

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)

М. С. РУСАКОВА, Е. В. РОГАЧЕВА

МОДЕЛИРОВАНИЕ ФИЗИЧЕСКИХ ПРОЦЕССОВ И ВИЗУАЛИЗАЦИЯ ДАННЫХ СРЕДСТВАМИ PYTHON

Рекомендовано редакционно-издательским советом федерального государственного автономного образовательного учреждения высшего образования «Самарский национальный исследовательский университет имени академика С. П. Королева» в качестве практикума для обучающихся по основной образовательной программе высшего образования по направлению подготовки 03.03.02 Физика

САМАРА
Издательство Самарского университета
2024

УДК 53.01(075)+004.438(075)

ББК В3с51я7+3973.22я7

Р880

Рецензенты: д-р физ.-мат. наук, профессор *В. А. Салеев*;
д-р физ.-мат. наук, профессор *А. Ф. Крутов*

Русакова, Маргарита Сергеевна

**Р880 Моделирование физических процессов и визуализация
данных средствами Python: практикум / М.С. Русакова,
Е.В. Рогачева. – Самара: Издательство Самарского
университета, 2024. – 76 с.**

ISBN 978-5-7883-2099-1

Излагаются основные методы работы с современными Python-библиотеками для организации научных вычислений, визуализации данных и построения графического интерфейса пользователя с использованием объектно-ориентированного подхода. Также практикум содержит ряд упражнений по моделированию физических процессов и математическому моделированию; упражнения могут быть предложены обучающимся специальности «Физика» как на лабораторных и практических занятиях, так и для самостоятельного выполнения.

УДК 53.01(075)+004.438(075)

ББК В3с51я7+3973.22я7

ISBN 978-5-7883-2099-1

© Самарский университет, 2024

Оглавление

Введение	5
1 Визуализация графиков средствами библиотеки matplotlib	6
1.1 Задание «Построение нескольких графиков функций»	7
2 Моделирование и визуализация процесса.....	13
2.1 Задание «Изучение случайного блуждания на плоскости» ..	13
3 Разработка класса как пользовательского типа данных и его применение.....	20
3.1 Задание «Разработка класса Polynom»	20
4 Разработка класса «Матрица» с использованием возможностей библиотеки numpy	23
4.1 Задание «Разработка класса MyMatrix»	23
5 Разработка графического интерфейса пользователя при помощи tkinter.....	30
5.1 Импортирование tkinter.....	30
5.2 Кнопки и обработчики событий.....	30
5.3 Метки	32
5.4 Текстовое поле ввода.....	33
5.5 Текстовая область	34
5.6 Радиокнопки.....	35
5.7 Флажки	36
5.8 Списки	37
5.9 Шкала	39
5.10 Скроллинг.....	40
5.11 Рисование на форме. Канва.....	41
5.12 Контрольные вопросы.....	44
5.13 Задания	44

6 Использование графической подсистемы для моделирования движущихся объектов	50
6.1 Задание «Моделирование движения объекта с учетом гравитации».....	50
7 Рисование самоподобных кривых средствами Python	53
7.1 Контрольные вопросы.....	68
7.2 Задания	69
Список использованной литературы	73

Введение

При изучении различных физических процессов немаловажную роль играет возможность наглядно представить происходящее. Как при решении задач, так и при выполнении работ лабораторного практикума чрезвычайно полезно проанализировать полученные зависимости графически, рассмотреть различные комбинации параметров.

Существует достаточно большое количество математических пакетов, способных выполнить подобный анализ, однако каждый из них де-факто снабжен собственным набором команд, что приводит к отсутствию универсальности разработанных средствами таких пакетов скриптов. Использование языка программирования общего назначения Python вкупе с набором библиотек, предназначенных для проведения научных расчетов, устраняет эту проблему, а также является более гибким решением, нежели вышеупомянутые пакеты.

Полезным дополнением могут оказаться библиотеки, реализующие работу с графической подсистемой.

Большое количество библиотек, ориентированных на разные предметные области, низкий порог входа, бесплатность и доступность транслятора и интегрированных сред разработки сделали Python популярным выбором для студентов и исследователей различных направлений.

Предлагаемый практикум ориентирован на уже знающих основы Python и владеющих основными приёмами работы в средах программирования, но не имеющих опыта работы с такими библиотеками, как *matplotlib*, *numpy*, *tkinter*. Последовательное выполнение заданий сформирует умения и навыки обработки и визуализации данных, численного и графического моделирования.

1 Визуализация графиков средствами библиотеки `matplotlib`

Библиотека `matplotlib` была разработана Джоном Хантером для визуализации данных с помощью двумерных и трехмерных графиков. Важно отметить, что библиотека поддерживает работу с форматами данных `numpy`.

Изначально Дж. Хантер подражал графическим командам `MATLAB`, и это отразилось в модуле `pylab`. Однако в настоящий момент рекомендуется пользоваться модулем `pyplot` и связанными с ним модулями. Модуль `pyplot` фактически предоставляет процедурный интерфейс к объектно-ориентированной библиотеке, скрывая от пользователя детали иерархической реализации графиков. Конечно, если возникает потребность в разработке собственного интерфейса средствами `matplotlib`, придётся обращаться к низкоуровневым возможностям библиотеки, но это выходит за рамки данного учебного пособия.

В модуле `pyplot` имеется ряд функций, которые обеспечивают рисование собственно графика, координатных осей и сетки, а также подписей к ним, легенды и иных вспомогательных элементов. Все эти функции подробно описаны в документации, которая, помимо прочего, позволяет просматривать их исходный код. В качестве примера рассмотрим одну из основных функций рисования — `plot`. Она содержит ряд параметров:

```
matplotlib.pyplot.plot(*args, scalex, scaley,  
data, **kwargs),
```

не все из которых являются обязательными. В качестве аргументов, как правило, в функцию передаются одномерные массивы, содержащие координаты точек на оси абсцисс и оси ординат. Простейшим обращением к функции будет `plot(x, y)`, в результате

которого будет отображен график, использующий настройки по умолчанию.

При необходимости можно организовать рисование нескольких графиков на одном поле — такой подход оказывается полезен в ряде случаев, в частности, когда необходимо изобразить график функции, имеющей особые точки. Такое задание и предлагается выполнить.

1.1 Задание «Построение нескольких графиков функций»

Разработайте приложение, которое строит графики указанных ниже функций. В каждом варианте два подзадания: первая функция содержит одну или несколько точек разрыва, вторая функция задается либо параметрически, либо в полярных координатах. Предусмотрите возможность задавать в интерактивном режиме значения коэффициентов функций и граничные значения x (параметров t или ϕ), возможность отображать на одном рисунке несколько графиков, поместите легенду на график. Масштаб, цвет фона и цвет графика, начертание линий и наличие/начертание маркера также может задавать пользователь. Предусмотрите возможность сохранения набора числовых значений, по которым строится график, а также настроек графика в текстовом файле, имя которого задается пользователем. Кроме того, предусмотрите возможность загрузки данных из такого текстового файла. Формат данных для заполнения такого файла разработайте самостоятельно.

Примечание 1. Рекомендуется сначала построить графики функции с помощью графического калькулятора (например, *desmos.com*), а потом для тех же значений параметров построить графики в разработанной вами программе. Кривые должны совпадать.

Примечание 2. Если график, построенный в полярных координатах в *matplotlib* выглядит иначе, чем построенный в графическом калькуляторе, рекомендуется перейти в вашей программе к параметрическому представлению графика.

Вариант 1

1. $(x + a)/(x - b)$;
2. постройте Декартов лист, уравнение которого в полярной системе координат приведено ниже:

$$\rho = \frac{3a \cos \phi \sin \phi}{\cos^3 \phi + \sin^3 \phi}.$$

Вариант 2

1. $a + b/x + c/x^2$;
2. постройте улитку Паскаля, уравнение в полярных координатах приведено ниже:

$$\rho = l - a \sin \phi.$$

Вариант 3

$$a - b/x - c/x^3;$$

1. постройте спираль Архимеда ($\rho = k\phi$ в полярных координатах).

Вариант 4

$$1. \frac{a}{bx^2 + cx};$$

2. постройте лемнискату Бернулли ($\rho^2 = 2a^2 \cos 2\phi$ в полярных координатах).

Вариант 5

$$1. \frac{a}{bx^3 + cx^2 + dx};$$

2. постройте кардиоиду ($\rho = 2r(1 + \cos \phi)$ в полярных координатах).

Вариант 6

1. $\frac{ax}{bx^2+cx+d}$;
2. постройте полярную розу ($\rho = a \sin k\phi$ в полярных координатах, a и k – положительные числа).

Вариант 7

1. $\frac{ax}{bx^3+cx^2+dx+e}$;
2. постройте в полярных координатах:
$$\rho = \exp(\sin\phi) - 2\cos(4\phi) + \sin\left(\frac{5}{24}(2\phi - \pi)\right), \phi \in [-8\pi, 8\pi].$$

Вариант 8

1. $\frac{a \cos(x-b)}{|x|}$;
2. постройте в полярных координатах:
$$\rho = 2 - 2\sin\phi + \frac{\sin\phi\sqrt{|\cos\phi|}}{\sin\phi+1.4}, \phi \in [0, 2\pi].$$

Вариант 9

1. $\frac{a}{\sin(bx+c)}$;
2. постройте дельтоиду:
$$x = 2\cos(t) + \cos(2t), y = 2\sin(t) - \sin(2t), t \in [0, 2\pi].$$

Вариант 10

1. $\tan(bx^2 + cx - d)$;
2. постройте астроиду:
$$x = 2\sin^3 t, y = 2\cos^3 t, t \in [0, 2\pi].$$

Вариант 11

1. $\frac{\exp(bx^2 - cx + d)}{x};$

2. постройте гипоциклоиду:

$$x = 20 \left(\cos t + \frac{\cos(5t)}{5} \right), y = 20 \left(\sin t - \frac{\sin(5t)}{5} \right), t \in [0, 2\pi].$$

Вариант 12

$\frac{\log(bx^2 + cx - d)}{x};$

1. постройте гипоциклоиду:

$$x = 4.4 \left(\cos t + \frac{\cos(1.1t)}{1.1} \right), y = 4.4 \left(\sin t - \frac{\sin(1.1t)}{1.1} \right), t \in [0, 20\pi].$$

Вариант 13

1. $\frac{ax}{\arctan x};$

2. постройте гипоциклоиду:

$$x = 24.8 \left(\cos t + \frac{\cos(6.2t)}{6.2} \right), y = 24.8 \left(\sin t - \frac{\sin(6.2t)}{6.2} \right), t \in [0, 10\pi].$$

Вариант 14

1. $a^{1+1/x};$

2. постройте эпициклоиду:

$$x = 6.2 \left(\cos t - \frac{\cos(3.1t)}{3.1} \right), y = 6.2 \left(\sin t - \frac{\sin(3.1t)}{3.1} \right), t \in [0, 20\pi].$$

Вариант 15

1. $\cot(bx + c);$

2. постройте фигуру Лиссажу:

$$x = \sin(3t + \pi/2), y = \sin(2t), t \in [0, 2\pi].$$

Вариант 16

1. $\frac{\sin x \cos(ax+bx^2+c)}{x}$;
2. постройте в полярных координатах:
$$\rho = 1 + \frac{1}{3}(2\cos 3\phi - \cos 6\phi).$$

Вариант 17

1. $\frac{d\cos x}{ax+bx^2+c}$;
2. постройте в полярных координатах:
$$\rho = 1 + \frac{1}{8}(\cos \phi + \cos 5\phi).$$

Вариант 18

1. $\sin x(ax^2 + bx + c)$;
2. постройте в полярных координатах:
$$\rho = 1 + \frac{1}{48}(27\cos \phi + 12\cos 3\phi + 8\cos 5\phi + \cos 7\phi).$$

Вариант 19

1. $b\cos x \exp(-a/x^2)$;
2. постройте в полярных координатах:
$$\rho = \frac{a}{\cos} \phi.$$

Вариант 20

1. $\frac{\exp(a/x)}{b\log(x^2)}$;
2. постройте в полярных координатах:
$$\rho = a \tan \phi \sin \phi.$$

Вариант 21

1. $\log \frac{ax^2 \cos(x)}{\exp(cx+d)}$;
2. постройте в полярных координатах:
 $\rho = 1 + 7\cos 5\phi + 4\sin^2 5\phi + 3\sin^4 5\phi$.

Вариант 22

1. $ax + \frac{b}{x+c} + d$;
2. постройте в полярных координатах:
 $\rho = 1 + 7\cos 8\phi + 4\sin^2 8\phi + 3\sin^4 8\phi$.

Вариант 23

1. $ax^3 + \frac{b}{x^2} + cx + d$;
2. постройте в полярных координатах:
 $\rho = (1 + \sin 9\phi)(1 + 0.03\sin 45\phi)(1 + 0.04\sin 297\phi)$.

Вариант 24

1. $ax^2 + \frac{\cos(bx)}{\sqrt{(x+c)}} + d$;
2. постройте в полярных координатах:
 $\rho = 2 - \frac{1}{2}\sin 50\phi + \cos 7\phi$.

Вариант 25

1. $\frac{\cos(bx)}{|x+a|}$;
2. постройте в полярных координатах:
 $\rho = 1 + \cos 3\phi + \sin^2 3\phi$.

2 Моделирование и визуализация процесса

Для изучения физических процессов часто используются компьютерные модели. Компьютерное моделирование позволяет получить результаты с меньшими временными и материальными затратами, чем обычный эксперимент. Разумеется, важную роль играет качество модели и её соответствие реальным данным. В настоящем разделе предлагается исследовать модели случайного блуждания на примере движения объектов на плоскости.

2.1 Задание «Изучение случайного блуждания на плоскости»

Напишите программу, которая моделирует описанный процесс. Используйте генератор псевдослучайных чисел из библиотеки *numpy*.

Вариант 1

Разработайте программу, моделирующую поведение «собаки, блуждающей в городе». Город представляет собой решетку с единичным шагом из улиц, параллельных осям координат. Изначально собака находится в точке с координатами $(0, 0)$.

Один раз в единицу времени собака выбирает направление вдоль одной из осей координат. Выбор направления осуществляется с помощью генератора случайных чисел. После выбора направления собака смещается на 1 в этом направлении и осуществляет следующий выбор.

Сохраните в файле (текстовом или собственного формата) последовательность точек, в которые попадет собака за время t .

Средствами *matplotlib* изобразите путь собаки, который она пройдет за заданное время t .

Постройте график изменения максимального удаления собаки от начальной точки в зависимости от времени.

Считайте, что на каждое перемещение собаке требуется одна единица времени. Постройте диаграмму, показывающую среднее время пребывания собаки в каждом квадранте координатной плоскости. К какому квадранту относить оси координат – выберите сами, можете отдельно подсчитать количество попаданий точек пребывания собаки на оси координат.

Посчитайте количество перемещений собаки между последовательными попаданиями в начальную точку, определите среднее значение количества таких перемещений. Изменяется ли оно со временем?

Вариант 2

Разработайте программу, моделирующее поведение «собаки, блуждающей в городе». Город представляет собой решетку с единичным шагом из улиц, параллельных осям координат. Изначально собака находится в точке с координатами (x, y) .

Один раз в единицу времени собака выбирает направление вдоль одной из осей координат. Выбор направления осуществляется с помощью генератора случайных чисел. После выбора направления собака смещается на 1 в этом направлении и осуществляет следующий выбор.

Сохраните в файле (текстовом или собственного формата) последовательность точек, в которые попадет собака за время t .

Средствами *matplotlib* изобразите путь собаки, который она пройдет за заданное время t .

Для достаточно большого t путь собаки будет иметь точки самопересечения (точки, в которые собака будет попадать

повторно). Постройте график зависимости количества самопересечений от времени.

Определим расстояние между точками (x_1, y_1) и (x_2, y_2) на решетке как $|x_1 - x_2| + |y_1 - y_2|$. Постройте диаграмму, показывающую распределение расстояний между соседними точками самопересечения.

Определим ранг точки как количество заходов собаки в эту точку. Постройте диаграмму (гистограмму) распределения точек по количеству заходов собаки в эту точку.

Вариант 3

Разработайте программу, моделирующее поведение «лягушки на лугу». Будем считать луг квадратом со стороной $2d$ на плоскости, исходно лягушка находится в точке с координатами $(0,0)$. Время считайте дискретным: все события происходят только в целые моменты времени.

В каждый очередной момент времени в произвольной точке на лугу возникает стайка комаров (в стайке не менее 1 комара и не более m). Эта стайка находится над лугом на протяжении некоторого времени (не большего k), если раньше комаров не спугнет лягушка.

Лягушка направляется к ближайшей (к ней) стайке по кратчайшему пути, но длина ее прыжка составляет не более единицы длины. Считайте, что лягушка всегда делает прыжки длиной одну единицу, быть может, за исключением последнего прыжка, когда она достигает стайки комаров. В последнем прыжке она ловит какое-то случайное количество комаров, не превышающее количества комаров в стайке. Оставшиеся комары разлетаются за пределы луга.

После каждого прыжка лягушка оценивает, какая стайка комаров из существующих над лугом в настоящий момент находится

ближе; при этом длительность пребывания стайки на лугу ей неизвестна. Лягушка всегда выбирает направление движения на ближайшую стайку. Если есть две или более стайки на одинаковом расстоянии, лягушка выбирает направление движения случайным образом.

Сохраните в файле (текстовом или собственного формата) последовательность точек, в которых окажется лягушка в течение времени t .

Средствами *matplotlib* изобразите путь лягушки, который она проделает за заданное время t .

Постройте диаграмму (гистограмму) показывающую, сколько раз лягушка меняла направление движения, прежде чем ей удавалось поймать хотя бы одного комара.

Выделите случаи, когда лягушка меняла направление из-за того, что стайка улетела за пределы луга (желательно на той же диаграмме).

Постройте диаграмму (или график), показывающую распределение длины пройденного лягушкой пути между двумя успешными поимками комаров.

Вариант 4

Разработайте программу, моделирующее поведение «лягушки на лугу». Будем считать луг квадратом со стороной $2d$ на плоскости, исходно лягушка находится в точке с координатами $(0,0)$. Время считайте дискретным: все события происходят только в целые моменты времени.

В каждый очередной момент времени в произвольной точке на лугу возникает стайка комаров (в стайке не менее 1 комара и не более m). Эта стайка находится над лугом на протяжении некоторого времени (не большего k), если раньше комаров не спугнет лягушка.

Предположим, что на прыжок (любой) лягушка тратит g единиц энергии (калорий), а поедание одного комара даст ей f единиц энергии. Также предположим, что лягушка обладает прекрасным зрением и может пересчитать комаров в стае. Лягушка будет прыгать в направлении стаи только в том случае, если поедание половины комаров стаи может принести ей прибавку энергии. В противном случае она эту стаю будет игнорировать и рассматривать другие стаи.

Если существует несколько стаяк, которые подходят лягушке, она будет выбирать ближайшую из них. Если до двух или более стаяк расстояние одинаковое, лягушка выберет ту, где больше комаров. Если же и количество комаров в стаеках совпадает, лягушка выберет стаю случайным образом.

Если не найдется ни одной стаи, которая ее устроит, лягушка будет оставаться на месте и ожидать появления более подходящей стаи.

Лягушка направляется к выбранной стае по кратчайшему пути, но длина ее прыжка составляет не более единицы длины. Считайте, что лягушка всегда делает прыжки длиной одну единицу, быть может, за исключением последнего прыжка, когда она достигает стаи комаров. В последнем прыжке она ловит какое-то случайное количество комаров, не превышающее количества комаров в стае. Оставшиеся комары разлетаются за пределы луга.

Выбор стаи лягушка осуществляет после каждого прыжка, поэтому появление новых стаяк или исчезновение существующих может заставить ее изменить направление движения.

Сохраните в файле (текстовом или собственного формата) последовательность точек, в которых окажется лягушка в течение времени t .

Средствами *matplotlib* изобразите путь лягушки, который она проделает за заданное время t .

Назовем рангом стайки ее номер при упорядочении стаек по степени близости к лягушке.

Постройте диаграмму (гистограмму) показывающую распределение рангов стаек, выбираемых лягушкой.

Постройте график (или диаграмму), показывающий изменения энергии лягушки в зависимости от времени.

Определите количество стаек, которые лягушка ни разу не рассматривала как потенциальную добычу, а также количество стаек, статус которых менялся со временем (лягушка рассматривала их как добычу, но потом перестала рассматривать, и наоборот).

Также определите время простоя лягушки в зависимости от величин g и f (предположите зависимость и попробуйте визуализировать ее).

Вариант 5

Разработайте программу, моделирующее поведение «лягушки на лугу». Будем считать луг кругом с радиусом d на плоскости, исходно лягушка находится в точке с координатами $(0,0)$. Время считайте дискретным: все события происходят только в целые моменты времени.

В каждый очередной момент времени в произвольной точке на лугу возникает стайка комаров (в стайке не менее 1 комара и не более m). Эта стайка находится над лугом на протяжении некоторого времени (не большего k), если раньше комаров не спугнет лягушка.

В начальный момент времени лягушка направляется к первой появившейся стайке по кратчайшему пути, но длина ее прыжка составляет не более единицы длины. Считайте, что лягушка всегда делает прыжки длиной одну единицу, быть может, за исключением последнего прыжка, когда она достигает стайки комаров.

В последнем прыжке она ловит какое-то случайное количество комаров, не превышающее количества комаров в стайке. Оставшиеся комары разлетаются за пределы луга.

После очередного прыжка лягушка анализирует все направления, по которым она может достичь какой-либо стайки комаров. Чем меньше угол между направлением, по которому она может достичь стайки, и направлением, в котором она совершила предыдущий прыжок, тем выше вероятность выбора этого направления (функцию, описывающую эту зависимость, предложите самостоятельно).

Если есть две или более стайки в одном направлении, считайте, что лягушка выбрала ближайшую к ней стайку в этом направлении.

Сохраните в файле (текстовом или собственного формата) последовательность точек, в которых окажется лягушка в течение времени t .

Средствами *matplotlib* изобразите путь лягушки, который она проделает за заданное время t . Представьте, что круг разбит концентрическими окружностями на кольца одинаковой ширины. Постройте диаграмму, показывающую частоту попадания лягушки в то или иное кольцо, когда она достигает стайки комаров.

Предположите, что лягушка уверенно видит стайки комаров на расстоянии, не превышающем w , и поэтому будет игнорировать стайки, расположенные дальше.

Если стаек, которые расположены в зоне ее видимости, не будет, лягушка будет совершать прыжки случайным образом – в надежде, что увидит комаров. Измените алгоритм появления стаек таким образом, чтобы количество стаек, появляющихся в зоне видимости лягушки, было в k раз меньше, чем вне зоны ее видимости. Постройте распределение количества «холостых» прыжков лягушки за время моделирования.

3 Разработка класса как пользовательского типа данных и его применение

Python позволяет программисту работать, используя различные парадигмы программирования. Объектно-ориентированный подход, в частности, позволяет группировать данные и методы их обработки в рамках единой структуры. Поскольку типизация в Python является динамической, использование такого подхода дает большие гарантии корректной передачи параметров. В библиотеке *numpy* предусмотрены мощные средства работы с многочленами и матрицами, но в следующем задании потребуются разработать собственный класс и ряд его методов самостоятельно.

3.1 Задание «Разработка класса *Polynom*»

Разработайте класс *Polynom* для описания типа данных «многочлен». В классе должны быть следующие методы:

1. Конструктор. Он должен реализовывать формирование многочлена либо в виде списка коэффициентов (индекс элемента списка – степень при этом коэффициенте), либо в виде словаря (ключ словаря – степень при коэффициенте, значение – сам коэффициент). Конструктор принимает единственный параметр – список (или словарь) коэффициентов многочлена (разумеется, единственный, кроме параметра *self*). Иными словами, если у вас есть многочлен вида $P_n = 3 - 5x + 2x^2 - x^3$, то его можно задать либо списком [3, -5, 2, -1], либо словарем {0:3, 1:-5, 2:2, 3:-1}.
2. Метод `__str__(self)`. Этот метод представляет многочлен в виде строки, подобной «3-5^х+2х^2-х^3». Срабатывает автоматически, когда к многочлену применяется функция *print*.
3. Метод `__add__(self, p)`. Этот метод складывает два многочлена – тот, который задан параметром *self*, с тем, который

задан параметром p . Срабатывает автоматически, когда применяем к двум многочленам оператор $+$, в результате выдает новый многочлен. Также возможен вариант, при котором метод не возвращает новый многочлен, а замещает текущий (заданный параметром *self*).

4. Метод `__mul__(self, p)`. Этот метод умножает многочлен *self* на многочлен p , срабатывает автоматически, когда мы применяем к многочленам оператор $*$, в результате выдает новый многочлен.

5. Метод `__call__(self, x)` вычисляет значение многочлена *self* в точке x .

6. Метод `table(self, fileName, a, b, step)`. Этот метод формирует в файле с именем *fileName* таблицу из двух столбцов – в первом столбце идут значения x (начиная с точки a и до точки b с шагом $step$ между соседними значениями), во втором столбце – значения многочлена *self*, вычисленные для этого x .

Указание: воспользуйтесь замещенным методом `__call__` для вычисления значений многочлена.

Например, для многочлена $3 + 2x$ на отрезке $[a, b]$ ($[0, 0.5]$) шагом $step=0.1$ содержимое файла может выглядеть так:

```
0 3.0
0.1 3.2
0.2 3.4
0.3 3.6
0.4 3.8
0.5 4.0
```

7. Метод `grafic1(self, a, b, step)`. Этот метод должен строить график (с помощью `matplotlib`) для многочлена *self*. Значения a и b – это границы по x , где надо строить график, $step$ – это расстояние по x между соседними точками на графике.

8. Метод *grafic2(self, fileName)*. Этот метод должен строить график по точкам, содержащимся в файле *fileName*. Напомним, что в файле записана таблица (см. п. 6) из двух столбцов. В каждой строке первое число – это x , второе число – это y , нужно сформировать два списка: из всех x и из всех y , после чего построить график в *matplotlib*.

9. Метод *derivative(self)* вычисляет список коэффициентов для производной многочлена *self*, возвращает новый многочлен-производную.

10. Метод *integrate(self, a, b)* вычисляет определенный интеграл от многочлена *self* на отрезке от a до b .

Указание: зная список коэффициентов для многочлена *self*, можно вычислить список коэффициентов для его первообразной, после чего вычислить разность значений первообразной в точках b и a – это и будет значением определенного интеграла на заданном отрезке.

Напишите демонстрационную программу, использующую класс *Polynom* и все его возможности. Предусмотрите (текстовый) интерфейс для ввода данных, выбора операций над полиномом, вывода результатов, завершения работы с программой.

Дополнительные задания:

1. предусмотрите возможность использования в многочленах отрицательных степеней;
2. опишите метод *__repr__(self)* для альтернативного представления при выводе многочлена.

4 Разработка класса «Матрица» с использованием возможностей библиотеки *numpy*

В библиотеке *numpy* есть средства для работы с многомерными массивами (*ndarray*). Однако именно для двумерных массивов — матриц — ввиду их важности в прикладных задачах предусмотрен отдельный класс *matrix*. Ряд операций, поддерживаемых этим классом, отличается от таковых для многомерных массивов (например, это касается операции умножения; создания матриц специального вида и т.п.). В следующем задании предлагается разработать собственный класс с методами, выполняющими операции над матрицами. В методах рекомендуется обращаться к операциям класса *matrix*, предоставляемого *numpy*.

4.1 Задание «Разработка класса *MyMatrix*»

Разработайте класс *MyMatrix*. В нем обязательно должно быть поле — матрица (*numpy*). Возможно, также будет удобно использовать поле — размер матрицы.

В классе должен быть конструктор, который создает матрицу размера $n \times n$ и заполняет ее случайными вещественными числами из диапазона $[f, s]$. Конструктор должен принимать все эти величины как параметры (наряду с *self*). Возможно, хорошим решением будет использовать для заполнения матрицы метод *numpy.random.uniform()*.

В классе следует переопределить метод *__str__(self)* так, чтобы обеспечить вывод матрицы с выравниванием столбцов по правому краю и по десятичной точке.

Кроме этого, в классе следует определить три метода согласно вашему варианту задания.

Первый метод описывает преобразование, которое выполняется над матрицей, записанной в поле класса. Рекомендуется назвать его *modify(self)*. В некоторых вариантах в этот метод потребуется передавать дополнительный параметр (или параметры).

Два других метода описывают способы получения объектов класса *MyMatrix* с именами *C* и *D* (соответственно) из объектов класса *MyMatrix* с именами *A* и *B*.

Обозначения:

A, *B* — объекты класса *MyMatrix*, содержащие матрицы размерности $n \times n$;

AT — транспонированная матрица;

E — единичная матрица;

max и *min* в задании подразумевают перебор всех элементов матрицы;

C, *D* — объекты класса *MyMatrix*, которые получаются в результате вычислений.

Рекомендуется назвать методы *makeC(self, M)* и *makeD(self, M)*, где *M* — передаваемый в метод объект класса *MyMatrix*.

Вызовы методов должны выглядеть так:

C = *A.makeC(B)*

D = *A.makeD(B)*

Порядок выполнения работы.

1. Создать и распечатать две матрицы (два экземпляра класса *MyMatrix*), заполненные случайными числами из диапазона $[f, s]$. Это будут матрицы *A* и *B*.
2. Вычислить значения обоих матричных выражений из индивидуального задания. Для каждого выражения должен быть написан отдельный метод, в результате выполнения которого

получается новый объект класса *MyMatrix* (*C* или *D*). Распечатать полученные матрицы.

3. Для каждой из матриц *A* и *B* вызвать метод, описанный в п. 1 индивидуального задания и распечатать результат.

Указание: при выполнении данной работы следует использовать возможности *numpy* для реализации всех требуемых в заданиях методов.

Вариант 1

В заданной числовой вещественной матрице размерности $n \times n$ поменять местами *K*-ую строку и *L*-ую строку, элементы которой переставлены в обратном порядке. Вычислить значения следующих выражений:

1. $C = (A^2 / \max |B_{ij}| * (7B - A)) T$;
2. $D = (B^2 + 3E) / \min |A_{ij}|$.

Вариант 2

В заданной числовой вещественной матрице размерности $n \times n$ *L*-ый столбец увеличить на величину, равную среднему арифметическому максимального и минимального элементов исходной матрицы. Вычислить значения следующих выражений:

1. $C = AT * B^2 - (A^2 * BT - E) / \min |A_{ij}|$;
2. $D = A^3 + 3A^2 - 5B / \max |B_{ij}|$.

Вариант 3

Получить новую числовую матрицу, удалив из каждого столбца исходной вещественной матрицы размерности $n \times n$ минимальный по абсолютной величине элемента данного столбца. Вычислить значения следующих выражений:

1. $C = (A + 15E) * (2A - B) T / \max |B_{ij}|$;
2. $D = 2((A * B)^2 - (A + B) T / \min |A_{ij}|)$.

Вариант 4

Получить новую числовую вещественную матрицу, удалив из каждой строки исходной матрицы размерности $n \times n$ максимальный элемент данной строки. Вычислить значение следующих выражений:

1. $C = 3A^3 / \min |B_{ij}| + 3A^2 * BT - 5B^2 * AT;$
2. $D = (A + E) / \max |A_{ij}| - BT.$

Вариант 5

В заданной числовой вещественной матрице размерности $n \times n$ L -ую строку увеличить на величину, равную максимальному по абсолютной величине элементу исходной матрицы. Вычислить значение следующих выражений:

1. $C = (B - 4E) * A^2 + BT^2 * A / \min |A_{ij}|;$
2. $D = ((A + E) / \max |B_{ij}| * (4B - A)^2) T.$

Вариант 6

В заданной числовой вещественной матрице размерности $n \times n$ K -ый столбец уменьшить на величину, равную минимальному элементу исходной матрицы. Вычислить значения следующих выражений:

1. $C = (A - 7E) / \max |A_{ij}| + BT;$
2. $D = (A^2 + B^2 * (B - 6A)) T / \min |B_{ij}|.$

Вариант 7

В каждой строке исходной вещественной матрицы размерности $n \times n$ найти минимальный элемент и поставить его на первое место. Вычислить значения следующих выражений:

1. $C = 3(A + B / \min |A_{ij}| - E)^2 + 4(A * B)^3;$
2. $D = 9(A - B)T - (A + B)^2 * B / \max |B_{ij}|.$

Вариант 8

В каждом столбце исходной вещественной матрицы размерности $n \times n$ найти максимальный элемент и поставить его на первое место. Вычислить значения следующих выражений:

1. $C = (BT * A - AT * B) + (A - 6E) T / \min |B_{ij}|;$
2. $D = (A + B)^2 / \max |A_{ij}| - 8B^2.$

Вариант 9

Получить новую числовую матрицу, умножив каждый столбец исходной вещественной матрицы размерности $n \times n$ на минимальный по абсолютной величине элемент данного столбца. Вычислить значения следующих выражений:

1. $C = AT * 3B^2 - 5E;$
2. $D = (18A - B)^2 / (\max |A_{ij}| + \min |B_{ij}|) - AT.$

Вариант 10

В заданной вещественной матрице размерности $n \times n$ элементы L -го столбца разделить на величину, равную среднему арифметическому максимального и минимального элементов исходной матрицы. Вычислить значения следующих выражений:

1. $C = (BT * A - AT * B) + (B - A^3) / \min |A_{ij}|;$
2. $D = ((A^2)T - B * A + 6E) * \max |B_{ij}|.$

Вариант 11

Каждый элемент заданной числовой вещественной матрицы размерности $n \times n$ разделить на максимальный по абсолютной величине член исходной матрицы. Вычислить значения следующих выражений:

1. $C = ((B^2)T - B * A - 13E) * \max |A_{ij}|;$
2. $D = A^3 + 3A^2 - 5B / \min |B_{ij}|.$

Вариант 12

Каждый элемент заданной числовой вещественной матрицы размерности $n \times n$ увеличить на минимальный по абсолютной величине член исходной матрицы. Выдать на печать номера строки и столбца минимального по абсолютной величине элемента. Вычислить значения следующих выражений:

1. $C = ((A^2)T - B * A - 6E) / \max |A_{ij}|;$
2. $D = A^2 * 14BT + 8E^3 + B / \min |B_{ij}|.$

Вариант 13

В заданной числовой вещественной матрице размерности $n \times n$ найти строку, сумма элементов которой максимальна. Поставить ее на место первой строки. Среди оставшихся строк найти ту, сумма элементов которой максимальна. Поставить ее на место второй строки и т. д. Вычислить значения следующих выражений:

1. $C = (A + E) * (2A - B) T / \max |B_{ij}|;$
2. $D = (E - AT / \min |A_{ij}|) * (A + 3B)^2.$

Вариант 14

В заданной числовой вещественной матрице размерности $n \times n$ найти столбец, сумма элементов которого минимальна. Поставить его на место первого столбца (поменяв местами). Среди оставшихся столбцов найти тот, у которого сумма элементов минимальна. Поставить его на место второго столбца и т. д. Вычислить значения следующих выражений:

1. $C = ((AT)^2 - 5E) * (AT / \max |A_{ij}| + 3E) * AT;$
2. $D = (B / \min |B_{ij}| - 7E) * A^2 + (BT)^2 * A.$

Вариант 15

В заданной числовой вещественной матрице размерности $n \times n$ упорядочить элементы каждого столбца по возрастанию. Вычислить значения следующих выражений:

1. $C = 4(A * 3B)^3 / \min |A_{ij}| + 3(A + 15B - 3E)^2$;
2. $D = (4A^2 + B^2 - E) * (BT - AT) / \max |B_{ij}|$.

Вариант 16

В заданной числовой вещественной матрице размерности $n \times n$ переставить элементы каждого столбца в обратном порядке. Вычислить значения следующих выражений:

1. $C = (BT * A^2 - 6B) * (A + 3E) / \min |B_{ij}|$;
2. $D = A^2 * 5BT + A^3 / \max |A_{ij}| - E$.

Вариант 17

В заданной числовой вещественной матрице размерности $n \times n$ поменять местами K -й столбец и L -й столбец, элементы которого переставлены в обратном порядке. Вычислить значения следующих выражений:

1. $C = ((AT)^2 - 5E) * BT - E / \max |B_{ij}|$;
2. $D = 6A^2 * BT + A^3 - 3B / \min |A_{ij}|$.

Вариант 18

В заданной числовой вещественной матрице размерности $n \times n$ L -й столбец умножить на величину, равную среднему арифметическому максимального и минимального элементов исходной матрицы. Вычислить значения следующих выражений:

1. $C = (A^3 + E) * (2A - B) T / \max |B_{ij}|$;
2. $D = (BT * 5A - AT * B) * (A + 6E) T / \min |A_{ij}|$.

5 Разработка графического интерфейса пользователя при помощи tkinter

5.1 Импортирование tkinter

Рассмотрим создание графического интерфейса пользователя средствами *Python*. Для этого воспользуемся библиотекой *tkinter*: импортируем все компоненты из *tkinter*, а потом создадим корневое окно приложения, вызвав конструктор *Tk()*.

```
from tkinter import *  
root = Tk()
```

Именно в корневом окне *root* далее будут располагаться все оконные элементы — кнопки, поля, метки, списки и т. д. Однако, если запустить программу прямо сейчас, то никакого окна вы не увидите. Чтобы оно отобразилось на экране, добавим в конце строку

```
root.mainloop()
```

Теперь после запуска вы увидите пустое окно: рамку, ограничивающую белое поле.

5.2 Кнопки и обработчики событий

Современные операционные системы, как правило, предполагают интерактивное взаимодействие с пользователем. Для этого используются различные элементы управления, способные обеспечить реакцию на действия пользователя или иные события. Эти элементы управления снабжаются методами — обработчиками событий. В отличие от приложений, ориентированных на вычисления, в которых методы, как правило, вызываются в определенном порядке, в приложениях с графическим интерфейсом обработчики событий вызываются только тогда, когда эти события инициируются внешними источниками.

Добавим в наше пока ещё пустое окно виджет-кнопку и определим для нее обработчики событий. Пусть щелчок по кнопке левой клавишей мыши выводит в консоль сообщение о нажатии кнопки, а щелчок по кнопке правой клавишей мыши — печатает в консоль приветствие. Для этого сначала запишем обработчики событий в виде функций, потом займемся описанием кнопки.

```
from tkinter import *
root = Tk()
def but_command(event):
    print("Hey!")
def clicked_command(event):
    print("I'm clicked")
but = Button(root, text="Нажми на меня")
but.bind("<Button-1>", clicked_command)
but.bind("<Button-3>", but_command)
but.pack()
root.mainloop()
```

Функции объявляются при помощи ключевого слова *def*. В качестве единственного параметра функция для обработки события должна принимать именно объект-событие. Далее создается объект-кнопка, для которой указывается родительское окно (*root*), и свойство-текст, который будет на ней написан. Далее кнопке назначаются обработчики события через метод *bind*: первым параметром указывается, для какого события назначен обработчик, вторым параметром назначается сам обработчик (здесь *<Button-1>* означает щелчок левой клавишей мыши, *<Button-3>* – правой, а *<Button-2>* – средней). Далее кнопку надо разместить в окне, для этого применяется метод *pack()*. Затем запускается сценарий окна через *root.mainloop()*. Если теперь вы запустите программу и выполните нажатия на кнопку левой и

правой клавишами мыши, в консоли появятся различные сообщения (рис. 1).

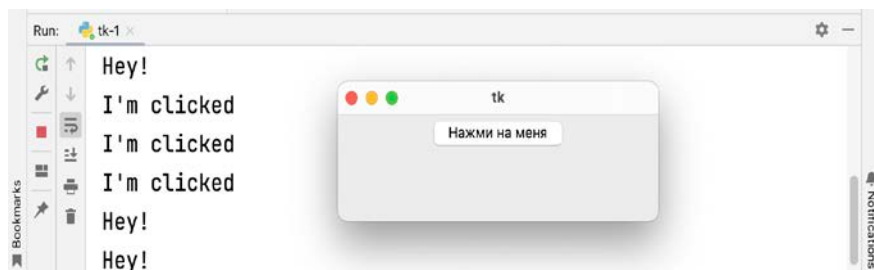


Рис. 1. Обработчики события для кнопки

При помощи метода *bind* можно делать привязку обработчиков самым различным событиям. Кроме метода *pack* существуют и другие способы размещения виджетов в окне приложения.

5.3 Метки

Добавим в наше приложение виджет-метку и будем по щелчку левой клавиши мыши с зажатой клавишей *Ctrl* выводить некоторое случайное число в эту метку.

Допишем перед *root.mainloop()* следующий фрагмент кода:

```
import random # для генератора случайных чисел
lb = Label(text='Я метка') # создаем саму метку
def lbl_command(event): # обработчик события
    lb['text'] = str(random.random())
    # к свойствам виджетов можно обращаться
    # по именам, используя их в виде индексов
lb.bind("<Control-Button-1>", lbl_command)
# делаем привязку события
lb.pack() # и размещаем метку
```

Теперь, запустив проект и нажимая на метку мышью (с зажатой клавишей *Ctrl*), можно убедиться, что каждый раз в нее записывается случайное число в диапазоне от 0 до 1 (рис. 2).

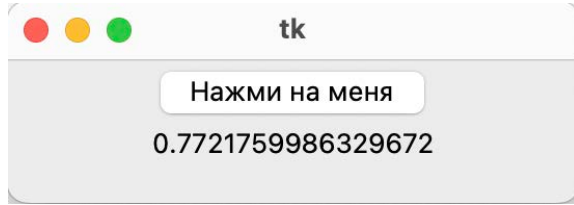


Рис. 2. Вывод случайного числа в метку

5.4 Текстовое поле ввода

Для добавления текстового поля ввода нам понадобится виджет *Entry*. Пусть текст из текстового поля копируется в нашу метку после нажатия кнопки (с зажатой клавишей *Shift*). Допишем перед последней строкой следующий фрагмент кода:

```
ent = Entry() #пустое текстовое поле ввода
ent.pack() #размещаем его
but1 = Button(text="В метку") #заводим кнопку
def printer(event): #обработчик для кнопки
    lb['text']=ent.get()
but1.bind("<Shift-Button-1>", printer)
# привязываем событие
but1.pack() # и размещаем кнопку
```

Теперь посмотрим, что получилось (рис. 3):

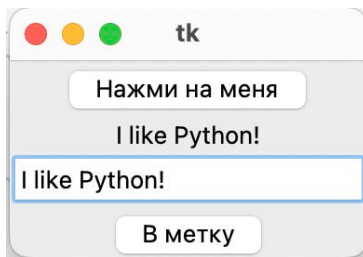


Рис. 3. Копирование текста из текстового поля в метку

5.5 Текстовая область

Для размещения многострочного текста потребуется виджет *Text*. Разместим текстовую область в нашем окне, и пусть щелчок по кнопке копирует содержимое области в консоль (рис. 4).

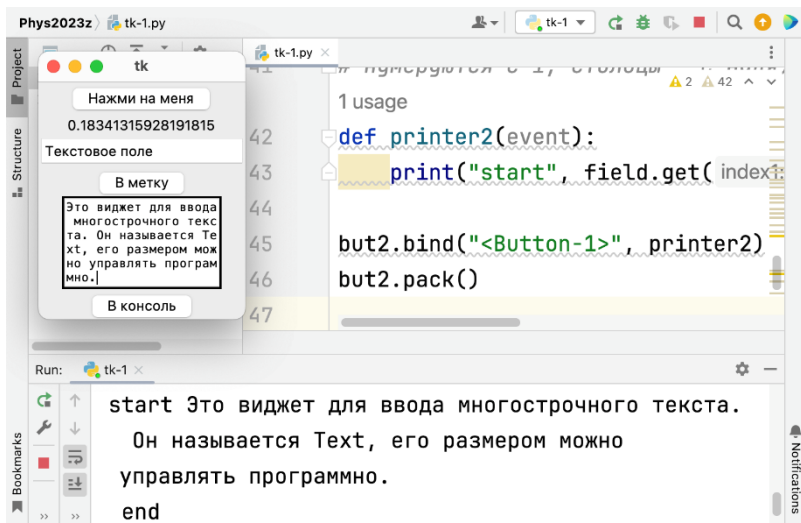


Рис. 4. Текстовая область

Допишем код (как обычно, перед *root.mainloop()*):

```
field = Text(width=20, height=6)
#наша текстовая область, 20 символов длиной,
#5 строк высотой
field.pack() #размещаем область
but2 = Button(text="В консоль")
#в обработчике мы заберем текст из области,
#первый параметр показывает индексы,
#начиная с которых надо делать выборку
#(строка-столбец, при этом строки
#нумеруются с 1, столбцы — с нуля),
```

```
#второй параметр — до какого элемента
def printer2(event):
    print("start", field.get('1.0', END), "end")
but2.bind("<Button-1>", printer2)
but2.pack()
```

Сохраним этот пример в модуле с названием *But-n-Text.py*.

5.6 Радиокнопки

Теперь научимся делать радиокнопки (переключатели) и связывать значения некоторой переменной и номер выбранного переключателя. Откроем новый *python*-файл с именем *Check-n-Radio.py* и запишем там следующий код:

```
from tkinter import *
root = Tk()
def change(): #обработчик выбора радиокнопки
    print(var.get())
#печатаем в консоль содержимое
#ассоциированной переменной
var=IntVar()
#ассоциированная переменная, целочисленная
var.set(1) #установим ее равной 1

rad0 = Radiobutton(root, text="Первая",
    variable=var, value=0, command=change)
#и при создании переключателей
#сопоставим им значение переменной
#мы таким образом включаем радиокнопки
#в одну радиогруппу
#и можем получать значения переменной
#для каждой радиокнопки
rad1 = Radiobutton(root, text="Вторая",
    variable=var, value=1, command=change)
rad2 = Radiobutton(root, text="Третья",
```

```

        variable=var, value=2, command=change)
rad0.pack()
rad1.pack()
rad2.pack()
root.mainloop()

```

При запуске этого кода окно будет выглядеть, как на рис. 5:

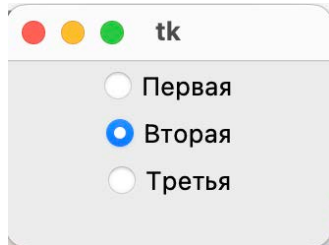


Рис. 5. Переключатели (радиокнопки)

Попробуйте выбирать разные радиокнопки и убедитесь, что в любой момент времени активна может быть только одна из них.

5.7 Флажки

Добавим теперь флажки в наш модуль и ассоциируем с ними переменные. Для включенного (поднятого) и выключенного флажка переменная может обладать различными значениями, и это при необходимости можно использовать в программе.

```

def change1():
    #обработчик события изменения состояния флажка 1
    print(c1.get())
def change2():
    #обработчик события изменения состояния флажка 2
    print(c2.get())
c1 = IntVar() # ассоциированные переменные
c2 = IntVar()
#создадим сами флажки, привяжем к ним переменные
#при включении флажка переменная

```

```

#получает значение 1 (2)
#при выключении - 0
che1 = Checkbutton(root, text="Первый флажок",
    variable=c1, onvalue=1, offvalue=0,
    command=change1)
che2 = Checkbutton(root, text="Второй флажок",
    variable=c2, onvalue=2, offvalue=0,
    command=change2)
che1.pack() #размещаем флажки
che2.pack()

```

При запуске кода и нажатиях на флажки у вас должно получиться примерно следующее поведение (рис. 6):

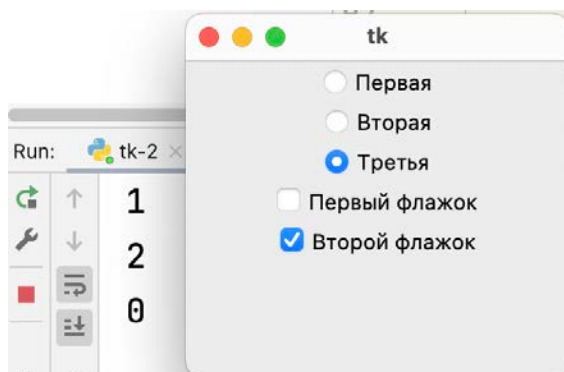


Рис. 6. Флажки и значение ассоциированной переменной

Попробуйте нажимать на флажки, убедитесь, что в консоль верно выводится значение ассоциированной переменной при работе с флажком.

5.8 Списки

Заведем новый python-файл *List-n-Scale.py*. Рассмотрим, как создать выпадающий список элементов. Для этого понадобится виджет *Listbox*. Разместим в документе следующий код:

```

from tkinter import *
root = Tk()
def confirm():#обработчик выбора элемента списка
    print(lst.curselection())
r = ['Delphi', 'Python', 'C++', 'Java']
#это будет содержимое нашего списка
lst = Listbox(root, selectmode=MULTIPLE,
              height=4)
#сам список на 4 строки высотой,
#можно выбирать несколько пунктов сразу
for i in r: #заполнение содержимого списка
    lst.insert(END, i)
lst.pack() #разместим список
but = Button(text=['Confirm'], command=confirm)
but.pack()
root.mainloop()

```

Поэкспериментируйте с программой: попробуйте изменить размеры списка, сделать одиночный выбор, выводить не индексы элементов (рис. 7), а сам текст (содержимое).

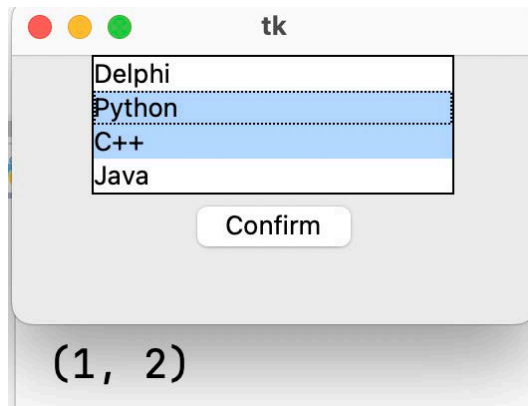


Рис. 7. Список с множественным выбором и вывод индексов выбора в консоль

5.9 Шкала

Очень часто бывает нужно разместить шкалу, которая, например, управляла бы масштабом какого-либо элемента в окне. Научимся работать с горизонтальной и вертикальной шкалой.

Поместим следующий код в документ:

```
def f1(a):  
    #это обработчик события для первой шкалы  
    print(a)  
def f2(b):  
    #обработчик событий для второй шкалы  
    print(b)  
  
    #теперь займемся шкалами.  
    #Сначала горизонтальная, можно управлять  
    #ее размерами и ее диапазоном значений,  
    #указать интервал делений,  
    #разрешение (то число, на которое можно  
    #дискретно двигать ползунок),  
    #назначить обработчик  
    sca1 = Scale(root, orient=HORIZONTAL, length=300,  
        from_=0, to=100, tickinterval=10,  
        resolution=5, command=f1)  
    #и вертикальная шкала  
    sca2 = Scale(root, orient=VERTICAL, length=200,  
        from_=1, to=2, tickinterval=0.2,  
        resolution=0.1, command=f2)  
    sca1.pack()  
    sca2.pack()
```

Подвигайте ползунки на шкалах (рис. 8) и посмотрите, какие именно значения печатаются в консоли. Почему происходит именно так?

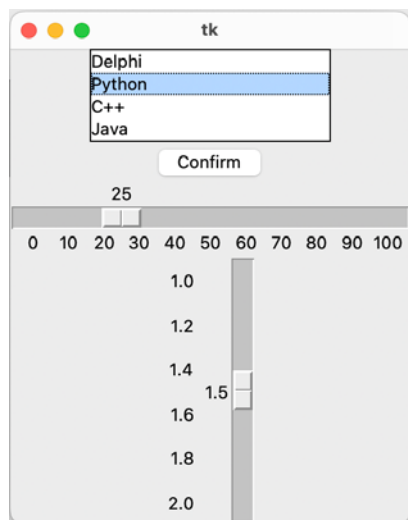


Рис. 8. Горизонтальная и вертикальная шкалы

5.10 Скроллинг

Полосы прокрутки, как вы наверное уже обратили внимание, автоматически не добавляются. Придется их добавлять вручную. Давайте разберемся, как это сделать. Заведем новый *python*-скрипт с именем *Scroll.py*.

```
from tkinter import *
root = Tk()
#заведем уже известную текстовую область.
#Она была ранее без полос прокрутки
tx = Text(root, width=15, height=3, font='14')
#а теперь создадим полосу прокрутки,
#и свяжем 2 виджета вместе
#первым шагом в связке назначим команду
#скроллу tx.yview
scrV = Scrollbar(root, orient=VERTICAL,
                 command=tx.yview)
#затем сконфигурируем текстовую область
```

```

#и назначим ей только что созданный
#скроллбар в качестве реакции на скроллинг по y
tx.configure(yscrollcommand=scrV.set)
#разместим все элементы в окне
#Здесь фигурирует новый для нас
#способ размещения – сетка.
#Для каждого элемента можно указать
#номер столбца и строки, куда его надо поместить
tx.grid(row=0, column=0)
scrV.grid(row=0, column=1)
root.mainloop()

```

В итоге проведенных действий у вас может получиться примерно следующий вид окна (рис. 9).

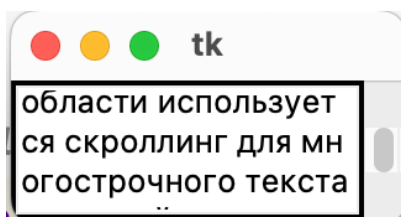


Рис. 9. Полоса прокрутки

Думаю, вы уже догадались, что аналогично можно назначить скроллинг и другим компонентам, например, тому же выпадающему списку.

5.11 Рисование на форме. Канва

Чтобы немного порисовать, нам понадобится элемент (виджет) канва. Создать его с настройками по умолчанию максимально просто:

```
canvas = Canvas(root)
```

Чтобы нарисовать линию, воспользуемся методом *create_line*. Параметрами будут координаты начала и конца линии (или

линий – можно одной командой рисовать мультилинии), также можно в качестве опции указать штрихование или цвет линии.

```
from tkinter import *
root = Tk()
canvas = Canvas(root)
canvas.create_line(15, 25, 200, 25)
canvas.create_line(300, 35, 300, 200,
    dash=(4, 2))
canvas.create_line(55, 85, 155, 85,
    105, 180, 55, 85, fill='green')
canvas.pack(fill=BOTH, expand=1)
root.mainloop()
```

Запустите этот фрагмент кода и посмотрите, что именно у вас нарисовалось.

Немного усложним задачу. Нарисуем на канве разноцветные (закрашенные) прямоугольники:

```
canvas.create_rectangle(30, 10, 20, 180,
    outline="black", fill="#fb0")
canvas.create_rectangle(150, 10, 240, 80,
    outline="#f50", fill="#f50")
canvas.create_rectangle(270, 10, 370, 80,
    outline="#05f", fill="#05f")
```

Очевидно, что первые 2 числа в параметрах задают координаты левого верхнего угла прямоугольника, последние 2 – координаты правого нижнего. Цвета «границы» и внутренней заливки могут быть разными и могут задаваться как значениями константами, так и шестнадцатеричными кодами цветов.

Более сложные фигуры, например, овалы, сектора и закрашенные многоугольники можно рисовать при помощи примитивов *create_oval*, *create_arc* и *create_polygon*.

```
canvas.create_oval(10, 10, 80, 80,
    outline="gray", fill="gray", width=2)
```

```

canvas.create_oval(110, 10, 210, 80,
    outline="gray", fill="gray", width=2)
canvas.create_arc(30, 200, 90, 100, start=0,
    extent=210, outline="gray",
    fill="gray", width=2)
points = [150, 100, 200, 120, 240, 180, 210,
    200, 150, 150, 100, 200]
canvas.create_polygon(points, outline='gray',
    fill='gray', width=2)

```

Ну и, наконец, надпись можно сделать так:

```

canvas.create_text(20, 220, anchor=W,
    font="Purisa",
    text="Вот такое рисование средствами Python")

```

Запустите полученный код. Окно с фигурами будет выглядеть примерно следующим образом (рис. 10):

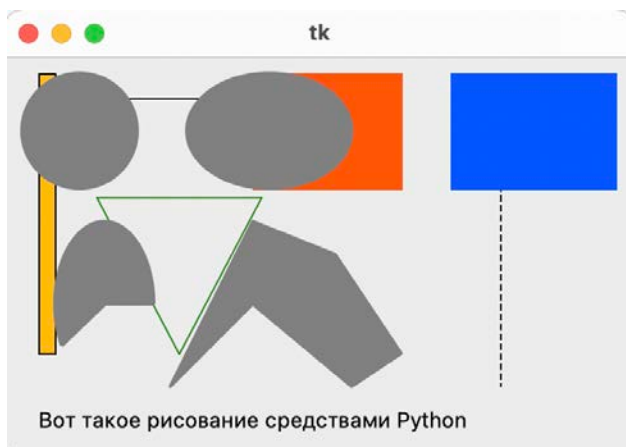


Рис. 10. Рисование средствами *Python* на канве

Сохраним этот файл с именем *Canvas.py*.

5.12 Контрольные вопросы

1. Как можно изменить значения свойств виджетов? Например, название кнопки, размеры текстового поля и т. д.?
2. Для чего служат ассоциированные переменные? Каких типов они должны быть для радиокнопок, флажков, текстовых полей?
3. Как можно забрать и поместить текст из текстовой области? Текстового поля?
4. Какие вы знаете способы размещения элементов на форме? Чем они отличаются? Как можно позиционировать виджеты в окне?
5. Как добавить скроллинг к выпадающему списку?
6. Самостоятельно изучите, как можно размещать виджеты во фреймах.
7. Как при помощи шкалы (ползунка) можно управлять размером текста в текстовой области?
8. Самостоятельно изучите способы создания и работы с диалоговыми окнами. Продемонстрируйте работу, например, с диалоговым окном выбора цвета.

5.13 Задания

Вариант 1

На форме расположены три флажка и кнопка. Флажки задают *red*, *green*, *blue* компоненты цвета (установите их равными 255). Щелчок по кнопке устанавливает цвет кнопки в соответствии с выбранными флажками компонентами цвета.

Вариант 2

Поместите на форму флажки и кнопку. Нажатие на кнопку включает все флажки. Повторное нажатие на кнопку снимает все

флажки. Рядом с флажками располагается текст, нажатие на текст или флажок должно также включать соответствующий флажок.

Вариант 3

На форму поместите бегущую строку и список с размерами шрифта. Выбор размера шрифта в списке меняет на таковой размер текста в бегущей строке.

Вариант 4

На форму поместите флажки и текстовые поля. Например, флажки будут отмечать список экзаменов в зимнюю сессию, а текстовые поля – оценки за экзамены. Поставьте еще один флажок «Сессия сдана». Как только вы проставите все оценки, автоматически появится галочка для флажка «Сессия сдана» и появится окошко с сообщением об итогах сессии (отлично, хорошо, удовлетворительно или неудовлетворительно). Если хотя бы по одному экзамену выставлена неудовлетворительная оценка, флажок «Сессия сдана» очищается и флажком отмечаем экзамен, сданный на 2.

Вариант 5

На форме располагается текстовая область, текстовое поле, кнопка и радиогруппа из 2 радиокнопок. При активной одной радиокнопке в поле вводится текст, после нажатия на кнопку он оказывается в текстовой области. В текстовую область непосредственно ничего впечатать нельзя, при попытке это сделать должно выдаваться сообщение-предупреждение. При активной другой радиокнопке в текстовую область вводится текст, после нажатия на кнопку он оказывается в текстовом поле. В текстовое поле непосредственно ничего впечатать нельзя, при попытке сделать это должно появляться предупреждение.

Вариант 6

Поместите на форму флажки и 4 кнопки. Нажатие на первую кнопку поднимает все флажки, нажатие на вторую – снимает все флажки. Нажатие на 3-ю кнопку позволяет поставить все флажки, если они сняты, и снять все флажки, если они подняты. Нажатие на 4 кнопку делает все флажки неактивными, повторное нажатие на эту кнопку делает флажки активными.

Вариант 7

Поместите на форму флажок, радиогруппу, список, и кнопку. Щелчок мышкой по тексту рядом с флажком должен включать-выключать его. Значения из выпадающего списка должны включать соответствующие радиокнопки. Щелчок по кнопке меняет местами надписи на кнопке и рядом с флажком.

Вариант 8

Поместите на форму флажки и примечание, что выбрать можно ограниченное количество пунктов. Попытка выбора большего числа флажков вызывает предупреждающее сообщение. При этом лишние флажки снимаются. Повторная попытка выбрать лишний флажок должна привести к появлению метки-предупреждения, которая будет «убегать» от курсора мыши, не выходя за форму.

Вариант 9

Поместите на форму флажки, несколько радиокнопок, несколько текстовых полей и кнопку. Флажки можно отмечать, а по нажатию на кнопку должно выводиться сообщение о количестве выбранных флажков, радиокнопок и заполненных текстовых полей. После этого заполненные поля становятся неактивными, поднятые флажки и радиокнопки становятся неактивными.

Вариант 10

Заполняем регистрационную карточку (например, на конференцию). Необходимо отметить имя, фамилию, университет (из списка), должность (из списка), страну (последнюю можно выбрать из выпадающего списка). Данные надо продублировать. Для этого поместите на форму все необходимые элементы и кнопку. Заполняется только одна карточка, нажатием на кнопку данные копируются во вторую карточку. Если какое-либо поле не заполнено, либо заполнено неверно, об этом должно быть выдано сообщение и предложено данное поле заполнить еще раз. Приветствуется возможность сохранить данные в файл.

Вариант 11

Поместите на форму радиокнопки и список. Выбор радиокнопки формирует значения в списке. Например, радиокнопки могут указывать страны, а в выпадающем списке будут присутствовать международные аэропорты данных стран.

Вариант 12

Поместите на форму несколько радиокнопок с обозначением цветов и выпадающий список с обозначением цветов. Выбор радиокнопки делает этот цвет цветом фона формы, выбор элемента в списке делает цвет цветом текста на форме.

Вариант 13

Установите на форму кнопку и список. Выбранное значение из списка должно отображаться на кнопке. Щелчок по кнопке запускает бегущую строку на кнопке. Повторный щелчок останавливает бегущую строку. Бегущая строка должна менять свой цвет.

Вариант 14

Делаем «планировщик заданий». Поместите на форму компонент (или компоненты) для выборы даты, компоненты (возможно, списки) для выбора часов и минут, текстовое поле, кнопку, текстовую область. В текстовое поле записываем задание, выбираем для него дату, устанавливаем время. Щелчок по кнопке добавляет выбранные данные в текстовую область.

Вариант 15

Поместите на форму компонент (или компоненты) для выбора даты, текстовую область, метку и кнопку. После выбора даты в метку выводится, какой это день недели, а в текстовой области появляется расписание на этот день. Щелчок по кнопке очищает текстовую область, значение метки, выбранную дату.

Вариант 16

На форме расположены флажки с названиями учебных дисциплин и радиогруппа с фамилиями студентов. Также в теле документа есть кнопка. Можно выбрать ФИО студента и отметить дисциплины, которые он изучает. Клик по кнопке добавляет ФИО студент и список его дисциплин в конец нумерованного списка в документе (это может быть виджет выпадающий список, либо виджет *Text*). Клик по пункту списка убирает его из списка.

Вариант 17

На форме расположен нумерованный список (это может быть виджет *Listbox*, либо *Text*, либо любой другой подходящий) и кнопка. Щелчок левой клавишей мыши по пункту списка помещает содержание этого пункта на кнопку. Клик по кнопке убирает первый пункт из списка и добавляет его в конец списка.

Вариант 18

На форме расположен выпадающий список и нумерованный список (вероятно, его удобно организовать в виде меток) с теми же самыми названиями пунктов. Выбор пункта в выпадающем списке меняет цвет соответствующего пункта в нумерованном списке и подсвечивает его каким-нибудь контрастным цветом. Клик левой клавишей по пункту нумерованного списка делает его выбранным в выпадающем списке.

Вариант 19

На форме расположены текстовое поле, кнопка и список. В текстовом поле можно набрать какой-то текст, клик по кнопке добавляет его в начало маркированного списка. Клик по пункту списка убирает этот пункт из списка. Двойной клик по телу документа делает кнопку синей с белым шрифтом на ней.

Вариант 20

На форме расположена радиогруппа из нескольких радиокнопок с указанием цвета и таблица (сетка) размером 5*5. Наведение мыши на ячейку таблицы (сетки) закрашивает ячейку в цвет, соответствующий выбранной радиокнопке. Когда мышь уходит с ячейки, ее цвет восстанавливается. Клик мыши по ячейке закрашивает всю строку таблицы выбранным цветом.

6 Использование графической подсистемы для моделирования движущихся объектов

При изучении физических явлений и процессов бывает полезно использовать компьютерную визуализацию этих явлений и процессов. Конечно, зачастую такие приложения носят скорее демонстрационный характер. Однако возможность изменять параметры в широком диапазоне, а также регулировать скорость происходящего дает некоторые преимущества по сравнению с реальным экспериментом.

В разделе 5.11 обсуждалось рисование графических примитивов на холсте Canvas, предоставляемом библиотекой *tkinter*.

Комбинируя рисование и очистку холста в течение коротких промежутков времени, можно добиться имитации движения объектов. Разумеется, координаты объектов должны получаться из уравнений, описывающих соответствующий физический процесс.

6.1 Задание «Моделирование движения объекта с учетом гравитации»

В этом задании нужно моделировать движение объекта с учетом гравитации. Будем считать, что сила тяготения сонаправлена с осью OY в *tkinter* (т.е. направлена вниз). Объектом должен быть шарик, который двигается внутри прямоугольника, испытывая абсолютно упругие соударения с его стенками.

Радиус шарика, его начальные координаты и начальные компоненты скорости по осям OX и OY задаются в интерактивном режиме. Должна быть панель (окно), в которой можно изменять характеристики шарика и применять сделанные изменения. Начальные координаты должны задаваться относительно одного из

углов прямоугольника (например, верхнего левого). Также следует предусмотреть панель (окно), в которой можно выполнять настройки «окружения шарика»: задавать размеры прямоугольника и ускорение свободного падения.

Предусмотрите возможность включать и выключать отображение траектории шарика (более точно, траекторию его центра как материальной точки).

Примерный внешний вид окна приведен на рис. 11.

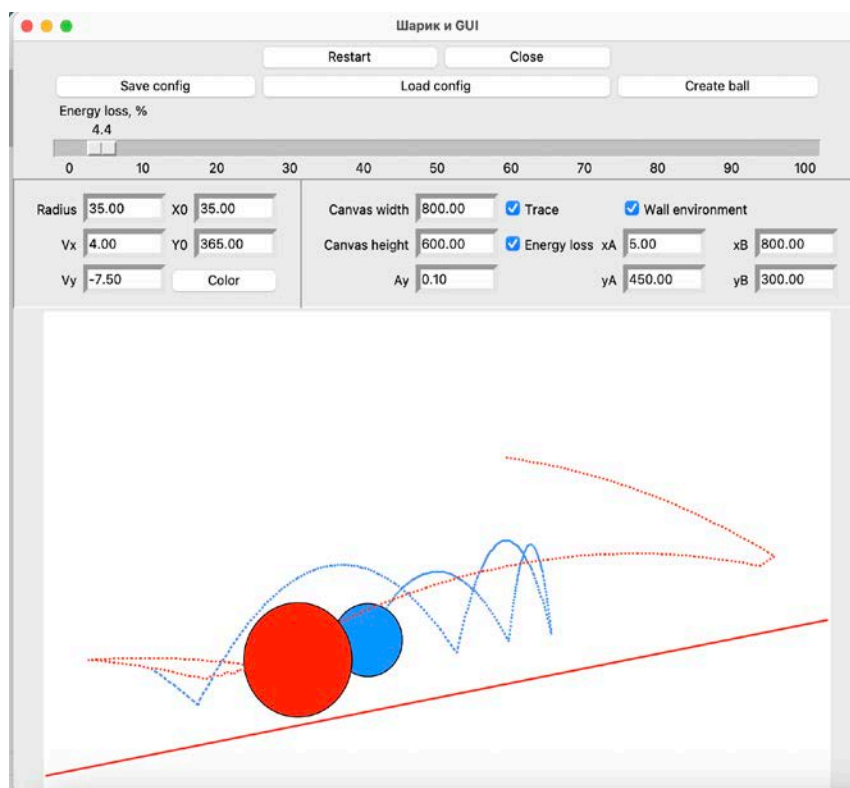


Рис. 11. Движение шариков при неупругих соударениях с препятствиями

Дальнейшая доработка проекта:

1. Считайте, что при каждом соударении шарик теряет некоторое количество энергии (уменьшается абсолютная величина компонентов его скорости).

2. Замените область движения шарика с прямоугольника на какую-либо кривую (например, окружность; допустимы ступенчатые функции, многоугольники и т. п.).

3. Предусмотрите возможность добавления еще одного шарика (или большего количества), который будет двигаться, оставляя траекторию другого цвета.

4. Предусмотрите возможность «паузы» (временной остановки) движения объектов.

5. Придумайте формат для хранения настроек шарика и области его движения (своего рода файл конфигурации), чтобы можно было загружать этот файл и использовать эти настройки при исполнении вашей программы.

7 Рисование самоподобных кривых средствами Python

Рассмотрим, как можно организовать рисование простых фигур и графиков на форме на примере построения самоподобных регулярных кривых. Фрактал — это фигура, состоящая из частей, которые в каком-то смысле подобны целому (такое неформальное определение дано фракталу основоположником фрактальной геометрии Бенуа Мандельбротом). В своей книге «Фрактальная геометрия природы» он приводит множество примеров «самоподобных» объектов в природе — русла рек, береговые линии, ветви деревьев, кровеносная система человека, морские волны... Фрактальные построения актуальны, например, в компьютерной графике для создания реалистичных пейзажей. Для описания фракталов существуют различные методики, например, *L*-системы (системы Линденмайера), применение систем итерирующих функций (*IFS*). *L*-система — это грамматика некоторого языка (достаточно простого), которая описывает инициатор и преобразование, выполняемое над ним. Система итерирующих функций — это совокупность сжимающих аффинных преобразований. Как известно, аффинные преобразования включают в себя масштабирование, поворот и параллельный перенос. Аффинное преобразование считается сжимающим, если коэффициент масштабирования меньше единицы.

Ознакомимся с основными приемами рисования в приложении с графическим интерфейсом на примере кривой Коха, салфетки Серпинского и драконовой ломаной (дракон Хартера-Хейтуэя).

Построение триадной кривой Коха в режиме *L*-системы начинается с отрезка единичной длины — это 0-е поколение кривой Коха. Далее каждое звено (в нулевом поколении один отрезок) за-

меняется на образующий элемент. В результате такой замены получается следующее поколение кривой Коха. В первом поколении – это кривая из четырех прямолинейных звеньев, каждое длиной по $1/3$. Для получения третьего поколения продельваются те же действия – каждое звено заменяется на уменьшенный образующий элемент. Итак, для получения каждого последующего поколения, все звенья предыдущего поколения необходимо заменить уменьшенным образующим элементом. Кривая n -го поколения при любом конечном n называется предфракталом. На рисунке 12 представлены шесть поколений кривой. При n стремящемся к бесконечности кривая Коха становится фрактальным объектом.

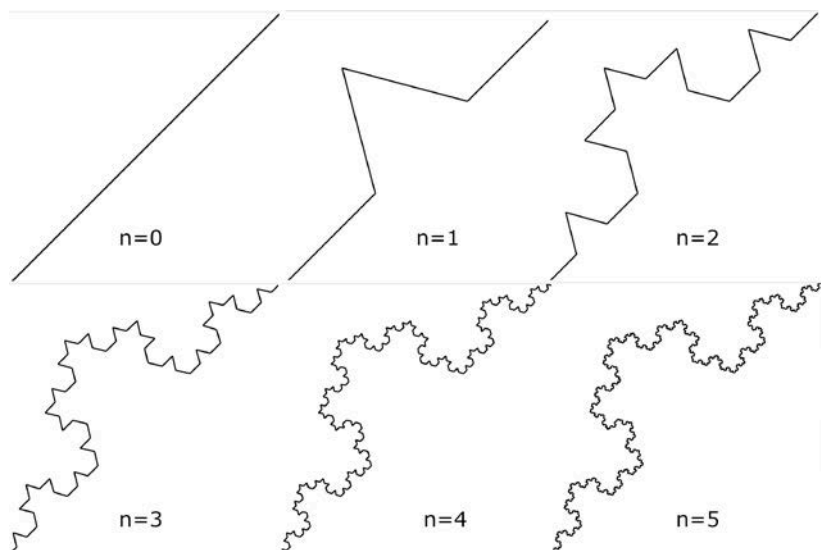


Рис. 12. Кривая Коха разных порядков

Салфетка Серпинского нулевого порядка состоит из закрашенного равностороннего треугольника (цвет закрашки на картинке — черный). Салфетка первого порядка получится, если разделить

треугольник на четыре равные части и «извлечь» центральную (по-красить центральную часть белым). Если для оставшихся закрашенных частей треугольника повторить процедуру, получится салфетка второго порядка, затем третьего (рис. 13) и т.д.

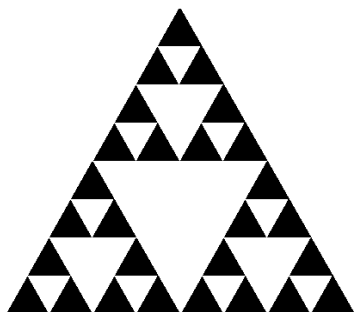


Рис. 13. Салфетка Серпинского третьего порядка

Драконова ломаная получится, если полосу бумаги согнуть пополам K раз (сгиб всегда производится в одну и ту же сторону), а потом разогнуть так, чтобы каждый угол был равен 90 градусам. Предельная фрактальная кривая (при K стремящемся к бесконечности) называется драконом Хартера-Хейтуэя (рис. 14).



Рис. 14. Драконова ломаная (порядки $K=1, 2, 3, 4, 5$)

Перейдем теперь непосредственно к разработке приложения для рисования данных фрактальных кривых. Создадим новый *python*-файл, в котором импортируем *tkinter*, создадим корневое окно и два фрейма, у которых родительским элементом назначим корневое окно.

```

from tkinter import *
root = Tk() #явно создать корневое окно
pict=Frame(root) #здесь будет канва и рисунок
manage=Frame(root)
#здесь будут управляющие элементы
pict.pack(side=LEFT) #разместим фреймы в окне
manage.pack(side=RIGHT)

```

Теперь выделим место под рисунок: организуем канву и зальем ее белым прямоугольником. Канву будем создавать во фрейме *pict*.

```

canvas=Canvas(pict, width=400, height=400)
canvas.create_rectangle(0,0, 400,
                        400, outline='#fff', fill = '#fff')
canvas.pack(fill=BOTH, expand=1)

```

Добавим элементы управления: радиокнопки для выбора вида фрактала, кнопку вызова диалогового окна для выбора цвета кривой, две шкалы для установки толщины линии и порядка фрактала, а также кнопки для рисования графика и стирания графика. Т. е. в итоге должно получиться что-то похожее (рис. 15):

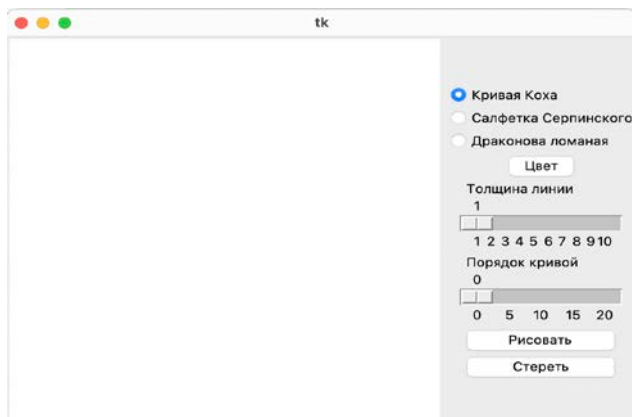


Рис. 15. Внешний вид окна разрабатываемого приложения

Чтобы получить вид приложения, как на рисунке 15, поместим в код программы следующие элементы:

```
rdVar=IntVar()  
#ассоциированная переменная для радиокнопок  
rdVar.set(0)  
#по умолчанию выбрана будет первая радиокнопка  
#сами радиокнопки, родительский элемент – фрейм  
rad0 = Radiobutton(manage,  
    text="Кривая Коха", variable=rdVar, value=0)  
rad1 = Radiobutton(manage,  
    text="Салфетка Серпинского",  
    variable=rdVar, value=1)  
rad2 = Radiobutton(manage,  
    text="Драконова ломаная",  
    variable=rdVar, value=2)  
#экспериментируем с положением радиокнопок  
rad0.pack(side=TOP, anchor=W)  
rad1.pack(side=TOP, anchor=W)  
rad2.pack(side=TOP, anchor=W)  
#чтобы можно было управлять цветом кривой,  
#заведем переменную и запишем туда черный цвет  
myColor="#000"  
#теперь, чтобы можно было вызывать  
#диалоговое окно выбора цвета, сделаем импорт  
from tkinter.colorchooser import askcolor  
#и заведем обработчик события,  
#где будет вызываться диалоговое окно для цвета  
def setColor(event):  
    global myColor  
    #тут уточняем, что работать будем  
    #с вышезаведенной переменной  
    (RGB, myColor)=askcolor()  
    #запоминаем результат выбора цвета
```

```

#а тут заводим кнопку, щелчок по которой
#вызовет диалог для выбора цвета
#родительский элемент — фрейм
butColor=Button(manage, text="Цвет")
butColor.bind('<Button-1>', setColor)
butColor.pack()

#теперь введем 2 шкалы
#и определим ассоциированные переменные
myPenWidth=IntVar() #для толщины линии фрактала
myPenWidth.set(1) #установим ее равной 1
penWidth = Scale(manage, label="Толщина линии",
    orient=HORIZONTAL, length=150,
    from_=1, to=10, tickinterval=1,
    resolution=1, variable=myPenWidth)

#и ассоциированная переменная для второй шкалы —
#порядок фрактала
myCurvePower=IntVar()
myCurvePower.set(0) #по умолчанию это 0
curvePower = Scale(manage,
    label="Порядок кривой",
    orient=HORIZONTAL, length=150,
    from_=0, to=20, tickinterval=5,
    resolution=1, variable=myCurvePower)

#разместим шкалы
penWidth.pack()
curvePower.pack()

#Теперь поставим кнопки для прорисовки фрактала
#и очищения области рисования
#Введем функцию-обработчик события
#для рисования кривой
#В зависимости от значения выбранной
#радиокнопки будем рисовать фрактал

```

```

def draw(event):
    if (rdVar.get()==0): # кривая Коха
        pass # рекурсия для Коха, напишем позже
    elif (rdVar.get()==1):
        pass # рекурсия для Серпинского
    else:
        pass # рекурсия для драконовой ломаной
#Теперь создадим и разместим кнопки для рисования
butDraw=Button(manage, text="Рисовать", width=12)
butDraw.bind("<Button-1>", draw)
butDraw.pack()
#А это примитивная очистка области рисования:
#мы просто закрашиваем все белым
#прямоугольником. В Python можно и иначе,
#как именно — предлагаю выяснить и реализовать
#самостоятельно в качестве упражнения
def clear(event):
    canvas.create_rectangle(0, 0,
        canvas.winfo_width(), canvas.winfo_height(),
        outline='#fff', fill='#fff')
butClear=Button(manage, text="Стереть", width=12)
butClear.bind("<Button-1>", clear)
butClear.pack()
#не забываем запустить сценарий корневого окна
root.mainloop()

```

Теперь, если запустить проект на выполнение, вы увидите приведенную на рисунке выше форму. Уже работают кнопки вызова диалогового окна выбора цвета, кнопка очистки формы, радиокнопки и ползунки. Давайте запрограммируем функцию *draw*, чтобы можно было рисовать фракталы.

Рассмотрим подробнее построение рекурсивных функций для рисования кривой Коха, салфетки Серпинского и драконовой лома-

ной. Чтобы построить кривую Коха, надо научиться рисовать ее генератор (см. рис. 16) для любых пар точек $(X1, Y1)$ и $(X2, Y2)$.

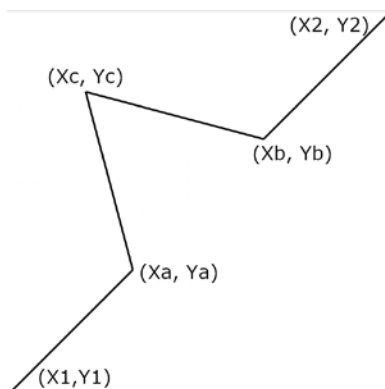


Рис. 16. Генератор кривой Коха

Точки (Xa, Ya) , (Xb, Yb) и (Xc, Yc) определяются формулами:

$$\begin{aligned}\alpha &= \text{atan2}(Y2 - Y1, X2 - X1), \\ R &= \sqrt{((X2 - X1)^2 + (Y2 - Y1)^2)}, \\ Xa &= X1 + \frac{R}{3} \cos \alpha, Ya = Y1 + \frac{R}{3} \sin \alpha, \\ Xb &= X1 + \frac{2R}{3} \cos \alpha, Yb = Y1 + \frac{2R}{3} \sin \alpha, \\ Xc &= Xa + \frac{R}{3} \cos \left(\alpha - \frac{\pi}{3} \right), Yc = Ya + \frac{R}{3} \sin \left(\alpha - \frac{\pi}{3} \right).\end{aligned}$$

Для справки: функция $\text{atan2}(y, x)$ возвращает угол между осью OX и отрезком, соединяющим начало координат с точкой (x, y) .

На псевдокоде алгоритм построения кривой Коха порядка $order$ на отрезке $(X1, Y1, X2, Y2)$ очень прост:

ЕСЛИ $order=0$, построить отрезок $(X1, Y1, X2, Y2)$

ИНАЧЕ

 построить кривую Коха порядка **order-1** на отрезке (X1, Y1, Xa, Ya)

 построить кривую Коха порядка **order-1** на отрезке (Xa, Ya, Xc, Yc)

 построить кривую Коха порядка **order-1** на отрезке (Xc, Yc, Xb, Yb)

 построить кривую Коха порядка **order-1** на отрезке (Xb, Yb, X2, Y2)

Реализуем алгоритм в программе. Введем рекурсивную функцию *Koch*:

```
import math
def Koch(order, x1, y1, x2, y2):
    if (order==0):
        canvas.create_line(x1,y1,x2,y2,
            fill=myColor, width=myPenWidth.get())
    else:
        alpha=math.atan2(y2-y1, x2-x1)
        R=math.sqrt((x2-x1)*(x2-x1)+
            (y2-y1)*(y2-y1))
        #вычислим xA, yA, xB, yB, xC, yC
        xA=x1+(R/3)*math.cos(alpha)
        yA=y1+(R/3)*math.sin(alpha)
        xC=xA+R*math.cos(alpha-math.pi/3)/3
        yC=yA+R*math.sin(alpha-math.pi/3)/3
        xB=x1+2*R*math.cos(alpha)/3
        yB=y1+2*R*math.sin(alpha)/3
        #рекурсивные вызовы
        Koch(order-1, x1, y1, xA, yA)
        Koch(order-1, xA, yA, xC, yC)
        Koch(order-1, xC, yC, xB, yB)
        Koch(order-1, xB, yB, x2, y2)
```

Разумеется, описание функции для рисования кривой Коха надо делать до того, как мы описываем функцию-обработчик *draw*. Теперь модифицируем функцию *draw*:

```
def draw(event):
    if (rdVar.get()==0): #кривая Коха
        x1=0
        y1=canvas.winfo_height()
        x2=canvas.winfo_width()
        y2=0
        #рекурсия для Коха
        Koch(myCurvePower.get(),x1, y1, x2, y2)
    elif (rdVar.get()==1):
        pass #рекурсия для Серпинского
    else:
        pass #рекурсия для драконовой ломаной
```

Запустите программу (рис. 17) и попробуйте нарисовать различными цветами кривые нулевого, первого, второго, третьего, четвертого и пятого порядков.

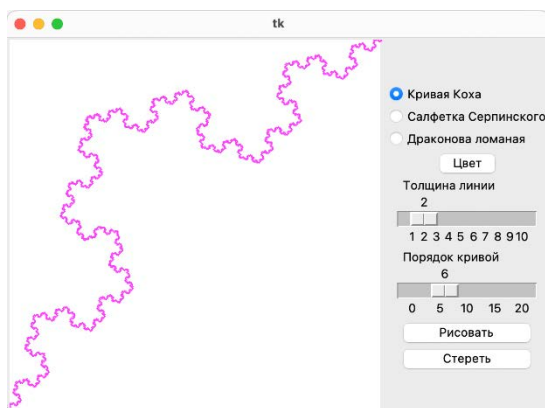


Рис. 17. Построение Кривой Коха 6 порядка

Для прорисовки салфетки Серпинского надо научиться рисовать заполненные цветом треугольники. Фактически, построив начальный цветной треугольник достаточно из него каждый раз «выбрасывать» (зарисовывая цветом фона) центральный треугольник.

Точкой отсчета рисования треугольника будет точка A с координатами (x, y) , длина стороны треугольника обозначена как len .

В псевдокоде процедура прорисовки салфетки выглядит просто:

```
построить «черный» треугольник ABC
ЕСЛИ order больше нуля
    построить белый треугольник MNK
    построить треугольник Серпинского порядка
    order-1 со стороной len/2 в точках A, M, K
```

Построение базового элемента салфетки Серпинского приведено на рис. 18.

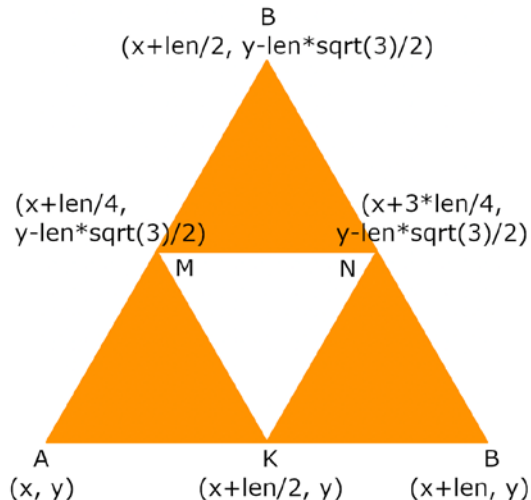


Рис. 18. Базовый элемент салфетки Серпинского

Добавим рекурсивную функцию для рисования салфетки Серпинского.

```
def Sierpinski(order, x, y, length):
    s3d2=math.sqrt(3)/2
    #строим цветной треугольник ABC
    #вычисляем координаты вершин треугольника
    points=[x, y,
            x+length/2, y-length*s3d2,
            x+length, y]
    canvas.create_polygon(points,
                          outline=myColor, fill=myColor,
                          width=myPenWidth.get())
    #теперь будем «выбрасывать»
    #средние треугольники
    if (order>0):
        #рисуем треугольник MNK цветом фона
        points=[x+length/4, y-length*s3d2/2,
                x+3*length/4, y-length*s3d2/2,
                x+length/2, y]
        canvas.create_polygon(points,
                              outline='#fff', fill='#fff',
                              width=myPenWidth.get())
        #рекурсивно вызываем функцию
        Sierpinski(order-1, x, y, length/2);
        #в точке A
        Sierpinski(order-1, x+length/2,
                    y, length/2); #в точке K
        Sierpinski(order-1, x+length/4,
                    y-length*s3d2/2, length/2);
        # в точке M
```

Теперь в обработчике события в функции *draw* надо вызвать эту рекурсивную функцию, например, так:

```
Sierpinski(myCurvePower.get(),0,
```

```

canvas.winfo_height()-20,
canvas.winfo_width()-10)

```

Попробуйте построить салфетку Серпинского разных порядков (рис. 19).

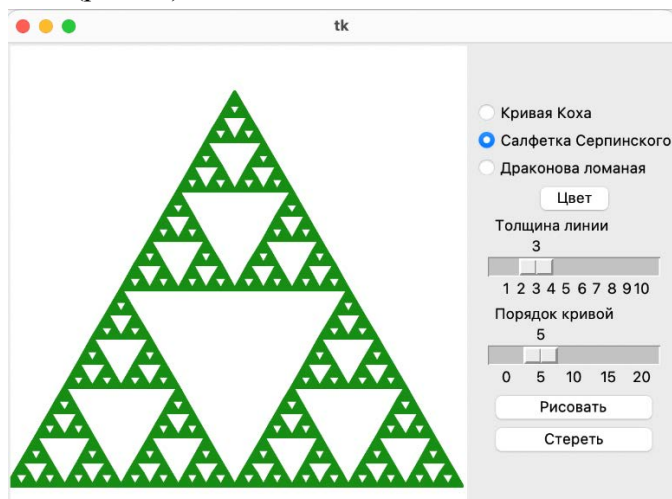


Рис. 19. Салфетка Серпинского 5 порядка

Черчение драконовой ломаной — задача более сложная. Рассмотрим, как ломаная N -ного порядка строится на основе ломаной порядка $N-1$ (рис. 20).

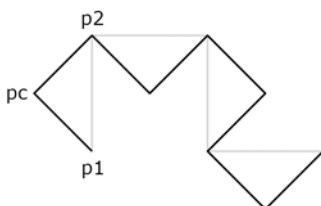


Рис. 20. Построение драконовой ломаной

При увеличении порядка ломаной каждый ее отрезок заменяется двумя отрезками, расположенными под прямым углом. Первая пара отрезков находится слева от сегмента ломаной предыдущего порядка, вторая пара — справа, третья — снова слева, четвертая справа и т.д. Построение ломаной можно свести к получению массива точек ломаной $N-20$ порядка из массива точек ломаной $N-1-20$ порядка. Т. е., как показано на рис. 20, зная пару точек (p_1, p_2) , можно вычислить координаты точки pc для следующего порядка, не забывая о чередовании положения точки pc слева-справа от отрезка ломаной предыдущего порядка. Полученный в итоге массив точек можно просто соединить линиями, чтобы получился дракон.

Опишем рекурсивную функцию, которая формирует список точек нашего дракона.

```
def getDragonPoints(order):
    x=canvas.wininfo_width()/5
    y=canvas.wininfo_height()/2
    if (order==0):
        #ломаная нулевого порядка
        #состоит из одного сегмента
        res = []
        res.append(x)
        res.append(y+x/2)
        res.append(canvas.wininfo_width()-x)
        res.append(y+x/2)
        return res
    prevRes=getDragonPoints(order-1)
    res=[]
    #направление: 1 - влево, -1 - вправо
    DirSign=1
    #начальная точка ломаной не изменяется
    res.append(prevRes[0])
    res.append(prevRes[1])
```

```

for i in range(0, len(prevRes)-3, 2):
    #считаем очередной сегмент ломаной
    p1x=prevRes[i]
    p1y=prevRes[i+1]
    p2x=prevRes[i+2]
    p2y=prevRes[i+3]
    alpha = math.atan2(p2y - p1y, p2x - p1x)-
        DirSign*math.pi/4
    R = math.sqrt(((p1x - p2x)*(p1x - p2x) +
        (p1y - p2y)*(p1y - p2y))/2)
    #найдем новую точку ломаной
    pcx=p1x+R*math.cos(alpha)
    pcy=p1y+R*math.sin(alpha)
    #добавляем ее и конечную точку
    #в список точек ломаной
    res.append(pcx)
    res.append(pcy)
    res.append(p2x)
    res.append(p2y)
    #меняем направление
    DirSign *=-1
return res

```

Так как в списке храним поочередно координаты x и y для каждой точки, обратите внимание на обход списка точек в цикле *for*.

Теперь в обработчике события щелчок по кнопке «Рисовать» во фрагменте кода, ответственном за рисование дракона, надо получить вектор точек драконовой ломаной и потом между каждой парой соседних точек нарисовать линии. Например, так:

```

points=getDragonPoints(myCurvePower.get())
canvas.create_line(points, fill=myColor,
    width=myPenWidth.get())

```

Запустите программу на выполнение и постройте драконов разного порядка. Драконы высоких порядков ($order \geq 12$) выглядят уже очень красиво (рис. 21):

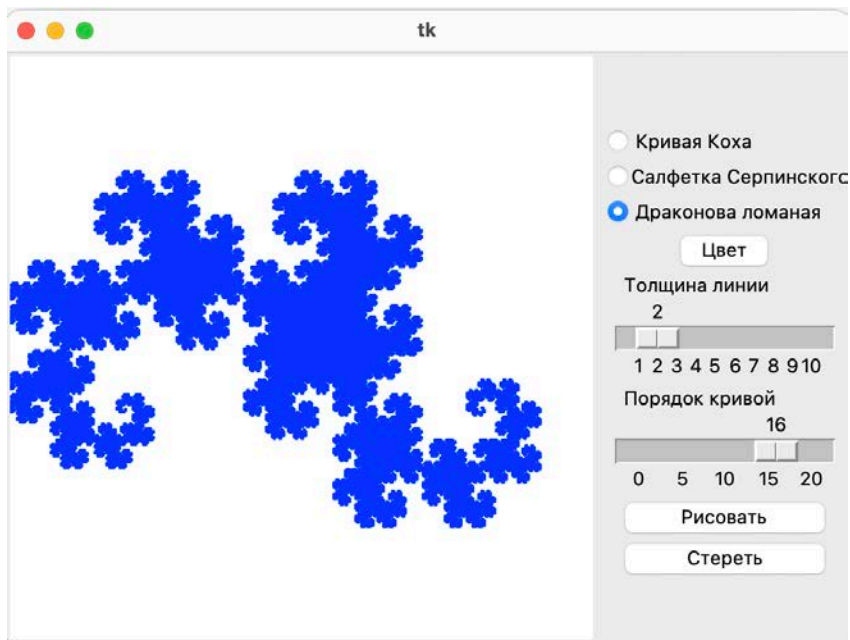


Рис. 21. Драконова ломаная 16 порядка

7.1 Контрольные вопросы

1. Что такое фрактал? Что такое самоподобная кривая?
2. Что такое генератор самоподобной кривой? Поясните на примере построения любой самоподобной кривой.
3. Какие способы компоновки виджетов вы знаете? Как компоновать компоненты при помощи фреймов? Что такое позиционирование виджетов при помощи якоря?
4. Какие виджеты и методы виджетов используются для рисования?

5. Как вызвать диалоговое окно выбора цвета? Как задать пользовательский цвет?

6. Поясните принципы построения рекурсивных функций для прорисовки самоподобных кривых (Коха, салфетки Серпинского, драконовой ломаной, кривой по своему варианту).

7. Для чего используются ассоциированные переменные? Предложите вариант использования ассоциированной переменной для выбора цвета.

7.2 Задания

Часть 1. Работа с заготовкой проекта.

1. Доделайте программу, чтобы в ней можно было строить не только кривую Коха, но и салфетку Серпинского, и драконову ломаную.

2. Измените проект так, чтобы вместо кривой Коха строилась снежинка Коха.

3. Модифицируйте программу так, чтобы порядок кривой вводился не при помощи шкалы, а, например, из текстового поля. Дайте пользователю возможность задавать начальные и конечные координаты кривых (либо координату точки и длину стороны для салфетки Серпинского).

Часть 2. Индивидуальное задание.

Дополните имеющийся проект (например, добавьте еще одну радиокнопку) рисованием самоподобной кривой в соответствии с вариантом.

Вариант 1

Канторово множество. Для канторова множества порядка K надо зарисовать все предыдущие K отрезков, чтобы наглядно проиллюстрировать процесс построения множества

Вариант 2

Фрактал Чезаро (разновидность кривой Коха, но угол между отрезком (X_a, Y_a, X_c, Y_c) и основанием не 60 градусов, а больше (но не превышает 90)).

Вариант 3

Квадратичная кривая Коха (вид 1 — на рис. 22 показаны основа и генератор, первые 2 итерации).



Рис. 22. Квадратичная кривая Коха первого вида

Вариант 4

Квадратичная кривая Коха (вид 2 — на рис. 23 показаны основа и генератор, первые 2 итерации).

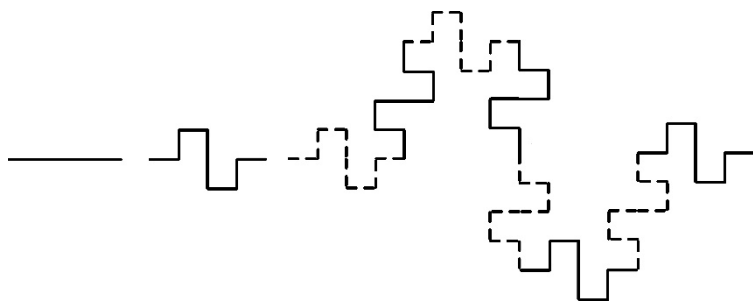


Рис. 23. Квадратичная кривая Коха второго вида

Вариант 5

Ковер Серпинского.

Вариант 6

Кривая Леви.

Вариант 7

Кривая Гильберта.

Вариант 8

Кривая Мура (*Moore curve*).

Вариант 9

Кривая Госпера.

Вариант 10

Н-дерево.

Вариант 11

Пифагорово дерево.

Вариант 12

Фрактал Вишека (*Vicsek fractal*).

Вариант 13

Т-квадрат.

Вариант 14

Шкура двойного дракона (вариант кривой Коха с генератором, показанном на рис. 24, средняя линия вдвое длиннее крайних).

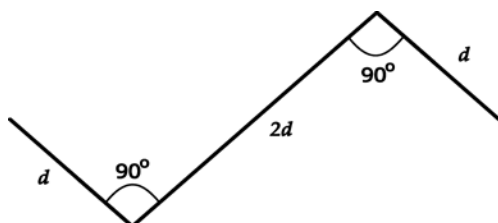


Рис. 24. Генератор шкуры двойного дракона

Вариант 15

«Перекошенная снежинка» – вариант драконовой ломаной, но угол между отрезками не 90 градусов, а 120 (рис. 25).

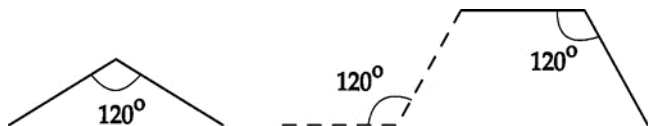


Рис. 25. Генератор драконовой ломаной с углом 120 градусов

Список использованной литературы

1. Matplotlib: визуализация с Python: официальный сайт. – The Matplotlib development team, 20 – 2024. – URL: <https://matplotlib.org> (дата обращения 10.09.2024).
2. Документация NumPy: официальный сайт. – NumPy Developers, 2008 – 2024. – URL: <https://numpy.org/doc/stable/> (дата обращения 10.09.2024).
3. Лутц, М. Изучаем Python. Том 2: учебное пособие / М. Лутц. – 5-е изд. – Санкт-Петербург: ООО «Диалектика», 2020. – 720 с.
4. Мандельброт, Б. Фрактальная геометрия природы: учебное пособие / Б. Мандельброт. – Москва: Институт компьютерных исследований, 2002. – 656 с.
5. Нуньес-Иглесиас, Х. Элегантный SciPy: учебное пособие / Х. Нуньес-Иглесиас, Ш. Уолт ван дер, Х. Дэшноу. – Москва: ДМК Пресс, 2018. – 266 с.
6. Хилл, К. Научное программирование на Python: учебное пособие / К. Хилл. – Москва: ДМК Пресс, 2021. – 646 с.

ДЛЯ ЗАМЕТОК

ДЛЯ ЗАМЕТОК

Учебное издание

*Русакова Маргарита Сергеевна,
Розачева Елена Валерьевна*

**МОДЕЛИРОВАНИЕ ФИЗИЧЕСКИХ ПРОЦЕССОВ
И ВИЗУАЛИЗАЦИЯ ДАННЫХ СРЕДСТВАМИ PYTHON**

Практикум

Редакционно-издательская обработка
издательства Самарского университета

Подписано в печать 27.11.2024. Формат 60х84 1/16.
Бумага офсетная. Печ. л. 4,75.
Тираж 27 экз. Заказ №

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«САМАРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
УНИВЕРСИТЕТ ИМЕНИ АКАДЕМИКА С.П. КОРОЛЕВА»
(САМАРСКИЙ УНИВЕРСИТЕТ)
443086, Самара, Московское шоссе, 34.

Издательство Самарского университета.
443086, Самара, Московское шоссе, 34.