М. И. Карякин
К. А. Ватульян
Р. М. Мнухин

Технологии программирования и компьютерный практикум на языке Python

учебное пособие

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное
учреждение высшего образования
«ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

М. И. Карякин
К. А. Ватульян
Р. М. Мнухин

Технологии программирования и компьютерный практикум на языке Python

Учебное пособие

Ростов-на-Дону – Таганрог
Издательство Южного федерального университета
2022

УДК 519.6+004.43+510.5
ББК 22.193+32.973
К27

*Печатается по решению кафедры теории упругости
Института математики, механики и компьютерных наук им. И.И. Воровича
Южного федерального университета (протокол № 6 от 18 апреля 2022 г.)*

Рецензенты:
заведующий кафедрой «Теоретическая и прикладная механика»
Донского государственного технического университета,
доктор физико-математических наук, доцент *А.Н. Соловьев*;

доцент кафедры теории упругости Южного федерального университета,
кандидат физико-математических наук, *О.В. Явруян*

Карякин, М.И.

К27 Технологии программирования и компьютерный практикум на языке Python : учебное пособие / М. И. Карякин, К. А. Ватульян, Р. М. Мнухин ; Южный федеральный университет. – Ростов-на-Дону ; Таганрог : Издательство Южного федерального университета, 2022. – 242 с.
ISBN 978-5-9275-4108-9

Пособие содержит теоретический материал, а также варианты индивидуальных и проектных заданий, связанных как с основными разделами языка программирования Python (функции, строки, списки и т. п.), так и с использованием распространенных библиотек научного программирования — Numpy, Matplotlib, Pandas. В качестве средства выполнения заданий предполагается использование среды Jupyter Notebook.

Предназначено для студентов бакалавриата укрупненной группы 01.03.00 «Математика и механика». Оно также может быть полезно учителям информатики для организации самостоятельной работы в старших классах средней школы.

УДК 519.6+004.43+510.5
ББК 22.193+32.973

ISBN 978-5-9275-4108-9

© Южный федеральный университет, 2022
© Карякин М. И., Ватульян К. А., Мнухин Р. М., 2022

Содержание

Предисловие **6**

1 Основы программирования на Python **8**
1.1 Базовые элементы программы 8
1.2 Переменные и значения 13
1.3 Ввод и вывод данных 18
1.4 Библиотека математических функций 20
1.5 Условный оператор 23
1.6 Цикл `while` 28
1.7 Цикл `for` 32

2 Индивидуальная работа «Функции и строки» **40**
2.1 Функции 40
2.1.1 Основные сведения и примеры 40
2.1.2 Модули 44
2.1.3 Аргументы функции: дополнительные вопросы . . . 47
2.1.4 Области видимости переменных 51
2.1.5 Рекурсия 57
2.2 Строки 63
2.2.1 Основные сведения и примеры 63
2.2.2 Escape-последовательности 67
2.2.3 f-строки 69
2.2.4 Функции и методы строк 73
2.3 Табулирование функции 76
2.3.1 Постановка задачи 76
2.3.2 Образец выполнения задания 79
2.4 Рекурсия 85
2.4.1 Постановка задачи 85
2.4.2 Образец выполнения задания 88
2.5 Работа со строками 92
2.5.1 Постановка задачи 92
2.5.2 Образец выполнения задания 95
2.6 Оформление отчета 100

3 Индивидуальная работа «Численное исследование графика функции» . . . 106
3.1 Списки и кортежи . . . 106
3.1.1 Определение и примеры списков . . . 106
3.1.2 Способы создания списков . . . 111
3.1.3 Основные методы списков . . . 113
3.1.4 Кортежи . . . 115
3.2 Основные определения . . . 117
3.3 Постановка задачи . . . 118
3.4 Образец выполнения задания . . . 123
3.5 Графики исследуемых функций . . . 129

4 Индивидуальная работа «Создание и обработка матриц стандартными средствами языка Python» . . . 140
4.1 Матрицы и их реализация . . . 140
4.2 Постановка задачи . . . 144
4.3 Образец выполнения задания . . . 151
4.4 Материалы к заданию . . . 156

5 Проектное задание «Вычисления с использованием пакета NumPy» . . . 162
5.1 Пакет NumPy . . . 162
5.1.1 Основные сведения о пакете . . . 162
5.1.2 Простая демонстрация возможностей . . . 164
5.1.3 Массивы и их создание . . . 166
5.1.4 Операции с массивами . . . 171
5.2 Постановка задачи . . . 176
5.3 Варианты заданий . . . 177
5.4 Образец выполнения задания . . . 183

6 Индивидуальная работа «Основные возможности пакета Matplotlib» . . . 191
6.1 Знакомство с Matplotlib . . . 191
6.2 Постановка задачи . . . 192
6.3 Методические указания . . . 197

7 Проектное задание «Обработка и анализ данных в Pandas» . . . 219
7.1 Знакомство с Pandas . . . 219
7.2 Данные для анализа . . . 223
7.3 Постановка задачи . . . 224
7.4 Образец выполнения проектного задания . . . 225

Список литературы . . . 236

Предисловие

Язык программирования Python в последнее время стал одним из самых распространенных и популярных языков программирования. Уже несколько лет подряд именно Python возглавляет рейтинг языков программирования, составляемый IEEE Spectrum [34]. В конце 2021 года Python переместился на первую строку популярного рейтинга Tiobe [33].

Области применения языка Python в современном мире чрезвычайно широки: на нем сегодня «говорят» специалисты в области математического моделирования, физики, астрономии, химии, биологии, биоинформатики, лингвистики, машинного обучения, анализа данных, статистики и других разделов естественных и инженерных наук. Подобная популярность неизбежно приводит к тому, что ежегодно появляется большое количество учебников и учебных пособий, посвященных изучению как языка в целом, так и его различных подразделов (библиотек, областей применения и т. д.). Работы [5, 8, 14–16] представляют лишь малую часть огромного списка публикаций, позволяющих новичкам погрузиться в увлекательный мир современного программирования.

Важным элементом изучения языка является практическая работа обучающихся. Именно в процессе самостоятельной работы можно достаточно быстро познакомиться с мощью и гибкостью Python, уверенно овладеть основными и продвинутыми навыками его использования, что позволит впоследствии успешно применять его для решения различных задач профессиональной деятельности в сфере современной прикладной математики и программирования. Настоящее учебное пособие как раз и предназначено для организации самостоятельной работы: содержащиеся в нем задачи могут быть использованы в качестве индивидуальных или групповых проектных заданий, как относящихся к «классическим» основам программирования, так и связанных с использованием основных

библиотек, разработанных для применения Python в сфере научного программирования.

Первый раздел данного пособия содержит базовые сведения о языке Python. Краткая теоретическая информация есть и в остальных разделах, посвященных описанию заданий по отдельным темам или библиотекам. Тем не менее, авторы настоятельно рекомендуют обучающимся не ограничиваться этими сведениями, а параллельно с выполнением заданий знакомиться и с более подробными материалами. В этом смысле отдельного упоминания заслуживает учебник К. Хилла [14], материал которого достаточно полно покрывает не только собственно язык, но и используемые в данном пособии библиотеки NumPy, Matplotlib, Pandas.

Общая структура разделов пособия, содержащих варианты заданий, приблизительно одинакова. Во вводном подразделе приводятся те или иные сведения о языковых структурах или библиотеках, важных для выполнения задания. Далее формулируется само задание (в случае индивидуальных работ — по 30 вариантов, для проектных заданий — порядка 10 вариантов), а затем приводится возможный образец выполнения одного из этих вариантов, во всех случаях он имеет номер «ноль». В разделе, посвященном пакету Matplotlib, вместо прямого решения одного из вариантов содержится целый ряд дополнительных вопросов и мини-заданий, анализ которых позволит обучающимся овладеть основными навыками создания научных иллюстраций и успешно выполнить любой вариант индивидуального задания.

Предполагается, что обучающиеся готовят по каждому заданию отчет, представляющий собой файл, созданный в среде Jupyter Notebook [23]. Краткие сведения о возможностях форматирования, предоставляемых используемым в этой среде языком разметки Markdown, приведены в разделе 2.6. Отметим также, что для набора математических формул в отчете должны быть использованы возможности системы компьютерной верстки LaTeX.

1 Основы программирования на Python

1.1 Базовые элементы программы

Программа на языке Python представляет собой последовательность **определений** и **команд**. Определения обрабатываются и команды выполняются в специальной среде (оболочка, shell). Они могут быть набраны прямо в оболочке или сохранены в файле, который впоследствии может быть загружен в оболочку и выполнен.

В данном пособии все примеры будут продемонстрированы средой специального вида — Jupyter Notebook. Она позволяет выполнять и сохранять не только тексты программ, но и блоки отформатированного текста, формулы, изображения, поэтому очень удобна как для интерактивных презентаций, так и для представления отчетов о выполненных работах. В то же время эта среда полностью сохраняет интерактивность, присущую базовой оболочке IDLE, входящей в состав дистрибутива Python. Если в ячейке ввода команд (она помечена символами `In []:`) ввести какое-то арифметическое выражение и «выполнить» его, нажав кнопку ▶ Run на панели задач или комбинацию клавиш Shift + Enter на клавиатуре, то под ячейкой будет сразу отображен результат введенной операции:

```
In [1]: 5 + 3

Out[1]: 8
```

Цифра внутри квадратных скобок — это счетчик операций, он увеличивается на единицу каждый раз после выполнения ячейки с командой или набором команд (кодом программы). **Команда** (или **оператор**) — инструкция интерпретатору осуществить то или иное действие.

Суть работы программы — манипулирование с данными, обработка **объектов**, содержащих данные. Каждый такой объект имеет важную

характеристику — **тип**, определяющий, что за данные он содержит, и какие операции к нему применимы.

Основная классификация объектов:

- *Скалярные* (их нельзя разделить на составные части).
- *Нескалярные* (составные, имеющие внутреннюю структуру, к которой можно получить доступ). Например, строка состоит из нескольких символов, а комплексное число — из вещественной и мнимой части.

Скалярные объекты Python:

- `int` — целые числа (`5, -23, 1234567890`).
- `float` — действительные числа (`3.14, 9.81, 1.2e-5`).
- `bool` — используется для представления двух логических значений: `True` (истина) и `False` (ложь).
- `NoneType` — у этого типа данных есть только одно значение, константа `None`. Эту константу обычно используют для обозначения того, что конкретное значение отсутствует.

Сразу отметим, что объекты целого типа языка Python могут хранить значения с очень большим количеством значащих цифр — оно ограничено только размерами оперативной памяти компьютера. Вычислим, для примера значение выражения 2^{150}:

```
In [2]: 2**150
```

```
Out[2]: 1427247692705959881058285969449495136382746624
```

Встроенная функция `type` служит для определения типа объекта.

```
In [3]: type(3)
```

```
Out[3]: int
```

```
In [4]: type(3.14)
```

```
Out[4]: float
```

Выражения.

С помощью **знаков операций** объекты объединяются в **выражения**, значение каждого из которых представляет собой объект определенного типа. Синтаксис простейшего выражения:

<объект> <знак операции> <объект>

Арифметические операции с числовыми объектами.

`a + b` — сумма. Если оба числа — целые, то результат — объект `int`, в остальных случаях результат имеет тип `float`.

`a - b` — разность.

`a * b` — произведение.

`a / b` — частное. В Python 3 тип у частного всегда `float`, независимо от типов входящих в него объектов.

`a // b` — результат деления нацело.

`a % b` — остаток от деления.

В двух предыдущих операциях, как правило, `a` и `b` имеют тип `int` и результат — тоже `int`. Нужно помнить, однако, что эти операции применимы и к вещественным числам.

`a**b` — возведение в степень, a^b. Если оба числа — целые, то результат — объект `int`, в остальных случаях — `float`.

Порядок действий аналогичен принятому в математике: сначала вычисляются выражения в скобках, затем выполняется операция возведения в степень `**`. Следующий приоритет имеют операции умножения и деления `*, /, //, %`. Их приоритет одинаков, они выполняются слева направо. Завершают вычисления операции сложения и вычитания `+, -`, которые тоже имеют одинаковый приоритет.

Правило выполнения слева направо операций одинакового приоритета является общим для языка Python за одним исключением: операции возведения в степень выполняются справа налево. Поэтому выражение `2**3**2` имеет смысл 2^{3^2}, то есть 2^9.

Операции сравнения.

Результатом операций сравнения

`a > b` (больше),

`a >= b` (больше или равно),

`a < b` (меньше),

`a <= b` (меньше или равно),

`a == b` (равно),

`a != b` (не равно)

является одно из двух значений — истина или ложь, `True` или `False`.

```
In [5]: 2022**2023 > 2023**2022
```

```
Out[5]: True
```

Операции с объектами логического типа.

`a and b` (результат `True`, если оба аргумента `True`),

`a or b` (результат `True`, если хотя бы один аргумент `True`),

`not a` (противоположное значение: если значением `a` является `True`, то результат — `False`, и наоборот).

Приоритет операций сравнения и логических операций меньше, чем у арифметических операций, перечисленных выше. Относительный их приоритет таков: сначала выполняются операции сравнения (все они имеют одинаковый приоритет), потом `not`, далее `and` и, наконец, `or`.

Игнорирование приоритетов может привести к неожиданным ошибкам. Так, например, выражение `True == not False` является ошибочным, правильная его запись такова: `True == (not False)`.

Преобразование типов (type casting).

На основе объекта одного типа может быть создан объект другого типа. Для преобразования используется имя типа следующим образом:

```
In [6]: float(100)
```

```
Out[6]: 100.0
```

```
In [7]: int(3.9)
```

```
Out[7]: 3
```

```
In [8]: bool(3.9)
```

```
Out[8]: True
```

```
In [9]: bool(0.0)
```

```
Out[9]: False
```

Последние примеры демонстрируют важную особенность приведения данных к логическому типу: все, что не равно нулю, сводится при таком преобразовании к значению `True`.

Нескалярные объекты.

Важным примером составного объекта является строка: набор символов, заключенный в кавычки. В языке Python возможны три варианта оформления строк: одинарные кавычки, двойные кавычки (их смысл одинаков, но смешивать — начинать строку одинарной, а заканчивать двойной — нельзя) и тройные кавычки (можно использовать и три одинарных, и три двойных), которые позволяют выделять т. н. многострочные строки. Примеры строк и простых операций с ними приведены ниже, более подробно строки будут рассмотрены в разделе 2.2.

```
In [10]: 'abc'
         "abc"
         '''Привет!
         В этой строке -
         сразу три строчки'''
```

```
Out[10]: 'Привет!\nВ этой строке - \nсразу три строчки'
```

Заметим, что для хранения информации о разрыве строки используется комбинация символов `\n`.

```
In [11]: type('123')
```

```
Out[11]: str
```

```
In [12]: 'мех' + 'мат'
```

```
Out[12]: 'мехмат'
```

```
In [13]: 'A'*3
```

```
Out[13]: 'AAA'
```

```
In [14]: str(100)
```

```
Out[14]: '100'
```

```
In [15]: int('100')
```

```
Out[15]: 100
```

1.2 Переменные и значения

Переменная в языке Python — это символическое имя, которое является ссылкой или указателем на тот или иной объект. После связывания переменной с объектом, которое часто называется *присваиванием* переменной значения, мы можем ссылаться на объект, используя имя переменной. Но надо помнить при этом, что данные по-прежнему принадлежат самому объекту.

В языке Python нет специальной команды объявления переменной. Переменная здесь создается в момент связывания с объектом. Тип переменной определяется типом этого объекта.

Рассмотрим следующий пример.

In [16]:
```
pi = 3.14159
radius = 11.2
area = pi*(radius**2)
area
```

Out[16]: 394.0810495999999

В первой строке создается объект вещественного типа `3.14159` и с ним связывается переменная `pi`. Операция этого связывания, обозначаемая в Python знаком равенства, называется *присваиванием*. Заметим здесь, что операция проверки на равенство двух величин обозначается двумя символами == как раз с целью отличить ее от присваивания.

Затем (вторая строка) в памяти создается объект `11.2`, с которым связывается переменная `radius`. И, наконец, в третьей строке создается еще один объект как результат вычислений, в которых вместо самих объектов — приближенного значения числа π и радиуса окружности — используются связанные с ними переменные. Результат вычислений получает имя `area`, позволяющее программе и дальше получать доступ к этому объекту, например, для вывода его значения в четвертой строке. Схема размещения объектов в памяти и связывания с ними переменных изображена на рис. 1.1.

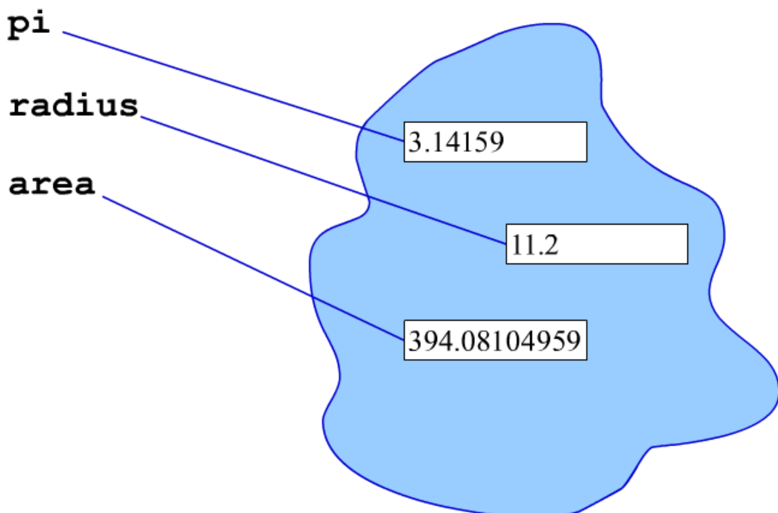


Рисунок 1.1 — Связывание переменных с объектами в памяти

Важно отметить, что имя, или переменная, получает тип в момент

присваивания выражению в правой части. Этот процесс называется *динамической типизацией*. В дальнейшем в ходе выполнения программы переменная может быть связана с объектом другого типа; ее тип при этом поменяется автоматически. Во многих других языках программирования тип переменной определяется в программе в момент ее описания и не может быть изменен. Область памяти для хранения значения такой переменной тоже выделяется в момент ее описания (с некоторыми оговорками). Это — *статическая типизация*.

Что произойдет, если мы изменим значение одной из переменных?

In [17]:
```
radius = 14.3
area
```

Out[17]: 394.0810495999999

Как и ожидалось, изменение значения переменной `radius` не приводит к изменению значения других переменных. Обратите внимание (см. рис. 1.2) на оставшееся в памяти неиспользуемое значение `11.2`. К этому объекту уже нельзя получить доступ в программе, поскольку с ним не связана ни одна переменная; он «засоряет память», является «мусором». Очисткой памяти занимается специальная программа — сборщик мусора (garbage collector).

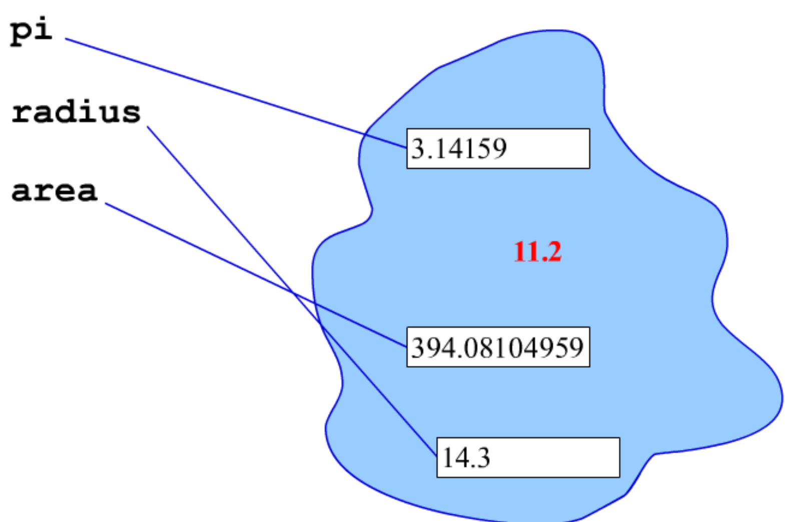


Рисунок 1.2 — Возникновение «мусора» в памяти

Существует ряд правил о том, как можно и как нельзя называть переменные. Прежде всего отметим строгие требования языка:

- Имя переменной может начинаться только с буквы или символа подчеркивания.
- Имя переменной может содержать только буквы, цифры и знак подчеркивания. Эти буквы могут быть из разных алфавитов, но хороший стиль подразумевает использование только латинских букв.
- Имя переменной не может совпадать с зарезервированными (служебными) словами языка Python:

```
False     None     True   and   as
assert    async    await  break class
continue  def      del    elif  else
except    finally  for    from  global
if        import   in     is    lambda
nonlocal  not      or     pass  raise
return    try      while  with  yield
```

Отметим, наконец, что регистр символов в имени переменной имеет значение, например, `Start` и `start` — это две разные переменные.

Правила хорошего тона, связанные с именованием переменных, содержатся в PEP8 [35] — важном документе, определяющем общие правила оформления кода программ, общепринятые сообществом Python-разработчиков. Его перевод на русский язык можно посмотреть, например, в [28].

Приведем здесь только некоторые правила, связанные с именованием переменных:

- Имена переменных, функций и методов должны следовать формату `lower_case_with_underscores` (нижний регистр букв, разделение слов символами подчеркивания).
- Константы должны именоваться по формату `ALL_CAPS` (все буквы прописные, в качестве разделителя используется символ подчерки-

вания).

- Имена классов оформляются по формату `CapitalizeWords` или, что эквивалентно, `StudlyCaps` (каждое слово начинается с прописной буквы).

В крайнем случае (для совместимости с уже имеющимся кодом) допустимо использование стиля `mixedCase` (он же `camelCase`).

Никогда не используйте символы:

- l (маленькая латинская буква «эль»),
- O (заглавная латинская буква «о»),
- I (заглавная латинская буква «ай»)

как однобуквенные идентификаторы. В некоторых шрифтах эти символы неотличимы от единицы и нуля.

Если очень нужно использовать имя переменной из одной буквы l, пишите вместо неё заглавную L.

Виды присваивания.

Обычная (каноническая) форма оператора присваивания связывает одну переменную с одним объектом:

```
variable = some_object
```

Еще один вариант — так называемое *множественное* или *позиционное* присваивание, когда одним оператором можно связать с объектами сразу несколько переменных. Выглядит это, например, так:

```
var1, var2, var3 = obj1, obj2, obj3
```

Выражения в правой части вычисляются до начала выполнения присваивания; затем присваивание производится слева направо. Каждому имени в левой части оператора присваивания соответствует объект (значение) в правой части оператора присваивания, который имеет ту же позицию. Разумеется, количество объектов и количество имен переменных должно совпадать.

Позиционное присваивание позволяет просто и эффективно организовать обмен значений двух переменных:

```
a, b = b, a
```

Еще один вариант присваивания, поддерживаемый языком Python, называется *каскадным* или *групповым* присваиванием вида

```
var1 = var2 = var3 = value
```

Присваивание идет каскадом справа налево; в результате значения всех переменных станут одинаковыми и равными `value`.

Завершает раздел *составное* присваивание, которое объединяет в одно действие присваивание и какую-либо бинарную операцию:

- `a += b` означает `a = a + b`,
- `a *= b` означает `a = a * b`,
- `a **= b` означает `a = a**b`

и так далее.

1.3 Ввод и вывод данных

Любая оболочка (IDLE, Jupyter Notebook и др.) позволяет выполнять отдельные команды. Однако, **программой** принято называть текст на языке программирования (в нашем случае — на языке Python), сохраненный в файле. В виде исключения в дальнейшем программой часто будем называть и код, содержащийся в одной ячейке среды Jupyter Notebook. Обычно такой код должен удовлетворять дополнительным правилам: в нем должно быть несколько действий, в том числе получение каких-то данных от пользователя и вывод результатов расчетов.

Как мы уже видели, в случае выполнения в оболочке одной команды результат ее работы сразу выводится на экран. В случае программы, т. е. последовательности команд, требуется специальный оператор — оператор вывода. В языке Python для этих целей используется функция `print`, вызов которой в общем случае имеет вид

```
print(a, b, ..., end='\n', sep=' ')
```

Здесь a, b и т. д. — переменные, значения которых должны быть выведены. Параметр end используется для указания на то, чем закончится выводимая строка данных: по умолчанию «курсор» будет перенесен на следующую строчку. Параметр sep определяет, каким символом будут разделены значения нескольких переменных при выводе; по умолчанию они отделяются друг от друга пробелами.

In [18]:
```
# Примеры
print('hello')
print(156*12)
print("I'll", "be", "back!")
print('''Привет!
В этой строке -
сразу три строчки.''')
```

```
hello
1872
I'll be back!
Привет!
В этой строке -
сразу три строчки.
```

При использовании оператора print после ячейки с кодом отсутствует блок Out [].

Обратите внимание на первую строчку в ячейке. Символ # (решетка) используется для добавления комментариев в текст программы. И сам символ, и весь текст после него до конца строки полностью игнорируется интерпретатором языка Python. Комментарии предназначены не для компьютера, а для человека: они позволяют сделать текст программы более понятным.

Простейший способ взаимодействия программы с пользователем — это запрос ввода каких-то данных. Такой ввод осуществляется в Python с помощью функции input(). Необязательным входным параметром этой

функции является строка — комментарий, появляющийся на экране ввода и уточняющий пользователю, какие именно данные от него ждет программа. Результат работы функции input — введенные пользователем данные — имеет тип str, поэтому для получения числовых значений необходимо осуществить преобразование типа.

Ниже приведен пример выполнения ячейки, содержащей три оператора ввода. Два первых уже выполнены и их результат присутствует на экране; программа ждет от пользователя ввода последних данных. Ячейка с кодом еще не выполнена, поэтому вместо номера изображен символ * (звездочка):

```
In [*]: name = input('Как Вас зовут?  ')
        age = int(input('Сколько Вам лет? '))
        weight = float(input('Каков Ваш вес в кг? '))
```

```
Как Вас зовут?  Александр
Сколько Вам лет? 22
Каков Ваш вес в кг?
```

1.4 Библиотека математических функций

Выше уже были приведены примеры нескольких полезных функций языка Python: input, print, type. Функциями являются также операции преобразования типа int, float, bool. В базовый состав языка входит еще много полезных функций, знакомство с которыми произойдет в следующих частях этого пособия. Здесь упомянем только две: функцию abs(x), позволяющую вычислять модуль (абсолютное значение) своего аргумента x, и функцию round(x,n), значением которой является величина аргумента x, округленная с точностью n знаков после запятой (если параметр n не указан, то x округляется до целого числа).

Стандартные математические функции ($\sin x$, $\cos x$, e^x и др.) в базовый состав Python не входят, однако они входят в «комплект поставки»

этого языка — в виде отдельной библиотеки, которая называется `math`. Для использования входящих в ее состав функций, библиотеку следует «подключить», или импортировать. Возможны следующие варианты такого подключения:

- `import math`

  При таком варианте для использования функций их имя следует дополнить префиксом `math.`, например, `math.sin` или `math.cos`.

- `import math as m`

  При таком варианте вместо префикса `math.` нужно использовать префикс `m.`, например, `m.sin`. В случае библиотеки `math` так обычно не поступают, однако для библиотек с длинными именами такое использование синонимов встречается очень часто и иногда даже считается стандартным.

- `from math import *`

  При таком варианте подключения доступно использование всех функций библиотеки `math`, при этом префикс совсем не используется. С одной стороны это удобно, потому что делает код более компактным, с другой стороны может привести к проблемам, особенно если программа использует сразу несколько библиотек: имена функций библиотеки, подключаемой второй, могут «стереть», т.е. переопределить, функции, реализованные в первой. Такое же переопределение могут осуществить и переменные, неудачно введенные программистом так, что их имена совпадают с библиотечными функциями.

- `from math import sin, cos, exp`

  Из библиотеки импортируется только несколько функций, требуемых для работы программы. Их имена в этом случае используются без префикса.

Формально не запрещен и такой импорт:

`from math import sin as s`,

позволяющий вместо `sin(x)` писать `s(x)`, вот только делать так категорически не рекомендуется!

Для получения полного списка всех функций и констант, доступных в библиотеке, служит функция `dir`. Используем ее, чтобы оценить разнообразие возможностей, предоставляемых библиотекой `math`:

```
In [19]: print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__',
'__spec__', 'acos', 'acosh', 'asin', 'asinh', 'atan',
'atan2', 'atanh', 'ceil', 'comb', 'copysign', 'cos',
'cosh', 'degrees', 'dist', 'e', 'erf', 'erfc', 'exp',
'expm1', 'fabs', 'factorial', 'floor', 'fmod', 'frexp',
'fsum', 'gamma', 'gcd', 'hypot', 'inf', 'isclose',
'isfinite', 'isinf', 'isnan', 'isqrt', 'ldexp', 'lgamma',
'log', 'log10', 'log1p', 'log2', 'modf', 'nan',
'perm', 'pi', 'pow', 'prod', 'radians', 'remainder',
'sin', 'sinh', 'sqrt', 'tan', 'tanh', 'tau', 'trunc']
```

Функции и переменные, наименования которых начинаются и заканчиваются двойным подчеркиванием, являются системными. Чтобы получить краткое описание остальных, используется функция `help`:

```
In [20]: help(math.factorial)
```

```
Help on built-in function factorial in module math:
factorial(x, /)
    Find x!.
    Raise a ValueError if x is negative or non-integral.
```

Если в качестве аргумента функции `help` использовать имя библиотеки, т. е. написать `help(math)`, то получим описание сразу всех констант и функций. Для рассматриваемой библиотеки это описание занимает 265 строк.

В завершение приведем два способа вычисления значения выражения

$$\sqrt{\ln\left(1+\sin^2\frac{4\pi}{7}\right)}$$

In [21]:
```
# вариант 1
import math
math.sqrt(math.log(1+math.sin(4*math.pi/7)**2))
```

Out[21]: 0.8173602445616616

In [22]:
```
# вариант 2
from math import sin, log, sqrt, pi
sqrt(log(1+sin(4*pi/7)**2))
```

Out[22]: 0.8173602445616616

1.5 Условный оператор

До сих пор написанные нами блоки кода имели **линейную структуру**: операторы выполнялись последовательно, один за другим. Многие алгоритмы решения простейших задач выглядят именно так. Рассмотрим, например, задачу **Begin14** из сборника М.Э. Абрамяна [2]:

Дана длина L окружности. Найти ее радиус R и площадь S круга, ограниченного этой окружностью, учитывая, что $L = 2\pi R$, $S = \pi R^2$. В качестве значения π использовать 3.14.

Блок-схема алгоритма ее решения приведена на рис. 1.3, ей соответствует следующий код на языке Python:

```
1 pi = 3.14
2 L = float(input("Введите длину окружности: "))
3 R = L/(2*pi)
4 S = pi*R**2
```

```
5  print(R)
6  print(S)
```

Но жизнь, к сожалению, не такая линейная и все время предлагает задачи выбора, как знаменитому витязю В. Васнецова (см. рис. 1.4).

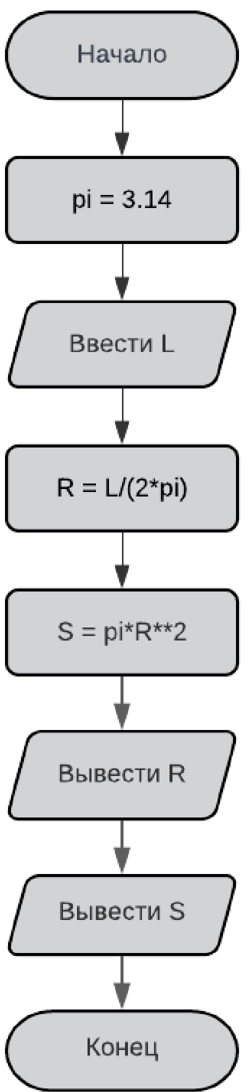


Рисунок 1.3 — Алгоритм линейной структуры

Рисунок 1.4 — Для выбора вариантов служит условный оператор

Условный оператор `if-elif-else` (его иногда называют *оператором ветвления*) — это основной инструмент выбора в Python. Он позволяет определить, какое действие следует выполнить программе в зависимости от значения переменных в момент проверки условия.

Рассмотрим простую задачу, демонстрирующую его использование: если число положительно, нужно извлечь из него квадратный корень и вывести на экран полученный результат. Считая, что библиотека `math` уже импортирована, получаем такой код:

```
1  if a>=0:
2  ␣␣␣␣print('число положительно')
3  ␣␣␣␣b = math.sqrt(a)
4  ␣␣␣␣print('квадратный корень =', b)
```

Обратите внимание на двоеточие после условия и пробелы, которые в приведенном коде для наглядности обозначены символом ␣ («корытце»). Все действия (операторы), которые должны быть выполнены в случае истинности проверяемого условия, располагаются с одинаковым отступом. Этот отступ, согласно общепринятым правилам, устанавливается равным четырем пробелам.

Ниже приведены еще две формы условного оператора, уже без выделения пробелов.

```
if a>=0:
    print('число положительно')
    b = math.sqrt(a)
    print('квадратный корень =', b)
else:
    print('число отрицательно')
    print('корень извлечь не удается')
```

```
if a>0:
    print('число положительно')
    b = math.sqrt(a)
    print('квадратный корень =', b)
elif a==0:
    print('число равно нулю')
    print('квадратный корень = 0')
else:
    print('число отрицательно')
    print('корень извлечь не удается')
```

Выделение блоков кода отступами — это, в некотором смысле know-how языка Python, унаследованное многими более «молодыми» языками программирования.

Синтаксис, близкий к условному оператору (только без отступов), используется еще в одном операторе присваивания, *тернарном операторе*, или *условном выражении*. В этом случае переменная получает одно из двух значений, в зависимости от истинности или ложности проверяемого условия. Вот как, например, тернарный оператор может быть использован для вычисления модуля числа без использования функции `abs`:

```
In [1]: x = -5
        x_abs = x if x>=0 else -x
        x_abs

Out[1]: 5
```

В завершение данного раздела в качестве примера приведем решение еще одной задачи из сборника [2].

If4. *Даны три целых числа. Найти количество положительных чисел в исходном наборе.*

Несмотря на очень короткое условие, задача может быть решена, как минимум, тремя разными способами. Возможные варианты решения приведены ниже. Подумайте, имеет ли какой-то из них заметное преимущество?

```
1  a = int(input("a: "))
2  b = int(input("b: "))
3  c = int(input("c: "))
4  # --------------------- Решение 1
5  if a>0 and b>0 and c>0:
6      print(3)
7  elif a>0 and b>0 or b>0 and c>0 or a>0 and c>0:
8      print(2)
9  elif a>0 or b>0 or c>0:
10     print(1)
11 else:
12     print(0)
13 # --------------------- Решение 2
14 k = 0
15 if a>0:
16     k += 1
17 if b>0:
18     k += 1
19 if c>0:
20     k += 1
21 print(k)
22 # --------------------- Решение 3
23 print(int(a>0) + int(b>0) + int(c>0))
```

1.6 Цикл while

Начнем с рассмотрения еще одной задачи из [2].

Integer8. *Дано двузначное число. Вывести число, полученное при перестановке цифр исходного числа.*

Схема ее решения проста: в двузначном числе количество единиц находится как остаток от деления заданного числа на 10, а количество десятков — как результат его деления на 10 нацело. Соответственно, чтобы переставить цифры, необходимо количество единиц умножить на 10 и прибавить количество десятков в исходном числе:

```
1  n = int(input("двузначное число: "))
2  e = n%10
3  d = n//10
4  print(10*e + d)
```

Немного подумав и удлинив код, можно записать решение и для трехзначных чисел. И даже для четырехзначных. А для 25-значных? А как быть в случае, если количество цифр заранее неизвестно?

На помощь приходят **циклы**. Они дают возможность выполнить одну и ту же последовательность действий много раз. Как много? Столько, столько требуется (если, конечно, программа написана правильно).

Цикл while («пока») позволяет выполнять последовательность одинаковых действий, пока проверяемое условие истинно. Его блок-схема изображена на рис. 1.5, а синтаксис выглядит так:

```
1  # предыдущий код
2  while условие:
3      действие 1      #
4      действие 2      #  тело цикла
5      ...             #
6      действие N      #
7  # код после цикла
```

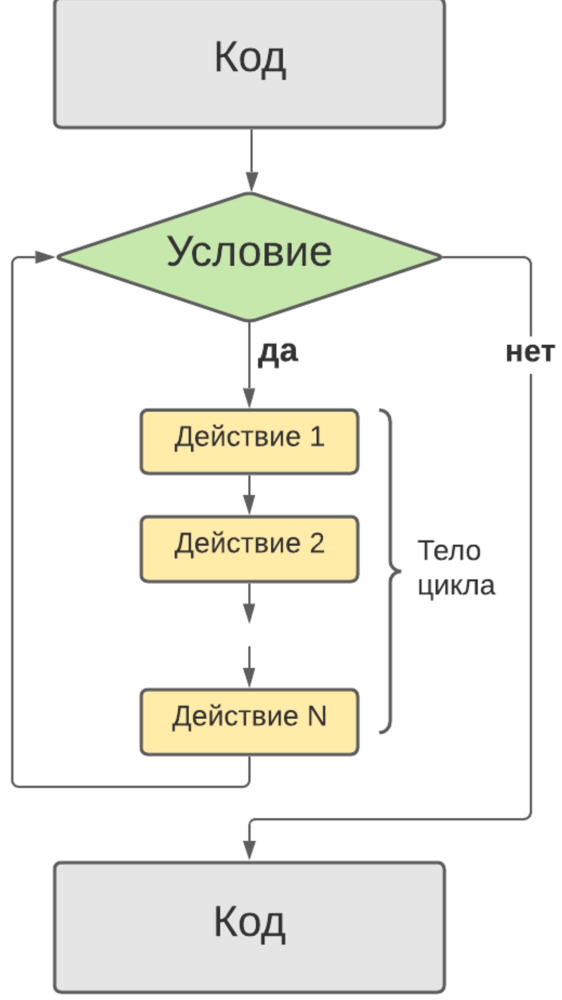


Рисунок 1.5 — Схема цикла `while`

При выполнении цикла `while` сначала проверяется истинность или ложность условия. Если условие истинно, то выполняется цепочка действий (*тело цикла*, или *одна итерация*), после чего условие проверяется снова и снова выполняется эта цепочка. Так продолжается до тех пор, пока условие будет истинно. Как только условие станет ложно, работа цикла завершится и управление передастся следующей инструкции после цикла.

Если условие ложно в самом начале, то цикл не начинается, управление сразу передается на следующую команду после тела цикла `while`.

Здесь, как и в случае условного оператора, группа или блок операторов, образующих тело цикла, выделяется одинаковым количеством про-

белов.

В качестве первого примера использования оператора `while` приведем решение усложненного варианта задачи **Integer8**:

Дано натуральное число. Вывести число, полученное при перестановке цифр исходного числа.

Для решения используется та же идея: в переменную `d` записываем последнюю цифру заданного числа `n`, которая находится как остаток от деления этого числа на 10. Затем эта цифра «стирается» из старого числа, поскольку операция деления нацело на 10 приводит к удалению из числа его последней цифры, и переносится в новое число `k`. На каждом шаге цикла перед добавлением очередной цифры число `k` умножается на 10. Таким образом, чем раньше цифра попадет в это число, тем на более старшую позицию в результате переместится. Работа программы продолжается, пока в числе есть цифры, т. е. это число больше нуля.

```
1  n = int(input("натуральное число: "))
2  k = 0
3  while n>0:
4      d = n%10
5      n //= 10
6      k = 10*k + d
7  print(k)
```

Еще один пример покажет использование оператора `while` для решения т.н. задач о последовательностях, когда количество входных данных заранее неизвестно. Данная задача позаимствована с сайта [10].

Стандартное отклонение.

Дана последовательность натуральных чисел $x_1, x_2, \ldots, x_n$. *Стандартным отклонением называется величина*

$$\sigma = \sqrt{\frac{(x_1 - s)^2 + (x_2 - s)^2 + \ldots + (x_n - s)^2}{n - 1}},$$

где $s = \dfrac{x_1 + x_2 + \ldots + x_n}{n}$ *— среднее арифметическое последовательности.*

Определите стандартное отклонение для данной последовательности натуральных чисел, завершающейся числом 0.

Комментарий к решению. На первый взгляд, задачу нельзя решить, не сохраняя все значения переменных x_k, потому что для вычисления, например, $(x_1 - s)$ нужно знать s, т. е. сначала нужно ввести все числа, найти их среднее арифметическое, а потом вернуться к x_1.

Решить задачу помогает школьная алгебра и знание формулы

$$(a-b)^2 = a^2 - 2ab + b^2.$$

Действительно,

$$(x_1 - s)^2 + (x_2 - s)^2 + \ldots + (x_n - s)^2 =$$

$$= x_1^2 - 2x_1 s + s^2 + x_2^2 - 2x_2 s + s^2 + \ldots x_n^2 - 2x_n s + s^2 =$$

$$= \sum_{i=1}^{n} x_i^2 - 2\sum_{i=1}^{n} x_i s + ns^2 =$$

$$= \sum_{i=1}^{n} x_i^2 - \frac{2}{n}\left(\sum_{i=1}^{n} x_i\right)^2 + \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2 =$$

$$= \sum_{i=1}^{n} x_i^2 - \frac{1}{n}\left(\sum_{i=1}^{n} x_i\right)^2.$$

Таким образом, в процессе ввода необходимо накапливать значение суммы вводимых чисел и суммы их квадратов.

```
1  # Решение
2  summ = 0
3  summ2 = 0
```

```
n = 0
x = int(input('x: '))
while x != 0:
    summ += x
    summ2 += x**2
    n += 1
    x = int(input('x: '))
sigma = ((summ2-summ**2/n)/(n-1))**0.5
print(sigma)
```

1.7 Цикл for

Цикл for, также называемый циклом с параметром, позволяет выполнить заранее известное количество итераций. Параметр цикла или *счетчик* — это переменная, указываемая в заголовке цикла for. В нем также задается множество (диапазон) значений, по которому будет пробегать эта переменная. Синтаксис схематически выглядит так:

```
# предыдущий код
for параметр_цикла in множество_значений:
    действие 1
    действие 2
    ...
    действие N
# код после цикла
```

Множество значений может быть задано, например, списком, строкой или диапазоном. Начнем с последнего — очень распространенного и полезного объекта range().

Функция range позволяет сгенерировать ряд чисел — арифметическую прогрессию. Строго говоря, результат ее работы — это не сам ряд чисел, а так называемый *итерируемый* объект, который при каждом обращении к нему возвращает очередной элемент прогрессии. Именно это

и нужно для организации цикла с известным числом повторений.

Аргументами функции `range` могут быть только целые объекты или переменные. В зависимости от того, с каким количеством аргументов вызывается функция `range`, ряд чисел, возвращаемых ее значением, строится по-разному:

- `range(start, finish, step)`
  В этом случае `start` — первый элемент арифметической прогрессии, `step` — шаг, или разность прогрессии. Что касается аргумента `finish`, то им определяется последнее допустимое значение элемента прогрессии. Точнее говоря, это последнее значение должно максимально приблизиться к `finish`, но не достичь его. Таким образом, для положительных значений шага `step` последним генерируемым числом будет наибольший элемент последовательности, который все-таки меньше, чем `finish`. Параметр `step` может быть и отрицательным, тогда последним генерируемым числом будет минимальный элемент прогрессии, больший чем `finish`.
- `range(start, finish)`
  Смысл параметров `start` и `finish` сохраняется, а шаг в этом случае считается равным единице.
- `range(finish)`
  В этом случае шаг равен единице, а стартовое значение полагается равным нулю.

Принцип «последнее число не включается в диапазон» используется не только в функции `range`. Аналогично устроены, например, срезы строк и списков, о которых речь пойдет в 2.2. Естественность этого принципа связана с тем, что при таком подходе объект `range(n)` сгенерирует ровно `n` значений, поскольку счет в языке Python начинается с нуля.

Рассмотрим несколько примеров.

```
In [2]: for i in  range(10):
            print(i, end=' ')
```

```
0 1 2 3 4 5 6 7 8 9
```

```
In [3]: for i in range(0,10,2):
            print(i, end=' ')
```

```
0 2 4 6 8
```

```
In [4]: for i in range(10,0,-1):
            print(i, end=' ')
```

```
10 9 8 7 6 5 4 3 2 1
```

Возможны случаи, когда функция `range` вообще не сгенерирует никакой последовательности, например, когда шаг прогрессии положителен, а параметр `finish` меньше или равен параметру `start`. Обратите внимание: такой вызов не приведет к ошибке, просто результат вызова функции не будет генерировать числа. Если такое выражение будет использовано в качестве множества значений для цикла `for`, то тело цикла не выполнится ни разу.

```
In [5]: for i in range(10,0):
            print(i, end=' ')
```

В данном случае вывод отсутствует.

Синтаксис оператора цикла, точнее ключевое слово `in` в его схеме, наводит на мысль, что с помощью функции `range` можно проверить принадлежность того или иного числа заданному диапазону. Это действительно так:

```
In [6]: 500 in range(1000)
```

```
Out[6]: True
```

```
In [7]: 20 in range(1,25,2)
```

```
Out[7]: False
```

Теперь приведем примеры других способов задания множества значений, которые пробегает параметр цикла. Очень распространенным является использование в операторе цикла списка или кортежа:

```
In [8]: print('Цвета российского флага:')
        i = 1
        for color in 'белый', 'синий', 'красный':
            print('Цвет ', i, ' - ', color)
            i += 1
```

```
Цвета российского флага:
Цвет  1  -  белый
Цвет  2  -  синий
Цвет  3  -  красный
```

В предыдущем примере, чтобы в цикле выводить не только название цвета, но и его номер, пришлось ввести дополнительную переменную, счетчик `i`, и на каждом шаге цикла увеличивать ее значение. Чтобы избежать этого в языке Python используется функция `enumerate`, которую можно применить к любому множеству значений оператора цикла, точнее, к любому итерируемому объекту, и в качестве результата получить набор пар вида *(номер очередного генерируемого значения, само значение)*.

Следующий пример демонстрирует с одной стороны, работу функции `enumerate`, а с другой — возможность использования строки символов в качестве итерируемого объекта в операторе цикла.

```
In [9]: for i, letter in enumerate('ПРИВЕТ'):
            print(i+1,'-я буква: ', letter, sep='')
```

```
1-я буква: П
2-я буква: Р
```

```
3-я буква: И
4-я буква: В
5-я буква: Е
6-я буква: Т
```

Обратим внимание, что в предыдущих примерах несколько громоздким и неочевидным является вызов функции `print`, когда на экран нужно вывести сразу несколько значений, текстовых и числовых. В современном языке Python, начиная с версии 3.6, для решения этой проблемы введен очень полезный строковый объект, называемый f-строкой. Его подробное описание и примеры использования будут даны в 2.2.3. Здесь отметим только, что f-строка позволяет включать имена переменных или даже выражения (в фигурных скобках), значения которых вычисляются в момент создания строки и подставляются в нее. С использованием f-строк последняя функция вывода может быть записана так:

```
print(f'{i+1}-я буква: {letter}')
```

В качестве еще одного примера использования оператора цикла `for` ответим практически на часто задаваемый вопрос: какой из двух вариантов обмена значений двух переменных `a` и `b` работает быстрее,

```
c = a
a = b
b = c
```

или

```
a, b = b, a
```

Один из возможных вариантов измерения временных интервалов — использование функции `time` из стандартной библиотеки `time`. Эта функция возвращает время в секундах, прошедшее с *начала отсчета*, в качестве которого в операционных системах Windows и UNIX используется 1 января 1970 года, 00:00:00 (UTC или Всемирное координированное время). Таким образом, чтобы узнать время, затраченное на выполнение фрагмента кода, функцию `time` нужно вызвать два раза: до фраг-

мента и после него, а затем вычислить разность полученных значений. Разумеется, эта информация носит достаточно приближенный характер, поскольку операционная система выделяет время не только на работу Python-кода, но и на реализацию других задач. Тем не менее, во многих случаях удается получить достаточно четкое представление об эффективности (или неэффективности) кода.

Для оценки временных затрат операцию обмена проведем достаточно большое количество раз, чтобы результаты измерения времени были более или менее достоверными. Именно для большого количества повторов и будет использоваться цикл `for`:

In [10]:
```
from time import time
n = 2*10**7
a = 3.1415
b = 1000000
start = time()
for i in range(n):
    c = a
    b = a
    a = c
time1 = time() - start
print(round(time1,3))
start = time()
for i in range(n):
    a, b = b, a
time2 = time() - start
print(round(time2,3))
```

```
1.328
1.007
```

Проведенные расчеты показывают, что второй способ эффективнее, так что в программах на Python рекомендуется использовать именно его. Отметим еще, что из полученных результатов не следует, что он эффективнее именно на 30%. В других условиях (другой компьютер,

другие запущенные программы и т. д.) этот процент может быть другим. А вот факт эффективности второго способа, скорее всего, подтвердят любые вычисления.

В завершение обсудим решение еще одной стандартной задачи из сборника [2].

For17. *Дано вещественное число* x *и целое число* n (> 0). *Используя один цикл (и не используя операцию возведения в степень), найти сумму*

$$1 + x + x^2 + x^3 + \ldots + x^n.$$

В качестве дополнения к задаче попробуем ответить на вопрос: а почему в таких вычислениях действительно лучше не использовать возведение в степень? Насколько «лобовое» решение, основанное на операции возведения в степень, уступает по времени?

Начнем с решения, использующего только один цикл. Чтобы решить задачу в один проход, заметим, что каждое следующее слагаемое в сумме получается путем умножения предыдущего на величину `x`. Поэтому для работы потребуются две переменные: `s`, в которой будем накапливать сумму, и `p`, в которой будем накапливать очередное слагаемое этой суммы. До цикла переменную `s` положим равной нулю, а переменную `p` — равной единице. Поэтому в теле цикла сначала будем выполнять суммирование, а потом корректировать значение очередного слагаемого по схеме:

```
s += p
p *= x
```

Что касается дополнительного вопроса, то сравним эффективность полученной программы и второго варианта кода, в котором очередное слагаемое будет вычисляться с помощью операции возведения `x` в нужную степень. В качестве результата работы выведем сумму, полученную двумя разными способами, и время, затраченное каждым из них.

Поскольку мы планируем выбирать значение `n` достаточно большим, значение переменной `x` придется ограничить диапазоном [0, 1], иначе можно получить переполнение.

In [11]:
```
x = float(input("введите x (0<x<1): "))
n = int(input("введите натуральное n: "))
# ------------------------ Правильное решение
start = time()
s = 0
p = 1
for i in range(n+1):
    s += p
    p *= x
time1 = time() - start
print(s, round(time1,3))
# ------------------------ с использованием **
start = time()
s = 0
for i in range(n+1):
    s += x**i
time2 = time() - start
print(s, round(time2,3))
```

```
введите x (0<x<1): 0.95
введите натуральное n: 20000000
19.999999999999975 2.303
19.999999999999986 3.513
```

Проведенные расчеты показали, что вариант, использующий возведение в степень, работает приблизительно в полтора раза медленнее. Что касается некоторой разницы в полученных суммах, то она демонстрирует общую проблему использования чисел типа `float`: возникающие в ходе расчетов погрешности позволяют быть уверенными не более чем в пятнадцати или шестнадцати значащих цифрах числа.

2 Индивидуальная работа «Функции и строки»

Индивидуальная работа состоит из трех заданий, условия и образцы выполнения которых приведены ниже. Но прежде, чем переходить к их представлению, приведем основные сведения об использовании функций и строк в программах на языке Python.

2.1 Функции

2.1.1 Основные сведения и примеры

Функция — это обособленный участок кода, имеющий свое имя и который можно вызывать, обратившись по этому имени. При вызове происходит выполнение команд *тела* функции.

Функции можно сравнить с небольшими программами, которые сами по себе, т. е. автономно, не исполняются, а встраиваются в обычную программу. Нередко их так и называют — *подпрограммы*. С точки зрения Python, функция — это объект, принимающий аргументы и возвращающий значение.

В предыдущих разделах уже упоминалось много *встроенных* функций, например, `input`, `abs`, `round`, `print` и др. Сейчас речь пойдет о тех функциях, которые может создавать программист.

Синтаксис описания функции таков:

```
1 def имя_функции(список_параметров):
2         '''docstring'''
3         действие 1   #
4         действие 2   # тело функции
5         ...          #
6         действие N   #
```

Многострочный объект, расположенный во второй строке приведенного кода, является (необязательным) текстовым описанием предназначения функции, смысла ее аргументов и т. п. Именно этот текст будет выведен на экран при запросе информации о вашей функции с помощью стандартной функции `help`.

Приведенный синтаксис проиллюстрируем примером функции, которая зависит от трех аргументов и в качестве результата должна возвращать их сумму:

```
1 def add(x, y, z):
2     '''
3     функция возвращает сумму
4     трех числовых аргументов
5     '''
6     result = x + y + z
7     return result
```

В приведенном примере `add` — это имя функции, `x`, `y`, `z` — аргументы функции. В данном случае функция вычисляет сумму своих аргументов.

Обратите внимание на ключевое слово `return`. Выражение, стоящее после него, и является возвращаемым значением функции. Оператор `return` является последним оператором функции, в том смысле, что если до него дошла очередь, то после его выполнения прекращается и выполнение функции, а управление передается в основную программу. Точнее, в то место, из которого произошел вызов функции.

Используем функцию `add` для решения следующей задачи.

Найти значение выражения

$$\frac{(a+b+c)^2-(a+b+5)^4}{\sqrt{x+a+2}-\sqrt[3]{x-a-b}}$$

при следующих значениях переменных:

$$a = 10, \ b = 6, \ c = 1, \ x = 24.$$

In [1]:
```
a, b, c, x = 10, 6, 1, 24
result = (add(a,b,c)**2 - add(a,b,5)**4)/
 ↪(add(x,a,2)**0.5-add(x,-a,-b)**(1/3))
print(result)
```

```
-48548.0
```

В приведенном фрагменте кода стрелкой отмечено место «искусственного» разрыва строки: этого разрыва нет в исходном коде, он добавлен только для соблюдения полей при печати.

Аргументы и параметры, или Немножко филологии.

Как известно, в математике используется следующая терминология: если $y = f(x)$, то x называют аргументом, а y — значением функции f.

В программировании терминология, связанная с функциями, немного меняется. Посмотрите на следующий пример кода.

In [2]:
```
def f(t):
    return t**5
x = 2
y = f(x)
print(f'y({x}) = {y}')
```

```
y(2)=32
```

Мы будем называть переменную `t` формальным параметром функции `f`, а переменную `x` — аргументом, или фактическим параметром функции `f`.

Функция может быть любой сложности и возвращать любые объекты (числа, логические значения, строки, списки, кортежи, и даже функции). Если в ходе работы функции оператор `return` не выполнился (например, он вообще отсутствует), то функция вернет значение `None`.

А что может быть параметром функции? Тоже любой объект, в том числе функция. Рассмотрим еще один пример с вычислением математического выражения.

Найти значение выражения

$$\frac{\ln^2 \alpha + \ln^2(\alpha + \beta) + \sin^2 \beta + \sin^2(\beta + 1)}{e^{2\alpha} + e^{2\beta} - \cos^2(\beta + 1) - \cos^2(\beta - 1)}$$

при следующих значениях переменных:

$$\alpha = 1, \ \beta = 2.$$

Для решения введем функцию add2, первым формальным параметром которой будет другая функция, f. При каждом вызове add2 формальный параметр f будет заменяться фактическим — именем одной из реально используемых функций:

In [3]:
```
from math import sin, cos, exp, log
def add2(f, a, b):
    return f(a)**2 + f(b)**2

alpha = 1
beta = 2
x = add2(log, alpha, alpha+beta
        ) + add2(sin, beta, beta+1)
y = add2(exp, alpha, beta
        ) - add2(cos, beta+1, beta-1)
print(x/y)
```

```
0.03382490409487421
```

Польза функций не только в возможности многократного вызова одного и того же кода из разных мест программы. Не менее важно, что благодаря им программа приобретает структуру. Функции разделяют ее на обособленные части, каждая из которых выполняет свою конкретную

задачу. В результате код становится более прозрачным, его гораздо проще понимать, корректировать, находить и исправлять ошибки. Каждую функцию можно отлаживать отдельно, а потом из готовых работающих кирпичиков собирать программу.

2.1.2 Модули

Написанные и проверенные функции хороши тем, что их можно использовать не только в одной программе. Но если у программиста 100 полезных функций, копировать их текст из программы в программу не очень удобно. Для решения этой проблемы служат *модули*.

Строго говоря, модуль в Python — это любой файл с программой. Однако в данном контексте представляет интерес файл, содержащий написанные (и проверенные!) функции. Подключение его к программе осуществляется командой `import`.

Вот как, например, может выглядеть модуль `geometry`, содержащий набор функций для вычисления площадей геометрических фигур:

```
# Файл geometry.py
# Файл необходимо поместить в ту же папку,
#                           что и ipynb-файл
def rectangle(a, b):
    '''
    функция вычисляет площадь прямоугольника
    ширины a и высоты b
    '''
    return a*b

def circle(r):
    '''
    функция вычисляет площадь круга
    радиуса r
    '''
    return 3.1415926535*r*r
```

```
def triangle(a, b, c):
    '''
    функция вычисляет площадь треугольника
    со сторонами a, b и c
    '''
    p = (a+b+c)/2
    return (p*(p-a)*(p-b)*(p-c))**0.5
```

Теперь этим модулем можно пользоваться, например, так:

In [4]:
```
import geometry
print(geometry.rectangle(10, 10))
print(geometry.circle(10))
print(geometry.triangle(3, 4, 5))
```

```
100
314.15926535
6.0
```

Как узнать, какие функции есть в модуле?

In [5]:
```
dir(geometry)
```

Out[5]:
```
['__builtins__',
 '__cached__',
 '__doc__',
 '__file__',
 '__loader__',
 '__name__',
 '__package__',
 '__spec__',
 'circle',
 'rectangle',
 'triangle']
```

Как узнать, что делает та или иная функция?

```
In [6]: help(geometry.triangle)
```

```
Help on function triangle in module geometry:
triangle(a, b, c)
    функция вычисляет площадь треугольника
    со сторонами a, b и c
```

Как назвать модуль?

Поскольку модуль будет импортироваться и использоваться в качестве переменной, то к его названию предъявляются общие требования к именам. Имя модуля не может начинаться с цифры и не может совпадать с одним из ключевых слов языка Python.

Не рекомендуется называть модуль так же, как какую-либо из встроенных функций.

Куда поместить модуль?

Самый простой вариант — разместить файл в одной папке с программой (или notebook-файлом), которая его вызывает. Если говорить более точно, то модуль должен находиться в той папке, в которой интерпретатор Python сможет его найти. Путь поиска модулей указан в переменной `sys.path` стандартной библиотеки `sys`. В него включены текущая директория (та самая папка с основной программой), а также директории, в которых установлен Python. Кроме того, переменную `sys.path` можно изменять вручную, что позволяет программисту разместить все свои модули в удобном для себя месте.

Можно ли использовать модуль как самостоятельную программу?

Можно. Обычно так часто делают, когда хотят проверить корректность работы всех функций модуля. Однако надо помнить, что при импортировании модуля его код выполняется полностью. Если в модуле

кроме описания функций есть текст программы, их вызывающей, то в процессе импорта эта программа полностью выполнится, что-то напечатает и т.п.

Чтобы этого избежать, нужно проверить, запущен ли скрипт как программа, или импортирован. Это можно сделать с помощью системной переменной `__name__`, которая определена в любой программе и имеет значение `'__main__'`, если скрипт запущен в качестве главной программы, и совпадает с именем модуля, если он импортирован. Например, модуль `geometry.py` может быть дополнен таким текстом:

```
# Дополнение в конец файла geometry.py
if __name__ == "__main__":
    # тестируем функции
    print(geometry.rectangle(10, 10))
    print(geometry.circle(10))
    print(geometry.triangle(3, 4, 5))
```

2.1.3 Аргументы функции: дополнительные вопросы

Ключевые и позиционные аргументы.

Начнем с примера. Пусть у нас есть функция, вычисляющая скорость по известным данным о пройденном пути и затраченном на него времени:

```
1 def velocity(distance, time):
2     return distance/time
```

Возможная проблема использования этой функции состоит в том, что глядя на фрагмент кода вида

`v = velocity(2, 50)`

трудно вспомнить или понять, где тут время, а где — расстояние.

Удобное решение этой проблемы состоит в том, что при вызове функции можно явно указывать не только значения фактических параметров, но и их имена. Это можно сделать так:

```
v = velocity(distance=2, time=50)
```

или так:

```
v = velocity(time=2, distance=50)
```

Обратите внимание, что использование имени переменной (*ключа*) позволяет изменять порядок аргументов по сравнению с тем, как были перечислены формальные параметры в описании функции.

Еще один допустимый вариант вызова вот такой:

```
v = velocity(50, time=5)
```

В этом случае у значения 50 нет имени, поэтому его соответствие аргументу будет определяться *позицией*, а значит это значение получит аргумент `distance`.

А вот такой вариант вызова приведет к ошибке:

```
In [7]: v = velocity(time=20, 5)
```

```
  File "<ipython-input-27-7b82a2c846ac>", line 1
    v = velocity(time=20, 5)
                         ^

SyntaxError: positional argument follows keyword argument
```

Полученное сообщение об ошибке означает, что позиционный аргумент не может располагаться после ключевого. Таким образом, сначала можно расположить аргументы, смысл которых определится их позицией, т.е. позиционные, а только потом (в произвольном порядке) располагать ключевые аргументы.

Еще один, очевидно, неправильный вариант:

```
In [8]: v = velocity(20, distance=5)
```

```
---------------------------------------------------------
TypeError                       Traceback (most recent call last)
<ipython-input-26-f8ee4f6193eb> in <module>
----> 1 v = velocity(20, distance=5)
TypeError: velocity() got multiple values for argument
↪'distance'
```

В этом случае значение 20 уже отнесено по позиции к аргументу `distance`, поэтому появление второго значения для этого аргумента вызывает ошибку.

Значение аргумента по умолчанию.

Предположим, мы хотим написать функцию вычисления веса объекта по известной массе этого объекта. Из школьной физики известна формула веса $P = mg$, где m — масса объекта, а g — ускорение свободного падения. Получаем такую функцию:

```
1 def p(m, g):
2     return m*g
```

Приведенная функция годится для любой планеты, главное — правильно задать величину ускорения свободного падения. Но если большинство программистов будут использовать эту функцию для вычисления веса объектов, находящихся на Земле, то хорошо бы освободить их от необходимости постоянно писать 9.81 в качестве значения параметра g. Делается это так:

```
1 def p(m, g=9.81):
2     return m*g
```

Теперь 9.81 — это значение параметра `g`, устанавливаемое «по умолчанию». Если теперь при вызове функции не указать значение аргумен-

та g, то оно будет принято равным именно 9.81. Принимая во внимание величину ускорения свободного падения на Марсе, 3.86, рассмотрим пример использования написанной функции:

```
In [9]: mass = 1000 # 1 тонна
        p_earth = p(mass) # вес одной тонны на Земле
        p_mars = p(mass, 3.86) # вес одной тонны на Марсе
        print(f'На Земле: {p_earth}, на Марсе: {p_mars}')
```

```
На Земле: 9810.0, на Марсе: 3860.0
```

Функции с произвольным количеством аргументов.

Мы уже знакомы со стандартной функцией print, количество аргументов у которой может быть любым. Этим свойством обладают и некоторые другие функции, например нахождения минимума или максимума своих аргументов:

```
In [10]: a = min(1,2,3,-1,-2,-10,20,30) # 8 аргументов
         b = max(a*a, a+20, 25)          # 3 аргумента
         print(a, b, a+b, a-b)           # 4 аргумента
```

```
-10 100 90 -110
```

Напишем функцию summ, которая вычисляет сумму всех своих аргументов, сколько бы их не было. Для этих целей вместо обычного списка формальных параметров у функции зададим только один (args), перед именем которого поставим звездочку. Разумеется, имя этого параметра можно выбрать любым, но args — это общепринятый вариант. Теперь имя параметра можно использовать в качестве имени итерируемого объекта для множества значений в операторе цикла for:

In [11]:
```
def summ(*args):
    s = 0
    for x in args:
        s += x
    return s
print(summ(1))
print(summ(1,2))
print(summ(1,3,10))
print(summ())
```

```
1
3
14
0
```

2.1.4 Области видимости переменных

Областью видимости имени переменной называется та часть программного кода, в которой к переменной с этим именем гарантируется однозначный доступ. Под переменной в этом контексте понимаются все именованные объекты языка — собственно переменные, функции, объекты и т.д. Переменную можно использовать только в пределах ее области видимости.

Переменная *видна* в том блоке, где она первый раз появилась или определена. В таблице ниже перечислены основные причины появления переменной:

Действие	**Оператор**
Присваивание	`x = value`
Подключение модулей	`import module` `from module import name`
Определение функции	`def my_func():`
Задание параметров функции	`def my_func(arg1, arg2):`
Описание класса	`class MyClass:`

Области видимости переменных в Python иллюстрирует диаграмма, изображенная на рис. 2.1.

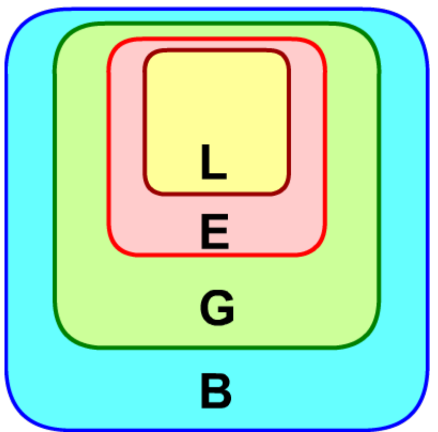


Рисунок 2.1 — Области видимости переменных: **L**ocal — локальная, **E**nclosing — объемлющая (другое название — nonlocal), **G**lobal — глобальная, **B**uilt-in — встроенная

Локальная область видимости — это блок кода (или тело) любой функции. В ней находятся имена, которые вы определяете внутри функции. Эти имена будут видны только в пределах тела функции. Локальная область видимости создается при вызове функции, а не при ее определении, поэтому у вас будет столько различных локальных областей, сколько различных вызовов функций произошло в программе. Каждый вызов приводит к созданию новой локальной области.

Объемлющая (или охватывающая, или нелокальная) область видимости — это локальная область видимости функции с точки зрения вложенной (описанной внутри нее) функции.

Глобальная область видимости — это самая верхняя область в программе или модуле. Имена этой области видны (т. е. доступны) всюду в программном коде.

Встроенная область видимости содержит такие имена как ключевые слова, функции, исключения и другие атрибуты, встроенные в Python. Имена из этой области также доступны всюду в программе.

Примеры кода ниже демонстрируют все четыре варианта областей видимости переменных:

In [1]:
```
def f():
    def g():
        x1 = 'local'
        print(x1)
    g()
f()
```

```
local
```

In [2]:
```
def f():
    x2 = 'enclosing'
    def g():
        print(x2)
    g()
f()
```

```
enclosing
```

In [3]:
```
x3 = 'global'
def f():
    def g():
        print(x3)
    g()
f()
```

```
global
```

In [4]:
```
def f():
    def g():
        x4 = min
        print(x4)
    g()
f()
```

```
<built-in function min>
```

Стоит повторить, что локальные переменные доступны только внутри тела функции, в котором они появились. Попытка их использовать «снаружи» приведет к ошибке.

Для понимания механизма устранения конфликта имен рассмотрим следующий пример. Что выведет следующий фрагмент кода, 10 или 100?

```
z = 10
def z_print():
    z = 100
    print(f'::::::> {z}')
z_print()
print(z)
```

Результатом выполнения пятой строчки этого кода, то есть результатом работы функции `z_print` будет строчка

```
::::::> 100
```

в то время как в результате выполнения функции `print` в шестой строке на экран будет выведено значение 10.

В общем случае поиск переменной идет по схеме $\mathbf{L} \to \mathbf{E} \to \mathbf{G} \to \mathbf{B}$. Внутри функции `z_print` существует локальная переменная `z`, имеющая значение 100. Поэтому именно ее выведет функция `print`, расположенная в четвертой строке. А снаружи, вне тела функции, этой локальной переменной уже нет, поиск по схеме продолжается и обнаруживает глобальную переменную `z`, равную десяти.

В завершение обсудим еще один пример. Почему первая функция здесь отработала успешно и вывела результат, а при вызове второй функции возникла ошибка?

```
In [5]: x = 100
        def x_print1():
            print(f'x плюс 1 = {x+1}')
        def x_print2():
            x = x+1
            print(f'x плюс 1 = {x}')
        x_print1()
        x_print2()
```

```
x плюс 1 = 101
```

```
---------------------------------------------------------------------------
UnboundLocalError                         Traceback (most recent call last)
<ipython-input-4-476965b0641a> in <module>
      6     print(f'x плюс 1 = {x}')
      7 x_print1()
----> 8 x_print2()

<ipython-input-4-476965b0641a> in x_print2()
      3     print(f'x плюс 1 = {x+1}')
      4 def x_print2():
----> 5     x = x+1
      6     print(f'x плюс 1 = {x}')
      7 x_print1()

UnboundLocalError: local variable 'x' referenced before␣
 ↪assignment
```

Кроме правила поиска имени **L** $\rightarrow$ **E** $\rightarrow$ **G** $\rightarrow$ **B** нужно помнить правила *появления* переменных (см. таблицу выше). Если мы что-то присваиваем переменной внутри тела функции, а именно это происходит

во второй функции, то переменная создается именно в этот момент, а значит, является локальной. Локальная переменная `x` внутри функции `x_print2` еще не имеет значения, поэтому выражение `x + 1` не может быть вычислено. В первой функции локальной переменной `x` нет, поэтому программа обращается к значению глобальной переменной.

А может ли функция изменить значение глобальной переменной?

Если по каким-то причинам функции все-таки требуется изменить значение глобальной переменной, то для доступа к ней используется описатель `global` в теле функции. С его использованием работоспособный вариант функции `x_print2` может быть записан так:

In [6]:
```
x = 100
def x_print2():
    global x
    x = x+1
    print(f'x плюс 1 = {x}')
x_print2()
print(x)
```

```
x плюс 1 = 101
101
```

Видим, что функция действительно изменила значение глобальной переменной.

Отметим, что в Python 3 существует еще и описатель `nonlocal` для доступа из внутренней функции к переменным, существующим во внешней для нее функции.

Можно ли придумать осмысленный пример, когда функция должна менять значение глобальной переменной? Рассмотрим один пример, позволяющий дать положительный ответ на этот вопрос:

In [7]:
```
def get_candy():
    global candy
    candy += 1
    print(f'Теперь у меня {candy} конфет.')

candy = 5
get_candy()
get_candy()
get_candy()
```

```
Теперь у меня 6 конфет.
Теперь у меня 7 конфет.
Теперь у меня 8 конфет.
```

Однако существует общее мнение: по возможности лучше избегать использования глобальных переменных вообще, а их изменения внутри функции — в особенности. Такое поведение функции называется *побочным эффектом* и может приводить к трудно обнаруживаемым ошибкам.

2.1.5 Рекурсия

Чтобы понять рекурсию,
нужно сначала понять рекурсию.

Мы видели, что одна функция может вызывать другую. Заметим, что функция также может вызывать и саму себя. Такой вызов называется *рекурсией*, а сама функция называется *рекурсивной*.

Проиллюстрируем это классическим примером функции вычисления факториала, то есть решим очередную задачу из сборника [2].

Recur1. *Описать рекурсивную функцию* `fact_r(n)` *целого типа, вычисляющую значение факториала*

$$n! = 1 \cdot 2 \cdot \ldots n$$

($n > 0$ — параметр целого типа). С помощью этой функции найти факториалы пяти данных целых чисел.

Прежде чем решать задачу, дадим *рекурсивное* определение факториала, т. е. выразим $n!$ через $(n-1)!$. Очевидно, что

$$n! = (n-1)! \cdot n$$

Проводя вычисления по этой цепочке, в конце концов придем к необходимости вычислить значение величины $0!$, но для нее известно значение: $0! \equiv 1$. На языке Python данная схема реализуется следующим образом:

In [8]:
```
def fact_r(n):
    if n == 0:
        return 1
    else:
        return n*fact_r(n-1)
# блок тестирования функции
nn = 10, 5, 40, 1, 8
for n in nn:
    print(f'{n}! = {fact_r(n)}')
```

```
10! = 3628800
5! = 120
40! = 815915283247897734345611269596115894272000000000
1! = 1
8! = 40320
```

Не менее знаменитая, чем факториал, рекурсия тоже может быть легко запрограммирована на Python:

```
1  def short_story():
2      print("У попа была собака, он её любил.")
3      print("Она съела кусок мяса, он её убил,")
4      print("В землю закопал,")
```

```
5	    print("Надпись написал, что")
6	    short_story()
7	
8	short_story()
```

Вот только запуская эту программу, нужно сразу понимать, что дело закончится плохо. Кстати, рекурсия без терминальной ветви имеет еще одно название — порочный круг.

Некоторые выводы:

- В описании рекурсивной функции должна присутствовать проверка условия, определяющего, по какой ветке пойдет вычисление: по *рекурсивной* — в этой ветке произойдёт вызов функцией себя самой, или *терминальной*, которая закончит вычисление и вернёт результат.
- Отсутствие терминальной ветки или ошибки в реализации условия перехода на эту ветку приведут к зацикливанию программы.
- К тестированию рекурсивных функций нужно относиться особенно ответственно. Особенно важно проверять срабатывание условия завершения рекурсии.

Визуализировать некоторые рекурсивные алгоритмы можно с помощью так называемой «черепашьей» графики: в состав стандартной библиотеки языка Python входит и библиотека `turtle`. К сожалению, она пока не поддерживается средствами Jupyter Notebook, в том смысле, что рисование происходит в отдельном окне. Для работы с упрощенным вариантом черепашьей графики и рисования прямо в среде Jupyter можно подключить специальную библиотеку. Для этого в командной среде Windows (например, PowerShell) нужно выполнить две команды:

```
pip install ipyturtle
jupyter nbextension enable -py -sys-prefix ipyturtle
```

После этого библиотека подключается обычной командой `import`, а код

создания пустого поля выглядит так

```
In [9]: from ipyturtle import Turtle
        my_t = Turtle(fixed=False)
        my_t
```

Библиотека ipyturtle содержит десять функций — команд управления черепашкой, смысл которых очевиден из названий:

back(length)

forward(length)

left(degree)

right(degree)

pendown()

penup()

pencolor(r, g, b)

showturtle()

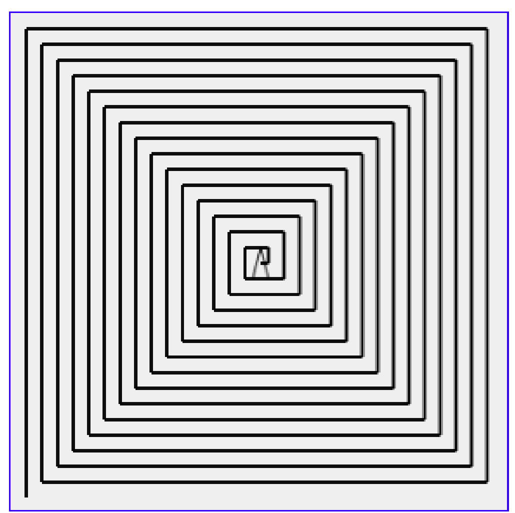
hideturtle()

reset()

В качестве примеров приведем две рекурсивные функции: рисования спирали и рисования дерева.

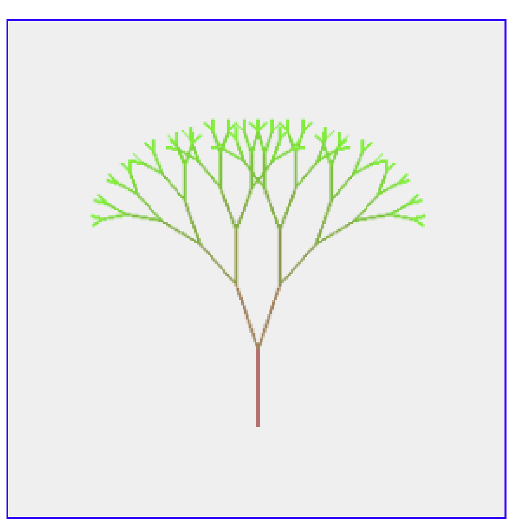
In [10]:
```
# рекурсивное рисование спирали
def draw_spiral(t, line_len):
        if line_len > 0:
            t.forward(line_len)
            t.right(90)
            draw_spiral(t,line_len-3)

# холст в исходном положении
my_t.reset()
# опускаем черепашку в левый нижний угол
my_t.penup()
my_t.back(150)
my_t.left(90)
my_t.forward(150)
my_t.right(90)
my_t.pendown()
# рисуем спираль
draw_spiral(my_t,300)
```



In [11]:
```
# рекурсивное рисование дерева
def tree(branch_len,t):
    t.pencolor(r=4*branch_len, g=255-3*branch_len,
               b=2*branch_len)
    t.pendown()
    if branch_len > 5:
        t.forward(branch_len)
        t.right(20)
        tree(branch_len-7,t)
        t.left(40)
        tree(branch_len-7,t)
        t.right(20)
        t.penup()
        t.back(branch_len)

# холст в исходном положении
my_t.reset()
# опускаем черепашку вниз
my_t.penup()
my_t.back(100)
my_t.pendown()
# рисуем дерево
tree(50,my_t)
my_t.hideturtle()
```



2.2 Строки

2.2.1 Основные сведения и примеры

Выше уже было сказано, что *строкой* называется набор символов в кавычках (одинарных, двойных или тройных). Одинарные и двойные имеют одинаковый смысл, но если строка началась с одинарной кавычки, то и заканчиваться тоже должна одинарной, и наоборот. Тройные кавычки позволяют создавать объекты, в которых содержится сразу несколько строк текста. Их основное назначение — создание строк документирования функций (`docstring`).

In [1]:
```
s1 = 'Привет!'
s2 = "John's dog"
s3 = '''
Еду. Тихо. Слышны звоны
Под копытом на снегу,
Только серые вороны
Расшумелись на лугу.
'''
print(s1)
print(s2)
print(s3)
```

```
Привет!
John's dog

Еду. Тихо. Слышны звоны
Под копытом на снегу,
Только серые вороны
Расшумелись на лугу.
```

Еще одна возможность тройных кавычек связана с созданием многострочных комментариев: строковый литерал, возникающий при их использовании, вычисляется и на какое-то время сохраняется в памяти, но содержащийся внутри него код, разумеется, не выполняется. Это позволяет, например, временно *закомментировать*, т. е. исключить из выполнения, большой фрагмент кода во время отладки программы. Разумеется, этой возможностью не следует злоупотреблять, особенно учитывая, что практически во всех средах программирования на Python есть команды быстрого комментирования и снятия комментария для выделенных блоков текста. Например, в IDLE для этого используются сочетания клавиш Alt + 3, Alt + 4 соответственно, в Jupyter Notebook — один переключатель: Ctrl + /.

Индексация (нумерация символов) в строке начинается с нуля. Кроме прямой, у символов строки есть и обратная нумерация (см. рис. 2.2): последний элемент имеет индекс -1, предпоследний -2 и т.д.

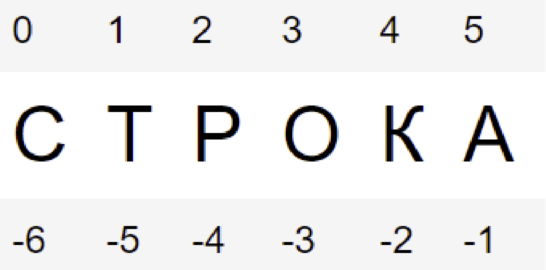


Рисунок 2.2 — Нумерация символов в строке

Чтобы получить информацию о том, какой символ расположен на заданной позиции строки, используется обозначение вида `s[1]` или `s[-2]`, где числа 1, -2 и т.д. — индексы требуемого символа:

```
In [2]: s = 'СТРОКА'
        print(s[1], s[-1])
```

```
Т А
```

Важным свойством строкового объекта является его *неизменяемость*.

После создания строки ни один из составляющих ее символов нельзя изменить. Доступ к символу с помощью кода вида `s[i]` — это доступ только на чтение:

```
In [3]: s = 'кот'
        s[1] = 'и'
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-3-c1e3c16d4598> in <module>
      1 s = 'кот'
----> 2 s[1] = 'и'
TypeError: 'str' object does not support item assignment
```

Полученное сообщение об ошибке гласит, что объект типа `str` не поддерживает присваивание отдельным его элементам.

Заметим здесь, что во многих задачах по программированию условия сформулированы так, что кажутся игнорирующими этот важный факт неизменяемости строк. Там часто встречаются формулировки вида: «Замените в строке строчные буквы заглавными», «Удалите из строки пробелы», «Поменяйте порядок символов в строке» и т. п. В случае языка Python такие задания следует интерпретировать так: Создайте **новую** строку, у которой «все буквы будут заглавными», «порядок символов изменен» и т. д.

Чтобы еще раз напомнить, как устроены переменные в языке Python, познакомимся с важной встроенной функцией `id`, которая по имени переменной возвращает уникальный идентификатор связанного с ней объекта. Этот идентификатор возникает в момент создания объекта в памяти. С некоторой долей условности его можно трактовать как адрес объекта в памяти компьютера.

In [4]:
```
s = 'кот'
print(id(s), s)
s += ' и пёс'
print(id(s), s)
```

```
1983976051600 кот
1983976051216 кот и пёс
```

Третья строчка приведенного выше кода может быть описана фразой «в результате выполнения операции += строка `s` стала длиннее». Но как видно из сравнения значений идентификаторов, на самом деле создана новая строка, значение которой действительно увеличено по сравнению с исходным, а переменная `s` теперь связана с этой новой строкой.

Очень важным при работе со строками является понятие **срез строки**. Оно означает новую строку, полученную как некоторую часть (подмножество) исходной строки. Срезом может быть непрерывная подстрока, или набор чередующихся с некоторым интервалом символов, или даже строка, составленная из символов исходной строки, взятых в обратном порядке. Синтаксис создания среза строки `s` выглядит следующим образом:

`s[start:finish:step]`

Три параметра в приведенном примере в некотором смысле аналогичны трем параметрам функции `range`: величина `start` — это номер (индекс) элемента строки, попадающего в срез первым, `finish` — индекс последнего элемента, до которого идет формирование среза, но сам он в это срез не попадает, `step` — шаг, с которым символы из строки переносятся в срез.

Как и в случае функции `range`, не все три параметра обязательны к использованию, однако здесь есть свои тонкости и отличия:

`s[start:finish]` — создание среза с шагом 1, т. е. непрерывной подстроки;

`s[start:]`, `s[start::step]` — создание среза до конца строки;

`s[:finish]`, `s[:finish:step]` — создание среза от начала строки;

`s[:]` — срез, являющийся копией строки;

`s[::-1]` — строка, полученная из исходной изменением порядка символов на противоположный.

2.2.2 Escape-последовательности

Escape-последовательность, или *управляющая* последовательность, или *экранированная* последовательность — это часть строки, которая начинается с символа «\», за которым следует один или более символов. Такие конструкции не являются видимой частью строки, в том смысле, что при выводе их на экран функцией `print` выполняется некоторое действие или изображается нечто другое.

В таблице приведены некоторые последовательности, работающие в Python 3 Shell и/или в среде Jupyter Notebook.

Последовательность	**Описание**
`\n`	Перенос строки (новая строка).
`\t`	Горизонтальная табуляция.
`\r`	Возврат курсора в начало строки[1]. Новые символы выводятся, «заменяя» старые.
`\b`	Аналог клавиши Backspace. Возврат курсора на одну позицию влево, удаление символа слева[1].
`\N{описание символа}`	Выводит символ по его описанию в базе данных Юникода. Например, так можно вывести многие эмодзи[2]: `print('\N{flushed face}')` выдает 😳

[1]Не работает в Shell (Windows)

[2]https://unicode.org/emoji/charts/emoji-list.html

`\xhh`	16-битный символ Юникода, т. е. символ, кодируемый двумя байтами (четыре шестнадцатеричные цифры). Например, `print('\u265E')` выдает ♞.
`\Uhhhhhhhh`	32-битный символ Юникода, т. е. символ, кодируемый четырьмя байтами (восемь шестнадцатеричных цифр). Таким способом тоже можно выводить эмодзи[2]: `print('\U0001F609')` выдает 😉.
`\\`	Экранирование обратного слэша. Позволяет включить в строку символ \
`\'`	Экранирование апострофа. Позволяет включить в строку символ апострофа '
`\"`	Экранирование кавычки. Позволяет включить в строку символ кавычки "

В приведенной таблице упоминается термин Юникод (от английского Unicode). Он означает преобладающий в настоящее время (и в Интернете, и в реализации операционных систем) стандарт кодирования символов. Применение этого стандарта позволяет закодировать очень большое число символов из разных систем письменности: в документах, закодированных по стандарту Юникод, могут соседствовать китайские иероглифы, математические символы, буквы греческого алфавита, латиницы и кириллицы, символы музыкальной нотной нотации и т. д. Каждому символу, допустимому по стандарту Юникод, присваивается уникальный код — неотрицательное целое число, записываемое обычно в шестнадцатеричной форме.

За работу с кодировкой символов в языке Python отвечают две функции, `ord` и `chr`. Результатом работы функции `ord(x)` является число —

порядковый номер, или позиция символа `x` в таблице Юникод. Функция `chr(n)` возвращает символ, соответствующий позиции с номером `n` в Юникоде. В качестве примера покажем как эти функции могут быть использованы для вывода на экран цепочки эмодзи:

```
In [5]: face = '\N{grinning face}'
        n = ord(face)
        for i in range(n, n+10):
            print(chr(i), end='')
```

Чтобы избежать экранирования символов в Python используются так называемые *сырые* строки, или r-строки (от английского raw): если перед открывающей кавычкой строки поставить букву r, то интерпретатор Python не будет воспринимать и выполнять управляющие последовательности: каждый символ в строке, в том числе и символ обратного слэша \, означает только самого себя. Сырые строки часто используются для задания пути к файлу в операционной системе, например:

```
path = r'C:\Users\Ivanov\Documents\data.csv'
```

2.2.3 f-строки

У разработчика программы часто возникает необходимость *форматирования* выводимой информации, т.е. ему недостаточно ограничиться печатью значений, разделенных пробелом. В языке Python существует несколько методов организации форматного вывода. В данном пособии будет рассмотрен только один, самый современный.

f-строки, или форматированные строковые литералы, появились в языке Python, начиная с версии 3.6, и продолжают развиваться. Перед открывающей кавычкой такой строки следует поставить букву f или F, тогда внутри строки в фигурных скобках можно помещать имена пе-

ременных и достаточно произвольные выражения. В момент создания строки значения этих переменных или выражений будут вычислены и подставлены в строку. Например, так

In [6]:
```
lang = "Python"
ver = 3.9
f'Используется язык {lang} версии {ver}'
```

Out[6]: `'Используется язык Python версии 3.9'`

Напомним еще раз, что строки в Python — это неизменяемые объекты. Поэтому изменение значения переменной, использованной для создания f-строки, не приведет к изменению этой строки.

Начиная с версии 3.8 языка Python, f-строки поддерживают еще один вариант добавления значения переменной или выражения: внутри фигурных скобок после имени можно поставить знак равенства. Тогда в создаваемую строку добавится и выражение, и его значение:

In [7]:
```
x = -3.14
f'Если {x = }, то {abs(x) = }'
```

Out[7]: `'Если x = -3.14, то abs(x) = 3.14'`

Разумеется, термин *форматирование* относится не к удобствам добавления в строку значения переменной или выражения, а к возможности определить, как это значение будет выглядеть при выводе на экран. f-строки поддерживают все возможности форматирования значений, доступные в предыдущих версиях языка. Для этого после имени переменной или выражения внутри фигурных скобок ставится двоеточие и задается требуемый формат вывода, например

```
f'{x:10.5f}'
```

Символ `f` в выражении `10.5f` означает, что выводится вещественное чис-

ло в формате с фиксированной точкой, а числа 10 и 5 являются параметрами этого вывода.

Перечислим некоторые возможные значения форматов:

`b` — двоичный формат;

`d` или `n` — формат вывода целых чисел;

`e` — научный формат, со строчной e, например `1.23e-01`;

`E` — научный формат, с заглавной E, например `1.23E-01`;

`f` или `F` — формат чисел с фиксированной точкой;

`o` — восьмеричный формат;

`x` или `X` — шестнадцатеричный формат;

`%` — процентный формат (значение умножается на 100, для вывода используется формат `f`, завершающийся символом %).

Теперь опишем параметры некоторых из перечисленных форматов. Для вещественных чисел (форматы `f` или `e`) самыми важными являются ширина, или количество позиций, отводимых на вывод значения, и точность — т. е. количество знаков после десятичной точки. Например, формат `f'{x:10.5f}'` означает, что под вывод числа будет отведено десять позиций, при этом после десятичной точки будет отображено пять цифр:

```
In [8]: x = 120.5344812
        f'{x:10.5f}  {x:16.10e}  {x:8.1e}'

Out[8]: '  120.53448  1.2053448120e+02   1.2e+02'
```

Следует уточнить, что если параметры ширина и точность противоречат друг другу, то во внимание прежде всего принимается точность: в следующем примере под вывод переменной `x` будет выделено не 8, а 10 позиций:

```
In [9]: x = 120.5344812
        f'{x:8.6f}'
```

```
Out[9]: '120.534481'
```

При выводе вещественных чисел можно указывать только один из параметров, например, ограничиваясь точностью: `{x:.3f}`. Можно ограничиться и шириной, но при этом нужно помнить, что точность тогда полагается равной шести, а приоритет по-прежнему отдается точности:

```
In [10]: x = 120.5344812
         f'{x:5f}'
```

```
Out[10]: '120.534481'
```

При выводе целых чисел и строковых значений основным параметром является ширина. При этом числовые значения располагаются выровненными по правому краю в пределах отведенных на них позиций, а строки — по левому:

```
In [11]: s = 'параметр:'
         d = 125
         print(f'{s:15}')
         print(f'{d:15}')
```

```
параметр:
            125
```

Чтобы поменять значение, установленное по умолчанию, нужно после двоеточия задать свое значение параметру выравнивания: знак < означает выравнивание значения по левому краю, знак > — выравнивание по правому краю, а символ ^ — по центру:

```
In [12]: print(f'{s:>15}')
         print(f'{d:<15}')
         print(f'{d:^15}')
```

```
      параметр:
125
      125
```

2.2.4 Функции и методы строк

В таблице ниже приведены основные операции со строковыми данными и методы, или функции, которые «умеет выполнять» объект типа `str`. С более полным списком можно познакомиться, например, в [11].

Операция, функция или метод	**Назначение**
`s1 + s2`	Конкатенация или слияние строк.
`s * n`	Повторение строки `n` раз.
`s[i]`	Доступ к символу по его индексу (отсчет индексов идёт с нуля).
`s[start:finish:step]`	Получение среза.
`len(s)`	Вычисление длины (количества символов) строки.
`s.find(s1, start, end)`	Поиск подстроки `s1` в строке. Возвращает номер **первого** вхождения или -1, если подстрока не найдена. По умолчанию `start=0`, то есть поиск идет с начала строки, а `end` соответствует последнему символу строки.

`s.rfind(s1, start, end)`	Поиск подстроки `s1` в строке. Возвращает номер **последнего** вхождения или -1.
`s.replace(pattern, new_substr, nmax)`	Замена шаблона `pattern` новой подстрокой. По умолчанию заменяются все вхождения шаблона. Необязательный параметр `nmax` может ограничить общее количество замен.
`s.upper()`	Преобразование символов строки к верхнему регистру.
`s.lower()`	Преобразование символов строки к нижнему регистру.
`s.count(s1, start, end)`	Возвращает количество непересекающихся вхождений подстроки `s1` в диапазоне [`start`, `end`]. По умолчанию — во всей строке.
`s.rstrip()`	Удаление пробельных символов в конце строки.
`s.ljust(width, fillchar=' ')`	Делает длину строки не меньшей `width`, по необходимости заполняя последние символы символом `fillchar`.
`s.rjust(width, fillchar=' ')`	Делает длину строки не меньшей `width`, по необходимости заполняя первые символы символом `fillchar`.
`s.center(width, fillchar=' ')`	Возвращает отцентрированную строку, по краям которой стоит символ `fillchar`

`s.split(sep)`	Разбиение строки на части по символу разделителю `sep`; по умолчанию строка разбивается на части по пробелу, при этом получающиеся пустые строки игнорируются.
`s.join(lst)`	Объединение списка строк в одну строку; в качестве склейки выступает строка `s`.

Еще раз обратим внимание, что ни один из методов не меняет исходную строку, а создает новую — с указанными свойствами:

In [13]:
```
me = 'I like algebra'
you = me.replace('I', 'You')
you = you.replace('algebra', 'apples')
print(me)
print(you)
```

```
I like algebra
You like apples
```

2.3 Табулирование функции

2.3.1 Постановка задачи

Вывести на экран таблицу значений функций $f(x)$, $g(x)$, а также их суперпозиций $f(g(x))$ и $g(f(x))$ при $x \in [a, b]$ с шагом h. Данные о функциях и значениях параметров a, b, h для каждого варианта приведены в таблицах 2.3–2.5. При программировании функций необходимо учитывать область их определения: если при каком-то значении x функция не определена, то в качестве значения она должна возвращать величину `None`.

Таблица 2.3 — Данные о функциях и значения параметров

Номер варианта	**Номер функции f (табл. 2.4)**	**Номер функции g (табл. 2.5)**	a	b	h
0	1	1	-3.0	2.0	0.25
1	2	2	-2.0	1.0	0.2
2	3	2	0.0	4.0	0.1
3	4	2	1.0	3.0	0.25
4	5	2	-1.0	3.0	0.2
5	6	2	-2.0	2.0	0.1
6	7	2	-3.0	-1.0	0.25
7	2	3	2.0	5.0	0.2
8	3	3	1.0	4.0	0.1
9	4	3	-2.0	2.0	0.25
10	5	3	0.0	4.0	0.2
11	6	3	-1.0	3.0	0.1
12	7	3	-1.0	2.0	0.25
13	2	4	-2.0	4.0	0.2

14	3	4	1.0	2.0	0.05
15	4	4	-2.5	1.5	0.25
16	5	4	0.0	1.0	0.05
17	6	4	-4.0	6.0	0.5
18	7	4	-1.0	1.0	0.2
19	2	5	-0.3	2.0	0.1
20	3	5	-0.5	1.5	0.1
21	4	5	1.0	3.0	0.25
22	5	5	0.0	4.0	0.2
23	6	5	-1.0	4.0	0.1
24	7	5	-2.0	2.0	0.25
25	2	6	-1.0	3.0	0.2
26	3	6	4.0	6.0	0.1
27	4	6	5.0	8.0	0.2
28	5	6	-1.0	3.0	0.2
29	6	6	2.0	6.0	0.1
30	7	6	0.0	4.0	0.25

Таблица 2.4 — Вид функции $f(x)$

№	$\boldsymbol{f(x)}$
1	$\sqrt{\sin\left(\frac{\pi}{2}+x^5\cos x-x\sqrt{3-\lvert x\rvert}\right)-0.5}+\frac{e^{3x+\cos x}}{1-x^5\sin x+e^x\,\mathrm{tg}\,x}$
2	$\frac{2}{\sqrt{0.5-\sin\left(x^5\cos x-\pi\sqrt{x+4}\right)}}+\frac{\mathrm{tg}(3-7x)}{1+x^7\lvert\sin x-0.3\rvert}$
3	$\sqrt{0.2-\cos\left(\frac{\pi}{2}+x^5\cos x-x\sqrt{6-\lvert x\rvert}\right)}+\frac{e^{3+\cos x+x^2}}{1+x^7\sin 2x+e^{2x}\,\mathrm{tg}\,3x}$
4	$\frac{3x-1}{\sqrt{\sin\left(\frac{\pi}{4}+x^3\cos 3x-\sqrt{3\lvert x\rvert+5\sin x}\right)-0.5}}+$ $+\frac{e^{4x^2+\cos(x-\pi)}}{3-x^5\sin x^2+e^{2x}\,\mathrm{tg}\left(x-\frac{\pi}{4}\right)}$
5	$\sqrt{\sin\left(-\frac{\pi}{4}+x^5\cos(2x^3-\pi^3)+\frac{\pi}{4}\sqrt{0.2\lvert x\rvert+4\cos x}\right)-0.5x}+$ $+\frac{2\cos\left(xe^{3x+\cos x}\right)}{2-x^5\sin 4x+e^x\,\mathrm{tg}\,4x}$
6	$\frac{1}{\sqrt{\sin\left(x^5\cos x+\frac{\pi}{4}\sqrt{x+4\cos x}\right)-0.5}}+\frac{1-x^5\sin x+e^x\,\mathrm{tg}\,x}{e^{3x+\cos x}}$
7	$\sqrt[4]{0.1+\cos\left(x^3\sin x-x\cos 3x\right)}+\frac{\sin(xe^{\cos x}+1)}{1+\sin 7x^7+\mathrm{ctg}\left(\frac{\pi-x}{x^2+2}\right)}$

Таблица 2.5 — Вид функции $g(x)$

№	$\boldsymbol{g(x)}$
1	$\sin\left(x+\pi\dfrac{\lg(x+\sqrt{25+2x})}{(1+x)^4+1}\right)$
2	$\operatorname{tg}\dfrac{\sqrt{x^2-\sin x^2+\cos^2 x}}{x+\lvert\sin x-\sin e^x\rvert}$
3	$\operatorname{ctg}\dfrac{\left\lvert x^3-e^x\ln(0.2x+1)-1\right\rvert}{x^7+x^5+x^3+\cos x}$
4	$\ln\left(2+\sin\dfrac{x^2}{\sqrt[4]{x+16\cos x}}-\cos\dfrac{1}{x^2+1}\right)$
5	$\log_2\left(\cos\dfrac{\pi(x^2-1)}{\sqrt{\sin x+4\cos x}}-\operatorname{tg}\dfrac{\pi}{x^2+1}+32\right)$
6	$e^{1+\sin\sqrt{\sin x-x^3}}\left(\dfrac{1}{2x^4+x^2+\cos x}-\dfrac{1}{\lvert x^3-27\rvert}\right)$

2.3.2 Образец выполнения задания

1. Условие задачи.

Вывести на экран таблицу значений функций $f(x)$, $g(x)$, а также их суперпозиций $f(g(x))$ и $g(f(x))$ при $x\in[a,b]$ с шагом h.

$$f(x)=\sqrt{\sin\left(\frac{\pi}{2}+x^5\cos x-x\sqrt{3-|x|}\right)-0.5+} \\ +\frac{e^{3x+\cos x}}{1-x^5\sin x+e^x\operatorname{tg}x}, \qquad (2.1)$$

$$g(x) = \sin\left(x + \pi \frac{\lg\left(x + \sqrt{25 + 2x}\right)}{(1 + x)^4 + 1}\right), \tag{2.2}$$

$$a = -3, \quad b = 2, \quad h = 0.25. \tag{2.3}$$

При программировании функций учитывать область их определения: если при каком-то значении функция не определена, выдавать в качестве ее значения величину `None`.

2. Анализ области определения функций.

2.1. В соотношение (2.1) для функции $f(x)$ входят такие выражения как квадратный корень, тангенс и дробь. Поэтому при вычислении области определения нужно проверять следующие условия:

- подкоренное выражение $3 - |x|$ неотрицательно;
- подкоренное выражение $\sin\left(\frac{\pi}{2} + x^5 \cos(x) - x\sqrt{3 - |x|}\right) - 0.5$ неотрицательно;
- $\cos x$ не равен нулю (чтобы был определен $\operatorname{tg} x$);
- знаменатель $1 - x^5 \sin x + e^x \operatorname{tg}(x)$ не равен нулю.

Поскольку решить систему перечисленных выше неравенств аналитически затруднительно (скорее всего, вообще нельзя), будем проверять выполнение этих условий (именно в таком порядке) численно при реализации алгоритма вычисления функции $f(x)$.

2.2. В соотношение (2.2) для функции $g(x)$ входят такие выражения как квадратный корень, логарифм и дробь. Поскольку знаменатель дроби всегда положителен, то при вычислении области определения нужно проверять следующие условия:

- подкоренное выражение $25 + 2x$ больше нуля;
- аргумент логарифма $x + \sqrt{25 + 2x}$ положителен.

Для единообразия и в этом случае ограничимся численной проверкой этих неравенств при реализации алгоритма вычисления функции $g(x)$.

3. Программная реализация функций $f(x)$ и $g(x)$ и их тестирование.

При реализации математических функций будем использовать модуль `math`. Код функций приведен ниже:

```
from math import sin, cos, tan, exp, sqrt, log10, pi
def f(x):
    r1 = 3 - abs(x)
    if r1 >= 0:
        r2 = sin(pi/2 + x**5*cos(x) - x*sqrt(r1)) - 0.5
        if r2 >= 0 and cos(x) != 0:
            z = 1 - x**5*sin(x) + exp(x)*tan(x)
            if z != 0:
                return sqrt(r2) + exp(3*x+cos(x))/z
def g(x):
    r1 = 25 + 2*x
    if r1 > 0 and x+sqrt(r1) > 0:
        return sin(x + pi*log10(x+sqrt(r1))/((1+x)**4+1))
```

В качестве первого тестового значения аргумента возьмем $x = 0$. Подставляя $x = 0$ в выражения (2.1) и (2.2), находим

$$f(0) = \sqrt{0.5} + e \approx 3.4253886$$
$$g(0) = \sin\left(\pi \frac{\lg(5)}{2}\right) \approx 0.8902708$$

В качестве второго тестового значения подберем такое значение x, когда функции не определены, например, $x = -20$. Нужно проверить, что значения функций действительно не определены, или, что эквивалентно, возвращаемое значение функций равно `None`.

Проверки предыдущей программы на указанных тестовых значениях в Jupyter Notebook прошли успешно.

```
In [2]: f(0)

Out[2]: 3.4253886096455926
```

In [3]:
```
g(0)
```

Out[3]:
```
0.8902708410246845
```

In [4]:
```
print(f(-20))
```

```
None
```

In [5]:
```
print(g(-20))
```

```
None
```

4. Построение таблицы значений.

Дополним программу циклом, в котором будем вычислять требуемые значения функций и выводить на экран в виде таблицы значений. Поскольку при вычислении суперпозиции, например, $f(g(x))$ существует опасность, что функция $g(x)$ не определена, т. е. имеет значение `None`, учтем это обстоятельство и добавим соответствующую проверку в функции $f(x)$ и $g(x)$, считая, что если аргумент функции не определен, то и значение — тоже не определено.

Чтобы столбцы значений функций выводились на печать выровненными вне зависимости от количества получившихся значащих цифр, будем использовать возможности форматирования чисел, предоставляемых f-строками языка Python: для вывода значения вещественных переменных служит конструкция:

```
print(f'{z:15.6f}')
```

Под вывод значения переменной z в этом примере будет отведено пятнадцать позиций, при этом в числе после десятичной точки будет выведено 6 цифр. Само число будет выровнено по правому краю. *Разумеется, числа 15 и 6 приведены просто для примера.*

Серьезная проблема использования f-строк связана с невозможностью такого форматирования, если переменная z имеет значение `None`.

Одно из решений проблемы: с использованием условного оператора превратим значение `None` в строку `'None'`, которую «отформатируем» используя метод `rjust(n)` строковых переменных, позволяющий создать строку нужной ширины, выровненную по правому краю.

Для удобства выполнения таких операций опишем, а затем используем в программе, отдельную функцию

```
def out(z, width, digits):
'''
Параметр z - вещественное число или None
преобразуется в строку шириной width позиций.
У числового значения отображается digits знаков
после запятой
'''
    if z==None:
        return 'None'.rjust(width)
    else:
        return f'{z: {width}.{digits}f}'
```

5. Окончательный текст программы.

Ниже приведен текст разработанной программы и демонстрация результатов ее выполнения в среде Jupyter Notebook.

```
from math import sin, cos, tan, exp, sqrt, log10, pi
def f(x):
    if x == None:
        return None
    r1 = 3 - abs(x)
    if r1 >= 0:
        r2 = sin(pi/2 + x**5*cos(x) - x*sqrt(r1)) - 0.5
        if r2 >= 0 and cos(x) != 0:
            z = 1 - x**5*sin(x) + exp(x)*tan(x)
            if z !=0:
                return sqrt(r2) + exp(3*x+cos(x))/z
def g(x):
    if x == None:
        return None
    r1 = 25 + 2*x
```

```
16      if r1 > 0 and x+sqrt(r1) > 0:
17          return sin(x+pi*log10(x+sqrt(r1))/((1+x)**4+1))
18
19  def out(z, width, digits):
20      if z==None:
21          return 'None'.rjust(width)
22      else:
23          return f'{z: {width}.{digits}f}'
24  w = 15 # позиций под значение функции
25  d = 6  # знаков после запятой
26  a = -3.0
27  b = 2
28  h = 0.25
29  x = a
30  while x<=b:
31      fx = out(f(x), w, d)
32      gx = out(g(x), w, d)
33      fg = out(f(g(x)), w, d)
34      gf = out(g(f(x)), w, d)
35      print(f"{x: .2f}", fx, gx, fg, gf)
36      x += h
```

```
-3.00           None      -0.165441       2.580795           None
-2.75           None      -0.442767       1.469440           None
-2.50           None      -0.713464       0.851529           None
-2.25       0.624613      -0.938390      -0.566516       0.798102
-2.00           None      -0.976322       0.037128           None
-1.75           None      -0.606154       1.021619           None
-1.50           None      -0.010228       3.369961           None
-1.25       0.508326       0.434993       5.861850       0.776616
-1.00       0.170383       0.731109       7.220981       0.815047
-0.75       0.834045       0.919520       7.993119       0.858597
-0.50       1.292850       0.985831       8.195231       0.982298
-0.25       2.198881       0.976570       8.172085       0.793476
 0.00       3.425389       0.890271       7.883723      -0.287503
 0.25       4.846559       0.790214       7.471141      -0.990638
 0.50       6.187900       0.775605       7.409320      -0.093865
```

```
0.75   7.300950    0.832320    7.648481    0.851330
1.00   8.226634    0.911281    7.963226    0.931181
1.25   7.290606    0.974808    8.167462    0.845859
1.50       None    0.999985    8.226603        None
1.75       None    0.974767    8.167353        None
2.00       None    0.894952    7.901801        None
```

2.4 Рекурсия

2.4.1 Постановка задачи

Реализовать рекурсивный и итерационный алгоритмы вычисления функции f в соответствии с выбранным вариантом. Подобрать не менее трех тестовых значений входных параметров и провести на них отладку реализованных функций.

Вариант 1-18. Составить таблицу значений функции f при $n = 1, 2, 3, \ldots, 20$.

Вариант 19-30. При $x = 1$ и при $x = 2$ составить таблицы значений функции $f(n, x)$ при $n = 1, 2, 3, \ldots, 10$.

Таблица 2.6 — Функции $f(x)$

№	**Выражение для функции $f(x)$**
0	$f(n) = \sqrt{1 + \sqrt{2 + \sqrt{3 + \ldots + \sqrt{n}}}}$
1	$f(n) = \sqrt{1 + \sqrt{\frac{1}{2} + \sqrt{\frac{1}{3} + \ldots + \sqrt{\frac{1}{n}}}}}$
2	$f(n) = \sqrt{1^2 + \sqrt{2^2 + \sqrt{3^2 + \ldots + \sqrt{n^2}}}}$

3	$f(n)=\sqrt{1+\sqrt{3+\sqrt{5+\ldots+\sqrt{2n-1}}}}$
4	$f(n)=\sqrt{2+\sqrt{4+\sqrt{6+\ldots+\sqrt{2n}}}}$
5	$f(n)=\underbrace{\sqrt{n+\sqrt{n+\sqrt{n+\ldots+\sqrt{n}}}}}_{n \text{ радикалов}}$
6	$f(n)=\underbrace{\sqrt{n+\sqrt{1+\sqrt{1+\ldots+\sqrt{1}}}}}_{n+1 \text{ радикал}}$
7	$f(n)=\sqrt{n+\sqrt{(n-1)+\sqrt{(n-2)+\ldots+\sqrt{2+\sqrt{1}}}}}$
8	$f(n)=\sqrt{n+\sqrt{(n+1)+\sqrt{(n+2)+\ldots+\sqrt{2n}}}}$
9	$f(n)=\sqrt{2n+\sqrt{2(n-1)+\sqrt{2(n-2)+\ldots+\sqrt{4+\sqrt{2}}}}}$
10	$f(n)=\sqrt{2n+1+\sqrt{2(n-1)+1+\sqrt{2(n-2)+1+\ldots+\sqrt{1}}}}$
11	$f(n)=\sin\left(1+\sin\left(\frac{1}{2}+\sin\left(\frac{1}{3}+\ldots+\sin\frac{1}{n}\right)\ldots\right)\right)$
12	$f(n)=\sin\left(1^2+\sin\left(2^2+\sin\left(3^2+\ldots+\sin n^2\right)\ldots\right)\right)$
13	$f(n)=\cos(1+\sin(2+\cos(3+\sin(4+\cos(5+\ldots+\sin 2n)\ldots))))$
14	$f(n)=\sin(n+\sin(n-1+\sin(n-2+\ldots+\sin 1)\ldots))$

15	$f(n) = \cos\left(1^3 + \cos\left(2^3 + \cos\left(3^3 + \ldots + \cos n^3\right)\ldots\right)\right)$
16	$f(n) = \cos(2 + \cos(4 + \cos(8 + \ldots + \cos 2n)\ldots))$
17	$f(n) = \cos\left(1^3 + \cos\left(2^3 + \cos\left(3^3 + \ldots + \cos n^3\right)\ldots\right)\right)$
18	$f(n) = \cos\left(\frac{1}{1^n} + \cos\left(\frac{1}{2^n} + \cos\left(\frac{1}{3^n} + \ldots + \cos\frac{1}{n^n}\right)\ldots\right)\right)$
19	$f(n,x) = \sqrt{x + \sqrt{(x+1) + \sqrt{(x+2) + \ldots + \sqrt{x+n}}}}$
20	$f(n,x) = \sqrt{x + \sqrt{2x + \sqrt{3x + \ldots + \sqrt{nx}}}}$
21	$f(n,x) = \underbrace{\sqrt{x + \sqrt{x + \sqrt{x + \ldots + \sqrt{x}}}}}_{n \text{ радикалов}}$
22	$f(n,x) = \underbrace{\sqrt{x + \sqrt{n + \sqrt{x + \ldots + \sqrt{n}}}}}_{2n \text{ радикалов}}$
23	$f(n,x) = \underbrace{\sqrt{x + \sqrt{n + \sqrt{n + \ldots + \sqrt{n}}}}}_{n+1 \text{ радикал}}$
24	$f(n,x) = \underbrace{\sqrt{x + \sqrt{x^2 + \sqrt{x^3 + \ldots + \sqrt{x^n}}}}}_{n \text{ радикалов}}$
25	$f(n,x) = \underbrace{\sqrt{x + \sqrt{2x + \sqrt{3x + \ldots + \sqrt{nx}}}}}_{n \text{ радикалов}}$
26	$f(n,x) = \sin(x + \sin((x+1) + \sin((x+2) + \ldots + \sin(x+n))))$

27	$f(n,x)=\sin\left(x+\sin\left(2x+\sin\left(3x+\ldots+\sin(nx)\right)\right)\right)$
28	$f(n,x)=\cos\left(nx+\cos\left((n-1)x+\cos\left((n-2)x+\ldots+\cos x\right)\right)\right)$
29	$f(n,x)=\cos\left(\frac{1}{x}+\cos\left(\frac{2}{x^2}+\cos\left(\frac{3}{x^3}+\ldots+\cos\frac{n}{x^n}\right)\ldots\right)\right)$
30	$f(n,x)=\sin\left(\frac{1}{x}+\cos\left(\frac{2}{x}+\sin\left(\frac{3}{x}+\ldots+\cos\frac{2n}{x}\right)\ldots\right)\right)$

Указания и комментарии к приведенному ниже решению.

1. В случае, если Ваша функция зависит от двух параметров (n, x), то для отладки программы можно выбрать в качестве x любое положительное значение, например, 1.0.
2. Основная сложность с реализацией рекурсивного алгоритма вычисления функции f состоит в том, что из формулы для функции сразу не виден ни «терминальный случай», ни рекурсивная формула. В отличие от, например, задачи вычисления факториала тут терминальным является не ноль или единица, а случай n, когда нужно просто вычислить квадратный корень, без дополнительного суммирования. В таких задачах используется прием введения дополнительного аргумента функции (в нашем примере — i), который и служит изменяющимся от 1 до n индексом глубины рекурсии, а переменная n выступает постоянным, не меняющимся параметром.
3. Для выравнивания результатов в таблице при печати использованы возможности форматирования, предоставляемые f-строками. Указанный прием был использован и в предыдущем задании.

2.4.2 Образец выполнения задания

1. Условие задачи.

Реализовать рекурсивный и итерационный алгоритмы вычисления

функции

$$f(n) = \sqrt{1 + \sqrt{2 + \sqrt{3 + \ldots + \sqrt{n}}}}$$

Составить таблицу значений функции f при $n = 1, 2, 3, \ldots, 20$.

2. Данные для отладки программы.

Воспользовавшись калькулятором, найдем значения функции для трех первых значений n, т. е. для $n = 1, 2, 3$.

$$\begin{aligned}
&f(1) = \sqrt{1} = 1,\\
&f(2) = \sqrt{1 + \sqrt{2}} \approx \sqrt{2.414214} \approx 1.553774,\\
&f(3) = \sqrt{1 + \sqrt{2 + \sqrt{3}}} \approx \sqrt{1 + \sqrt{3.73205}} \approx \sqrt{2.93185} \approx 1.71226.
\end{aligned}$$

3. Разработка рекурсивной версии функции.

Вводим дополнительную переменную i, показывающую с какого значения начинается вычисление радикалов:

$$\begin{aligned}
&f(1, n) = \sqrt{1 + \sqrt{2 + \sqrt{3 + ... + \sqrt{n}}}},\\
&f(2, n) = \sqrt{2 + \sqrt{3 + ... + \sqrt{n}}},\\
&f(3, n) = \sqrt{3 + ... + \sqrt{n}},\\
&\ldots\\
&f(n, n) = \sqrt{n}.
\end{aligned}$$

Терминальный случай: $i = n$, в этом случае просто вычисляется квадратный корень из n. В остальных случаях получается следующая рекурсивная формула

$$f(i, n) = \sqrt{i + f(i + 1, n)},$$

которую и программируем

```
from math import sqrt
def f_recur(i, n):
    if i == n:
        return sqrt(n)
    else:
        return sqrt(i+f_recur(i+1,n))
```

4. Разработка итерационной версии функции.

На первом шаге цикла вычисляем $\sqrt{n}$, потом последовательно осуществляем следующие операции: прибавляем к полученному числу $n-1$, извлекаем корень, прибавляем $n-2$, извлекаем корень и т. д., пока очередное прибавляемое слагаемое больше нуля.

```
from math import sqrt
def f_iter(n):
    s = sqrt(n)
    for i in range(n-1, 0, -1):
        s = sqrt(i+s)
    return s
```

5. Итоговая программа.

```
from math import sqrt
def f_recur(i, n):
    if i == n:
        return sqrt(n)
    else:
        return sqrt(i+f_recur(i+1,n))
def f_iter(n):
    s = sqrt(n)
    for i in range(n-1,0,-1):
        s = sqrt(i+s)
    return s
for n in range(1,21):
    print(f' n = {n:2d}  f_r = {f_recur(1,n):.14f}'
          f'  f_it = {f_iter(n):.14f}')
```

6. Анализ результатов.

Результатом работы программы в Jupyter Notebook является представленная ниже таблица.

```
n =  1  f_r = 1.00000000000000  f_it = 1.00000000000000
n =  2  f_r = 1.55377397403004  f_it = 1.55377397403004
n =  3  f_r = 1.71226506492953  f_it = 1.71226506492953
n =  4  f_r = 1.74876271325514  f_it = 1.74876271325514
n =  5  f_r = 1.75623848758234  f_it = 1.75623848758234
n =  6  f_r = 1.75764123504158  f_it = 1.75764123504158
n =  7  f_r = 1.75788564609644  f_it = 1.75788564609644
n =  8  f_r = 1.75792555756826  f_it = 1.75792555756826
n =  9  f_r = 1.75793171051457  f_it = 1.75793171051457
n = 10  f_r = 1.75793261139383  f_it = 1.75793261139383
n = 11  f_r = 1.75793273728207  f_it = 1.75793273728207
n = 12  f_r = 1.75793275414063  f_it = 1.75793275414063
n = 13  f_r = 1.75793275631170  f_it = 1.75793275631170
n = 14  f_r = 1.75793275658137  f_it = 1.75793275658137
n = 15  f_r = 1.75793275661376  f_it = 1.75793275661376
n = 16  f_r = 1.75793275661753  f_it = 1.75793275661753
n = 17  f_r = 1.75793275661795  f_it = 1.75793275661795
n = 18  f_r = 1.75793275661800  f_it = 1.75793275661800
n = 19  f_r = 1.75793275661800  f_it = 1.75793275661800
n = 20  f_r = 1.75793275661800  f_it = 1.75793275661800
```

Из нее видно, что первые три значения совпадают с полученными в п. 2 данными для отладки программы; значения функции, вычисленные с помощью итерационного и рекурсивного алгоритма совпадают; последовательность $a_n = f(n)$, по-видимому, имеет предел.

2.5 Работа со строками

2.5.1 Постановка задачи

Разработайте функцию, которая в заданной строке s ищет подстроку, удовлетворяющую условиям, перечисленным в таблице 2.7.

При работе с таблицей 2.7 нужно учитывать только непустые ячейки в строке, соответствующей номеру варианта.

Если длина искомой подстроки не задана, то следует искать кратчайшую из возможных.

Результатом функции должен быть кортеж, состоящий из целого числа и строки, определяемых по следующему правилу:

- Если подходящая подстрока обнаружена, то результатом функции является номер ее первого символа внутри строки s и сама эта подстрока;
- Если подходящая подстрока не обнаружена, то результатом функции должно быть число -1 и пустая строка;
- Если входные параметры не имеют смысла (например, длина подстроки отрицательна, или последовательности символов — пустые), результатом функции должно быть число -2 и пустая строка.

Требуется разработать не менее 7 вариантов значений входных параметров и проверить, что на всех этих значениях функция работает корректно. В качестве строки s выбирать строки, приведенные в таблице 2.8, в соответствии с вариантом. Значения остальных параметров функции подобрать самостоятельно.

Таблица 2.7 — Варианты задания № 3

Вариант	Начинается с последовательности символов	Заканчивается последовательностью символов	Имеет длину	В свою очередь содержит подстроку	Не содержит внутри себя подстроки	От начала строки расположена не далее, чем	К началу строки расположена не ближе, чем
0	stS		subL	hasS			leftM
1		enS	subL	hasS		leftD	leftM
2	stS		subL	hasS	noS		
3	stS		subL		noS	leftD	
4		enS	subL	hasS	noS		leftM
5	stS	enS			noS	leftD	
6	stS	enS		hasS		leftD	
7	stS	enS			noS	leftD	leftM
8	stS		subL	hasS	noS		leftM
9		enS	subL	hasS	noS	leftD	
10	stS		subL	hasS		leftD	leftM
11	stS	enS			noS		leftM
12	stS		subL	hasS		leftD	
13	stS	enS				leftD	leftM
14		enS	subL		noS	leftD	leftM
15	stS	enS		hasS	noS		leftM
16	stS		subL	hasS	noS	leftD	
17		enS	subL	hasS	noS		
18	stS		subL		noS	leftD	leftM

19	stS	enS		hasS	noS	leftD	
20		enS	subL		noS		leftM
21		enS	subL	hasS		leftD	
22	stS	enS		hasS			leftM
23		enS	subL		noS	leftD	
24	stS	enS	subL	hasS			
25		enS	subL	hasS			leftM
26	stS		subL		noS		leftM
27	stS	enS		hasS		leftD	leftM
28	stS		subL			leftD	leftM
29	stS	enS		hasS	noS		
30		enS	subL			leftD	leftM

Таблица 2.8 — Строки к заданию № 3

Вариант	**Строка для тестирования**
0–5	*Чудесная загадка соответствия математического языка законам физики является удивительным даром, который мы не в состоянии понять и которого мы, возможно, недостойны. (Юджин Пол Вигнер)*
6–10	*Человек, не способный к математике, не является разумным. Этого недочеловека в лучшем случае можно терпеть, раз он научился носить ботинки, мыться и не сорить в доме. (Роберт Ханлайн)*
11–15	*Если поручить двум людям, один из которых математик, выполнение любой незнакомой им работы, то результат всегда будет следующим: математик сделает ее лучше. (Гуго Штейнгауз)*

16–20	*Если нам не дано достичь полного знания о движении жидкости, то причину неудач следует приписывать не механике и не недостаточности известных законов движения. Нам недостаёт математического анализа. (Леонард Эйлер)*
21–25	*Истинная математика заключается не в нагромождении искусственных вычислительных приёмов, а в умении получать нетривиальные результаты путём размышления при минимуме применяемого аппарата. (Ганс Радемахер, Отто Теплиц)*
26–30	*Если бы Ньютон и Лейбниц знали, что непрерывные функции необязательно должны иметь производные, то дифференциальное исчисление никогда не было бы создано. (Эмиль Пикар)*

2.5.2 Образец выполнения задания

1. Условие задачи.

Выбирая непустые клеточки в строке таблицы 2.7, соответствующей варианту №0, получаем условие задачи в окончательном виде:
Разработать функцию, которая в данной строке s ищет подстроку, удовлетворяющую следующим условиям:

А. Начинается с последовательности символов `stS`;

Б. Имеет длину `subL`;

В. В свою очередь содержит подстроку `hasS`;

Г. К началу строки расположена не ближе чем `leftM`.

Таким образом, разрабатываемая функция должна иметь вид

```
findSubs(s, stS, subL, hasS, leftM)
```

2. Подбор входных данных для тестирования.

Поскольку строка s определена номером варианта:

```
s = 'Чудесная загадка соответствия математического \
языка законам физики является удивительным даром, \
который мы не в состоянии понять и которого мы, \
возможно, недостойны. (Юджин Пол Вигнер)'
```

подберем остальные параметры так, чтобы проверялись основные возможные ситуации.

А. Неправильные входные данные: зададим `subL` отрицательным

Б. Несогласованные входные данные: для `hasS` используем значение `'физики'`, а `subL` возьмем равным 5, т.е. меньшим, чем длина `hasS`

В. Входные данные согласованы, но подстрока с нужными свойствами отсутствует:

- В1. Зададим `stS = 'привет'`. В строке s ни одна подстрока не начинается с такого текста
- В2. Зададим `stS = 'загадка'`, но в качестве левого отступа зададим 12 — подстрока `'загадка'` находится ближе к левому краю, чем 12
- В3. Зададим `stS = 'загадка'`, но в качестве внутренней подстроки выберем `'физика'`

Г. Входные данные согласованы, и подстрока присутствует (2 варианта)

В итоге определили следующие 7 вариантов вызова функции `findSubs`:

Вариант вызова	Ожидаемый результат
`findSubs(s,'загадка',-5,'физики',5)`	`-2, ''`
`findSubs(s,'загадка',5,'физики',5)`	`-1, ''`
`findSubs(s,'привет',58,'физики',5)`	`-1, ''`
`findSubs(s,'загадка',58,'физики',12)`	`-1, ''`
`findSubs(s,'загадка',58,'физика',5)`	`-1, ''`
`findSubs(s,'загадка',20,'соответствия',5)`	`9,'загадка соответствия'`
`findSubs(s,'мы',17,'не',50)`	`104,'мы не в состоянии'`

3. Алгоритм поиска подстроки.

1. Проверяем, имеют ли смысл входные параметры. Возвращаем -2, если выполнено одно из условий:
   (a) `stS = ''`
   (b) `subL <= 0`
   (c) `hasS = ''`
   (d) `leftM <= 0`
2. Проверяем, не противоречат ли входные параметры функции друг другу. Возможные противоречия:
   (a) `subL > len(s)` (длина искомой подстроки больше длины самой строки)
   (b) `len(stS) > subL` (длина части подстроки больше длины всей подстроки)
   (c) `len(hasS) > subL` (длина части подстроки больше длины всей подстроки)
   (d) `subL + leftM > len(s)` (подстрока не поместится в отведенном месте строки)

   В любом из указанных случаев возвращаем -1.
3. Вводим переменную i — номер символа в строке s. Перебираем поочередно символы строки s, начиная с `leftM`, пока выполнено условие, что `i + subL < len(s)`. На каждом шаге проверяем, не совпадает ли срез строки s, начинающийся с i-го символа, со строкой `stS`, и если да — то не содержит ли срез длины `subL` подстроку `hasS`. Если оба условия выполнены — возвращаем необходимый результат.

4. Код функции на языке Python.

```
1 def findSubs(s, stS, subL, hasS, leftM):
2     '''
3     Поиск в строке s подстроки,
4     удовлетворяющей следующим условиям:
```

```
5         - начинается с stS,
6         - имеет длину subL,
7         - в свою очередь содержит подстроку hasS,
8         - отстоит от начала строки s
9         не менее чем на leftM позиций.
10        Результат работы:
11        номер позиции, с которой начинается такая подстрока
12        и сама эта подстрока;
13        если подстрока не найдена,
14        вместо номера возвращается -1;
15        если входные данные некорректны, возвращается -2.
16        '''
17        n = len(s)
18        n1 = len(stS)
19        # Проверяем значения параметров функции
20        # на корректность
21        if stS=='' or hasS=='' or subL<=0 or leftM<=0:
22            return -2, ''
23        # Проверяем, не противоречат ли параметры функции
24        # друг другу
25        if subL > n or len(stS) > subL or \
26           len(hasS) > subL or subL + leftM > n:
27            return -1, ''
28        i = leftM
29        while i+subL <= n:
30            if s[i:i+n1]==stS and s[i:i+subL].count(hasS)>0:
31                return i, s[i:i+subL]
32            i += 1
33        return -1, ''
```

5. Итоги работы, анализ.

Тестирование разработанной функции было проведено в среде Jupyter Notebook. Ниже приведены итоги этого тестирования. Видно, что все ожидаемые результаты получены. Это позволяет сделать заключение о том, что разработанная функция, скорее всего, реализована без ошибок.

In [2]:
```
s = 'Чудесная загадка соответствия математического \
языка законам физики является удивительным даром, \
который мы не в состоянии понять и которого мы, \
возможно, недостойны. (Юджин Пол Вигнер)'
print(findSubs(s, 'загадка', -5, 'физики', 5))
print(findSubs(s, 'загадка', 5, 'физики', 5))
print(findSubs(s, 'привет', 58, 'физики', 5))
print(findSubs(s, 'загадка', 58, 'физики', 12))
print(findSubs(s, 'загадка', 58, 'физика', 5))
print(findSubs(s, 'загадка', 20, 'соответствия', 5))
print(findSubs(s, 'мы', 17, 'не', 50))
```

```
(-2, '')
(-1, '')
(-1, '')
(-1, '')
(-1, '')
(9, 'загадка соответствия')
(104, 'мы не в состоянии')
```

2.6 Оформление отчета

В *записной книжке* — файле, создаваемом в среде Jupyter — есть два основных типа ячеек (см. рис. 2.3): Code и Markdown. Первые содержат фрагменты программ на языке Python, вторые — форматированный текст на языке разметки Markdown. Выполнение (комбинация клавиш Shift + Enter) ячеек первого типа приводит к выполнению кода, а выполнение ячеек второго типа — к обработке команд разметки и отображению отформатированного текста.

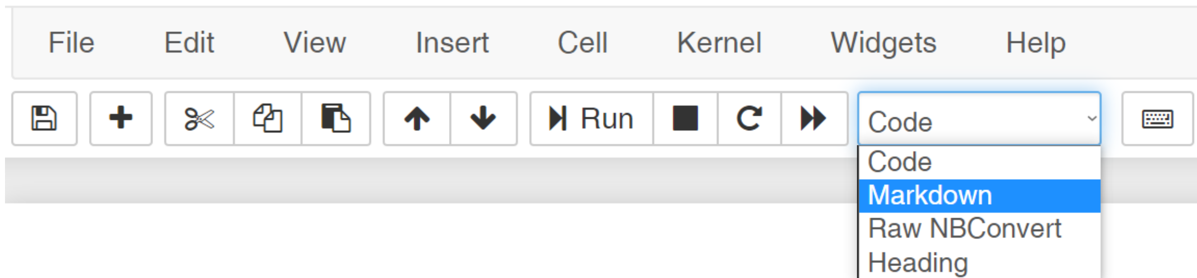


Рисунок 2.3 — Строка меню среды Jupyter

В ячейках типа Code необходимо привести фрагменты кода (если нужно их тестировать) и окончательное решение задачи. Отчет следует сохранить вместе с результатами выполнения всех ячеек с кодом.

Что касается оформления текстовой части отчета, то информацию о правилах разметки ячеек типа Markdown можно посмотреть, например, в [7, 29]. Перечислим только те из возможностей этой разметки, которые наверняка пригодятся при оформлении текстовой части отчета. В примерах ниже сначала приводится текст, который набирается и отображается в ячейке Markdown в процессе редактирования, а затем — результат выполнения этой ячейки.

1. Заголовки.

Markdown	Результат
# 1 уровень ## 2 уровень ### 3 уровень #### 4 уровень	1 уровень 2 уровень 3 уровень 4 уровень

2. Начертание шрифтов.

Markdown	Результат
обычный *курсив* **полужирный** ***полужирный курсив***	обычный *курсив* **полужирный** ***полужирный курсив***

3. Разрыв строки.

Язык разметки Markdown игнорирует разрывы строк, созданные в ячейке нажатием клавиши Enter. Вот какой результат получится при наборе стихотворных строк:

Markdown	Результат
Вот моя деревня: Вот мой дом родной; Вот качусь я в санках По горе крутой;	Вот моя деревня: Вот мой дом родной; Вот качусь я в санках По горе крутой;

Есть три основных варианта «ручного» разрыва строки в нужном месте:

- строку следует закончить двумя пробелами;

- строку следует завершить симолом \, после которого не должно быть

пробелов;
- учитывая, что Markdown поддерживает возможности языка разметки HTML, строку следует завершить тегом `<br>`.

Все эти варианты использованы в примере ниже

Markdown	Результат
Вот моя деревня: Вот мой дом родной;\ Вот качусь я в санках По горе крутой;	Вот моя деревня: Вот мой дом родной; Вот качусь я в санках По горе крутой;

4. Использование таблиц (это неудобно, да!).

Markdown

```
|Вызов функции |Значение функции|
|:–|–:|
|`findSubstr(s, 'загадка', -5, 'физики', 5)`|-2,
|`findSubstr(s, 'загадка', 5, 'физики', 5) `|-1,
|`findSubstr(s, 'загадка', 58, 'физика', 5)`|-1,
|`findSubstr(s, 'мы', 25, 'не', 50)`|104, 'мы понимаем'|
```

Результат

Вызов функции	**Значение функции**
`findSubstr(s, 'загадка', -5, 'физики', 5)`	-2,"
`findSubstr(s, 'загадка', 5, 'физики', 5)`	-1,"
`findSubstr(s, 'загадка', 58, 'физика', 5)`	-1,"
`findSubstr(s, 'мы', 25, 'не', 50)`	104, 'мы понимаем '

5. Фрагменты кода.

Очень важно проследить, чтобы все команды языка Python в тексте Markdown-ячейки обязательно были выделены именно стилем *код*. Для этого используется символ «обратная кавычка» (`).

Markdown	Результат
Для вывода результата на экран удобно использовать f-строки, например, \`print(f"x = {x}")\`	Для вывода результата на экран удобно использовать f-строки, например, `print(f"x = {x}")`

В Markdown-ячейку можно вставить и целый блок кода на Python.

Markdown

````
```python
def sign(x):
    if x<0:
        return -1
    elif x>0:
        return 1
    else:
        return 0
```
````

Результат

```python
def sign(x):
    if x<0:
        return -1
    elif x>0:
        return 1
    else:
        return 0
```

6. Формулы в нотации LaTeX.

Основная информация о возможностях системы компьютерной верстки LaTeX может быть найдена в [4, 9]. Здесь ограничимся несколькими примерами.

Фрагмент кода	Результат
`$x^2$`	x^2
`$x^{y+z}$`	x^{y+z}
`$\frac{x}{y+z}$`	$\frac{x}{y+z}$
`$\sin x$`	$\sin x$
`$\cos x$`	$\cos x$
`$\lg x$`	$\lg x$
`$\ln x$`	$\ln x$
`$\sqrt{x+y}$`	$\sqrt{x+y}$
`$\tan x$`	$\tan x$
`${\rm tg}\,x$`	${\rm tg}\,x$

Более сложные примеры.

Код:

```
$$\frac{e^{3x}+\cos x}{1-x^5\sin x+e^x\,{\rm tg}\, x}$$
```

Результат:

$$\frac{e^{3x}+\cos x}{1-x^5\sin x+e^x\,{\rm tg}\, x}$$

Код:

```
$$g(0)=\sin\left(\pi\frac{\lg 5}{2}\right)\approx0.8902$$
```

Результат:

$$g(0)=\sin\left(\pi\frac{\lg 5}{2}\right)\approx0.8902$$

Код:

```
$$f(n)=\sqrt{1+\sqrt{2+\sqrt{3+\ldots+\sqrt{n}}}}$$
```

Результат:

$$f(n)=\sqrt{1+\sqrt{2+\sqrt{3+\ldots+\sqrt{n}}}}$$

В завершение этого подраздела приведем образец оформления первой ячейки файла с отчетом по индивидуальному заданию и возможную реализация этого оформления в среде Jupyter Notebook.

<table><tr><td align="center">ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
Институт математики, механики и компьютерных наук
имени И.И. Воровича

ИНДИВИДУАЛЬНАЯ РАБОТА №1
«Функции и строки»
по курсу
«Технологии программирования и компьютерный практикум»
Вариант 15.
 ВЫПОЛНИЛ: студент 7 группы 1 курса
Иванов Олег Александрович
Ростов-на-Дону, 2022</td></tr></table>

```
<h3 align="center">
    ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ <br>
    Институт математики, механики и компьютерных наук
    имени И. И. Воровича</h3>
<p style="text-align:center; font-size:18px">
    ИНДИВИДУАЛЬНАЯ РАБОТА № 1 <br>
    «Функции и строки» <br>
    по курсу «Технологии программирования
    и компьютерный практикум»</p>
<h4 align="center">Вариант 15.</h4>
<div style="height:25px"></div>
<p style="text-align:right">
ВЫПОЛНИЛ: студент 7 группы 1 курса <br>
Иванов Олег Александрович</p>
<div style="height:30px"></div>
<p style="text-align:center">Ростов-на-Дону, 2022</p>
```

3 Индивидуальная работа «Численное исследование графика функции»

Исследование поведения функции на отрезке — весьма распространенная задача теоретической и прикладной математики. В случае функций, дифференцируемых нужное количество раз, это исследование может быть строго выполнено средствами математического анализа. Однако, встречающиеся на практике функции часто не являются достаточно гладкими, или не могут быть записаны в явном виде, или их аналитические выражения являются слишком сложными для дифференцирования и последующего нахождения нулей производной. Поэтому распространенным способом нестрогого (приближенного) анализа, который часто может дать достаточно много полезной информации, является численное исследование. Функция при этом заменяется достаточно большим набором точек ее графика, среди которых уже ищется максимальное значение, экстремумы, участки возрастания и т. д. Именно такой анализ и будет целью данного индивидуального задания.

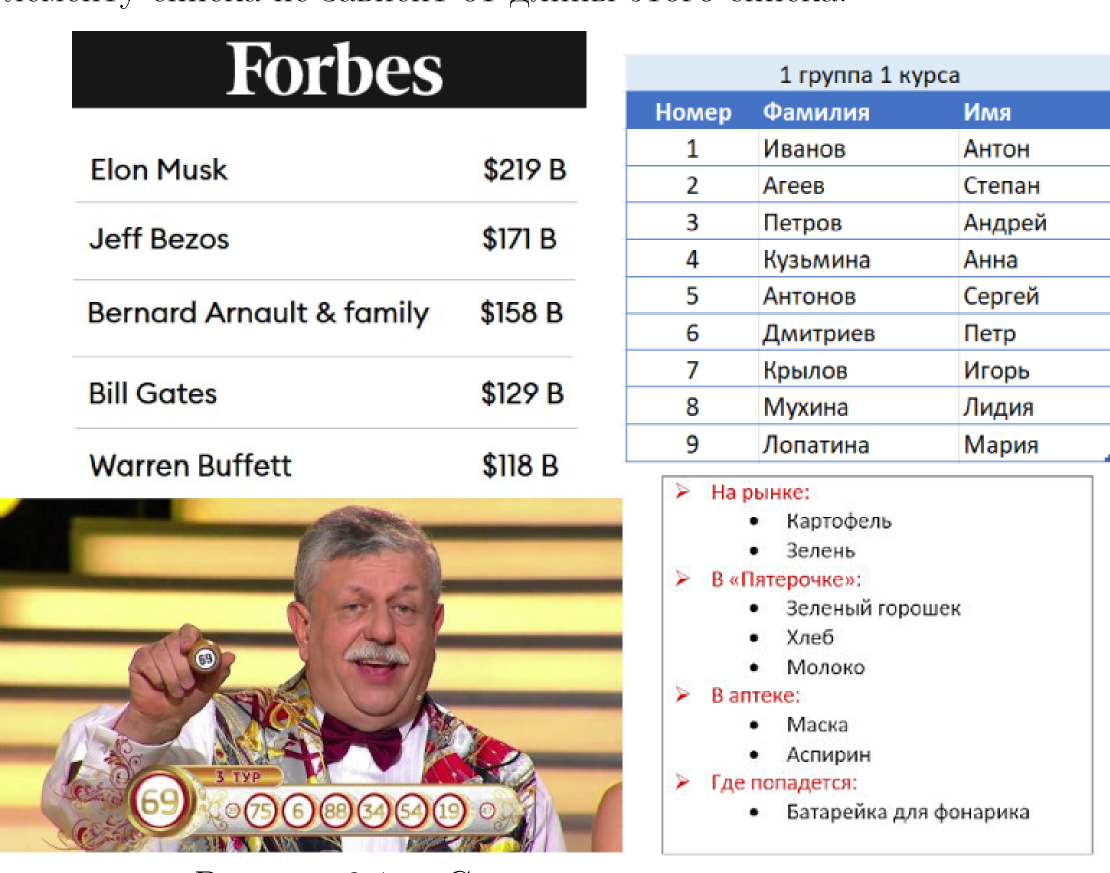
3.1 Списки и кортежи

3.1.1 Определение и примеры списков

Список — объект типа `list` — это очень важная структура данных, используемая в большом количестве языков программирования. Он представляет собой упорядоченный изменяемый набор объектов произвольного типа. Несмотря на несколько неформальный характер приведенного определения, многие слова в нем очень важны. *Упорядоченность* в данном случае означает, что элементы списка идут один за другим, то есть пронумерованы, или проиндексированы целыми числами. *Изменяемость* означает, что в отличие от, например, строки, каждый

элемент списка может быть заменен другим или вообще удален, а сам список может быть расширен на любое количество элементов. И, наконец, *произвольность* типа элемента списка означает, что список может быть *неоднородным*, он может включать в свой состав объекты любой природы, в том числе и другие списки.

Аналогом списков Python в других языках программирования являются массивы и связанные списки. В отличие от массивов, списки являются неоднородными, а в отличие от связных списков доступ к элементам объектов `list` осуществляется за константное время, т. е. время доступа к элементу списка не зависит от длины этого списка.



1 группа 1 курса		
Номер	**Фамилия**	**Имя**
1	Иванов	Антон
2	Агеев	Степан
3	Петров	Андрей
4	Кузьмина	Анна
5	Антонов	Сергей
6	Дмитриев	Петр
7	Крылов	Игорь
8	Мухина	Лидия
9	Лопатина	Мария

Рисунок 3.1 — Списки в окружающем мире

Разумеется, списки возникли не в программировании, они постоянно встречаются нам в повседневной жизни. На рис. 3.1 приведены некоторые примеры списков: список Форбс самых богатых людей мира, список студенческой группы (возможно, будущих самых богатых людей мира),

список выпавших номеров лотереи, список покупок. Создадим все эти списки с помощью языка Python. Для этого список объектов, составляющих список, нужно перечислить через запятую и заключить в квадратные скобки.

In [1]:
```
# список группы (список строк)
['Иванов Антон', 'Петров Андрей', 'Агеев Степан',
 'Кузьмина Анна', 'Антонов Сергей', 'Дмитриев Петр',
 'Крылов Игорь', 'Мухина Лидия',  'Лопатина Мария']
```

Out[1]:
```
['Иванов Антон',
 'Петров Андрей',
 'Агеев Степан',
 'Кузьмина Анна',
 'Антонов Сергей',
 'Дмитриев Петр',
 'Крылов Игорь',
 'Мухина Лидия',
 'Лопатина Мария']
```

In [2]:
```
# список победных номеров (список целых чисел)
[69, 75, 6, 88, 34, 54, 19]
```

Out[2]:
```
[69, 75, 6, 88, 34, 54, 19]
```

In [3]:
```
# Список Форбс - люди и их деньги
['Elon Musk', 219, 'Jeff Bezos', 171,
 'Bernard Arnault & family', 158,
 'Bill Gates', 129, 'Warren Buffett', 118]
```

```
Out[3]: ['Elon Musk',
         219,
         'Jeff Bezos',
         171,
         'Bernard Arnault & family',
         158,
         'Bill Gates',
         129,
         'Warren Buffett',
         118]
```

```
In [4]: # список покупок (список списков)
        [['Картофель','Зелень'],
         ['Зеленый горошек', 'Хлеб', 'Молоко'],
         ['Маска', 'Аспирин'], ['Батарейка']]
```

```
Out[4]: [['Картофель', 'Зелень'],
         ['Зеленый горошек', 'Хлеб', 'Молоко'],
         ['Маска', 'Аспирин'],
         ['Батарейка']]
```

Доступ к элементу списка или к части списка осуществляется аналогично доступу к элементам строки. Как и в случае строк можно использовать срезы, которые в этом случае представляют собой новые списки. В примерах ниже сначала создан список `a`, потом продемонстрирован доступ к его элементам, а также к элементам его элементов:

```
In [5]: a = [1, 2, 3, [10, 20], [4, 5, 6]]
        a[0]
```

```
Out[5]: 1
```

In [6]:
```
a[3]
```

Out[6]: [10, 20]

In [7]:
```
a[4][1]
```

Out[7]: 5

In [8]:
```
a[1:4]
```

Out[8]: [2, 3, [10, 20]]

In [9]:
```
a[::-1]
```

Out[9]: [[4, 5, 6], [10, 20], 3, 2, 1]

Обратите внимание на последний пример: с помощью среза можно обратить список аналогично обращению строки, при этом содержащиеся в нем списки, как и любые другие элементы, сами не изменяются.

В отличие от строки, список — это изменяемый объект. Любой его элемент, в том числе и составной, может быть изменен:

In [10]:
```
a[0] = 0
a[4] = [5,6,7,8]
a
```

Out[10]: [0, 2, 3, [10, 20], [5, 6, 7, 8]]

Отметим, что новое значение можно присвоить не только отдельному элементу списка, но и целому срезу:

In [11]:
```
a[0:3] = [7, 8, 9]
a
```

Out[11]: [7, 8, 9, [10, 20], [5, 6, 7, 8]]

При этом нужно внимательно следить за размерами списков в левой и правой части, потому что язык Python допускает замену среза одной длины списком совсем другой длины и не выдает в этом случае никаких предупреждений.

3.1.2 Способы создания списков

Как уже неоднократно было продемонстрировано выше, списки можно создать **вручную**:

In [12]:
```
a = [10, 20, 30, 40, 50]
a
```

Out[12]: [10, 20, 30, 40, 50]

Этот способ, очевидно, не годится, если речь идет о создании больших списков. Возможным вариантом является **последовательное добавление элементов**, основанное на использовании метода append, добавляющего очередное значение в конец списка:

In [13]:
```
a = [] # пустой список
for i in range(1,6):
    a.append(i*10)
a
```

Out[13]: [10, 20, 30, 40, 50]

Более удобным, наглядным и, что немаловажно, быстрым способом создания списков является их **генерирование**. Используемые при этом конструкции на английском языке называются List Comprehensions, но, к сожалению, единого удачного перевода на русский язык это словосочетание пока не нашло.

В настоящем пособии будет употребляться терминология «создание списка на основе генератора». Схема такого создания выглядит следующим образом:

```
new_list = [expression for x in iterable]
```

Здесь

- new_list — создаваемый список;
- expression — любое выражение (формула, функция и т. п.), возвращающее значение;
- iterable — строка, список, объект range или любой другой итерируемый объект, т. е. объект, который может по очереди возвращать конечное число значений (элементов);
- x — очередной элемент, генерируемый объектом iterable. Как правило, expression — это некоторая функция параметра x.

Предыдущий список может быть получен следующим образом:

```
In [14]: a = [i*10 for i in range(1,6)]
         a

Out[14]: [10, 20, 30, 40, 50]
```

В генератор списка можно добавлять условие по следующей схеме:

```
new_list = [expression for x in iterable if condition(x)]
```

Тогда выражение expression будет вычисляться и добавляться в список только для тех значений x, для которых выполняется условие, т. е. выражение condition(x) является истинным.

В качестве примера создадим список двузначных чисел, делящихся на 5 и не делящихся на 3:

```
In [15]: a = [x for x in range(10,100) if x%5==0 and x%3!=0]
         a

Out[15]: [10, 20, 25, 35, 40, 50, 55, 65, 70, 80, 85, 95]
```

Как и в случае строк, новый список может быть получен **сложением** двух списков (слагаемые объединяются в один список) и **умножением** списка на натуральное число:

In [16]:
```
a = [1,2,3]
b = [-1, -2, -3]
print(a+b)
print(a*5)
```

```
[1, 2, 3, -1, -2, -3]
[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Еще один распространенный способ создания списка основан на использовании метода `split` строковых объектов. В результате его работы создается список подстрок, из которых состояла исходная строка:

In [17]:
```
s = 'Шла  Саша по шоссе  и сосала сушку '
a = s.split()
print(a)
```

```
['Шла', 'Саша', 'по', 'шоссе', 'и', 'сосала', 'сушку']
```

In [18]:
```
ip_address = '195.208.253.2'
blocks = ip_address.split('.')
print(blocks)
```

```
['195', '208', '253', '2']
```

3.1.3 Основные методы списков

В таблице ниже приведено краткое описание основных методов, которые «умеет выполнять» объект типа `list`.

Функция или метод	Назначение
a.append(x)	Элемент x добавляется в конец списка a.
a.extend(L)	Список (точнее, итерируемая последовательность) L добавляется в конец списка.
a.insert(i, x)	Вставка на i-е место в список a элемента x; список «раздвигается».
a.remove(x)	Удаление первого элемента в списке a, имеющего значение x. Если такого элемента не существует, возникает ошибка ValueError.
a.pop(i)	Удаление элемента с индексом i или последнего элемента, если индекс не указан. Значением функции является значение этого элемента.
a.index(x, start , end)	Определение индекса первого (с начала списка или от позиции start) элемента со значением x. По умолчанию ищется элемент от начала и до конца списка.
a.count(x)	Вычисление количества элементов списка a, имеющих значение x.
a.sort()	Упорядочивание списка на месте.
sorted(a)	Создание упорядоченного списка.
a.reverse()	Разворачивание списка на месте.
reversed(a)	Создание развернутого списка.
a.copy()	Создание поверхностной копии списка.
a.clear()	Удаляет все элементы списка. Не имеет возвращаемого значения.

Обратите внимание: методы sort и reverse не возвращают значения, а изменяют существующий список!

3.1.4 Кортежи

Кортеж — объект типа `tuple` — в Python во многом похож на список. Он тоже является индексируемым набором объектов произвольного типа. Основное отличие, резко уменьшающее по сравнению со списком количество допустимых операций, — это неизменяемость. Один раз созданный, кортеж не может быть изменен: не допускается (с некоторой оговоркой) изменять отдельные его элементы, нельзя удалять или добавлять элементы в кортеж. Что касается оговорки, то она связана с тем, что элементами кортежа могут быть изменяемые объекты, например, списки. В этом случае мы можем изменять отдельные элементы таких списков.

Синтаксическое отличие кортежа от списка состоит в том, что содержащиеся в нем значения заключаются не в квадратные, а в круглые скобки.

Ниже приведен ряд примеров, демонстрирующих сходство и отличия объектов типа список и типа кортеж.

```
In [1]: a = [1,2,3] # список
        t = (1,2,3) # кортеж
```

```
In [2]: type(a)
```

```
Out[2]: list
```

```
In [3]: type(t)
```

```
Out[3]: tuple
```

```
In [4]: a[0]
```

```
Out[4]: 1
```

```
In [5]: t[0]
```

```
Out[5]: 1
```

```
In [6]: a[0] = 10
        a
```

```
Out[6]: [10, 2, 3]
```

```
In [7]: t[0] = 10
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-88963aa635fa> in <module>
----> 1 t[0] = 10

TypeError: 'tuple' object does not support item assignment
```

```
In [8]: # пустой список
        a = []
        print(a, type(a))
```

```
[] <class 'list'>
```

```
In [9]: # пустой кортеж
        t = ()
        print(t, type(t))
```

```
() <class 'tuple'>
```

```
In [10]: # кортеж из одного элемента? нет!
         t = (5)
         print(t, type(t))
```

```
5 <class 'int'>
```

```
In [11]: # если нужно, то вот так:
t = (5,)
print(t, type(t))
```

```
(5,) <class 'tuple'>
```

3.2 Основные определения

Точкой графика функции $y = f(x)$ будем называть **кортеж** вещественных чисел $(x, f(x))$. Число x при этом будем называть *абсциссой точки*, а $y = f(x)$ — ее *ординатой*.

Числовыми данными графика будем называть **список** точек графика.

Локальным максимумом графика называется точка графика, ордината (координата y) которой больше ординат соседних точек (у первого и последнего элементов — по одному соседу, у остальных — по два).

Локальным минимумом графика называется точка графика, ордината которой меньше таких же ординат соседних точек.

Максимумом графика называется точка графика, ордината которой не меньше ординат всех остальных точек.

Минимумом графика называется точка графика, ордината которой не превосходит ординат всех остальных точек.

Участком возрастания графика называется **список** последовательных точек графика $[(x_k, y_k), k = m, \ldots, n]$, удовлетворяющих следующим условиям:

1. Для всех точек списка, кроме последней, выполнено условие возрастания: $y_i < y_{i+1}, i = m, \ldots, n-1$.
2. Последняя точка списка, (x_n, y_n), является либо конечной точкой графика, т.е. $x_n = b$, либо точкой, в которой нарушено условие возрастания, т.е. $y_n \geqslant y_{n+1}$.
3. Первая точка, (x_m, y_m), является либо начальной точкой графика,

т.е. $x_m = a$, либо точкой, у левого соседа которой нарушено условие возрастания, т.е. $y_{m-1} \geqslant y_m$.

Участком убывания графика называется **список** последовательных точек графика $[(x_k, y_k), k = m, \ldots, n]$, удовлетворяющих следующим условиям:

1. Для всех точек списка, кроме последней, выполнено условие убывания: $y_i > y_{i+1}, i = m, \ldots, n-1$.
2. Последняя точка списка, (x_n, y_n), является либо конечной точкой графика, т.е. $x_n = b$, либо точкой, в которой нарушено условие убывания, т.е. $y_n \leqslant y_{n+1}$.
3. Первая точка, (x_m, y_m), является либо начальной точкой графика, т.е. $x_m = a$, либо точкой, у левого соседа которой нарушено условие убывания, т.е. $y_{m-1} \leqslant y_m$.

3.3 Постановка задачи

Задание 1. Используя информацию из таблиц 3.2—3.5, сгенерировать числовые данные графика функции $y = f(x)$ на отрезке $[a, b]$ на основе N равномерно распределенных точек.

Задание 2. В соответствии с номером варианта и таблицей 3.6 описать на языке Python требуемые функции, определить и вывести на экран необходимые характеристики графика.

Задание 3. Используя представленный в 3.5 график функции $f(x)$, проанализировать правильность проведенных расчетов.

Таблица 3.2 — Выбор функций, отрезка построения графика и вычисляемых характеристик

№	$g(x)$ Табл. 3.3	$h(x)$ Табл. 3.4	$f(x)$ Табл. 3.5	$[a, b]$	N	Определяемые характеристики функции по табл. 3.6
0	7	7	7	[3.4, 3.8]	300	4, 13, 21
1	6	5	6	[1.9, 2.1]	400	3, 5, 10, 14, 18
2	4	5	3	[0.0, 1.5]	800	2, 5, 10, 15, 20
3	1	1	6	[1.0, 3.0]	500	2, 8, 10, 17, 19
4	1	3	6	[1.5, 3.4]	650	1, 7, 11, 14, 19
5	1	2	1	[-1.0, 1.1]	400	1, 6, 9, 16, 20
6	2	6	3	[-1.1, 1.85]	300	1, 7, 10, 16, 18
7	1	6	1	[-1.0, 1.4]	450	2, 7, 9, 16, 19
8	2	2	4	[-1.5, 2.6]	600	1, 8, 12, 17, 19
9	1	5	4	[-0.1, 2.5]	450	3, 8, 9, 17, 19
10	4	1	4	[0.3, 1.7]	500	1, 8, 9, 15, 19
11	3	5	2	[-1.1, 2.0]	600	2, 6, 11, 14, 19
12	5	3	1	[2.5, 4.1]	700	2, 7, 12, 15, 18
13	3	3	5	[3.1, 4.5]	400	1, 8, 9, 16, 20
14	6	3	4	[2.1, 4.8]	450	3, 7, 11, 15, 20
15	6	1	1	[1.1, 3.5]	500	3, 6, 9, 16, 19
16	4	4	2	[-0.6, 0.6]	300	3, 5, 12, 17, 18

17	2	4	6	[-2.1, 1.4]	550	3, 7, 10, 17, 19
18	6	4	4	[-2.1, 2.5]	600	3, 6, 9, 17, 20
19	3	6	2	[-1.5, 1.5]	600	1, 7, 12, 17, 20
20	6	2	2	[1.5, 2.85]	500	2, 8, 11, 16, 19
21	3	2	6	[-1.7, 1.5]	450	1, 7, 10, 17, 19
22	2	5	5	[-1.5, 1.35]	350	2, 5, 11, 17, 20
23	3	1	5	[0.0, 3.5]	450	2, 8, 9, 17, 20
24	6	6	6	[-0.6, 1.65]	550	3, 8, 10, 14, 20
25	3	4	1	[-1.1, 1.64]	500	1, 5, 10, 17, 18
26	5	2	5	[0.7, 1.55]	400	1, 5, 9, 15, 18
27	1	4	3	[-3.0, 1.28]	600	3, 5, 9, 16, 18
28	4	6	4	[0.3, 1.8]	500	3, 8, 12, 16, 20
29	4	2	3	[-0.65, 0.55]	300	2, 7, 12, 17, 20
30	4	3	3	[-0.5, 2.55]	450	1, 5, 10, 15, 18

Таблица 3.3 — $g(x)$

	$\boldsymbol{g(x)}$
1	$e^{\sqrt{1-\sin 10x}}$
2	$\ln(1+\cos^6 x)$
3	$e^{\sqrt{1-\sin x^2}}$
4	$\ln(2+\sin x^4)$
5	$e^{\sqrt{1-\cos(x+2)}}$
6	$\ln(2-\cos^5 2x)$
7	$\cos^{10} x-\sqrt{1+(x/10)^{10}}$

Таблица 3.4 — $h(x)$

	$\boldsymbol{h(x)}$
1	$\cos(x+e^x)$
2	$x^5 e^{x/2}$
3	$\sin(1-e^x)$
4	$x^3 e^x$
5	$\sin(e^x+e^{2x})$
6	$(x^2-3)e^{2x}$
7	$3\cos(2e^x)$

Таблица 3.5 — $f(x)$

	$\boldsymbol{f(x)}$
1	$\sin(h(x))+\cos(g(x))$
2	$h(x/3)+g(3x)$
3	$h^2(x)-g^3(x/2)$
4	$\sin(h(x-1))+g(x+1)$
5	$\sin(h(x))-\cos(g(x))$
6	$\cos(h(x)-g(x))$
7	$\sqrt{1+h^2(x)g^4(x)}$

Таблица 3.6 — Характеристики графика

№	**Описание**	**Название функции**	**Количество баллов**
1	Абсцисса точки максимума[3]	xMaxFirst	1
2	Ордината точки максимума[4]	yMaxLast	1
3	Ордината точки минимума[3]	yMinFirst	1
4	Абсцисса точки минимума[4]	xMinLast	1
5	Количество локальных максимумов	nMin	1
6	Количество локальных минимумов	nMax	1
7	Список точек — локальных максимумов	listMax	1
8	Список точек — локальных минимумов	listMin	1
9	Сумма максимального и минимального значений функции на отрезке	sumMM	1
10	Разность максимального и минимального значений функции на отрезке	amplitude	1
11	Произведение максимального и минимального значений функции на отрезке	prodMM	1

[3]Если их несколько — выбрать первый
[4]Если их несколько — выбрать последний

12	Среднее арифметическое максимального и минимального значения функции на отрезке	meanMM	1
13	Среднее арифметическое всех ординат числовых данных графика	meanValue	1
14	Список точек, образующих первый возрастающий участок графика	firstUp	2
15	Список точек, образующих последний возрастающий участок графика	lastUp	2
16	Список точек, образующих первый убывающий участок графика	firstDown	2
17	Список точек, образующих последний убывающий участок графика	lastDown	2
18	Количество участков возрастания графика	nUp	2
19	Количество участков убывания графика	nDown	2
20	Самый большой по протяженности участок возрастания[4]	longUp	2
21	Самый большой по протяженности участок убывания[3]	longDown	2

3.4 Образец выполнения задания

Задание 1. По таблице 3.2 находим, что функции h, g и f определяются седьмыми строчками в соответствующих таблицах. Таким образом

$$g(x) = \cos^{10} x - \sqrt{1+(x/10)^{10}}, \quad h(x) = 3\cos(2e^x),$$

$$f(x) = \sqrt{1+h^2(x)g^4(x)}.$$

Используя библиотеку `math`, запишем код этих функций на языке Python:

```
1  from math import *
2  def g(x):
3      return cos(x)**10 - sqrt(1+(x/10)**10)
4  def h(x):
5      return 3*cos(2*exp(x))
6  def f(x):
7      return sqrt(1+h(x)**2*g(x)**4)
```

Для числовых данных графика — списка точек — введем переменную `data`. С учетом информации в таблице 3.2 об отрезке и количестве точек этот список может быть создан следующим образом:

```
1  a = 3.4
2  b = 3.8
3  N = 300
4  step = (b-a)/(N-1)
5  data = []
6  for i in range(N):
7      x = a + i*step
8      data.append((x,f(x)))
```

Здесь отрезок $[a, b]$ разделен на $N-1$ равную часть длины `step`. Общее количество точек в списке `data` при этом как раз равно N.

Задание 2. В соответствии с номером варианта определяем, что

необходимо описать функции для вычисления характеристик № 4, 13, 21. Все они зависят от одного аргумента — числовых данных графика, т. е. списка кортежей. Учтем это при написании кода функций. Полный текст программы имеет вид:

```
1  from math import *
2  def g(x):
3      return cos(x)**10 - sqrt(1+(x/10)**10)
4  def h(x):
5      return 3*cos(2*exp(x))
6  def f(x):
7      return sqrt(1+h(x)**2*g(x)**4)
8  a = 3.4
9  b = 3.8
10 N = 300
11 step = (b-a)/(N-1)
12 data = []
13 for i in range(N):
14     x = a + i*step
15     data.append((x,f(x)))
16 def xMinLast(d):
17     """
18     функция находит абсциссу точки минимума
19     (последнего, если их несколько)
20     """
21     xMin, yMin = d[0]
22     for i in range(1, len(d)): # нестрогое равенство
23         if d[i][1]<=yMin:      # использовано потому, что
24             xMin, yMin = d[i]  # нужен последний минимум
25     return xMin
26 def meanValue(d):
27     """
28     функция находит среднее арифметическое всех ординат
29     числовых данных графика
30     """
31     # создаем список всех ординат, находим  их сумму
32     # и делим на количество
33     return sum([d[i][1] for i in range(len(d))])/len(d)
34 def longestDown(d):
35     """
36     функция находит самый большой по протяженности
```

```
37      участок убывания графика
38      """
39      n = len(d)
40      maxDown = [] # "кандидат" на звание самого большого.
41                   # может так и остаться пустым,
42                   # если f возрастает
43      nMax = 0     # число элементов у "кандидата"
44      lDown = []   # текущий участок убывания
45      nD = 0       # его длина
46      go = False   # режим - идем ли по участку убывания
47      for i in range(n-1):
48          if d[i][1]>d[i+1][1]: # мы на участке убывания
49              if i==0 or d[i-1][1]<=d[i][1]: # его начало!
50                  lDown = [d[i]]
51                  nD = 1
52                  go = True
53              else:
54                  lDown.append(d[i]) # это продолжение
55                  nD += 1
56          else:        # мы не на участке убывания
57              if go:   # он только что закончился,
58                       # подводим итоги
59                  lDown.append(d[i])
60                  nD += 1
61                  if nD > nMax:
62                      nMax = nD
63                      maxDown = lDown[:]
64                  go = False
65      # осталось (при необходимости)
66      # добавить последнюю точку
67      if go:
68          lDown.append(d[n-1])
69          nD += 1
70          if nD > nMax:
71                      nMax = nD
72                      maxDown = lDown[:]
73      return maxDown
74  # выводим результаты работы
75  print('Координата последней точки минимума:',\
76        xMinLast(data))
77  print('''Среднее арифметическое всех ординат\n\
78  числовых данных графика:''', meanValue(data))
```

```
79 print('''Список точек самого протяженного\n\
80 участка возрастания:''')
81 for d in longestDown(data):
82     print(f'{d[0]:10.6f} {d[1]:10.6f}')
```

Результаты ее работы в Jupyter Notebook приведены ниже.

```
Координата последней точки минимума: 3.6903010033444814
Среднее арифметическое всех ординат
числовых данных графика: 1.3936892897735362
Список точек самого протяженного
участка возрастания:
  3.449498   1.090239
  3.450836   1.089963
  3.452174   1.088389
  3.453512   1.085521
  3.454849   1.081397
  3.456187   1.076098
  3.457525   1.069743
  3.458863   1.062493
  3.460201   1.054551
  3.461538   1.046157
  3.462876   1.037589
  3.464214   1.029154
  3.465552   1.021183
  3.466890   1.014019
  3.468227   1.008008
  3.469565   1.003480
  3.470903   1.000736
  3.472241   1.000034
```

Задание 3. Для того, чтобы проанализировать правильность про-

веденных расчетов, последовательно нанесем их результаты на график, представленный в задании. Выполнять это будем схематично, поскольку нас интересует правдоподобие результатов, т. е. не количественное совпадение, а качественная близость.

1) Последний минимум

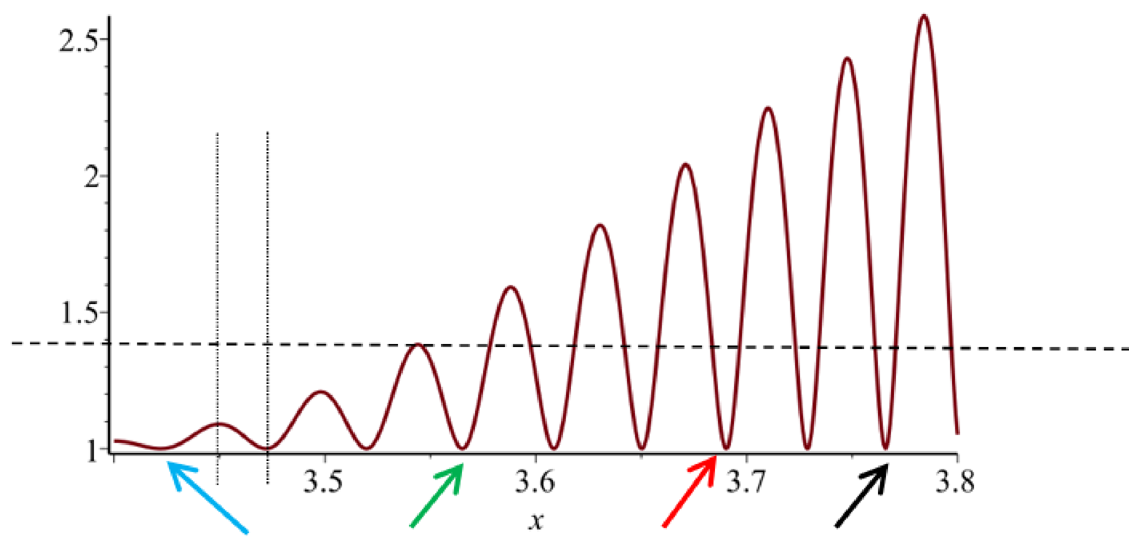


Рисунок 3.2 — Вычисленная координата $x = 3.69$. Результат отмечен красной стрелкой

Судя по графику — это действительно минимум, но не последний. Но может быть значение функции в этой точке на самом деле немного меньше, чем в остальных?

Проверим это, увеличив детальность вычислений — вместо $N = 300$ проведем вычисления с $N = 3000$. Результат: $x = 3.42$ (синяя стрелка). Координата изменилась, но не в том направлении, в котором ожидалось.

Еще раз увеличим детальность, положим $N = 30000$. Результат: $x = 3.57$ (зеленая стрелка). Дальнейшее увеличение числа N приводит к одному из предыдущих результатов.

Подобное поведение программы свидетельствует о проблемах вычислительного анализа. Скорее всего, значения функции во всех точках минимума действительно одинаковы, но из-за погрешностей вычислительного характера проверить это невозможно, потому что при вычислениях

с плавающей точкой проверить точное равенство двух чисел нельзя.

Чтобы «заставить» программу решать поставленную задачу, внесем в текст функции `xMinLast` следующее изменение: заменим сравнение «меньше или равно» сравнением «меньше или приблизительно равно», Для этого выражение

```
d[i][1] <= yMin
```

заменим другим:

```
d[i][1] < yMin or abs(d[i][1] - yMin) < epsilon
```

предварительно определив константу `epsilon` значением

```
epsilon = 1e-6
```

В этом случае увеличение числа разбиений действительно повышает точность определения последнего минимума, но для этого требуется значение N, равное 20000. Расчеты показали, что при бо́льших значениях N модифицированная функция стабильно выдает значение $x = 3.77$, что соответствует именно последнему минимуму (черная стрелка).

2) Среднее значение функции

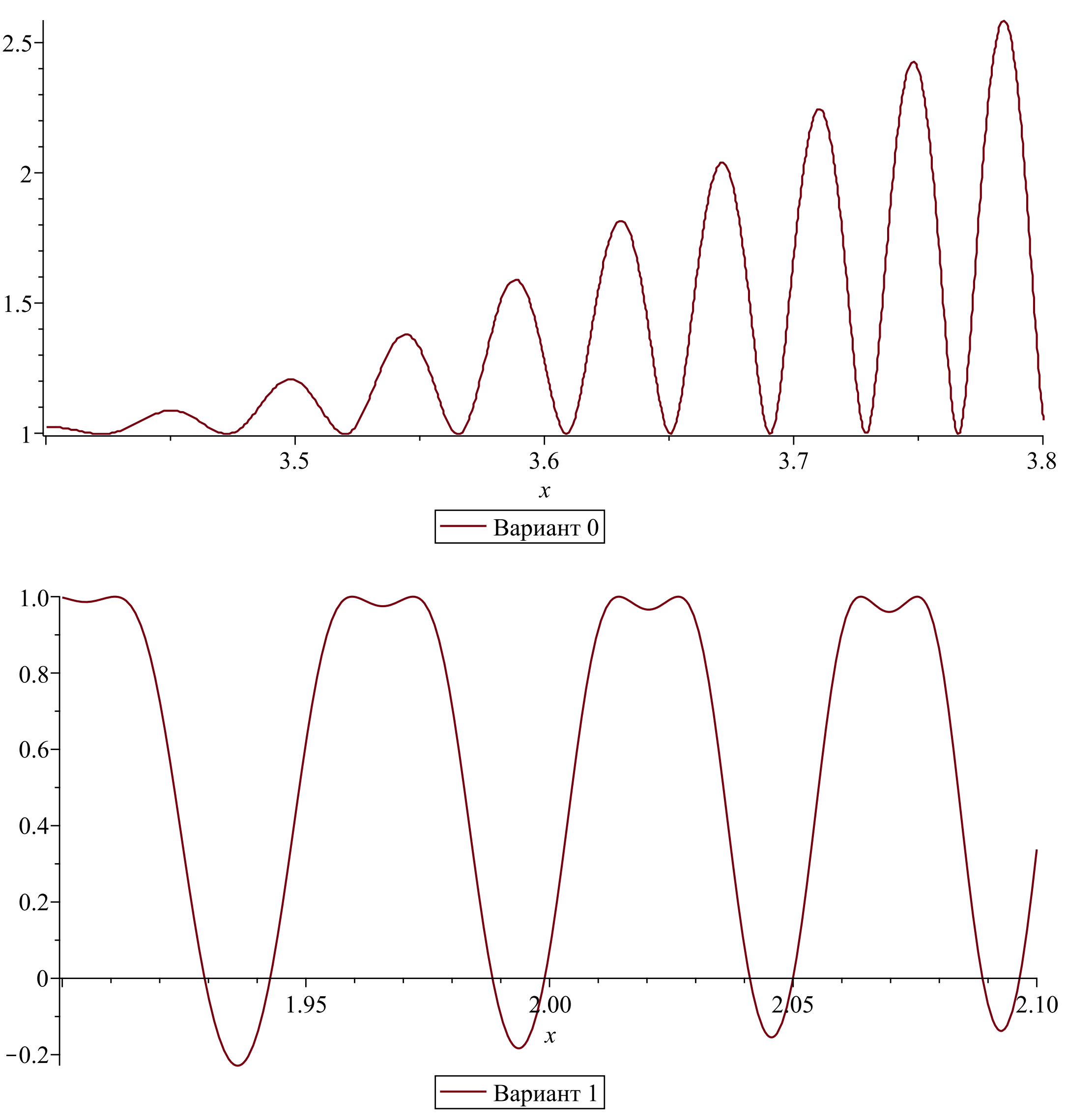
На рисунке 3.2 проведена (на глаз) горизонтальная пунктирная линия $y = 1.39$, что соответствует найденному среднему значению функции $f(x)$ на отрезке. Данное среднее значение визуально кажется вполне правдоподобным.

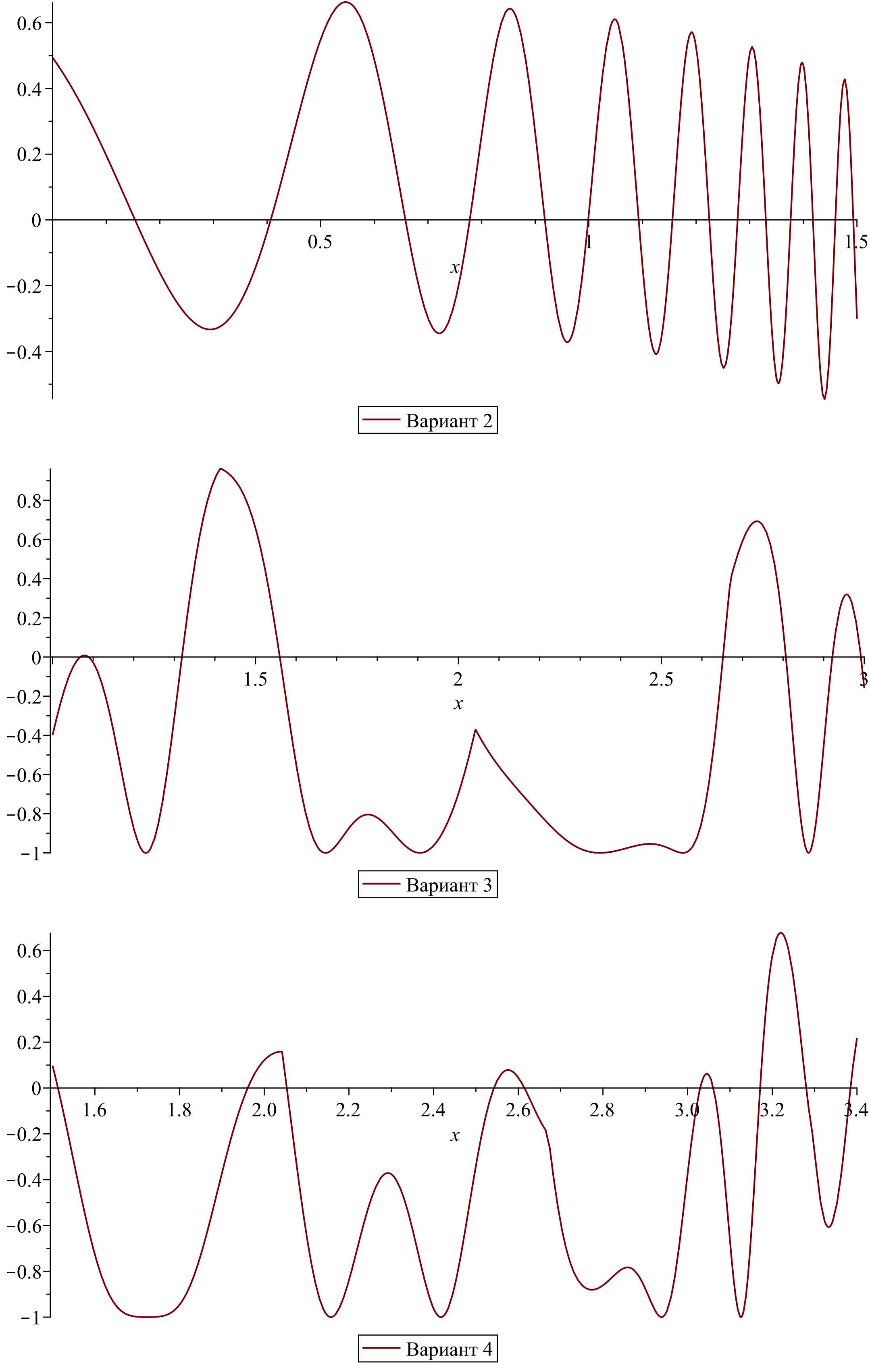
3) Наибольший участок убывания

На рисунке 3.2 вертикальными точечными линиями отмечен участок, который программа определила как участок убывания наибольшей длины. Из графика видно, что это действительно участок убывания. Что касается длины, то визуально определить ее трудно, но противоречия с графиком не обнаружено. Результаты расчетов, таким образом, подтверждены.

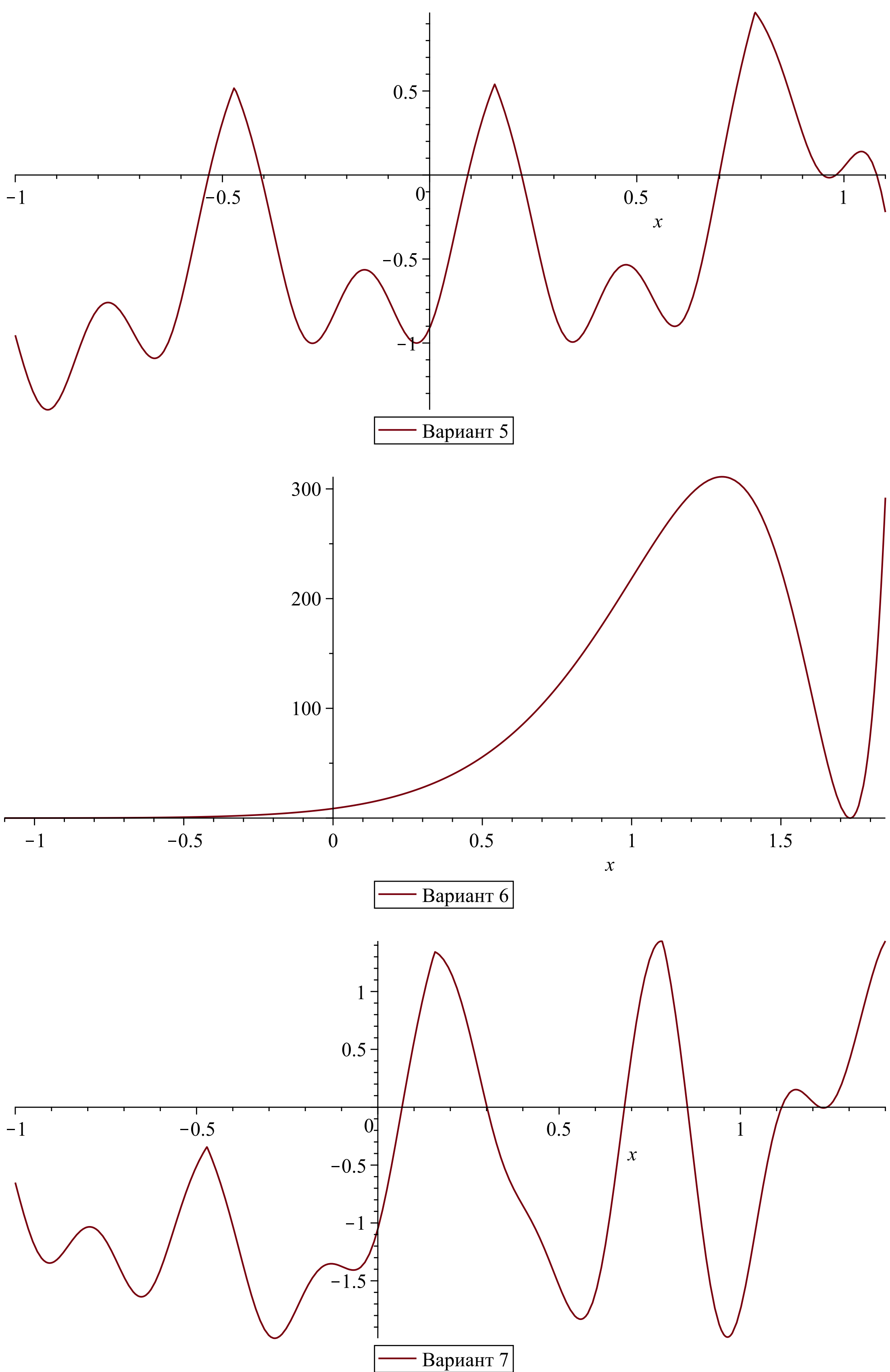
3.5 Графики исследуемых функций

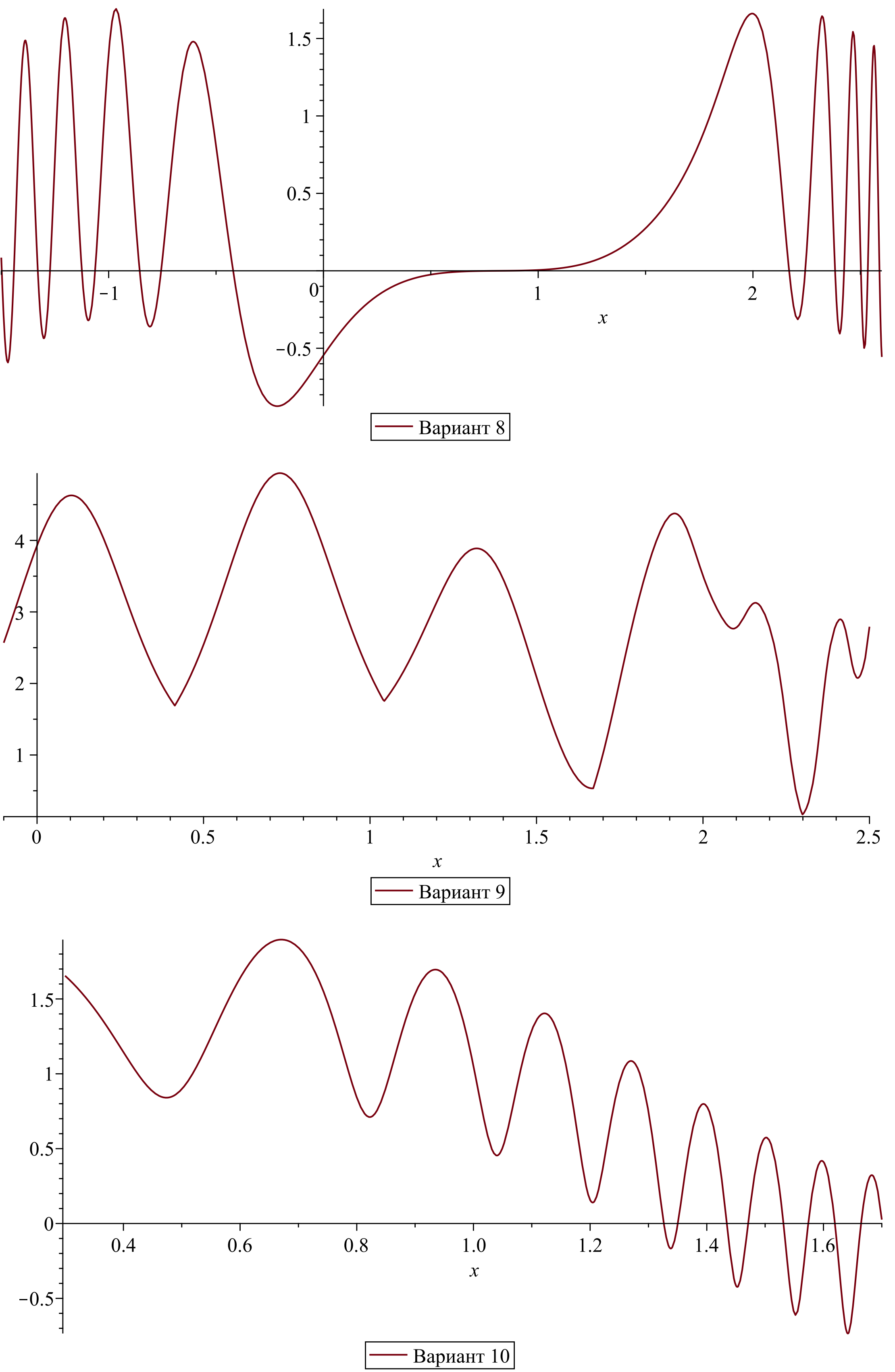
В данном разделе представлены графики функции $f(x)$ согласно вариантам из таблиц 3.2 и 3.5.

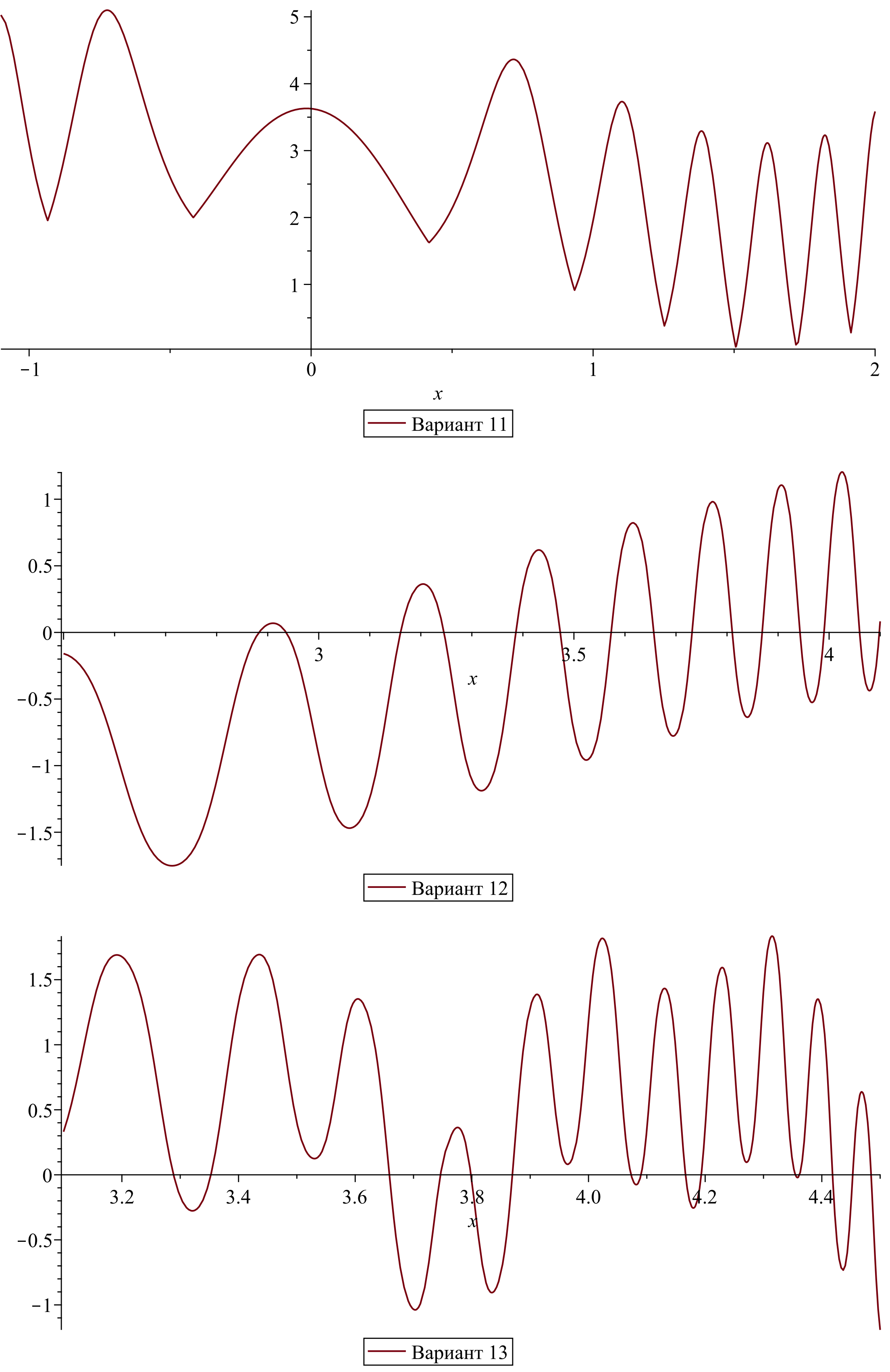


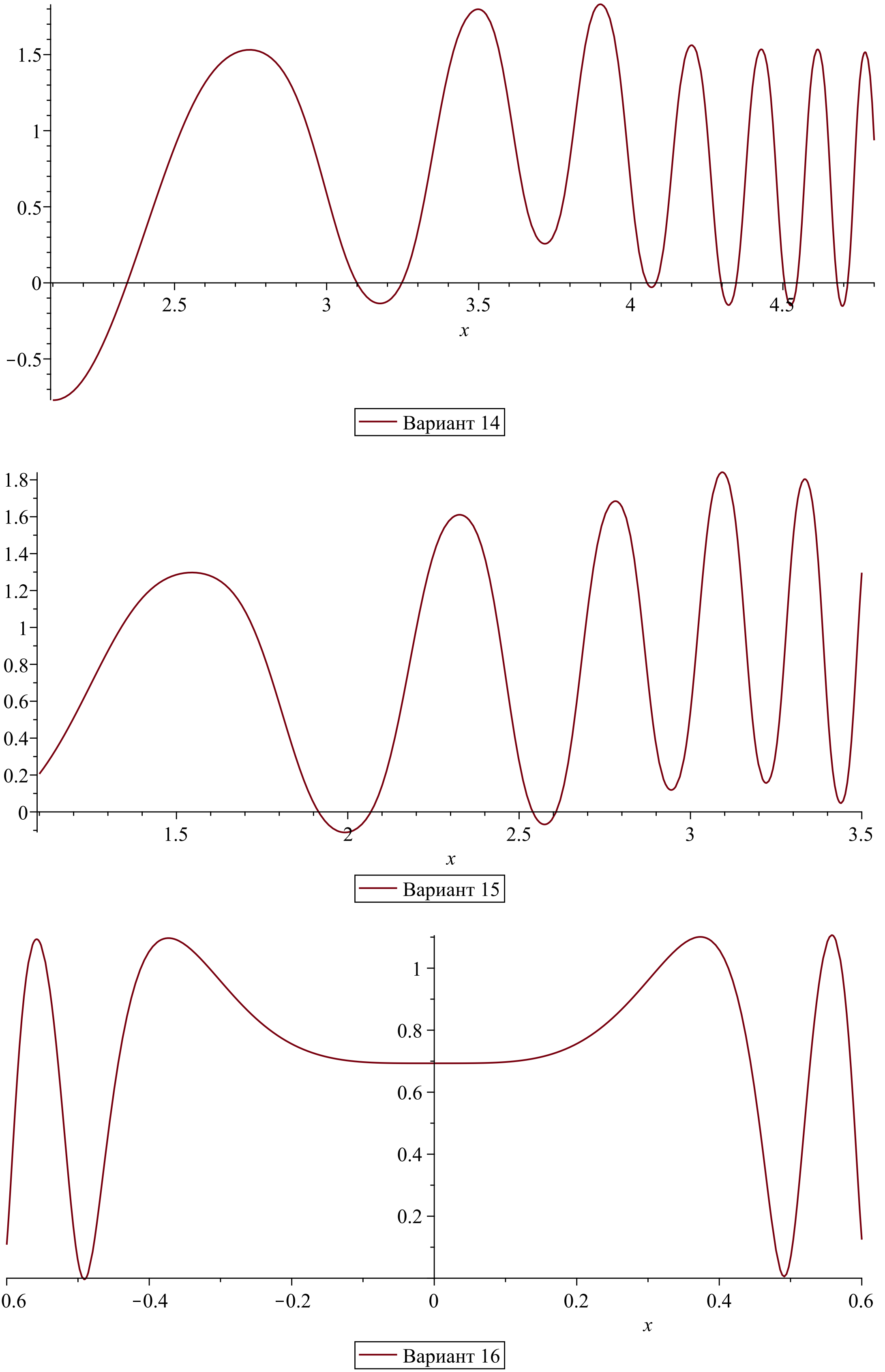


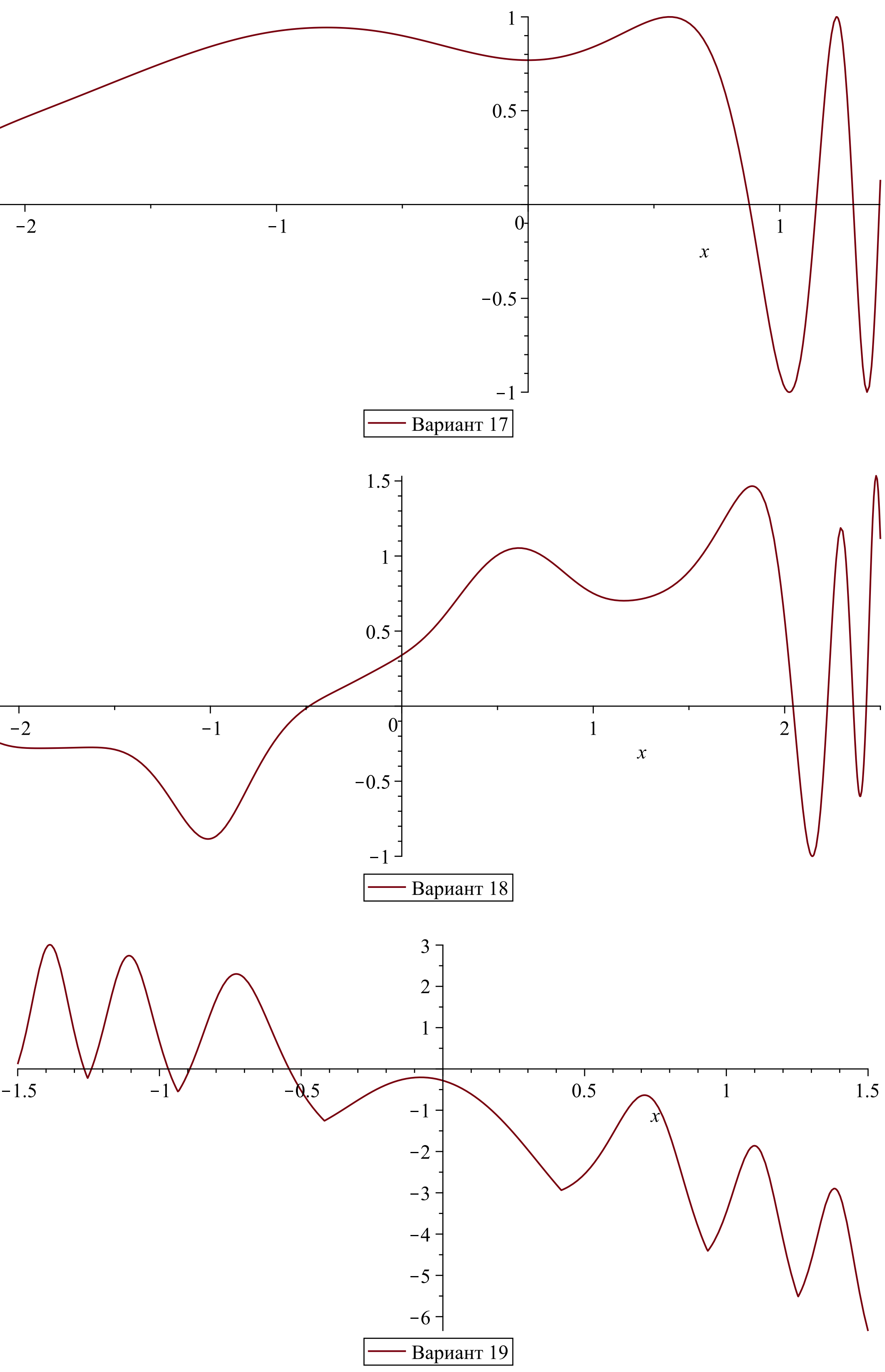
0.6
0.4
0.2
0
−0.2
−0.4
0.5
1
1.5
x
Вариант 2
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
1.5
2
2.5
3
x
Вариант 3
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
−1
1.6
1.8
2.0
2.2
2.4
2.6
2.8
3.0
3.2
3.4
x
Вариант 4

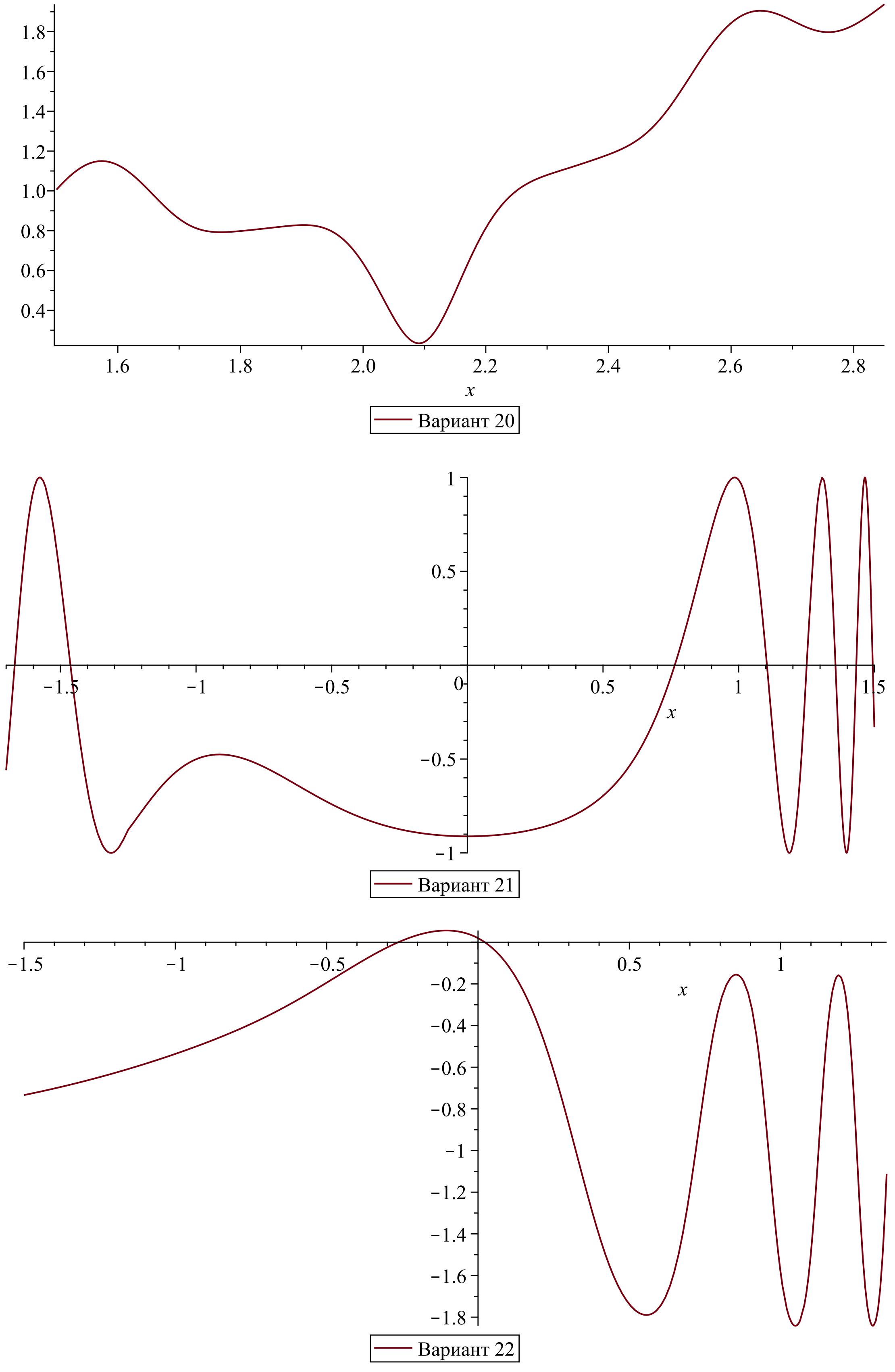

0.5
0
−0.5
−1
−1
−0.5
0.5
1
x
Вариант 5
300
200
100
−1
−0.5
0
0.5
1
1.5
x
Вариант 6
1
0.5
0
−0.5
−1
−1.5
−1
−0.5
0.5
1
x
Вариант 7

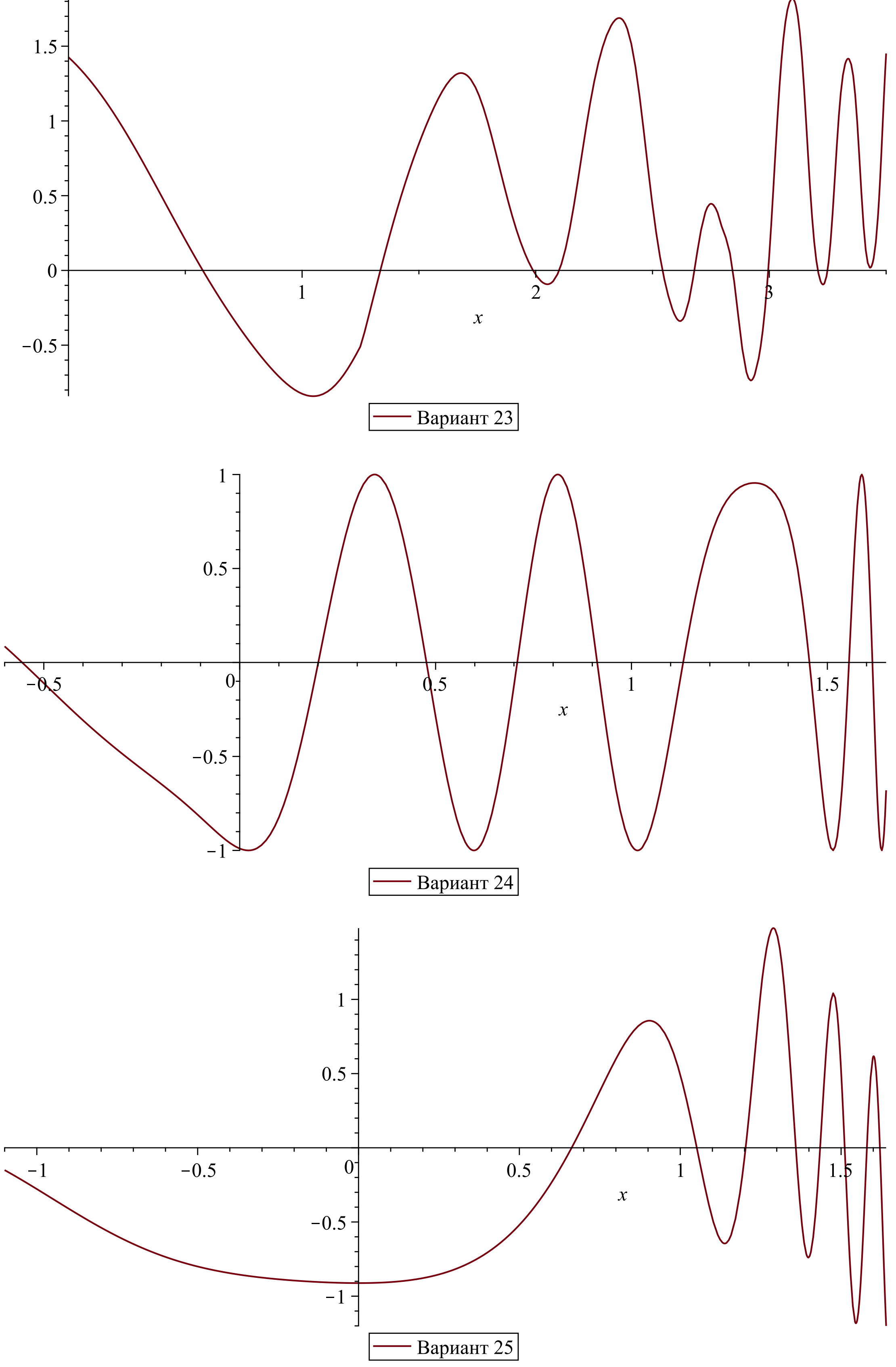

1.5
1
0.5
0
−0.5
−1
1
2
x
Вариант 8
4
3
2
1
0
0.5
1
1.5
2
2.5
x
Вариант 9
1.5
1
0.5
0
−0.5
0.4
0.6
0.8
1.0
1.2
1.4
1.6
x
Вариант 10

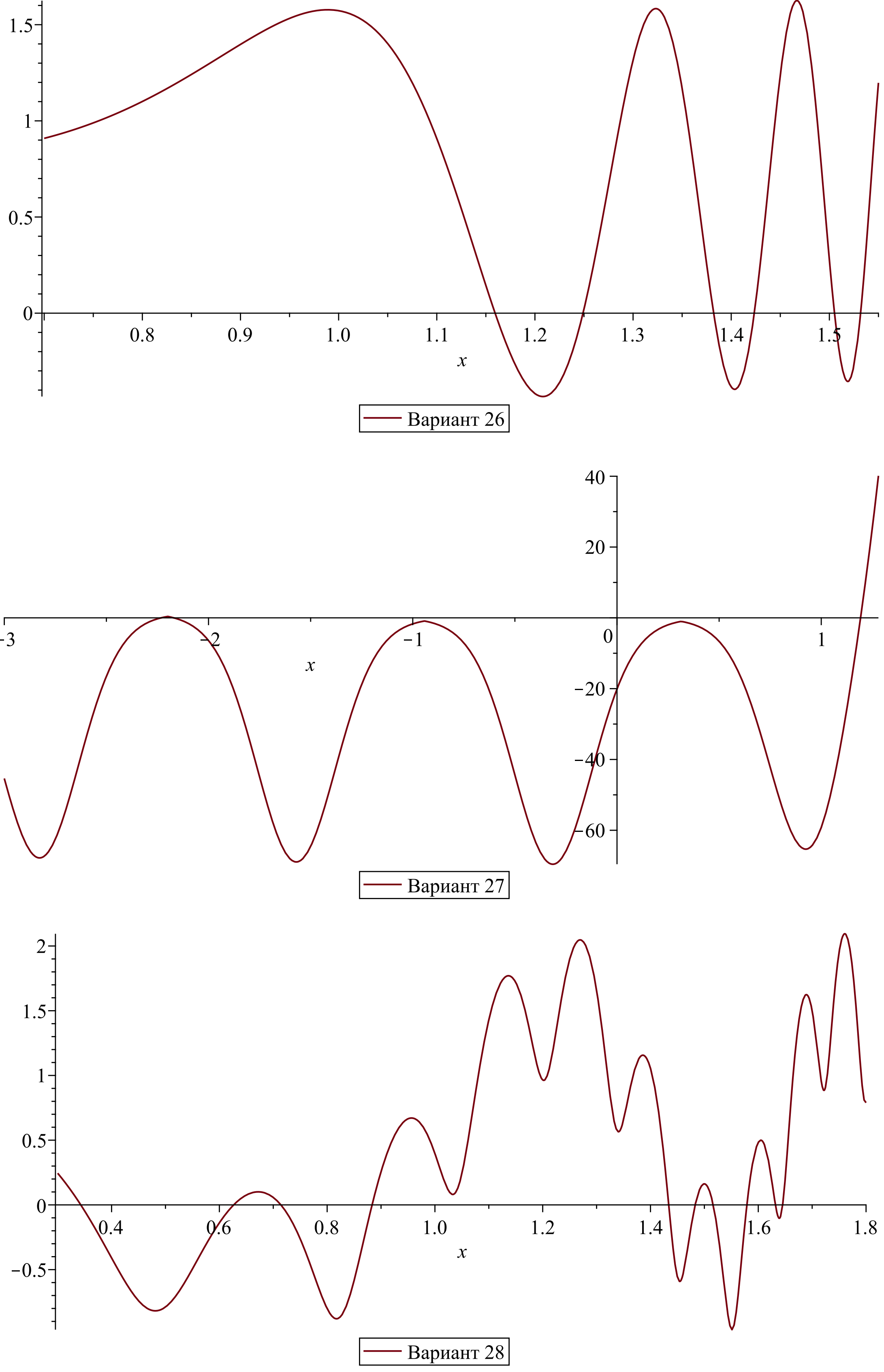

5
4
3
2
1
−1
0
1
2
x
Вариант 11
1
0.5
0
−0.5
−1
−1.5
3
3.5
4
x
Вариант 12
1.5
1
0.5
0
−0.5
−1
3.2
3.4
3.6
3.8
4.0
4.2
4.4
x
Вариант 13

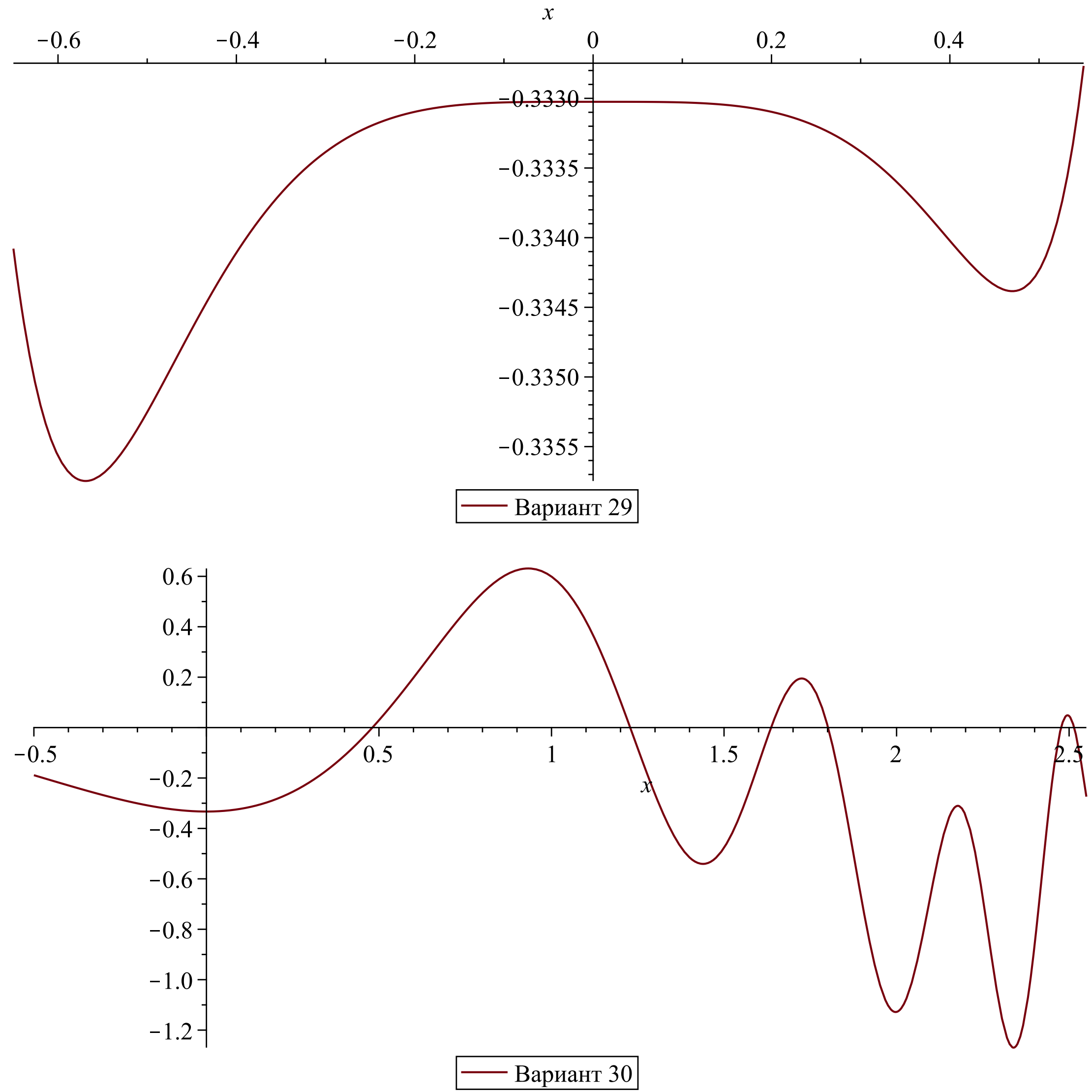

1.5
1
0.5
0
−0.5
2.5
3
3.5
4
4.5
x
Вариант 14
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
1.5
2
2.5
3
3.5
x
Вариант 15
1
0.8
0.6
0.4
0.2
−0.6
−0.4
−0.2
0
0.2
0.4
0.6
x
Вариант 16


1
0.5
−2
−1
0
1
x
−0.5
−1
Вариант 17
1.5
1
0.5
−2
−1
0
1
2
x
−0.5
−1
Вариант 18
3
2
1
−1.5
−1
−0.5
0.5
1
1.5
x
−1
−2
−3
−4
−5
−6
Вариант 19


1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
1.6
1.8
2.0
2.2
2.4
2.6
2.8
x
Вариант 20
1
0.5
−1.5
−1
−0.5
0
0.5
1
1.5
x
−0.5
−1
Вариант 21
−1.5
−1
−0.5
0.5
1
x
−0.2
−0.4
−0.6
−0.8
−1
−1.2
−1.4
−1.6
−1.8
Вариант 22


1.5
1
0.5
0
−0.5
1
2
3
x
Вариант 23
1
0.5
−0.5
0
0.5
1
1.5
x
−0.5
−1
Вариант 24
1
0.5
−1
−0.5
0
0.5
1
1.5
x
−0.5
−1
Вариант 25


1.5
1
0.5
0
0.8
0.9
1.0
1.1
1.2
1.3
1.4
1.5
x
Вариант 26
40
20
−3
−2
−1
0
1
x
−20
−40
−60
Вариант 27
2
1.5
1
0.5
0
−0.5
0.4
0.6
0.8
1.0
1.2
1.4
1.6
1.8
x
Вариант 28


x
−0.6
−0.4
−0.2
0
0.2
0.4
−0.3330
−0.3335
−0.3340
−0.3345
−0.3350
−0.3355
Вариант 29
0.6
0.4
0.2
−0.5
0.5
1
1.5
2
2.5
x
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
Вариант 30

4 Индивидуальная работа «Создание и обработка матриц стандартными средствами языка Python»

4.1 Матрицы и их реализация

Матрица — математический объект, представляющий собой прямоугольную таблицу чисел, символов или выражений. Другими словами, матрица — это совокупность строк и столбцов, на пересечении которых находятся её элементы.

Все строки матрицы имеют одинаковую длину, т. е. содержат одинаковое количество элементов. Количество строк и столбцов задает размер матрицы.

$$\begin{array}{c} \\ 1 \\ 2 \\ \dots \\ m \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & \dots & n \end{array} \\ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} \end{array}$$

Приведенная выше таблица представляет собой матрицу $m \times n$: она содержит m рядов (или строк) и n колонок (или столбцов).

Каждый элемент матрицы обозначается переменной с двумя индексами. Например, a_{21} представляет элемент во второй строке и первом столбце.

Существует множество приложений матриц как в математике, так и в других науках. Со многими из них наши читатели, скорее всего, уже знакомы (матрица коэффициентов линейной системы уравнений, матрица линейного оператора, единичная матрица, матрица смежности графа и др.), с другими — познакомятся в будущем.

Обсудим теперь, как прямоугольная таблица чисел может быть реа-

лизована программными средствами. При этом важно обеспечить доступ к отдельному элементу матрицы, а также удобство основных операций: сложение матриц, умножение матрицы на число.

Самый распространенный способ реализации матриц средствами языка Python состоит в том, что матрица представляется *списком списков*, а именно, списком ее строк. При этом каждая строка, в свою очередь, является списком (например, вещественных чисел).

Матрица $m \times n$ — это список из m строк, каждая из которых является списком из n элементов.

В отличие от математики, индексы в Python отсчитываются от нуля, поэтому `a[2][1]` — это элемент, расположенный на пересечении третьей строки и второго столбца матрицы.

В некоторых случаях матрицу представляют списком столбцов; в данном пособии такой способ рассматриваться не будет.

Посмотрим на пример.

In [1]:
```
a = [[1.0, 2.0, 3.0],
     [4.0, 5.0, 6.0],
     [7.0, 8.0, 9.0]]
print(a[2][1])
print(a)
```

```
8.0
[[1.0, 2.0, 3.0], [4.0, 5.0, 6.0], [7.0, 8.0, 9.0]]
```

Обратите внимание, что функция `print` ничего не знает о наших планах считать список списков чисел матрицей и выводит этот список «как получится». Разработаем функцию `mprint` для более «красивого» вывода матрицы на экран. Ее параметрами будет сама матрица, а также количество позиций, отводимых на отображение каждого элемента

матрицы, и точность — количество десятичных цифр, с которой будет отображаться этот элемент.

Схема работы функции такова: каждый ряд матрицы, являющийся списком чисел, превратим в список строк с помощью функции map. Для преобразования числа в строку воспользуемся инструментом f-строк языка Python, оформив это преобразование lambda-функцией. Получившийся список строк «склеим» в одну строку, которую затем выведем на экран стандартной функцией print:

In [2]:
```
def mprint(a, pos=6, digits=3):
    for row in a:
        print(''.join(map(
            lambda x : f'{x:{pos}.{digits}f}', row)))
```

In [3]:
```
mprint(a, 7, 2)
```

```
   1.00   2.00   3.00
   4.00   5.00   6.00
   7.00   8.00   9.00
```

Приведем еще один пример работы с матрицами. Во многих задачах предполагается, что матрицы вводятся пользователем. Чтобы избежать многочисленных процедур ручного ввода данных, напишем функцию make_matr для автоматического генерирования случайной матрицы размера $m \times n$ целых или вещественных чисел. Входными параметрами функции будут размеры матрицы (количество строк m, количество столбцов n), тип элемента (целый или нет) и ширина диапазона, r. Последний параметр отвечает за «разброс» случайных значений матрицы: они будут генерироваться в пределах от -r до r.

Для генерирования случайных целых чисел будем использовать функцию randint, у которой границы диапазона являются входными параметрами. Для генерирования случайных вещественных чисел используем

функцию `random`. Поскольку ее результат — случайное число в диапазоне от 0 до 1, то для расширения этого диапазона до области `(-r, r)` потребуется простое арифметическое преобразование. Для использования разработанной функции нужно подключить библиотеку `random`.

In [4]:
```
import random
def make_matr(m, n, integer=True, r=10):
    if integer:
        a = [[random.randint(-r, r)
               for j in range(n)]
              for i in range(m)]
    else:
        a = [[2*r*(random.random()-0.5)
               for j in range(n)]
              for i in range(m)]
    return a
```

In [5]:
```
mprint(make_matr(6, 6, r=20), digits=0)
```

```
 -17  -13   10  -16   10   -9
  19    2    4   -2   -5   20
  -4    2   -1  -19   13  -12
 -11   13  -18    7   -8  -10
  10   -7  -10   14    3    9
  20    2  -20  -15    0    6
```

In [6]:
```
mprint(make_matr(4, 4, integer=False), 8, 4)
```

```
-0.7557 -4.3991  7.0470  6.5549
 6.6361  1.8277  5.9337  5.6018
 0.3152 -6.8579  8.8247  9.1836
-1.3709 -7.2816  4.7642 -2.9552
```

Обе разработанные функции рекомендуется сохранить в отдельный модуль, например, `matrix.py`, чтобы их можно было использовать во всех учебных программах.

4.2 Постановка задачи

Задание 1. По заданной матрице вещественных чисел размера 5×5 своего варианта (см. Табл. 4.4) подобрать соответствующее аналитическое выражение для коэффициентов матрицы из Таблицы 4.1.

Задание 2. Выбрать в Таблице 4.2 и реализовать на языке Python функции матричного аргумента, необходимые для выполнения основного задания (задания 3) своего варианта. Отладку разработанных функций осуществить с использованием исходной матрицы размера 5×5.

Задание 3. В соответствии с номером варианта и подобранным ранее аналитическим выражением для коэффициентов матрицы $\boldsymbol{A}$ решить задачу из Таблицы 4.3.

Таблица 4.1 — Аналитические выражения элементов матрицы

0	$A_{ij} = \sin^4(i-j)$	16	$A_{ij} = j^2 \sin^6 i$
1	$A_{ij} = \sin^2 i + j$	17	$A_{ij} = \dfrac{\sin(i-j)}{e^{i+j}}$
2	$A_{ij} = \sin^3(i+j)$	18	$A_{ij} = ij\dfrac{e^{-\frac{i+j}{20}} - 1}{e^{i+j}}$
3	$A_{ij} = \cos^3 i - \cos^3 j$	19	$A_{ij} = \dfrac{e^{-i} - \sin j}{e^{-i} + \cos^2 j}$
4	$A_{ij} = i^2 + \cos^4 j$	20	$A_{ij} = \left(\sin e^{-i} - \sin e^{-j}\right)^2$
5	$A_{ij} = (i+j)\sin^6(i-j)$	21	$A_{ij} = j^2 \sin^4 i$
6	$A_{ij} = (i-j)\cos^5(i+j)$	22	$A_{ij} = \sin\left(e^{-i} + j\right)$
7	$A_{ij} = \sin^3(i^2 - j^2)$	23	$A_{ij} = \dfrac{1 - \cos^2(i-j)}{i^2 + 2j^2}$

8	$A_{ij} = ij\sin^2(i^3 - j^3)$	24	$A_{ij} = \dfrac{1 - e^{\frac{i+j}{50}}}{i^2 + j^2}$
9	$A_{ij} = \cos^4(i + j)$	25	$A_{ij} = \dfrac{\sin^2(i - j)}{2i + \cos j}$
10	$A_{ij} = j^2 + \cos^3 i$	26	$A_{ij} = \dfrac{\sin i - \sin j}{2 + \cos(ij)}$
11	$A_{ij} = \dfrac{\sin i}{j}$	27	$A_{ij} = \sin(\cos j - \cos i)$
12	$A_{ij} = \dfrac{\cos j}{i}$	28	$A_{ij} = \dfrac{\cos^2(\sin i - \sin j)}{\cos i \cos j}$
13	$A_{ij} = \dfrac{\sin^2 j}{i}$	29	$A_{ij} = \sin\left(ije^{\frac{-i-j}{100}} - i - j\right)$
14	$A_{ij} = \dfrac{\cos^2 i}{e^{i/10} + j}$	30	$A_{ij} = \dfrac{\cos(i - j) - 1}{j^2}$
15	$A_{ij} = \dfrac{\cos(i - j)}{\sin i + \cos j}$	31	$A_{ij} = \dfrac{\cos(i - j) - 1}{i^2 + 1}$

Таблица 4.2 — Функции для реализации средствами языка Python[5]

№	Название	Описание
1	`tr(A)`	Вычисляет *след* (сумму элементов, лежащих на главной диагонали) квадратной матрицы.
2	`atr(A)`	Вычисляет сумму элементов, лежащих на побочной диагонали квадратной матрицы.
3	`sum_row(A, k)`	Вычисляет сумму элементов, находящихся в `k`-й строке матрицы.
4	`sum_col(A, k)`	Вычисляет сумму элементов, находящихся в `k`-м столбце матрицы.

[5]Внимание: реализовать нужно только функции, используемые в задании Вашего варианта.

5	`mean_row(A, k)`	Вычисляет среднее арифметическое элементов, находящихся в k-й строке матрицы
6	`mean_col(A, k)`	Вычисляет среднее арифметическое элементов, находящихся в k-м столбце матрицы.
7	`max_in_krow(A, k)`	Вычисляет максимальный среди элементов, находящихся в k-й строке матрицы.
8	`max_in_col(A, k)`	Вычисляет максимальный среди элементов, находящихся в k-м столбце матрицы.
9	`min_in_row(A, k)`	Вычисляет минимальный среди элементов, находящихся в k-й строке матрицы.
10	`min_in_col(A, k)`	Вычисляет минимальный среди элементов, находящихся в k-м столбце матрицы.
11	`max_in_sec_diag(A)`	Вычисляет максимальный среди элементов побочной диагонали (secondary diagonal) квадратной матрицы.
12	`min_in_sec_diag(A)`	Вычисляет минимальный среди элементов побочной диагонали (secondary diagonal) квадратной матрицы.
13	`max_el(A)`	Вычисляет максимальный элемент матрицы.
14	`min_el(A)`	Вычисляет минимальный элемент матрицы
15	`row_with_max(A)`	Определяет номер строки, в которой расположен максимальный элемент матрицы. Если максимальных элементов несколько — выводит наименьший номер.

16	col_with_max(A)	Определяет номер столбца, в котором расположен максимальный элемент матрицы. Если максимальных элементов несколько — выводит наименьший номер.
17	ordered_rows(A)	Вычисляет количество строк в матрице, элементы которых упорядочены (либо по возрастанию, либо по убыванию).
18	ordered_cols(A)	Вычисляет количество столбцов в матрице, элементы которых упорядочены (либо по возрастанию, либо по убыванию).
19	minmax_row(A)	Вычисляет минимальный из максимальных элементов всех строк матрицы.
20	maxmin_row(A)	Вычисляет максимальный из минимальных элементов всех строк матрицы.
21	minmax_col(A)	Вычисляет минимальный из максимальных элементов всех столбцов матрицы.
22	maxmin_col(A)	Вычисляет максимальный из минимальных элементов всех столбцов матрицы.

Необходимо вычислить значение выражения из столбца «Условие» согласно своему варианту задания.

Таблица 4.3 — Варианты основного задания[6]

№	Условие
0	$\sum_{n=10}^{600} \texttt{atr}(\boldsymbol{A}_n) + \prod_{n=125}^{250} \texttt{col_with_max}(\boldsymbol{A}_n)$

[6]В формулах использовано обозначение: A_n — квадратная матрица размера $n \times n$. Расшифровка имен функций приведена в Таблице 4.2.

1	$\sum_{n=1}^{610}\texttt{tr}(\boldsymbol{A}_n) + \sum_{n=100}^{650}\texttt{min_in_sec_diag}\ (\boldsymbol{A}_n)$
2	$\sum_{k=100}^{600}\texttt{sum_row}(\boldsymbol{A}_{750}, k) + \prod_{n=50}^{250}\texttt{max_el}(\boldsymbol{A}_n)$
3	$\sum_{k=120}^{620}\texttt{sum_col}(\boldsymbol{A}_{850}, k) + \sum_{n=1}^{450}\texttt{min_el}(\boldsymbol{A}_n)$
4	$\sum_{k=110}^{535}\texttt{mean_row}(\boldsymbol{A}_{755}, k) - \prod_{n=195}^{220}\texttt{row_with_max}(\boldsymbol{A}_n)$
5	$\sum_{k=400}^{650}\texttt{mean_col}(\boldsymbol{A}_{750}, k) + \sum_{n=11}^{600}\texttt{ordered_rows}(\boldsymbol{A}_n)$
6	$\sum_{k=100}^{840}\texttt{max_in_row}(\boldsymbol{A}_{845}, k) - \sum_{n=1}^{600}\texttt{ordered_cols}(\boldsymbol{A}_n)$
7	$\sum_{k=10}^{555}\texttt{max_in_col}(\boldsymbol{A}_{700}, k) - \prod_{n=150}^{300}\texttt{minmax_row}(\boldsymbol{A}_n)$
8	$\sum_{k=20}^{740}\texttt{min_in_row}(\boldsymbol{A}_{760}, k) - \sum_{n=10}^{640}\texttt{maxmin_row}(\boldsymbol{A}_n)$
9	$\sum_{k=15}^{525}\texttt{min_in_col}(\boldsymbol{A}_{650}, k) + \sum_{n=50}^{620}\texttt{minmax_col}(\boldsymbol{A}_n)$
10	$\prod_{k=50}^{250}\texttt{max_in_sec_diag}(\boldsymbol{A}_{260-k}) + \sum_{n=10}^{630}\texttt{maxmin_col}(\boldsymbol{A}_n)$
11	$\sum_{n=1}^{380}\texttt{tr}(\boldsymbol{A}_n) - \prod_{n=600}^{680}\texttt{max_el}(\boldsymbol{A}_n)$

12	$\sum_{k=50}^{300} \texttt{sum_row}(\boldsymbol{A}_{500}, k) + \sum_{n=10}^{500} \texttt{min_el}(\boldsymbol{A}_n)$
13	$\sum_{k=10}^{600} \texttt{sum_col}(\boldsymbol{A}_{650}, k) - \prod_{n=550}^{600} \texttt{row_with_max}(\boldsymbol{A}_n)$
14	$\sum_{k=100}^{650} \texttt{mean_row}(\boldsymbol{A}_{660}, k) - \sum_{n=1}^{520} \texttt{ordered_rows}(\boldsymbol{A}_n)$
15	$\sum_{k=100}^{500} \texttt{mean_col}(\boldsymbol{A}_{660}, k) + \sum_{n=1}^{620} \texttt{ordered_cols}(\boldsymbol{A}_n)$
16	$\sum_{k=1}^{500} \texttt{max_in_row}(\boldsymbol{A}_{510}, k) - \prod_{n=500}^{530} \texttt{minmax_row}(\boldsymbol{A}_n)$
17	$\sum_{k=10}^{555} \texttt{max_in_col}(\boldsymbol{A}_{565}, k) - \sum_{n=190}^{590} \texttt{maxmin_row}(\boldsymbol{A}_n)$
18	$\sum_{k=100}^{700} \texttt{min_in_row}(\boldsymbol{A}_{800}, k) + \sum_{n=10}^{600} \texttt{minmax_col}(\boldsymbol{A}_n)$
19	$\sum_{k=50}^{660} \texttt{min_in_col}(\boldsymbol{A}_{760}, k) + \sum_{n=10}^{600} \texttt{maxmin_col}(\boldsymbol{A}_n)$
20	$\prod_{k=250}^{280} \texttt{max_in_sec_diag}(\boldsymbol{A}_{k+50}) + \prod_{n=350}^{380} \texttt{col_with_max}(\boldsymbol{A}_n)$
21	$\sum_{n=1}^{630} \texttt{tr}(\boldsymbol{A}_n) - \sum_{n=100}^{420} \texttt{min_el}(\boldsymbol{A}_n)$
22	$\sum_{k=150}^{670} \texttt{sum_row}(\boldsymbol{A}_{700}, k) - \prod_{n=360}^{380} \texttt{row_with_max}(\boldsymbol{A}_n)$

23	$\sum_{k=100}^{340} \texttt{sum_col}(\boldsymbol{A}_{570}, k) - \sum_{n=150}^{680} \texttt{ordered_rows}(\boldsymbol{A}_n)$
24	$\sum_{k=100}^{575} \texttt{mean_row}(\boldsymbol{A}_{675}, k) + \sum_{n=220}^{630} \texttt{ordered_cols}(\boldsymbol{A}_n)$
25	$\sum_{k=100}^{500} \texttt{mean_col}(\boldsymbol{A}_{550}, k) - \prod_{n=500}^{520} \texttt{minmax_row}(\boldsymbol{A}_n)$
26	$\sum_{k=1}^{750} \texttt{max_in_row}(\boldsymbol{A}_{800}, k) - \sum_{n=40}^{590} \texttt{maxmin_row}(\boldsymbol{A}_n)$
27	$\sum_{k=10}^{420} \texttt{max_in_col}(\boldsymbol{A}_{750}, k) - \sum_{n=150}^{550} \texttt{minmax_col}(\boldsymbol{A}_n)$
28	$\sum_{k=100}^{520} \texttt{min_in_row}(\boldsymbol{A}_{520}, k) - \sum_{n=10}^{550} \texttt{maxmin_col}(\boldsymbol{A}_n)$
29	$\sum_{k=50}^{600} \texttt{min_in_col}(\boldsymbol{A}_{605}, k) + \prod_{n=415}^{435} \texttt{col_with_max}(\boldsymbol{A}_n)$
30	$\prod_{k=900}^{915} \texttt{max_in_sec_diag}(\boldsymbol{A}_k) - \prod_{k=501}^{513} \texttt{max_in_krow}(\boldsymbol{A}_{550}, k)$
31	$\sum_{n=10}^{600} \texttt{tr}(\boldsymbol{A}_n) + \sum_{n=20}^{550} \texttt{ordered_rows}(\boldsymbol{A}_n)$

4.3 Образец выполнения задания

Задание 1. Из таблицы 4.4 находим, что матрица $\boldsymbol{A}_5$ размера 5×5 имеет вид

0.00000	0.50137	0.68363	0.00040	0.32804
0.50137	0.00000	0.50137	0.68363	0.00040
0.68363	0.50137	0.00000	0.50137	0.68363
0.00040	0.68363	0.50137	0.00000	0.50137
0.32804	0.00040	0.68363	0.50137	0.00000

Чтобы определить, какая строка в таблице 4.1 описывает выражение ее коэффициентов, сначала обратим внимание на некоторые особенности данной матрицы и сравним их с особенностями матриц, представленных в таблице 4.1.

Обращаем внимание на следующие характеристики:

- Являются ли все элементы данной матрицы неотрицательными числами? В таблице 4.1, судя по формулам, таких около половины, например формулы № 0, 1, 4, ... Ответ на этот вопрос — положителен, поэтому удается сократить перебор вариантов в два раза.
- Является ли матрица *симметричной* (значение коэффициентов не меняются, если поменять в формуле для них индексы i и j местами)? Такими являются, например, формулы 0, 2, 5 и др. Ответ на этот вопрос также положителен.
- Является ли матрица *антисимметричной* (значения коэффициентов меняют знак, если поменять в формуле для них индексы i и j местами)? Такими являются, например, формулы 3, 6, 7 и др. Ответ на этот вопрос отрицателен.

  Анализ симметричности позволяет сократить число вариантов еще примерно в два раза.
- Что стоит на главной диагонали у матрицы? У многих матриц в таблице 4.1 на главной диагонали стоят нули, потому что соответ-

ствующие выражения обращаются в нуль при $i = j$ (например, № 0, 3, 5, 6 и др.). В данном случае — на диагонали стоят нули, и количество вариантов сокращается еще примерно наполовину.

На основе проведенного анализа отбираем в таблице 4.1 симметричные матрицы с неотрицательными коэффициентами и нулевыми элементами на главной диагонали. Таковых только четыре: 0, 8, 15, 20. С помощью соответствующих генераторов создадим эти матрицы и проверим, какая из них совпадает с заданной.

Ниже приведен текст программы, которая генерирует и выводит на экран элементы отобранных матриц.

```
1  from math import *
2
3  def mprint(a, pos=6, digits=3):
4      for row in a:
5          print(''.join(map(
6              lambda x : f'{x:{pos}.{digits}f}', row)))
7
8  M = 5
9  A0 = [[sin(i-j)**4 for j in range(1,M+1)]
10       for i in range(1,M+1)]
11 A8 = [[sin(i**3-j**3)**2 for j in range(1,M+1)]
12       for i in range(1,M+1)]
13 # код вычисления двух других матриц опущен
14 print('№ 0')
15 mprint(A0, digits=5, pos=8)
16 print('№ 8')
17 mprint(A8, digits=5, pos=8)
18 # код печати двух других матриц опущен
```

В программе учтено, что в таблице 4.1 задано математическое описание элементов матрицы, т. е. считается, что индексы меняются от одного до пяти, а не от нуля до четырех, как принято в Python. Поэтому вместо `range(M)` написано `range(1, M+1)`. Для вывода на печать использована функция `mprint`. Количество выводимых на печать символов выбрано в примерном соответствии с числом цифр в заданной матрице.

```
№ 0
 0.00000 0.50137 0.68363 0.00040 0.32804
 0.50137 0.00000 0.50137 0.68363 0.00040
 0.68363 0.50137 0.00000 0.50137 0.68363
 0.00040 0.68363 0.50137 0.00000 0.50137
 0.32804 0.00040 0.68363 0.50137 0.00000
```

Из приведенного фрагмента вывода видно, что искомой является матрица № 0. Значит, в дальнейшем для задания элементов матрицы будем использовать формулу:

$$A_{ij} = \sin^4(i - j),$$

а для создания матрицы произвольного размера $M \times M$ генератор вида `A0` из программы выше.

Задание 2. В соответствии с формулой, приведенной в варианте 0 таблицы 4.3, для выполнения основного задания требуется разработать две вспомогательные функции:

- `atr(A)` — для вычисления суммы элементов, лежащих на побочной диагонали квадратной матрицы.
- `col_with_max(A)` — для определения номера столбца, в котором расположен максимальный элемент матрицы.

Ниже приведен текст программы, содержащий описания этих функций и выводящий на экран результаты их применения к заданной матрице размера 5×5.

```
1  from math import *
2  M = 5
3  A0 = [[sin(i-j)**4  for j in range(1,M+1)]
4        for i in range(1,M+1)]
5  def atr(A):
6      M = len(A) # количество строк
7      N = len(A[0]) # количество столбцов
8      if M!=N:
9          print('Матрица - не квадратная')
```

```
10            return None
11        else:
12            return sum([A[i][N-1-i] for i in range(N)])
13    def colWithMax(A):
14        M = len(A) # количество строк
15        N = len(A[0]) # количество столбцов
16        # создадим список максимальных элементов
17        #                        каждого столбца
18        maxColList = [max([A[i][j] for i in range(M)])
19                      for j in range(N)]
20        # найдем в нем максимальный элемент и его позицию
21        return maxColList.index(max(maxColList))+1
22        # единица добавлена для перехода
23        # к математической системе индексов.
24    print(f'Сумма элементов побочной диагонали: {atr(A0)}')
25    print(f'Столбец с макс. элементом: {colWithMax(A0)}')
```

Результаты работы программы представлены ниже

```
Сумма элементов побочной диагонали: 2.023353776139226
Столбец с макс. элементом: 1
```

С использованием калькулятора убеждаемся, что

1. $0.32804 + 0.68363 + 0 + 0.68363 + 0.32804 = 2.02334$, т. е. функция `atr` работает корректно.
2. В столбце 1 действительно находится максимальный элемент матрицы, равный 0.68363. Этот элемент не единственный; в соответствии с заданием программа выдала один из возможных вариантов. Функция `col_with_max` проверена.

Задание 3. Для выполнения последнего задания дополним программу функцией генерирования матрицы произвольного размера $n \times n$, которую так и назовем `A(n)`. Кроме того, подключим модуль `time` для учета времени работы.

Окончательный текст программы примет вид:

```
from math import *
from time import time
def atr(A):
    M = len(A) # количество строк
    N = len(A[0]) # количество столбцов
    if M!=N:
        print('Матрица - не квадратная')
        return None
    else:
        return sum([A[i][N-1-i] for i in range(N)])
def colWithMax(A):
    M = len(A) # количество строк
    N = len(A[0]) # количество столбцов
    # создадим список максимальных элементов
    #                         каждого столбца
    maxColList = [max([A[i][j] for i in range(M)])
                  for j in range(N)]
    # найдем в нем максимальный элемент и его позицию
    return maxColList.index(max(maxColList))+1
    # единица добавлена для перехода
    # к математической системе индексов.

def A(n):
    return [[sin(i-j)**4 for j in range(1,n+1)]
            for i in range(1,n+1)]
now = time()
s = 0
for n in range(10, 600):
    s += atr(A(n))
p = 1
for n in range(125, 250):
    p *= colWithMax(A(n))
delta_t = time() - now
print(f'Результат {s+p} получен за {delta_t:.6f} сек')
```

```
Результат 67371.31505924519 получен за 17.944041 сек
```

Из приведенного выше результата работы видно, что программа выполнялась достаточно долго, поэтому возможно стоит уделить больше внимания оптимизации ее работы.

4.4 Материалы к заданию

В данном разделе представлены матрицы $\boldsymbol{A}$ вещественных чисел размера 5×5. Только одна из представленных матриц соответствует Вашему варианту аналитического выражения, взятого из таблицы 4.1.

Вариант 0

0.00000	0.50137	0.68363	0.00040	0.32804
0.50137	0.00000	0.50137	0.68363	0.00040
0.68363	0.50137	0.00000	0.50137	0.68363
0.00040	0.68363	0.50137	0.00000	0.50137
0.32804	0.00040	0.68363	0.50137	0.00000

Вариант 1

0.00000	-0.04189	-0.01665	-0.00095	0.00188
0.04189	0.00000	-0.00567	-0.00225	-0.00013
0.01665	0.00567	0.00000	-0.00077	-0.00031
0.00095	0.00225	0.00077	0.00000	-0.00010
-0.00188	0.00013	0.00031	0.00010	0.00000

Вариант 2

0.50137	2.00547	4.51231	8.02189	12.53420
0.68363	2.73454	6.15271	10.93815	17.09086
0.00040	0.00159	0.00357	0.00635	0.00992
0.32804	1.31217	2.95238	5.24868	8.20106
0.84555	3.38218	7.60991	13.52874	21.13865

Вариант 3

0.00000	-0.22985	-0.70807	-0.99500	-0.82682
-0.09194	0.00000	-0.09194	-0.28323	-0.39800
-0.14161	-0.04597	0.00000	-0.04597	-0.14161
-0.11706	-0.08330	-0.02704	0.00000	-0.02704
-0.06360	-0.07654	-0.05447	-0.01768	0.00000

Вариант 4

0.00000	0.44706	0.81863	0.01479	0.25080
0.15595	0.00000	0.23524	0.24708	0.00465
0.12642	0.12681	0.00000	0.13244	0.13158
0.00233	0.10902	0.10101	0.00000	0.08548
0.05434	0.00208	0.09177	0.07576	0.00000

Вариант 5

-0.54525	-0.99998	-0.53533	0.42150	0.99081
0.86023	0.03571	-0.82164	-0.92358	-0.17639
0.78680	-0.09425	-0.88865	-0.86603	-0.04719
-0.81372	0.04946	0.86717	0.88760	0.09198
-0.98247	-0.37397	0.57836	0.99895	0.50111

Вариант 6

0.02999	0.96057	0.18254	0.00647	0.84995
0.96057	0.18254	0.00647	0.84995	0.32304
0.18254	0.00647	0.84995	0.32304	0.00045
0.00647	0.84995	0.32304	0.00045	0.68916
0.84995	0.32304	0.00045	0.68916	0.49567

Вариант 7

0.00000	-0.11492	-0.15735	-0.12437	-0.06615
-0.45970	0.00000	-0.05108	-0.08851	-0.07960
-1.41615	-0.11492	0.00000	-0.02873	-0.05665
-1.98999	-0.35404	-0.05108	0.00000	-0.01839
-1.65364	-0.49750	-0.15735	-0.02873	0.00000

Вариант 8

0.00000	0.86326	1.74449	0.11203	4.95696
0.86326	0.00000	0.13478	2.17612	4.75683
1.74449	0.13478	0.00000	4.96970	4.93150
0.11203	2.17612	4.96970	0.00000	18.66767
4.95696	4.75683	4.93150	18.66767	0.00000

Вариант 9

0.54030	-0.41615	-0.98999	-0.65364	0.28366
0.27015	-0.20807	-0.49500	-0.32682	0.14183
0.18010	-0.13872	-0.33000	-0.21788	0.09455
0.13508	-0.10404	-0.24750	-0.16341	0.07092
0.10806	-0.08323	-0.19800	-0.13073	0.05673

Вариант 10

0.75183	0.00281	-0.43346	-0.88177	-0.02181
0.00281	-0.43346	-0.88177	-0.02181	0.28358
-0.43346	-0.88177	-0.02181	0.28358	0.96841
-0.88177	-0.02181	0.28358	0.96841	0.06999
-0.02181	0.28358	0.96841	0.06999	-0.16101

Вариант 11

-0.02041	-0.01237	-0.00833	-0.00619	-0.00490
-0.01237	-0.01041	-0.00809	-0.00637	-0.00518
-0.00833	-0.00809	-0.00708	-0.00601	-0.00510
-0.00619	-0.00637	-0.00601	-0.00542	-0.00481
-0.00490	-0.00518	-0.00510	-0.00481	-0.00443

Вариант 12

-0.71777	-1.00067	0.16822	1.41446	2.95935
-1.65270	-2.50868	-0.00519	1.58578	5.07072
-2.31681	-3.85491	-0.08868	1.69083	7.74435
-2.65327	-4.65280	-0.12300	1.73963	9.89311
-2.79489	-5.01656	-0.13618	1.75936	11.07383

Вариант 13

0.70807	0.82682	0.01991	0.57275	0.91954
0.35404	0.41341	0.00996	0.28638	0.45977
0.23602	0.27561	0.00664	0.19092	0.30651
0.17702	0.20671	0.00498	0.14319	0.22988
0.14161	0.16536	0.00398	0.11455	0.18391

Вариант 14

-0.01288	-0.01387	-0.00996	-0.00596	-0.00321
-0.01387	-0.01328	-0.00894	-0.00514	-0.00269
-0.00996	-0.00894	-0.00578	-0.00323	-0.00166
-0.00596	-0.00514	-0.00323	-0.00177	-0.00089
-0.00321	-0.00269	-0.00166	-0.00089	-0.00045

Вариант 15

0.00000	0.22980	1.12801	0.43700	0.13490
-0.22980	0.00000	0.89821	0.20720	-0.09489
-1.12801	-0.89821	0.00000	-0.69101	-0.99310
-0.43700	-0.20720	0.69101	0.00000	-0.30209
-0.13490	0.09489	0.99310	0.30209	0.00000

Вариант 16

0.00000	0.05050	0.09602	0.11650	0.12454
0.05050	0.00000	0.00725	0.01360	0.01643
0.09602	0.00725	0.00000	0.00099	0.00185
0.11650	0.01360	0.00099	0.00000	0.00013
0.12454	0.01643	0.00185	0.00013	0.00000

Вариант 17

0.00000	1.06502	2.26098	0.00004	1.12732
1.06502	0.00000	1.77503	3.39146	0.00006
2.26098	1.77503	0.00000	2.48504	4.52195
0.00004	3.39146	2.48504	0.00000	3.19505
1.12732	0.00006	4.52195	3.19505	0.00000

Вариант 18

-0.85200	-0.87192	-0.89907	-0.93025	-0.96116
-0.87192	-0.15620	0.64984	0.99933	0.72954
-0.89907	0.64984	0.61762	-0.86599	-0.42272
-0.93025	0.99933	-0.86599	0.46769	0.14563
-0.96116	0.72954	-0.42272	0.14563	0.05454

Вариант 19

0.00000	-0.04282	0.69341	1.18711	0.78838
0.04282	0.00000	0.25950	0.89841	1.60925
-0.69341	-0.25950	0.00000	0.31574	0.88691
-1.18711	-0.89841	-0.31574	0.00000	0.08393
-0.78838	-1.60925	-0.88691	-0.08393	0.00000

Вариант 20

0.00000	-0.81715	-0.99918	-0.92983	-0.25383
0.81715	0.00000	-0.54287	-0.23527	0.64407
0.99918	0.54287	0.00000	0.33004	0.95618
0.92983	0.23527	-0.33004	0.00000	0.80597
0.25383	-0.64407	-0.95618	-0.80597	0.00000

Вариант 21

-0.00000	0.95095	0.23864	-0.00551	-3.26438
-0.95095	-0.00000	-0.00184	-1.63219	-0.73063
-0.23864	0.00184	0.00000	-0.24354	0.00013
0.00551	1.63219	0.24354	-0.00000	0.62792
3.26438	0.73063	-0.00013	-0.62792	-0.00000

Вариант 22

0.00000	0.07867	0.04352	0.00060	0.01123
0.11801	0.00000	0.03219	0.02297	0.00037
0.07517	0.04165	0.00000	0.01727	0.01401
0.00111	0.03445	0.02083	0.00000	0.01073
0.02121	0.00060	0.01923	0.01242	0.00000

Вариант 23

1.08522	1.02999	1.96057	1.18254	1.00647
4.08522	4.02999	4.96057	4.18254	4.00647
9.08522	9.02999	9.96057	9.18254	9.00647
16.08522	16.02999	16.96057	16.18254	16.00647
25.08522	25.02999	25.96057	25.18254	25.00647

Вариант 24

0.84147	0.42074	0.28049	0.21037	0.16829
0.90930	0.45465	0.30310	0.22732	0.18186
0.14112	0.07056	0.04704	0.03528	0.02822
-0.75680	-0.37840	-0.25227	-0.18920	-0.15136
-0.95892	-0.47946	-0.31964	-0.23973	-0.19178

Вариант 25

0.13867	0.09401	0.07111	0.05718	0.04782
0.07796	0.05376	0.04102	0.03317	0.02784
0.41708	0.29258	0.22531	0.18320	0.15435
0.17146	0.12236	0.09512	0.07780	0.06581
0.03038	0.02205	0.01731	0.01424	0.01210

Вариант 26

1.15773	4.15773	9.15773	16.15773	25.15773
0.92793	3.92793	8.92793	15.92793	24.92793
0.02972	3.02972	8.02972	15.02972	24.02972
0.72073	3.72073	8.72073	15.72073	24.72073
1.02282	4.02282	9.02282	16.02282	25.02282

Вариант 27

0.72371	1.27033	2.80193	-5.27076	-0.58095
0.37273	2.02778	-6.69560	-1.62777	-0.82986
-0.61070	-1.96454	-1.17803	-1.05420	-0.97967
4.57271	0.35479	-0.30931	-0.70900	-1.14195
1.56142	0.71996	0.21353	-0.33506	-1.48091

Вариант 28

1.70807	2.70807	3.70807	4.70807	5.70807
1.82682	2.82682	3.82682	4.82682	5.82682
1.01991	2.01991	3.01991	4.01991	5.01991
1.57275	2.57275	3.57275	4.57275	5.57275
1.91954	2.91954	3.91954	4.91954	5.91954

Вариант 29

0.00000	-0.00281	-0.96841	-0.27499	0.74264
0.00281	0.00000	0.88177	0.15448	-0.58565
0.96841	-0.88177	0.00000	-0.28358	0.02386
0.27499	-0.15448	0.28358	0.00000	-0.06999
-0.74264	0.58565	-0.02386	0.06999	0.00000

Вариант 30

0.35501	1.42002	3.19505	5.68009	8.87513
0.56524	2.26098	5.08719	9.04390	14.13109
0.00001	0.00003	0.00007	0.00013	0.00020
0.18789	0.75155	1.69098	3.00618	4.69716
0.77751	3.11004	6.99759	12.44016	19.43775

Вариант 31

3.42552	-4.42708	-1.09299	-0.00214	0.33795
-4.42708	5.77440	1.25544	0.03329	-0.72755
-1.09299	1.25544	1.02032	0.60025	-0.73254
-0.00214	0.03329	0.60025	2.34055	-5.17599
0.33795	-0.72755	-0.73254	-5.17599	12.42788

5 Проектное задание «Вычисления с использованием пакета NumPy»

5.1 Пакет NumPy

5.1.1 Основные сведения о пакете

NumPy (Numerical Python) — это библиотека Python с открытым исходным кодом, которая используется практически во всех областях науки и техники. В настоящее время она считается универсальным стандартом для организации и проведения вычислений в Python. Число пользователей NumPy чрезвычайно обширно, оно включает и начинающих кодеров, и опытных исследователей, занимающихся современными научными и промышленными исследованиями и разработками. NumPy широко используется в Pandas, SciPy, Matplotlib, scikit-learn, scikit-image и большинстве других пакетов Python для обработки и анализа данных.

В качестве основной информации о NumPy приведем несколько заголовков и цитат с официального сайта [27] этой библиотеки.

Мощные N-мерные массивы: Быстрые и универсальные концепции векторизации, индексации и приведения размерностей пакета NumPy являются сегодня де-факто стандартами организации массивов и операций с ними.

Вычислительные инструменты: NumPy предлагает исчерпывающий набор математических функций, генераторы случайных чисел, процедуры линейной алгебры, преобразования Фурье и многое другое.

Совместимость: NumPy поддерживает широкий спектр аппаратных и вычислительных платформ и хорошо работает с библиотеками для разреженных массивов, библиотеками распределенных вычислений и вычислений на графических процессорах.

Производительность: Ядро NumPy — это хорошо оптимизирован-

ный код C. Наслаждайтесь гибкостью языка Python в сочетании со скоростью скомпилированного кода.

Простота использования: Синтаксис высокого уровня NumPy делает его доступным и продуктивным для программистов с любым опытом и уровнем подготовки.

Открытый исходный код: Распространяемый под лицензией BSD, NumPy разрабатывается и поддерживается публично на GitHub динамичным, доброжелательным и широким по охвату сообществом.

Программирование массивов обеспечивает мощный, компактный и выразительный синтаксис для доступа и обработки данных, хранящихся в таких структурах, как векторы, матрицы и многомерные массивы. NumPy — это основная библиотека программирования массивов для языка Python. Она играет важную роль в научных исследованиях в таких разнообразных областях, как физика, химия, астрономия, география и геология, биология, психология, материаловедение, инженерия, финансы и экономика. Например, в астрономии NumPy был важной частью стека программного обеспечения, используемого при открытии гравитационных волн и при создании первого изображения черной дыры. 16 сентября 2020 года в журнале Nature была опубликована первая официальная статья [18], посвященная пакету NumPy (спустя 14 лет после выхода в свет версии 1.0).

Подводя короткие итоги, получаем, что NumPy — это библиотека поддержки нового типа данных, а именно *массивов*. Его установка в систему осуществляется следующей командой, которую можно выполнить в командной строке или Windows PowerShell:

```
c:\> pip install numpy
```

5.1.2 Простая демонстрация возможностей

Возможности пакета NumPy по ускорению программного кода продемонстрируем на примере решения простой задачи:

Дан список из N вещественных чисел. Найти сумму синусов элементов этого списка.

Общепринятым и ставшим уже каноническим способом подключения библиотеки NumPy является следующая строчка:

```
In [1]: import numpy as np
```

Теперь все константы, типы данных и функции пакета будут начинаться с префикса `np.` Во всех дальнейших примерах этого раздела будем считать данную строчку выполненной.

Для решения задачи средствами классического Python подключим библиотеку `math`. Кроме того нам понадобятся библиотеки `random` для генерирования списка чисел и `time` для оценки временных затрат разных способов решения задачи. В качестве N возьмем достаточно большое число, например 10^7.

```
In [2]: import time
        import random
        import math

        n = 10**7
        print('Создаем большой список...')
        lst = [random.random() for i in range(n)]
        arr = np.array(lst)
        print('Сделано.')
```

```
Создаем большой список...
Сделано.
```

Поставленную задачу можно решать разными способами. Начнем с самого плохого (и запомним, что больше так задачи суммирования на Python решать не будем)

In [3]:
```
t1 = time.time()
s = 0
for i in range(n):
    s += math.sin(lst[i])
dt1 = time.time()-t1
print('Плохое решение')
print(f'затраченное время: {dt1}')
print(f'рассчитанная сумма: {s}')
```

```
Плохое решение
затраченное время: 1.539259433746338
рассчитанная сумма: 4596801.055678344
```

Приведенное решение названо «плохим» не потому что оно неправильное; требуемая сумма найдена. Его проблемы в другом: код решения занимает три строчки вместо одной и выполняется в полтора раза медленнее, чем в следующем случае:

In [4]:
```
t2 = time.time()
s = sum(math.sin(x) for x in lst)
dt2 = time.time()-t2
print('Стандартный Python')
print(f'затраченное время: {dt2}')
print(f'рассчитанная сумма: {s}')
```

```
Стандартный Python
затраченное время: 0.9662299156188965
рассчитанная сумма: 4596801.055678344
```

Посмотрим теперь, как с этой же задачей справится пакет NumPy. Обратите внимание, что список `lst` заменен массивом `arr`, и использованы собственные функции пакета для вычисления синуса и суммы:

```
In [5]: t3 = time.time()
        s = np.sum(np.sin(arr))
        dt3 = time.time()-t3
        print('Python + numpy')
        print(f'затраченное время: {dt3}')
        print(f'рассчитанная сумма: {s}')
```

```
Python + numpy
затраченное время: 0.06240344047546387
рассчитанная сумма: 4596801.055678513
```

```
In [6]: print(f'вычисления ускорены в {round(dt2/dt3,␣
         ↪ndigits=1)} раз')
```

```
вычисления ускорены в 15.5 раз
```

Код программы стал еще компактнее, а вычисления ускорены в 15 раз! Именно такая эффективность реализации NumPy и определила его популярность и востребованность.

5.1.3 Массивы и их создание

Основным объектом библиотеки NumPy является *массив* — однородный контейнер фиксированного размера; все хранящиеся в нем объекты имеют одинаковый тип (как правило, числовой) и размер. Для работы с массивами в NumPy служит класс `ndarray`, который используется для представления как матриц, так и векторов. *Вектор* — это одномерный массив, и в этом смысле нет никакой разницы между вектором-строкой

и вектором-столбцом. *Матрица* — это массив с двумя измерениями. Для трехмерных массивов или массивов большей размерности в программировании и науке о данных обычно используется термин *тензор*.

В NumPy измерения называются *осями*.

Например, массив координат точки в трехмерном пространстве имеет одну ось. Эта ось содержит три элемента, поэтому будем говорить, что длина этой оси равна трем. Приведенный ниже массив имеет две оси:

```
[[1., 0., 0.],
 [0., 1., 2.]]
```

Первая ось (количество строк) имеет длину 2, а вторая (количество столбцов) имеет длину 3.

Независимо от интерпретации (вектор, матрица, тензор) физически массив NumPy представляет собой непрерывный блок данных в памяти. Но кроме собственно данных, объект типа `ndarray` содержит и небольшой набор атрибутов, уточняющих, как интерпретировать эти данные, как осуществлять к ним доступ и т. д. Перечислим некоторые из этих атрибутов.

- `ndarray.ndim`

  Количество осей (размерностей) массива.

- `ndarray.shape`

  Форма, или размерность массива: кортеж целых чисел, указывающий размерность массива в каждом измерении. Например, матрица из `n` строк и `m` столбцов будет иметь форму `(n, m)`. Таким образом, `len(shape)`, т. е. количество элементов в этом кортеже, совпадает с количеством осей — с упомянутым выше параметром `ndim`.

- `ndarray.size`

  Общее количество элементов массива. Оно равно произведению всех элементов параметра `shape`.

- `ndarray.dtype`

Тип элементов в массиве. Это могут быть стандартные типы языка Python; кроме них, NumPy предоставляет собственные типы. К ним относятся, например, `numpy.int32` и `numpy.float64`. Обратите внимание, что в отличие от типа `int` целые типы пакета NumPy ограничены в размерах.

- `ndarray.itemsize`
  Размер в байтах каждого элемента массива. Например, у массива элементов типа `float64` каждый элемент имеет размер 8 байт (64 — это количество бит).
- `ndarray.data`
  Участок памяти, содержащий фактические элементы массива. Этот атрибут обычно не используется, потому что доступ к элементам в массиве осуществляется с помощью индексации.

Перечислим некоторые способы **создания массивов** NumPy.

Массив может быть создан из обычного списка или кортежа Python, используя функцию `array`. Именно таким образом был создан массив `arr` из списка `lst` в примере выше. По умолчанию тип результирующего массива определяется по типу элементов последовательности, на основе которой он создается. Массивы, созданные из списков смешанного типа, получают при этом тип, требующий наибольшее количество места для хранения элементов в памяти.

In [7]:
```
a = np.array([1,2,3])
a
```

Out[7]:
```
array([1, 2, 3])
```

In [8]:
```
b = np.array([[1.5, 2, 3],
              [4, 5, 6]])
b
```

```
Out[8]: array([[1.5, 2. , 3. ],
               [4. , 5. , 6. ]])
```

При необходимости тип элемента может быть задан явно, используя параметр dtype функции array:

```
In [9]: a = np.array([1,2,3], dtype=np.float64)
        a
```

```
Out[9]: array([1., 2., 3.])
```

Достаточно часто встречается случай, когда программисту требуется задать массив, заполненный нулями. Еще один распространенный вариант — просто выделить в памяти место под массив, который будет заполнен реальными данными позже. Наконец, существенно более редкий, но тоже заслуживающий упоминания случай, когда нужно создать массив, состоящий из единиц. Для всех этих трех случаев в библиотеке NumPy есть соответствующие функции.

Функция zeros создает массив, заполненный нулями, функция ones создает массив из единиц, а результатом функции empty является массив, начальное содержимое которого является случайным и зависит от состояния памяти. По умолчанию тип элементов создаваемых массивов — float64; при необходимости его можно указать явно с помощью ключевого слова dtype.

Аргументом этих трех функций является кортеж, определяющий форму создаваемого массива:

```
In [10]: a = np.ones(6)
         a
```

```
Out[10]: array([1., 1., 1., 1., 1., 1.])
```

```
In [11]: b = np.zeros((4,4), dtype=np.int32)
         print(b)
```

```
[[0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]
 [0 0 0 0]]
```

```
In [12]: c = np.empty((3,3))
         print(c)
```

```
[[2.12199579e-314 6.35962139e-311 6.36386538e-311]
 [4.24399158e-314 6.36174338e-311 6.36598737e-311]
 [6.36598737e-314 6.36386538e-311 6.36810937e-311]]
```

Если вам кажется, что из-за единицы и нуля несправедливо обидели остальные числа, то можете смело использовать функцию `full`, имеющую два основных параметра — `shape` и `value`. Результатом ее работы является массив формы `shape`, все элементы которого имеют одно и то же значение `value`:

```
In [13]: p = np.full((2, 3), 3.14)
         print(p)
```

```
[[3.14 3.14 3.14]
 [3.14 3.14 3.14]]
```

Сейчас мы переходим к двум, пожалуй, самым распространенным и важным способам создания массивов. Они еще неоднократно встретятся ниже в примерах кода, связанного с построением графиков.

Функция `arange` создает последовательность с заданным постоянным шагом. Три ее параметра — `start`, `end`, `step` — аналогичны по смыслу

параметрам традиционной функции range, вот только шаг не обязан быть целым числом:

In [14]:
```
x = np.arange(10, 50, 5)
print(x)
y = np.arange(0, 15, 2.5)
print(y)
```

```
[10 15 20 25 30 35 40 45]
[ 0.   2.5  5.   7.5 10.  12.5]
```

Как и в случае функции range значение параметра end не достигается.

При использовании нецелого шага возможны проблемы с определением количества элементов. Если важно именно количество, то используется функция linspace(a, b, n), основными аргументами которой являются:

a, b — границы отрезка,

n — количество частей, на которые этот отрезок нужно разбить.

По умолчанию начальная и конечная точки отрезка попадают в результирующий массив:

In [15]:
```
a = np.linspace(0, 2, 9)  # 9 чисел  на [0,2]
print(a)
```

```
[0.   0.25 0.5  0.75 1.   1.25 1.5  1.75 2.  ]
```

5.1.4 Операции с массивами

Как уже отмечалось выше, для определения формы массива, например, количества строк и столбцов, служит атрибут shape класса ndarray.

In [16]:
```
b = np.array([(1.5,2,3),(4,5,6)])
print(b.shape)
```

```
(2, 3)
```

In [17]:
```
a = np.linspace(0, 19, 20)
print(a.shape)
```

```
(20,)
```

Обратите внимание, что `shape` всегда является кортежем, даже если состоит из одного элемента в случае одномерных массивов.

Существует несколько вариантов **поменять форму массива**. Еще раз уточним, что это не приводит к изменению его содержимого, а только меняет способ индексации и доступа к этим данным. Заметим, что для доступа к элементу двумерного массива (матрицы) `z`, находящемуся в строке с номером `i` и столбце с номером `j`, можно, как и раньше, использовать нотацию `z[i][j]`, однако NumPy поддерживает более естественную систему записи: `z[i,j]`.

In [18]:
```
# Вариант 1
a.shape = (5,4)
print(a)
```

```
[[ 0.  1.  2.  3.]
 [ 4.  5.  6.  7.]
 [ 8.  9. 10. 11.]
 [12. 13. 14. 15.]
 [16. 17. 18. 19.]]
```

In [19]:
```
# Вариант 2
a.resize(2,10)
print(a)
```

```
[[ 0.  1.  2.  3.  4.  5.  6.  7.  8.  9.]
 [10. 11. 12. 13. 14. 15. 16. 17. 18. 19.]]
```

Следующие операции тоже относятся к группе формоизменения, однако на самом деле форму, т. е. атрибут `shape` массива, не изменяют. Они влияют на то, как массив будет отображен, например, командой `print`.

In [20]:
```
# при выводе форма меняется
print(a.reshape(4,5))
# параметр shape остается прежним
print(a.shape)
```

```
[[ 0.  1.  2.  3.  4.]
 [ 5.  6.  7.  8.  9.]
 [10. 11. 12. 13. 14.]
 [15. 16. 17. 18. 19.]]
(2, 10)
```

Еще один вариант использования метода `reshape` — присваивание одного массива другому с одновременным изменением формы:

In [21]:
```
z = np.zeros(9).reshape(3,3)
print(z)
```

```
[[0. 0. 0.]
 [0. 0. 0.]
 [0. 0. 0.]]
```

Основное достоинство массивов NumPy — векторизация проводимых с ними операций. Это означает, в частности, что арифметические и логические операции выполняются над массивами *поэлементно*.

In [22]:
```
a = np.array([20,30,40,50])
b = np.arange(4)
print(a)
print(b)
```

```
[20 30 40 50]
[0 1 2 3]
```

```
In [23]: a - b
```

```
Out[23]: array([20, 29, 38, 47])
```

```
In [24]: a - 10*b
```

```
Out[24]: array([20, 20, 20, 20])
```

```
In [25]: b**2
```

```
Out[25]: array([0, 1, 4, 9])
```

```
In [26]: a < 35
```

```
Out[26]: array([ True,  True, False, False])
```

Все «классические» математические функции пакета NumPy тоже работают поэлементно; поэтому результатом их работы является массив той же формы и размерности, что и исходный:

```
In [27]: np.sin(np.pi*b/2)
```

```
Out[27]: array([ 0.0000000e+00,  1.0000000e+00,  1.
  ↪2246468e-16, -1.0000000e+00])
```

Еще раз обратим внимание, что операция * для массивов NumPy всегда означает поэлементное умножение, независимо от формы этих массивов.

```
In [28]: a*b
```

```
Out[28]: array([  0,  30,  80, 150])
```

In [29]:
```
a.reshape(2,2) * b.reshape(2,2)
```

Out[29]:
```
array([[  0,  30],
       [ 80, 150]])
```

Операции +=, *= и т. п. также допустимы, при этом новый массив не создается, а меняется исходный.

In [30]:
```
a+=b
a
```

Out[30]:
```
array([20, 31, 42, 53])
```

Перечислим самые распространенные унарные операции, применимые к любому NumPy массиву `a`:

- a.sum()
- a.min()
- a.argmin()
- a.max()
- a.argmax()
- a.mean()

По умолчанию все они работают с массивом, как со сплошным списком чисел, независимо от его формы. Чтобы работать с отдельным «измерением» используется понятие *ось*: параметр `axis` каждой из перечисленных выше функций позволяет вычислить соответствующую характеристику в нужном направлении. Напомним, что нулевая ось соответствует движению вниз по строкам матрицы поэтому, например, результатом вычисления суммы для матрицы размера 3x4 в направлении нулевой оси будет массив из четырех сумм ее столбцов, а сумма этой матрицы в направлении первой оси — это массив из трех сумм ее строк:

In [31]:
```
b = np.arange(12).reshape(3,4)
b
```

Out[31]:
```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11]])
```

In [32]:
```
b.sum()
```

Out[32]:
```
66
```

In [33]:
```
b.sum(axis=0)
```

Out[33]:
```
array([12, 15, 18, 21])
```

In [34]:
```
b.sum(axis=1)
```

Out[34]:
```
array([ 6, 22, 38])
```

5.2 Постановка задачи

В отличие от остальных заданий по курсу данное задание выполняется в малых группах — 3–4 человека. **Цель работы** состоит в сравнительном исследовании нескольких способов решения задачи, условие которой сформулировано в варианте для проектной группы (выдается преподавателем).

В ходе выполнения задания необходимо реализовать три или четыре (в зависимости от количества участников группы) перечисленных в условии варианта решения задачи, подтвердить самостоятельно сконструированными тестовыми примерами правильность разработанных алгоритмов, а потом провести анализ временных затрат на их реализацию двумя способами, условно называемыми «*экстенсивным*» и «*интенсивным*». Варианты проектных заданий представлены в разделе 5.3.

В первом случае («экстенсивный» анализ) решается набор одинаковых (или практически одинаковых) задач, причем количество этих задач растет. Изучается зависимость времени решения набора задач от количества элементов в наборе.

Во втором случае («интенсивный» анализ) решается только одна задача, при этом изменяющимся параметром является размерность этой задачи (например, размеры матрицы). Изучается зависимость времени решения от параметра размерности.

Результаты анализа должны быть представлены в табличном виде. В отчете должна содержаться информация об основных характеристиках компьютера, на котором выполнялись вычисления. В заключении к отчету должны быть сформулированы выводы по итогам работы.

Код, разработанный в ходе проекта, должен содержать подробные комментарии. На итоговом занятии группа представляет результаты своей работы в виде файла отчета, а также в устном виде с использованием презентации (в формате ppt/ppptx или pdf). Во время презентации все участники группы должны продемонстрировать работоспособность разработанного ими кода.

5.3 Варианты заданий

Вариант 0. Вычисление определителя.

Дана квадратная матрица A вещественных чисел размера $m \times m$. Вычислить определитель этой матрицы.

Способы решения:

1. «Стандартный» Python. Рекурсивная функция, основанная на разложении определителя по элементам первой строки.
2. «Стандартный» Python. Функция, основанная на приведении матрицы к диагональному виду методом исключения Гаусса.
3. NumPy. Использование функции `linalg.det` пакета NumPy.

Вариант 1. Умножение матриц.

Даны матрицы A и B размера $m \times k$ и $k \times n$ соответственно. Найти их произведение: $C = A \cdot B$.

Способы решения:

1. «Стандартный» Python. Матрица — это список списков вещественных чисел. Необходимо явно запрограммировать следующую формулу вычисления элемента c_{ij} по известным значениям элементов a_{is} и b_{sj}:
$$c_{ij} = \sum_{s=0}^{k-1} a_{is} b_{sj}.$$
2. «Стандартный» Python. Матрица — это список списков вещественных чисел. Умножение матриц осуществить по схеме:
    - Транспонировать матрицу B, получить B^{T}. Элемент матрицы C находится как произведение строк матриц A и B^{T}. Эти строки, в свою очередь, являются списками длины k.
    - Оформить вычисление произведения строк отдельной функцией:
    $x \cdot y =$ `sum(a*b for a,b in zip(x,y))`
    - Понять, как для транспонирования матрицы X можно использовать выражение `list(zip(*X))`.
3. NumPy. Матрица — это двумерный массив пакета NumPy. Реализовать умножение строки на столбец путем их перемножения как одномерных массивов пакета NumPy (используя, например, функцию `dot` или `vdot`).
4. NumPy. Матрица — это двумерный массив пакета NumPy. Умножение осуществляется операцией `@`.

Вариант 2. След произведения квадратных матриц.

Даны две квадратные матрицы A и B размера $m \times m$. Найти след произведения этих матриц.

Способы решения:

1. «Стандартный» Python. Матрица — это список списков вещественных чисел.
   - Написать свои функции умножения квадратных матриц и вычисления следа матрицы;
   - Оптимизировать программу, учтя, что для вычисления следа произведения матриц нет необходимости вычислять всё произведение: достаточно перемножить только первую строку на первый столбец, потом вторую строку на второй столбец и т. д., а потом найти сумму получившихся чисел.
2. NumPy. Описать матрицы как двумерные массивы пакета NumPy, выполнить их умножение (`@`), вычислить след получившейся матрицы (`trace`).

Вариант 3. Поиск в матрице.

Дана квадратная матрица A вещественных чисел размера $m \times m$. Заменить ее максимальный элемент нулем.

Способы решения:

1. «Стандартный» Python. Матрица определяется как список списков вещественных чисел. С помощью двойного цикла ищется позиция максимального элемента и происходит присваивание нуля элементу, стоящему на этой позиции.
2. NumPy. Решение не использует циклов. Сначала находится список максимумов в каждой строке, потом максимум в этом списке и, следовательно, номер строки, содержащей максимальный элемент. Определение индекса максимума в этой строке позволяет решить задачу.
3. NumPy. Решение не использует циклов и основывается на функциях `argmax` и `ravel`.

Вариант 4. Вектор в кубе.

Дан вектор X большой размерности. Постройте вектор Y, каждый

элемент которого равен кубу соответствующего элемента вектора X.

Способы решения:

1. «Стандартный» Python. Создать новый список из старого с использованием генератора.
2. NumPy. Создать NumPy-массив на основе вектора X. Вычислить массив Y с использованием функции `power`.
3. NumPy. Создать NumPy-массив основе вектора X. Вычислить массив Y двукратным умножением (*) массива X на себя.

Вариант 5. Норма матрицы.

Нормой Фробениуса квадратной матрицы A размера $m \times m$ называется выражение вида

$$\|A\| = \sqrt{\sum_{i,j=1}^{m} a_{ij}^2}, \tag{5.1}$$

где a_{ij} — элементы матрицы.

Дана матрица. Вычислить ее норму.

Способы решения:

1. «Стандартный» Python. Матрица определяется как список списков вещественных чисел. Норма вычисляется на основе двойного цикла (вариант: через суммирование элементов списка квадратов).
2. NumPy. Вычисление нормы основывается на формуле

   $$\|A\| = \sqrt{\mathrm{tr}(\mathrm{A}^\mathrm{T} \cdot \mathrm{A})}, \tag{5.2}$$

   где операция T означает транспонирование матрицы, а tr — вычисление следа матрицы (докажите, что определения (5.1) и (5.2) эквивалентны).
3. NumPy. Функция `norm` (не забудьте проконтролировать, что вычисляется именно норма Фробениуса).

Вариант 6. Наибольшие элементы.

Дан массив вещественных чисел и число n, меньшее размерности мас-

сива. Нужно найти n наибольших элементов массива.

Способы решения:

1. «Стандартный» Python. Упорядочить массив по возрастанию и взять n последних элементов.
2. «Стандартный» Python. Последовательно (n раз) искать в массиве максимальный элемент, копировать его в новый список, и удалять элемент из массива (либо менять значение, например, на «минус бесконечность»).
3. NumPy. Использовать функцию `argsort` пакета NumPy (логика решения совпадает с первым способом).

Вариант 7. Решение системы линейных уравнений с невырожденной матрицей.

Решить систему линейных уравнений $A \cdot x = b$, где A — (невырожденная) матрица коэффициентов системы размера $n \times n$, b — вектор-столбец правых частей, x — вектор-столбец неизвестных.

Способы решения:

1. «Стандартный» Python. Метод Гаусса (написать самостоятельно на «стандартном» Python. Найти готовое решение в Интернете не возбраняется, если оно будет внятно изложено в презентации проекта, и на все вопросы по реализованной программе во время презентации будет дан ответ).
2. NumPy. Правило Крамера (для вычисления определителя использовать функцию `linalg.det` пакета NumPy).
3. NumPy. Функция `linalg.solve`.

Вариант 8. Полиномы.

Вычислить значение полинома

$$p(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

в заданной точке $x = x_*$. Полином определяется списком (массивом) своих вещественных коэффициентов a_k размера $n + 1$.

Способы решения:

1. «Стандартный» Python. Решение «в лоб».
2. «Стандартный» Python. Схема Горнера.
3. NumPy. Функция `polyval`.

Вариант 9. Самый частый элемент.

Дан массив целых положительных чисел, не превосходящих $M = 1000$. Определить, какое из чисел встречается в массиве чаще всего.

Способы решения:

1. «Стандартный» Python. Решение «в лоб». Определить, сколько раз встречается каждый элемент, из полученных значений выбрать наибольшее.
2. «Стандартный» Python. Использовать словарь `dict`.
3. «Стандартный» Python. С использованием вспомогательного массива размера M, который инициализируется нулевыми значениями (идея аналогична словарю, только нерациональна в смысле памяти).
4. NumPy. Функция `bincount` (нерационально в смысле памяти, но должно быть быстро).

Вариант 10. Заменить дубли.

Дан массив целых положительных чисел. Двигаясь слева направо (от начала к концу массива), замените элементы, которые в нем уже встречались, нулями.

Например, массив [1, 1, 3, 1, 2, 4, 2, 2, 2, 2] должен принять вид [1, 0, 3, 0, 2, 4, 0, 0, 0, 0].

Способы решения:

1. «Стандартный» Python. Использовать вспомогательный массив (цикл от начала к концу).
2. «Стандартный» Python. Без использования вспомогательного массива (цикл от конца к началу).
3. NumPy. Функция `unique`.

5.4 Образец выполнения задания

Задание. Дана квадратная матрица A вещественных чисел размера $m \times m$. Вычислить определитель этой матрицы.

Способы решения:

1. Рекурсивная функция, основанная на разложении определителя по элементам первой строки.
2. Функция, основанная на приведении матрицы к диагональному виду методом исключения Гаусса.
3. Использование функции `linalg.det` пакета NumPy.

Первый способ решения. Из курса алгебры известна формула вычисления определителя, основанная на его разложении по элементам первой строки:

$$\det A = \sum_{j=1}^{n} (-1)^{1+j} a_{1j} M_j^1, \tag{5.3}$$

где M_j^1 — *дополнительный минор* к элементу a_{1j}, т. е. определитель матрицы, полученной вычеркиванием из исходной матрицы A первой строки и j-го столбца. Формула (5.3), очевидно, является рекурсивной, в том смысле, что задачу вычисления определителя n-го порядка сводят к задаче вычисления определителя порядка $n-1$. Терминальным является случай $n = 1$, когда определитель матрицы совпадает с ее единственным элементом. Для ускорения работы (уменьшения числа рекурсивных вызовов) терминальным можно считать и случай $n = 2$, когда для определителя есть простая явная формула.

С учетом начала индексации элементов не с единицы, а с нуля, принятой в Python, перед разработкой программного кода формулу (5.3) необходимо скорректировать следующим образом

$$\det A = \sum_{j=0}^{n-1} (-1)^{j} a_{0j} M_j^0. \tag{5.4}$$

При реализации вычислений матрицу считаем объектом, создаваемым средствами пакета NumPy, т. е. мы имеем возможность работать с ее срезами. Для формирования матрицы минора воспользуемся срезами следующим образом. Поскольку минор M_j^0 получается вычеркиванием из матрицы нулевой строки, то в нужный для его вычисления срез войдут все строки, начиная с первой (`A[1:,***]`). Вместо звездочек нужно указать номера всех столбцов, кроме j-го. Для этого удобно воспользоваться возможностью пакета NumPy индексировать массив другим (целочисленным) массивом: вместо звездочек должен быть указан массив (список), состоящий из всех чисел от 0 до $n-1$ включительно, кроме числа j , который можно создать так:

```
[i for i in range(n) if i!=j]
```

или, например, так

```
list(range(j))+list(range(j+1,n))
```

Таким образом, минор M_j^0 запишется как

```
A[1:, list(range(j))+list(range(j+1,n))]
```

Другой способ записи этого же минора, приведенный в коде реализации функции `det1` ниже, основан на использовании функции `hstack` пакета NumPy, позволяющей «объединить» две матрицы в одну, располагая их последовательно по горизонтали. Минор в этом случае составляется из двух блоков: в первом находятся столбцы с номерами от нуля до $j-1$, а во втором — столбцы с номерами от $j+1$ и до последнего, которые потом объединяются в одну матрицу:

```
np.hstack((A[1:,:j], A[1:,j+1:]))
```

Приведенная ниже рекурсивная функция `det1` обеспечивает точное вычисление определителя матрицы в соответствии с (5.4).

```
1  def det1(A):
2      n = A.shape[0]
3      if n == 1:
4      # для матрицы размера 1×1
```

```
    # определителем является ее единственный элемент
        return A[0,0]
    elif n == 2:
    # для матрицы размера 2×2 определитель вычисляется
    # по известной простой формуле
        return A[0,0]*A[1,1] - A[0,1]*A[1,0]
    else: # рекурсия
        return sum((-1)**j*A[0,j]*det1(
            np.hstack((A[1:,:j], A[1:,j+1:])))
                for j in range(n))
```

Второй способ решения. Для вычисления определителя разработана функция `det2`, основанная на приведении матрицы к диагональному виду методом исключения Гаусса.

```
def det2(A):
    X = A.copy() # чтобы не портить исходную матрицу
    n = X.shape[0]
    p = 1
    for k in range(n-1):
        # ищем максимум в k-м столбце,
        #       начиная с k-го элемента
        i = np.abs(X[k:,k]).argmax()
        amax = X[k+i,k]
        if abs(amax) < 1e-15:
            # все числа в столбце очень малы,
            # можно считать определитель нулем
            return 0
        p *= amax
        # если нужно, меняем строки k и k+i
        if i!=0:
            p *= -1
            X[k,k:],X[k+i,k:] =  X[k+i,k:],X[k,k:].copy()
        # делаем нули в k-м столбце под диагональю
        for i in range(k+1,n):
            X[i,k:] = X[i,k:]-X[k,k:]*X[i,k]/amax
    return p*X[n-1,n-1]
```

Третий способ решения. Данный способ не требует самостоятельного программирования. Достаточно подключить пакет NumPy, создать

матрицу средствами данного пакета и вычислить ее определитель, используя функцию `linalg.det`.

Тестирование. Для проведения тестирования используем матрицы размера 3×3 следующего вида:

$$M_1 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}, \quad M_2 = \begin{pmatrix} 5 & 5 & 5 \\ 0 & 5 & 5 \\ 0 & 0 & 5 \end{pmatrix}, \quad M_3 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 2 & 1 & 3 \end{pmatrix},$$

а также матрицу размера 4×4 вида

$$M_4 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 2 & 2 \\ 1 & 2 & 3 & 3 \\ 1 & 2 & 3 & 4 \end{pmatrix}.$$

Точные значения их определителей имеют вид:

$$\det(M_1) = 0, \quad \det(M_2) = 125, \quad \det(M_3) = -12, \quad \det(M_4) = 1.$$

Результаты работы программы подтверждают правильность реализации функций.

```
1 import numpy as np
2 M1 = np.array([[1,2,3], [4,5,6], [7,8,9]])
3 M2 = np.array([[5,5,5], [0,5,5], [0,0,5]])
4 M3 = np.array([[1,2,3], [3,2,1], [2,1,3]])
5 M4 = np.array([[1,1,1,1],[1,2,2,2],[1,2,3,3],[1,2,3,4]])
6 print("det(M1): ", det1(M1), det2(M1), np.linalg.det(M1))
7 print("det(M2): ", det1(M2), det2(M2), np.linalg.det(M2))
8 print("det(M3): ", det1(M3), det2(M3), np.linalg.det(M3))
9 print("det(M4): ", det1(M4), det2(M4), np.linalg.det(M4))
```

```
det(M1):  0 6.661338147750937e-16 -9.51619735392994e-16
det(M2):  125 125.0 124.99999999999994
```

```
det(M3):  -12 -12.000000000000002 -12.000000000000005
det(M4):  1 0.9999999999999999 1.0
```

Исследование скорости.

1. *Экстенсивный* способ. Ограничимся матрицами размера 5×5 и сравним время работы каждой из этих функций при вычислении N определителей. Матрицы будем создавать из случайных вещественных чисел в диапазоне от 0 до 1 с использованием функции `random()` одноименного пакета. Для измерения времени будем использовать функцию `time()` из одноименного пакета.

Фрагмент программы, отвечающий за это сравнение, приведен ниже:

```
1  # подключение пакетов
2  from random import random
3  import time
4  # количество определителей
5  N = 100
6  # таймеры
7  t1 = 0
8  t2 = 0
9  t3 = 0
10 for i in range(N):
11     # генерируем матрицу размера 5×5
12     M=[[random() for j in range(5)] for i in range(5)]
13     # преобразовываем ее к NumPy матрице
14     M_np = np.array(M)
15     t = time.time() # фиксируем время старта (1-й способ)
16     a = det1(M_np)  # вычисляем определитель (1-й способ)
17     t1 += time.time()-t # определяем затраченное время
18     t = time.time() # фиксируем время старта (2-й способ)
19     a = det2(M_np)  # вычисляем определитель (2-й способ)
20     t2 += time.time()-t # определяем затраченное время
21     t = time.time() # фиксируем время старта (3-й способ)
22     # вычисляем определитель (3-й способ)
23     a = np.linalg.det(M_np)
24     t3 += time.time()-t # определяем затраченное время
25 # печать результата
26 print('t1 =', t1, ' t2 =', t2, ' t3 = ', t3)
```

Для вычислений здесь и далее использовался персональный компьютер с процессором Intel i7-10700K, 3.8 GHz, ОЗУ 16 Гб, 64-разрядная операционная система Windows 10. Jupyter Notebook файл был открыт в Microsoft Edge. Данные о проведенных расчетах занесены в таблицу:

Таблица 5.1 — Время расчетов «*экстенсивным*» способом, с

N	100	500	1000	2000	5000	10000	20000	50000	100000
t_1	0.0470	0.1910	0.3821	0.7710	1.918	3.839	7.701	19.23	38.98
t_2	0.0050	0.0341	0.0599	0.0980	0.2650	0.5624	1.088	2.615	5.257
t_3	0.0010	0.0010	0.0050	0.0190	0.0300	0.0530	0.1340	0.3700	0.7040

2. *Интенсивный* способ. Изучим, как время работы каждой из функций зависит от размера матрицы. Для тестирования будем использовать матрицу следующего вида, определитель которой равен единице для любого значения n:

$$\begin{pmatrix} 1 & 1 & 1 & \dots & 1 & 1 \\ 1 & 2 & 2 & \dots & 2 & 2 \\ 1 & 2 & 3 & \dots & 3 & 3 \\ \dots & \dots & \dots & \dots & \dots & \dots \\ 1 & 2 & 3 & \dots & n-1 & n-1 \\ 1 & 2 & 3 & \dots & n-1 & n \end{pmatrix}.$$

Для элемента m_{ij} этой матрицы существует простая формула

$$m_{ij} = \min(i, j).$$

Фрагмент программы, отвечающий за это сравнение, приведен ниже:

```
1  N = 50 # пример для матрицы 50×50
2  # генерируем матрицу размера N×N
3  M=[[min(i,j) for j in range(1,N+1)]
4     for i in range(1,N+1)]
5  M_np = np.array(M) # преобразовываем ее к NumPy матрице
6  t = time.time() # фиксируем время старта (1-й способ)
7  a1 = det1(M_np)     # вычисляем определитель (1-й способ)
```

```
8   t1 = time.time()-t # определяем затраченное время
9   t = time.time() # фиксируем время старта (2-й способ)
10  a2 = det2(M_np)    # вычисляем определитель (2-й способ)
11  t2 = time.time()-t # определяем затраченное время
12  t = time.time() # фиксируем время старта (3-й способ)
13  # вычисляем определитель (3-й способ)
14  a3 = np.linalg.det(M_np)
15  t3 = time.time()-t # определяем затраченное время
16  # печатаем значения счетчиков времени
17  print('t1 =', t1, ' t2 =', t2, ' t3 = ', t3)
18  # и значения определителей
19  print(a1, a2, a3)
```

Данные о проведенных расчетах занесены в таблицу:

Таблица 5.2 — Время расчетов «*интенсивным*» способом, с

N	5	6	7	8	9	10	100	1000	2000	5000
t_1	0.001	0.005	0.022	0.134	1.102	11.05	—	—	—	—
t_2	0.000	0.000	0.000	0.000	0.000	0.000	0.024	0.704	17.61	522.1
t_3	0.000	0.000	0.000	0.000	0.000	0.000	0.003	0.031	0.081	0.641

ЗАКЛЮЧЕНИЕ

Вывод 1. Первый способ вычисления определителя существенно проигрывает в скорости. Для матриц размера 5×5 его можно рекомендовать использовать лишь в том случае, если количество вычислений не превосходит 1000. Если необходимы десятки тысяч вычислений, его использование потребует порядка десяти секунд. Третий способ является, как правило, самым эффективным; для матриц размера 5×5 он в 7–10 раз превосходит по скорости второй способ.

Вывод 2. Первый способ вычисления определителя существенно проигрывает в скорости, его можно рекомендовать использовать лишь в том случае, если размеры матрицы не превосходит 10×10. Скорость работы второго способа вполне удовлетворительна до размеров 100×100, а для больших матриц нужно пользоваться только третьим способом, которой даже для матриц размером 5000×5000 работает менее секунды.

Общий вывод. Первый способ вычисления определителя можно рекомендовать использовать лишь в том случае, если принципиально важным является точное значение определителя матрицы небольшого размера. Во всех остальных случаях нужно пользоваться возможностями функции `det` пакета NumPy.

6 Индивидуальная работа «Основные возможности пакета Matplotlib»

6.1 Знакомство с Matplotlib

Matplotlib — библиотека для визуализации данных с помощью 2D и 3D графики. Разработана в 2003 году Джоном Хантером с целью «подражания» командам пакета MATLAB, базируется на принципах объектно-ориентированного программирования. На сегодняшний день является гибким пакетом, который вместе с NumPy, SciPy и IPython предоставляет широкие возможности обработки данных различного вида. Пакет поддерживает многие виды графиков и диаграмм, позволяет создавать анимированные изображения. Полезную информацию о возможностях пакета Matplotlib можно почерпнуть из источников [1, 13, 26, 30, 32].

К основным поддерживаемым форматам изображений относятся:

- **EPS**: Encapsulated Postscript;
- **JPG**, **JPEG**: Joint Photographic Experts Group;
- **PDF**: Portable Document Format;
- **PGF**: PGF code for LaTeX;
- **PNG**: Portable Network Graphics;
- **PS**: Postscript;
- **RAW**, **RGBA**: Raw RGBA bitmap;
- **SVG**: Scalable Vector Graphics;
- **TIFF**: Tagged Image File Format.

Pyplot — модуль (набор команд) библиотеки Matplotlib, позволяющий осуществлять работу с ней почти аналогично MATLAB. Основным объектом при работе с Pyplot является *Фигура* (Figure), определение которой будет дано позже.

6.2 Постановка задачи

Общие указания. Общий отчет должен начинаться титульным листом. Отчет по каждой из трех задач должен состоять из условия задачи и текста программы, строящей требуемый график. График должен быть оформлен в соответствии с требованиями, приведенными в задании. При отсутствии требований оформление может быть произвольным. В отчет по первой задаче дополнительно включается табличная информация, использовавшаяся для построения графика.

Задача 1. Найдите в сети Интернет данные в соответствии с заданием (таблица 6.1) и оформите результат диаграммой требуемого типа. В отчет включите также таблицу с найденной числовой информацией, которая послужила основой диаграммы. При оформлении отчета не забудьте указать источник получения информации (адреса веб-сайта или сайтов).

Таблица 6.1 — Данные и тип диаграммы

№	Данные	Тип диаграммы
1	Десять самых длинных рек мира (длина в км)	Столбиковая диаграмма, вертикальные столбики разного цвета
2	Десять самых больших по населению городов мира (население, млн. человек)	Столбиковая диаграмма, вертикальные столбики разного цвета
3	Десять самых больших по площади стран Африки (площадь, кв. км)	Столбиковая диаграмма, вертикальные столбики разного цвета
4	Десять самых больших по площади стран Азии (площадь, кв. км)	Столбиковая диаграмма, горизонтальные столбики разного цвета

5	Десять самых больших по населению европейских столиц (население, млн. человек)	Столбиковая диаграмма, горизонтальные столбики разного цвета
6	Десять самых высоких деревьев в мире (высота, м)	Столбиковая диаграмма, вертикальные столбики разного цвета
7	Десять самых быстрых животных (скорость, км/ч)	Столбиковая диаграмма, горизонтальные столбики разного цвета
8	Десять самых быстрых птиц (скорость, км/ч)	Столбиковая диаграмма, горизонтальные столбики разного цвета
9	Десять самых больших городов России (население, млн. человек)	Столбиковая диаграмма, горизонтальные столбики разного цвета
10	Десять самых больших по площади морей (площадь, тыс.кв.км.)	Столбиковая диаграмма, вертикальные столбики разного цвета
11	Десять самых маленьких стран мира (население, тыс. чел.)	Столбиковая диаграмма, вертикальные столбики разного цвета
12	Десять самых маленьких стран мира (площадь, кв. км)	Столбиковая диаграмма, вертикальные столбики разного цвета
13	Десять самых маленьких по площади морей (площадь, тыс.кв.км.)	Столбиковая диаграмма, горизонтальные столбики разного цвета

14	Десять самых больших городов Китая (население, млн. человек)	Столбиковая диаграмма, вертикальные столбики разного цвета
15	Десять самых глубоких мест на Земле (глубина, м)	Столбиковая диаграмма, вертикальные столбики разного цвета
16	Площадь районов Ростова-на-Дону (площадь, кв. км)	Круговая диаграмма, сектора разного цвета
17	Население районов Ростова-на-Дону (население, тыс. чел.)	Круговая диаграмма, сектора разного цвета
18	Население регионов России, входящих в Южный федеральный округ (население, млн. чел.)	Круговая диаграмма, сектора разного цвета
19	Площадь регионов России, входящих в Южный федеральный округ (площадь, тыс. кв. км)	Круговая диаграмма, сектора разного цвета
20	Население десяти административных округов Москвы (численность население, тыс. чел.)	Круговая диаграмма, сектора разного цвета
21	Площадь десяти административных округов Москвы (площадь, кв. км.)	Круговая диаграмма, сектора разного цвета
22	Население восемнадцати районов Санкт-Петербурга (численность населения, тыс. чел.)	Круговая диаграмма, сектора разного цвета

23	Итоги выборов в государственную Думу России в 2011 году по семи политическим партиям (процент избирателей, %)	Круговая диаграмма, сектора разного цвета
24	Площадь стран Южной Америки, не считая колонии и спорные острова (площадь, тыс. кв. км)	Круговая диаграмма, сектора разного цвета
25	Население стран Южной Америки, не считая колонии и спорные острова (население, млн. чел.)	Круговая диаграмма, сектора разного цвета
26	Площадь районов Нью-Йорка (площадь, кв. км)	Круговая диаграмма, сектора разного цвета
27	Население районов Нью-Йорка (численность, тыс. чел.)	Круговая диаграмма, сектора разного цвета
28	Площадь земель Германии (площадь, тыс. кв. км)	Круговая диаграмма, сектора разного цвета
29	Население земель Германии (численность населения, млн. чел.)	Круговая диаграмма, сектора разного цвета
30	Площадь французских регионов (провинций) с 01.01.2016 г., не считая «заморские» (площадь, тыс. кв. км)	Круговая диаграмма, сектора разного цвета
31	Население французских регионов (провинций) с 01.01.2016 г., не считая «заморские» (численность населения, тыс. чел.)	Круговая диаграмма, сектора разного цвета

Задача 2. Постройте кривую, заданную уравнением в полярной системе координат, при заданных значениях параметров, используя для

построения графика не менее 10000 точек. Призвав на помощь воображение, придумайте название построенному изображению. Добавьте заголовок с этим названием на построенный график.

Таблица 6.2 — Кривая в полярной системе координат

№	**Уравнение**	**Значения параметров**
1, 13, 25	$r = e^{\sin\varphi} - a\cos 4\varphi + \sin^n \frac{2\varphi - \pi}{24}$	$a = 1,\ n = 6$
2, 14	$r = e^{\sin\varphi} - a\cos 4\varphi + \sin^n \frac{2\varphi - \pi}{24}$	$a = 2,\ n = 5$
3, 15	$r = e^{\sin\varphi} - a\cos 4\varphi + \sin^n \frac{2\varphi - \pi}{24}$	$a = 3,\ n = 2$
4, 16, 26	$r = 2 - a\sin\varphi + \sin\varphi \frac{\sqrt{\lvert\cos\varphi\rvert}}{\sin\varphi + b}$	$a = 0.2,\ b = 1.1$
5, 17	$r = 2 - a\sin\varphi + \sin\varphi \frac{\sqrt{\lvert\cos\varphi\rvert}}{\sin\varphi + b}$	$a = 2,\ b = 1.4$
6, 18, 27	$r = 2 - a\sin\varphi + \sin\varphi \frac{\sqrt{\lvert\cos\varphi\rvert}}{\sin\varphi + b}$	$a = 3,\ b = 1.5$
7, 19, 28	$r = (1 + \sin\varphi)(1 + a\cos 7\varphi)(1 + b\sin 77\varphi)$	$a = 0.8,\ b = 0.07$
8, 20	$r = 1 + a\cos k\varphi + 4\sin^2 k\varphi + 3\sin^4 k\varphi$	$a = 7, k = 5.5,$ $\phi \in [0, 4\pi]$
9, 21	$r = 1 + a\cos k\varphi + 4\sin^2 k\varphi + 3\sin^4 k\varphi$	$a = 3, k = 7.25,$ $\varphi \in [0, 8\pi]$
10, 22, 29	$r = \frac{100}{100 + \left(\varphi - \frac{\pi}{2}\right)^8}\left(2 - \sin(a\varphi) - \frac{1}{2}\cos b\varphi\right)$	$a = 7, b = 30,$ $\varphi \in \left[-\frac{\pi}{2}, \frac{3\pi}{2}\right]$
11, 23	$r = a + \frac{\lvert\cos n\varphi\rvert + (1/2 - 2\lvert\sin n\varphi\rvert)}{2 + 8\lvert\sin 2n\varphi\rvert}$	$a = 1,\ n = 3$
12, 24, 30	$r = 5 - \frac{2.5\sin\varphi}{\sqrt{0.2 + \lvert\cos\varphi\rvert}} + \sin\varphi + \cos 2\varphi + 0.3\sin 70\varphi$	

Задача 3. Постройте, используя не менее 2000 точек, соответствующий номеру варианта график функции $y = f(x)$ из индивидуального задания № 2 (таблицы 3.3, 3.4, 3.5). Проверьте, совпадает ли он с тем, который приведен в разделе 3.5. Нанесите на график сетку. Отметьте на графике стрелками разных цветов первую точку локального минимума и последнюю точку локального максимума.

6.3 Методические указания

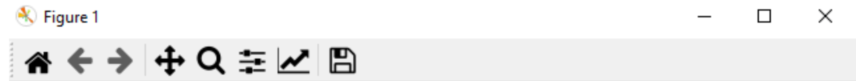
Пример 1. Базовые команды.

```
1  # подключение библиотеки NumPy
2  import numpy as np
3  # и модуля Pyplot библиотеки Matplotlib
4  import matplotlib.pyplot as plt
5  # массив из 100 равноотстоящих точек на отрезке [0, 10]
6  x = np.linspace(0, 10, 100)
7  y = np.power(x, 2) # значения y(x), вычисленные поточечно
8  plt.plot(x, y) # строим график по двум массивам
9  plt.show() # отображаем его
```

Полезным инструментом при работе в Jupyter Notebook является регулировка появления картинок. Команда `%matplotlib inline` отвечает за расположение графиков непосредственно рядом с ячейками с кодом. В свою очередь, если необходимо появление картинок во всплывающих окнах, можно воспользоваться командой `%matplotlib qt`. Использование этих команд не является обязательным. Результат работы указанной выше программы представлен на рисунке 6.1.

Вопросы и задания.

1. Для чего служит функция `linspace`?
2. График какой функции изображен на рисунке?
3. Как изменится график, если строку 5 записать в виде `plt.plot(y)`?
4. Каково назначение кнопок панели навигации окна «Figure 1»?



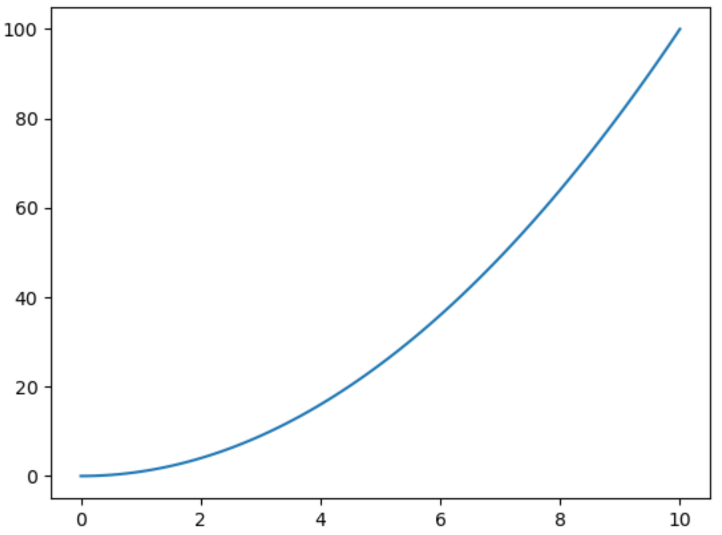


Рисунок 6.1 — График функции $y(x)$

Пример 2. Работа с метками и заголовками.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  x = np.linspace(0, 10, 1000)
4  y = np.power(x, 2)
5  plt.plot(x, y)            # строим график по 2 массивам
6  plt.xlim((0, 5))          # отрезок по OX
7  plt.ylim((0, 30))         # отрезок по OY
8  plt.xlabel('my x label') # имя оси OX
9  plt.ylabel('my y label') # имя оси OY
10 # задаем подпись с именем графика
11 plt.title('text+TEX: $y = x^2$')
12 plt.tight_layout()
13 plt.savefig('figure.pdf')
```

Вопросы и задания.

1. Проверьте, отображаются ли в заголовке и метках русские буквы.
2. Проверьте, можно ли использовать математическую нотацию TEX в подписях осей.
3. Выясните самостоятельно, что определяют строчки 6 и 7.

4. Выясните, для чего служит код в строке 12. Для этого попробуйте, убрав его, уменьшить размер окна. Что происходит с подписями осей?
5. В какую папку сохраняет файл код в строке 13? Что произойдет, если вместо .pdf в этой строке написать .png? А если .jpg?

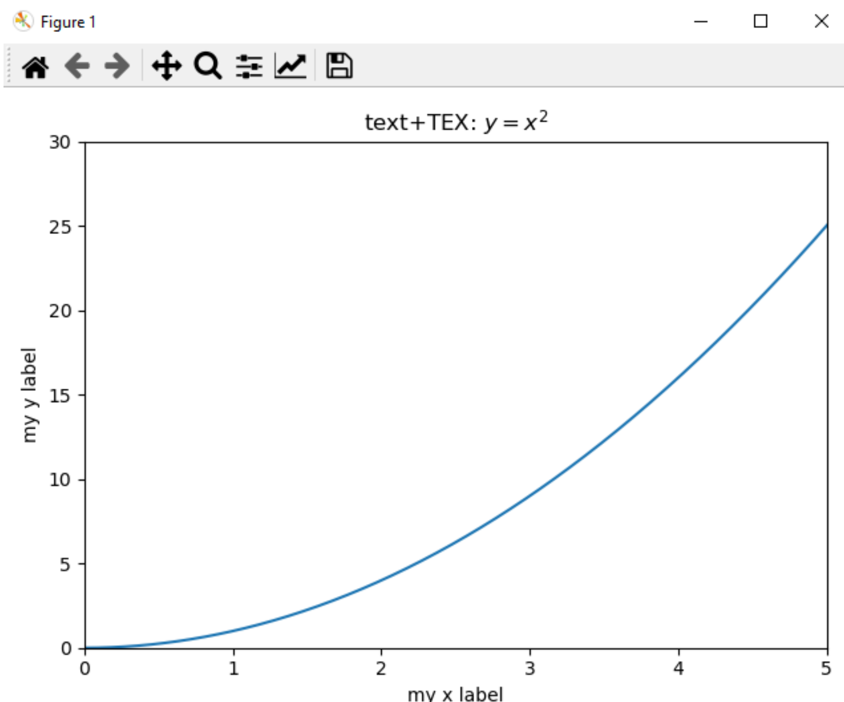


Рисунок 6.2 — Настройка меток и заголовков для графика

Пример 3. Построение графиков нескольких функций в одном окне и добавление «легенды».

```
1   import numpy as np
2   import matplotlib.pyplot as plt
3   x = np.linspace(0, 10, 50)
4   y2 = np.power(x, 2) # график y = x^2
5   y3 = np.power(x, 3) # график y = x^3
6   plt.plot(x, y2, 'b-', label = '$x^2$')
7   plt.plot(x, y3, 'go', label = '$x^3$')
8   plt.xlim((0, 5))    # x in [0, 5]
9   plt.ylim((0, 30))   # y in [0, 30]
10  plt.xlabel('$x$')
11  plt.ylabel('$f(x)$')
```

```
12 plt.title('$x^2$ and $x^3$')
13 plt.legend() # отображение легенды (из label)
14 plt.savefig('figure3.pdf') # сохранение
```

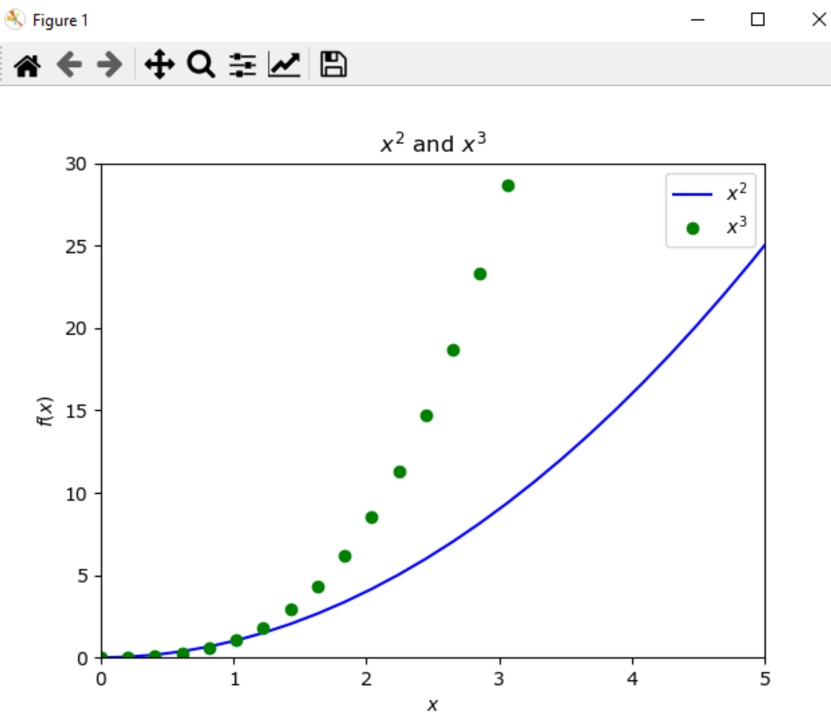


Рисунок 6.3 — Несколько функций на одном рисунке с «легендой»

В последнем примере необходимо обратить внимание на то, что третий параметр функции `plot` отвечает за цвет и стиль линии. Подробное описание этой функции приведено на официальном сайте Matplotlib в справочной системе pyplot.plot. Несколько примеров этих характеристик представлено в таблице 6.3.

Более сложные чертежи.

Иерархия.

Пакет Matplotlib оперирует такими понятиями как *фигура* (`figure`) и *область рисования* (`axes`). До сих пор мы пользовались ими неявно: для быстрого построения графиков это очень удобно. В более сложных случаях — когда, например, мы хотим каждый график разместить в своем окне — приходится явно вызывать команды `figure`, `subplot` и

Таблица 6.3 — Цвета и стили

Символ	Цвет линии
'b'	синий
'g'	зеленый
'r'	красный
'y'	желтый
'k'	черный

Символ	Стиль линии
'-'	сплошная линия
'--'	штриховая линия
'-.'	штрих-пунктир
':'	пунктир
'o'	круговые маркеры
'v'	треугольные маркеры
's'	квадратные маркеры

`axes`.

Понятие *фигуры* (`figure`) в Matplotlib означает «все окно», «вся область рисования». Внутри фигуры могут помещаться *под-фигуры*, или *под-графики* (`subplots`). Функция `subplot` четко позиционирует эти графики как ячейки таблицы. В свою очередь, функция `axes` позволяет произвольно разместить новую систему координат (новую область рисования графика) внутри фигуры.

Еще раз укажем, что фигуры и координатные системы могут вызываться неявно. Когда мы вызываем функцию `plot`, Matplotlib вызывает `gca()`, чтобы узнать текущую координатную систему, а `gca`, в свою очередь, вызывает `gcf()`, чтобы получить информацию о текущей фигуре. Если никакой фигуры не создано явно, то она создает ее сама, чтобы внутри нее создать единственный под-график (строго говоря, этот под-график соответствует вызову функции `subplot(111)`). Переходим к подробностям.

Фигуры (`figure`).

Фигура — это окно, которое имеет заголовок «Figure №». Фигуры нумеруются с единицы (в отличие от всей остальной нумерации Python, начинающейся с нуля). Это связано с происхождением Matplotlib — он ориентирован на пакет MATLAB. Фигура определяется следующими параметрами:

Таблица 6.4 — Параметры `figure`

Аргумент	Значение по умолчанию	Описание
`num`	1	Номер фигуры
`figsize`	`figure.figsize`	Размер фигуры в дюймах (ширина, высота)
`dpi`	`figure.dpi`	Разрешение фигуры (число точек на дюйм)
`facecolor`	`figure.facecolor`	Цвет фона фигуры
`edgecolor`	`figure.edgecolor`	Цвет зоны вокруг графиков
`frameon`	`True`	Рисовать ли рамку вокруг фигуры

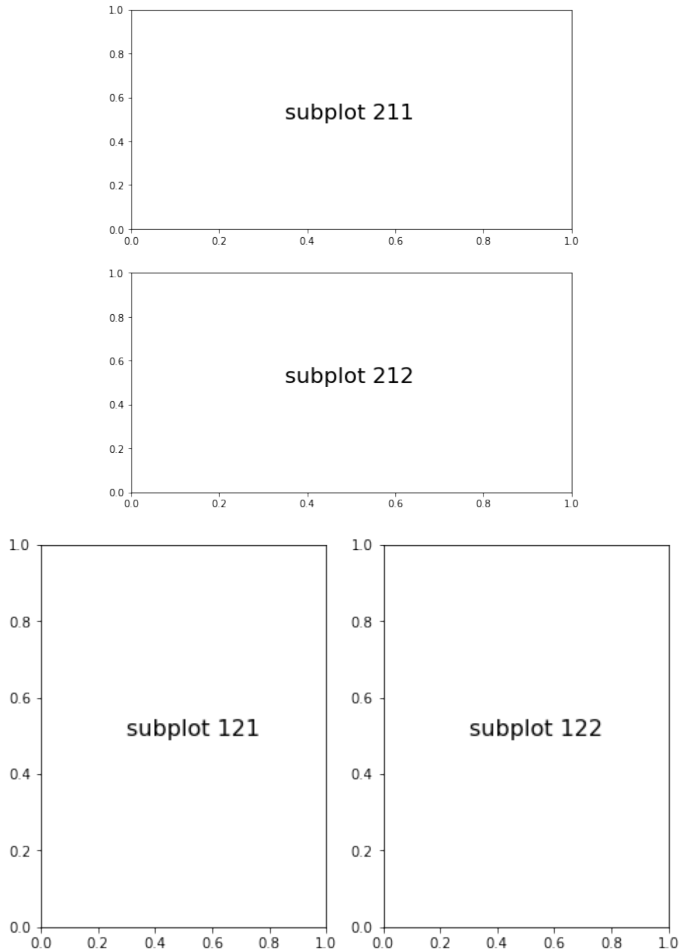
Значения по умолчанию определены в ресурсном файле и меняются очень редко. Исключение составляет номер фигуры. Работая в Windows, фигуру можно закрыть обычным щелчком на кнопке с крестиком, но это же можно сделать и программно, вызвав метод `close()`. В зависимости от аргумента можно закрыть текущую фигуру (`close()`), фигуру с заданным номером (`close(N)`) или все фигуры (`close(all)`).

Под-графики (`subplots`).

С помощью функции `subplot` можно располагать графики на регулярной сетке. Достаточно указать количество рядов, количество колонок и номер графика.

Например, график с двумя рядами и одним столбиком создается командами `subplot(211)` и `subplot(212)`.

Если нужно создать два рядом расположенных графика, то создается один ряд с двумя колонками с помощью функций `subplot(121)` и `subplot(122)`. Результат выглядит примерно так:



Таким образом, можно упорядочить любое количество графиков. Например, четыре графика на одной фигуре могут быть упорядочены с помощью `subplot(221)`, `subplot(222)`, `subplot(223)` и `subplot(224)`, а результат их взаимного расположения приведен на рис. 6.4.

Пример 4. Фигуры и под-графики.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3
4  # первая фигура
5  plt.figure(1)
6  # первый под-график первой фигуры
7  plt.subplot(211)
8  plt.plot([1, 2, 3])
9  # добавляем ему название
10 plt.title('Plot 1 at Fig. 1')
```

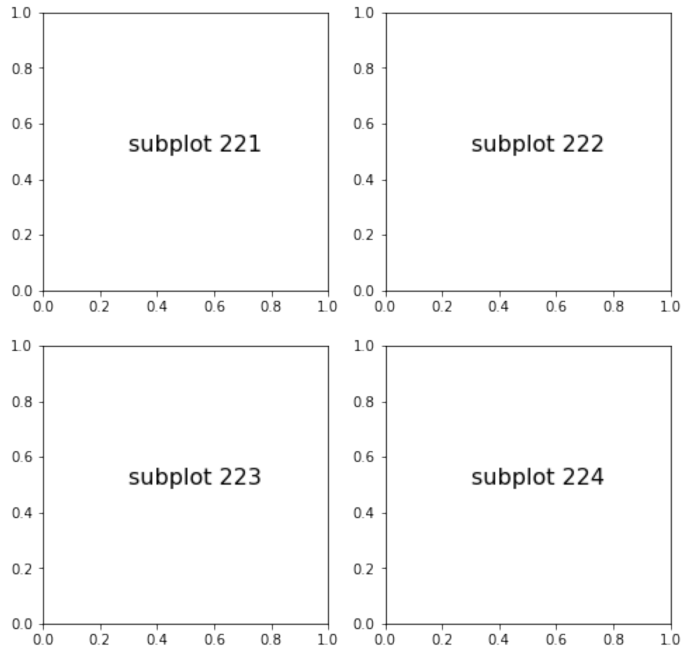


Рисунок 6.4 — Два столбца и две строки

```
11 # второй под-график первой фигуры
12 plt.subplot(212)
13 plt.plot([4, 5, 6])
14
15 # вторая фигура
16 plt.figure(2)
17 # единственный под-график (111) - по умолчанию
18 plt.plot([0,1,4,9,16,25,36,49,64,81])
```

В результате работы программы будет получено две фигуры, на первой из которых будет два графика, а на второй — один (рис. 6.5).

Область рисования (`axes`).

Функция `axes` также служит для позиционирования очередного графика, но в отличие от `subplot` позволяет поместить новый график в любом месте фигуры. Положение и размер области рисования задается списком из четырех параметров: отступ слева и отступ снизу (для точки, являющейся началом отсчета), ширина и высота. Эти параметры изменяются в пределах от нуля до единицы и измеряются в долях от соответствующего размера всей фигуры.

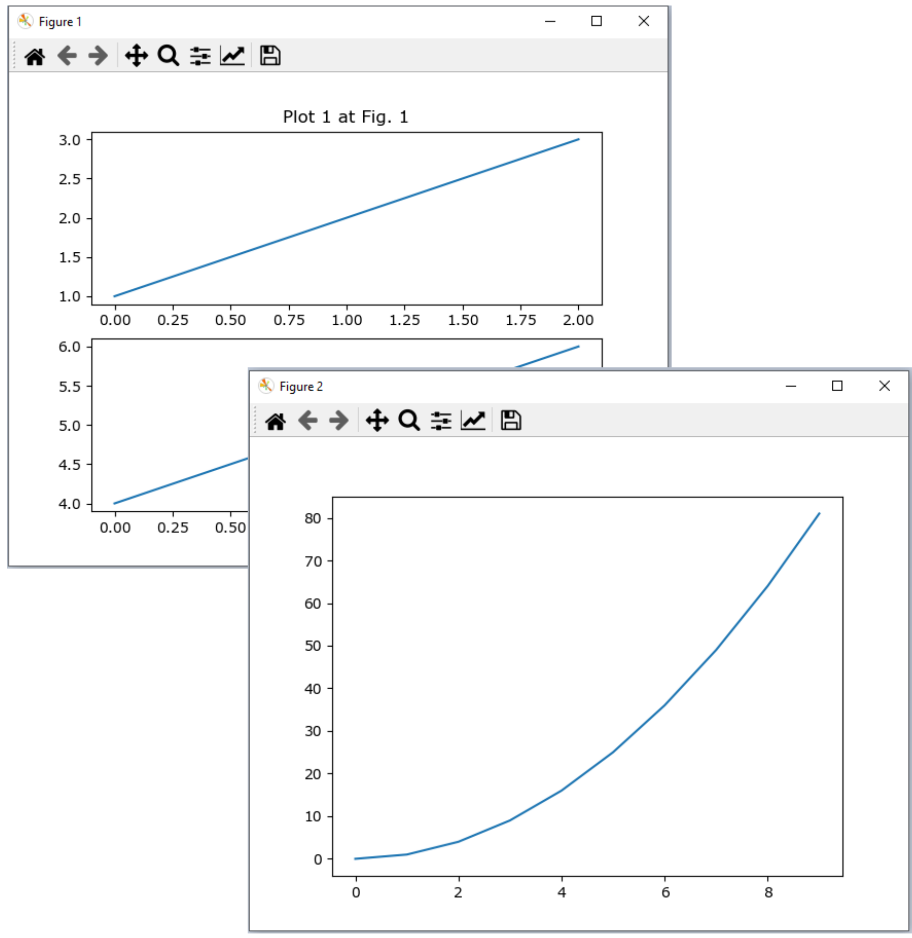


Рисунок 6.5 — Работа с фигурами

Пример 5. Меньший график внутри большего.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  plt.plot(np.arange(11)**2)     # создали большой график
4  plt.title('main window')       # название
5  plt.axes([0.2,0.5,0.25,0.25]) # оси маленького графика
6  plt.title('small window')      # название
7  plt.plot(np.sin(np.linspace(0, 20, 100)))
```

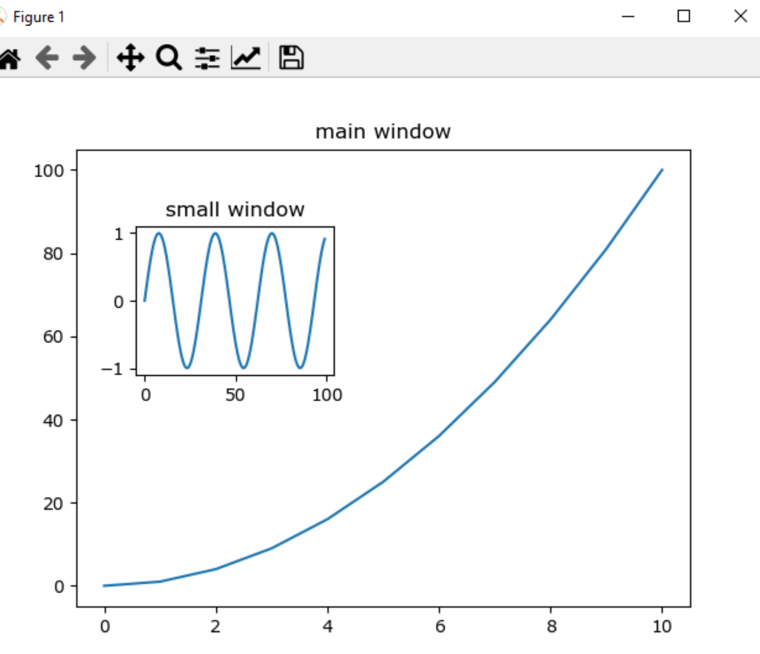


Рисунок 6.6 — Вложенные графики

Дополнительные возможности и «украшения» графиков.

Добавление текста.

Ранее мы уже добавляли текст на график, используя функции `title`, `xlabel`, `ylabel`, `label` и `legend`. Для размещения текста в произвольной заданной позиции на чертеже используются еще две функции, различающиеся способом задания координат выводимого текста. Функция `text` основывается на тех координатах, которые используются на графике (координаты данных), а функция `figtext` использует координаты фигуры — изменяющиеся в пределах от нуля до единицы относительные координаты точки на чертеже.

Для добавления текста на график часто используется также и функция `annotate`, позволяющая отметить определенное место на чертеже. Ее основными параметрами, кроме текстовой строки, служат координаты выделяемой (аннотируемой) точки и координаты размещения текста. Кроме того, в ней можно задать характеристики стрелки, указывающей на выделенное место чертежа (см. пример 6).

Пример 6. Простейшие функции вывода текста.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  x = np.linspace(1, 3, 100)
4  y = np.sin(10*x)/x**2
5  plt.ylim((-1, 1))
6  plt.plot(x, y)
7  plt.text(2, 0, 'Text in the middle')
8  plt.figtext(0.75, 0.8, 'Text in Top-Right')
9  plt.annotate('global max',
10              xytext = (1.5, 0.7), xy = (1.4, 0.5),
11              arrowprops=dict(facecolor='red', shrink=0.1))
```

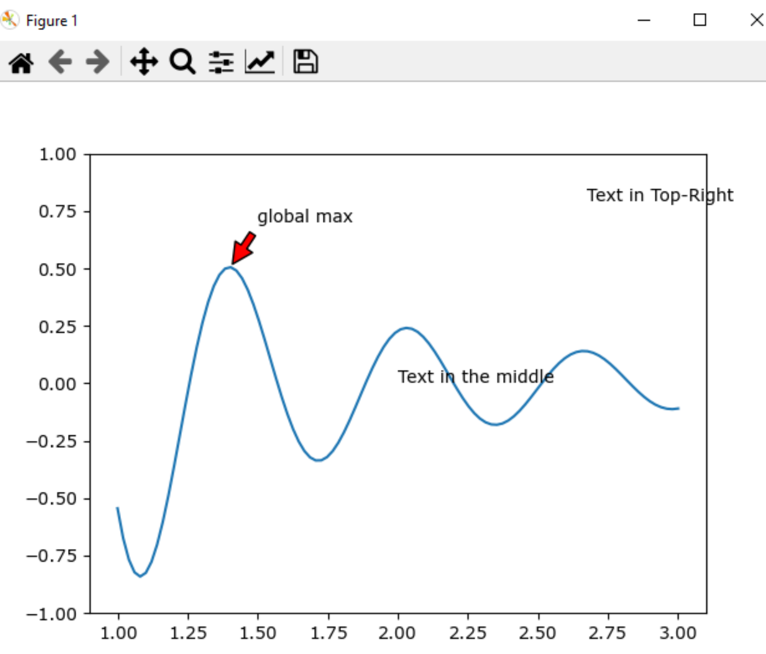


Рисунок 6.7 — Размещение текста и стрелок

Вопросы и задания.

1. Добавьте на чертеж из примера 6 аннотацию «global min» со стрелкой в соответствующем месте.
2. Исследуйте возможность добавления на график текстов и пояснений на русском языке.
3. Для чего используется параметр `shrink`?

Деления на осях.

Правильно сформированные и отрисованные деления оси (черточки, `ticks`) являются важной составной частью графика, подготовленного к публикации. Пакет Matplotlib работу с этими делениями разделяет на две части: задание положения делений и форматирование, или определение внешнего вида, подписей к ним. За работу с делениями отвечает область рисования (`axes`) — задание положения и вывод на экран делений реализованы как ее методы (функции), поэтому для того, чтобы задавать положение и подписи к делениям, необходимо явно создать эту область рисования, например, так:

```
ax = plt.axes()
```

или так

```
fig, ax = plt.subplots()
```

После этого можно задавать характеристики маркеров как для оси x (`ax.xaxis`), так и для оси y (`ax.yaxis`). В таблице 6.5 приведены некоторые широко используемые функции для задания положения делений и формата подписей (меток) под ними.

Таблица 6.5 — Настройка делений и подписей

Положение делений		**Формат подписей**	
NullLocator	Нет делений	NullFormatter	Подписи отсутствуют
LinearLocator	Равномерно распределенные деления от минимального до максимального значения	FormatStrFormatter	Внешний вид подписи задается форматной строкой (см. Пример 7)

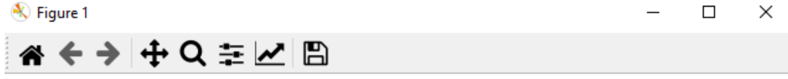
LogLocator	Логарифмическая шкала расположения делений	ScalarFormatter	Формат подписи определяется автоматически (это значение используется по умолчанию для скалярных величин)
MultipleLocator (base)	Деления выводятся в позициях, получаемых умножением на значение параметра base — цены деления	DateFormatter	Используется для нестандартного форматирования подписей — дат

Данные функции находятся в библиотеке `matplotlib.ticker`.

В научных публикациях принято различать два вида делений на осях: основные (`major`), рядом с которыми при необходимости располагаются подписи, и вспомогательные (`minor`). Соответственно и цену деления нужно задавать для каждой из систем делений отдельно.

Пример 7. Работа со шкалой.

```
1  # Сначала построим график функции
2  # с автоматической разметкой осей
3  import numpy as np
4  import matplotlib.pyplot as plt
5
6  X = np.linspace(-20, 20, 1024)
7  Y = np.sinc(X)
8  plt.plot(X, Y)
9  plt.show()
```



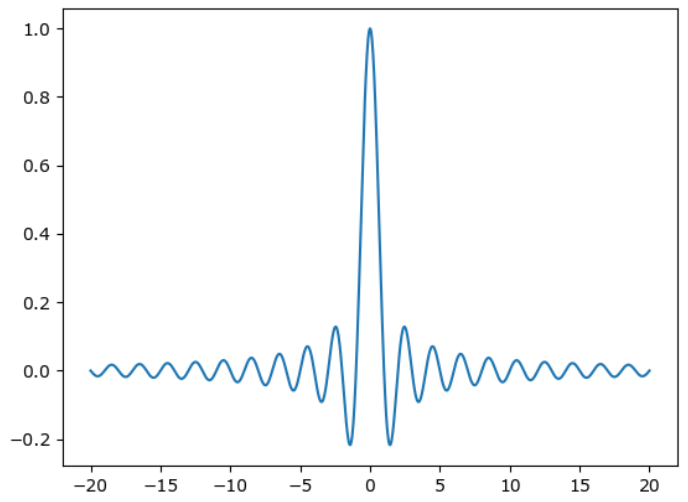


Рисунок 6.8 — Автоматическая разметка осей

```
# Теперь добавим разметку в "ручном" режиме
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
X = np.linspace(-20, 20, 1024)
Y = np.sinc(X)
ax = plt.axes()

# Работаем с осью X
# Цена основных делений - 10
ax.xaxis.set_major_locator(ticker.MultipleLocator(10))
# Формат подписей: 1 знак после запятой
ax.xaxis.set_major_formatter(
    ticker.FormatStrFormatter('%5.1f'))
# Цена вспомогательных делений - 2
ax.xaxis.set_minor_locator(ticker.MultipleLocator(2))

# Работаем с осью Y
# Цена основных делений - 0.5
ax.yaxis.set_major_locator(ticker.MultipleLocator(0.5))
# Формат подписей: 2 знака после запятой
ax.yaxis.set_major_formatter(
    ticker.FormatStrFormatter('%4.2f'))
```

```
24 # Цена вспомогательных делений - 0.1
25 ax.yaxis.set_minor_locator(ticker.MultipleLocator(0.1))
26 plt.plot(X, Y)
```

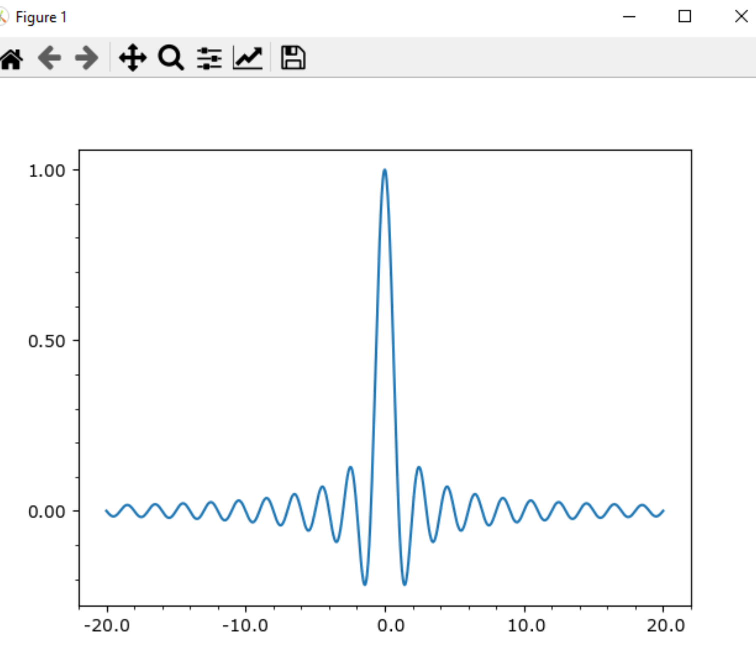


Рисунок 6.9 — Разметка осей вручную

Вопросы и задания.

1. Что из себя представляет функция `sinc`?
2. Как изменится разметка горизонтальной оси, если функцию `MultipleLocator` заменить функцией `LinearLocator`?

В качестве еще одного примера работы с осями рассмотрим более «ручной» случай, когда деления задаются списком координат — они тогда автоматически считаются основными. Задача состоит только в замене подписей под ними. В этом случае обращаться к методам области рисования не требуется. Англоязычная версия данного примера входит в комплект документации по пакету Matplotlib.

Пример 8. Пользовательские деления оси и подписи к ним.

```
import matplotlib.pyplot as plt
# Подключаем шрифт с кириллицей
from matplotlib import rc
font = {'family':'Verdana', 'weight':'normal'}
rc('font', **font)

X = [1, 2, 3, 4]
Y = [1, 4, 9, 6]
labels = ['Точки', 'Бочки', 'Почки', 'Кочки']
plt.plot(X, Y, 'ro')

# Задаем поворот меток с помощью ключевого слова
plt.xticks(X, labels, rotation = 'vertical')
# Увеличиваем поля, чтобы маркеры не обрезались осями
plt.margins(0.2)
# Добавляем места, чтобы не обрезались подписи
plt.subplots_adjust(bottom = 0.15)
plt.show()
```

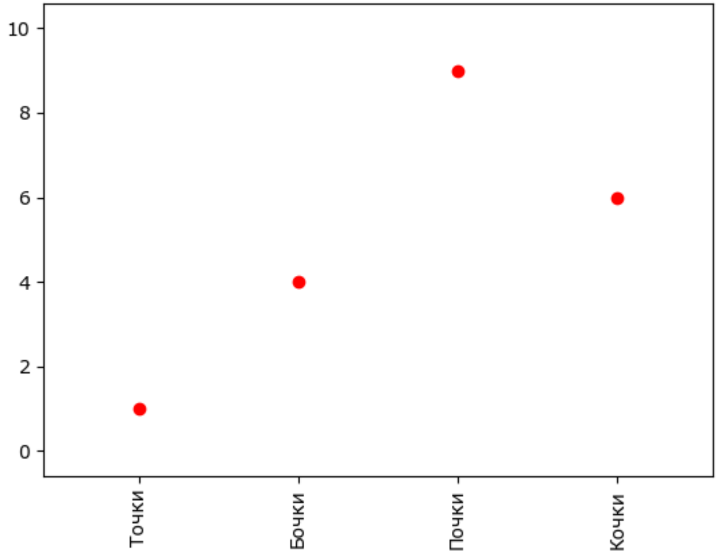


Рисунок 6.10 — Регулировка меток к осям

Вопросы и задания.

1. Вместо ключевого слова `vertical` при вызове функции `xticks` можно задавать значение угла поворота надписи в градусах. Попробуйте расположить метки под углом 45°, а потом -45°. Есть ли отличие между углами 90° и -90°?
2. Как изменится вид графика, если закомментировать строчку № 15?

Пример 9. Добавление сетки.

Следующий пример демонстрирует сразу три вещи:

- Еще один способ простого управления делениями на осях;
- График в виде набора отдельных точек (`scatter`);
- Добавление сетки (`grid`).

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  X = np.arange(0, 1, 0.05)
4  Y = np.power(X, 2)
5  ax = plt.axes()
6  ax.set_xticks(np.arange(0, 1, 0.1))
7  ax.set_yticks(np.arange(0, 1, 0.1))
8  plt.scatter(X, Y)
9  plt.grid()
10 plt.show()
```

Вопросы и задания.

1. Увеличьте в два раза размер сетки.
2. Замените функцию `scatter` функцией `plot` и добейтесь, чтобы график выглядел аналогично — синие кружки, не соединенные линией. Как избежать перекрывания кружков графика осями?

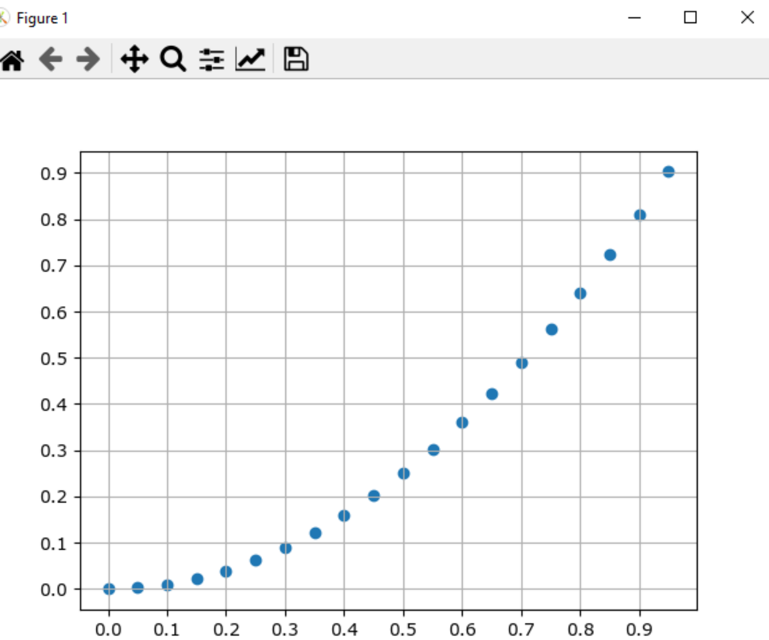


Рисунок 6.11 — Точечный график с сеткой

Другие разновидности графиков.

Пример 10. Столбиковые диаграммы.

```
1  import matplotlib.pyplot as plt
2  f1 = plt.figure()
3  # вертикальные столбики
4  plt.bar([1, 2, 3], [4, 3, 7])
5  f2 = plt.figure()
6  # горизонтальные столбики
7  plt.barh([1, 2, 3], [4, 3, 7])
```

Вопросы и задания.

1. За ширину колонок отвечает параметр `width` (для горизонтальной диаграммы — `height`), равный по умолчанию 0.8. Добавьте на фигуру 2 (Figure 2) столбик длины 8 и сделайте ширину всех столбиков на обеих фигурах равной 0.5. Покрасьте столбики на фигуре 2 в красный цвет.
2. Добавьте после четвертой строчки следующий текст:

```
plt.xticks([1,2,3], ('apples', 'bananas', 'oranges'),
       rotation = 90)
```

Что произошло? Как увидеть надписи полностью? Расположите надписи на оси по центру столбиков.

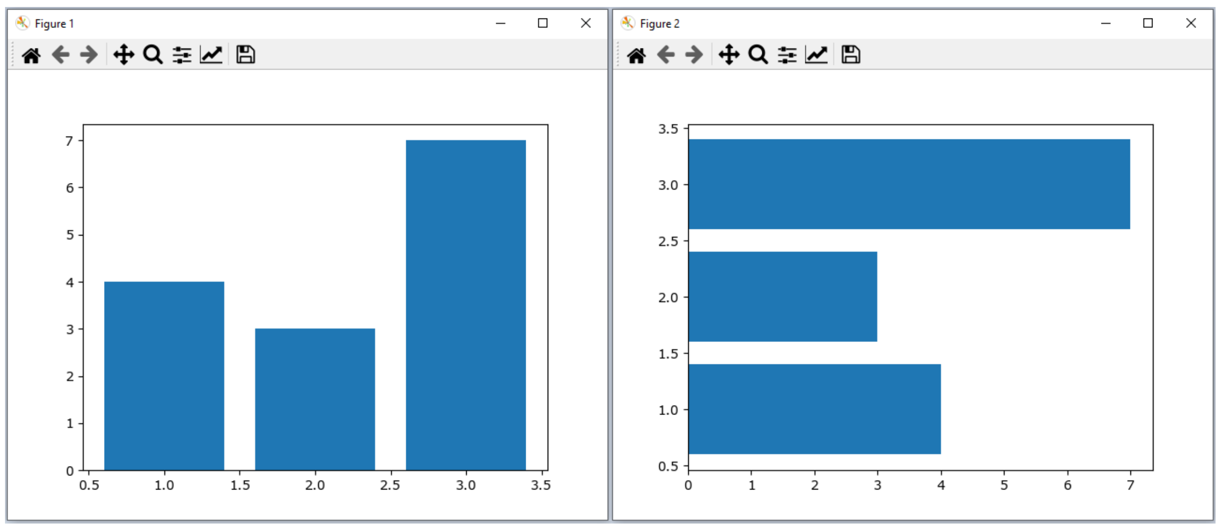


Рисунок 6.12 — Вертикальные и горизонтальные диаграммы

Пример 11. Гистограмма.

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  data = np.random.randn(1000) # случайные значения
4  plt.hist(data, bins = 20, color = 'g', histtype = "bar")
5  plt.grid()
6  plt.show()
```

Вопросы и задания.

1. Обратите внимание, что при каждом следующем запуске форма диаграммы меняется. Почему?
2. Кроме значения `'bar'` параметр `histtype` может принимать значения `'barstacked'`, `'step'` и `'stepfilled'`. Что они означают?

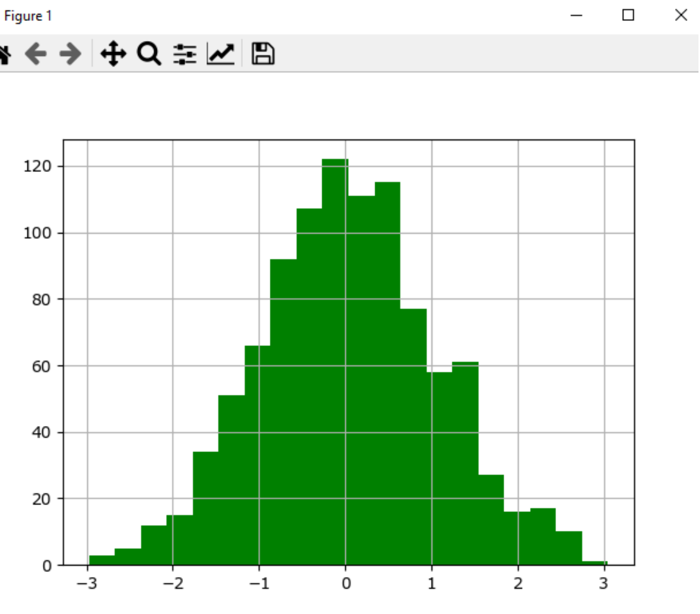
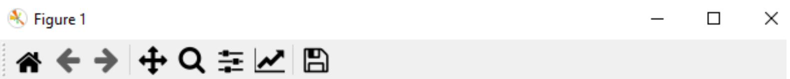


Рисунок 6.13 — Гистограмма

Пример 12. Круговая диаграмма.

```
1  import matplotlib.pyplot as plt
2  data = [500, 700, 300]
3  lbs = ['cats', 'dogs', 'other']
4  plt.pie(data, labels = lbs)
```



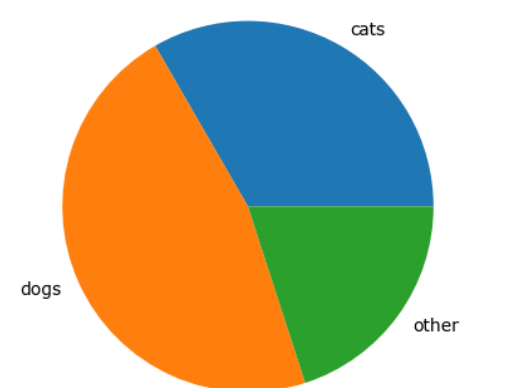


Рисунок 6.14 — Круговая диаграмма

Вопросы и задания.

Измените цвета на диаграмме: dogs — синий, cats — желтый, other — зеленый.

Пример 13. График в полярных координатах.

```
import numpy as np
import matplotlib.pyplot as plt
phi = np.linspace(0, 2*np.pi, 1000)
r = 2*phi
plt.polar(phi, r)
plt.show()
```

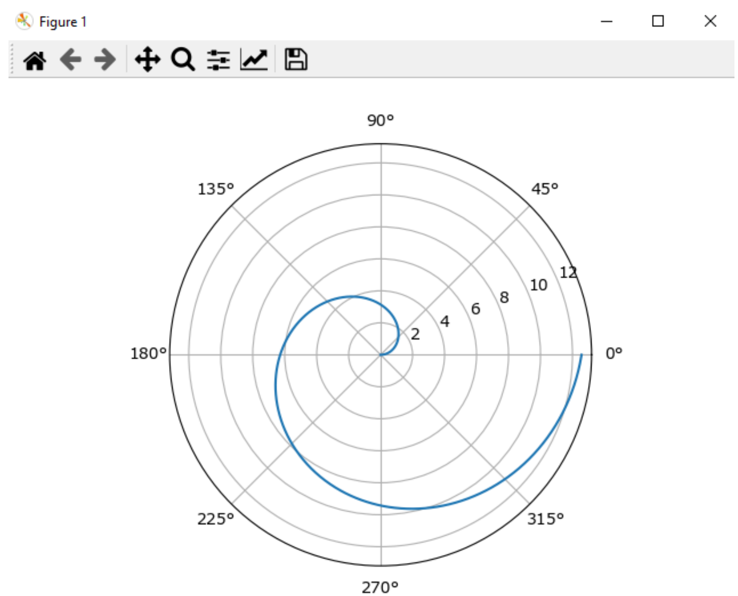


Рисунок 6.15 — Спираль в полярных координатах

Вопросы и задания.

Спираль на графике соответствует кривой $r = 2\varphi$. Постройте кривую, задаваемую уравнением $r = 1.1 - \cos\varphi$.

Пример 14. Графики функций двух переменных.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import axes3d

```

```
5  def f1(x, y):
6      return np.sin(x**2 + y**2)
7
8  def f2(x, y):
9      return x**2 - y**2
10
11 fig1 = plt.figure()
12 ax = plt.subplot(111, projection = '3d')
13 N = 100
14 a = np.linspace(-2, 2, N)
15 X, Y = np.meshgrid(a, a)
16 Z = f1(X, Y)
17 ax.plot_wireframe(X, Y, Z, linewidth = 0.1)
18
19 fig2 = plt.figure()
20 ax = plt.subplot(111, projection = '3d')
21 N = 200
22 a = np.linspace(-5, 5, N)
23 b = np.linspace(-2, 2, N)
24 X, Y = np.meshgrid(a, b)
25 Z = f2(X, Y)
26 ax.plot_surface(X, Y, Z, linewidth = 0.1)
```

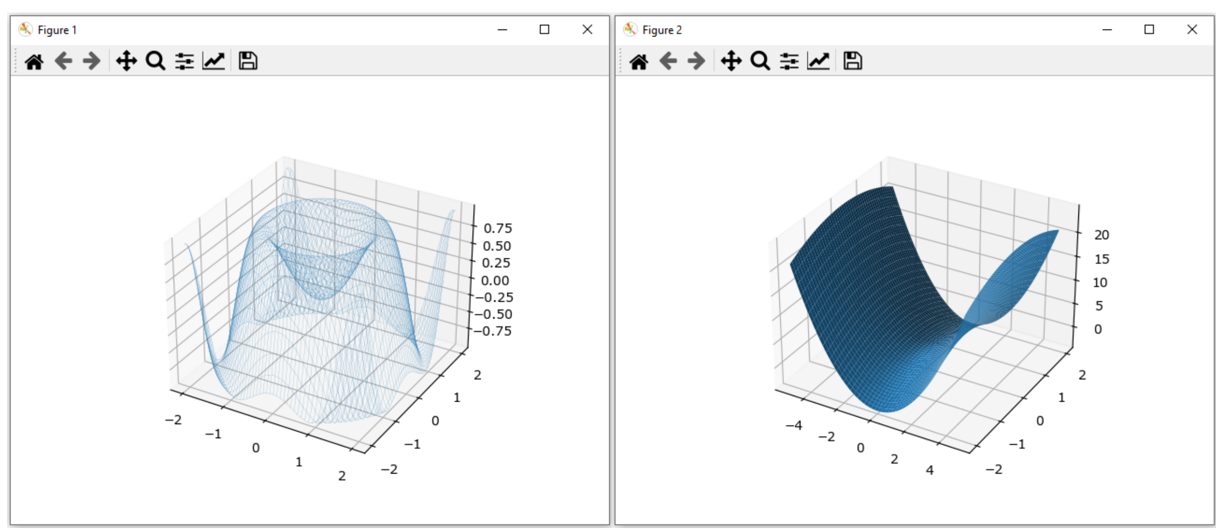


Рисунок 6.16 — Варианты изображения поверхностей

1. Проверьте, что графики являются «интерактивными» — их можно не только масштабировать, но и вращать.
2. Попробуйте «перекрасить» график на фигуре 2.

7 Проектное задание «Обработка и анализ данных в Pandas»

7.1 Знакомство с Pandas

Pandas — это высокоуровневая библиотека с открытым исходным кодом на Python. Разработка пакета начата в 2008 году Уэсом Маккинни, в 2012 году к проекту присоединился второй главный разработчик библиотеки Чан Шэ.

Модуль Pandas работает поверх NumPy и широко используется для обработки и анализа данных. Полезную информацию по библиотеке и разбор примеров можно найти в [3, 6, 12].

Основными структурами хранения данных в Pandas являются:

- Series;
- DataFrame.

Series.

`Series` — это объект библиотеки Pandas, спроектированный для представления одномерных структур данных, похожих на массивы и списки, но с дополнительными возможностями. Он состоит из двух связанных между собой массивов. Основной содержит данные (данные любого типа NumPy), а в дополнительном, `index`, хранятся метки. То есть этот объект напоминает по структуре столбец в Excel — там тоже есть название столбца, индексы ячеек и данные.

Создать `Series` можно, передав в конструктор в качестве аргумента просто список:

```
In [1]: import pandas as pd
        # индексами будут числа от 0 до 2
        S1 = pd.Series([5,6,7])
```

или сразу со списком индексов:

```
In [2]: S2 = pd.Series([5, 6, 7, 8],
                        index = ['a', 'b', 'c', 'd'])
```

Получить значение из объекта `Series` можно так:

```
In [3]: # как при работе с массивами в Numpy
        S1[0], S2['a'], S1[0:2], S2[S2>5]

Out[3]: (5,
         5,
         0    5
         1    6
         dtype: int64,
         b    6
         c    7
         d    8
         dtype: int64)
```

DataFrame.

`DataFrame` — самая важная и широко используемая в Pandas структура, а также стандартный способ хранения данных. Она содержит данные, выровненные по строкам и столбцам, как в таблице SQL или Excel. Каждый столбец `DataFrame` является структурой `Series`.

Пользователь может сортировать `DataFrame` по значениям строки или столбца, а также по индексу строки или столбца.

Создать `DataFrame` можно с помощью конструктора с параметрами:

`pd.DataFrame(data, index, columns, dtype, copy)`

где

- `data` — входные данные. Это может быть список, словарь, объект `Series`, массив NumPy или даже любой другой `DataFrame`;
- `index` — метки строк;
- `columns` — подписи столбцов;

- `dtype` — используется для указания типа данных каждого столбца, необязательный параметр;
- `copy` — копирование данных, необязательный параметр.

Пример создания `DataFrame`:

In [4]:
```
df = pd.DataFrame(
    {'Страна':
     ['Саудовская Аравия', 'Россия', 'Ирак',
      'Кувейт', 'Объединенные Арабские Эмираты'],
    '2016 год': [130, 90, 50, 49, 38],
    '2017 год': [150, 100, 54, 45, 65]},
    index = ['SA', 'RU', 'IQ', 'KW', 'AE'])
display(df)
```

	Страна	2016 год	2017 год
SA	Саудовская Аравия	130	150
RU	Россия	90	100
IQ	Ирак	50	54
KW	Кувейт	49	45
AE	Объединенные Арабские Эмираты	38	65

Мы создали словарь и передали его в качестве аргумента в метод `DataFrame`, указав в качестве меток буквенные коды соответствующих стран. Если смотреть на конструктор, то мы использовали только два параметра: `data` и `index`, остальные остались со значениями по умолчанию.

Кроме того, можно создать объект `DataFrame`, выгрузив данные из файлов Excel, CSV, текстовых файлов и т. д.

Файл CSV — это текстовый файл с одной записью данных в каждой строке, он может содержать только цифры и буквы и структурировать содержащиеся в нем данные в табличной форме. Значения в записи разделяются символом «запятая».

Осуществить загрузку данных из .txt или .csv в `DataFrame` можно, используя метод `read_csv`, из Excel-таблицы — методом `read_excel`, правда, для него ещё понадобится установка пакета `xlrd`.

Пример 1.

Пусть у нас есть файл «df1.txt», его содержимое в блокноте выглядит так:

```
country, area
Russia, 17125191
Canada, 9576140
India, 3287590
```

Вызываем метод `read_csv`, получаем:

```
In [5]: df1 = pd.read_csv("df1.txt")
        display(df1)
```

	country	area
0	Russia	17125191
1	Canada	9576140
2	India	3287590

Пример 2.

Пусть у нас есть файл «df2.xlsx», содержащий данные о сотрудниках:

	A	B	C
1	Сотрудник	Дата рождения	Зарплата
2	Васильев Василий Васильевич	22.02.2000	57000
3	Иванов Иван Иванович	03.03.2003	45000
4	Петров Петр Петрович	05.05.2005	15000
5	Ляликова Лилия Никитична	01.01.2001	21000

Устанавливаем нужный пакет и вызываем метод `read_excel`:

In [6]:
```
!pip install xlrd
df2 = pd.read_excel("df2.xlsx")
display(df2)
```

	Сотрудник	Дата рождения	Зарплата
0	Васильев Василий Васильевич	2000-02-22	57000
1	Иванов Иван Иванович	2003-03-03	45000
2	Петров Петр Петрович	2005-05-05	15000
3	Ляликова Лилия Никитична	2001-01-01	21000

7.2 Данные для анализа

Студенты разбиваются на малые группы по 3–4 человека для выполнения группового проекта. Каждая подгруппа выбирает себе тему исследования из предложенных или формулирует задачу самостоятельно. Возможные темы и наборы данных:

1. Данные о количестве и ценах проданных авокадо различных типов за период с 2015 по 2018 гг. в магазинах США [25].
2. Данные о Покемонах [19].
3. Данные об оценках на экзаменах студентов США [31].
4. Данные о случаях заболеваний, смертей от COVID-19 в различных странах за одну неделю [17].
5. Данные о погоде в Канаде в течение года [21].
6. Любой другой набор данных, обязательные требования:
    - Не должно быть много «пустых» ячеек;
    - Количество записей должно быть не меньше 100;
    - Среди данных должны быть числовые характеристики.

Следует обращать внимание на критерий «Удобство использования (Usability)», который говорит о том, что данные в наборе представлены в виде, удобном для анализа.

На выполнение задания группе студентов отводится 4 недели. По итогам работы требуется представить к защите получившийся проект в виде презентации.

7.3 Постановка задачи

Первая неделя.

Распределить роли и спланировать работу над проектом. Представить преподавателю план работ с подробным распределением по неделям.

В команде должен быть:

- Ответственный за тему, задачи исследования и организацию работы;
- Разработчик (человек, пишущий код);
- Ответственный за демонстрацию и оформление результатов.

Найти набор больших данных по выбранной тематике из открытых источников: с помощью базы Kaggle [24] или одной из аналогичных баз. Представить выбранный набор данных с подробным описанием на русском языке. Сформулировать задачи и планируемые результаты исследования, исходя из выбранных данных.

Вторая и третья неделя.

Трансформировать данные, приведя их к нужному виду с помощью библиотеки Pandas, визуализировать данные (или их небольшую часть) с использованием сводных таблиц и диаграмм. Построить минимум 5 диаграмм различных типов, отражающих представленные в наборе данные.

Четвертая неделя.

Оформить в виде презентации получившиеся результаты, в том числе отчет по результатам анализа работы команды над проектом.

Критерии оценивания проектного задания:

1. Выбран набор данных (*датасет*, dataset), сформулированы задачи исследования по нему.

2. Датасет импортирован в Jupyter Notebook и некоторые таблицы или их части напечатаны, есть попытки анализа этих таблиц.
3. Содержимое датасета визуализировано с помощью диаграмм.
4. Подготовлена и представлена презентация с полученными диаграммами и их анализом, а также отчет по командной работе.

7.4 Образец выполнения проектного задания

Рассмотрим популярный датасет о среднем росте мужчин и женщин в странах мира по данным на 2022 год [22]. Данные представлены в файле csv, что позволяет сразу на сайте увидеть их структуру и содержимое.

В описании к данным говорится о том, что в таблице представлены данные о росте мужчин и женщин из стран мира в различных единицах измерения (сантиметрах и футах). В нашей стране принята система СИ, поэтому столбцы с данными о росте в футах мы использовать не будем, для нас это «лишние» данные.

В таблице всего 6 столбцов: номер п/п, страна, средний рост мужчин в см, средний рост женщин в см, средний рост мужчин в футах, средний рост женщин в футах.

К описанию датасета есть также идеи для исследования, однако мы можем и сами сформулировать вопросы, ответы на которые можно получить, проанализировав представленный набор данных.

Вопросы:

1. Каков средний рост мужчин и женщин на планете в настоящее время?
2. В каких странах рост мужчин и женщин больше всего отличается? Представить выборку 10 стран с максимальной разницей.
3. В каких странах самые высокие мужчины и женщины? Представить выборку 10 стран для мужчин и женщин.
4. В каких странах самые низкие мужчины и женщины? Представить

выборку 10 стран для мужчин и женщин.

5. Как влияет близость к экватору на средний рост людей?

Импортируем все необходимые пакеты для работы с данными. Файл csv скачаем и поместим в ту же папку, что и ipynb-файл.

```
1 import numpy as np
2 import pandas as pd
3 import matplotlib as mpl
4 import matplotlib.pyplot as plt
```

Создадим объект `DataFrames` с помощью метода `read_csv`. Этот объект из первой строки csv-файла «берет» названия столбцов таблицы. Строки по умолчанию пронумерованы индексами, начинающимися с 0 (самый левый столбец). Откроем наш файл и покажем заголовочную часть таблицы и первые пять строк (метод `head`).

```
1 data=pd.read_csv('HeightofMaleandFemalebyCountry2022.csv')
2 # копируем исходный объект, дальше будем его менять
3 data3 = data.copy()
4 data.head()
```

	Rank	Country Name	Male Height in Cm	Female Height in Cm	Male Height in Ft	Female Height in Ft
0	1	Netherlands	183.78	170.36	6.03	5.59
1	2	Montenegro	183.30	169.96	6.01	5.58
2	3	Estonia	182.79	168.66	6.00	5.53
3	4	Bosnia and Herzegovina	182.47	167.47	5.99	5.49
4	5	Iceland	182.10	168.91	5.97	5.54

Вопрос 1. Каков средний рост мужчин и женщин на планете в настоящее время?

Для того, чтобы хранящие ненужные данные столбцы таблицы не отображались (но остались в ней), используем выборку по именам столбцов:

```
In [3]: data[['Country Name','Male Height in Cm',
              'Female Height in Cm']]
```

	Country Name	Male Height in Cm	Female Height in Cm
0	Netherlands	183.78	170.36
1	Montenegro	183.30	169.96
2	Estonia	182.79	168.66
3	Bosnia and Herzegovina	182.47	167.47
4	Iceland	182.10	168.91
...	...	...	...
194	Mozambique	164.30	155.42
195	Papua New Guinea	163.10	156.89
196	Solomon Islands	163.07	156.79
197	Laos	162.78	153.10
198	Timor-Leste	160.13	152.71

199 rows × 3 columns

Если хотим удалить столбцы, содержащие данные о росте в футах (нам они не нужны), используем `drop` с параметром `axis = 1`, означающим удаление столбцов, и `inplace = True`, позволяющий изменить существующий объект «на месте». Покажем для разнообразия последние пять строк таблицы (метод `tail`):

In [4]:
```
data.drop(
    ['Male Height in Ft', 'Female Height in Ft'],
    axis = 1, inplace = True)
data.tail()
```

	Rank	Country Name	Male Height in Cm	Female Height in Cm
194	195	Mozambique	164.30	155.42
195	196	Papua New Guinea	163.10	156.89
196	197	Solomon Islands	163.07	156.79
197	198	Laos	162.78	153.10
198	199	Timor-Leste	160.13	152.71

Для отображения пяти случайных строк можно воспользоваться методом `sample`:

```
In [5]: data.sample(5)
```

	Rank	Country Name	Male Height in Cm	Female Height in Cm
194	195	Mozambique	164.30	155.42
134	135	Gabon	170.48	160.05
152	153	Vietnam	168.89	158.43
2	3	Estonia	182.79	168.66
71	72	Mali	175.02	161.99

Для того, чтобы отфильтровать все страны, в которых средний мужской рост находится в пределах от 170 до 175 см, можно воспользоваться одним из двух вариантов:

```
In [6]: # Вариант 1.
        # data.loc[170:175,'Male Height in Cm']
        # Вариант 2.
        data_max = data[(data['Male Height in Cm'] > 170)
                        & (data['Male Height in Cm'] < 175)]
        display(data_max[['Country Name',
                          'Male Height in Cm']])
```

	Country Name	Male Height in Cm
72	Kuwait	174.96
73	Jordan	174.84
74	Hong Kong	174.83
75	Argentina	174.76
76	North Korea	174.69
...	...	...
138	Mexico	170.29
139	Niger	170.26
140	Panama	170.19
141	Togo	170.14
142	Kiribati	170.09

71 rows × 2 columns

Для подсчета количества строк можно использовать `count`:

In [7]:
```
data_max[['Country Name']].count()
```

Out[7]:
```
Country Name    71
dtype: int64
```

Вычислим средний рост мужчин и женщин на планете:

In [8]:
```
print(data['Male Height in Cm'].mean())
print(data['Female Height in Cm'].mean())
```

```
173.08904522613054
160.9429145728643
```

Ниже представлено использование метода `describe`:

In [9]:
```
data.drop(['Rank'], axis = 1, inplace = True)
data.describe()
```

	Male Height in Cm	Female Height in Cm
count	199.000000	199.000000
mean	173.089045	160.942915
std	4.949832	4.076377
min	160.130000	150.910000
25%	169.490000	158.240000
50%	173.530000	160.620000
75%	176.510000	163.870000
max	183.780000	170.360000

Вопрос 3. В каких странах самые высокие женщины?
Для ответа на этот вопрос воспользуемся сортировкой по указанному столбцу методом `sort_values` с параметром `ascending = False`, что означает «сортировка по убыванию».

```
In [10]: top10_f = data.sort_values(
             'Female Height in Cm',
             ascending = False).head(10)
         display(top10_f[['Country Name',
                              'Female Height in Cm']])
```

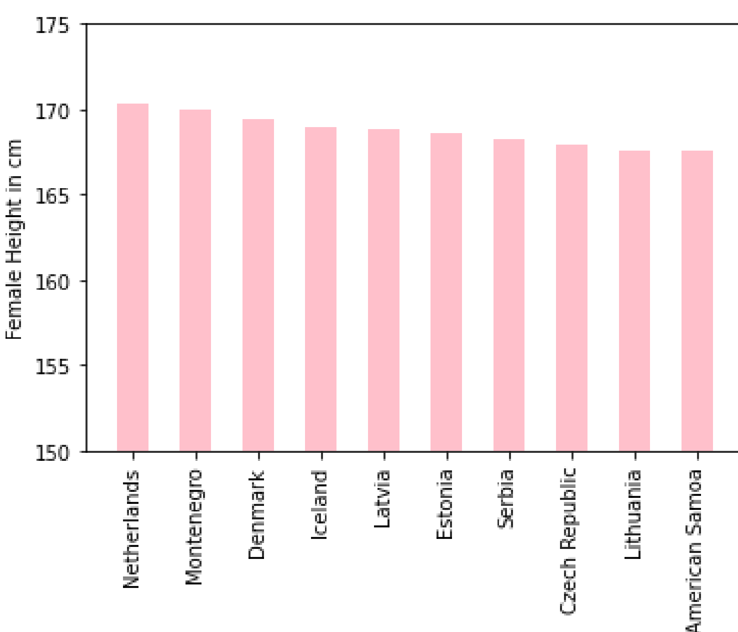
	Country Name	Female Height in Cm
0	Netherlands	170.36
1	Montenegro	169.96
5	Denmark	169.47
4	Iceland	168.91
7	Latvia	168.81
2	Estonia	168.66
12	Serbia	168.29
6	Czech Republic	167.96
13	Lithuania	167.63
43	American Samoa	167.55

Отразим полученную выборку на вертикальной гистограмме:

```
In [11]: plt.bar(data = top10_f, x = 'Country Name',
                 height = 'Female Height in Cm', width = 0.5,
                 color = 'pink')
         plt.xticks(rotation=90)
         plt.ylim(ymin=150,ymax=175)
         plt.ylabel('Female Height in cm')
         plt.xlabel('Top 10 tallest countries, female')
         plt.show()
```

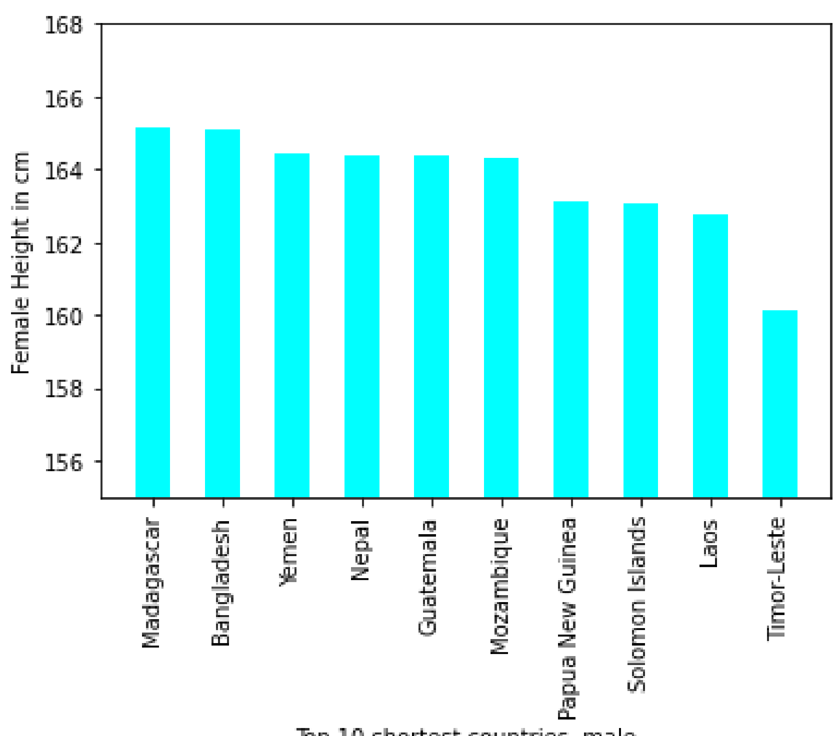
Вопрос 4. В каких странах самые низкие мужчины?

Аналогично предыдущему, сортируем объект по нужному столбцу и представляем полученные результаты в форме диаграммы и таблицы.



In [12]:
```
t10_sm = data.sort_values('Male Height in Cm',
                          ascending = False).tail(10)
display(t10_sm[['Country Name',
                'Male Height in Cm']])
plt.bar(data = t10_sm, x = 'Country Name',
        height = 'Male Height in Cm', width = 0.5,
        color = 'cyan')
plt.xticks(rotation = 90)
plt.ylim(ymin = 155,ymax = 168)
plt.ylabel('Female Height in cm')
plt.xlabel('Top 10 shortest countries, male')
plt.show()
```

	Country Name	Male Height in Cm
189	Madagascar	165.16
190	Bangladesh	165.08
191	Yemen	164.42
192	Nepal	164.36
193	Guatemala	164.36
194	Mozambique	164.30
195	Papua New Guinea	163.10
196	Solomon Islands	163.07
197	Laos	162.78
198	Timor-Leste	160.13



Вопрос 2. В каких странах рост мужчин и женщин больше всего отличается?

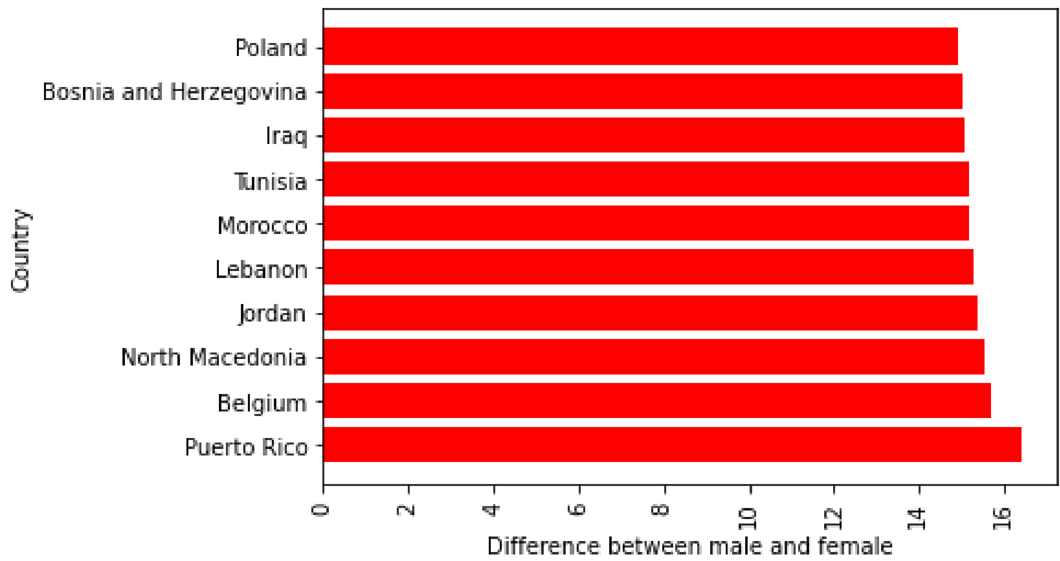
Сначала создадим новый объект `Series`, содержащий разницу в росте мужчин и женщин. Затем создадим новый объект `DataFrame`, содержащий названия стран и значение предыдущего ряда и отсортируем его по убыванию соответствующего столбца:

```
In [13]: d_h = data['Male Height in Cm'] - data[
             'Female Height in Cm']
         data2 = {'Country':data['Country Name'],
                  'Difference between male and female':d_h}
         df_d = pd.DataFrame(data2)
         df_ds = df_d.sort_values(
             by = "Difference between male and female",
             ascending = False).head(10)
         display(df_ds)
```

Построим также горизонтальную гистограмму, иллюстрирующую последний пример.

	Country	Difference between male and female
21	Puerto Rico	16.42
23	Belgium	15.69
51	North Macedonia	15.55
73	Jordan	15.38
25	Lebanon	15.29
54	Morocco	15.17
47	Tunisia	15.16
95	Iraq	15.04
3	Bosnia and Herzegovina	15.00
14	Poland	14.91

In [14]:
```
plt.barh(df_ds['Country'],
         df_ds['Difference between male and female'],
         align = 'center', color = 'red')
plt.xticks(rotation = 90)
plt.ylim()
plt.ylabel('Country')
plt.xlabel('Difference between male and female')
plt.show()
```



Вопрос 5. Как влияет близость к экватору на средний рост людей? Чтобы ответить на этот вопрос, необходимо показать распределение ро-

ста на карте мира. Для начала установим специальные пакеты, необходимые для работы со странами и картами:

In [15]:
```
!pip install pycountry_convert
!pip install plotly_express
import pycountry_convert as pc
import plotly_express as px
```

В копии данных, созданной в самом начале, необходимо заменить название страны Конго на классическое (без этого карта работать не будет):

In [16]:
```
data3['Country Name'].replace(
    'DR Congo', 'Democratic Republic of the Congo',
    inplace = True)
```

Затем применить «приведение» названия страны к формату `alpha_3` — название после метода `pc.country_name_to_country_alpha3` выглядит как уникальная аббревиатура из 3-х букв — и сформировать новый объект Series для построения диаграммы в виде карты:

In [17]:
```
data3['alpha3'] = data3['Country Name'].apply(
    lambda x: pc.country_name_to_country_alpha3(x))
```

Теперь используем специальный метод `choropleth` из `plotly_express`, о котором можно почитать в [20]:

In [18]:
```
fig = px.choropleth(
    data3, locations = 'alpha3',
    color = 'Male Height in Cm', scope = 'world',
    title = 'World Map for Height of Male')
fig.show()
```

По полученной диаграмме (рис. 7.1) можно судить о том, что в африканских странах и странах Латинской Америки, близких к экватору, проживают мужчины с самым низким средним ростом, и наоборот, чем дальше от экватора, тем выше средний рост мужского населения планеты.

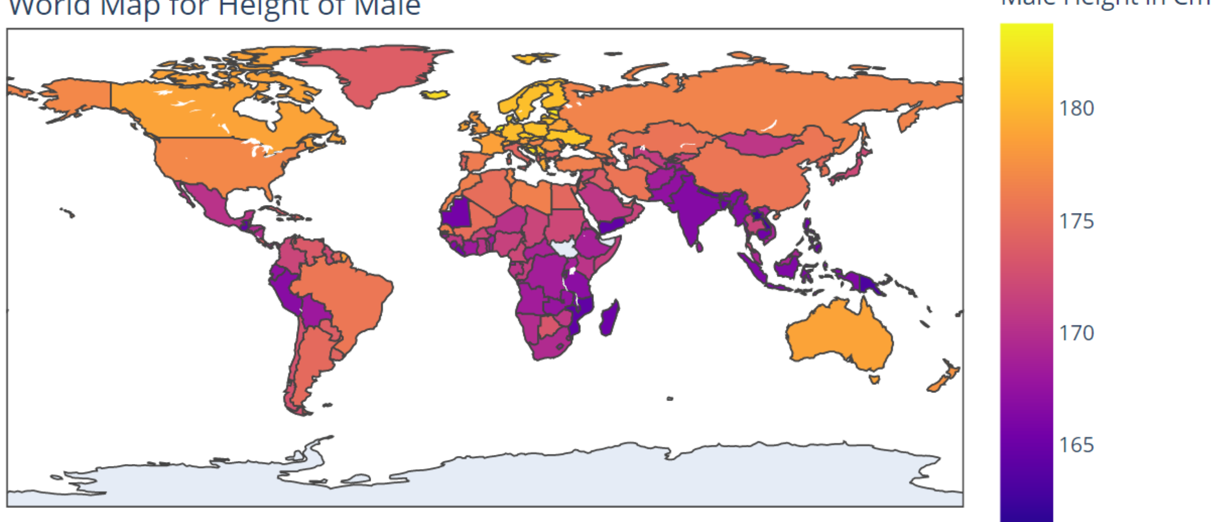


Рисунок 7.1 — Средний рост мужчин в мире

Аналогичную картину получим, проанализировав распределение по среднему росту среди женщин.

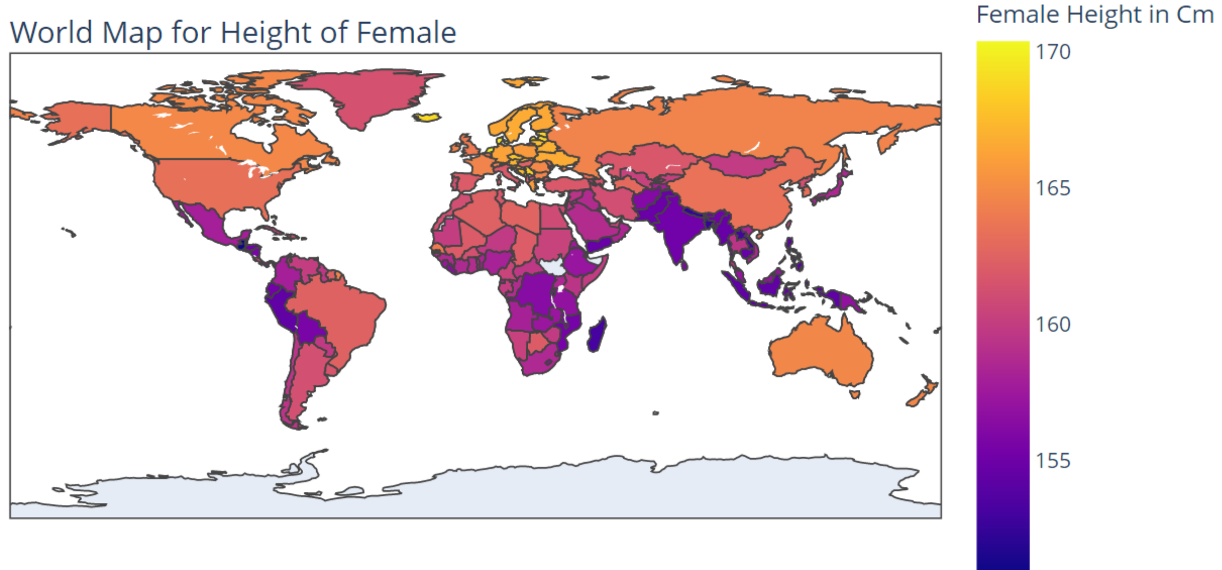


Рисунок 7.2 — Средний рост женщин в мире

Использование пакетов, позволяющих работать с картами, — это, конечно, экзотика. Но возможно, вам пригодятся эти пакеты для исследования ваших наборов данных.

Ищите, исследуйте, анализируйте!

Список литературы

1. Абдарахманов М. Matplotlib. Урок 1. Быстрый старт [Электронный ресурс]. 2019. URL: https://devpractice.ru/matplotlib-lesson-1-quick-start-guide/. (дата обращения: 13.04.2022).

2. Абрамян М. Э. Programming Taskbook: Электронный задачник по программированию [Электронный ресурс]. 2022. URL: http://ptaskbook.com/ru/. (дата обращения: 13.04.2022).

3. Васильев А.Н. Библиотека Pandas в Python [Электронный ресурс]. 2021. URL: https://pythonim.ru/libraries/biblioteka-pandas-python. (дата обращения: 13.04.2022).

4. Викиучебник. Математические формулы в LaTeX [Электронный ресурс]. 2021. URL: ru.wikibooks.org/wiki/Математические_формулы_в_LaTeX. (дата обращения: 13.04.2022).

5. Долгих Т. Ф., Ширяева Е. В. Язык Python 3 для научных исследований. Ростов н/Д., 2017. 90 с.

6. Егоров Е. Pandas — обработка и анализ данных в 2021 году [Электронный ресурс]. 2020. URL: https://egorovegor.ru/pandas-obrabotka-i-analiz-dannyh-v-python/. (дата обращения: 13.04.2022).

7. Иваничев И. Исчерпывающая шпаргалка по синтаксису разметки Markdown [Электронный ресурс]. 2020. URL: https://texterra.ru/blog/ischerpyvayushchaya-shpargalka-po-sintaksisu-razmetki-markdown-na-zametku-avtoram-veb-razrabotchikam.html. (дата обращения: 13.04.2022).

8. Лутц М. Изучаем Python. 5-е изд.: Пер. с англ. СПб.: ООО «Диалектика», 2019. Т. 1. 832 с.

9. Львовский С. Набор и верстка в системе LATEX. М.: МЦНМО, 2014. 400 с.

10. ПитонТьютор: Интерактивный учебник языка Python [Электронный ресурс]. 2012. URL: https://http://pythontutor.ru. (дата обращения: 13.04.2022).

11. Строки в Python 3: методы, функции, форматирование [Электронный ресурс]. 2015. URL: https://pythonru.com/osnovy/stroki-python. (дата обращения: 13.04.2022).

12. Структуры данных в Pandas / pd 2 [Электронный ресурс]. 2021. URL: https://pythonru.com/biblioteki/struktury-dannyh-v-pandas. (дата обращения: 13.04.2022).

13. Физтех.Статистика. Python для анализа данных. Библиотека Matplotlib [Электронный ресурс]. 2021. URL: https://mipt-stats.gitlab.io/courses/python/06_matplotlib.html. (дата обращения: 13.04.2022).

14. Хилл К. Научное программирование на Python. Пер. с англ. А. В. Снастина. М.: ДМК Пресс, 2021. 646 с.

15. Шелудько В. М. Основы программирования на языке высокого уровня Python: учебное пособие. Ростов н/Д., Таганрог: Изд-во ЮФУ, 2017. 146 с.

16. Шелудько В. М. Язык программирования высокого уровня Python. Функции, структуры данных, дополнительные модули: учебное пособие. Ростов н/Д., Таганрог: Изд-во ЮФУ, 2017. 107 с.

17. Anandhu H. Dataset. COVID-19 Weekly Trends In World — Latest Data. Weekly trends of Covid-19 as on March 17, 2022 [Electronic resource]. 2022. URL: https://www.kaggle.com/anandhuh/covid19-weekly-trends-in-world-latest-data. (accessed: 2022-04-13).

18. Array programming with NumPy / C. R. Harris, K. J. Millman, S. J. van der Walt et al. // Nature. 2020. Sep. Vol. 585, no. 7825. P. 357–362.

19. Barradas A. Dataset. Pokemon with stats. 721 Pokemon with stats and types [Electronic resource]. 2016. URL: https://www.kaggle.com/abcsds/pokemon. (accessed: 2022-04-13).

20. Choropleth Maps in Python [Electronic resource]. 2022. URL: https://plotly.com/python/choropleth-maps/. (accessed: 2022-04-13).

21. Dataset. Canada Weather. Data about weather in winter, summers and throughout the year [Electronic resource]. 2022. URL: https://www.kaggle.com/hemil26/canada-weather. (accessed: 2022-04-13).

22. Dataset. Height of Male and Female by Country 2022. Average Height of Male and Female by Country [Electronic resource]. 2022. URL: https://www.kaggle.com/majyhain/height-of-male-and-female-by-country-2022. (accessed: 2022-04-13).

23. Jupyter Notebook для начинающих: учебник [Электронный ресурс]. 2019. URL: https://webdevblog.ru/jupyter-notebook-dlya-nachinajushhih-uchebnik/. (дата обращения: 13.04.2022).

24. Kaggle: Your Machine Learning and Data Science Community [Electronic resource]. 2022. URL: https://www.kaggle.com/. (accessed: 2022-04-13).

25. Kiggins J. Dataset. Avocado Prices. Historical data on avocado prices and sales volume in multiple US markets [Electronic resource]. 2018. URL: https://www.kaggle.com/neuromusic/avocado-prices. (accessed: 2022-04-13).

26. Matplotlib: Visualization with Python [Electronic resource]. 2021. URL: https://matplotlib.org/. (accessed: 2022-04-13).

27. NumPy: The fundamental package for scientific computing with Python [Электронный ресурс]. 2020. URL: https://numpy.org/. (дата обращения: 13.04.2022).

28. PEP 8 – руководство по написанию кода на Python [Электронный ресурс]. 2014. URL: https://pythonworld.ru/osnovy/pep-8-rukovodstvo-po-napisaniyu-koda-na-python.html. (дата обращения: 13.04.2022).

29. Pritchard A. Шпаргалка по Markdown (переведено с английского) [Электронный ресурс]. 2018. URL: https://bustep.ru/markdown/shpargalka-po-markdown.html. (дата обращения: 13.04.2022).

30. Schmit S. CME 193: Introduction to Scientific Python. Lecture 5: Numpy, Scipy, Matplotlib [Electronic resource]. 2015. URL: http://stanford.edu/~schmit/cme193/index.html. (accessed: 2022-04-13).

31. Seshapanpu J. Dataset. Students Performance in Exams. Marks secured by the students in various subjects [Electronic resource]. 2019. URL: https://www.kaggle.com/spscientist/students-performance-in-exams. (accessed: 2022-04-13).

32. Solomon B. Построение графиков в Python при помощи Matplotlib (переведено с английского) [Электронный ресурс]. 2021. URL: https://python-scripts.com/matplotlib. (дата обращения: 13.04.2022).

33. TIOBE Index April 2022 [Electronic resource]. URL: https://www.tiobe.com/tiobe-index/. (accessed: 2022-04-13).

34. Top Programming Languages 2021 [Electronic resource]. URL: https://spectrum.ieee.org/top-programming-languages/. (accessed: 2022-04-13).

35. van Rossum G., Warsaw B., Coghlan N. PEP8 – Style Guide for Python Code [Электронный ресурс]. 2013. URL: https://peps.python.org/pep-0008/. (дата обращения: 13.04.2022).

Учебное издание

Карякин Михаил Игорьевич,
Ватульян Карина Александровна,
Мнухин Роман Михайлович

Технологии программирования и компьютерный практикум на языке Python

Подписано в печать 19.07.2022 г.
Бумага офсетная. Формат 60×84 1/16. Тираж 30 экз.
Усл. печ. лист. 14,07. Уч.-изд. л. 7,0. Заказ № 8578.

Издательство Южного федерального университета.

Отпечатано в отделе полиграфической, корпоративной и сувенирной продукции
Издательско-полиграфического комплекса КИБИ МЕДИА ЦЕНТРА ЮФУ.
344090, г. Ростов-на-Дону, пр. Стачки, 200/1, тел (863) 243-41-66.

ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
ЮФУ
РОСТОВ-НА-ДОНУ·ТАГАНРОГ

9 785927 541089