МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Языки представления знаний

Методические указания к лабораторным работам

для направления подготовки:

09.04.02 «Информационные системы и технологии»

Ставрополь

ОГЛАВЛЕНИЕ

Введение

Целью освоения дисциплины «Языки представления знаний» является формирование профессиональной компетенции (ПК-2) будущего магистра по направлению подготовки 09.04.02 «Информационные системы и технологии».

Задачи освоения дисциплины:

– формирование знаний методов и средств разработки и исследования теоретических и экспериментальных моделей объектов профессиональной деятельности в различных областях;

– формирование умения разрабатывать и исследовать теоретические и экспериментальные модели объектов профессиональной деятельности в различных областях;

– формирование умения разрабатывать план испытаний информационной системы и / или программного продукта;

– формирование навыков проведения испытаний информационных систем и программных продуктов.

Методические указания к выполнению лабораторных работ по дисциплине «Языки представления знаний» содержат 8 лабораторных работ.

Перечень планируемых результатов обучения по дисциплине (модулю), соотнесённых с планируемыми результатами освоения образовательной программы

Код, формулировка компетенции	Код, формулировка индикатора	Планируемые результаты обучения по дисциплине (модулю), характеризующие этапы формирования компетенций, индикаторов
ПК-2 Способен разрабатывать и исследовать теоретические и экспериментальные модели объектов профессиональной деятельности в различных областях	ПК-2$_{ИД-2.1}$ Демонстрирует знание методов и средств разработки исследования теоретических и экспериментальных моделей объектов профессиональной деятельности в различных областях	Сформированы знания методов и средств разработки и исследования теоретических и экспериментальных моделей объектов профессиональной деятельности в различных областях. Сформированы умения разрабатывать и исследовать теоретические и экспериментальные модели объектов профессиональной деятельности в различных областях. Сформированы умения разрабатывать план испытаний информационной системы и / или программного продукта.
	ПК-2$_{ИД-2.2}$ Разрабатывает и исследует теоретические и экспериментальные модели объектов профессиональной деятельности в различных областях	
	ПК-2$_{ИД-2.3}$ Разрабатывает план испытаний информационной системы и /или программного продукта	
	ПК-2$_{ИД-2.4}$ Проводит испытания информационных систем и программных продуктов	

Лабораторная работа 1. Линейные алгоритмы. Операции с числами и строками. Ветвления и оператор выбора

Цель: При разборе задач в этой части будем обращать внимание на постановку задачи (что именно нужно сделать) и собственно алгоритм, который будет описываться как блок-схемой, так и на "псевдоязыке" программирования (подобие "школьного алгоритмического языка"). И только после этого можно приступать к написанию программы на Python с учётом всех тех его особенностей и возможностей, которые были описаны в предыдущей части.

Требования к отчету: Выполненные задания с результатами вывода оформить в виде отчета со скриншотами и прикрепить здесь. На каждом скриншоте в коде решения задачи прописать ФИО автора решения задачи и группу, в которой учится слушатель.

Линейные алгоритмы. Операции с числами и строками

Линейный алгоритм – алгоритм, в котором вычисления выполняются строго последовательно. Типичная блок-схема линейного алгоритма показана на рис. 2.1.

Далее рассмотрим типичные задачи с линейной структурой алгоритма.

Задача 1. Дано два числа a и b. Сделать так, чтобы их значения поменялись местами.

Постановка задачи: Имеются две переменные с какими-то определёнными значениями. Пусть значение a равно x, а значение b равно y. Требуется, чтобы значение a стало равно y, а значение b стало равно x.

Метод решения (общий): Использовать дополнительную переменную c, в которую временно записать начальное значение переменной a, присвоить переменной a значение переменной b, а потом переменной b присвоить значение переменной c.

Блок-схема такого алгоритма показана на рис. 2.2.

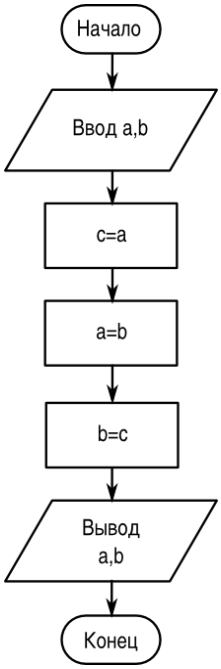


Рис. 2.2. Блок-схема алгоритма обмена значениями

Текст программы на "псевдоязыке":

```
ввод
     a, b
c=a
a=b
b=c
вывод
a, b
```

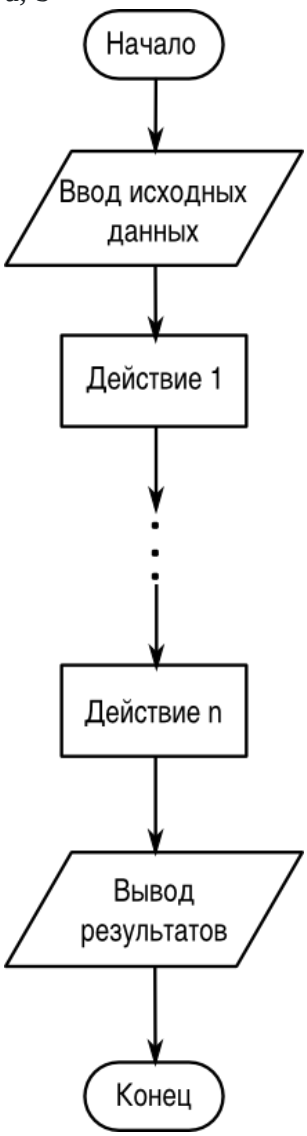


Рис. 2.1. Типичная схема линейного алгоритма

Метод решения с использованием особенностей Python: использовать два кортежа. В первом будут определены переменные a и b и их значения, а второй сформируем из этих же переменных, но в обратном порядке.

Текст программы на Python:

```
# -*- coding: utf-8 -*-
```

```
# Перестановка местами двух чисел с использованием кортежа

#

(a, b)=input('Введите исходные значения (a, b) через запятую: ')

(a, b)=(b, a)

print 'Новое значение a: ', a, '\n', 'Новое значение b: ', b
```

Как описано в разделе 1.4.2, комбинация '\n' означает директиву на перевод строки для команды **print**.

Задача 2. Известны оклад (зарплата) и ставка процента подоходного налога. Определить размер подоходного налога и сумму, получаемую на руки.

Постановка задачи: Исходными данными являются величина оклада (переменная oklad, выражаемая числом) и ставка подоходного налога (переменная procent, выражаемая числом). Размер налога (переменная nalog) определяется как oklad*procent/100, а сумма, получаемая на руки (переменная summa) – как oklad-nalog.

Блок-схема алгоритма показана на рис. 2.3.

Текст программы на "псевдоязыке":

```
ввод oklad, procent
nalog=oklad * procent /100
summa=oklad-nalog
вывод summa, nalog
```

Программа на Python:

```
# -*- coding: utf-8 -*-

#

oklad=input("Оклад: ")

procent=input("% налога: ")

nalog=float(oklad*procent) /100

summa=oklad-nalog

print"Сумма на руки: ", summa

print"Налог: ", nalog
```

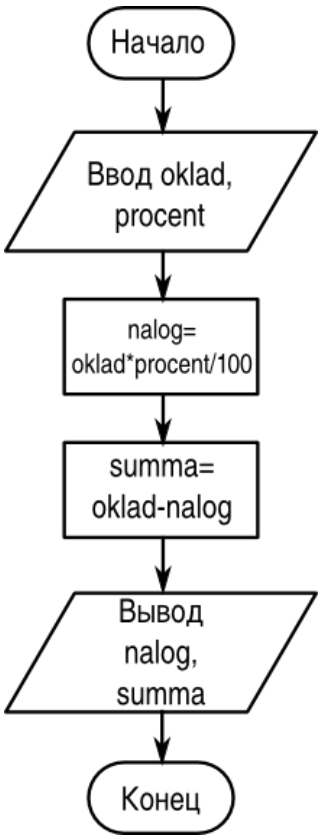


Рис. 2.3. Блок-схема задачи о налоге

Если все числа в этом примере использовать как целые, то результат может получиться неверным. Поэтому при вычислении налога используется преобразование числителя из целого числа в вещественное (функция float()).

Задача 3. Используя данные таблицы определить общую стоимость обеда в столовой. Определить, во сколько раз возрастёт стоимость обеда, если цена котлеты увеличится вдвое.[1]

Блюдо	Цена
Борщ	35
Котлета	40
Каша	20
Чай	3

Постановка задачи (формализованная): Имеется четыре числа, которые требуется просуммировать (обозначим их переменными a, b, c и d соответственно). Сумму их значений обозначим S1. Требуется найти также величину S2=S1+b и определить отношение S2/S1 (обозначим это отношение переменной res). В результате нужно вывести значения переменных S1 и res.

Блок-схема показана на рис. 2.4.

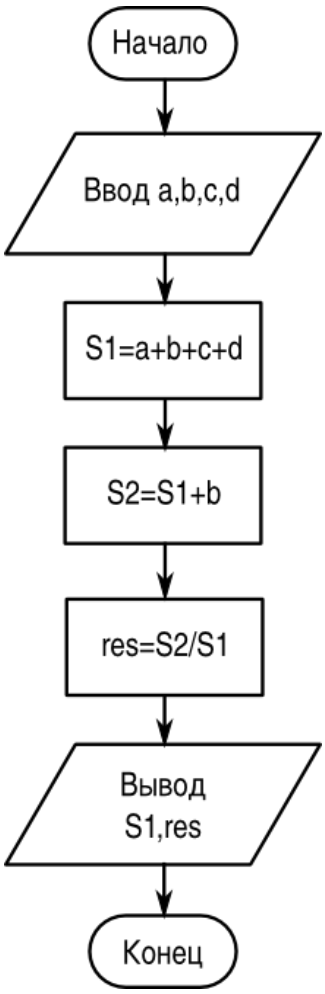


Рис. 2.4. Блок-схема задачи об обеде

Текст программы на "псевдоязыке":

ввод a, b, c, d
S1=a, b, c, d
S2=S1+b
res=S2/S1
вывод S1, res

В программе на Python разумно будет использовать кортеж:

```
# -*- coding: utf-8 -*-

#

t=(a, b, c, d)=input('Введите значения через запятую: ')

S1=sum(t)

S2=S1+b

res=float(S2) /S1
```

print'Начальная_стоимость : ', S1, ' \n ', 'Увеличение, _раз : ', res

И снова для преобразования целого числа в вещественное использована функция float(). (Полезно сравнить результат, получаемый при использвании выражения res=float(S2)/S1 и выражения res=float(S2/S1)).

Задача 4. Преобразовать дату в "компьютерном" представлении (системную дату) в "российский" формат, т. е. день/месяц/год (например, 17/05/2009).

Постановка задачи: Системная дата имеет вид 2009-06-15. Нужно преобразовать это значение в строку, строку разделить на компоненты (символразделитель – дефис), потом из этих компонентов сконструировать нужную строку.

Сразу перейдём к программе на Python. Функциями работы с датами и временем в Python "заведует" модуль datetime, а непосредственно для работы с датами используется объект date и его методы.

Воспользуемся знанием методов строк и списков.

```
# -*- coding: utf-8 -*-

#

# Подключаем нужный программный модуль

from datetime import date

# Получаем текущую дату

d1=date.today()

# Преобразуем результат в строку

ds=str(d1)

print"Системная дата ", ds

# Используем методы строки и списка

lst=ds.split('-')

lst.reverse()

# Составляем новую строку для даты

rusdate="/".join(lst)

print"Российский стандарт ", rusdate
```

Комментарии в тексте программы помогают понять происходящее.

2.1.1 Задачи для самостоятельного решения

1. Нарисуйте блок-схему к задаче 4 этой лабораторной работы.
2. Даны действительные числа A,B,C. Найти максимальное и минимальное из этих чисел.
3. Известны длины трёх сторон треугольника. Вычислить периметр треугольника и площадь по формуле Герона (указание: использовать модуль math и функцию sqrt()).
4. Задан вес в граммах. Определить вес в тоннах и килограммах.
5. Известен объем информации в байтах. Перевести в килобайты, мегабайты.
6. Определить значение функции Z=1/(XY) при X и Y не равных 0.

Ветвления и оператор выбора

В решениях задач по алгоритмизации одним из важнейших элементов является так называемое "ветвление", которое хорошо описывается сказочной формулой "Направо пойдёшь – голову потеряешь, прямо пойдёшь – коня потеряешь...", а проще говоря, ситуация "если ..., то ..., иначе ...". Типовая блок-схема алгоритма с ветвлением (проверкой условия) показана на рис. 2.5.

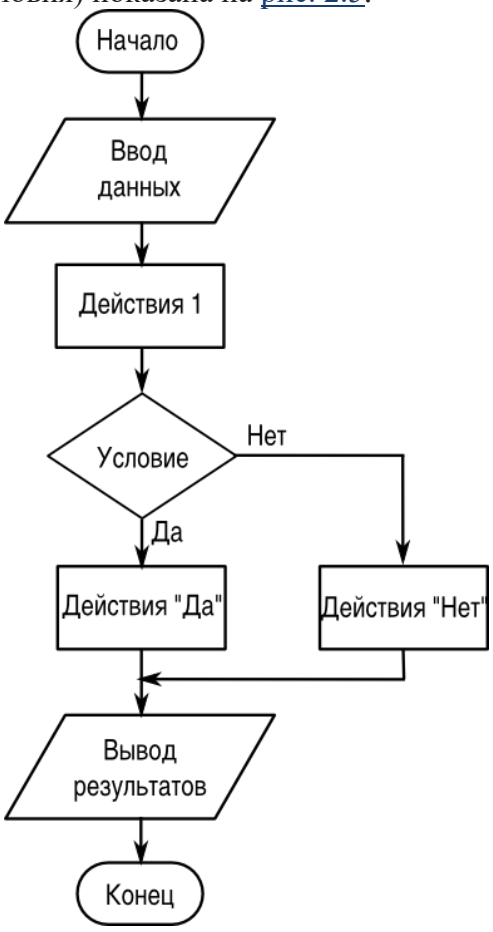
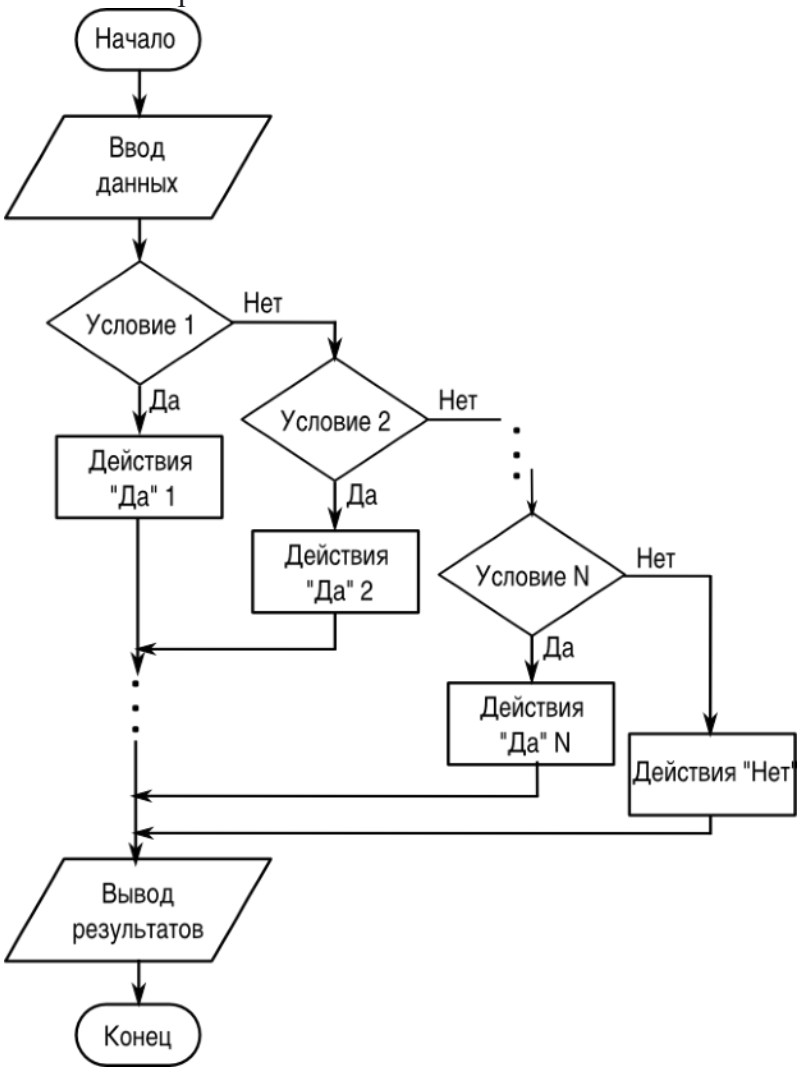


Рис. 2.5. Типовая схема алгоритма с ветвлением

Если условие, указанное в блоке "Условие", выполняется, то далее производятся действия, соответствующие "ветви ДА" ("Действия ДА"), иначе выполняются действия, соответствующие "ветви НЕТ" (Действия НЕТ"). Условия нужно составлять так, чтобы

результат проверки любого условия допускал только два исхода – условие либо выполняется, либо не выполняется.
В случае, когда одной проверкой не удаётся охватить все варианты, используются "вложенные" условия, как показано на рис. 2.6. Условия могут быть вложены друг в друга любое количество раз (уровень вложенности не ограничен). Такая ситуация также называется "выбор".
В языках программирования для обеспечения проверки условий используется специальный составной оператор IF ("если"). В этом операторе указывается условие, которое нужно проверить, и действия для ветвей "ДА" и "НЕТ".
Чтобы понять, как работает оператор IF, рассмотрим типичные задачи на проверку условий и выбор.



увеличить изображение

Рис. 2.6. Блок-схема алгоритма выбора

Задача 1. Составить программу ввода значения температуры воздуха t и выдачи текста "Хорошая погода!", если $t > 10$ градусов и текста "Плохая погода!", если $t \leq 10$ градусов[2].
Постановка задачи: Исходными данными является значение t, необходимо сформировать строку s. При $t < 10$ s='Плохая погода!', иначе s='Хорошая погода!'.
Блок-схема алгоритма показана на рис. 2.7.

Текст программы на "псевдоязыке":
ввод t
если (t <10) то
	s='Плохая погода!'
иначе
	s='Хорошая погода!'
конец если
вывод s

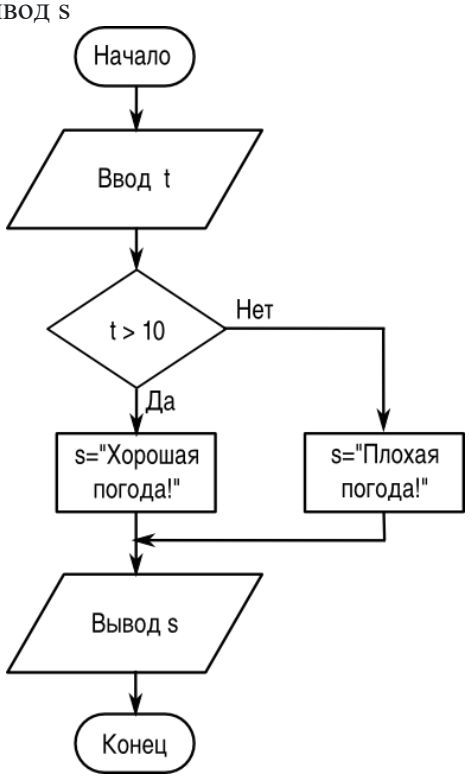


Рис. 2.7. Блок-схема алгоритма задачи про погоду
Текст на Python:

```
# -*- coding: utf-8 -*-
#
t=input('Введите температуру в градусах: ')
if t <10:
s='Плохая погода!'
else:
s='Хорошая погода!'
print s
```

Начало каждой "ветви" программы обозначается символом ":". Условие в операторе IF ("если") записывается без скобок. Как таковое окончание оператора IF отсутствует. Python считает, что следующий оператор начинается в строке без отступа. Таким образом, в Python отступы играют важную роль.

Задача 2 (источник тот же). Составить программу ввода оценки P, полученной учащимся, и выдачи текста "Молодец!", если $P = 5$, "Хорошо!", если $P = 4$ и "Лентяй!", если $P \leq 3$.

Постановка задачи: Дано значение P, которое является натуральным числом и не может быть больше 5. В зависимости от величины P нужно сформировать строку s по

правилам, указанным в условии. Необходимо выполнить две последовательные проверки значения P.
Блок-схема алгоритма показана на рис. 2.8.

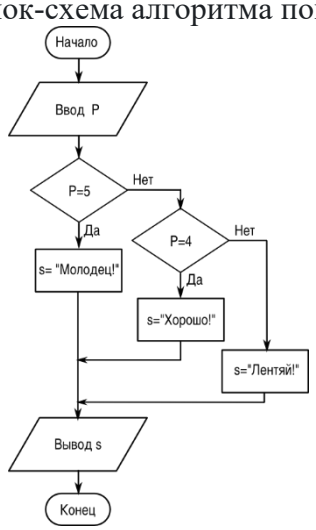


Рис. 2.8. Блок-схема алгоритма к задаче про оценки
Текст программы на "псевдоязыке":

```
ввод P
если (P=5) то
        s='Молодец!'
иначе если (P=4)
        s='Хорошо!'
иначе
        s='Лентяй!'
конец если
вывод s
```

Программа на Python:

```
# -*- coding: utf-8 -*-
#
P=input('Ваши баллы? ')
if P==5:
s='Молодец!'
elif P==4:
s='Хорошо!'
else:
s='Лентяй!'
print s
```

Ключевое слово **elif** в Python является сокращением от **else if** ("иначе если") и используется для организации вложенных условий (алгоритмов выбора).

2.2.1 Задачи для самостоятельного решения

1. Дано натуральное число. Определить, будет ли это число: чётным, кратным 4.
2. Дано натуральное число. Определить, будет ли это число: нечётным, кратным 5.
3. Дано натуральное число. Определить, будет ли это число: нечётным, кратным 7.
4. Дано натуральное число. Определить, будет ли это число: чётным, кратным 10.
5. Имеется коробка со сторонами: $A \times B \times C$. Определить, пройдёт ли она в дверь с размерами $M \times K$.

6. Дано вещественное число. Определить, какое это число: положительное, отрицательное, ноль.
7. Можно ли из бревна, имеющего диаметр поперечного сечения D, выпилить квадратный брус шириной A?
8. Можно ли в квадратном зале площадью S поместить круглую сцену радиусом R так, чтобы от стены до сцены был проход не менее K?
9. Дан номер места в плацкартном вагоне. Определить, какое это место: верхнее или нижнее, в купе или боковое.
10. Известна денежная сумма. Разменять её купюрами 500, 100, 10 и монетой 2 руб., если это возможно.
11. Имеются две ёмкости: кубическая с ребром A, цилиндрическая с высотой H и радиусом основания R. Определить, поместится ли жидкость объёма M в первую ёмкость, во вторую, в обе.
12. Имеются две ёмкости: кубическая с ребром A, цилиндрическая с высотой H и радиусом основания R. Определить, можно ли заполнить жидкостью объёма M первую ёмкость, вторую, обе.
13. Даны вещественные числа: X, Y, Z. Определить, существует ли треугольник с такими длинами сторон и, если существует, будет ли он прямоугольным.
14. Дано число X. Определить, принадлежит ли это число заданному промежутку $[a, b]$.
15. Определить значение функции $Z = 1/(XY)$ при произвольных X и Y.
16. Даны вещественные числа: A, B, C. Определить, выполняются ли неравенства $A < B < C$ или $A \geq B \geq C$ и какое именно неравенство выполняется.
17. Даны вещественные числа X и Y. Вычислить Z. $Z = \sqrt{X * Y}$ при $X > Y$, $Z = \ln(X + Y)$ в противном случае.
18. Даны вещественные положительные числа a, b, c, d. Выясните, может ли прямоугольник со сторонами a, b уместиться внутри прямоугольника со сторонами c, d так, чтобы каждая сторона внутреннего прямоугольника была параллельна или перпендикулярна стороне внешнего прямоугольника.
19. Дано вещественное число A. Вычислить $f(A)$, если $f(x) = x^2 + 4x + 5$, при $x \leq 2$; в противном случае $f(x) = 1/(x^2 + 4x + 5)$.
20. Дано вещественное число A. Вычислить $f(A)$, если $f(x) = 0$, при $x \leq 0$; $f(x) = x$ при $0 < x \leq 1$, в противном случае $f(x) = x^4$.
21. Дано вещественное число A. Вычислить $f(A)$, если $f(x) = 0$ при $x \leq 0; f(x) = x^2 - x$ при $0 < x \leq 1$, в противном случае $f(x) = x^2 - \sin(\pi x^2)$.

22. Составить алгоритм и программу для реализации логических операций "И" и "ИЛИ" для двух переменных.
23. Известен ГОД. Определить, будет ли этот год високосным, и к какому веку этот год относится.

Указание. При вычислении корней и логарифмов используйте функции sqrt() и log() модуля $math$. В этом же модуле определена константа pi ($math.pi$).

Лабораторная работа 2. Циклические алгоритмы. Обработка последовательностей и одномерных массивов

Требования к отчету: Выполненные задания с результатами вывода оформить в виде отчета со скриншотами и прикрепить здесь. На каждом скриншоте в коде решения задачи прописать ФИО автора решения задачи и группу, в которой учится слушатель.
Циклом называется фрагмент алгоритма или программы, который может повторяться несколько раз (в том числе и нуль раз). Каждая циклическая конструкция начинается заголовком цикла и заканчивается конечным оператором. Между ними располагаются операторы, называемые "телом цикла". Количество повторений выполнения команд (операторов), составляющих тело цикла, определяется условием окончания цикла. Условием окончания может быть достижение некоторого значения специальной переменной, называемой параметром цикла (переменной цикла), или выполнение (прекращение выполнения) некоторого условия.

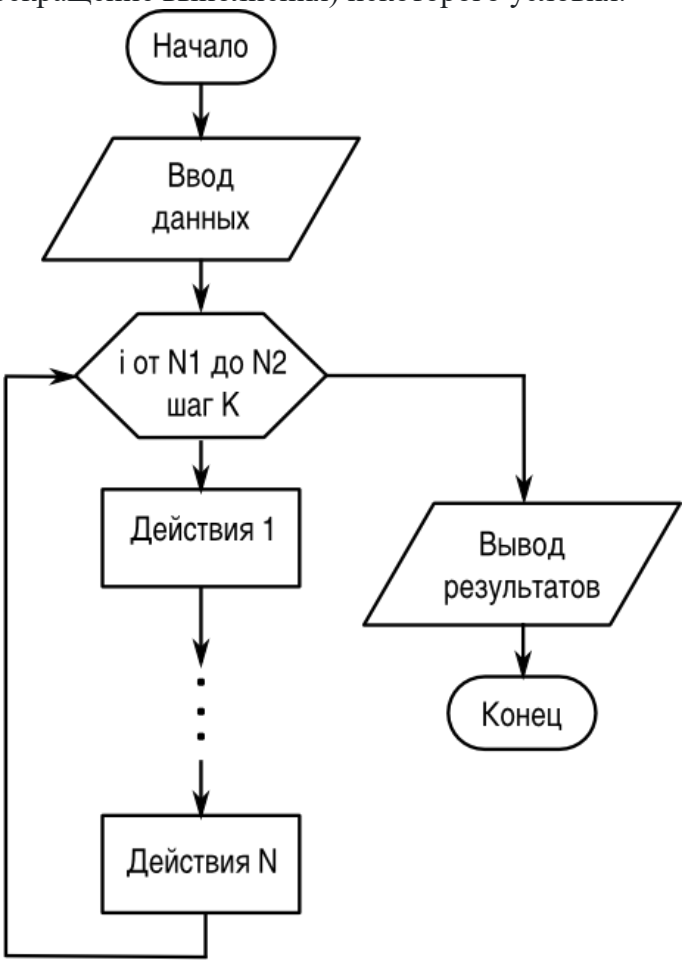


Рис. 2.9. Пример блок-схемы цикла с параметром
Для организации циклов с параметром в языках программирования используется составной оператор FOR ("для"), а в циклах с условием чаще всего используется составной оператор WHILE ("пока").
В случае цикла с параметром количество повторений ("оборотов") цикла известно заранее и задаётся специальным выражением в заголовке цикла, а в случае цикла с условием при каждом следующем повторении требуется проверять условие прекращения цикла.

Если при написании операторов в теле цикла допущена ошибка, условие прекращения

цикла может не выполниться никогда и цикл окажется бесконечным ("программа зациклится").
Пример блок-схемы цикла с параметром (переменной) показан на рис. 2.9, а пример блок-схемы цикла с условием окончания – на рис. 2.10. Для обозначения заголовка цикла с параметром используется специальный графический элемент – блок модификации, в котором указывается правило изменения параметра цикла.
Для работы с одномерными массивами целесообразно использовать циклы с параметром, поскольку до начала цикла может быть определено количество повторений. В этом случае цикл с параметром требуется для ввода элементов массива, а для выполнения каких-либо действий с этими элементами и вывода результатов также могут потребоваться циклы.

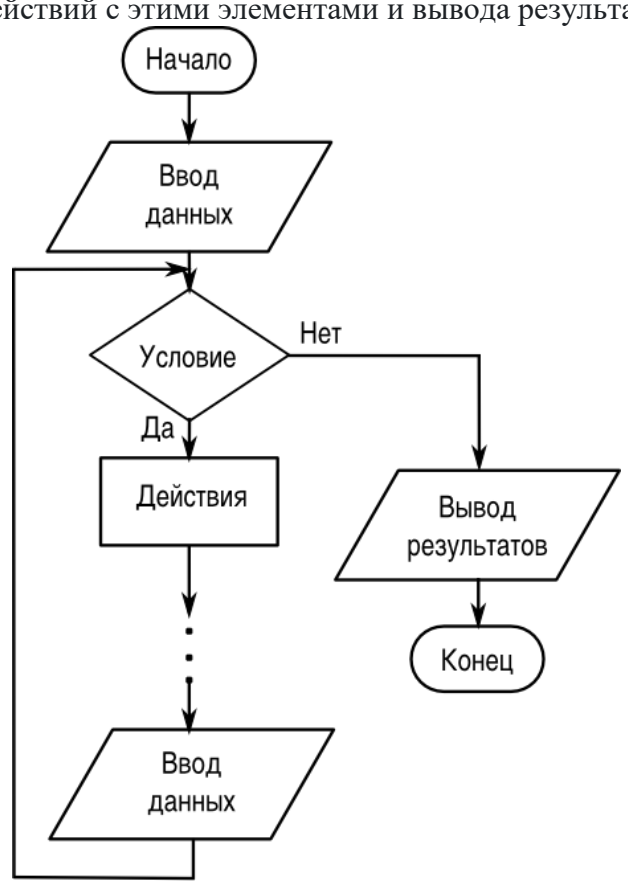


Рис. 2.10. Пример блок-схемы цикла с условием
В блок-схеме на рис. 2.10 действия повторяются, пока выполняется некоторое условие. Когда условие перестаёт выполняться, цикл завершается.
Такие циклы целесообразно использовать в ситуации, когда данные вводятся (поступают из какого-то источника), пока не произойдёт некоторое событие. При этом всю обработку чаще всего приходится выполнять "на лету", не создавая массив, поскольку количество элементов заранее неизвестно.
Рассмотрим типичные задачи, решение которых требует вычислений в цикле.
Задача 1. Дан одномерный массив A числовых значений, насчитывающий N элементов. Найти среднее арифметическое элементов массива.
Постановка задачи:
Дано:

- N – количество элементов в массиве;
- i - индекс элемента массива (параметр цикла).

- $A[i]$ – элемент массива;

Найти:

- S – сумма элементов массива
- C – среднее арифметическое элементов массива, C=S/N.

Блок-схема алгоритма показана на рис. 2.11.

Текст программы на "псевдоязыке":

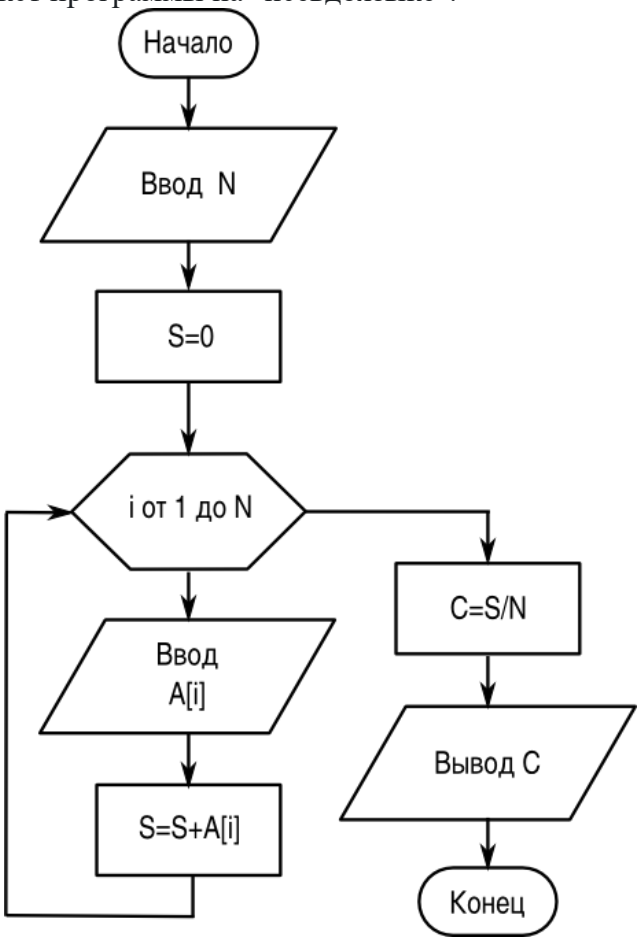


Рис. 2.11. Блок-схема алгоритма вычисления среднего значения в массиве

```
ввод N
S=0
нц для i от 1 до N
        ввод A[i]
        S=S+A[i]
кц
C=S/N
вывод C
```

Здесь **нц** и **кц** обозначают, соответственно, начало и конец цикла, строка с **нц** является заголовком цикла. Как видно из текста, указываются начальное и конечное значение переменной цикла, которая обязательно должна быть целым числом. В приведённой здесь записи переменная цикла увеличивается на 1 при каждом повторении ("шаг переменной цикла" равен 1). Если требуется шаг, не равный 1, это указывается специально.

Тело цикла состоит из двух операторов – ввода очередного числа и прибавления этого числа к текущему значению суммы.

На Python можно написать практически то же самое (с учётом особенностей, связанных с использованием функции range()).

```
# -*- coding: utf-8 -*-
#
N=input('Количество элементов: ')
S=0
for i in range(N-1) :
   a=input('Введите число: ')
   S=S+a
C=S/N
print'Результат:',C
```

Поскольку диапазон чисел, формируемых функцией range(), начинается с 0, то верхней границей должно быть $N-1$. Так как массив хранить нет необходимости, можно просто вводить числа и добавлять их к текущему значению суммы.

Тело цикла начинается после символа ":", и все операторы тела цикла в Python должны иметь одинаковый отступ от начала строки. Как только отступ исчезает, Python считает, что тело цикла закончилось.

А вот вариант решения этой же задачи на Python с использованием списка и методов списка.

```
# -*- coding: utf-8 -*-
#
N=input('Количество элементов: ')
S=0
lst=[]
for i in range(N-1) :
   a=input('Введите число: ')
   lst.append(a)
C=sum(lst)/N
print'Результат:',C
```

В этом варианте формируется список, а сумма элементов списка вычисляется с помощью встроенной функции. Программа увеличилась на одну строку (создание пустого списка), но зато мы научились формировать список в цикле.

Задача 2. Определить, является ли введённая строка палиндромом ("перевёртышем") типа ABBA, kazak и пр.

Постановка задачи: Требуется сравнивать попарно символы с начала и с конца строки S (первый и последний, второй и предпоследний и т.д.). Если в каждой такой паре символы одинаковы, строка является палиндромом. Соответственно, каждая проверка пары символов должна получить некоторый признак (flag – "флаг"), который будет равен 1, если символы в паре совпадают и 0, если не совпадают. Окончательный результат обработки строки получится как произведение всех значений "флагов". Если хотя бы один раз "флаг" оказался равен нулю, строка палиндромом не является и произведение всех "флагов" окажется равным 0. Количество пар не превышает половины длины строки L (точно равно половине длины для строк с чётным количеством символов и результат целочисленного деления длины строки на 2 для строк с нечётным количеством символов, поскольку "центральный" символ строки с нечётным количеством символов очевидно совпадает сам с собой).

Блок-схема алгоритма показана на рис. 2.12.

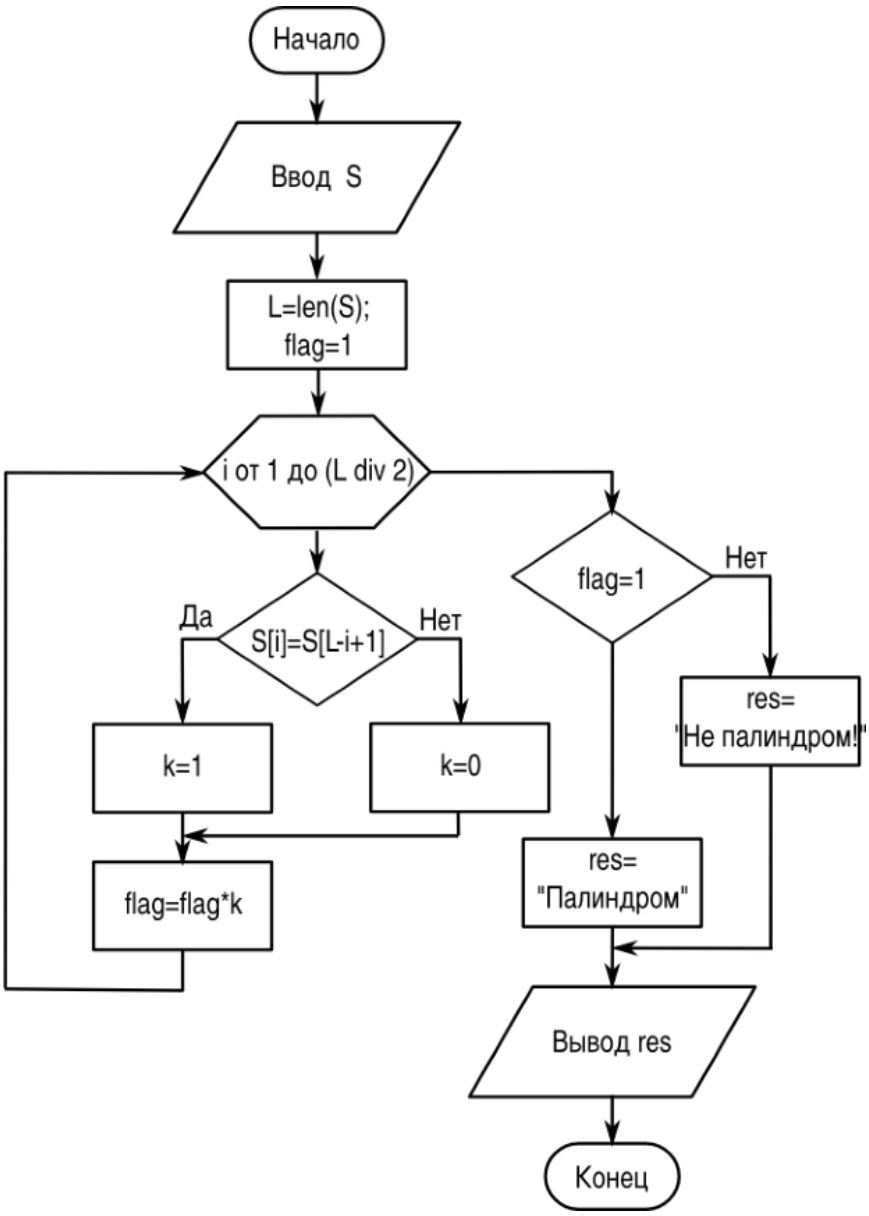


Рис. 2.12. Блок-схема алгоритма определения палиндрома

Текст программы на "псевдоязыке":

```
ввод S
flag=1
L=длина(S)
N=L div 2
нц для i от 1 до N
        если S[i]=S[L-i +1] то
                k=1
        иначе
                k=0
        конец если
        flag=flag *k
кц
если flag=1 то
        вывод 'Палиндром'
```

иначе
вывод 'Не палиндром!'
конец если

При проверке каждой пары устанавливается коэффициент k , который затем умножается на текущее значение "флага". Окончательный вывод делается по итоговому значению "флага".

Текст программы на Python может быть очень похож на текст на псевдоязыке.

```
# -*- coding: utf-8 -*-
#
s1=raw_input('Исходная строка: ')
# Определяем длину строки
L=len(s1)
flag=1
for i in range(L//2):
   if s1[i]==s 1[-i-1]:
      k=1
   else:
      k=0
   flag=flag*k
if flag==1:
   print 'Палиндром'
else:
   print 'Не палиндром!'
```

Для ввода строки использован оператор raw_input(), при этом не требуется записывать строку в кавычках.

Небольшие синтаксические особенности всё-таки есть – условие равенства двух переменных записывается знаком "==", начало каждого составного оператора обозначается символом ":", и, как всегда, необходимо следить за отступами. Кроме того, чтобы отсчитывать символы с конца строки, использованы "отрицательные" индексы элементов строки.

Однако использование особенностей строк в Python, их функций и методов, позволяет решить эту задачу более изящно. Например, так.

```
# -*- coding: utf-8 -*-
#
s1=raw_input('Исходная строка: ')
lst=list(s1)
lst.reverse()
s 2=' '.join(lst)
if s1==s2:
   print'Палиндром'
else:
   print'Не палиндром!'
```

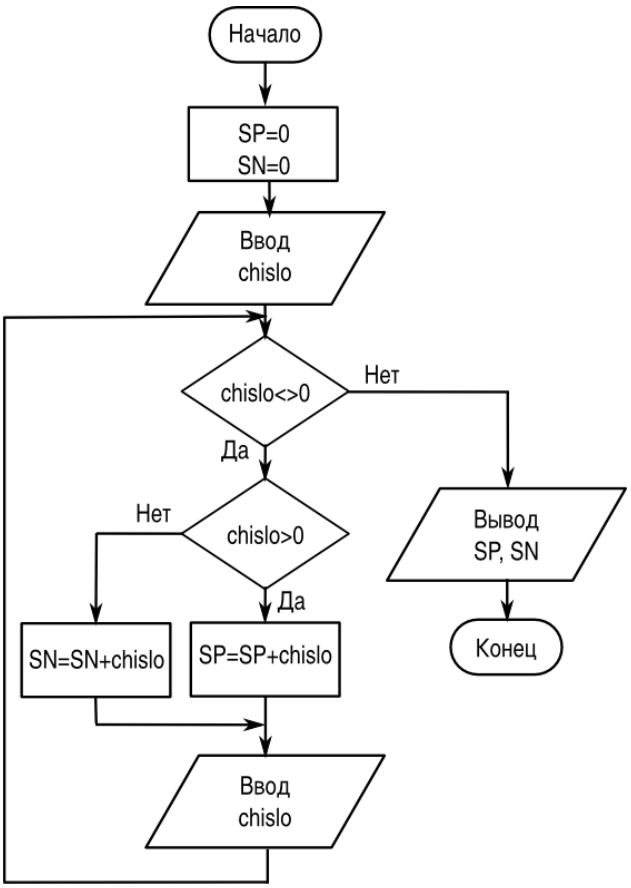


Рис. 2.13. Блок-схема алгоритма обработки последовательности

Здесь исходная строка преобразуется в список, затем список "переворачивается" и из него с помощью пустой "строки-объединителя" формируется новая строка. Затем строки сравниваются. Цикл оказывается не нужен! Всю работу делает Python.

Если количество повторений операций заранее неизвестно, но известно условие прекращения выполнения операций, используется цикл (составной оператор) WHILE. Покажем его использование на следующем примере.

Задача 3. Последовательно вводятся ненулевые числа. Определить сумму положительных и сумму отрицательных чисел. Закончить ввод чисел при вводе 0.

Задача настолько проста, что дополнительных уточнений в качестве постановки задачи не требуется. Пусть сумма положительных чисел называется SP, а сумма отрицательных чисел – SN.

Блок-схема алгоритма показана на рис. 2.13.

Текст программы на "псевдоязыке":

```
SP=0
SN=0
ввод chislo
нц пока chislo <> 0
        если chislo > 0 то
                SP=SP+chislo
        иначе
                SN=SN+chislo
        конец если
ввод chislo
кц
```

вывод SP
вывод SN
Условие "неравенства" в языках программирования Pascal и BASIC обозначается как "<>", поэтому здесь сохранено это обозначение.
Следует обратить внимание, что проверяемое число нужно определить до начала цикла, поскольку возможна ситуация, что неопределённое значение окажется равным 0 и программа закончится, не успев начаться. А потом числа вводятся в цикле и каждое вновь поступившее число сравнивается с 0 (после ввода каждого числа следует проверка условия). Порядок операций и проверок в цикле WHILE может оказаться важным для получения верного результата.
Текст программы на Python не имеет каких-то существенных особенностей. Для удобства чтения программа поделена на "блоки" с помощью символа комментария.

```
# -*- coding: utf-8 -*-
#
SP=0
SN=0
#
chislo=input('Следующее число: ')
#
while chislo!=0:
   if chislo > 0 :
      SP=SP+chislo
   else:
      SN=SN+chislo
   chislo=input('Следующее число: ')
#
print'Сумма положительных:',SP
print'Сумма отрицательных:',SN
```

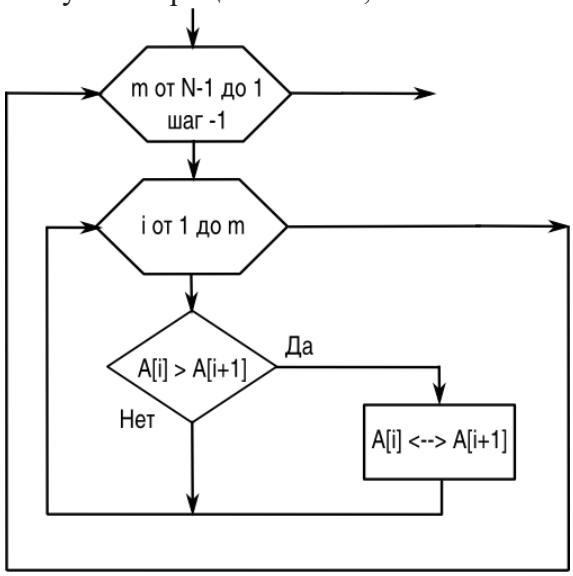


Рис. 2.14. Алгоритм сортировки "методом пузырька"

2.3.1 Сортировка массива

Задача сортировки, а также задача поиска максимального или минимального элемента в массиве встречается довольно часто. Средствами Python такие задачи решаются очень просто, но тем не менее рассмотрим общую задачу сортировки массива.

Под сортировкой понимается процедура, в результате выполнения которой изменяется исходный порядок следования данных. Причём новый порядок их следования отвечает требованию возрастания или убывания значений элементов одномерного массива. Например, при сортировке по возрастанию из одномерного массива[3 1 0 5 2 7] получается массив[0 1 2 3 5 7]. Возможны и более сложные критерии сортировки. Символьные данные обычно сортируются в алфавитном порядке.
Один из наиболее наглядных методов сортировки – "метод пузырька".
Пусть необходимо упорядочить элементы массива A из N элементов по возрастанию. Просматривая элементы массива "слева направо" (от первого элемента к последнему), меняем местами значения каждой пары соседних элементов в случае неравенства $A[i] > A[i+1]$, передвигая тем самым наибольшее значение на последнее место. Следующие просмотры начинаем опять с первого элемента массива, последовательно уменьшая на единицу количество просматриваемых элементов. Процесс заканчивается после $N-1$ просмотра.
Метод получил такое название, потому что каждое наибольшее значение как бы всплывает вверх.
Фрагмент блок-схемы алгоритма показан на рис. 2.14.
Действие $A[i] \leftrightarrow A[i+1]$ означает перестановку значений элементов массива.
Текст соответствующего фрагмента программы на "псевдоязыке":

```
ввод N, A
нц для m от N-1 до 1 шаг -1
        нц для i от 1 до m
                если A[i] > A[i +1] то
                X=A[i]
                A[i]=A[i +1]
                A[i +1]=X
                конец если
        кц
кц
вывод A
```

В этом фрагменте для перестановки значений элементов массива используется промежуточная переменная.
Задача поиска максимального элемента в массиве решается следующим образом.
Пусть maxA – требуемое значение максимального элемента. Сначала присваиваем переменной maxA значение первого элемента массива, потом сравниваем первый элемент со следующим. Если следующий элемент (второй) больше первого, присваиваем его значение переменной maxA, а если нет – переходим к следующему (третьему элементу) и т. д.
Аналогично решается задача поиска минимального элемента в массиве.
В Python эти алгоритмы уже реализованы в функциях max(), min() и в методе sort() (метод sort() сортирует список по возрастанию значений элементов).

2.3.2 Задачи для самостоятельного решения

1. Составьте блок-схему поиска максимального элемента в одномерном массиве.
2. Нарисуйте полную блок-схему алгоритма сортировки массива "методом пузырька".
3. Дан одномерный массив числовых значений, насчитывающий N элементов. Поменять местами элементы, стоящие на чётных и нечётных местах: $A[1] \leftrightarrow A[2]; A[3] \rightarrow A[4]...$

4. Дан одномерный массив числовых значений, насчитывающий N элементов. Выполнить перемещение элементов массива по кругу вправо, т. е. $A[1] \to A[2]; A[2] \to A[3]; ...A[n] \to A[1]$.
5. Дан одномерный массив числовых значений, насчитывающий N элементов. Поменять местами первую и вторую половины массива.
6. Дан одномерный массив числовых значений, насчитывающий N элементов. Поменять местами группу из M элементов, начинающихся с позиции K с группой из M элементов, начинающихся с позиции P.
7. Дан одномерный массив числовых значений, насчитывающий N элементов. Вставить группу из M новых элементов, начиная с позиции K.
8. Дан одномерный массив числовых значений, насчитывающий N элементов. Сумму элементов массива и количество положительных элементов поставить на первое и второе место.
9. Дан одномерный массив числовых значений, насчитывающий N элементов. Исключить из него M элементов, начиная с позиции K.
10. Дан одномерный массив числовых значений, насчитывающий N элементов. Исключить все нулевые элементы.
11. Дан одномерный массив числовых значений, насчитывающий N элементов. После каждого отрицательного элемента вставить новый элемент, равный квадрату этого отрицательного элемента.
12. Дан одномерный массив числовых значений, насчитывающий N элементов. Определить, образуют ли элементы массива, расположенные перед первым отрицательным элементом, возрастающую последовательность.
13. Дан одномерный массив числовых значений, насчитывающий N элементов. Определить, образуют ли элементы массива, расположенные перед первым отрицательным элементом, убывающую последовательность.
14. Дан одномерный массив числовых значений, насчитывающий N элементов. Из элементов исходного массива построить два новых. В первый должны входить только элементы с положительными значениями, а во второй – только элементы с отрицательными значениями.
15. Дан одномерный массив числовых значений, насчитывающий N элементов. Добавить столько элементов, чтобы элементов с положительными и отрицательными значениями стало бы поровну.
16. Дан одномерный массив числовых значений, насчитывающий N элементов. Добавить к элементам массива такой новый элемент, чтобы сумма элементов с положительными значениями стала бы равна модулю суммы элементов с отрицательными значениями.
17. Дан одномерный массив числовых значений, насчитывающий N элементов. Дано положительное число T. Разделить это число между положительными элементами массива пропорционально значениям этих элементов и добавить полученные доли к соответствующим элементам.
18. Дан одномерный массив числовых значений, насчитывающий N элементов. Исключить из массива элементы, принадлежащие промежутку $[B; C]$.

19. Дан одномерный массив числовых значений, насчитывающий N элементов. Вместо каждого элемента с нулевым значением поставить сумму двух предыдущих элементов массива.
20. Дан одномерный массив числовых значений, насчитывающий N элементов. Определить, имеются ли в массиве два подряд идущих нуля.
21. Дан одномерный массив числовых значений, насчитывающий N элементов. Подсчитать количество чисел, делящихся на 3 нацело, и среднее арифметическое чисел с чётными значениями. Поставить полученные величины на первое и последнее места в массиве (увеличив массив на 2 элемента).
22. Заданы M строк символов, которые вводятся с клавиатуры. Найти количество символов в самой длинной строке. Выровнять строки по самой длинной строке, поставив перед каждой строкой соответствующее количество звёздочек.
23. Заданы M строк символов, которые вводятся с клавиатуры. Из заданных строк, каждая из которых представляет одно слово, составить одну длинную строку, разделяя слова пробелами.
24. Заданы M строк слов, которые вводятся с клавиатуры. Подсчитать количество гласных букв в каждой из заданных строк.
25. Заданы M строк слов, которые вводятся с клавиатуры (в каждой строке – одно слово). Вводится слог (последовательность букв). Подсчитать количество таких слогов в каждой строке.
26. Заданы M строк слов, которые вводятся с клавиатуры (в каждой строке – одно слово). Вводится слог (последовательность букв). Удалить данный слог из каждой строки.
27. Заданы M строк символов, которые вводятся с клавиатуры. Напечатать все центральные буквы строк нечетной длины.
28. Заданы M строк символов, которые вводятся с клавиатуры. Каждая строка содержит слово. Записать каждое слово в разрядку (вставить по пробелу между буквами).
29. Задана строка символов, в которой встречается символ ".". Поставить после каждого такого символа системное время ПК.
30. Заданы M строк, которые вводятся с клавиатуры. Подсчитать количество пробелов в каждой из строк.
31. Заданы M строк символов, которые вводятся с клавиатуры. Каждая строка представляет собой последовательность символов, включающих в себя вопросительные знаки. Заменить в каждой строке все имеющиеся вопросительные знаки звёздочками.
32. Последовательно вводятся числа. Определить сумму чисел с нечётными номерами и произведение чисел с чётными номерами (по порядку ввода). Подсчитать количество слагаемых и количество сомножителей. При вводе числа 55555 закончить работу.
33. Определить сумму вводимых положительных чисел. Причём числа с нечётными номерами (по порядку ввода) суммировать с обратным знаком, а числа с чётными номерами перед суммированием возводить в квадрат. Подсчитать количество слагаемых. При вводе первого отрицательного числа закончить работу.
34. Даны число P и число H. Определить сумму чисел меньше P, произведение чисел больше H и количество чисел в диапазоне значений P и H. При вводе числа равного P или H, закончить работу.

35. Суммировать вводимые числа, среди которых нет нулевых. При вводе нуля обеспечить вывод текущего значения суммы. При вводе числа 99999 закончить работу.
36. Вводятся положительные числа. Определить сумму чисел, делящихся на положительное число B нацело. При вводе отрицательного числа закончить работу.
37. Для вводимых чисел определить процент положительных и отрицательных чисел. При вводе числа -65432 закончить работу.

Лабораторная работа № 3. Обработка двумерных массивов (матриц). Работа с ассоциативными массивами (таблицами данных)

Требования к отчету: Выполненные задания с результатами вывода оформить в виде отчета со скриншотами и прикрепить здесь. На каждом скриншоте в коде решения задачи прописать ФИО автора решения задачи и группу, в которой учится слушатель.

Обработка двумерных массивов (матриц)
Двумерные массивы являются аналогами матриц и имеют "прямоугольную" (табличную) структуру. Описываются массивы так же, как одномерные. Разница состоит в том, что у элемента двумерного массива две координаты (два индекса) – номер строки и номер столбца, в которых находится элемент.
Ввод массива осуществляется построчно при помощи двух циклов. Пусть M – количество столбцов, N – количество строк. Элементы массива обозначим как mas[i, j], первый индекс – номер строки, второй – номер столбца.

В Python для работы с многомерными (когда используется два и более индексов) массивами можно использовать вложенные списки (списки списков, списки списков списков и т. д.).
Однако Python предоставляет более удобный инструмент создания и преобразования многомерных массивов – библиотеку numpy (Numeric Python).
Создание двумерного массива в Python может выглядеть так:

```
# -*- coding: utf-8 -*-
#
import numpy
n=input('Количество строк: ')
m=input('Количество столбцов: ')
# Создаём "нулевую" матрицу
a=numpy.zeros([n-1,m-1])
# Заполняем матрицу
for i in range(n-1) :
   for j in range(m-1) :
      print'Элемент матрицы[',i,'][',j,']'
      a[i, j]=input('Введите элемент: ')
#
```

Сначала с помощью функции (метода) numpy.zeros() создаётся двумерный массив (матрица), заполненный нулями, а потом вместо нулей подставляются реальные значения. Индексы элементов, так же как в строках, кортежах и списках, начинаются с 0 (первый – верхний левый – элемент матрицы в Python имеет индекс [0,0]). Оператор **print** выводит индексы очередного элемента матрицы, который нужно ввести.

Задача 1. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Найти среднее арифметическое элементов массива.
Постановка задачи:
Дано:

- n – количество строк в массиве;
- m – количество столбцов в массиве;
- $A[i, j]$ – элемент массива;
- i, j – индексы элемента массива.

Найти:

- S – сумма элементов массива (сумма всех $A[i,j]$ при всех i и j)
- K – количество элементов в массиве ($K = m * n$)
- C – среднее арифметическое элементов массива ($C = S/K$)

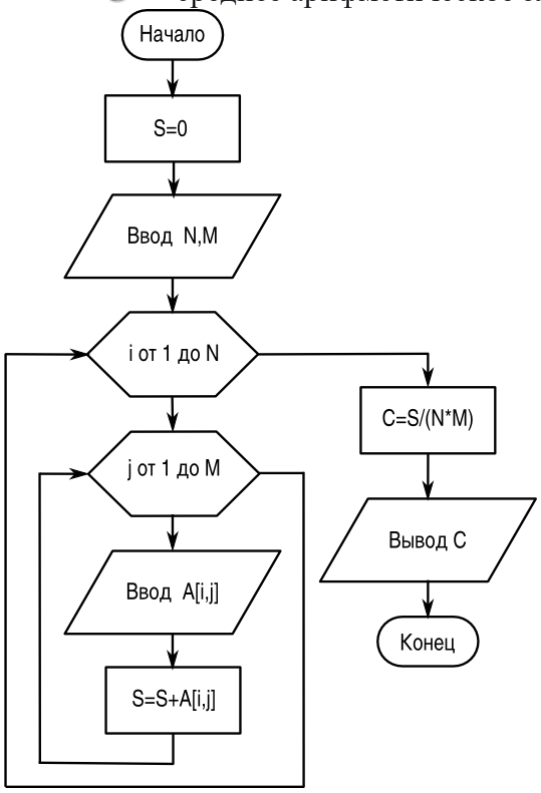


Рис. 2.15. Блок-схема алгоритма вычисления среднего значения матрицы

Блок-схема алгоритма решения показана на рис. 2.15.

Текст программы на "псевдоязыке":

```
ввод n,m
S=0
нц для i от 1 до n
        нц для j от 1 до m
                ввод A[i, j]
                S=S+A[i, j]
        кц
кц
K=n*m
C=S/K
вывод C
```

Текст программы на Python:

```
# -*- coding: utf-8 -*-
#
import numpy
n=input('Количество строк: ')
m=input('Количество столбцов: ')
S=0.0
# Создаём нулевую матрицу
a=numpy.zeros([n-1,m-1])
```

```
# Заполняем матрицу
for i in range(n-1) :
   for j in range(m-1) :
      print 'Элемент матрицы [',i,'][',j,']'
      a[i, j]=input('Введите элемент: ')
      S=S+a[i, j]
#
K=n*m
C=S/K
print 'Среднее значение по строкам:',C
```

2.4.1 Задачи для самостоятельного решения

1. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти наибольший элемент столбца матрицы A , для которого сумма абсолютных значений элементов максимальна.
2. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти наибольшее значение среди средних значений для каждой строки матрицы.
3. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти наименьший элемент столбца матрицы A , для которого сумма абсолютных значений элементов максимальна.
4. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти наименьшее значение среди средних значений для каждой строки матрицы.
5. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Определить средние значения по всем строкам и столбцам матрицы. Результат оформить в виде матрицы из $N+1$ строк и $M+1$ столбцов.
6. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти сумму элементов всей матрицы. Определить, какую долю в этой сумме составляет сумма элементов каждого столбца. Результат оформить в виде матрицы из $N+1$ строк и M столбцов.
7. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Найти сумму элементов всей матрицы. Определить, какую долю в этой сумме составляет сумма элементов каждой строки. Результат оформить в виде матрицы из N строк и $M+1$ столбцов.
8. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Определить, сколько отрицательных элементов содержится в каждом столбце и в каждой строке матрицы. Результат оформить в виде матрицы из $N+1$ строк и $M+1$ столбцов.
9. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Определить, сколько нулевых элементов содержится в верхних L строках матрицы и в левых K столбцах матрицы.

10. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Перемножить элементы каждого столбца матрицы с соответствующими элементами K -го столбца.
11. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Просуммировать элементы каждой строки матрицы с соответствующими элементами L -й строки.
12. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Разделить элементы каждой строки на элемент этой строки с наибольшим значением.
13. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Разделить элементы каждого столбца матрицы на элемент этого столбца с наибольшим значением.
14. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Разделить элементы матрицы на элемент матрицы с наибольшим значением.
15. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Все элементы имеют целый тип. Дано целое число H . Определить, какие столбцы имеют хотя бы одно такое число, а какие не имеют.
16. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Исключить из матрицы строку с номером L . Сомкнуть строки матрицы.
17. Выполнить обработку элементов прямоугольной матрицы A , имеющей N строк и M столбцов. Добавить к матрице строку и вставить ее под номером L .
18. Выполнить обработку элементов квадратной матрицы A , имеющей N строк и N столбцов. Найти сумму элементов, стоящих на главной диагонали, и сумму элементов, стоящих на побочной диагонали (элементы главной диагонали имеют индексы от $[0, 0]$ до $[N, N]$, а элементы побочной диагонали – от $[N, 0]$ до $[0, N]$).
19. Выполнить обработку элементов квадратной матрицы A , имеющей N строк и N столбцов. Определить сумму элементов, расположенных параллельно главной диагонали (ближайшие к главной). Элементы главной диагонали имеют индексы от $[0, 0]$ до $[N, N]$.
20. Выполнить обработку элементов квадратной матрицы A , имеющей N строк и N столбцов. Определить произведение элементов, расположенных параллельно побочной диагонали (ближайшие к побочной). Элементы побочной диагонали имеют индексы от $[N, 0]$ до $[0, N]$.
21. Выполнить обработку элементов квадратной матрицы A , имеющей N строк и N столбцов. Каждой паре элементов, симметричных относительно главной диагонали (ближайшие к главной), присвоить значения, равные полусумме этих симметричных значений (элементы главной диагонали имеют индексы от $[0, 0]$ до $[N, N]$).

22. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Исходная матрица состоит из нулей и единиц. Добавить к матрице еще один столбец, каждый элемент которого делает количество единиц в каждой строке чётным.
23. Выполнить обработку элементов квадратной матрицы A, имеющей N строк и N столбцов. Найти сумму элементов, расположенных выше главной диагонали, и произведение элементов, расположенных выше побочной диагонали (элементы главной диагонали имеют индексы от $[0, 0]$ до $[N, N]$, а элементы побочной диагонали – от $[N, 0]$ до $[0, N]$).
24. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Дан номер строки L и номер столбца K, при помощи которых исходная матрица разбивается на четыре части. Найти сумму элементов каждой части.
25. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Определить, сколько нулевых элементов содержится в каждом столбце и в каждой строке матрицы. Результат оформить в виде матрицы из $N + 1$ строк и $M + 1$ столбцов.
26. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Дан номер строки L и номер столбца K, при помощи которых исходная матрица разбивается на четыре части. Найти среднее арифметическое элементов каждой части.
27. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Все элементы имеют целый тип. Дано целое число H. Определить, какие строки имеют хотя бы одно такое число, а какие не имеют.
28. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Исключить из матрицы столбец с номером K. Сомкнуть столбцы матрицы.
29. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Добавить к матрице столбец чисел и вставить его под номером K.
30. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Добавить к элементам каждого столбца такой новый элемент, чтобы сумма положительных элементов стала бы равна модулю суммы отрицательных элементов. Результат оформить в виде матрицы из $N + 1$ строк и M столбцов.
31. Выполнить обработку элементов прямоугольной матрицы A, имеющей N строк и M столбцов. Добавить к элементам каждой строки такой новый элемент, чтобы сумма положительных элементов стала бы равна модулю суммы отрицательных элементов. Результат оформить в виде матрицы из N строк и $M + 1$ столбцов

Работа с ассоциативными массивами (таблицами данных)

Ассоциативный массив лучше всего описывается табличным представлением данных, когда каждая строка таблицы описывает характеристики какого-то объекта из множества однородных объектов (типичный пример – список учеников, их домашних телефонов и адресов). Таким образом, по значению из первого столбца такой таблицы (ключу) можно

однозначно определить значения из остальных столбцов, т. е. значение ключа ассоциируется с остальными характеристиками объекта (в случае ученика – по фамилии можно найти другую информацию).
Если в ассоциативном массиве только два столбца ("ключ" и "значение"), то такой массив называется "хэш". Такие ассоциативные массивы очень часто используются в современных информационных системах (например, пары "логин–пароль").
В области моделирования процессов и явлений часто встречаются задачи, в которых значению "ключа" соответствует несколько параметров (например, номеру химического элемента однозначно соответствует название, атомный вес, валентность, количество протонов и пр.). В таких задачах простые хэш-массивы использовать уже неудобно.
Эффективный алгоритм обработки ассоциативных массивов (поиска значений, добавления и удаления значений и ключей, сортировки и пр.) в значительной степени зависит от используемого языка программирования и определённых в этом языке типов и структур данных. Так, в языке программирования Basic ассоциативный массив образуется из нескольких согласованных одномерных массивов. В языке программирования Pascal для представления ассоциативных массивов используется структура данных "запись" (record). В Python для ассоциативных массивов определена специальная структура данных – словарь, но мы рассмотрим работу с ассоциативными массивами с помощью списков и функций работы со списками.
Рассмотрим задачу из области экономического анализа.
Оценить экономическую деятельность нескольких предприятий. Известны названия предприятий, значения планового объёма розничного товарооборота и значения фактического объёма розничного товарооборота.
Требуется определить:

1. процент выполнения плана каждым предприятием;
2. количество предприятий, недовыполнивших план;
3. наибольший плановый товарооборот;
4. упорядочить предприятия по возрастанию планового товарооборота.

Обозначим количество предприятий как k и сформируем три списка – список названий предприятий (пусть он называется name), список значений планового товарооборота (назовём его plan), список значений фактического товарооборота (с именем fact). На основании этих данных создадим список значений процентов выполнения плана (пусть он называется procent).
Количество предприятий, недовыполнивших план, будем определять в результате сравнения процента выполнения со 100 процентами в цикле по всем предприятиям.
Текст программы на Python может выглядеть, как показано ниже.

```
# -*- coding: utf-8 -*-
#
# k - количество предприятий
# name - список названий предприятий
# plan - список значений планового товарооборота
# fact - список значений фактического товарооборота
# procent - список значений % выполнения плана
#
k=input("Количество предприятий: ")
name=[]
plan=[]
fact=[]
#
```

```
for i in range(k):
    n=raw_input("Название: ")
    name.append(n)
    p1=input("План: ")
    plan.append(p1)
    p2=input("Факт: ")
    fact.append(p2)
#
procent=map(lambda x, y:x*100/y, fact, plan)
fakt y=zip(name, procent)
plan y=zip(plan, name)
plany.sort()
print 16* "="
print "Процент выполнения плана каждым предприятием:"
#
nedo=0
for i in range(k):
    s1=fakt y[i][0]
    s2=fakt y[i][1]
    if s2 < 100:
        nedo=nedo+1
    #
    print s1,": ", s2
print "Количество предприятий, недовыполнивших план: ", nedo
print "Наибольший плановый товарооборот: ", max(plan)
#
print "Предприятия по возрастанию плана:"
for i in range(k):
    s1=plan y[i][1]
    s2=plan y[i][0]
    print s1, ": ", s2
```

Здесь с помощью функции map() и "одноразовой" lambda-функции создаётся список процентов выполнения плана и с помощью функции zip() формируется два итоговых ассоциативных массива. Сортировка таких ассоциативных массивов производится "по первому столбику", поэтому важен порядок аргументов в функции zip(), а также порядок индексов при выводе результатов.

Пример решения задачи показан на рис. 2.16.

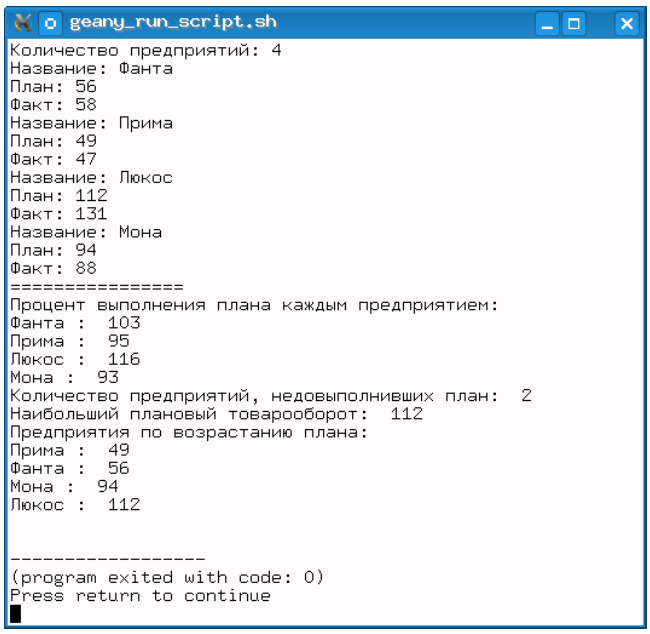


Рис. 2.16. Пример решения задачи с ассоциативным массивом

2.5.1 Задачи для самостоятельного решения

1. Используя данные таблицы отсортировать блюда по возрастанию цены. Вывести отсортированный вариант списка блюд.

Блюдо	Цена
Борщ	35
Котлета	40
Каша	20
Чай	3

2. Имеется список учеников и результаты трёх тестов (баллы от 0 до 100). Определить средний балл каждого ученика по трём тестам, вывести список учеников по убыванию среднего балла.
3. Известны данные о количестве мальчиков и девочек в нескольких классах. Отсортировать названия классов по возрастанию процента мальчиков, определить количество классов, в которых мальчиков больше, чем девочек, и вывести названия этих классов отдельно.
4. Решить задачу, связанную с оценкой экономической деятельности группы предприятий на основе известных данных:
   - название предприятий;
   - плановый объем розничного товарооборота;
   - фактический объем розничного товарооборота.

   Требуется определить:
   4. процент выполнения плана каждым предприятием;
   5. количество предприятий, недовыполнивших план на 10% и более;
   6. наименьший плановый товарооборот;
   7. упорядочить предприятия по убыванию планового товарооборота

Лабораторная работа № 4. Графика в Python и задачи моделирования

Цель: Python может работать с несколькими графическими библиотеками, обеспечивая создание сложных приложений с развитым графическим пользовательским интерфейсом. В этой части мы научимся пользоваться самыми простыми графическими возможностями Python – управлять исполнителем "черепашка" для создания графических примитивов и перемещаться на плоскости и использовать методы модуля Tkinter для задач моделирования математических функций и физических явлений.

Требования к отчету: Выполненные задания с результатами вывода оформить в виде отчета со скриншотами и прикрепить здесь. На каждом скриншоте в коде решения задачи прописать ФИО автора решения задачи и группу, в которой учится слушатель.

Управление исполнителем "черепашка"

Исполнитель "черепашка" управляется командами относительных ("вперёд- назад" и "направо-налево") и абсолютных ("перейти в точку с координатами...") перемещений. Исполнитель представляет собой "перо", оставляющее след на плоскости рисования. Перо можно поднять, тогда при перемещении след оставаться не будет. Кроме того, для пера можно установить толщину и цвет. Все эти функции исполнителя обеспечиваются модулем turtle ("черепаха").

Приведённый ниже код создаёт графическое окно (рис. 3.1) и помещает перо ("черепашку") в исходное положение.

```
# -*- coding: utf-8 -*-
  import turtle
# Инициализация
turtle.reset()
# Здесь могут быть вычисления и команды рисования
# turtle._root.mainloop() # для Python 2.4.x и 2.5.x
# Эта команда показывает окно, пока его не закроют
turtle.mainloop() # для Python 2.6.x
```

Как видно из этого примера, способ вызова метода mainloop() для показа графического окна зависит от версии Python. Поэтому если не работает один вариант, смело используйте другой. В остальных примерах будет использоваться вариант для Python 2.6.

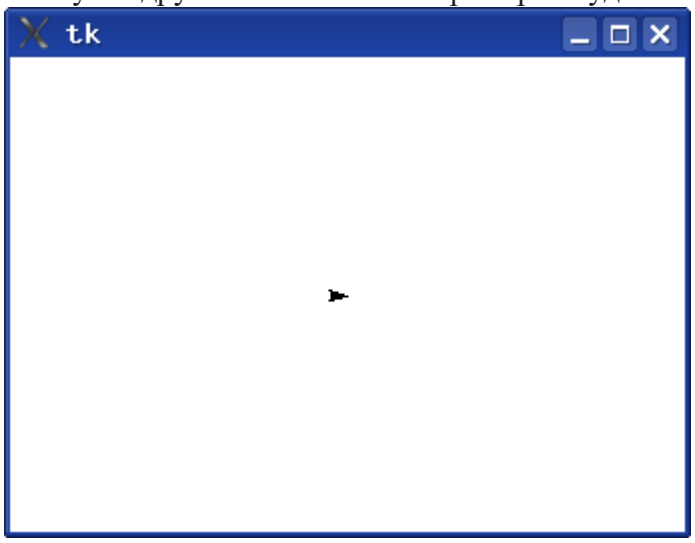


Рис. 3.1. Окно рисования модуля turtle

Полученное окно имеет фиксированный размер, который зависит от версии Python, перо позиционируется в центре. Идея рисования заключается в перемещении пера ("черепашки") в точки окна рисования с указанными координатами или в указанных направлениях на заданные расстояния, а также в проведении отрезков прямых, дуг и окружностей.
Текущее направление перемещение пера (соответствующее направлению "вперёд") указывается остриём стрелки изображения "черепашки".
Полный список команд управления "черепашкой" (и, соответственно, рисования), а также функций, обеспечиваемых модулем, можно получить, набрав в окне выполнения любой системы программирования на Python команду help('turtle').
Список этот довольно длинный, а среди предоставляемых функций имеются также математические, поскольку они могут быть востребованы при вычислении параметров отрезков, дуг и окружностей.
Команды, обеспечивающие рисование, приведены ниже.

Команда	Назначение	Пример
up()	Поднятие "пера", чтобы не оставалось следа его при перемещении	turtle.up()
down()	Опускание "пера", чтобы при перемещении оставался след (рисовались линии)	turtle.down()
goto(x,y)	Перемещение "пера" в точку с координатами x, y в системе координат окна рисования	turtle.goto(50,20)
color ('цвет')	Установка цвета "пера" в значение, определяемое строкой цвета	turtle.color('blue') turtle.color('#0000ff')
width(n)	Установка толщины "пера" в точках экрана	turtle.width(3)
forward(n)	Передвижение "вперёд" (в направлении острия стрелки, см. рис. 3.1) на n точек	turtle.forward(100)
backward(n)	Передвижение "назад" на n точек	turtle.backward(100)
right(k)	Поворот направо (по часовой стрелке) на k единиц	turtle.right(75)
left(k)	Поворот налево (против часовой стрелки) на k единиц	turtle.left(45)
radians ()	Установка единиц измерения углов в радианы	turtle.radians()
degrees ()	Установка единиц измерения углов в градусы (включён по умолчанию)	turtle.degrees()
circle(r)	**Рисование окружности радиусом $\|r\|$ точек из текущей позиции "пера". Если r положительно, окружность рисуется против часовой стрелки, если отрицательно – по часовой стрелке.**	turtle.circle(40) turtle.circle(-50)
circle(r,k)	Рисование дуги радиусом $\|r\|$ точек и углом k единиц. Вариант команды circle()	turtle.circle(40,45) turtle.circle(-50,275)
fill (flag)	В зависимости от значения flag включается (flag=1) и выключается (flag=0) режим закрашивания областей. По умолчанию выключен.	Круг: turtle.fill(1) turtle.circle(-50) turtle.fill(0)
write	Вывод текста в текущей позиции пера	turtle.write('Начало

('строка')		координат!')
tracer(flag)	Включение (flag=1) и выключение (flag=0) режима отображения указателя "пера" ("черепашки"). По умолчанию включён.	turtle.tracer(0)
clear()	Очистка области рисования	turtle.clear(0)

При выключенном режиме отображения указателя "черепашки" рисование происходит значительно быстрее, чем при включённом.

Нужно заметить, что хотя углы поворота исполнителя изначально интерпретируются в градусах, при использовании тригонометрических функций модуля turtle (например, turtle.sin()) аргументы этих функций воспринимаются как радианы.

Проделаем упражнение с целью определить систему координат окна рисования.

Приведённый ниже код формирует картинку, показанную на рис. 3.2.

```
# -*- coding: utf-8 -*-
 import turtle
#
turtle.reset()
turtle.tracer(0)
turtle.color('#0000ff')
#
turtle.write('0,0')
#
turtle.up()
x=-170
y=-120
coords=str(x)+","+str(y)
turtle.goto(x, y)
turtle.write(coords)
#
x=130
y=100
coords=str(x)+","+str(y)
turtle.goto(x, y)
turtle.write(coords)
#
x=0
y=-100
coords=str(x)+","+str(y)
turtle.goto(x, y)
turtle.write(coords)
#
turtle.down()
x=0
y=100
coords=str(x)+","+str(y)
turtle.goto(x, y)
turtle.write(coords)
#
turtle.up()
```

```
x=-150
y=0
coords=str(x)+","+str(y)
turtle.goto(x, y)
turtle.write(coords)
#
turtle.down()
x=150
y=0
coords=str(x)+","+str(y)
turtle.goto(x, y)
turtle.write(coords)
#
turtle.mainloop()
```

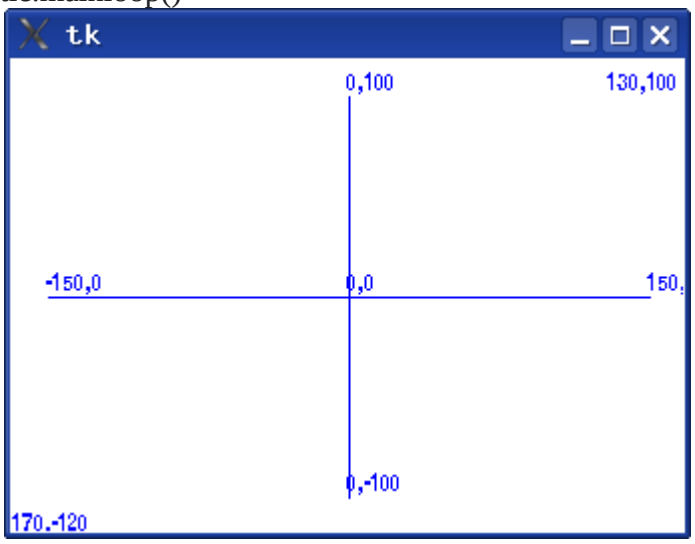


Рис. 3.2. Система координат окна рисования

Здесь строка с координатами формируется "в лоб", путём конкатенации преобразованных в строки значений координат.

Картинка, показанная на рис. 3.3, сформирована нижеследующим кодом.

```
# -*- coding: utf-8 -*-
  import turtle
#
turtle.reset()
turtle.tracer(0)
turtle.width(2)
#
turtle.up()
x=0
y=-100
turtle.goto(x, y)
turtle.fill(1)
turtle.color('#ffaa00')
turtle.down()
turtle.circle(100)
turtle.fill(0)
```

```
turtle.color('black')
turtle.circle(100)
turtle.up()
#
x=-45
y=50
turtle.goto(x, y)
turtle.down()
turtle.color('#0000aa')
turtle.fill(1)
turtle.circle(7)
turtle.up()
turtle.fill(0)
#
x=45
y=50
turtle.goto(x, y)
turtle.down()
turtle.color('#0000aa')
turtle.fill(1)
turtle.circle(7)
turtle.up()
turtle.fill(0)
#
x=-55
y=-50
turtle.goto(x, y)
turtle.right(4 5)
turtle.width(3)
turtle.down()
turtle.color('#aa0000')
turtle.circle(80, 90)
turtle.up()
#
turtle.right(135)
x=0
y=50
turtle.goto(x, y)
turtle.width(2)
turtle.color('black')
turtle.down()
turtle.forward(100)
#
turtle.mainloop()
```

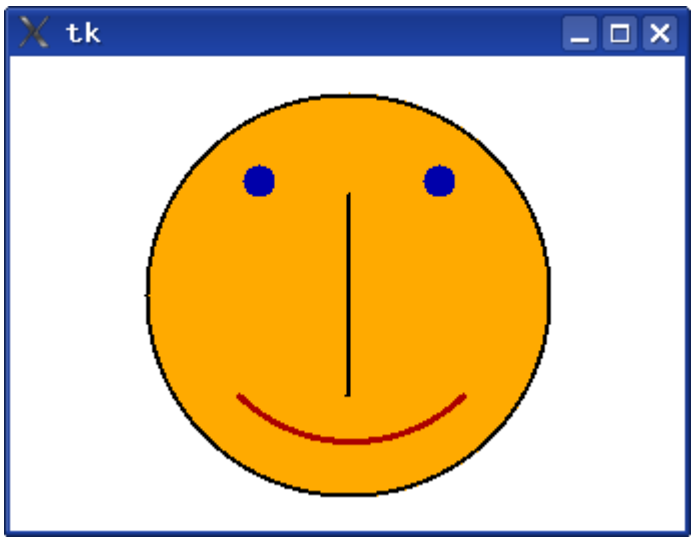


Рис. 3.3. Пример формирования изображения

Для того, чтобы изобразить улыбку, потребовалось после перемещения пера в начальную точку дуги (левую) повернуть перо на 45 градусов. Дело в том, что изначально направлением "вперёд" для пера является направление вправо (как показано на рис. 3.1). Окружности и дуги рисуются как касательные к этому "вектору", начинаясь в точке с текущими координатами пера. Поэтому для улыбки потребовалось изменить направление "вектора".

Далее, перо, первоначально сориентированное на 45 градусов вправо, после прохождения дуги в 90 градусов соответственно изменило своё направление. Поэтому для получения вертикальной линии его пришлось дополнительно повернуть.

Можно поэкспериментировать с рисованием домиков, солнышка и более сложных композиций. Однако для формирования сложных кривых (например, графиков функций) с помощью этого модуля придётся многократно выполнять команду goto(x,y). В этом легко убедиться, попытавшись нарисовать, например, график параболы.

3.1.1 Задания и упражнения

1. Как в примерах кода, формирующего изображения на рис. 3.2 и рис. 3.3, применить кортежи?
2. Напишите код для создания изображения "домика" (квадрат под треугольником) без подъёма пера при условии однократного перемещения по каждой линии.
3. Рассчитайте координаты и напишите код для создания изображения "солнца" (круг и расходящиеся от него отрезки) так, чтобы "лучи" начинались на расстоянии 2 точки от круга (не менее 8-ми лучей).
4. Напишите код для построения графика степенной функции ($y = ax^b$) с началом координат в левой нижней четверти окна рисования так, чтобы кривая проходила практически через всё окно.

Лабораторная работа № 5. Работа с библиотекой Pandas

Цель: Изучить работу в Google Colab. Изучить работу библиотеки Pandas. Работа с данными в Pandas

Ссылка на Google Colab + инструкция по работе

Нажмите на ссылку https://colab.research.google.com/notebooks/welcome.ipynb#scrollTo=GJBs_flRovLc, чтобы открыть ресурс.

Pandas

Нажмите на ссылку https://colab.research.google.com/drive/1swnmVmmCpAIA147_B6TBbGCaLx61K4Fu?usp=sharing, чтобы открыть ресурс.

Документ, который вы читаете, размещен не на статической веб-странице, а в интерактивной среде под названием **блокнот Colab**, позволяющей писать и выполнять код.
Например, вот **ячейка** с коротким скриптом Python, который позволяет рассчитать значение, выразить его в виде переменной и распечатать результат:

[ ]

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

Чтобы выполнить код в ячейке выше, выберите ее, а затем нажмите кнопку воспроизведения слева от кода или используйте сочетание клавиш Cmd/Ctrl + Ввод.
Чтобы изменить код, достаточно нажать на ячейку.
Переменные, заданные в одной ячейке, можно будет использовать в других ячейках:

[ ]

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Благодаря блокнотам Colab вы можете использовать в одном документе **исполняемый код**, **форматированный текст**, **изображения**, **разметку HTML**, **набор LaTeX** и не только. Блокноты Colab будут храниться на вашем Google Диске. Вы сможете открыть к ним доступ коллегам или друзьям, разрешив им просматривать или даже редактировать документ, а также оставлять комментарии. Подробная информация доступна на этой странице. Чтобы создать блокнот Colab, можно воспользоваться меню "Файл" выше или перейти по этой ссылке.

---

keyboard_arrow_down

Анализ и обработка данных

Colab позволяет использовать для анализа и визуализации данных все возможности популярных библиотек Python. Например, в ячейке ниже используется библиотека **numpy** для генерации случайных данных, а также библиотека **matplotlib** для их визуализации. Чтобы изменить код, достаточно нажать на ячейку.

[ ]

```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)

data = io.BytesIO()
plt.savefig(data)
image = F"{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"""![{alt}]({image})"""))
plt.close(fig)
```

Вы можете импортировать в блокноты Colab данные из своего аккаунта Google Диска, в том числе из таблиц, а также из GitHub и многих других источников. Чтобы узнать больше об импорте данных и о том, как можно использовать Colab для их анализа и обработки, изучите ссылки в разделе Работа с данными.

---

keyboard_arrow_down

Машинное обучение

В Colab вы можете импортировать набор данных изображения, сориентировать на него классификатор изображений и оценить модель с помощью нескольких строк кода. Код в блокнотах Colab исполняется на облачных серверах Google. Это означает, что вы можете использовать аппаратное обеспечение Google, в том числе графические процессоры и TPU, независимо от мощности вашей машины. Вам нужен только браузер.

Colab активно используется в области машинного обучения, в том числе для:

- знакомства с TensorFlow;
- разработки и обучения нейронных сетей;
- экспериментов с TPU;
- распространения исследований в области ИИ;
- создания руководств.

Примеры использования блокнотов Colab в сфере машинного обучения приведены в разделе Примеры использования в машинном обучении ниже.

```
import numpy as np
import pandas as pd

data=pd.read_csv('breast-cancer-dataset.csv')

data.head()
```

```
data.info()

data.tail()

data[['Age', 'Diagnosis Result']]
```

Загрузка файла .csv в pandas DataFrame

```
!wget https://pythonru.com/downloads/pandas_tutorial_read.csv
article_read=pd.read_csv('pandas_tutorial_read.csv', sep=';',names=['event', 'country', 'user_id', 'source', 'topic'])
```

Вывод части dataframe

```
article_read.head()
article_read.tail()
article_read.sample(5)
```

Вывод определенных колонок из dataframe

```
article_read[['country', 'user_id']]
```

Вывод одной колонки в виде списка

```
article_read.user_id
```

Получение сведений о датафрейме

```
article_read.info()
```

подсчет количества различных значений в столбце

```
article_read.country.value_counts()
```

Удаление заданных столбцов

```
article1=article_read.drop(['my_datetime','event'], axis=1)
article1.head()
```

из исходного датафрейма они не удалились (это можно было бы сделать, используя параметр inplace)

```
article_read.head()
```

Фильтрация определенных значений в dataframe

```
article_read[article_read.source == 'SEO']
```

Использование нескольких функций

```
article_read[['country', 'user_id']].head()
article_read[article_read.country == 'country_2'][['user_id','topic', 'country']].head()
```

Агрегация данных pandas №1: .count()

```
article_read.count()
```

Агрегация данных pandas №2: .sum()

```
article_read.sum()
article_read.user_id.sum()

Агрегация данных pandas №3 и №4: .min() и .max()
article_read.user_id.min()
article_read.topic.max()

Агрегация данных pandas №5 и №6: .mean() и .median()
article_read.user_id.mean()
article_read.user_id.median()

Функция .groupby в действии
article_read.groupby('topic').mean()
article_read.groupby('source').count()
article_read.groupby('source').count()[['user_id']]

Какие самые популярные источник и страна для пользователей country_2?
article_read[article_read.country == 'country_2'].groupby(['source',
'topic']).count()[['user_id']]
```

Лабораторная работа № 6. Работа с библиотекой Seaborn

Seaborn – популярная библиотека готовых шаблонов для статистической визуализации, написанная на бэкенде matplotlib. Две основные причины не проходить мимо:

1. Выразительный высокоуровневый интерфейс: построение большинства простых графиков происходит в одну строчку кода.
2. Более эстетичные графики: часто встроенные в seaborn стили достаточно хороши и без вашего вмешательства.

Изучите и рассмотрите пример на визуализацию данных.

Пример на визуализацию данных

Нажмите на ссылку https://colab.research.google.com/drive/135CEW5vdi1vwviymT2aVaigNf7IMvtK3?usp=sharing, чтобы открыть ресурс.

Библиотека Seaborn

[Seaborn](https://seaborn.pydata.org/) – популярная библиотека готовых шаблонов для статистической визуализации, написанная на бэкенде `matplotlib`. Две основные причины не проходить мимо:

1. Выразительный высокоуровневый интерфейс: построение большинства простых графиков происходит в одну строчку кода.

2. Более эстетичные графики: часто встроенные в `seaborn` стили достаточно хороши и без вашего вмешательства.

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

x = np.linspace(-1, 0.5, 100)
y = np.arctan(x)+x**2
plt.plot(x, y)

x=4
y=5
x*y
```

Начнём с простого: рассмотрим отрисовку синуса при помощи matplotlib и улучшим её с помощью seaborn.

```
plt.figure(figsize=(10, 3))
sns.set(style='whitegrid')
x = np.linspace(-6, 6, 100)
plt.plot(x, np.sin(x))
```

Ирисы Фишера

Теперь более интересный пример. Загрузим датасет Ирисы Фишера – [классический учебный датасет](https://en.wikipedia.org/wiki/Iris_flower_data_set), который встроен в seaborn. Числовые столбцы отвечают за длину и ширину наружной и внутренней доли

околоцветника для трех сортов ириса: setosa, virginica, versicolor.

```
iris = sns.load_dataset('iris')
iris.sample(5)
```

displot

displot показывает гистограмму распределения.

```
sns.displot(iris['sepal_length']);
```

kdeplot

kdeplot - график плотности распределения

```
sns.kdeplot(iris['petal_width'])
```

Линейный график – **Line Plot**

воспользуемся функцией lineplot с набором данных и столбцами, представляющими оси x и y.

```
plt.figure(figsize=(8, 6))
sns.lineplot(data=iris, x="sepal_length", y="petal_length")
```

PairGrid

Сетка графиков для визуализации попарных отношений в данных.

`class sns.PairGrid(data, hue=None, hue_order=None, palette=None, hue_kws=None, vars=None, x_vars=None, y_vars=None, diag_sharey=True, size=2.5, aspect=1, despine=True, dropna=True)`

* `data` — данные;
* `hue` — категории, которые будут закрашиваться в разные цвета;
* `palette` — цветовая схема, может быть задана в виде словаря цветов;
* `height` — высота каждой грани (в дюймах).

Возвращает объект, у которого доступны перечисленные ниже функции. В эти функции нужно передать функцию `func`, с помощью которой будет построен график по паре переменных (или по одной на диагонали), а так же параметры этой функции.

Визуализируем данные об ирисах Фишера.
* на диагонали расположим одномерные ядерные оценки плотности;
* под диагональю — двумерные;
* над диагональю изобразим сами точки.

```
g = sns.PairGrid(iris, hue="species")
g.map_upper(sns.histplot,bins=40)
```

```
g.map_lower(sns.kdeplot, bw_adjust=0.7)
g.map_diag(sns.histplot, kde=True);
```

Heatmap

Визуализирует двумерную таблицу в виде тепловой карты.

sns.heatmap(data, vmin=None, vmax=None, cmap=None, center=None, robust=False, annot=None, fmt='.2g', annot_kws=None, linewidths=0, linecolor='white', cbar=True, cbar_kws=None, cbar_ax=None, square=False, xticklabels='auto', yticklabels='auto', mask=None, ax=None, **kwargs)

data – 2D-данные;

vmin и vmax – минимальное и максимальное значения цветов;

cmap – цветовая схема;

annot – в какие ячейки записывать данные;

fmt – формат записи данных

linewidths – ширина линий между ячейками;

linecolor – цвет линий между ячейками;

cbar – рисовать ли colorbar.

Типичное применение – визуализация корреляции между признаками.

```
data=iris.drop('species', axis=1)

data.corr()

plt.figure(figsize=(7, 5))
sns.heatmap(data.corr(), annot=True,  cmap="YlGnBu")
```

3d графики с matplotlib

```
d={'setosa':0,'virginica':1, 'versicolor':2}
iris['species']=iris['species'].map(d)
iris.sample(5)


fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')
ax.scatter(iris['petal_length'], iris['sepal_length'], iris['sepal_width'], c=iris['species'],   s=50, cmap="viridis")
```

Визуализация категориальных данных

sns.countplot(x='species', data=iris);

sns.barplot(x='species', y='petal_length', data=iris);

sns.lineplot(data=data, palette="husl", linewidth=2)

data.plot()

Лабораторная работа № 7. Предсказание цен на недвижимость

Предсказание цен на недвижимость

Бизнес-постановка задачи

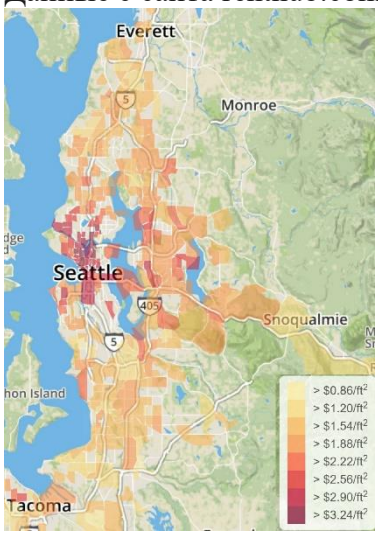
Компании по продаже недвижимости оценивают ее стоимость, используя методы машинного обучения. Задача оценки цены на недвижимость также может быть нужна для:

- выявления аномально низких цен на недвижимость,
- оценки макроэкономических показателей,
- уточнения ставок по ипотеке

Постановка задачи анализа данных

Целью данной задачи является прогнозирование стоимости домов в округе Кинг (штат Вашингтон, США) с помощью построения регрессионных моделей и их анализа. Набор данных состоит из цен на дома в округе Кинг, проданных в период с мая 2014 года по май 2015 года. Данные опубликованы в открытом доступе на платформе Kaggle.

Данные с сайта renthub.com по стоимости квартир для аренды в Сиэтле:



Обзор доступных данных

В выборке 15129 наблюдений и 16 переменных. Таким образом, про каждый из 15129 объектов недвижимости мы знаем значения 16 их характеристик (число спален, оценка состояния риелтором, наличие вида на воду и т.п.)

Данные содержат два типа переменных:

- Целевая: **Целевая. Цена**
- Остальные переменные: **15 переменных, могут использоваться для прогноза целевой переменной.**

План анализа данных (data mining):

1. Загрузить данные для обучения
2. Обработать данные перед обучением модели
3. Обучить модель на обучающей выборке
4. Загрузить и предобработать данные для тестирования
5. Провалидировать модель на тестовой выборке

keyboard_arrow_down

1. Загрузить данные для обучения

Шаг 1.1. Загружаем библиотеки

Для корректной работы с данными в python требуется загрузить специальную библиотеку **pandas**, программную библиотеку на языке python для обработки и анализа данных.

[ ]

```
import pandas as pd # загружаем библиотеку и для простоты обращения в коде называем е
ё сокращенно pd
```

Для корректной работы с графиками в python требуется загрузить специальную библиотеку **matplotlib**, программную библиотеку на языке python для визуализации данных двумерной и трехмерной графикой.
Графики используются для облегчения интерпретации полученных результатов, а также в качестве иллюстраций в презентациях и отчетах.
Оснвные методы для построения:

- plot() - графики
- semilogy() - график логарифметический
- hist() - гистограммы

[ ]

```
import matplotlib.pyplot as plt # загружаем библиотеку и для простоты обращения в коде н
азываем её сокращенно plt
# указываем, чтобы картинки отображались прямо в ноутбуке
%matplotlib inline
```

Шаг 1.2. Загрузим данные
Для решения задачи мы будем использовать данные. Загружаем данные по ссылке из интернета с помощие команды !wget. Для того, чтобы игнорировать сообщения в процессе загрузки используем команду %%capture в первой строке.

[ ]

```
%%capture
!wget https://www.dropbox.com/s/afwb0tnqm9izxha/predict_house_price_training_data.xlsx
```

Так как данные в формате xlsx (Excel), мы будем использовать специальную функцию из библиотеки pandas для загрузки таких данных **read_excel**.
В функции передаем один атрибут: название таблицы с данными.

[ ]

```
data = pd.read_excel('predict_house_price_training_data.xlsx') # загружаем таблицу в переме
нную data
```

Что важно посмотреть после того, как мы загрузили данные?

- проверить, что данные действительно загрузились
- посмотреть на данные, чтобы удостовериться, что они правильные: колонки имеют те же названия, что и в таблице и т.д.

Для того чтобы это сделать, нужно вызвать от переменной *data* метод **head()**, который выводит первые 5 строк таблицы.
Для вызова метода объекта необходимо сначала написать *имя объекта*, затем поставить *точку*, затем уже написать *название метода*. Обратите внимание, что в конце обязательно ставить скобочки, потому что метод - это функция и в ней есть аргументы, просто в данном случае мы их не передаем, поэтому оставляем поле пустым

[ ]

```
data.head()
```

Шаг 1.3. Посмотрим на размеры загруженной таблицы, у которой мы видели только первые 5 строк.
Для этого вызываем поле **shape** у нашей переменной *data*. Поле вызывается также как метод, но в конце скобки не ставятся, так как для поля не предусмотрена передача аргументов.

[ ]

```
data.shape
```

(15129, 16)

Что означает первое и второе число?
Итак, таблица содержит 15129 строк (объектов) и 16 столбцов (признаков), включая выходной (целевой) признак.
Таблицу проверили, теперь можно приступать к обработке данных.
keyboard_arrow_down
2. Обработать данные перед обучением модели
Шаг 2.1. Проверяем данные на наличие пропусков и типов переменных
Начнем с проверки общей информации о данных. Для того чтобы это сделать, нужно обратиться вызвать у переменной *data* метод **info()**.
Напомним, что в конце необходимо поставить скобочки.
[ ]

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15129 entries, 0 to 15128
Data columns (total 16 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Целевая.Цена        15129 non-null  int64  
 1   Спальни             15129 non-null  int64  
 2   Ванные              15129 non-null  float64
 3   Жилая площадь       15129 non-null  int64  
 4   Общая площадь       15129 non-null  int64  
 5   Количество этажей   15129 non-null  float64
 6   Вид на воду         15129 non-null  int64  
 7   Просмотрены ранее   15129 non-null  int64  
 8   Состояние           15129 non-null  int64  
 9   Оценка риелтора     15129 non-null  int64  
 10  Площадь без подвала 15129 non-null  int64  
 11  Площадь подвала     15129 non-null  int64  
 12  Год постройки       15129 non-null  int64  
 13  Год реновации       15129 non-null  int64  
 14  Широта              15129 non-null  float64
 15  Долгота             15129 non-null  float64
dtypes: float64(4), int64(12)
memory usage: 1.8 MB
```

Анализируем результата выполнения команды:
- 15129 строк (entries)
- 16 столбцов (Data columns)

В данных присутствует всего два типа dtypes:
- int64 - целое число (12 столбцов)
- float64 - дробное число (4 столбца)

Цифры в каждой строчке обозначают количество заполненных (*non-null*) значений. Так как эти цифры в каждой строчке совпадают с числом строк (15129), то в данных нет пропусков и можно двигаться дальше.
Шаг 2.2. Работаем с целевой переменной
Какая переменная целевая?
В данном случае по условию задачи мы должны прогнозировать стоимость, поэтому целевая переменная - это цена.
[ ]

```
y = data['Целевая.Цена']
```
Посмотрим на среднюю стоимость дома

[ ]
```
y.mean()
```
538795.2958556415

Зададим входы модели

[ ]
```
X = data.drop('Целевая.Цена', axis=1)
```
Разбиваем исходные данные на обучающую и тестовую выборку. Доли общей выборки для обучения и тестирования обычно 80% и 20% соотвественно. Любые разумные числа подходят, если для обучения используется достаточно много данных (обычно больше 50%), но и для тестирования что-то остается (10% и больше).

[ ]
```
from sklearn.model_selection import train_test_split
# Разбиение набора данных на обучающую и тестовую части в соотношении 4:1.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=21)
```
Так как данные измерены в разных шкалах, выполним нормировку данных, переведя все входные признаки в один диапазон

[ ]
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```
---

keyboard_arrow_down

3. Обучить модель на обучающей выборке

[ ]

subdirectory_arrow_rightСкрыто 9 ячеек.

---

keyboard_arrow_down

5. Провалидировать модель на тестовой выборке

Получим прогноз целевой переменной на тестовых данных для модели линейной регрессии .

Для этого вызовем метод **predict()**, в качестве аргумента передадим *X_test*.

[ ]
```
test_predictions = linear_regression_model.predict(X_test)
```

Качество регрессионных моделей оценим двумя способами:

1. Сравним визуально прогнозы с настоящими ценами (тестовые с предсказанием)
2. Сравним метрики качества

Визуализируем прогноз линейной модели и настоящие значения из тестовой выборки.

[ ]
```
%matplotlib inline
```

[ ]

```
plt.figure(figsize=(5, 5))
plt.scatter(y_test, test_predictions) # рисуем точки, соответствущие парам настоящее значен
ие - прогноз
plt.plot([0,  5000000], [0, 5000000]) # рисуем прямую, на которой предсказания и настоящ
ие значения совпадают
plt.xlabel('Настоящая цена', fontsize=15)
plt.ylabel('Предсказанная цена', fontsize=15);
```

Видим, что предасказанные цены не совсем совпадают с настоящими

Для корректного подсчета метрик качества модели в python требуется загрузить их из библиотеки **sklearn**.
Мы используем три метрики качества:

- *R^2*- точность модели (коэффициент детерминации)- чем ближе к 1, тем лучше
- *mean_absolute_error* - средняя абсолютная ошибка |yi−y^i| - чем меньше, тем лучше
- *mean_squared_error* - средняя квадратичная ошибка (yi−y^i)2 - чем меньше, тем лучше

Найдем сначала точность

[ ]

```
r2=linear_regression_model.score(X_test,y_test)
print(r2)
```

0.692424990673975

Считается, что точность модели хорошая, если R^2 больше, чем 0.8. В данном случае точность модели получилась не очень высокая.

[ ]

```
from sklearn.metrics import mean_absolute_error, mean_squared_error
```

Подсчитаем теперь ошибки для линейной модели.
Для этого вызовем методы **mean_absolute_error()** и **mean_squared_error()**. На вход им передается столбец настоящих значений *y_test* и столбец значений, предсказанных моделью линейной регрессии *test_predictions*.

[ ]

```
mae_linear_model = mean_absolute_error(y_test, test_predictions)
mse_linear_model = mean_squared_error(y_test, test_predictions)
```

Теперь напечатаем полученные ошибки. Обычно смотрят на корень из среднеквадратичной ошибки, RMSE.

[ ]

```
print('MAE:', mae_linear_model,'RMSE:',   mse_linear_model**0.5)
```

MAE: 124458.83371356507 RMSE: 192887.04627541732

У нас получилось, что предсказывая цену дома, наша модель в среднем ошибается на 124 458 долларов (при средней стоимости 538 795 долларов) - что, конечно, много

Исследуем, какие признаки сильнее всего влияют на цену дома

[ ]
```
yy=linear_regression_model.coef_
```

[ ]
```
plt.figure(figsize=(10, 4))
plt.ylabel('Важность (полезность)')
plt.xlabel('Название признака')
plt.bar(X.columns,yy, color='orange')
plt.xticks(rotation=90, fontsize=8)
plt.show()
```

Чем больше (по модулю) столбик на графике, тем сильнее признак влияет на цену. Если важность положительная, то, чем больше значение признака, тем выше цена. Если отрицательная, то, чем больше значение признака, тем меньше цена

Обзор результатов
В этом ноутбуке мы научились

1. Загружать библиотеки, необходимые для работы.
2. Загрузить данные для обучения, представленные в формате excel таблицы.
3. Проводить предрватилельную обработку данных перед построением и использованием модели машинного обучения: смотреть на части таблицы, понимать, какой размер у выборки данных, выделять отдельные столбцы таблицы в новые таблицы.
4. Обучать модель линейной регрессии на обучающей выборке.
5. Валидировать модель на тестовой выборке с помощью кросс-плота для модели и реальных значений, стандартных ошибок модели.

Лабораторная работа № 8. Эффективное использование Matplotlib

Введение

В этой практической работе будет показано, как используют *matplotlib*, и предоставлены некоторые рекомендации для начинающих пользователей. *matplotlib* является неотъемлемой частью стека науки о данных *Python*, и надеюсь, что эта работа поможет вам понять, как использовать его для собственных визуализаций.

---

keyboard_arrow_down

Откуда негатив по отношению к matplotlib?

На наш взгляд, есть несколько причин, по которым сложно изучить *matplotlib*.

Во-первых, у *matplotlib* два интерфейса. Первый основан на *MATLAB* и использует интерфейс на основе состояний. Второй вариант - это *объектно-ориентированный интерфейс*. Причины этого выходят за рамки публикации, но знание того, что есть два подхода, жизненно важно при построении графика с помощью *matplotlib*.

Причина, по которой два интерфейса вызывают путаницу, заключается в том, что в мире *stack overflow* и информации, доступной через гугл, новые пользователи находят несколько похожих решений.

Могу сказать из собственного опыта: оглядываясь назад на часть моего старого кода, существует мешанина из кода *matplotlib*, которая сбивает с толку (даже если я сам ее написал).

Новые пользователи *matplotlib* должны изучить и использовать объектно-ориентированный интерфейс.

Еще одна историческая проблема с *matplotlib* заключается в том, что некоторые стили по умолчанию были довольно непривлекательными. В мире, где *R* мог генерировать несколько действительно крутых графиков с помощью *ggplot*, параметры *matplotlib* выглядели бледно. Хорошая новость заключается в том, что *matplotlib 3.3* имеет гораздо более приятные возможности.

Третья проблема, которую я вижу, заключается в том, что существует путаница относительно того, когда вы должны использовать чистый *matplotlib*, по сравнению с такими инструментами, как *pandas* или *seaborn*, которые построены поверх *matplotlib*.

Зачем использовать matplotlib?

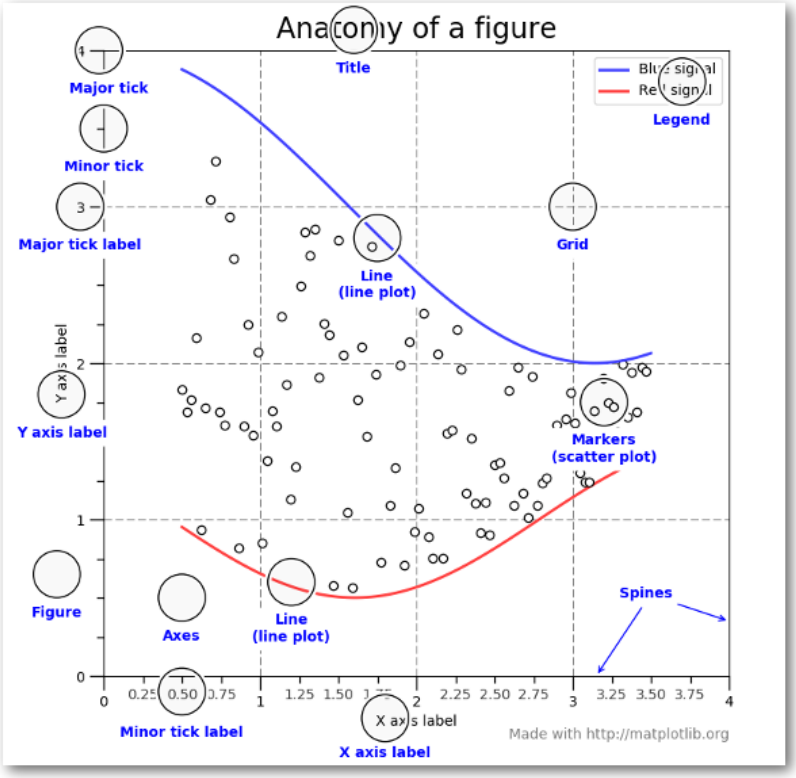
Несмотря на некоторые из этих проблем *matplotlib* чрезвычайно мощный инструмент. Библиотека позволяет создавать практически любую визуализацию, которую вы только можете себе представить. Кроме того, вокруг нее построена обширная экосистема инструментов *Python*, и многие из более продвинутых инструментов визуализации используют *matplotlib* в качестве базовой библиотеки. Если вы работаете в стеке науки о данных *Python*, вам необходимо получить базовые знания о том, как использовать *matplotlib*.

Основные предпосылки

Рекомендую следующие шаги для изучения того, как использовать *matplotlib*:

1. Изучите основную терминологию *matplotlib*, в частности, что такое Figure (фигура) и Axes (оси).
2. Всегда используйте объектно-ориентированный интерфейс. Возьмите за привычку использовать его с самого начала анализа.
3. Начните свои визуализации с простых графиков (*plotting*) в *pandas*.
4. Используйте *seaborn* для более сложных статистических визуализаций.
5. Используйте *matplotlib* для настройки визуализации *pandas* или *seaborn*.

Следующий рисунок из часто задаваемых вопросов о *matplotlib* - золотой. Держите его под рукой, чтобы понимать терминологию графика (*plot*).



Большинство терминов просты, но главное помнить, что Figure - это окончательное изображение, которое может содержать 1 или более *осей* (*axes*).
Axes (оси) представляют собой отдельный график (*plot*). Как только вы поймете, что это такое и как получить к ним доступ через *объектно-ориентированный API*, остальная часть процесса станет на свои места.
Другое преимущество этих знаний состоит в том, что у вас есть отправная точка, когда вы встречаете код в сети.
Наконец, я не говорю, что вам следует избегать других хороших вариантов, таких как ggplot (aka ggpy), bokeh, plotly или altair. Я просто думаю, что для начала вам понадобится базовое понимание matplotlib + pandas + seaborn. Поняв базовый стек визуализации, вы сможете изучить другие варианты и сделать осознанный выбор в зависимости от ваших потребностей.

---

keyboard_arrow_down
Начнем
Остальная часть этой практической работы является руководством по созданию базовой визуализации в *pandas* и настройке наиболее распространенных элементов с помощью *matplotlib*.
Я сосредоточился на наиболее распространенных задачах построения графиков, с которыми я сталкиваюсь, таких как маркировка осей (*labeling axes*), настройка пределов (*limits*), обновление заголовков графиков (*plot titles*), сохранение фигур (*figures*) и корректировка легенд (*legends*).
Для начала я собираюсь настроить импорт и прочитать данные о продажах:

[ ]

[ ]

```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import FuncFormatter

%matplotlib inline
```

[ ]

```
df = pd.read_excel("https://github.com/chris1610/pbpython/blob/master/data/sample-salesv3.xlsx?raw=true")
df.head()
```

[ ]

```
top_10 = (df.groupby('name')[['ext price', 'quantity']].agg({'ext price': 'sum', 'quantity': 'count'})
         .sort_values(by='ext price', ascending=False))[:10].reset_index()
```

[ ]

```
top_10.rename(columns={'name':'Name',
                       'ext price':'Sales',
                       'quantity':'Purchases'},
              inplace=True)
```

[ ]

```
top_10
```

Теперь, когда данные отформатированы в виде простой таблицы, давайте поговорим о представлении этих результатов в виде гистограммы (*bar chart*).
Как я упоминал ранее, у *matplotlib* есть много разных стилей, доступных для отображения графиков (*plots*). Вы можете увидеть, какие из них доступны в вашей системе, используя plt.style.available:

[ ]

```
plt.style.available
['Solarize_Light2',
 '_classic_test_patch',
 '_mpl-gallery',
 '_mpl-gallery-nogrid',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn-v0_8',
 'seaborn-v0_8-bright',
 'seaborn-v0_8-colorblind',
```

```
'seaborn-v0_8-dark',
'seaborn-v0_8-dark-palette',
'seaborn-v0_8-darkgrid',
'seaborn-v0_8-deep',
'seaborn-v0_8-muted',
'seaborn-v0_8-notebook',
'seaborn-v0_8-paper',
'seaborn-v0_8-pastel',
'seaborn-v0_8-poster',
'seaborn-v0_8-talk',
'seaborn-v0_8-ticks',
'seaborn-v0_8-white',
'seaborn-v0_8-whitegrid',
'tableau-colorblind10']
```

```
[ ]
top_10.plot(kind='barh', y="Sales", x="Name")
```

Причина, по которой я рекомендую в первую очередь использовать построение (*plotting*) в *pandas*, заключается в том, что это быстрый и простой способ прототипирования визуализации.

keyboard_arrow_down

Настройка графика

Предполагая, что вы понимаете суть графика, следующим шагом будет его настройка. Некоторые настройки (например, добавление заголовков и меток) очень просты в функции *plot*. Однако в какой-то момент вам, вероятно, придется выйти за рамки этой функциональности.

Вот почему я рекомендую выработать привычку делать следующее:

```
[ ]
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
```

```
[ ]
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set_xlabel('Total Revenue')
ax.set_ylabel('Customer');
```

Вот еще один прием, который мы можем использовать для изменения заголовка и обеих меток:

```
[ ]
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue', ylabel='Customer')
```

[ ]

```
fig, ax = plt.subplots(figsize=(5, 6))
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue')
ax.legend().set_visible(False)
```

Есть много вещей, которые вы, вероятно, захотите сделать, чтобы очистить этот график. Одна из самых больших неприятностей - это форматирование чисел в Total Revenue (общего дохода).
Matplotlib может помочь нам в этом с помощью FuncFormatter. Эта универсальная функция позволяет применять пользовательскую функцию к значению и возвращать красиво отформатированную строку для размещения на оси.
Вот функция форматирования валюты для корректной обработки долларов США в диапазоне нескольких сотен тысяч:

[ ]

```
def currency(x, pos):
    'Два аргумента - это значение и позиция отметки.'
    if x >= 1000000:
        return '${:1.1f}M'.format(x*1e-6)
    return '${:1.0f}K'.format(x*1e-3)
```

[ ]

```
fig, ax = plt.subplots()
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue', ylabel='Customer')
formatter = FuncFormatter(currency)
ax.xaxis.set_major_formatter(formatter)
ax.legend().set_visible(False)
```

Это намного приятнее и демонстрирует хороший пример гибкости, позволяющей найти собственное решение проблемы.
Последняя функция настройки, которую я рассмотрю, - это возможность добавлять *аннотации* к графику. Чтобы нарисовать вертикальную линию, можно использовать ax.axvline(), а для добавления собственного текста - ax.text().
В этом примере мы нарисуем линию, показывающую среднее значение, и добавим метки, показывающие трех новых клиентов.
Вот полный код с комментариями, чтобы собрать все воедино:

[ ]

```
fig, ax = plt.subplots()

top_10.plot(kind='barh', y="Sales", x="Name", ax=ax)
```

```
avg = top_10['Sales'].mean()

ax.set_xlim([-10000, 140000])
ax.set(title='2014 Revenue', xlabel='Total Revenue', ylabel='Customer')

ax.axvline(x=avg, color='b', label='Average', linestyle='--', linewidth=1)

for cust in [3, 5, 8]:
    ax.text(115000, cust, "New Customer")

formatter = FuncFormatter(currency)
ax.xaxis.set_major_formatter(formatter)

ax.legend().set_visible(False)
```

Хотя это не самый захватывающий график, он все же показывает, сколько у вас возможностей.

keyboard_arrow_down

Фигуры и графики (Figures and Plots)

До сих пор все изменения, которые мы вносили, касались отдельного графика. К счастью, у нас есть возможность добавить несколько графиков к фигуре, а также сохранить фигуру целиком, используя различные параметры.

Если мы хотим нанести два графика на одну и ту же фигуру, то должно быть понимание того, как это сделать.

Сначала создайте фигуру, потом оси, а затем нанесите все вместе.

Можем сделать это с помощью plt.subplots():

[ ]

```
fig, (ax0, ax1) = plt.subplots(nrows=1, ncols=2, sharey=True, figsize=(7, 4))
```

[ ]

```
fig, (ax0, ax1) = plt.subplots(nrows=1,ncols=2, sharey=True, figsize=(7, 4))
top_10.plot(kind='barh', y="Sales", x="Name", ax=ax0)

ax0.set_xlim([-10000, 140000])
ax0.set(title='Revenue', xlabel='Total Revenue', ylabel='Customers')

avg = top_10['Sales'].mean()
ax0.axvline(x=avg, color='b', label='Average', linestyle='--', linewidth=1)

top_10.plot(kind='barh', y="Purchases", x="Name", ax=ax1)
avg = top_10['Purchases'].mean()

ax1.set(title='Units', xlabel='Total Units', ylabel='')
ax1.axvline(x=avg, color='b', label='Average', linestyle='--', linewidth=1)

fig.suptitle('2014 Sales Analysis', fontsize=14, fontweight='bold');
```

```
ax1.legend().set_visible(False)
ax0.legend().set_visible(False)
```

До сих пор я полагался на *jupyter блокнот* для отображения с помощью встроенной директивы %matplotlib inline.
Тем не менее, будет много случаев, когда вам понадобится сохранить фигуру в определенном формате и интегрировать ее с какой-либо другой презентацией.
Matplotlib поддерживает множество различных форматов для сохранения файлов. Вы можете использовать fig.canvas.get_supported_filetypes(), чтобы узнать, что поддерживает ваша система:

[ ]

```
fig.canvas.get_supported_filetypes()
{'eps': 'Encapsulated Postscript',
 'jpg': 'Joint Photographic Experts Group',
 'jpeg': 'Joint Photographic Experts Group',
 'pdf': 'Portable Document Format',
 'pgf': 'PGF code for LaTeX',
 'png': 'Portable Network Graphics',
 'ps': 'Postscript',
 'raw': 'Raw RGBA bitmap',
 'rgba': 'Raw RGBA bitmap',
 'svg': 'Scalable Vector Graphics',
 'svgz': 'Scalable Vector Graphics',
 'tif': 'Tagged Image File Format',
 'tiff': 'Tagged Image File Format',
 'webp': 'WebP Image Format'}
```

[ ]

```
fig.savefig('sales.png', transparent=False, dpi=80, bbox_inches="tight")
```

Эта версия сохраняет график в формате png с непрозрачным фоном. Я также указал dpi и bbox_inches="tight", чтобы убрать пустое пространство.

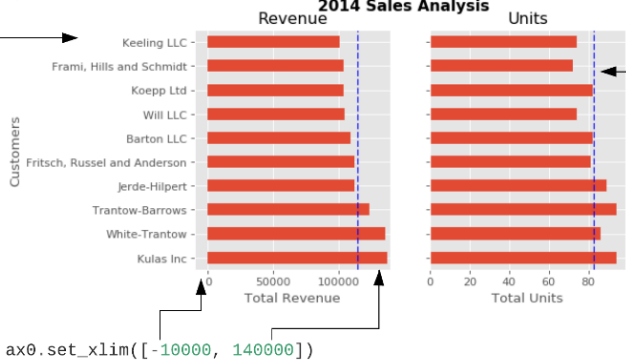
keyboard_arrow_down

Заключение

Надеюсь, этот процесс помог вам понять, как более эффективно использовать *matplotlib* в ежедневном анализе данных. Если вы привыкнете использовать этот подход при проведении анализа, вы сможете быстро узнать, как сделать все, что вам нужно, чтобы настроить график.
В качестве последнего бонуса я добавляю краткое руководство по унификации всех концепций. Я надеюсь, что это поможет объединить этот пост и окажется полезным справочником для будущего использования.

matplotlib customization example

```
fig, (ax0, ax1) = plt.subplots(nrows=1,ncols=2, sharey=True, figsize=(7, 4))

fig.suptitle('2014 Sales Analysis', fontsize=14, fontweight='bold')

top_10.plot(kind='barh', y="Sales",
            x="Name", ax=ax0)

top_10.plot(kind='barh', y="Purchases",
            x="Name", ax=ax1)

ax1.axvline(x=avg, color='b',
            label='Average',
            linestyle='--',
            linewidth=1)

ax0.set_xlim([-10000, 140000])

ax0.set(title='Revenue', xlabel='Total
        Revenue', ylabel='Customers')

ax1.set(title='Units',
        xlabel='Total Units', ylabel='')

fig.savefig('sales.png', transparent=False, dpi=80, bbox_inches="tight")
```



pbpython.com