МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ**

**Н.Г. ЧЕРНЫШЕВСКОГО»**

УТВЕРЖДАЮ

Зав. кафедрой,

к. ф.-м. н.

\_\_\_\_\_\_\_\_\_\_ М. В. Огнева

**ОТЧЕТ О ПРАКТИКЕ**

Студентки 2 курса 241 группы факультета КНиИТ   
Крулевой Марии Михайловны

вид практики: учебная технологическая (проектно-технологическая)

кафедра: информатики и программирования

курс: 1

семестр: 2

продолжительность: 2 нед., с 23.06.2025 г. по 06.07.2025 г.

Руководитель практики от университета:

ст. пр.                                   \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_                          Е. Е. Лапшева

Тема практики: «Библиотеки предобработки данных»

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 4](#_Toc210674808)

[1 Основные сведения о библиотеках Python для обработки данных 6](#_Toc210674809)

[1.1 Обработка исключений 6](#_Toc210674810)

[1.2 Библиотека random 7](#_Toc210674811)

[1.3 Библиотека Math 7](#_Toc210674812)

[1.4 Библиотека NumPy 7](#_Toc210674813)

[1.5 библиотека Pandas 8](#_Toc210674814)

[1.6 Библиотека Matplotlib 8](#_Toc210674815)

[1.7 Библиотека NetworkX 9](#_Toc210674816)

[2 Решение задач с использованием изученных библиотек 10](#_Toc210674817)

[2.1 Обработка исключений. 10](#_Toc210674818)

[2.2 Встроенные модули 15](#_Toc210674819)

[2.3 Модуль NumPy 24](#_Toc210674820)

[2.4 Модуль Pandas 31](#_Toc210674821)

[2.5 Визуализация 37](#_Toc210674822)

[3. Решение математических задач 42](#_Toc210674823)

[3.1 Линейные подпространства 42](#_Toc210674824)

[3.2 Ядро, образ 44](#_Toc210674825)

[3.3 Норма 47](#_Toc210674826)

[3.4 Угол, расстояние 48](#_Toc210674827)

[3.5 Ортогональная проекция 50](#_Toc210674828)

[ЗАКЛЮЧЕНИЕ 54](#_Toc210674829)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 55](#_Toc210674830)

[ПРИЛОЖЕНИЕ А Обработка исключений 56](#_Toc210674831)

[ПРИЛОЖЕНИЕ Б Встроенные модули 59](#_Toc210674832)

[ПРИЛОЖЕНИЕ В Библиотека NumPy 65](#_Toc210674833)

[ПРИЛОЖЕНИЕ Г Модуль Pandas 69](#_Toc210674834)

[ПРИЛОЖЕНИЕ Д Визуализация 71](#_Toc210674835)

# **ВВЕДЕНИЕ**

В мире анализа данных Python стал одним из важнейших языков программирования. Согласно индексу TIOBE, который измеряет популярность языков программирования, Python является самым популярным языком программирования в мире. Его популярность среди аналитиков данных обусловлена универсальностью, обширными библиотеками и интуитивно понятным синтаксисом. [1]

Python также может использоваться для структурирования больших наборов данных, упрощения математических операций и представления содержательной информации в понятном формате с помощью библиотек визуализации.

Не менее значима роль и линейной алгебры как основы анализа данных. Линейная алгебра упрощает управление и анализ больших наборов данных. Понятие векторов, линейных операторов, а также операции проекции, нормы и расстояния лежат в основе многих методов обработки и предобработки данных.

**Актуальность** инструментов, изучаемых в ходе летней практики, заключается в том, что они формируют основу современного стека технологий анализа данных. Эти библиотеки повышают эффективность и точность работы. Умение применять их на практике позволяет автоматизировать рутинные задачи, значительно ускорить обработку данных и визуализировать результаты. Это является основой для принятия обоснованных решений в самых разных предметных областях.

Основной целью практики является ознакомление с возможностями предобработки данных с использованием языка Python и его стандартных модулей, и библиотек NumPy, Pandas, библиотек визуализации Matplotlib и NetworkX. Дополнительно уделяется внимание обработке исключений, что позволит в дальнейшем корректно работать с непредвиденными ситуациями при выполнении программного кода.

Также еще одним направлением является изучение разделов линейной алгебры: линейные подпространства, ядро и образ линейных операторов, норма, угол, расстояние и ортогональная проекция. Эти знания позволят производить различные операции с данными.

Для достижения поставленной цели необходимо решить следующие задачи:

* изучить возможности и инструменты программной среды Google Colab;
* изучить стандартные модули Math и Random языка Python;
* изучить объекты и методы библиотек NumPy, Pandas, библиотек визуализации;
* изучить основные разделы линейной алгебры
* применить полученные знания для решения поставленных задач, а также для самостоятельной обработки и анализа набора данных из реальной жизни;
* применить полученные знания в линейной алгебре для решения поставленных задач.

# **1 Основные сведения о библиотеках Python для обработки данных**

Python в последние годы стал одним из самых популярных языков программирования благодаря своей универсальности для широкого спектра задач: от обработки и анализа данных до машинного обучения и веб-разработки. Помимо гибкости и простого синтаксиса, Python располагает богатой стандартной библиотекой и широкой экосистемой внешних пакетов, которые упрощают сложные задачи и способствуют быстрой разработке.

Сервис Google Colaboratory (Colab) – это облачное исполнение среды Jupyter Notebook, не требующее настройки и поддерживающее выполнение кода в браузере с доступом к мощным вычислительным ресурсам бесплатно. Colab особенно хорошо подходит для машинного обучения, анализа и визуализации данных, благодаря своей интеграции с Python-библиотеками и удобству совместной работы.

## **1.1 Обработка исключений**

Даже при корректном синтаксисе при выполнении программы может возникнуть ошибка. Ошибки, обнаруженные во время выполнения, называются исключениями. Большинство из них приводят к остановке программы с сообщением об ошибке, где указывется тип исключения (например, TypeError, ZeroDivisionError, NameError).

Для обработки подобных ситуаций используется конструкция try/except. Если в блоке try не возникает ошибок, то выполнение продолжается без перехода в except. При возникновении исключения блок try прерывается, и управление передается обработчику except, если его тип совпадает с возникшим исключением. В противном случае это необработанное исключение и выполнение останавливается с сообщением об ошибке.

Один оператор try может содержать более одного предложения except, однако выполняется только одно из них – соответствующее типу возникшего исключения. Это позволяет избирательно обрабатывать различные типы ошибок и продолжать выполнение программы. [2]

## **1.2 Библиотека random**

Модуль random реализует генераторы псевдослучайных чисел для различных распределений и операций, связанных со случайностью. Генераторы не предназначены для криптографически стойких задач. Основные функции:

* random.randrange(start, stop, step) – выбор случайного элемента из диапазона;
* random.randint(a,b) – случайное целое число N, удовлетворяющее условию a <= N <= b. Полезен для получения псевдослучайных чисел в указанном диапазоне;
* random.choice(seq) ‑‑ случайный элемент из непустой последовательности seq. Если seq пуста, вызывает исключение [IndexError](https://docs.python.org/3/library/exceptions.html#IndexError);
* random.choices(population, weights, k) ‑‑ список из k элементов, выбранных из последовательности с возможностью повторения. Если последовательность пуста, вызывает исключение IndexError;
* random.shuffle(x) ‑ случайная перетасовка элементов последовательности. [3]

## **1.3 Библиотека Math**

Встроенный модуль math в Python предоставляет набор функций для математических, тригонометрических и логарифмических вычислений. Среди наиболее часто используемых:

* pow(num, power): возведение числа num в степень power;
* sqrt(num): квадратный корень числа num;
* factorial(num): факториал числа;
* log(n, base): логарифм числа n по основанию base;
* cos(rad): косинус угла в радианах. [4]

## **1.4 Библиотека NumPy**

NumPy — это фундаментальный пакет для научных вычислений на Python. Это библиотека Python, предоставляющая многомерный массив, различные производные объекты, а также набор функций для быстрых операций с массивами, включая математические и логические операции, базовые операции линейной алгебры, базовые статистические операции, моделирование случайных чисел и многое другое.

Популярные методы библиотеки:

* numpy.array() — создает массив NumPy из списка или кортежа;
* numpy.zeros() - массив, заполненный нулевыми элементами;
* numpy.sort(arr) – сортировка числа по возрастанию;
* reshape() ‑‑ придает массиву новую форму, не изменяя данные;
* numpy.flip() ‑‑ позволяет перевернуть содержимое массива по оси.

Ключевое преимущество NumPy — высокая производительность при работе с большими данными. [5]

## **1.5 библиотека Pandas**

Библиотека Pandas предназначена для анализа и обработки данных в Python в табличной форме. Эта библиотека построена на базе NumPy и предоставляет удобный интерфейс для работы с объектами Series и DataFrame.

Основные методы библиотеки:

* pandas.Series() – создание Series ‑‑ одномерный массив, способный хранить значения любого типа;
* pandas.DataFrame() — создание DataFrame из словарей, списков и других структур данных для удобного представления данных в табличной форме;
* pandas.read\_csv() — загрузка данные из CSV-файла в DataFrame;
* groupby() — группировка данных по столбцам;
* pandas.merge() — объединение объектов DataFrame или именованные объекты Series по ключу. [6]

## **1.6 Библиотека Matplotlib**

Matplotlib используется для визуализации данных в Python и позволяет широкий спектр графиков: линейные графики, столбчатые, круговые диаграммы, гистограммы и диаграммы рассеяния.

Основные методы:

* plt.plot() — строит линейные графики по значениям осей X и Y;
* plt.scatter() — строит диаграммы рассеяния;
* plt.pie() — рисует круговые диаграммы;
* plt.bar() — рисует столбчатые диаграммы;
* plt.hist() — создает гистограммы. [7]

## **1.7 Библиотека NetworkX**

Библиотека NetworkX предназначена для работы со сложными сетями и графами. Она используется в обучении теории графов, исследованиях и прикладных задачах. С помощью NetworkX можно загружать и хранить сети в стандартных и нестандартных форматах данных, генерировать множество типов случайных и классических сетей, анализировать структуру сетей, строить сетевые модели, проектировать новые сетевые алгоритмы, рисовать сети и многое другое.

Важные методы библиотеки:

* nx.Graph() – создание пустого графа без узлов и ребер;
* G.add\_node() – добавление узлов к графу;
* [compose](https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.operators.binary.compose.html#networkx.algorithms.operators.binary.compose)(G, H) ‑‑ объединение узлов и ребер графов G и H в один граф;
* dfs\_tree(G) – построение ориентированного дерева поиска в глубину;
* draw(G) – визуализация графа G с помощью Matplotlib. [8]

# **2 Решение задач с использованием изученных библиотек**

Во время летней практики были использованы различные модули Python для решения прикладных задач. В этой части отчета уделим внимание примерам задач и способам их решения с применением таких библиотек, как random, math, NumPy, Pandas, matplotlib, plotly, а также обработке исключений с помощью конструкции try-except.

## **2.1 Обработка исключений.**

В данном разделе будут рассмотрены задачи, в которых требуется обработка исключений с использованием конструкции try-except. Подробное решение задач на обработку исключений представлено в Приложении А.

Задача 1. Корень зла 2.

Необходимо реализовать функцию find\_roots(), принимающую три параметра: коэффициенты квадратного уравнения и возвращает его корни в виде кортежа из двух значений. Также требуется создать два собственных исключения:

* NoSolutionsError – вызывается, если уравнение не имеет решений;
* InfiniteSolutionsError – вызывается, если уравнение имеет бесконечное множество решений.

Если переданные коэффициенты не являются рациональными числами, должно быть вызвано исключение TypeError.

Решение: создаются исключения NoSolutionsError и InfiniteSolutionsError.

Функция find\_roots проверяет корректность типов аргументов и предотвращает дальнейшие ошибки при вычислениях.

При решении задачи рассматривались частные случаи:

Если все коэффициенты равны нулю, то вызывается исключение InfiniteSolutionsError, если корни квадратного уравнения отсутствуют – NoSolutionsError.

При корни равны нулю. Когда уравнение является линейным, и корень вычисляет как . При и корни вычисляются как  
, если , вызывается NoSolutionsError.

def find\_roots(a, b, c):

d = b \*\* 2 - 4 \* a \* c

roots = ()

if not all(isinstance(n, (int, float)) for n in (a, b, c)):

raise TypeError

if a == b == c == 0:

raise InfiniteSolutionsError

elif a == b == 0:

raise NoSolutionsError

elif a == c == 0:

roots = (0, 0)

else:

if a == 0:

print(-c / b)

if b == 0:

if c > 0:

x1 = (c / a) \*\* 0.5

x2 = -((c / a) \*\* 0.5)

roots = tuple(sorted([x1, x2]))

else:

raise NoSolutionsError

Для квадратичного уравнения , вычисляется дискриминант

В случаях, когда обрабатывается исключение NoSolutionsError, при один действительный корень и вычисляется по формуле: , в случаях, когда , будет два действительных корня .

if a != 0:

if b == c == 0:

roots = ((0 / a), (0 / a))

if d < 0:

raise NoSolutionsError

if d == 0:

roots = (-b / (2 \* a), -b / (2 \* a))

if d > 0:

x1 = (-b + d \*\* 0.5) / (2 \* a)

x2 = (-b - d \*\* 0.5) / (2 \* a)

roots = tuple(sorted([x1, x2]))

return roots

Функция find\_roots обрабатывает исключения для предотвращения неправильных вычислений и ошибок при решении уравнений. Собственные исключения позволяют точно понимать ситуации, когда отсутствуют решения уравнений или наоборот, когда решений уравнения бесконечное множество.

Задача 2. Валидация пользователя

Необходимо реализовать функцию user\_validation, принимающую параметры:

* last\_name — фамилия;
* first\_name — имя;
* username — имя пользователя.

Условия:

При передаче в функцию неизвестного параметра или отсутствии обязательного – вызывается исключение KeyError

Если один из параметров не является строкой - TypeError.

Обработка данных осуществляется в порядке: фамилия, имя, имя пользователя. При успешном выполнении функция возвращает словарь с данными пользователя.

Решение: были созданы собственные ошибки:

* CyrillicError — вызывается, если значение не состоит только из кириллических букв;
* CapitalError — вызывается, если значение не начинается с заглавной буквы или найдена заглавная буква не в начале значения.
* BadCharacterError — вызывается, если значение состоит не только из латинских букв, цифр и символа нижнего подчёркивания;
* StartsWithDigitError — вызывается, если значение начинается с цифры

Для реализации были созданы вспомогательные функции:

Функция name\_validation(name) принимает на вход один позиционный аргумент – фамилия или имя пользователя.

Если параметр не является строкой, то обрабатывается исключение TypeError.

В случае, если значение не состоит только из кириллических букв, вызывается исключение CyrillicError

Если значение не начинается с заглавной буквы или содержит заглавную букву в других позициях, обрабатывается исключение CapitalError

def name\_validation(name):

cyrillic = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'

if not isinstance(name, str):

raise TypeError

if not all(s.lower() in cyrillic for s in name):

raise CyrillicError

if not name.istitle():

raise CapitalError

return name

Функция username\_validation(username) предназначена для проверки корректности имени пользователя. Она также принимает один позиционный аргумент.

Если параметр не является строкой – вызывается исключение TypeError.

При использовании недопустимых символов (отличных от латинских букв, цифр и символа нижнего подчеркивания), обрабатывается исключение BadCharacterError.

Если значение начинается с цифры, вызывается исключение StartsWithDigitError.

В случае успешной проверки функция возвращает исходное значение без изменений.

def username\_validation(username):

tmp = f'{ascii\_letters}{digits}\_'

if not isinstance(username, str):

raise TypeError

if not all(s in tmp for s in username):

raise BadCharacterError

if username[0].isdigit():

raise StartsWithDigitError

return username

Функции name\_validation и username\_validation помогают в создании и корректной работы основной функции этой задачи - user\_validation:

def user\_validation(\*\*user):

if list(user.keys()) != ['last\_name', 'first\_name', 'username']:

raise KeyError

if not all(isinstance(x, str) for x in user.values()):

raise TypeError

name\_validation(user['last\_name'])

name\_validation(user['first\_name'])

username\_validation(user['username'])

return user

Задача 3. Валидация пароля.

Следующим этапом стала реализация функции password\_validation, которая принимает один позиционный параметр — пароль, и следующие именованные параметры:

min\_length — минимальная длина пароля, по умолчанию 8;

possible\_chars — разрешенные символы, по умолчанию латинские буквы и цифры;

at\_least\_one — функция, возвращающая логическое значение, по умолчанию str.isdigit.

Виды исключений, которые необходимо обработать:

* TypeError, переданный позиционный параметр не является строкой
* MinLengthError, если пароль меньше заданной длины;
* PossibleCharError, если в пароле используется недопустимый символ;
* NeedCharError, если в пароле не найдено ни одного обязательного символа.

Примечание: проверка условий должна происходить в порядке, указанном в задании.

При успешной проверке пароль преобразуется в хеш с помощью функции sha256 из пакета hashlib и возвращается в шестнадцатеричном представлении.

Решение: реализация собственных исключений MinLengthError, PossibleCharError, NeedCharError представлена в Приложении А.

Продемонстрируем работу функции password\_validation:

def password\_validation(password, min\_length=8, possible\_chars=symbol, at\_least\_one=str.isdigit):

if not isinstance(password, str):

raise TypeError

if len(password) < min\_length:

raise MinLengthError

if any(i not in possible\_chars for i in password):

raise PossibleCharError

if not any(at\_least\_one(x) for x in password):

raise NeedCharError

return sha256(password.encode()).hexdigest()

Использование конструкции try-except помогает программисту обрабатывать ошибки или исключения, которые могут возникнуть при выполнении программы, и тем самым облегчает ее отладку. Исключения также используются и для подачи сигналов о возникших проблемах. Например, при решении этих задач, обработка исключений позволяет избежать последующих ошибок в коде, связанных с неправильным заполнением данных о пользователе или его пароля.

## **2.2 Встроенные модули**

Подробное решение задач из этого раздела представлено в Приложении Б.

**2.2.1 Модули math и itertools**

В этом подразделе будут рассмотрены задачи на модули math и itertools. Вместе эти модули позволяют эффективно решать широкий спектр математических задач, включая задачи комбинаторики и работы с системами счисления.

Задача 1: Числа в p-ичной системе счисления

Требуется вывести все -значные числа в -ичной системе счисления. Числа с лидирующими нулями выводить не нужно.

Решение: количество -значных чисел в системе счисления с основанием , где первая цифра не ноль, вычисляется по формуле:

Для генерации всех возможных кортежей длины с цифрами от в лексикографическом порядке используется функция product() из модуля itertools. После этого «отсеиваются» варианты с ведущим нулем, а кортеж преобразовывается в строковое представление числа (например (1, 0, 1) превращается в 101).

k, p = map(int, input().split())

print((p - 1) \* p \*\* (k - 1))

for i in product(range(p), repeat=k):

if i[0] != 0:

print(''.join(map(str, i)))

Задача 2: Размещения с повторениями

Размещением с повторениями (из по ) называется упорядоченный набор из чисел от до , причем одно число может встречаться в наборе несколько раз. Выведите все размещения с повторениями из по в лексикографическом порядке.

Решение: количество размещений с повторениями вычисляется по формуле: . Функция product() из модуля itertools выполняет полный перебор всех размещений с повторениями из элементов по позиций в лексикографическом порядке.

n, k = map(int, input().split())

print(n \*\* k)

for x in product(range(1, n + 1), repeat=k):

print(' '.join(map(str, x)))

Задача 3: Сочетания

Сочетанием из по называется неупорядоченный набор из различных чисел от до . Сгенерируйте все сочетания из по в лексикографическом порядке. Лексикографический порядок для сочетаний означает, что сначала элементы внутри сочетания упорядочиваются по возрастанию, а затем сами сочетания упорядочиваются лексикографически как векторы длины .

Решение: число сочетаний (биномиальный коэффициент) определяется формулой: , Для вычисления факториала самым быстрым и эффективным способом применяется функция factorial() из модуля math. А для генерации всех сочетаний из по используется функция combinations() из модуля itertools.

n, m = map(int, input().split())

c = factorial(n) // (factorial(m) \* factorial(n - m))

print(c)

for i in combinations(range(1, n + 1), m):

print(' '.join(map(str, i)))

**2.2.2 Модуль random**

В этом подразделе рассматривается задача генерации пользовательских данных с использованием стандартного модуля random.

id — случайное пятизначное число,  
логин — случайная последовательность из 6 маленьких английских букв,  
пароль — случайная последовательность из 10 неповторяющихся больших и маленьких английских букв и цифр

Задача: id — случайное пятизначное число, логин — случайная последовательность из 6 маленьких английских букв, пароль — случайная последовательность из 10 неповторяющихся больших и маленьких английских букв и цифр

Создайте функцию генерации id, функцию генерации логина и функцию генерации пароля. С использованием этих трёх функций напишите функцию, генерирующую список из N троек вида (id, логин, пароль), в которых ни id, ни логины, ни пароли не повторяются (при этом повторный вызов функции не должен учитывать результаты предыдущей генерации). Также должны выполняться следующие условия: последние две цифры id совпадают, никакая буква логина не встречается в нём больше двух раз, пароль начинается не с цифры, но хотя бы одна цифра в нём обязательно есть.

Решение: была создана функция, генерирующая id длиной 5 символов, состоящий из случайный цифр. Важное условие, заложенное в функцию – последние две цифры id совпадают. Для этого используется цикл, который будет генерировать id до тех пор, пока не выполнится условие задачи.

def generation\_id():  
 while True:  
 user\_id = ''.join(random.choices(digits, k=3))  
 last\_digit = ''.join(random.choices(digits))  
 return f'{user\_id}{last\_digit}{last\_digit}'

Затем создается функция, генерирующая логин из 6 строчных английских букв. Было соблюдено условие задачи: никакая буква логина не встречается в нем больше двух раз. Для решения этой задачи также используется цикл, генерирующий логин до тех пор, пока он не будет удовлетворять условие задачи.

def generation\_username():  
 while True:  
 user\_name = ''.join(random.choices(ascii\_lowercase, k=6))  
 letters = Counter(x for x in user\_name)  
 if all(cnt <= 2 for cnt in letters.values()):  
 return user\_name

Далее была создана функция генерации случайного пароля из 10 неповторяющихся заглавных и строчных английских букв и цифр. По условию, пароль начинается не с цифры, но хотя бы одна цифра в нем обязательно есть. В этой функции также использовался цикл, позволяющий создавать пароль пока не станет таким, каким требует данная задача.

def generation\_password():  
 symbols = f'{ascii\_letters}{digits}'  
 while True:  
 pwrd = ''.join(random.choices(symbols, k=10))  
 if pwrd[0] not in digits and any(n in digits for n in digits):  
 return pwrd

В конце задачи была создана функция, генерирующая список уникальных пользователей, каждый из которых имеет уникальный id, логин и пароль:

def n\_lst(n):  
 lst = []  
 us\_id = set()  
 username = set()  
 password = set()  
  
 while len(lst) < n:  
 uid = generation\_id()  
 user = generation\_username()  
 passw = generation\_password()  
  
 if all([uid not in us\_id, user not in username, passw not in password]):  
 us\_id.add(uid)  
 username.add(user)  
 password.add(passw)  
 lst.append((uid, user, passw))  
 return lst

В результате выполнения функции n\_lst() будут созданы из N троек вида (id, логин, пароль), в которых ни id, ни логины, ни пароли не повторяются, при этом повторный вызов функции не учитывает результаты предыдущей генерации. Функция n\_lst() объединяет все раннее созданные функции для генерации n пользователей с уникальными данными.

**2.2.3 Измерение времени выполнения**

В этом подразделе будут рассмотрены экспериментальные задачи, направленные на сравнение эффективности различных алгоритмов и структур данных. Важной частью анализа является измерение времени выполнения функций при работе с входными данными различного размера.

Задача 1. Сравнить время выполнения алгоритмов на различных размерах входных данных:

1. Проверка на простоту перебором до , // и до.
2. Поиск элемента в списке и во множестве.
3. Ответ на запросы о количестве вхождений элементов в список или букв в строку с помощью вызова метода count и с использованием Counter'а.
4. Квадратичная сортировка, быстрая сортировка, встроенная сортировка на числах.
5. Квадратичная сортировка, быстрая сортировка, встроенная сортировка на строках.

Решение:

1. Проверка чисел на простоту перебором до , // и до.

Из результатов экспериментов, представленных в Таблице1 видно, что функция перебора до is\_prime\_n\_sqrt ‑ наиболее эффективный алгоритм.

Все потому, что для любого числа проверяются делители только до квадратного корня, что значительно сокращает количество итераций.

На втором месте стоит перебор до // Наименее эффективным оказался перебор до . Например, можно заметить колоссальную разницу по сравнению с самой быстрой функцией — перебор до на больших числах.

Так, время выполнения функции is\_prime\_n\_sqrt и is\_prime\_n при  
 отличается примерно в раза.

Таблица 1. Время выполнения функций проверки на простоту.

|  |  |
| --- | --- |
| Название функции | Время выполнения при |
| is\_prime\_n |  |
| is\_prime\_n\_div2 |  |
| is\_prime\_n\_sqrt |  |

1. Поиск элемента в списке и во множестве.

Согласно данным из Таблицы 2 видно, что поиск элементов во множестве выполняет в среднем быстрее в раза, чем в списке. Это подтверждает асимптотическое преимущество множеств при решении задач с поиском элемента.

Таблица 2. Время выполнения поиска элемента в списке и во множестве.

|  |  |
| --- | --- |
| Функция хранения элементов | Время выполнения на элементов |
| list |  |
| set |  |

1. Ответ на запросы о количестве вхождений элементов в список или букв в строку с помощью вызова метода count и с использованием Counter'а.

В Таблице 3 можно заметить, что Counter из модуля collections превосходит стандартный метод count() примерно в раз. Это обусловлено тем, что Counter один раз проходится по всей последовательности и создает словарь частот всех элементов, тогда как метод count() многократно обходит список для каждого элемента.

Таблица 3. Время выполнения count и Counter'a.

|  |  |
| --- | --- |
| Ответ на запросы о количестве вхождений элементов в список или букв в строку | Время выполнения на чисел |
| count |  |
| Counter |  |

1. Квадратичная сортировка, быстрая сортировка, встроенная сортировка на числах.

В Таблице 4 продемонстрировано, что сортировка пузырьком является самой медленной: она работает в 30.5 раза медленнее, чем быстрая сортировка. Однако встроенная функция sorted() опережает быструю сортировку в 992 раза, потому что встроенная функция использует оптимизированный алгоритм Timsort, который разработан специально для эффективной работы с данными.

def bubble(a):

n = len(a)

for i in range(n - 1):

for j in range(i + 1, 0, -1):

if a[j] < a[j - 1]:

temp = a[j]

a[j] = a[j - 1]

a[j - 1] = temp

def quicksort(a, i, j):

if i >= j:

return

pivotindex = findpivot(a, i, j)

pivot = a[pivotindex]

a[pivotindex], a[j] = a[j], a[pivotindex]

k = partition(a, i, j, pivot)

a[k], a[j] = a[j], a[k]

quicksort(a, i, k - 1)

quicksort(a, k + 1, j)

def findpivot(a, i, j):

    firstkey = a[i]

    for k in range(i + 1, j + 1):

        if a[k] > firstkey:

            return k

        elif a[k] < firstkey:

            return i

    return i

def partition(a, i, j, pivot):

    l = i

    r = j - 1

    while l <= r:

        while a[l] < pivot:

            l += 1

        while r >= 0 and a[r] >= pivot:

            r -= 1

        if l < r:

            temp = a[r]

            a[r] = a[l]

            a[l] = temp

    return l

Таблица 4 Время выполнения квадратичной, быстрой и встроенной сортировок на числах.

|  |  |
| --- | --- |
| Название функции | Время выполнения на чисел |
| bubble |  |
| quicksort |  |
| sorted |  |

1. Квадратичная сортировка, быстрая сортировка, встроенная сортировка на строках.

Результаты Таблицы 5 аналогичны: самой быстрой сортировкой является встроенная функция sorted(), на втором месте ‑‑ quicksort, а на третьем – сортировка пузырьком.

Таблица 5. Время выполнения квадратичной, быстрой и встроенной сортировок на строках.

|  |  |
| --- | --- |
| Название функции | Время выполнения на строк |
| bubble |  |
| quicksort |  |
| sorted |  |

Задача 2. Проверить экспериментально, подходит ли коллекция list как основа для абстрактных структур данных стек и очередь. Для этого нужно определить характер зависимости времени выполнения операций (добавления и извлечения элементов) от размера коллекции и сравнить с использованием коллекции deque.

Решение:

Из Таблицы 6 для стека и очереди имеем разные выводы:

* Для стека: разница между list и deque незначительная. Поэтому для простоты и читаемости кода можно использовать списки.
* Для очереди: list работает существенно медленнее, чем коллекция deque. Коллекция deque оптимизирована для операций добавления и извлечения элементов с обеих сторон, что делает ее наиболее эффективной реализацией.

Таблица 6. Время выполнения стека и очереди на структуре list и deque.

|  |  |  |  |
| --- | --- | --- | --- |
| Абстрактная структура данных | Коллекция | Действие | Время выполнения на 1000 элементов |
| Стек | list | Добавление |  |
| Извлечение |  |
| deque | Добавление |  |
| Извлечение |  |
| Очередь | list | Добавление |  |
| Извлечение |  |
| deque | Добавление |  |
| Извлечение |  |

Проведенные эксперименты показали, что нужно осознанно выбирать алгоритмы и структуры данных для обеспечения эффективной производительности программ.

## **2.3 Модуль NumPy**

В этом разделе представлены решения задач с использованием изученной библиотеки NumPy, позволяющей работать с многомерными массивами и матрицами. Эта библиотека предоставляет большой набор функций для выполнения быстрых операций с массивами, включая математические и логические операции, базовые операции линейной алгебры, моделирование случайных чисел и многое другое.

Подробное решение задач из раздела «Библиотека NumPy» продемонстрировано в Приложении В.

**2.3.1 Модули math и numpy**

**Задача 1. Числовая змейка 3.0**

**Необходимо реализовать числовую змейку на основе массивов. Нужно реализовать функцию snake, которая принимает ширину и высоту змейки, а также именованный параметр direction, который может принимать значения:**

* **H – горизонтальная змейка, используется по умолчанию;**
* **V – вертикальная змейка.**

**Функция должна вернуть матрицу, представляющую змейку, с ячейками типа int16.**

**Решение: чтобы сделать горизонтальную змейку, необходимо создать массив чисел от до . Метод reshape(n, m) преобразует матрицу на .**

**Далее все нечетные строки оставляются без изменений, а четные строки разворачиваются:**

**a = np.arange(1, m \* n + 1, dtype=np.int16).reshape(n, m)**

**a[1::2, :] = a[1::2, ::-1]**

**Для того, чтобы создать вертикальную змейку необходимо транспонировать созданную матрицу . Все нечетные столбцы остаются без изменений, а четные столбцы переворачиваются.**

**a = np.arange(1, m \* n + 1, dtype=np.int16).reshape(m, n).T**

**a[:, 1::2] = a[::-1, 1::2]**

**Таким образом, была реализована функция snake, которая позволяет создать оба варианта числовой змейки.**

**Задача 2. Вращение**

**Необходимо написать функцию rotate, принимающую двумерную матрицу и один из углов поворота: и . Функция должна вернуть новую матрицу, соответствующую заданному повороту по часовой стрелке.**

**Решение: используется функция из библиотеки NumPy np.rot90(matrix, k), где k – количество поворотов на градусов против часовой стрелки.**

**Поскольку функция np.rot90() поворачивает матрицу против часовой стрелки, а задача требует поворот по часовой стрелке, необходимо воспользоваться некоторыми знаниями из курса школьной математики: поворот на против часовой стрелки эквивалентен повороту на по часовой стрелке, поскольку оба действия приводят к одной и той же точке на окружности или к одному и тому же положению. Поэтому была применена следующая формула:**

**np.rot90(matrix, (360 - angle) // 90)**

**Задача 3. Лесенка**

**Требуется написать функцию stairs, принимающую вектор и возвращающую матрицу, в которой каждая строка является копией вектора со смещением.**

**Решение: используется построение матрицы индексов с последующим вычислением остатка от деления на . Каждая строка матрицы будет представлена как циклический сдвиг исходного вектора.**

def stairs(v):

n = v.shape[0]

i = (np.arange(n) - np.arange(n).reshape(-1, 1)) % n

return v[i]

**2.3.2 NumPy. Знакомство**

В данном подразделе представлено решение ряда практических задач, направленных на ознакомление с библиотекой Python.

Задача:

1. Сгенерировать матрицу размером , заполненную целыми числами.
2. Вывести на экран элемент с индексами
3. Вывести на экран четвертую строку матрицы.
4. Вывести на экран предпоследний столбец матрицы в обратном порядке.
5. Изменить форму матрицы с на .
6. Разделить каждый элемент матрицы на заданное число.
7. Найти максимум в каждой строке.
8. Найти минимальный элемент на побочной диагонали.
9. Дан одномерный массив. Определить процент четных и нечетных чисел.
10. В заданной матрице подсчитать количество двузначных чисел.

Решение:

1. Был создан массив размеров , заполненный целыми случайными числами от до с помощью метода np.random.randint

matrix = np.random.randint(0, 1024, (12, 6))

1. Чтобы получить элемент из матрицы с нужными индексами, применяется обращение по индексам matrix[7, 5]. Это соответсвует 8-й строке и 6-му столбцу, так как индексация начинается с 0.
2. Чтобы вывести четвертую строку матрицы на экран, нужно обратится к индексу этой строки: matrix[4], который выводит на экран всю строку матрицы.
3. Для вывода на экран предпоследнего столбца матрицы в обратном порядке необходимо взять срез предпоследнего столбца и оператор [::-1] для вывода порядка элементов в обратном порядке.

matrix[::-1, -2]

1. Для того, чтобы изменить форму матрицы на нужно воспользоваться функцией reshape() из библиотеки NumPy, которая позволяет изменить форму массива на новую форму, при этом сохраняет количество элементов прежним.

matrix.reshape(6, 12)

1. Чтобы разделить каждый элемент матрицы на заданное число, можно применить математическую операцию деления в Python. Это работает благодаря векторизованным операциям NumPy.

matrix / n

1. Для поиска максимума в каждой строке, нужно обратится к методу max с параметром axis=1. Это позваоляет легко вычислять максимум по строкам.

np.max(matrix, axis=1)

1. Чтобы найти минимальный элемент на побочной диагонали был использован метод np.fliplr, отражающий матрицу по вертикали, метод np.diagonal возвращает главную диагональ матрицы. Таким образом, вместе эти функции помогли найти минимальный элемент на побочной диагонали матрицы.

np.min(np.diagonal(np.fliplr(matrix)))

1. Для того, чтобы определить процент четных и нечетных чисел в одномерном массиве, необходимо подсчитать количество четных и нечетных элементов в массиве

even\_num = np.sum(m % 2 == 0)

odd\_num = len(m) - even\_num

Далее вычисляется процент четных и нечетных элементов в массиве с помощью пропорции:

print(f'Процент четных элементов в массиве: {(100 \* even\_num / len(m)):.2f}')

print(f'Процент нечетных элементов в массиве: {(100 \* odd\_num / len(m)):.2f}')

1. Чтобы в заданной матрице подсчитать количество двузначных чисел. В этом помогает функция np.sum(), которая суммирует элементы массива по следующим условиям: матрица должна быть не менее 10 и не более 99.

print(f'Количество двузначных чисел: {np.sum((mtrx >= 10) & (mtrx <= 99))}')

**2.3.3 NumPy. Обработка матрицы**

В этом подразделе показана задача, которая направлена на эффективную работу с матрицами с помощью библиотеки NumPy.

Задача: необходимо выполнить обработку элементов прямоугольной матрицы , имеющей строк и столбцов. Найти наибольший элемент столбца матрицы, для которого сумма абсолютных значений элементов максимальна.

Решение: создается случайная матрица из 1000 чисел с помощью np.random.randint(). Для нахождения средних значений каждой строки матрицы используется функция np.mean() с параметром axis=1. С помощью метода max() находится максимальное значение среди средних значений для каждой строки матрицы.

A = np.random.randint(0, 1000, (n, m))

avrg = np.mean(A, axis=1)

print(np.max(avrg))

**2.3.4 NumPy. Вектор и матрица**

В этом подразделе рассматривается задача, направленная на работу с векторами и матрицами с помощью библиотеки NumPy.

Задача: дан вектор 𝑥 и матрица . Умножить все нечетные строки матрицы на вектор в обратном порядке. Результат записать в новую матрицу и найти сумму ее элементов.

Примечание 1. Считать, что размерности вектора и матрицы соответствуют выполняемым операциям.

Примечание 2. Фраза умножение в обратном порядке означает, что умножение происходит по правилу: последний элемент строки на первый элемент вектора, …, первый элемент строки на последний элемент вектора.

Решение:создается матрица размером и вектор длиной с помощью функции np.random.randint(). С помощью оператора вектор инвертируется. Затем умножение выполняется построчно для нечетных строк матрицы : new\_A = A[1::2, :] \* new\_x.

Затем сумма элементов новой матрицы вычисляется методом np.sum(new\_A).

**2.3.5. NumPy. Сравнение эффективности**

В данном подразделе рассматривается экспериментальная задача по сравнению эффективности реализации алгоритма нахождения наиболее часто встречающихся значений в матрице с помощью стандартных средств Python и библиотеки NumPy.

Задача:написать программу, реализующую алгоритм нахождения наиболее часто встречающихся значений в матрице с использованием list, замерить время выполнения и сравнить с готовой реализацией алгоритма из библиотеки NumPy. Для более полного анализа проводить сравнения на разных размерностях входных данных.

Решение: первая реализация алгоритма нахождения наиболее часто встречающихся значений в матрице основана на использовании list:

def most\_value(matrix):

lst = []

for row in matrix:

lst.extend(row)

cnt = Counter(lst)

max\_cnt = max(cnt.values())

repeat = [key for key, value in cnt.items() if value == max\_cnt]

return repeat[0] if len(repeat) == 1 else repeat

Вторая реализация алгоритма выполнена с использованием готовой реализации из библиотеки NumPy:

def np\_most\_value(matrix):

matrix\_rvl = matrix.ravel()

el, cnt = np.unique(matrix\_rvl, return\_counts=True)

max\_cnt = np.max(cnt)

repeat = el[cnt == max\_cnt]

return repeat[0] if len(repeat) == 1 else repeat

Теперь на основе Таблицы 7 проанализируем результаты и сравним время выполнения функций most\_value и np\_most\_value.

Рассмотрим результаты, когда имеется матрица :

Функция np\_most\_value работает примерно раза медленнее. Это обусловлено тем, что на маленьких данных подготовительные действия библиотеки NumPy (проверка типа данных, выделение памяти под массивы, запуск оптимизированных функций) занимают больше времени, чем работа алгоритма. Игра не стоит свеч, поэтому на маленьких данных лучше использовать методы из встроенных библиотек.

Рассмотрим результаты, когда имеется матрица :

NumPy оказывается в раза быстрее стандартных модулей Python.Начиная с этого момента, можно увидеть преимущества использования библиотеки NumPy.

Рассмотрим результаты, когда имеется матрица :

NumPy выполняется примерно в раз быстрее. Здесь можно заметить, что библиотека NumPy демонстрирует значительное превосходство во время работы с большими массивами данных.

Таблица 7. Время выполнения алгоритмов нахождения наиболее часто встречающихся элементов на разных размерностях матрицы

|  |  |  |  |
| --- | --- | --- | --- |
| Название функции | Время выполнения на матрице | Время выполнения на матрице | Время выполнения на матрице |
| most\_value |  |  |  |
| np\_most\_value |  |  |  |

## **2.4 Модуль Pandas**

Полное решение задач из раздела «Модуль Pandas» показано в Приложении Г.

**2.4.1 Хендбук модуль Pandas**

Задача 1. Длины всех слов по чётности

Требуется придумать функцию length\_stats, которая получает текст, а возвращает пару объектов Series со словами в качестве индексов и их длинами в качестве значений.

Решение: для начала весь текст приводится к нижнему регистру. Извлечение слов происходит с помощью регулярного выражения, позволяющего выделять последовательности латинских и русских букв. Далее удаляются дубликаты с помощью коллекции set(), остаются только уникальные слова, а оставшиеся слова сортируются в алфавитном порядке с использованием встроенной сортировки sorted(). На основании списка слов формируется объект Series, где индексами выступают сами слова, а значениями – их длины. Затем происходит разделение на четные и нечетные группы. Фильтрация происходит с помощью булевой индексации.

wrd = sorted(set(re.findall(r'[a-zа-я]+', text.lower())))

wrd\_series = pd.Series([len(i) for i in wrd], index=wrd)

return wrd\_series[wrd\_series % 2 != 0], wrd\_series[wrd\_series % 2 == 0]

Функция возвращает пару объектов Series, содержащих уникальные слова и их длины, разделенные по признаку четности.

Задача 2. Акция

Магазин проводит акцию «При покупке больше двух товаров – скидка », но скидка распространяется только на товары, купленные в количестве более двух штук. Требуется написать функцию discount, принимающую чек и возвращающую новый с учетом акции.

Решение: перед тем, как приступить к написанию требуемой функции, нужно написать функцию cheque, которая принимает прайс-лист и список покупок в виде неопределенного количества именованных аргументов (ключ – название товара, значение – количество). Функция возвращает объект DataFrame со столбцами:

* Наименование продукта (product).
* Цена за единицу (price).
* Количество (number).
* Итоговая цена (cost).

pd.DataFrame(

{

'product': product\_cheque,

'price': [price\_list[i] for i in product\_cheque],

'number': [kwargs[i] for i in product\_cheque],

'cost': [price\_list[i] \* kwargs[i] for i in product\_cheque]

}

Строки чека отсортированы по названию продуктов в лексикографическом порядке.

После этого создается функция discount(). Создается булева маска, где True для товаров с количество больше двух, чтобы сохранить требуемое условие задачи. Далее строится математическая модель скидки:

Если товаров было куплено не более 2, то для маски, равной False, строится следующая модель без скидки: cost / (1 + 0) = cost.

Если товаров было куплено более 2, то для маски, равной True, будет строится следующее утверждение со скидкой в : cost / (1 + 1) = cost / 2.

mask = new['number'] > 2

new['cost'] = new['cost'] / (1 + mask)

Задача 3. Обновление журнала

Дан журнал успеваемости учеников, представленный в виде объекта DataFrame, содержащий следующие столбцы:

* name – имя;
* maths – оценка по математике;
* physics – оценка по физике;
* computer science – оценка по информатике.

Требуется написать функцию update, которая добавляет к данным столбец average, содержащий среднюю оценку ученика, а также сортирует данные по убыванию этого столбца, а при равенстве средних – по имени лексикографически.

Решение: средняя оценка для каждой строки вычисляется с помощью метода .mean(axis=1), где параметр axis=1 указывает на выполнение операции по строкам. Для вычисления среднего арифметического берутся все три оценки ученика. Результат сохраняется в новом столбце average.

Затем происходит сортировка данных по убыванию этого столбца, а в случае, если средние баллы учеников равны – по имени в алфавитном порядке.

Для сортировки используется метод .sort\_values(), который позволяет задать сразу несколько ключей и направление сортировки для каждого из них.

j['average'] = j.iloc[:, 1:4].mean(axis=1)

return j.sort\_values(['average', 'name'], ascending=[False, True])

По итогу была реализована функция update, предназначенная для обработки журнала успеваемости студентов с целью вычисления средней оценки каждого ученика, сортировки студентов по успеваемости и алфавитному порядку и возвращает обновленный журнал без изменения исходных данных.

**2.4.2 Модуль Pandas. Туры**

В данном подразделе рассматриваются задачи анализа и визуализации данных на основе фрагмента базы данных, содержащей информацию о продажах туров в различные города россии.

Задача: дан фрагмент базы данных, содержащий информацию о продажах туров в различные города России. База данных состоит из трёх таблиц в формате csv с разделителем «точка с запятой».

Файл travel\_agents.csv содержит информацию о туроператорах. Заголовок таблицы имеет следующий вид: ID туроператора; Название; Адрес

Файл travels.csv содержит информацию об основных характеристиках каждого тура. Заголовок таблицы имеет следующий вид: ID тура; Город; Продолжительность, дней; Стоимость, на 1 человека

Файл sale\_of\_tour\_packages.csv содержит информацию о проданных турах за первый квартал 2022 года. Заголовок таблицы имеет следующий вид: ID операции; Дата; ID тура; ID туроператора; Количество проданных путёвок;

1. Найти количество продаж туров длительностью неделя и более.
2. Построить круговую диаграмму, отображающую количество человек, которые совершили путешествия, пользуясь услугами туроператора «Вместе». Диаграмма должна отображать общее количество человек и распределение их по городам путешествия.
3. Построить диаграмму, показывающую зависимость стоимости путёвки от её продолжительности.

Решение:

1. Для решения задачи объединяются данные из файлов sale\_of\_tour\_packages.csv и travels.csv по ID тура. После этого отбираются туры с продолжительностью неделя и более, и подсчитывается общее количество проданных путевок:

data\_merge = pd.merge(sales, travels, on='ID тура')

tour\_weekly = data\_merge[data\_merge['Продолжительность, дней'] >= 7]

sale = tour\_weekly['Количество проданных путёвок'].sum()

Так можно узнать, что было продано путевок длительностью неделя и более.

1. Сначала определяется ID туроператора «Вместе» ‑‑ Т114.

agent\_id = agents[agents['Название'] == 'Вместе']['ID туроператора'].values[0]

Затем из базы продаж можно узнать продажи только этого туроператора, а затем объединить данные о турах с csv-файлом travels:

agent\_sales = sales[sales['ID туроператора'] == agent\_id]

mrg\_data = pd.merge(agent\_sales, travels, on='ID тура')

Сгруппировав данные по городам и количеству проданных путевок, можно просуммировать, сколько всего было продано путевок в один город.

city = mrg\_data.groupby('Город')['Количество проданных путёвок'].sum()

diagrama = city.plot.pie(autopct='%.2f%%', figsize=(10, 10))

На Рисунке 1 представлена круговая диаграмма, позволяет оценить, какие города пользовались наибольшей популярностью у клиентов туроператора «Вместе».

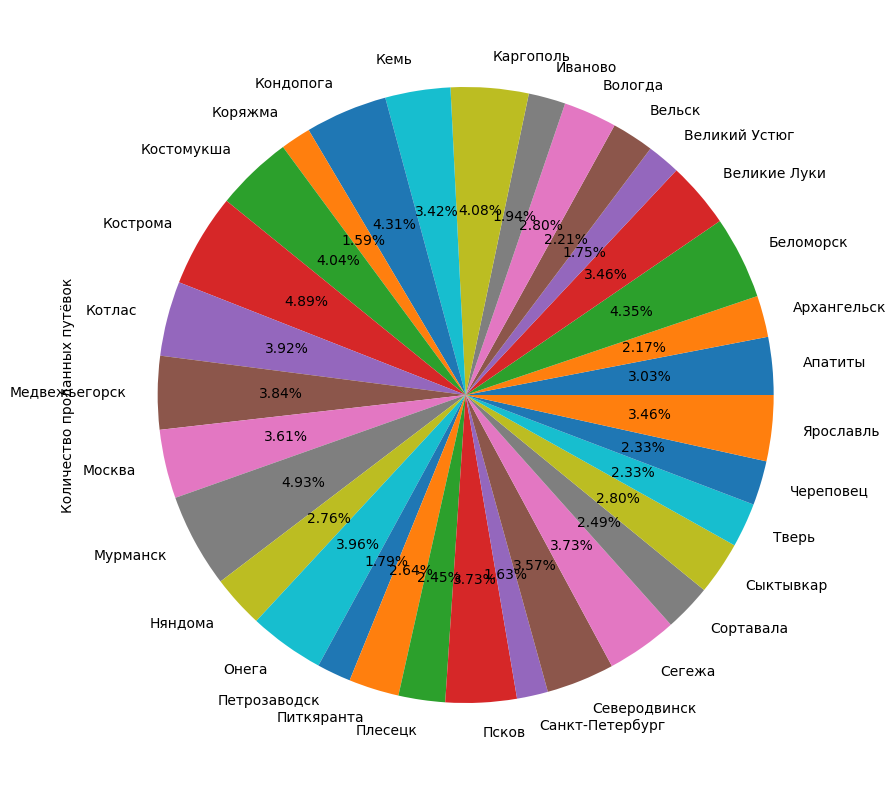


Рисунок 1. Круговая диаграмма, отображающая количество человек, которые совершили путешествия, пользуясь услугами туроператора «Вместе».

1. Для анализа зависимости используется диаграмма рассеяния, где по оси

откладывается продолжительность тура, а по оси – его стоимость:

d = travels.plot.scatter(x = 'Продолжительность, дней', y = 'Стоимость, на 1 чел', figsize=(10, 10))

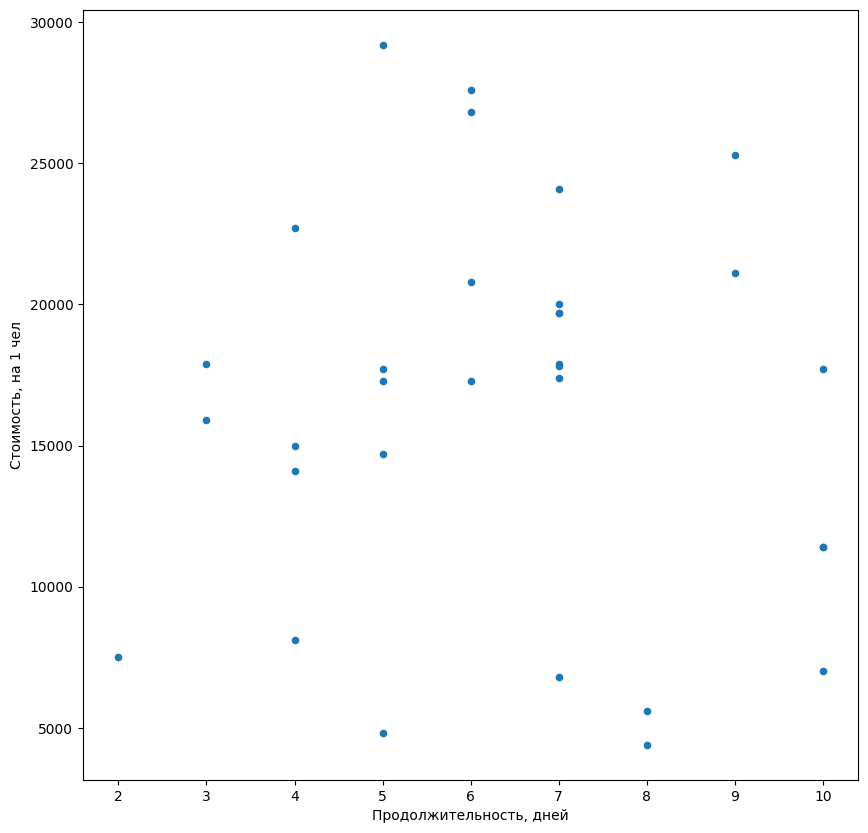


Рисунок 2. График увеличения стоимости туров при росте их продолжительности

На Рисунке 2 изображен график, который показывает, как увеличивается стоимость туров при росте их продолжительности, а также позволяет выявить выбросы и нетипичные значения.

## **2.5 Визуализация**

В этом разделе для более наглядного представления закономерности в данных, рассматривается визуализация результатов статистического анализа при помощи использования библиотек Pandas и Matplotlib. Полный код заданий этого раздела можно увидеть в Приложении Д.

**2.5.1 Перевозки (статистика)**

В данном подразделе будет проведено статистическое исследование, связанное с анализом данных о перевозках, и представление результатов с использованием различных диаграмм.

Задача: провести небольшое статистическое исследование, для визуализации результатов построить не менее трех диаграмм и графиков (например, круговую диаграмму, показывающую суммарный расход бензина по датам по всем направлениям).

Решение: всего было построено три графика:

* круговая диаграмма суммарного расхода бензина по датам по всем направлениям;
* диаграмма рассеяния зависимости расхода бензина от расстояния;
* столбчатая диаграмма средней массы груза по пунктам отправлениям.

1. Для анализа суммарного расхода бензина данные были сгруппированы по дате, после чего вычислили суммарный расход бензина вне зависимости от направления, куда движется автомобиль с грузом:

petrol = data.groupby('Дата')['Расход бензина'].sum()

Построенная круговая диаграмма, представленная на Рисунке 3 позволила наглядно оценить, какой расход топлива по датам. Диаграмма показывает процентное соотношение ежедневного расхода относительного общего. Видно, что в некоторые дни расход топлива немного превышал среднее значение, что может говорить о возможной повышенной интенсивности перевозок или использования более энергоемких транспортных средств.

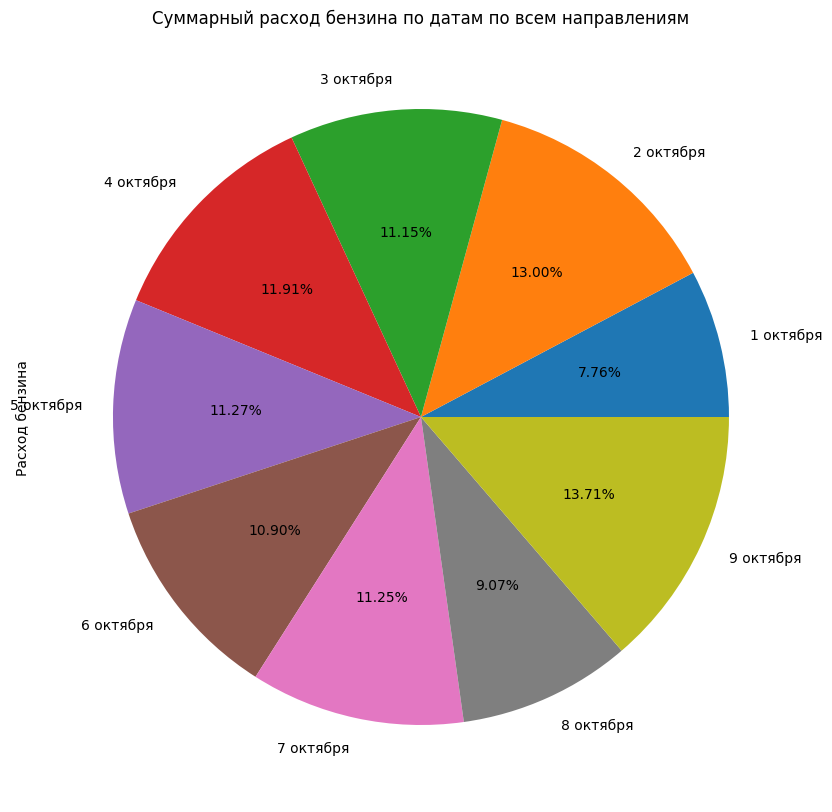


Рисунок 3. Суммарный расход бензина по датам по всем направлениям.

1. Чтобы проанализировать зависимость расхода бензина от расстояния, построили диаграмму рассеяния. На оси отложено расстояние перевозки, а по оси – расход бензина:

plt.figure(figsize=(10, 10))

plt.scatter(x=data['Расстояние'], y=data['Расход бензина'])

На диаграмме, показанной на Рисунке 4, можно отследить, что существует прямая зависимость: чем больше расстояние перевозки, тем выше расход топлива. При этом на графике можно заметить, что бывают случаи, когда расход бензина оказался выше ожидаемого для данного расстояния.

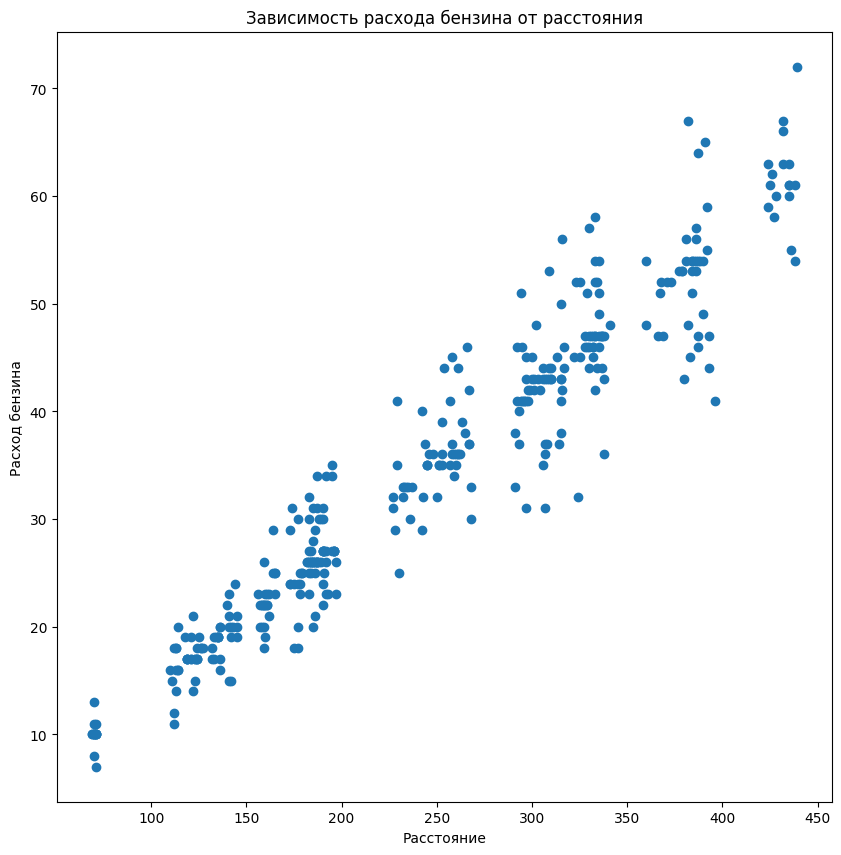


Рисунок 4. Зависимость расхода бензина от расстояния.

1. Для выявления закономерностей в распределении грузов по пунктам отправления, была построена столбчатая диаграмма, где данные посчитаны с помощью среднего значения массы груза:

data.groupby('Пункт отправления')['Масса груза'].mean().plot.bar()

Столбчатая диаграмма на Рисунке 5 показала, что масса груза практически не изменяется в зависимости от пункта направления. Это свидетельствует о том, что существенных различий по направлениям не наблюдается.

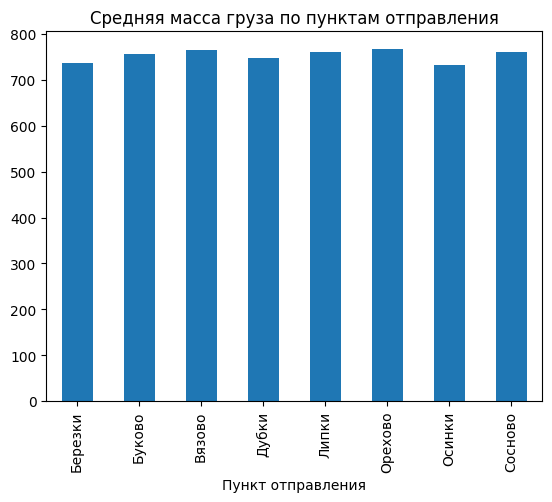


Рисунок 5. Диаграмма средней массы груза по пунктам отправления.

# **3. Решение математических задач**

## **3.1 Линейные подпространства**

Задача: для подпространства , порожденного системой векторов и подпространства всех решений однородной системы уравнений: найти:

* размерность и базис суммы ;
* размерность и базис пересечения .

Решение:

Для подпространства необходимо собрать матрицу, столбцами которой являются векторы , найти ее ранг с помощью метода элементарных преобразований строк. Ранг матрицы – максимальное количество линейно независимых строк. Так можно узнать размерность

Приведение этой матрицы к ступенчатому виду демонстрирует, что в матрице есть три линейно независимых столбца, следовательно,

Подпространство по условию задачи задается двумя уравнениями:

Таким образом, общее решение можно записать параметрически:

Отсюда можно выразить векторы подпространства

Теперь можно узнать размерность

Чтобы найти размерность : , необходимо составить матрицу, столбцами которой являются векторы и вычислить ее ранг.

После приведения матрицы к ступенчатому виду, можно заметить, что из этого следует, что .

Базис суммы можно выбрать как любой набор из четырех линейно независимых векторов:

Используя формулу размерностей подпространств, можно вывести размерность :

Подставляя известные значения, можно получить:

Так можно понять, что пересечение является одномерным. Теперь нужно найти его ненулевой вектор:

Подпространство

Далее для дальнейших вычислений системы и объединяются, затем из решения системы линейных уравнений можно найти

При помощи математических преобразований этой системы линейных уравнений, можно вычислить все неизвестные переменные:

Был найден базис пересечения подпространств :

Ответ:

* + 1. ;
    2. Базис суммы подпространств – ;
    4. Базис пересечения подпространств: .

## **3.2 Ядро, образ**

Задача: найти ранг, базисы ядра и образа линейного оператора такого, что для любой матрицы :

Решение: Стандартный базис пространства , имеющее размерность 4 задается матрицами:

Любую матрицу можно записать как:

Перед тем, как получить матрицу линейного оператора, применяем оператор к каждому и раскладываем результат по базису . Выполненные преобразования дают:

Для :

Для

Для

Для :

Соответствующая матрица линейного оператора в базисе {e1, e2, e3, e4} имеет вид:

Приведем матрицу линейного оператора к ступенчатому виду :

Можно заметить, что матрица содержит три линейно независимые строки, следовательно .

Теперь можно приступить к нахождению базиса ядра.

Ядро линейного оператора — полный прообраз множества {0}, т. е. множество всех векторов, которые переводятся линейным оператором в 0 и вычисляется следующим образом: .

Чтобы упростить систему линейных уравнений, можно убрать из матрицы одну из линейно зависимых строк – (0, 3, 2, -2) и система будет выглядеть следующим образом:

,

Любую матрицу подпространства можно разложить по базису:

– это и есть базис ядра.

Затем был найден образ линейного оператора:

Образ линейного оператора – это линейная оболочка столбцов матрицы . Чтобы его найти, нужно дополнить до базиса всего пространства:

, поскольку

Поскольку , а также имеется один линейно зависимый столбец, тополучаем, что образ линейного оператора представляет собой:

Ответ:

* + 1. ;

## **3.3 Норма**

Задача: в следующих примерах найти длины (нормы) элементов и угол между ними в евклидовом пространстве всех многочленов с вещественными коэффициентами и скалярным произведением , .

Решение:

* 1. Найдем нормы (длины) элементов :
  2. Найдем скалярное произведение
  3. Найдем угол между :

Ответ:

1. ;

## **3.4 Угол, расстояние**

Задача: найти проекцию вектора на подпространство решений следующих систем уравнений и его ортогональную составляющую . Найти угол между вектором и подпространством. Найти расстояние от вектора до этого подпространства.

Решение:

Найдем базис подпространства :

Найдем ортогональную проекцию X на U:

Найдем проекцию , так как ортогональны, коэффициенты найдем следующим образом:

В формулу (\*\*) подставим значения b :

Чтобы найти ортогональную составляющую выразим из формулы (\*) следующее:

– ортогональная составляющая

Найдем угол между и подпространством :

Найдем расстояние от вектора до этого подпространства:

Ответ:

1. ;

## **3.5 Ортогональная проекция**

Задача: найти ортогональное преобразование, приводящее следующие формы к каноническому виду (приведение к главным осям), и написать этот канонический вид (преобразование определено неоднозначно):

Решение:

Приведем к каноническому виду формулу с помощью ортогональных преобразований:

Найдем собственное значение и собственный вектор и решим характерическое уравнение:

Определитель равен сумме произведений элементов любой строки на их алгебраические дополнения:

Формула (\*\*\*) сводит вычисление определителя -го порядка к вычислению определителей-го порядка.

Разложим определитель по 4-й стройке:

Далее представлено готовое решение алгебраических дополнений:

Из выражения (1) получим уравнение:

Решим биквадратное уравнение и получим корни уравнения:

Для каждого решают систему:

При , получаем:

Составим систему линейных уравнений:

При , получаем:

Составим систему линейных уравнений:

Таким же способом вычислим для и , получим:

Найдем ортогональный базис:

В каждом евклидовом пространстве существует ортонормированный базис. Воспользуемся процессом Грама-Шмидта:

Находим ортонормированный базис:

Теперь покажем матрицу перехода:

Теперь выведем канонический вид. В базисе из собственных векторов квадратичная форма принимает вид:

Где коэффициенты – собственные значения матрицы

# **ЗАКЛЮЧЕНИЕ**

В ходе практики были изучены инструменты предобработки и анализа данных с использованием языка Python и его стандартных модулей, а также библиотек NumPy, Pandas и библиотек визуализации Matplotlib и NetworkX. Отдельное внимание было уделено обработке исключений, что является важным аспектом для дальнейшей корректно работы с непредвиденными ситуациями при выполнении программного кода.

В рамках летней практики было решено 47 задач по программированию и 5 задач по линейной алгебре. Это позволило закрепить полученные знания и приобрести практические навыки решения прикладных задач, связанных с предобработкой, анализом и визуализацией данных.

Таким образом, по результатам летней практики были сформированы важные навыки работы с данными в Python, научиться предобрабатывать и визуализировать большие массивы данных. Полученные навыки позволили заложить фундамент знаний для дальнейшего изучения дисциплин, связанный с анализом данных.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. TIOBE Index for September 2025 [Электронный ресурс]. URL: <https://www.tiobe.com/tiobe-index/> (дата обращения: 25.09.2025)
2. Python Documentation. Errors and Exceptions [Электронный ресурс]. URL: <https://docs.python.org/3/tutorial/errors.html> (дата обращения: 03.07.2025)
3. Python Documentation. Random — Generate pseudo-random numbers [Электронный ресурс]. URL: <https://docs.python.org/3/library/random.html> (дата обращения: 05.07.2025)
4. Python Documentation. Math — Mathematical functions [Электронный ресурс]. URL: <https://docs.python.org/3/library/math.html> (дата обращения: 03.07.2025)
5. NumPy Documentation [Электронный ресурс]. URL: <https://numpy.org/> (дата обращения: 12.07.2025)
6. Pandas documentation [Электронный ресурс]. URL: <https://pandas.pydata.org/docs/index.html> (дата обращения: 14.07.2025)
7. Matplotlib documentation [Электронный ресурс]. URL: <https://matplotlib.org/stable/index.html> (дата обращения: 17.07.2025)
8. NetworkX documentation [Электронный ресурс]. URL: <https://networkx.org/documentation/stable/index.html> (дата обращения: 29.06.2025)
9. МакКинни У. Python и анализ данных / 2-е изд. пер. с анг. А. А. Слинкина – ДМК Пресс, 2020. – 180 с
10. Hagberg A.A., Schult D.A., Swart P.J. Exploring Network Structure, Dynamics, and Function using NetworkX // Proceedings of the 7th Python in Science Conference. Pasadena, CA, USA, 2008. 11 – 15 c
11. Deisenroth M.P., Faisal A.A., Ong C.S. Mathematics for Machine Learning. Cambridge University Press, 2020, 70 – 81 c.
12. Курош А.Г. Курс высшей алгебры / 9-е изд. «Наука», Москва 1968. 184 – 222 с.

# **ПРИЛОЖЕНИЕ А Обработка исключений**

**Хендбук 5.3. Модель исключений Python. Try, except, else, finally. Модули**

**Корень зла 2**

class NoSolutionsError(Exception):

pass

class InfiniteSolutionsError(Exception):

pass

def find\_roots(a, b, c):

d = b \*\* 2 - 4 \* a \* c

roots = ()

if not all(isinstance(n, (int, float)) for n in (a, b, c)):

raise TypeError

if a == b == c == 0:

raise InfiniteSolutionsError

elif a == b == 0:

raise NoSolutionsError

elif a == c == 0:

roots = (0, 0)

else:

if a == 0:

print(-c / b)

if b == 0:

if c > 0:

x1 = (c / a) \*\* 0.5

x2 = -((c / a) \*\* 0.5)

roots = tuple(sorted([x1, x2]))

else:

raise NoSolutionsError

if a != 0:

if b == c == 0:

roots = ((0 / a), (0 / a))

if d < 0:

raise NoSolutionsError

if d == 0:

roots = (-b / (2 \* a), -b / (2 \* a))

if d > 0:

x1 = (-b + d \*\* 0.5) / (2 \* a)

x2 = (-b - d \*\* 0.5) / (2 \* a)

roots = tuple(sorted([x1, x2]))

return roots

**Валидация пользователя**

from string import ascii\_letters, digits

class CyrillicError(Exception):

pass

class CapitalError(Exception):

pass

class BadCharacterError(Exception):

pass

class StartsWithDigitError(Exception):

pass

def name\_validation(name):

cyrillic = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'

if not isinstance(name, str):

raise TypeError

if not all(s.lower() in cyrillic for s in name):

raise CyrillicError

if not name.istitle():

raise CapitalError

return name

def username\_validation(username):

tmp = f'{ascii\_letters}{digits}\_'

if not isinstance(username, str):

raise TypeError

if not all(s in tmp for s in username):

raise BadCharacterError

if username[0].isdigit():

raise StartsWithDigitError

return username

def user\_validation(\*\*user):

if list(user.keys()) != ['last\_name', 'first\_name', 'username']:

raise KeyError

if not all(isinstance(x, str) for x in user.values()):

raise TypeError

name\_validation(user['last\_name'])

name\_validation(user['first\_name'])

username\_validation(user['username'])

return user

**Валидация пароля**

from string import ascii\_letters, digits

from hashlib import sha256

class MinLengthError(Exception):

pass

class PossibleCharError(Exception):

pass

class NeedCharError(Exception):

pass

symbol = f'{ascii\_letters}{digits}'

def password\_validation(password, min\_length=8, possible\_chars=symbol, at\_least\_one=str.isdigit):

if not isinstance(password, str):

raise TypeError

if len(password) < min\_length:

raise MinLengthError

if any(i not in possible\_chars for i in password):

raise PossibleCharError

if not any(at\_least\_one(x) for x in password):

raise NeedCharError

return sha256(password.encode()).hexdigest()

# **ПРИЛОЖЕНИЕ Б Встроенные модули**

**Модули math и itertools**

**Задача 3. Размещения с повторениями**

from itertools import product

n, k = map(int, input().split())

print(n \*\* k)

for x in product(range(1, n + 1), repeat=k):

print(' '.join(map(str, x)))

**Задача 4. Числа в p-ичной системе счисления**

from itertools import product

k, p = map(int, input().split())

print((p - 1) \* p \*\* (k - 1))

for i in product(range(p), repeat=k):

if i[0] != 0:

print(''.join(map(str, i)))

**Задача 7. Сочетания**

from math import factorial

from itertools import combinations

n, m = map(int, input().split())

c = factorial(n) // (factorial(m) \* factorial(n - m))

print(c)

for i in combinations(range(1, n + 1), m):

print(' '.join(map(str, i)))

**Модуль random**

Вариант 8

import random  
  
from string import ascii\_letters, digits, ascii\_lowercase  
from collections import Counter  
  
  
def generation\_id():  
 while True:  
 user\_id = ''.join(random.choices(digits, k=3))  
 last\_digit = ''.join(random.choices(digits))  
 return f'{user\_id}{last\_digit}{last\_digit}'  
  
  
def generation\_username():  
 while True:  
 user\_name = ''.join(random.choices(ascii\_lowercase, k=6))  
 letters = Counter(x for x in user\_name)  
 if all(cnt <= 2 for cnt in letters.values()):  
 return user\_name  
  
  
def generation\_password():  
 symbols = f'{ascii\_letters}{digits}'  
 while True:  
 pwrd = ''.join(random.choices(symbols, k=10))  
 if pwrd[0] not in digits and any(n in digits for n in digits):  
 return pwrd  
  
  
def n\_lst(n):  
 lst = []  
 us\_id = set()  
 username = set()  
 password = set()  
  
 while len(lst) < n:  
 uid = generation\_id()  
 user = generation\_username()  
 passw = generation\_password()  
  
 if all([uid not in us\_id, user not in username, passw not in password]):  
 us\_id.add(uid)  
 username.add(user)  
 password.add(passw)  
 lst.append((uid, user, passw))  
 return lst  
  
  
print('Введите сколько хотите увидеть N троек вида (id, логин, пароль):')  
num = int(input())  
print(n\_lst(num))

**Измерение времени выполнения**

import time

from collections import Counter, deque

from random import randint, choice

from string import ascii\_letters

# проверка на простоту перебором до N,N//2 и до √N

def is\_prime\_n(n):

  if n < 2:

    return False

  for d in range(2, n + 1):

    if n % d == 0:

      return False

  return True

def is\_prime\_n\_div2(n):

    if n < 2:

        return False

    for d in range(2, (n // 2) + 1 ):

        if n % d == 0:

            return False

    return True

def is\_prime\_n\_sqrt(n):

    if n < 2:

      return False

    for d in range(2, round(n \*\* 0.5) + 1):

      if n % d == 0:

        return False

    return True

# Рассмотрим время выполнения проверки на простоту перебором до N:

%timeit is\_prime\_n(7)

%timeit is\_prime\_n(10007)

%timeit is\_prime\_n(1000003)

%timeit is\_prime\_n(10000019)

# Рассмотрим время выполнения проверки на простоту перебором до N//2:

%timeit is\_prime\_n\_div2(7)

%timeit is\_prime\_n\_div2(10007)

%timeit is\_prime\_n\_div2(1000003)

%timeit is\_prime\_n\_div2(10000019)

# Рассмотрим время выполнения проверки на простоту перебором до √N:

%timeit is\_prime\_n\_sqrt(7)

%timeit is\_prime\_n\_sqrt(10007)

%timeit is\_prime\_n\_sqrt(1000003)

%timeit is\_prime\_n\_sqrt(10000019)

# поиск элемента в списке и во множестве

data = [x for x in range(10\_000)]

data\_set = {x for x in range(10\_000)}

%%timeit

for x in data:

    if x in data:

        pass

%%timeit

for x in data\_set:

    if x in data\_set:

        pass

# Рассмотрим количество вхождений элементов в список с помощью count:

lst = [1, 2, 3] \* 1000000

%timeit lst.count(1)

%timeit lst.count(2)

%timeit lst.count(3)

# Рассмотрим количество вхождений элемента в список Counter'а:

%timeit cnt\_lst[1]

%timeit cnt\_lst[2]

%timeit cnt\_lst[3]

# квадратичная сортировка, быстрая сортировка, встроенная сортировка

# на числах

# Для квадратичной сортировки возьмем сортировку пузырьком:

array = [randint(1, 99) for n in range(10000)]

def bubble(a):

  n = len(a)

  for i in range(n - 1):

    for j in range(i + 1, 0, -1):

      if a[j] < a[j - 1]:

        temp = a[j]

        a[j] = a[j - 1]

        a[j - 1] = temp

%timeit bubble(array)

# Рассмотрим время выполнения быстрой сортировки:

def quicksort(a, i, j):

    if i >= j:

      return

    pivotindex = findpivot(a, i, j)

    pivot = a[pivotindex]

    a[pivotindex], a[j] = a[j], a[pivotindex]

    k = partition(a, i, j, pivot)

    a[k], a[j] = a[j], a[k]

    quicksort(a, i, k - 1)

    quicksort(a, k + 1, j)

def findpivot(a, i, j):

    firstkey = a[i]

    for k in range(i + 1, j + 1):

        if a[k] > firstkey:

            return k

        elif a[k] < firstkey:

            return i

    return i

def partition(a, i, j, pivot):

    l = i

    r = j - 1

    while l <= r:

        while a[l] < pivot:

            l += 1

        while r >= 0 and a[r] >= pivot:

            r -= 1

        if l < r:

            temp = a[r]

            a[r] = a[l]

            a[l] = temp

    return l

%timeit quicksort(array, 0, len(array) - 1)

# Теперь посмотрим встроенную сортировку на числах:

%timeit sorted(array)

# квадратичная сортировка, быстрая сортировка,

#встроенная сортировка на строках

s = [''.join(choice(ascii\_letters) for \_ in range(randint(3, 6))) for \_ in range(10000)]

%timeit bubble(s)

%timeit quicksort(s, 0, len(s) - 1)

%timeit sorted(s)

# Узнаем скорость добавления в стек с помощью списков

num = 10000

def lst\_append(n):

  lst = []

  for i in range(n):

    lst.append(i)

%timeit lst\_append(num)

# Теперь рассмотрим извлечение элементов в стеке на основе списков:

def lst\_pop(n):

  lst = list(range(n))

  for \_ in range(n):

    lst.pop()

%timeit lst\_pop(num)

# Посмотрим время выполнения добавления в стек с использованием

# коллекции deque:

def append\_stack(n):

  stack = deque()

  for i in range(n):

    stack.append(i)

%timeit append\_stack(num)

# Посмотрим время выполнения извлечение элементов в стеке

# с использованием коллекции deque:

def pop\_stack(n):

  stack = deque(range(n))

  for \_ in range(n):

    stack.pop()

%timeit pop\_stack(num)

# Реализуем добавление элемента в очереди с помощью списка:

def queue\_append(n):

  queue = []

  for i in range(n):

    queue.append(i)

%timeit queue\_append(num)

# Теперь рассмотрим извлечение элементов в очереди с помощью списка:

def queue\_pop(n):

  queue = list(range(n))

  for \_ in range(n):

    queue.pop(0)

%timeit queue\_pop(num)

# Теперь будем рассматривать добавление и извлечение элементов

# в очереди с помощью коллекции deque:

def queue\_a(n):

  queue = deque()

  for i in range(n):

    queue.append(i)

%timeit queue\_a(num)

def queue\_p(n):

  queue = deque(range(n))

  for \_ in range(n):

    queue.popleft()

%timeit queue\_p(num)

# **ПРИЛОЖЕНИЕ В Библиотека NumPy**

**Хендбук 6.1. Модули math и numpy**

**Числовая змейка 3.0**

import numpy as np

def snake(m, n, direction='H'):

if direction == 'H':

a = np.arange(1, m \* n + 1, dtype=np.int16).reshape(n, m)

a[1::2, :] = a[1::2, ::-1]

if direction == 'V':

a = np.arange(1, m \* n + 1, dtype=np.int16).reshape(m, n).T

a[:, 1::2] = a[::-1, 1::2]

return a

**Вращение**

import numpy as np

def rotate(matrix, angle):

return np.rot90(matrix, (360 - angle) // 90)

**Лесенка**

import numpy as np

def stairs(v):

n = v.shape[0]

i = (np.arange(n) - np.arange(n).reshape(-1, 1)) % n

return v[i]

[**NumPy Знакомство**](https://school.sgu.ru/mod/assign/view.php?id=18065)

Вариант 23

import numpy as np

matrix = np.random.randint(0, 1024, (12, 6))

print('Матрица 12x6:\n', matrix)

print('Элемент с индексами [7, 5]: \n', matrix[7, 5])

print('Четвертая строка матрицы: \n', matrix[4])

print(' Предпоследний столбец матрицы в обратном порядке: \n', matrix[::-1, -2])

print('\n')

print(' Изменить форму матрицы с 12×6 на 6×12: \n', matrix.reshape(6, 12))

# Разделить каждый элемент матрицы на заданное число.

n = int(input('Задайте число: '))

print(matrix / n)

# Найти максимум в каждой строке.

# Возвратим максимум по заданной оси axis=1 — строки, а axis=0 — столбцы

print(np.max(matrix, axis=1))

# Найти минимальный элемент на побочной диагонали.

print(np.min(np.diagonal(np.fliplr(matrix))))

# Дан одномерный массив. Определить процент четных и нечетных чисел.

m = np.random.randint(0, 100, 30)

print(m)

# Подсчитаем количество четных и нечетных элементов в массиве:

even\_num = np.sum(m % 2 == 0)

odd\_num = len(m) - even\_num

# Вычислим процент четных и нечетных элементов в массиве:

print(f'Процент четных элементов в массиве: {(100 \* even\_num / len(m)):.2f}')

print(f'Процент нечетных элементов в массиве: {(100 \* odd\_num / len(m)):.2f}')

# В заданной матрице подсчитать количество двузначных чисел.

# Зададим матрицу, с которым мы будем работать в дальнейшем:

print('Задайте размер матрицы')

n, m = map(int, input().split())

mtrx = np.random.randint(0, 1024, (n, m))

print(mtrx)

print(f'Количество двузначных чисел: {np.sum((mtrx >= 10) & (mtrx <= 99))}')

**NumPy Обработка матрицы.**

Вариант 22

import numpy as np

n, m = map(int, input().split())

A = np.random.randint(0, 1000, (n, m))

print(A)

avrg = np.mean(A, axis=1)

print(avrg)

print(np.max(avrg))

**NumPy Вектор и матрица**

Вариант 21

import numpy as np

A = np.random.randint(0, 100, (4, 3))

x = np.random.randint(-100, 100, 3)

new\_x = x[::-1]

new\_A = A[1::2, :] \* new\_x

print(np.sum(new\_A))

**NumPy Сравнение эффективности**

Вариант 20

import time

from collections import Counter

import random

import numpy as np

# Вычислим время выполнения индивидуального задания с использованием list:

def most\_value(matrix):

  lst = []

  for row in matrix:

    lst.extend(row)

  cnt = Counter(lst)

  max\_cnt = max(cnt.values())

  repeat = [key for key, value in cnt.items() if value == max\_cnt]

  return repeat[0] if len(repeat) == 1 else repeat

matrix = [[random.randint(1, 100) for \_ in range(10)] for \_ in range(10)]

%timeit most\_value(matrix)

matrix = [[random.randint(1, 100) for \_ in range(100)] for \_ in range(100)]

%timeit most\_value(matrix)

matrix = [[random.randint(1, 100) for \_ in range(500)] for \_ in range(500)]

%timeit most\_value(matrix)

# Теперь узнаем время выполнения индвидуального задания

# с использованием готовой реализации алгоритма из библиотеки NumPy

def np\_most\_value(matrix):

  matrix\_rvl = matrix.ravel()

  el, cnt = np.unique(matrix\_rvl, return\_counts=True)

  max\_cnt = np.max(cnt)

  repeat = el[cnt == max\_cnt]

  return repeat[0] if len(repeat) == 1 else repeat

matrix = np.random.randint(1, 100, (10, 10))

%timeit np\_most\_value(matrix)

matrix = np.random.randint(1, 100, (100, 100))

%timeit np\_most\_value(matrix)

matrix = np.random.randint(1, 100, (500, 500))

%timeit np\_most\_value(matrix)

# **ПРИЛОЖЕНИЕ Г Модуль Pandas**

**Хендбук 6.2. Модуль pandas**

**Длины всех слов по чётности**

import pandas as pd

import re

def length\_stats(text):

wrd = sorted(set(re.findall(r'[a-zа-я]+', text.lower())))

wrd\_series = pd.Series([len(i) for i in wrd], index=wrd)

return wrd\_series[wrd\_series % 2 != 0], wrd\_series[wrd\_series % 2 == 0]

**Акция**

import pandas as pd

def cheque(price\_list, \*\*kwargs):

product\_cheque = sorted(kwargs)

return pd.DataFrame(

{

'product': product\_cheque,

'price': [price\_list[i] for i in product\_cheque],

'number': [kwargs[i] for i in product\_cheque],

'cost': [price\_list[i] \* kwargs[i] for i in product\_cheque]

}

)

def discount(df):

new = df.copy()

mask = new['number'] > 2

new['cost'] = new['cost'] / (1 + mask)

return new

**Обновление журнала**

import pandas as pd

def update(journal):

j = journal.copy()

j['average'] = j.iloc[:, 1:4].mean(axis=1)

return j.sort\_values(['average', 'name'], ascending=[False, True])

**Модуль pandas**

Вариант 19

import pandas as pd

sales = pd.read\_csv('sale\_of\_tour\_packages.csv', delimiter=';')

travels = pd.read\_csv('travels.csv', delimiter=';')

agents = pd.read\_csv('travel\_agents.csv', delimiter=';')

# Объединим данные из sale\_of\_tour\_packages.csv и travels.csv по ID тура

# Решим первое задание:

data\_merge = pd.merge(sales, travels, on='ID тура')

tour\_weekly = data\_merge[data\_merge['Продолжительность, дней'] >= 7]

sale = tour\_weekly['Количество проданных путёвок'].sum()

print(sale)

# Выполним второе задание:

agent\_id =  agents[agents['Название'] == 'Вместе']['ID туроператора'].values[0]

agent\_sales = sales[sales['ID туроператора'] == agent\_id]

mrg\_data = pd.merge(agent\_sales, travels, on='ID тура')

city = mrg\_data.groupby('Город')['Количество проданных путёвок'].sum()

diagrama = city.plot.pie(autopct='%.2f%%', figsize=(10, 10))

# Выполним третье задание:

d = travels.plot.scatter(x = 'Продолжительность, дней', y = 'Стоимость, на 1 чел', figsize=(10, 10))

# **ПРИЛОЖЕНИЕ Д Визуализация**

**Перевозки (статистика)**

import pandas as pd

import matplotlib.pyplot as plt

data = pd.read\_csv('task19.csv', encoding='cp1251', delimiter=';')

# Построим диаграмму, показывающую суммарный расход бензина по датам по всем направлениям

petrol = data.groupby('Дата')['Расход бензина'].sum()

petrol.plot.pie(autopct='%.2f%%', figsize=(10, 10))

print(plt.title('Суммарный расход бензина по датам по всем направлениям'))

# Построим диаграмму зависимости расхода бензина от расстояния:

plt.figure(figsize=(10, 10))

plt.scatter(x=data['Расстояние'], y=data['Расход бензина'])

print(plt.title('Зависимость расхода бензина от расстояния'))

print(plt.xlabel('Расстояние'))

print(plt.ylabel('Расход бензина'))

# Теперь визиализируем диаграмму средней массы грузы по пунктам отправления:

data.groupby('Пункт отправления')['Масса груза'].mean().plot.bar()

print(plt.title('Средняя масса груза по пунктам отправления'))