрического роста нагрузки на шину CPU-RAM
* Проблема поддержания когерентности кэшей
* Трудоемкая организация эффективного использование памяти в NUMA-системах
* Необходимость синхронизации при доступе к общим данным (критические секции)

Принципы многопроцессорной архитектуры с распределенной памятью (distributed memory):

* несколько процессоров работают с собственной памятью, недоступной напрямую для других процессоров (отсутствует общая адресация памяти)
* обмен данными между процессорами производится через коммуникационную сеть и явно определяется исполняемой программой

Преимущества и недостатки:

* Высокая масштабируемость
* Объем памяти растет пропорционально количеству ядер
* Возможность использовать недорогие массовые компоненты
* Специальные подходы к программированию: необходимость использования передачи сообщений (message passing)
* Сложность реализации некоторых структур данных и алгоритмов
* Высокая латентность и низкая скорость обмена данными между узлами
* Неоднородность, отказы узлов

1. Подходы к декомпозиции крупных вычислительных задач на подзадачи для параллельного исполнения

Весь процесс решения некоторой задачи на параллельной ВС можно разбить на следующие этапы:

* формулировка задачи;
* составление модели исследуемого в задаче объекта;
* определение метода решения задачи для получения необходимой результирующей информации на основании используемой модели;
* разработка алгоритма решения задачи на основе используемого метода и модели исследуемого в задаче объекта;
* выбор технологии программирования;
* разработка программы для соответствующей параллельной ВС на основе имеющегося алгоритма и получение результирующей информации после выполнения программы.

Существует две концепции, два базовых подхода к декомпозиции крупных задач на пользовательские истории – «горизонтальное» и «вертикальное» разбиение:

В случае «горизонтальной» декомпозиции, задачи разбиваются по типу работы (функции), которую необходимо выполнить, по компонентам, которые задействованы в работе. В этом случае при разбиении общей большой задачи разработчику будет выделена одна часть, тестировщику другая, техническому писателю третья и так далее. Фактически каждая из частей не приводит к конечному результату сама по себе, чтобы выпустить готовый функционал, необходима реализация всей совокупности связанных задач всеми участниками процесса.

«Вертикальный» метод декомпозиции напротив предполагает выделение более мелких задач, фич, функций таким, образом, что каждая такая пользовательская история может быть реализована и выпущена отдельно от остальных задач. При этом в разработку могут быть вовлечены различные роли, могут быть задействованы несколько модулей и систем.

1. Модели параллельного программирования и их сочетаемость с архитектурами параллельных вычислительных систем

Разделяемая память (shared memory):

* Аналогия - доска объявлений
* Подзадачи используют общее адресное пространство (оперативной памяти)
* Подзадачи взаимодействуют, асинхронно читая и записывая информацию в общем пространстве
* Реализация: многопоточные приложения, OpenMP

Передача сообщений (message passing):

* Аналогия – отправка писем с явным указанием отправителя и получателя
* Каждая подзадача работает с собственными локальными данными
* Подзадачи взаимодействуют за счет обмена сообщениями
* Реализация: MPI (message passing interface)

Параллельная обработка данных (data parallelization):

* Строго описанные глобальные операции над данными
* (Может обозначаться как чрезвычайная параллельность (embarrassingly parallel) - очень хорошо распараллеливаемые вычисления)
* Обычно данные равномерно разделяются по подзадачам
* Подзадачи выполняются как набор независимых операций
* Реализация может быть сделана как с помощью разделяемой памяти, так и с помощью передачи сообщений

Модель параллельного программирования на основе передачи сообщений

Основные характеристики модели на основе передачи сообщений:

* Набор задач, имеющих свою собственную локальную память во время вычислений
* Задачи могут находиться как на одной машине (в т. ч. с разделяемой памятью), так и на разных машинах
* Задачи обмениваются данными с помощью отсылки и приема сообщений явно описываемых в программном коде
* Зачастую передача данных подразумевает их сериализацию / десериализацию, что требует соответствующих накладных расходов
* Как правило передача данных требует совместной работы, выполняемой как задачей-отправителем, так и задачей-получателем

Программирование для модели на основе передачи сообщений:

* С точки зрения программирования модель на основе передачи сообщений выглядит как внедрение вызовов специализированной библиотеки в программный код.
* За реализацию параллелизма полностью отвечает программист, а не компилятор
* Общепринятым стандартом для модели параллельного программирования на основе передачи сообщений является библиотека MPI (Message Passing Interface).
* При отправке определяется процесс отправитель и буфер, содержащий передаваемую информацию
* При получении определяется процесс получатель и место для сохранения результата
* Возможны дополнительные атрибуты, обуславливающие передачу сообщения
* Передача сообщения несет много накладных расходов

Модель параллельного программирования на основе параллельной обработки данных

Параллельная обработка данных (data parallelization):

* Строго описанные глобальные операции над данными (может обозначаться как чрезвычайная параллельность (embarrassingly parallel) - очень хорошо распараллеливаемые вычисления)
* Обычно данные равномерно разделяются по подзадачам
* Подзадачи выполняются как последовательность независимых операций
* Реализация может быть сделана как с помощью разделяемой памяти, так и с помощью передачи сообщений

Основные характеристики модели на основе параллельной обработки данных:

* Основные параллельные задачи сфокусированы на выполнении операций над неким массивом данных
* Массив данных обычно организован в виде однородной структуры, например массива или гиперкуба
* Задачи обычно параллельно выполняют аналогичные операции над выделенными им фрагментами одного массива данных
* В реализации на архитектурах без разделяемой памяти массив данных делится на фрагменты, которые находятся в распоряжении отдельных задач
* Программирование для данной модели обычно представляет собой написания программы, оперирующей с конструкциями для параллельной обработки данных, например в виде вызовов специализированной библиотеки

1. Профилирование реализации алгоритмов на Python, принципы решения задачи оптимизации производительности алгоритма

Профиль - это множество статистических данных, определяющих, как часто и как долго выполняются различные части программы.

Профилирование включает в себя так же несколько подходов:

Метод пристального взгляда: просто садимся перед текстовым редактором, открываем код и думаем, где может быть проблема, пробуем починить, смотрим на результат, откатываемся, крайне неэффективный подход.

Ручное профилирование – перед выполнением спорного участка программы сохраняем в переменную текущее системное время (с точностью до микросекунд), а после заново получаем текущее время и вычитаем из него значение сохранённой переменной. Получаем (с достаточной для нас погрешностью) время выполнения анализируемого кода. Для эффективности, необходимо повторять несколько раз.

Профилирование с использованием инструментов – Данный метод применяется, когда разработчик не знает, почему программа работает не так, как надо, либо же если нет возможности применить ручное профилирование. Сами же инструменты делятся на два вида: статистический профайлер и событийный.

Python обладает встроенными модулями для профилирования – такими как:

cProfile - который используется для измерения времени выполнения программы. Модуль cProfiler предоставляет всю информацию о том, как долго выполняется программа и сколько раз функция вызывается в программе.

Profile – нативная реализация профайлера, в отличие от cProfile написанного на C, вследствие чего более быстрого, Profile написан на чистом питоне, от чего более медленный.

Принципы решения задачи оптимизации –

-Использование сторонних модулей и библиотек, например библиотека написанная на C будет работать заметно быстрее, что так же ускорит и программу.

-Избегание глобальных переменных - Python очень медленно обрабатывает доступ к глобальным переменным, особенно внутри циклов, использование глобальных переменных так же приводит к слабо структурированному коду.

-Использование встроенных инструментов python – применение встроенных инструментов так же приводит к ускорению работы программы, поскольку встроенные инструменты предварительно скомпилированы и оптимизированы.

1. Проблема Global Interpreter Lock в Python и способы обхода ее ограничений

Проблема Global Interpreter Lock (GIL)\_\_

Global Interpreter Lock - способ синхронизации потоков, используемый в рефернсной реализации Python (CPython) и в реализациях некоторых других интерпретируемых языков программирования.

* Интерпретатор CPython не является потоко-безопасным, т.к. некоторые ключевые структуры данных могут быть одновременно доступны только одному потку.
* GIL является самым простым и быстрым при исполнении однопоточных приложений способом обеспечения потоковой безопасности при одновременном обращении разных потоков к одним и тем же участкам памяти.
* Наличие GIL не является требованием языка программирования Python, а только спецификой реализации самого популярного интерпретатора CPython, существуют другие интерпретаторы Python не имеющие GIL.

Для обхода проблемы GIL для реализации параллельных вычислений в Python вместо многопоточного подхода с разделяемой памятью используется более тяжеловесная конструкция:

* множество процессов, в каждом из которых работает собственный интерпретатор с собственным GIL и имеется собственная копия данных и кода
* обмен данными между процессами обычно производится не через разделяемую память (это иногда возможно, но чревато ошибками), а через передачу данных и кода с помощью сериализации
* по сути, это вариация на тему модели параллельного программирования на основе передачи сообщений (реализуемой в т.ч. при вычислении на компьютере с разделяемой памятью)

1. Технологический стек Python для обработки и анализа данных, Python как glue language, специфика библиотеки NumPy и ее роль в экосистеме Python

NumPy (от Numerical Python) - библиотека (пакет) для Python, интегрированная с кодом на C и Fortran, решающая задачи математических расчетов и манипулирования массивами данных (в первую очередь - числовыми).

NumPy - это краеугольный камень технологического стека Python для научных расчетов и обработки данных. NumPy - открытая библиотека, поставляемая с базовым дистрибутивом Python.

NumPy используется практически во всех вычислительных приложениях, использующих Python. Сочетание реализации векторных функций на C и Fortran и оперирования данных на Python позволяет совместить высокую производительность и гибкость и удобство использования библиотеки. В этом смысле NumPy является ярким примером использования концепции "Python as a glue language".

1. Организация массивов в NumPy: хранение данных, создание массивов, принципы реализации операций с едиными исходными данными

Создать массив numpy можно тремя способами:

1. из списков или кортежей Python
2. с помощью функций, которые предназначены для генерации массивов numpy
3. из данных, хранящихся в файле

Хранение данных:

Numpy-массивы хранят указатель на кусок память, где лежат все данные, плюс метаинформацию об этих данных: типа, размерность и т.д. Тогда, при последовательном обращении к элементам массива эффективно используется кэш процессора, что позволяет снизить количество кэш-промахов, так как работает механизм предвыборки

Выводы:

+ эффективно хранятся в памяти (для хранения значений используется непрерывная область

памяти с простой индексацией, как это принято в C или Fortran)

+ операции над массивами numpy могут быть реализованы на компилируемых языках (C, Fortran).

Это на порядок повышает скорость выполнения операций. Для массивов numpy в виде

высокоэффективных функций реализованы основные математические операции.

- не обладают гибкостью списков Python ("не прямоугольные" вложенные списки, разнотипные

элементы в списках)

- прежде всего ориентированы на работу с числовой информацией (т.е. имеют ограничения по

типам используемой информации)

**Примеры**

Создание ndarray на базе списка Python (не эффективный способ!):

*a = np.array([1,2,3,4,5,6,7,8])*

*A = np.array([[1, 2, 3], [4, 5, 6]])*

С помощью функции np.arange:

*import numpy as np*

*np\_array = np.arange(12)*

Построение обычного двухмерного массива ndarray:

b = np.array([[0, 1, 2, 3, 4], [5, 6, 7, 8, 9], [10, 11, 12, 13, 14]])

Построение np.array из данных, хранящихся в файле

a = np.load('example\_1.npy')

1. Универсальные функции и применение функций по осям в NumPy

Универсальные функции (ufuncs) - функции, выполняющие поэлементные операции над данными, хранящимися в массиве. Это векторные операции на базе простых функций, работающих с одним или несколькими скалярными значениями и возвращающими скаляр.

Основные универсальные функции:

операции сравнения: <, <=, ==, !=, >=, >

арифметические операции: +, -, \*, /, %, reciprocal, square

экспоненциальные функции: exp, expm1, exp2, log, log10, log1p, log2, power, sqrt

тригонометрические функции: sin, cos, tan, acsin, arccos, atctan

гиперболические функции: sinh, cosh, tanh, acsinh, arccosh, atctanh

побитовые операции: &, |, ~, ^, left\_shift, right\_shift

логические операции: and, logical\_xor, not, or

предикаты: isfinite, isinf, isnan, signbit

другие функции: abs, ceil, floor, mod, modf, round, sinc, sign, trunc

Пример:

array([[ 0, 1, 2, 3, 4],

[ 5, 6, 7, 8, 9],

[10, 11, 12, 13, 14]])

ar1.sum(axis=None)

Out[249]:

105

ar1.sum(axis=0)

Out[250]:

array([15, 18, 21, 24, 27])

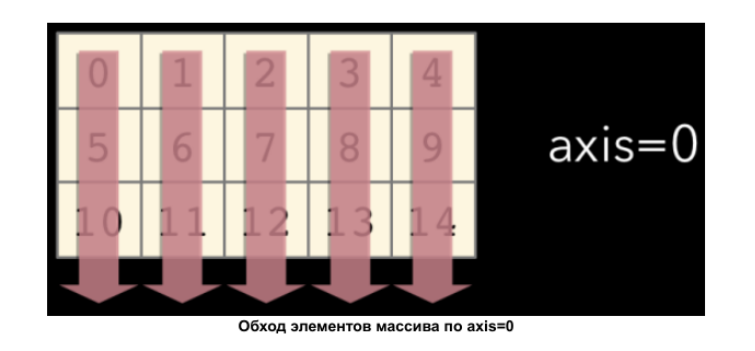
ar1.sum(axis=1)

Out[251]:

array([10, 35, 60])

Изображение выглядит как стол

Автоматически созданное описание



1. Принцип распространения значений при выполнении операций в NumPy: общий алгоритм и примеры

В качестве аргументов универсальных функций могут быть массивы с различной, но сравнимой формой. В этом случае применяется механизм распространения (broadcsting).

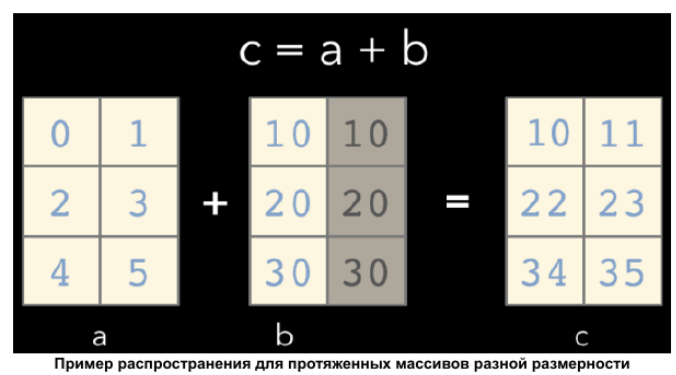
Изображение выглядит как текст

Автоматически созданное описание

np.arange(5) + 10

Out[272]:

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



Правила выполнения распространения:

соответствующие измерения двух массивов должны либо совпадать

либо одно из них должно быть равно единице.

Если в одном из массивов не хватает измерений, то считается, что недостающее количество измерений - это младшие измерения (измерения с наименьшими номерами), которым приписывается размерность 1

1. Маскирование и прихотливое индексирование в NumPy

Для индексирования мы можем использовать маски (маскирование): если массив NumPy содержит элементы типа bool , то элемент выбирается в зависимости от булевского значения.

f = np.arange(5)

fb = np.array([True, False, True, False, False])

f, fb

Out[447]:

(array([0, 1, 2, 3, 4]), array([ True, False, True, False, False]))

В "прихотливой" индексации передаются массивы индексов. Это дает возможность быстрого доступа к различным подмножествам.

В случае "прихотливой" индексации форма результата отражает форму массивов индексов(index arrays), а не форму индексированного массива.

1. Векторизация в numpy: ключевые параметры функции, примеры применения, использование обобщенной сигнатуры функции

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.

numpy.vectorize(pyfunc, otypes=None, doc=None, excluded=None, cache=False, signature=None)

1. Numba: принципы работы, базовые примеры использования

Numba генерирует оптимизированный машинный код из чистого кода Python, используя Инфраструктура компилятора LLVM, Скорость выполнения кода с использованием Numba сопоставима с аналогичным кодом на C, C ++ или Fortran. Сначала функция Python берется, оптимизируется и преобразуется в промежуточное представление numba, затем после вывода типа, который похож на вывод типа numpy (так что python float - это float64), он преобразуется в интерпретируемый код LLVM. Затем этот код передается компилятору LLVM "точно в срок" для выдачи машинного кода.

В numba исходили из того, что чаще всего требует ускорения (cpu bound) - математические вычисления, соответственно, они выделили часть языка, связанную с вычислениями и начали разгонять её, постепенно увеличивая «охват» (например, до недавнего времени не было поддержки строк, сейчас она появилась). Соответственно, в numba разгоняется не вся программа, а отдельные функции, это позволяет совместить высокую скорость и обратную совместимость с библиотеками, которые numba (пока) не поддерживает.

Чтобы ускорить функцию, надо перед её определением вписать декоратор njit

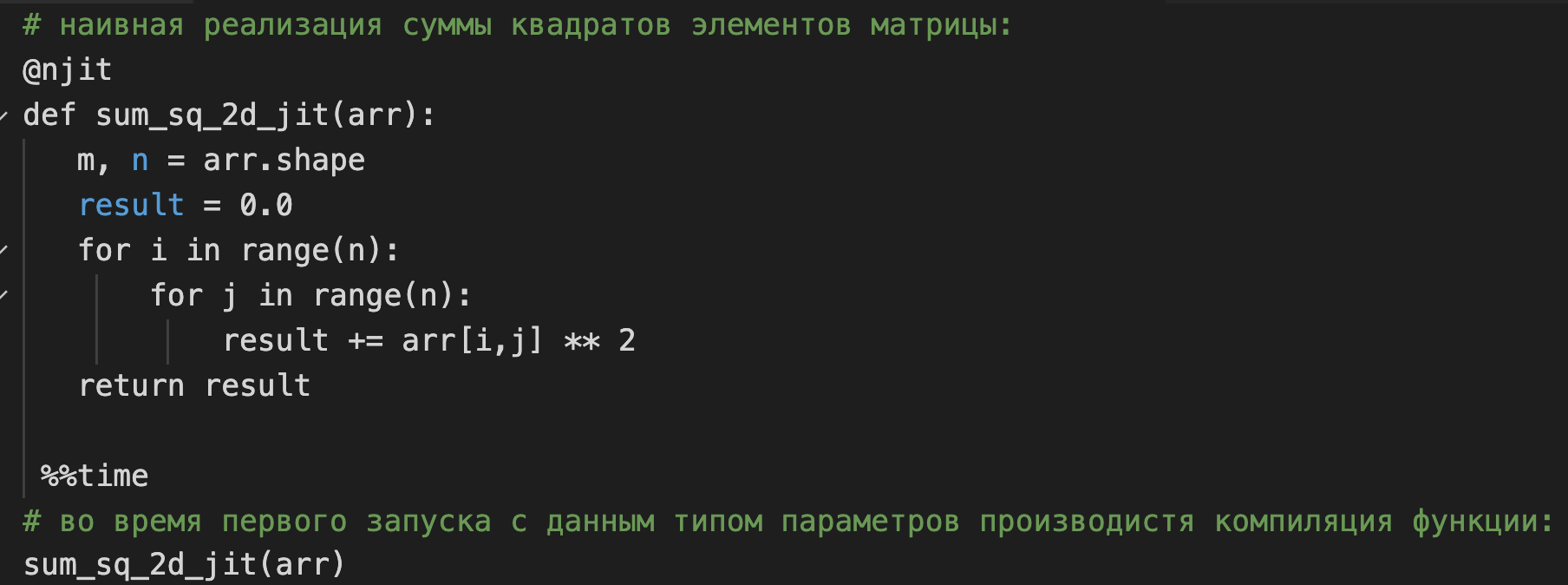
В нумбе каждой функции сопоставляется один или несколько типов входных и выходных аргументов, т.н. сигнатуры. При первом вызове функции сигнатура формируется и автоматически компилируется соответствующий бинарный код функции. При запуске с другими типами аргументов будут создаваться новые сигнатуры и новые бинарники (старые при этом сохраняются). Таким образом, «выход на режим» по скорости исполнения для каждой сигнатуры наступает, начиная со второго запуска с этими типами аргументов

При запуске функции с сигнатурой, указанной в декораторе (@njit(int32(int16, int16))), уже первый запуск будет быстрым: компиляция произойдёт в тот момент, когда питон увидит определение функции, а не при первом запуске. Сигнатур может быть несколько, порядок их следования имеет значение.

**Пример использования с помощью декоратора @jit.**

Декоратор @jit имеет два режим работы:

1. Режим nopython. Устанавливается параметром nopython=True или использованием декоратора @njit. Это рекомендуемый для использования и наиболее быстрый режим. Приводит к компиляции кода функции практически не использующего интерпретатор Python.
2. Режим object



1. Организация Pandas DataFrame и организация индексации для DataFrame и Series

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.

DataFrame - аналог двухмерного массива с гибкими индексами строк и гибкими именами столбцов. Аналогично тому, что двумерный массив можно рассматривать как упорядоченную последовательность выровненных столбцов, объект DataFrame можно рассматривать как упорядоченную последовательность выровненных объектов Series. Под «выравниванием» понимается то, что они используют один и тот же индекс.

Главная особенность этих структур - наличие объекта Index, который в них интегрирован.

Объекты Index являются метками осей и содержат другие метаданные. Вы уже знаете, как массив с метками превращается в объект Index, и что для него нужно определить параметр index в конструкторе. объекты index - неизменяемые. Это обеспечивает безопасность, когда нужно передавать данные между разными структурами. Есть методы для получения информации об индексах из структуры данных. Например, idmin() и idmax() - структуры, возвращающие индексы с самым маленьким и большим значениями.

У каждого объекта Index есть методы и свойства, которые нужны, чтобы узнавать значения.

в NumPy элементы по оси 0 принято рассматривать как строки (т.е. считается, что np1[1] - вернет

строку с индексом 1)

в Pandas аналогичная конструкция ( pd1[1] ) возвращает столбец типа Series.

1. Применение универсальных функций и работа с пустыми значениями в Pandas

Все универсальные функции библиотеки NumPy работают с объектами Series и DataFrame библиотеки Pandas. Результатом применения универсальной функции NumPy к объектам Pandas будет новый объект с сохранением индексов. При бинарных операциях над двумя объектами Series или DataFrame библиотека Pandas будет выравнивать индексы в процессе выполнения операции. Получившийся в итоге массив содержит объединение индексов двух исходных массивов. Недостающие значения будут отмечены как NaN («нечисловое значение»), с помощью которого библиотека Pandas отмечает пропущенные данные.

В Pandas в качестве пустых значений рассматривается значение NaN ("Not a Number"), поддерживаемое форматом чисел с плавающей точкой ( np.nan в NumPy) и значением None для объектов Python.

# получение маски пустых значений

sr8.isna()

# очистка от пустых значений:

sr8.dropna()

pd7.fillna(0.0) # заполнение NaN заданными значениями

replace() замена значений

1. Объединение данных из нескольких Pandas DataFrame: общая логика и примеры

Для объединения данных из нескольких dataframe в pandas есть несколько методов

Функция concat() в pandas используется для добавления столбцов или строк из одного фрейма данных в другой.

import pandas as pd

df1 = pd.DataFrame({'id': ['A01', 'A02', 'A03', 'A04'],

                    'Name': ['ABC', 'PQR', 'DEF', 'GHI']})

df2 = pd.DataFrame({'id': ['B05', 'B06', 'B07', 'B08'],

                    'Name': ['XYZ', 'TUV', 'MNO', 'JKL']})

  frames = [df1, df2]

  result = pd.concat(frames)

Здесь просто к одному датафрейму добавляется второй

Другой способ объединить фреймы данных - использовать столбцы в каждом наборе данных, которые содержат общие значения (общий уникальный идентификатор)

import pandas as pd

  df1 = pd.DataFrame({'id': ['A01', 'A02', 'A03', 'A04'],

                    'Name': ['ABC', 'PQR', 'DEF', 'GHI']})

  df3 = pd.DataFrame({'City': ['MUMBAI', 'PUNE', 'MUMBAI', 'DELHI'],  'Age': ['12', '13', '14', '12']})

result = pd.concat([df1, df3], axis=1, join='inner')

display(result)

-----

Ставим join = ‘inner’ для объединения с использованием общего поля

Объединение с помощью append()

import pandas as pd

df1 = pd.DataFrame({'id': ['A01', 'A02', 'A03', 'A04'],

                    'Name': ['ABC', 'PQR', 'DEF', 'GHI']})

  df2 = pd.DataFrame({'id': ['B05', 'B06', 'B07', 'B08'],

                    'Name': ['XYZ', 'TUV', 'MNO', 'JKL']})

  result = df1.append(df2)

display(result)

Можно соединять несколько датафреймов сразу:

result = df1.append([df2, df3])

С помощью append по факту так же как и с concat просто к одному датафрейму добавляется второй

Изображение выглядит как стол

Автоматически созданное описание

1. Операция GroupBy в Pandas DataFrame и реализация в ней подхода «разбиение, применение и объединение»

Операцию GroupBy удобно представить в виде последовательного применения операций: разбиение, применение и объединение (split, apply, combine):

split (шаг разбиения): включает разделение на части и группировку объекта DataFrame на основе

значений заданного ключа.

apply (шаг применения): включает вычисление какой-либо функции, обычно агрегирующей,

преобразование или фильтрацию в пределах отдельных групп.

combine (шаг объединения): во время шага выполняется слияние результатов предыдущих

операций в выходной массив.

Для DataFrame операцию "разбить, применить, объединить" можно реализовать с помощью метода groupby(), передав в него имя желаемого ключевого столбца. Функция groupby() возвращает не набор объектов DataFrame, а объект DataFrameGroupBy, который можно рассматривать как специально представление объекта DataFrame, готовое к группировке, но не выполняющее никаких фактически вычислений до этапа применения агрегирования (используется принцип отложенных вычислений).

Для получения результата нужно вызвать один из агрегирующих методов объекта DataFrameGroupBy, что приведет к выполнению соответствующих шагов применения/объединения.

На этапе применения у объектов GroupBy кроме обычных агрегирующих методов, таких как sum(), median() и т. п., имеются методы aggregate(), filter(), transform() и apply(), эффективно выполняющие множество полезных операций до объединения сгруппированных данных.

Метод aggregate() может принимать на входе строку, функцию или список и вычислять все сводные показатели сразу.

Метод apply() позволяет применять произвольную функцию к результатам группировки. В качестве параметра эта функция должна получать объект DataFrame, а возвращать или объект библиотеки Pandas (например, DataFrame, Series), или скалярное значение, в зависимости от возвращаемого значения будет вызвана соответствующая операция объединения.

# Специфика текстовых и бинарных файлов, форматы файлов CSV и Pickle, представление данных в этих форматах и взаимодействие с ними в Python

Форматы файлов можно разделить по способу организаци хранения данных на:

Tекстовые файлы - файл, содержащий текстовые данные, представленные как оследовательность символов (в основном печатных знаков, принадлежащих к определенному набору символов - кодовой таблице (кодировке)). Эти символы обычно сгруппированы в строки (lines, rows). В современных системах строки разделяются разделителями строк, иногда конец текстового файла также отмечается одним или более специальными знаками.

Двоичные (бинарные) файлы (binary file) - в узком смысле "не текстовые файлы". Файлы, байты в которых интерпретируются не как текст (при этом они могут включать фрагменты текста).

pickle - бинарный формат для сериализации объектов Python в поток байтов. Реализуется

как последовательность операций для выполнения на Pickle Virtual Machine. Формат

определяется прежде всего реализацией Pickle Virtual Machine.

Реализуется как последовательность операций для выполнения на Pickle Virtual Machine.

Формат определяется прежде всего реализацией Pickle Virtual Machine.

Pickle небезопасный формат сериализации так как не защищен от ошибочных или вредоносных

данных.

Модуль pickle позволяет также преобразовать объект в строку байтов и восстановить объект изстроки. Для этого предназначены две функции:

dumps(<Объект> [, <Протокол>] [, fix\_imports=True]) - производит сериализацию

объекта и возвращает последовательность байтов специального формата.

lоаds(<Последовательность байтов>[, fix\_imports=True] [, errors="strict"]) -

преобразует последовательность байтов обратно в объект.

Чтобы открыть файл с базой объектов, используется функция open() . Функция имеет следующий формат:

open (<Путь к файлу> [, flag="c"] [, protoco1=None] [, writeback=Fa1se])

Функция open() возвращает объект, с помощью которого производится дальнейшая работа с базой данных. Этот объект имеет следующие методы:

close() - закрывает файл с базой данных.

keys() - возвращает объект с ключами;

values() - возвращает объект to значениями;

items() - возвращает объект, поддерживающий итерации. На каждой итерации возвращается

кортеж, содержащий ключ и значение.

get(<Ключ> [, <Значение по умолчанию>] ) - если ключ присутствует; то метод возвращает

значение, соответствующее этому ключу. Если ключ отсутствует, то возвращается начение None или значение, указанное во втором nараметре;

setdefault(<Ключ> [, <Значение по умолчанию>] ) ---: если ключ nрисутствует, то метод

возвращает значение, соответствующее этому ключу. Если ключ отсутствует, то вставляет новый элемент со значением, указанным во втором параметре, и возвращает это значение. Если второй nараметр не указан, значением нового элемента будет None;

рор (<Ключ> [, <Значение по умолчанию>] ) - удаляет элемент с указанным ключом и

возвращает его значение. Если ключ отсутствует, то возвращается значение из второго

nараметра. Если ключ отсутствует, и второй nараметр не указан, то возбуждается исключение KeyError;

popitem() - удаляет произвольный элемент и возвращает кортеж из ключа и значения. Если

файл nустой, возбуждается исключение KeyError;

clear () - удаляет все элементы. Метод ничего не возвращает в качестве значения;

update () - добавляет элементы. Метод изменяет текущий объект и ничего не возвращает.

1. Задача сериализации и десериализации, описание формата файла JSON и пример описания данных в этом формате и взаимодействия с ним в Python

Сериализация (serialization) - процесс перевода какой-либо структуры данных в

последовательность битов. Обратной к сериализации является операция десериализации

(deserialization) - восстановление начального состояния структуры данных из битовой

последовательности.

Сериализация используется:

для передачи объектов по сети и для сохранения их в файлы.

для сохранения состояния приложения или некоторых его структур данных, хранения в

файле и последующего восстановления данных.

JSON (JavaScript Object Notation) - текстовый формат обмена данными, основанный на

JavaScript. JSON легко читается людьми. Несмотря на происхождение от JavaScript, формат

считается независимым от языка и может использоваться практически с любым языком

программирования.

Синтаксис JSON

Массив: [значение1, значение2, значениеN]

Объект (можно интерпретировать как словарь в Python): {"имя1":значение1,

"имя2":значение2, "имяN":значениеN} , имена (ключи) здесь должны быть строками.

Пример: {"имя1":"строка", "имя2":13, "имя3":true, "имя4":false, "имя5":null}

Сложный объект: {"имя1":значение1,

"имя2": {"имя2\_1":значение2\_1, "имя2\_2":значение2\_2} }

Список литералов JSON:

cтрока (заключается только в двойные кавычки: "example" )

число (целые и числа с плавающей точкой)

логическое значение ( true , false )

значение

1. Формат XML и модель DOM: общая характеристика, пример описания данных в XML и DOM, работа с ними с помощью библиотеки BeautifulSoup

XML (англ. eXtensible Markup Language) - расширяемый язык разметки.

Цели создания:

Эффективное описание (структурирование) данных.

Обмен данными между информационными системами (в первую очередь – через интернет).

Основные особенности:

Простой синтаксис.

Удобство создания и обработки документов программами.

Удобство чтения и создания документов человеком (профессиональными пользователями).

Удобство обмена документами в интернете.

Гибкость применения и расширяемость - возможнсть создавать собственные расширения XML.

Очень широкая распространенность и доступность инструментов.

Расширение XML - это конкретная грамматика, созданная на базе XML и представленная

словарём тегов и их атрибутов, а также набором правил, определяющих какие атрибуты и

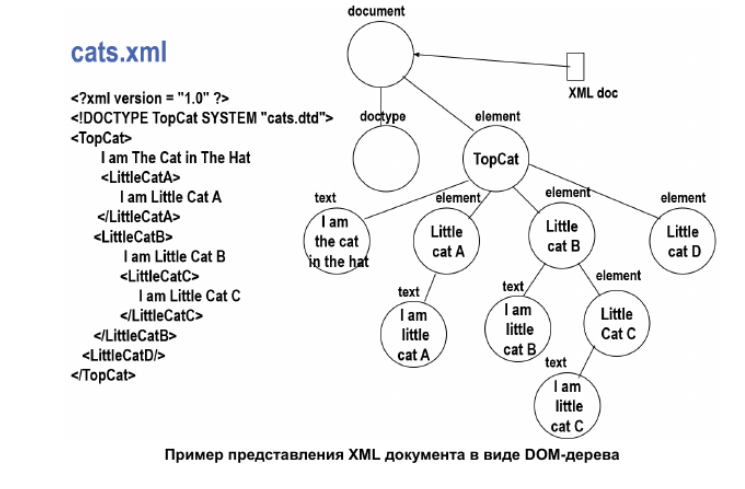
элементы могут входить в состав других элементов.

DOM (Document Object Model):

+ Естественное соответствие древовидной структуры документа и его объектной модели.

+ Возможность навигироваться по документу в любом направлении.

- Необходимо прочитать весь документ в память.



BeautifulSoup является библиотекой Python для парсинга HTML и XML документов. Часто

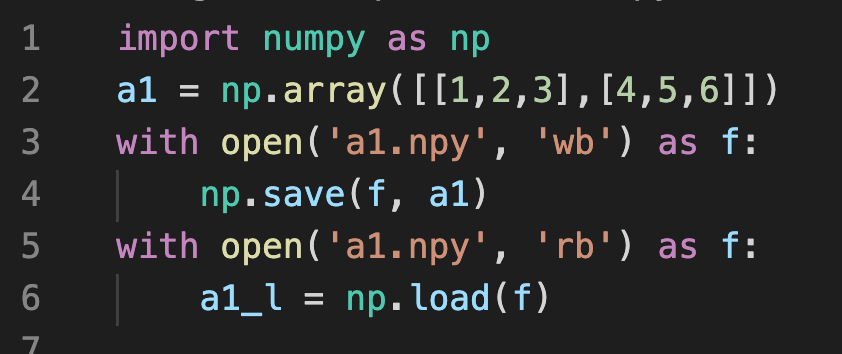
используется для скрапинга веб-страниц. BeautifulSoup позволяет трансформировать XML или HTML документ в древо объектов Python, аналогичное DOM-дереву. Элементами этого дерева могут быть теги, навигация или комментарии.

1. Форматы файлов NPY и HDF общая характеристика, пример взаимодействие с данными этих форматов в Python

**Формат файла NPY** - это файл данных, связанный с языком программирования Python. В основном используются для хранения объемных данных

* **Основное использование:** NumPy - один из важных пакетов, необходимых для выполнения вычислений и анализа данных. Файлы NPY используются в NumPy для хранения информации о файлах массивов, необходимой для проведения требуемого анализа данных. Файлы NPY хранят данные массива в формате двоичного файла. Эти файлы NPY состоят из всей информации, необходимой для регенерации массива в его нынешнем виде, вместе с данными dtype и shape, на другом компьютере с различной архитектурой.

**Пример:**



**HDF5** позволяет эффективно хранить большие объемы данных. Формат HDF5 поддерживает файлы любого размера, и каждый файл имеет внутреннюю структуру, которая позволяет вам искать определенный набор данных. Это можно представить как отдельный файл со своей собственной иерархической структурой. По умолчанию данные хранятся в двоичном формате, и библиотека совместима с разными типами данных. Одним из наиболее важных вариантов формата HDF5 является то, что он позволяет прикреплять метаданные к каждому элементу структуры, что делает его идеальным для создания автономных файлов.

В Python интерфейс с форматом HDF5 можно построить с помощью пакета h5py. Одной из наиболее интересных особенностей этого пакета является то, что данные считываются из файла только тогда, когда это необходимо.

****

1. Взаимодействие с Excel из Python с помощью XLWings: принципы работы и примеры использования

Для чего используется Excel?

Интерфейс для ввода и редактирования структурированных данных

"База данных" - хранилище структурированной информации

Движок для проведения расчетов и моделирования

Презентация результатов в виде: табличных отчетов графических отчетов

Работа c xlwings

Инструменты для взаимодействия с Excel

Приложение Microsoft Excel может запускаться и управляться внешней программой через Win32 COM API. Пакет pywin32 предоставляет интерфейс между Win32 COM и Python.

- С помощью скрипта Python с правильными командами pywin32 вы можете полностью управлять приложением Excel (открывать файлы Excel, запрашивать данные из ячеек, записывать данные в ячейки, сохранять файлы Excel и т. д.).

- Команды pywin32, которые вы можете использовать, отражают команды Excel VBA, хотя и с использованием синтаксиса Python.

xlwings - xlwings - это библиотека для автоматизации Excel с помощью Python вместо VBA:

Удобная оболочка для pywin32. Она представляет несколько кратких, но мощных методов. Примером могут служить методы прямого преобразования диапазона ячеек excel в массив numpy или pandas DataFrame и наоборот.

Работает в Windows и macOS.

Вы можете вызывать Python из Excel и наоборот и писать пользовательские функции в Python (только для Windows)

xlwings - это библиотека с открытым исходным кодом

Возможности xlwings

Взаимодействие / автоматизация

Макросы и инструменты Excel

Определяемые пользователем функции

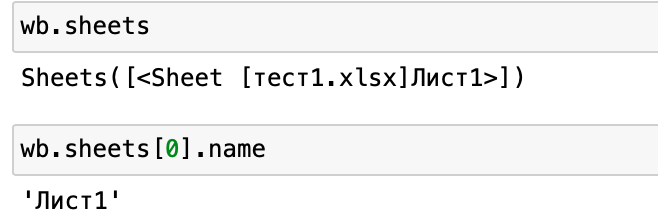
REST API

**Работа с базовыми конструкциями:**

* Книги:
  + создаем новую книгу

import xlwings as xw

wb = xw.Book()

* подключаемся к существующему файлу  
  import xlwings as xw  
  wb = xw.Book('тест1.xlsx')
* Листы  
  

Обращение к листу:

* + По индексу  
    wb.sheets[1]  
    xw.Range((2,3), (3,5)) *(индекс строки; индекс столбца)*
  + По имени  
    wb.sheets[‘Лист1’]

1. Основы работы с регулярными выражениями: базовый синтаксис, примеры использования модуля re в Python

Регулярные выражения (regular expressions, regexp) - формальный язык, используемый в программах, работающих с текстом, для поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании шаблонов, содержащих метасимволы (символы-джокеры, англ. wildcard characters).

Для поиска используется строка-образец (pattern, по-русски её часто называют «шаблоном», «маской»), состоящая из символов и метасимволов и задающая правило поиска. Например, регулярное выражение (шаблон) Иван\* (\* - здесь wildcard character) соответствует строкам любым строкам, начинающимся с Иван, например: Иван, Ивану, Иванов.

Современные языки высокого уровня поддерживают работу с регулярными выражениями либо на уровне языка либо на уровне библиотек (обычно базовых, поставляющихся с языком).

Реrулярные выражения предназначены для выполнения сложного поиска или замены в строке. В языке Python использовать регулярные выражения позволяет встроенный модуль re.

Создать откомпилированный шаблон регулярного выражения позволяет функция compile(). Функция имеет следующий формат:

<Шаблон> = rе.соmрilе(<Регулярное выражение>[, <Модификатор>])

# Пример регулярного выражения

p = re.compile(r"[а-яё]{0,6}", re.I) # Шаблон, соответствующий строке, начинающейся на 0-6 русских букв

p.match('тиквввввtik').group(0) # Возвращаем результат поиска подстроки по указанному шаблону в строке "тикtik"

Специальные символы регулярных выражений

Внутри регулярного выражения символы . ^ $ \* + ? { } [ ] \ | ( ) - имеют специальное значение.

Если эти символы требуется выводить как есть, то их следует экранировать с помощью слэша \.

Некоторые сnециальные символы теряют свое особое значение, если их разместить внутри квадратных скобок []. В этом случае экранировать их не нужно.

В квадратных скобках [] можно указать символы, которые могут встречаться на этом месте в строке. Можно перечислять символы подряд или указать их диапазон через тире. Например:

* [09] - соответствует цифре 0 или 9
* [0-9] - соответствует одной цифре от 0 до 9
* [абв] - соответствует букве "а", "б" или "в"
* [а-г] - соответствует букве "а", "б", "в" или "г"
* [а-я] - соответствует любой букве от "а" до "я", кроме буквы "ё" (т.к. "ё" находится вне непрерывного дипаозона символов русского алфавита)
* [а-яё] - соответствует любой букве от "а" до "я"
* [АБВ] - соответствует букве "А", "Б" или "В"
* [А-ЯЁ] - соответствует любой букве от "А" до "Я"
* [а-яА-ЯёЁ] - соответствует любой русской букве в любом регистре
* [0-9а-яА-ЯёЁа-zА-Z] - любая цифра и любая буква независимо от регистра и языка

1. Сегментация и токенезация текста на естественном языке, стеммминг и лемматизация, примеры на Python

Токенизация (иногда – сегментация) по предложениям – это процесс разделения письменного языка на предложения-компоненты. Идея выглядит довольно простой. В английском и некоторых других языках мы можем вычленять предложение каждый раз, когда находим определенный знак пунктуации – точку.

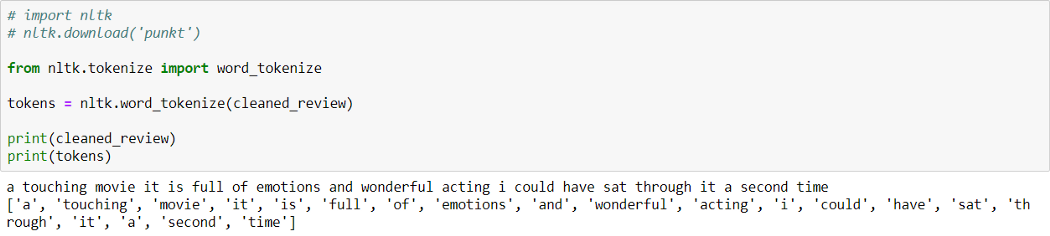
Обычно тексты содержат разные грамматические формы одного и того же слова, а также могут встречаться однокоренные слова. Лемматизация и стемминг преследуют цель привести все встречающиеся словоформы к одной, нормальной словарной форме.

Приведение разных словоформ к одной: собака, собачка, собаки, собачонка = собака

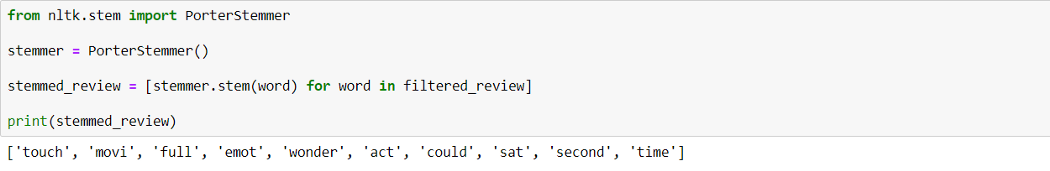
Лемматизация и стемминг – это частные случаи нормализации и они отличаются.  
  
Стемминг – это грубый эвристический процесс, который отрезает «лишнее» от корня слов, часто это приводит к потере словообразовательных суффиксов.  
  
Лемматизация – это более тонкий процесс, который использует словарь и морфологический анализ, чтобы в итоге привести слово к его канонической форме – лемме.

Отличие в том, что стеммер (конкретная реализация алгоритма стемминга – прим.переводчика) действует без знания контекста и, соответственно, не понимает разницу между словами, которые имеют разный смысл в зависимости от части речи. Однако у стеммеров есть и свои преимущества: их проще внедрить и они работают быстрее. Плюс, более низкая «аккуратность» может не иметь значения в некоторых случаях.

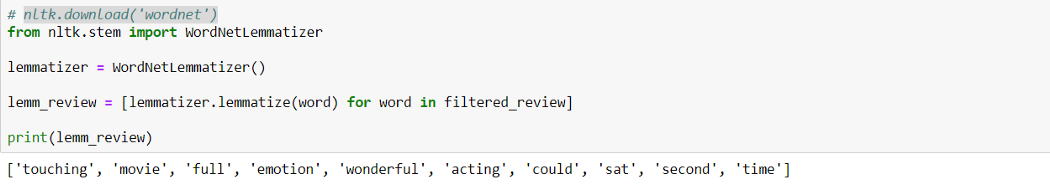
Пример токенизации



Пример стеммизации



Пример лемматизации



1. Расстояние Левеншнтейна: определение, алгоритм эффективного поиска оптимального редакционного предписания, пример поиска на Python

Расстояние Левенштейна, или редакционное расстояние, - метрика cходства между двумя строковыми последовательностями. Чем больше расстояние, тем более различны строки. Для двух одинаковых последовательностей расстояние равно нулю. По сути, это минимальное число односимвольных преобразований (удаления, вставки или замены), необходимых, чтобы превратить одну последовательность в другую.

Например, LEV(’БИБА’, ‘БОБА’) = 1, так как потребуется провести одну замену ‘И’ на ‘О’:

from Levenshtein import distance as lev

#calculate Levenshtein distance

lev('party', 'park')

Результат: 2

1. Векторное представление текста на естественном языке: общий алгоритм подходов TF; TF-IDF

Все слова (в более общем случае - термы: слова и другие значимые элементы текста) которые встречаются в документах обрабатываемой коллекции, можно упорядочить. Если теперь для некоторого документа выписать по порядку веса всех термов, включая те, которых нет в этом документе, получится вектор, который и будет представлением данного документа в векторном пространстве.

* Размерность этого вектора, как и размерность пространства, равна количеству различных термов во всей коллекции, и является одинаковой для всех документов.

Методы взвешивания термов

Для полного определения векторной модели необходимо указать, каким именно образом будет отыскиваться вес терма в документе. Существует несколько стандартных способов задания функции взвешивания:

* булевский вес - равен 1, если терм встречается в документе и 0 в противном случае;
* tf (term frequency, частота терма) - вес определяется как функция от количества вхождений терма в документе;
* tf-idf (term frequency - inverse document frequency, частота терма - обратная частота документа) — вес определяется как произведение функции от количества вхождений терма в документ и функции от величины, обратной количеству документов коллекции, в которых встречается этот терм.

1. Модуль multiprocessing – назначение и основные возможности, API multiprocessing.Pool

Модуль многопроцессорной обработки данных предлагает как локальную, так и удаленную параллельную обработку данных, эффективно обходя GIL (глобальную блокировку интерпретатора) и используя ядра процессора вместо потоков. Благодаря этому, этот модуль позволяет программисту полностью использовать несколько процессоров на данной машине. Он работает как под Unix, так и под Windows.

В зависимости от платформы модуль multiprocessing поддерживает три способа запуска процесса: spawn, fork, forkserver.

\_Класс `Process`\_\_

Самый простой подход заключается в использовании класса Process из модуля multiprocessing. Рассмотрим пример с использованием простой функции очереди для параллельной генерации четырех случайных строк.

\_\_Как получить результат в конкретном порядке?\_\_

Порядок полученных результатов не обязательно будет совпадать с порядком процессов (в списке `processes`). Поскольку мы в конечном итоге используем метод `.get()` для последовательного получения результатов из `Queue`, порядок, в котором завершаются процессы, определяет порядок наших результатов.

Например, если второй процесс завершится до первого процесса, порядок строк в списке `results` может быть иным:

`['PQpqM', 'yzQfA', 'SHZYV', 'PSNkD']` вместо `['yzQfA', 'PQpqM', 'SHZYV', 'PSNkD']`

Если наше приложение требует, чтобы мы извлекали результаты в определенном порядке, можно использовать атрибут процесса `.\_identity`. В этом случае мы также могли бы просто использовать значения из нашего объекта `range` в качестве аргумента для хранения номера процесса.

Класс `Pool`

Более простой способ поддерживать упорядоченный список результатов - использовать функции `Pool.apply` и `Pool.map`, которые мы обсудим в следующем разделе.

Другой, более удобный подход для простых задач параллельной обработки - это класс `Pool`. Особенно интересны четыре метода:

\* `Pool.apply`

\* `Pool.map`

\* `Pool.apply\_async`

\* `Pool.map\_async`

Методы `Pool.apply` и `Pool.map` эквивалентны встроенным методам `apply` и `map`.

Прежде чем мы перейдем к асинхронным вариантам методов `Pool`, давайте рассмотрим простой пример с использованием `Pool.apply` и `Pool.map`. Здесь мы установим количество процессов на 4, что означает, что класс Pool разрешит запускать только 4 процесса одновременно.

1. Различия между потоками и процессами, различие между различными планировщиками в Dask

Изображение выглядит как стол

Автоматически созданное описание

\_Знакомство с планировщиками\_\_

После того как мы создадим граф Dask мы используем планировщик для его запуска. В настоящее время Dask реализует несколько различных планировщиков:

\* `dask.threaded.get`: планировщик, построенный на основе thread pool

\* `dask.multiprocessing.get`: планировщик, построенный на основе process pool

\* `dask.get`: синхронный планировщик, удобный для выполнения отладки

\* `distributed.Client.get`: распределенный планировщик для выполнения графов на нескольких машинах. Оформлен как внешний модуль `distributed`.

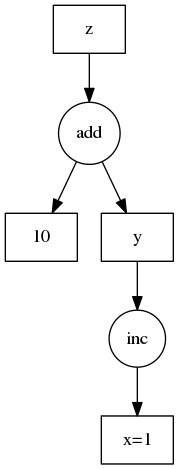
1. Граф зависимостей задач – суть структуры данных, ее построение и использование в Dask

Граф зависимостей задач

* Распространенным подходом к параллельным вычислениям является планирование задач.
* При этом подходе программа разбивается на большое количество задач (tasks) среднего размера - блоков последовательных вычислений, обычно представляющих собой вызов функции для некоторого набора данных.
* Эти задачи представляются в виде
  + вершин ориентированного графа зависимостей задач (task graph),
  + с дугами отражающими зависимость одной задачи от данных, рассчитанных другой задачей.

Этот подход позволяет программисту явно определить участки кода, подлежащие распараллеливанию.

Внутренне Dask кодирует алгоритмы в простом формате, включающем словари, кортежи и функции. Этот формат графа может использоваться отдельно от коллекций Dask.



**Пример**



**И теперь можно закодировать это как словарь таким образом:**



Библиотека Dask в настоящее время содержит несколько планировщиков для выполнения этих графиков. Каждый планировщик работает по-разному, гарантируя разную производительность в разного характера задачах и работая в разных предметных областях.

В Dask существует два основных семейства планировщиков задач. (подробнее см. вопрос №31)

1. Локальные планировщики - для локального пк
2. Распределенные планировщики - для кластерных вычислений

Локальные планировщики по принципу разбиения задач на:

* процессы - processes
* потоки - local threads
* одиночный поток (без параллелизации) - single thread
* Распределение Dask - Dask Distributed (local**)**

Кластерный планировщик:

* Распределение Dask - Dask Distributed (cluster**)**

Кроме того, существует возможность писать свои планировщики под свои задачи.

При создании пользователем задачи, Dask анализирует её для того, чтобы вычислить её оптимальным способом.

**Определения**

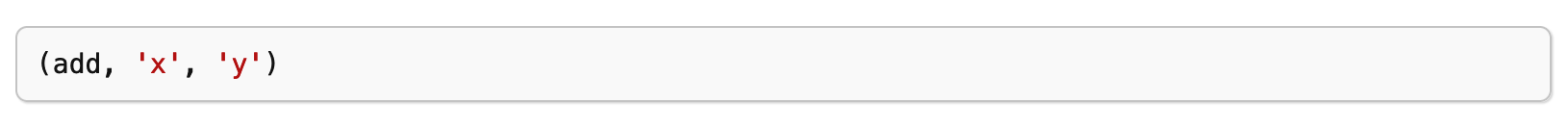
Граф задач - это словарь, отображающий ключи к вычислениям:



Ключ это хешируемое значение, которое не является таском



Task - это кортеж с вызываемым первым элементом. Задачи представляют собой атомарные единицы работы, предназначенные для выполнения одиночным исполнителем задач. Пример:



Мы представляем задачу(task) как кортеж, первый элемент которого - функция(напр. add). А последующие элементы этого кортежа это аргументы, передаваемые этой функции. Аргумент также может являться любым валидным вычислением.

**Правильные примеры использования:**



**Точка входа - функция Get**

Функция get служит точкой входа для всех планировщиков. Эта функция получает значение по переданному ей ключу. Этот ключ может ссылаться на хранимую информацию, как например в случае с “x” или так же может ссылаться на задачу, как в случае с “z”. Ну и в последнем случае, с “w”, get должен выполнить все необходимые вычисления для извлечения вычисленного значения.



1. Dask.Array – структура данных, специфика реализации и применения, процедура создания

\_Dask Array\_\_

\* Dask Array реализует \_\_подмножество интерфейса NumPy ndarray\_\_, используя \_\_алгоритмы в блочной форме\_\_

\* Большой массив разбивается на относительно небольшие блоки которые обрабатываются независимо

\* Эта техника позволяет:

\* оперировать массивами, большими чем оперативная память

\* использовать все доступные ядра.

\* \_\_Координация задач\_\_, возникающих при исполнении блочной формы алгоритмов, осуществляется при помощи реализованного в Dask \_\_графа зависимостей задач\_\_.

Реализация Dask Array:

\* Dask Array представляет собой \_\_сетку из массивов NumPy\_\_, обработку которых он организует \_\_порождая\_\_ для каждой операции со всем массивом \_\_множество операций с массивами NumPy\_\_.

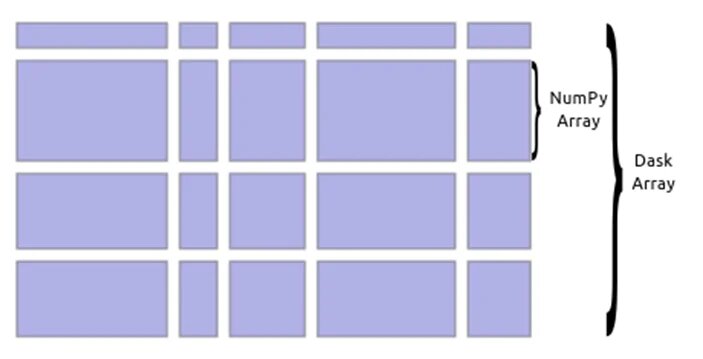
\* Массивы NumPy могут:

\* находится в оперативной в памяти

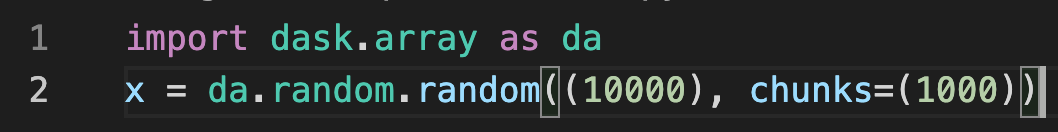
\* находится в распределенной оперативной в памяти кластера (т.е. хранится на узлах кластера)

\* находится на диске (по крайней мере часть времени вычислений).

Принцип организации данных в Dask Array

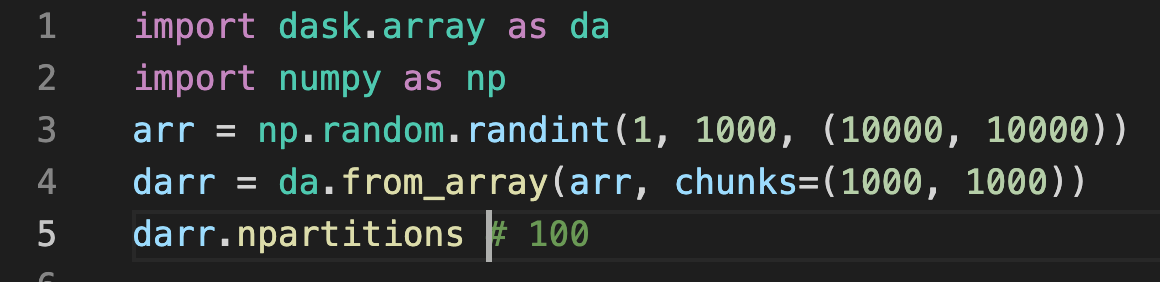


Пример создания случайного массива

****

Dask Array может читать из любого массива, например, структуры, поскольку он поддерживает numpy, как срезы.shape собственность с помощью dask.array.from\_array метод. Он также может читать из .npy а также .zarr файлы.

**Пример**



В настоящее время, Dask делает ленивую оценку каждого метода. Итак, чтобы на самом деле вычислить значение функции, вы должны использовать .compute() метод. Он будет вычислять результат параллельно в блоках, распараллеливая каждую независимую задачу в то время.

1. Dask.Array – поддерживаемые операции и отличия от NumPy ndarray

\_Основные возможности Dask Array\_\_

\_\_Dask Array поддерживает большинство интерфейсов NumPy, в частности\_\_:

\* Арифметические операции и скалярные функции: `+`, `\*`, `exp`, `log`, ...

\* Агрегирующие функции (в т.ч. вдоль осей): `sum()`, `mean()`, `std()`, `sum(axis=0)`, ...

\* Умножение матриц, свёртка тензоров: `tensordot`

\* Получение срезов: `x[:100, 500:100:-2]`

\* Прихотливое индексирование вдоль одной оси: `x[:, [10, 1, 5]]`

\* Работу с протоколами массивов `\_\_array\_\_` и `\_\_array\_ufunc\_\_`

\* Некоторые операции линейной алгебры: `svd`, `qr`, `solve`, `solve\_triangular`, `lstsq`

Но, Dask Array не поддерживает следующих возможностей NumPy:

\* Не реализована большая часть пакета np.linalg

\* Не поддерживаются операции с массивами неизвестного размера

\* Операции наподобие sort , которые по своей сути сложно выполнять параллельно не поддерживаются. Зачастую, вместо таких операций предлагается альтернативная функция, дружественная к параллельному вычислению

\* Не поддерживаются операции типа tolist, т.к. это очень неэффективно для больших наборов данных, тем более что, обход этих данных в циклах очень неэффективен.

1. Dask.Bag - структура данных, специфика реализации и применения, процедура создания DaskBag

Структура данных Bag

Мультимножество (bag, multiset) в математике - обобщение понятия множества, допускающее

включение одного и того же элемента по нескольку раз. Число элементов в мультимножестве, с учётом повторяющихся элементов, называется его размером или мощностью.

list : упорядоченная коллекция, допускающая повторы элементв.

Пример: [1, 2, 3, 2]

set : неупорядоченная коллекция, не допускающая повторы элементов.

Пример: {3, 1, 2} (эквиваленто {1, 2, 3} , данный вариант {1, 2, 2, 3}

автоматически перобразуется к {1, 2, 3} )

bag : неупорядоченная коллекция, допускающая повторы элементов.

Пример: bag(1, 2, 2, 3) (эквивалентно bag(2, 1, 2, 3) )

Таким образом, bag можно рассматривать как список, не гарантирующий порядка элементов.

Dask.Bag реализует такие операции, как map , filter , fold (аналог reduce ) и groupby над

коллекциями объектов Python.

Dask.Bag хорошо подходит для распараллеливания простой обработки неструктурированных

или полу-структурированных данных, таких как:

текстовые данные

файлы логирования

записи в формате JSON

специальных oбектов Python и т.д.

Специфика реализации

По умолчанию, Dask.Bag использует для исполнения планировщик dask.multiprocessing .

Это позволяет обойти проблему GIL и полноценно использовать несколько процессорных ядер для объектов реализованных на чситом Python.

Минусом этого подходя является наличие больших накладных расходов при обмене данных

между исполнителями, что важно для производительности вычислений, требующих

интенсивного обмена данными. Это редко бывает проблемой, так как типичный поток задач для Dask.Bag подразумевает:

- или черезвычайно параллельные вычисления

- или обмен небольшим объемом данных в процессе свертки (англ. folding, также известна

как reduce, accumulate).

1. Организация вычислений с помощью Map / Filter / Reduce : общий принцип и специфика параллельной реализации обработки данных в Dask.Bag

Чрезвычайная параллельность (embarrassingly parallel) - тип задач в системах параллельных

вычислений, для которых не требуется прилагать больших усилий при разделении на несколькоотдельных параллельных задач (распараллеливании).

- Чаще всего не существует зависимости (или связи) между параллельными задачами, то

есть их результаты не влияют друг на друга.

- Чрезвычайно параллельные задачи практически не требуют согласования между

результатами выполнения отдельных этапов, что отличает их от задач распределённых

вычислений, которые требуют связи промежуточных результатов.

- Такие задачи легки для исполнения массово паралельных системах (кластерах с очень

большим количеством вычислительных узлов).

Функция map() является типом высшего порядка. Как упоминалось ранее, эта функция принимает другую функцию в качестве параметра вместе с итерируемой последовательностью и возвращает выходные данные после применения функции на каждый итерируемый элемент из последовательности. Синтаксис выглядит следующим образом:

**map(function, iterables)**

Функция filter() используется для создания списка, состоящего из значений, для которых функция возвращает true. Синтаксис этого следующий:

**filter(function, iterables)**

Функция reduce(), как можно понять из названия, применяет переданную функцию к итерируемому объекту и возвращает одно значение.

**reduce(function, iterables)**

1. Изображение выглядит как стол

   Автоматически созданное описаниеAPI Dask.Bag – функции мэппинга, фильтрации и преобразования

Примеры использования функции `map`

b1 = db.from\_sequence(range(6), npartitions=2)

b1.map(lambda x: x + 1).compute()

\_\_Преобразование строк\_\_

Одиним из целвых объектов для обработки с помощью Dask Bag являются строки, поэтому для работы с ними поддержвается специализированный набор функций.

Преобразование строк для Dask Bag можно выполнять с использованием функций из пространства имен `str`.

Обработку строк, находящихся в объектах Bag, можно осуществлять используя пространство имен `str`, которое напрямую привязано к объектам Bag. Таким образом мэппинг функций из str можно производить напрямую, не исопльзуя функцию `map`.

Изображение выглядит как стол

Автоматически созданное описание

Примеры использования функции `filter`

b = db.from\_sequence(range(5))

list(b.filter(lambda x: x % 2 == 0))

1. API Dask.Bag – функции группировки и свертки

Изображение выглядит как стол

Автоматически созданное описание

Изображение выглядит как стол

Автоматически созданное описание