МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«КУРГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Кафедра программного обеспечения
автоматизированных систем

ИЗУЧЕНИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ PYTHON

Методические рекомендации к выполнению лабораторных работ
для студентов направлений
01.03.01 «Математика»,
01.05.01 «Фундаментальная математика и механика»

Курган 2023

Кафедра: «Программное обеспечение автоматизированных систем».

Дисциплины: «Языки программирования»

Направления: 01.03.01 «Математика», 01.05.01 «Фундаментальная математика и механика»

Составил: канд. пед. наук, доцент А. А. Медведев.

Утверждены на заседании кафедры «30» июня 2023 г.

Печатается в соответствии с планом издания, утвержденным методическим советом университета «28» декабря 2023 г.

Лабораторная работа № 1. Действия с числами. Условные и циклические конструкции

Цель работы – изучить использование указанных конструкций при обработке числовой информации.

1 Фрагмент теории

Основные типы данных в языке программирования Python:

- **bool** – логический тип данных. Может содержать значения **True** или **False**, которые ведут себя как числа 1 и 0, соответственно;
- **NoneType** – объект со значением **None** (обозначает отсутствие значения). В логическом контексте значение **None** интерпретируется как **False**;
- **int** – целые числа. Размер числа ограничен лишь объемом оперативной памяти;
- **float** – вещественные числа;
- **complex** – комплексные числа;
- **str** – Unicode-строки;
- **list** – списки. Этот тип данных аналогичен массивам в других языках программирования;
- **tuple** – кортежи;
- **range** – диапазоны;
- **diet** – словари. Этот тип данных аналогичен ассоциативным массивам в других языках программирования;
- **set** – множества (коллекции уникальных объектов);
- **function** – функции;
- **module** – модули;
- **type** – классы и типы данных. Все данные в языке Python являются объектами, даже сами типы данных!

Для преобразования одного типа в другой используются функции, имена которых, как правило, совпадают с именами соответствующих типов.

Основные типы данных делятся на *изменяемые* и *неизменяемые*. К *изменяемым типам* относятся: списки и словари. Неизменяемые типы – это числа, строки, кортежи и диапазоны.

Кроме того, типы данных делятся на *последовательности* и *отображения*. К *последовательностям* относятся: строки, списки, кортежи, и диапазоны, а к *отображениям* – словари. Последовательность можно «обойти» с использованием цикла, в то время как отображения устанавливают зависимость между двумя объектами («ключ-значение»).

1.1 Условная конструкция

Общий вид условной конструкции следующий:

```
if <Логическое выражение>:
 <Блок, выполняемый, если условие истинно>
[elif <Логическое выражение>:
```

```
 <Блок, выполняемый, если условие истинно> ]
[else:
 <Блок, выполняемый, если все условия ложны> ]
```

Конструкции, расположенные в квадратных скобках, могут отсутствовать.

В качестве примера приведем текст программы, которая проверяет, является ли введенное число четным.

```
x = int(input("Введите число: "))
# Введенную строку перевели в целый тип
if x % 2 == 0: # Вычисляем остаток от деления 2
 print(x, " - четное число")
else:
 print(x, " - нечетное число")
```

Условный оператор имеет еще один формат:

```
 <Переменная> = <Если истина> if <Условие> else <Если ложь>
```

Перепишем программу, используя указанную конструкцию:

```
x = int (input ("Введите число: "))
s = " - четное число" if x % 2 == 0 else " - нечетное число"
print(x, s)
```

1.2 Циклические конструкции

Цикл **for** применяется для перебора элементов последовательности и имеет такой формат:

```
for <Текущий элемент> in <Последовательность>:
 <Инструкции внутри цикла>
[else:
 <Блок, выполняемый, если не использовался оператор break>]
```

Здесь присутствуют следующие конструкции:

- *Последовательность* – объект, поддерживающий механизм итерации. Например: строка, список, кортеж, диапазон, словарь и др.;
- *Текущий элемент* – на каждой итерации через этот параметр доступен текущий элемент последовательности или ключ словаря;
- *Инструкции внутри цикла* – блок, который будет многократно выполняться;
- если внутри цикла не использовался оператор **break**, то после завершения выполнения цикла будет выполнен блок в инструкции **else**. Этот блок не является обязательным.

Приведем пример программы, перебирающей буквы в слове:

```
for s in "str":
 print(s, end=" ")
else:
 print ("\nЦикл выполнен")
```

Результат выполнения программы:

```
s t r
Цикл выполнен
```

Другой циклической конструкцией является **while.** Выполнение инструкций в этом цикле продолжается до тех пор, пока логическое выражение истинно. Цикл **while** имеет следующий формат:

```
 <Начальное значение>
```

```
while <Условие>:
 <Инструкции>
 <Приращение>
[else:
 <Блок, выполняемый, если не использовался оператор break>]
```

Последовательность работы цикла **while**.

1 Переменной-счетчику присваивается начальное значение.

2 Проверяется условие и, если оно истинно, выполняются инструкции внутри цикла, иначе выполнение цикла завершается.

3 Переменная-счетчик изменяется на величину, указанную в параметре *Приращение*.

4 Переход к пункту 2.

5 Если внутри цикла не использовался оператор **break**, то после завершения выполнения цикла будет выполнен блок в инструкции **else**. Этот блок не является обязательным.

Перепишем последнюю программу, используя конструкцию **while**:

```
s = "str" # Заданная строка
i = 0 # Начальное значение переменной-счетчика
while i < len(s): # Пока счетчик меньше длины строки
 print(s[i], end=" ") # Вывод i-го символа
 i += 1 # Приращение
else:
 print ("\nЦикл выполнен")
```

1.3 Дополнительные конструкции

Перечислим конструкции, которые достаточно часто применяются совместно с циклами.

Функция range() применяется для создания диапазона значений. Ее формат:

```
range([<Начало>, ]<Конец>[, <Шаг>])
```

Здесь первый параметр задает начальное значение. Если параметр *Начало* не указан, то по умолчанию используется значение 0. Во втором параметре указывается конечное значение. Следует заметить, что это значение не входит в возвращаемые значения. Если параметр *Шаг* не указан, то используется значение 1. Функция возвращает диапазон – особый объект, поддерживающий итерационный протокол. С помощью диапазона внутри цикла **for** можно получить значение текущего элемента.

Диапазон поддерживает два полезных метода:

- **index(<*Значение*>)** – возвращает индекс элемента, имеющего указанное значение. Если значение не входит в диапазон, возбуждается исключение **ValueError**;
- **count(<*Значение*>)** – возвращает количество элементов с указанным значением. Если элемент не входит в диапазон, возвращается значение 0;

Оператор continue позволяет перейти к следующей итерации цикла до завершения выполнения всех инструкций внутри цикла.

Оператор break позволяет осуществить выход из цикла, т. е. передать управление конструкции, идущей за циклом.

2 Задачи для самостоятельного решения

1.1 Составить программу, проверяющую, будут ли взаимно просты два натуральных (целых) числа.

1.2 Составить программу, проверяющую, будет ли простым данное натуральное число.

1.3 Напишите программу, которая выводит на экран все простые числа из интервала 1..N, используя решето Эратосфена.

1.4 Написать программу, которая выводит на экран первые N простых чисел.

1.5 Найти все делители натурального числа N.

1.6 Разложить целое число на простые множители. Вывести на экран все простые множители (в порядке возрастания) и их порядки.

1.7 Натуральное число называется совершенным, если оно равно сумме всех своих делителей, включая единицу. Вывести первые N (N<5) совершенных чисел на экран.

1.8 Проверить, какие нечетные натуральные числа из интервала N..M можно представить в виде суммы трех простых чисел.

1.9 Проверить, будет ли данное число числом Фибоначчи.

1.10 Вычислить (N)!!, где

(2N)!! = 2*4*...*(2N)

(2N+1)!! = 1*3*...*(2N+1).

1.11 Найти все различные пифагоровы тройки из интервала от N до M.

1.12 Написать программу умножения (деления) двух данных рациональных чисел. Ответ должен быть несократимой дробью.

1.13 Написать программу сложения (вычитания) двух данных рациональных чисел. Ответ должен быть несократимой дробью.

1.14 Найти все целые числа из интервала от N до M, которые делятся на каждую из своих цифр.

1.15 Найти все целые числа из интервала от N до M, которые делятся на сумму всех своих цифр.

1.16 Проверить, являются ли два числа дружественными (сумма делителей одного числа равна другому числу). Например, сумма делителей числа 220 равна: 1 + 2 + 4 + 5 + 10 + 11 + 20 + 22 + 44 + 55 + 110 = 284, а сумма делителей числа 284 равна: 1 + 2 + 4 + 71 + 142 = 220, поэтому числа 220 и 284 – дружественные.

1.17 Составить программу перевода чисел из непозиционной, например, римской системы счисления в десятичную.

1.18 Определить, можно ли представить данное число в виде суммы квадратов двух (трех) целых чисел.

1.19 Определить, можно ли представить данное число в виде суммы кубов двух (трех) целых чисел.

1.20 Определить, можно ли представить число A в виде линейной целочисленной комбинации чисел B и C.

1.21 Определим операцию # так, что A#B = A –0 B + A % B. Найти все числа из интервала от N до M, для которых эта операция коммутативна.

1.22 Характером натурального числа назовем сумму всех его делителей, не равных единице и самому числу. Характером простого числа будем считать нуль. Написать программу, которая вычисляет характер числа.

1.23 Составить алгоритм, находящий по целым числам A , B и C такие целые X и Y , что A * X + B * Y = C (если такие X и Y существуют).

1.24 Составить программу поиска среди чисел n, n+1, ..., 2*n так называемых близнецов, т. е. двух простых чисел, разность между которыми равна двум.

1.25 Число из n цифр называется числом Армстронга, если сумма его цифр в степени n равна самому числу. Например: $1634 = 1^4 + 6^4 + 3^4 + 4^4$.

Составить программу, находящую все числа Армстронга из двух и трех цифр.

1.26 Составить программу, находящую все трехзначные числа abc такие, что abc = a! + b! + c! .

1.27 Гипотеза Симона о факториале состоит в следующем: только четыре факториала являются произведениями трех последовательных целых чисел. Вот два из них: 4! = 2*3*4, 5! = 4*5*6. Составить программу, находящую следующее число, обладающее указанным свойством. Сможете ли вы опровергнуть эту гипотезу?

1.28 Число называется абсолютно простым, если при любой перестановке его цифр также образуется простое число. Найти все абсолютно простые числа из интервала [n, m].

1.29 Простое число Мерсенна – это число, которое может быть представлено в виде: $2^n - 1$, где n тоже простое. Составить программу, находящую числа Мерсенна.

1.30 Сформировать и вывести на экран в порядке возрастания все числа из четырех цифр, причем внутри числа не должно быть двух одинаковых цифр. Например, такими числами являются 123(0123), 2715 и т. д. Число 24 таким не является (24 = 0024).

1.31 По заданной формуле члена ряда с номером k составить две программы:

а) программу вычисления суммы первых n членов заданного ряда;

б) программу вычисления всех членов ряда, не меньших заданного числа E.

1 $\dfrac{1}{(2k-1)(2k+1)}$.

2 $\dfrac{5}{k(k+1)+4}$.

3 $\dfrac{k}{(k+1)(k+2)}$.

4 $\dfrac{2k+1}{(2k^2+1)k}$.

5 $\dfrac{k+1}{k^2(k^2+2)}$

6 $\dfrac{8k}{3k^2+10}$.

7 $\dfrac{5k}{k^2+81}$.

8 $\dfrac{3k-1}{7k^2+9}$.

9 $\dfrac{k+0{,}5}{3k^2+2}$.

10 $\dfrac{1}{k^2+15}$.

11 $\dfrac{1}{k^2+3k+4}$.

12 $\dfrac{k+1}{k(k+2)(k+3)}$.

13 $\frac{k+2}{k(k^2+4)}$.

14 $\frac{4k}{5k^2+8k-1}$.

15 $\frac{k+4}{(k+2)(k+8)}$.

16 $\frac{2k}{3k^2+k+4}$.

17 $\frac{2}{k(k+3)}$.

18 $\frac{3}{(k+1)(k+7)}$.

19 $\frac{2}{2k^2+5k-1}$.

20 $\frac{3}{2k^2+5}$.

21 $\frac{4}{k^2+3k+1}$.

22 $\frac{2}{2k(k-0,2)}$.

23 $\frac{5,6}{k(3k+0,75)}$.

24 $\frac{2}{4k^2-2k-1}$.

1.32 Дано натуральное число. Найти сумму и количество четных и нечетных цифр в нем.

1.33 Дано натуральное число. Получить и вывести число, у которого цифры расположены в обратном порядке, например: дано число 2354, получить число 4532.

1.34 Дано число N (N<=40). Составить программу нахождения всех делителей всех натуральных чисел от 1 до N.

1.35 Дано число N (N<=40). Составить программу нахождения четных делителей всех натуральных чисел от 1 до N.

1.36 Натуральное число называется совершенным, если оно равно сумме всех своих делителей, включая единицу. Число, меньшее суммы своих делителей, назовем избыточным, а число, более суммы своих делителей, – неполным. Найти N избыточных и M неполных чисел.

1.37 Дана сумма денег N (N<100). Нужно представить ее как можно меньшим количеством монет достоинством 10, 5, 2 и 1 рубль.

1.38 Задано натуральное число N. Найти количество натуральных чисел, не превосходящих N и не делящихся ни на одно из чисел 2, 3, 5.

1.39 Два двузначных числа, записанных одно за другим, образуют четырёхзначное число, которое делится на их произведение. Найти эти числа.

1.40 Даны два двузначных натуральных числа A и B. Из этих чисел составили 2 четырехзначных числа: первое число получили путем написания сначала числа A, затем B. Для получения второго числа сначала записали число B, затем A. Найти числа A и B если известно, что первое четырехзначное число нацело делится на 99, а второе на 49.

1.41 Из натурального числа удалить заданную цифру. Число и цифру вводить с клавиатуры. Например, задано число 5683. Требуется удалить из него цифру 8. Получится число 563.

1.42 Найти сумму первой и последней цифр любого целого положительного числа.

1.43 Дано целое число, состоящее из разных цифр. Определить, какая из цифр заданного числа больше, т. е. найти наибольшую цифру числа.

1.44 Дана последовательность чисел, состоящих только из цифр 0 и 5, в порядке возрастания: 0 5 50 55 500 505 5000 5005 5050 и так далее. Найти k-ое по порядку в этой последовательности число.

1.45 Написать программу, которая проверяла бы, что каково бы ни было натуральное число n, среди первых $n^2 - 1$ чисел Фибоначчи найдется хотя бы одно, делящееся на n.

1.46 Проверка гипотезы Сиракуз. Возьмем любое натуральное число. Если оно четное – разделим его пополам, если нечетное – умножим на 3, прибавим 1 и разделим пополам. Повторим эти действия с вновь полученным числом. Гипотеза гласит, что независимо от выбора первого числа рано или поздно мы получим 1. Проверьте гипотезу для всех чисел от 20 до 30.

1.47 Даны два числа. Определить цифры, входящие в запись, как первого, так и второго чисел.

1.48 Старинная задача. Сколько можно купить быков, коров и телят, если плата за быка – 10 рублей, за корову – 5 рублей, за теленка – полтинник (0,5 рубля), если на 100 рублей надо купить 100 голов скота.

1.49 Написать программу, которая находит и выводит на печать все четырехзначные abcd, числа a, b, c, d – различные цифры, для которых выполняется: ab-cd=a+b+c+d.

1.50 Если мы сложим все цифры какого-либо числа, затем все цифры найденной суммы и будем повторять много раз, мы, наконец, получим однозначное число (цифру), называемое цифровым корнем данного числа. Например, цифровой корень числа 34697 равен 2 (3+4+6+9+7=29; 2+9=11; 1 + 1=2). Составить программу для нахождения цифрового корня натурального числа.

Лабораторная работа № 2. Преобразование символьных величин

Цель работы – познакомиться с основными конструкциями, используемыми для обработки строковой информации.

1 Фрагмент теории

Строки представляют собой упорядоченные последовательности символов. Длина строки ограничена лишь объемом оперативной памяти компьютера. Как и все последовательности, строки поддерживают обращение к элементу по индексу, получение среза, конкатенацию (оператор «+»), повторение (оператор «*»), проверку на вхождение (операторы **in** и **not in**).

К любому символу строки можно обратиться как к элементу списка – достаточно указать его индекс в квадратных скобках. Нумерация начинается с нуля. Если символ, соответствующий указанному индексу, отсутствует в строке, то возбуждается исключение **IndexError**. В качестве индекса можно указать отрицательное значение. В этом случае смещение будет отсчитываться от конца строки, а точнее – чтобы получить положительный индекс, значение вычитается из длины строки.

Так как строки относятся к неизменяемым типам данных, то изменить символ по индексу нельзя. Чтобы выполнить изменение, можно воспользоваться

операцией извлечения среза, которая возвращает указанный фрагмент строки. Формат операции:

```
[<Начало>:<Конец>:<Шаг>]
```

Все параметры здесь не являются обязательными. Если параметр *Начало* не указан, то используется значение 0. Если параметр *Конец* не указан, то возвращается фрагмент до конца строки. Следует также заметить, что символ с индексом, указанным в этом параметре, не входит в возвращаемый фрагмент. Если параметр *Шаг* не указан, то используется значение 1. В качестве значения параметров можно указать отрицательные значения.

Перечислим основные функции для работы со строками:

- **str([<*Объект*>])** – преобразует любой объект в строку. Если параметр не указан, то возвращается пустая строка;
- **len(<*Строка*>)** – возвращает количество символов в строке.

Основные методы для работы со строками:

- **strip([<*Символы*>])** – удаляет указанные в параметре символы в начале и в конце строки. Если параметр не задан, удаляются пробельные символы: пробел, символ перевода строки («\n»), символ возврата каретки («\r»), символы горизонтальной («\t») и вертикальной («\v») табуляции;
- **split([<*Разделитель*>[, <*Лимит*>]])** – разделяет строку на подстроки по указанному разделителю и добавляет эти подстроки в список, который возвращается в качестве результата. Если первый параметр не указан или имеет значение **None**, то в качестве разделителя используется символ пробела. Во втором параметре можно задать количество подстрок в результирующем списке – если он не указан или равен –1, в список попадут все подстроки. Если подстрок больше указанного количества, то список будет содержать еще один элемент – с остатком строки;
- **rsplit([<*Разделитель*>[, <*Лимит*>]])** – аналогичен методу **split()**, но поиск символа-разделителя производится не слева направо, а, наоборот, справа налево;
- **splitlines([True])** – разделяет строку на подстроки по символу перевода строки («\n») и добавляет их в список. Символы новой строки включаются в результат, только если необязательный параметр имеет значение **True**. Если разделитель не найден в строке, то список будет содержать только один элемент;
- **partition(<*Разделитель*>)** – находит первое вхождение символа-разделителя в строку и возвращает кортеж из трех элементов: первый элемент будет содержать фрагмент, расположенный перед разделителем, второй элемент – сам разделитель, а третий элемент – фрагмент, расположенный после разделителя. Поиск производится слева направо. Если символ-разделитель не найден, то первый элемент кортежа будет содержать всю строку, а остальные элементы останутся пустыми;
- **join()** – преобразует последовательность в строку. Элементы добавляются через указанный разделитель. Формат метода:

```
<Строка> = <Разделитель>.join(<Последовательность>)
```

- **find()** – ищет подстроку в строке. Возвращает номер позиции, с которой начинается вхождение подстроки в строку. Если подстрока в строку не входит, то возвращается значение -1. Метод зависит от регистра символов. Формат метода:

```
<Строка>.find (<Подстрока> [, <Начало> [, <Конец>] ])
```

Если начальная позиция не указана, то поиск будет осуществляться с начала строки. Если параметры *Начало* и *Конец* указаны, то производится операция извлечения среза и поиск подстроки будет выполняться в этом фрагменте;

- **index()** – метод аналогичен методу **find()**, но если подстрока в строку не входит, то возбуждается исключение **ValueError**. Формат метода:

```
<Строка>.index(<Подстрока>[, <Начало>[, <Конец>]])
```

- **count()** – возвращает число вхождений подстроки в строку. Если подстрока в строку не входит, то возвращается значение 0. Метод зависит от регистра символов. Формат метода:

```
<Строка>.count(<Подстрока>[, <Начало>[, <Конец>]])
```

- **replace()** – производит замену всех вхождений заданной подстроки в строке на другую подстроку и возвращает результат в виде новой строки. Метод зависит от регистра символов. Формат метода:

```
<Строка>.replace(<Подстрока для замены>, <Новая подстрока>
[, <Максимальное количество замен>]
```

Если количество замен не указано, будет выполнена замена всех найденных подстрок.

2 Задачи для самостоятельного решения

2.1 Дана строка символов. Сформировать новую строку, состоящую из символов с номерами три, шесть, девять и т. д. данной строки.

2.2 Даны две строки. Вывести большую по длине строку столько раз, на сколько символов отличаются строки.

2.3 Дана строка символов. Определите, какой из символов «s» или «q» встречается в ней раньше. Если какой-то из символов в строке отсутствует, то сообщить об этом.

2.4 Написать программу, которая считает число слов в предложении.

2.5 Написать программу, которая считает число слов в предложении, начинающихся на заданную букву.

2.6 Написать программу, которая проверяет, можно ли из букв, входящих в слово A, составить слово B.

2.7 Дана строка, в которой нет начальных и конечных пробелов. Необходимо изменить ее так, чтобы длина строки стала равна заданной длине, большей, чем текущая длина строки. Это следует сделать путем вставки между словами дополнительных пробелов. Количество пробелов между отдельными словами не должно отличаться более чем на один пробел (то есть пробелы добавляются равномерно).

2.8 Написать программу, которая удаляет в данном тексте все лишние пробелы.

2.9 Составить список слов, которые входят в данное предложение.

2.10 Подсчитать, из скольких разных слов состоит данное предложение.

2.11 Подсчитать число различных гласных (согласных), входящих в данное слово (предложение).

2.12 Упорядочить данный набор слов по алфавиту.

2.13 Написать все предложения, которые можно составить из слов: «ваши прекрасные глаза», «прекрасная маркиза», «от любви», «сулят», «мне», «смерть» путем их всевозможных перестановок (данная ситуация обыгрывается в пьесе Мольера «Мещанин во дворянстве»).

2.14 Проверьте правильность расстановки скобок в выражении. Алгоритм может быть, например, таким: при подсчете скобок слева направо число открывающих скобок не должно быть меньше числа закрывающих, причем при завершении подсчета эти числа должны совпадать.

2.15 Из данного предложения выбрать слова, имеющие заданное число букв.

2.16 Составить программу, подсчитывающую число предложений в тексте (предложение оканчивается символами «.», «?», «!»).

2.17 Составить программу, вычеркивающую из предложения данное слово.

2.18 Сформировать и вывести на экран все возможные переносы данного слова. Перенос почти всегда будет выполняться правильно, если пользоваться такими правилами:

а) две идущие подряд гласные можно разделить, если первой из них предшествует согласная, а за второй идет хотя бы одна буква (буква «й» при этом рассматривается с предшествующей гласной как единое целое);

б) две идущие подряд согласные можно разделить, если первой из них предшествует гласная, а в той части слова, которая идет за второй согласной, имеется хотя бы одна гласная (буквы «ь», «ъ» при этом рассматриваются как единое целое с предшествующей согласной);

в) если правила, указанные в пунктах (а) и (б) применить невозможно, то следует попытаться разбить слово так, чтобы первая часть содержала более чем одну букву, и оканчивалась на гласную, а вторая содержала хотя бы одну гласную.

2.19 Разбить данное слово на слоги.

2.20 Из заданного набора слов выбрать все слова, имеющие рифмы (рифма определяется по принципу, придуманному Незнайкой: два слова рифмуются, если последние слоги у них совпадают, например, «палка – селедка»).

2.21 Заменить в данном тексте все малые латинские буквы на заглавные (то же самое для русских букв).

2.22 Перепечатать заданный текст, удалив из него знаки «+», непосредственно за которыми идет цифра.

2.23 Перепечатать заданный текст, удалив из него все буквы «b», непосредственно перед которыми находится буква «c».

2.24 Написать программу, генерирующую строки длины 10, причем первые 4 символа – цифры, следующие два символа – различные буквы, следующие 4 символа – нули или единицы, причем одна единица точно присутствует.

2.25 Дана непустая последовательность непустых слов из латинских букв; соседние слова отделяются друг от друга запятой, за последним словом – точка. Определить количество слов, которые содержат ровно три буквы «е».

2.26 Преобразовать выражение (т. е. текст специального вида), составленное из цифр и знаков четырех арифметических операций (сложения, вычитания, умножения и деления), в постфиксную форму. В постфиксной форме сначала записываются операнды, а затем знак операции, например:

Обычная запись	Постфиксная запись
3+4	34+
(5–4)+2	54–2+
2*(3+4)*5	234+*5*

2.27 Возьмем произвольное слово и проделаем с ним следующую операцию: поменяем местами его первую согласную букву с последней согласной буквой, вторую согласную букву с предпоследней согласной буквой и т. д. Если после этой операции мы вновь получим исходное слово, то будем называть такое слово негласным палиндромом. Например, слова «sos»,» rare», «rotor», «gong», «karaoke» являются негласными палиндромами. Определить, является ли заданное слово негласным палиндромом.

2.28 Дан текст. Если первый символ текста не является малой латинской буквой, то оставить его без изменения. Если же это малая латинская буква, но за начальной группой малых латинских букв не следует цифра, то также оставить текст без изменения. Иначе каждую цифру, принадлежащую группе цифр, следующей за начальной группой малых латинских букв, заменить символом «*».

2.29 Дан текст, каждый символ которого может быть малой буквой, цифрой или одним из знаков «+», « –», «*». Группой букв будем называть такую совокупность последовательно расположенных букв, которой непосредственно не предшествует и за которой непосредственно не следует буква. Аналогично определим группу цифр и группу знаков. Если в данном тексте имеется не менее двух групп букв, то каждый знак «+», встречающийся между двумя группами букв, заменить цифрой 1, знак « – » заменить цифрой 2, а знак «*» – цифрой 3. Иначе оставить текст без изменений.

2.30 Дана строка, состоящая из слов, разделенных символами, которые перечислены во второй строке. Получить все слова.

2.31 С клавиатуры вводятся две строки. Найти количество вхождений одной (являющейся подстрокой) в другую.

2.32 Во введенной строке удалить пробелы между первым и вторым вопросительным знаком.

2.33 Дана строка слов, разделенных пробелами. Между словами может быть несколько пробелов, в начале и конце строки также могут быть пробелы. Требуется преобразовать строку так, чтобы в ее начале и конце пробелов не было, а слова были разделены одиночным символом «*» (звездочка).

2.34 Найти слово, стоящее в тексте под определенным номером, и вывести его первую букву.

2.35 Дана строка символов. Группы символов, разделенные одним или несколькими пробелами и не содержащие пробелов внутри себя, будем называть словами. В самом длинном слове заменить все буквы «a» на «b».

2.36 Дана строка, состоящая из слов, разделенных пробелами и знаками препинания. Определить длину самого короткого слова.

2.37 Дана строка слов. Заменить последние три символа слов, имеющих выбранную длину на символ «$».

2.38 В произвольной строке удалить последнее слово.

2.39 Добавить в строку пробелы после знаков препинания, если они там отсутствуют.

2.40 Найти в строке определенную последовательность символов (подстроку) и заменить ее другой.

2.41 Дана строка из символов латинского алфавита. Вставьте пробел перед каждой заглавной буквой. Перед первой буквой пробел добавлять не надо. Пример. Исходная строка: AtTimesYouMayWantToReadDataFromTheKeyBoard

Полученная строка: At Times You May Want To Read Data From The Key Board

2.42 Дана строка. Подсчитать общее количество содержащихся в ней строчных латинских и русских букв.

2.43 Дана строка. Если она представляет собой запись целого числа, то вывести 1, если вещественного (с дробной частью) − вывести 2; если строку нельзя преобразовать в число, то вывести 0.

2.44 Дана строка, изображающая целое положительное число. Вывести знакочередующуюся сумму цифр этого числа. Пример: для числа 294 сумма равна –3 (2–9+4=-3).

2.45 Дана строка, содержащая натуральное число. Получить строку, содержащую соответствующее число в двоичной системе счисления.

2.46 Дана строка, содержащая натуральное число, представленное в двоичной системе счисления. Получить строку, содержащую соответствующее число в десятичной системе счисления.

2.47 Даны целые положительные числа N1 и N2 и строки S1 и S2. Получить из этих строк новую строку, содержащую первые N1 символов строки S1 и последние N2 символов строки S2 (в указанном порядке).

2.48 Дан символ C и строки S, S0. Перед каждым вхождением символа C в строку S вставить строку S0.

Лабораторная работа № 3. Структуры данных. Списки

Цель работы – познакомиться с основными конструкциями, используемыми для работы со списками.

1 Фрагмент теории

Списки, кортежи, множества и *диапазоны* – это нумерованные наборы объектов. Каждый элемент набора содержит лишь *ссылку на объект* – по этой причине они могут содержать объекты произвольного типа данных и иметь неограниченную степень вложенности. Позиция элемента в наборе задается *индексом*. Нумерация элементов начинается с 0, а не с 1.

Списки и *кортежи* являются просто упорядоченными последовательностями элементов. Как и все последовательности, они поддерживают обращение к элементу по индексу, получение среза, конкатенацию (оператор «+»), повторение (оператор «*»), проверку на вхождение (оператор **in**) и не вхождение (оператор **not in**).

Списки относятся к *изменяемым типам данных*. Это означает, что мы можем не только получить элемент по индексу, но и изменить его.

Кортежи относятся к *неизменяемым типам данных*. Иными словами, можно получить элемент по индексу, но изменить его нельзя, т. е. *кортеж – это неизменяемый список*.

Для добавления и удаления элементов списка используются следующие методы:

- **append(<*Объект*>)** – добавляет один объект в конец списка. Метод изменяет текущий список и ничего не возвращает;
- **extend(<*Последовательность*>)** – добавляет элементы последовательности в конец списка. Метод изменяет текущий список и ничего не возвращает;
- **insert (<*Индекс*>, <*Объект*>)** – добавляет один объект в указанную позицию. Остальные элементы смещаются. Метод изменяет текущий список и ничего не возвращает;
- **pop([<*Индекс*>])** – удаляет элемент, расположенный по указанному индексу, и возвращает его. Если индекс не указан, то удаляет и возвращает последний элемент списка. Если элемента с указанным индексом нет, или список пустой, возбуждается исключение **IndexError**;
- **remove(<*Значение*>)** – удаляет первый элемент, содержащий указанное значение. Если элемент не найден, возбуждается исключение ValueError. Метод изменяет текущий список и ничего не возвращает;
- **clear()** – удаляет все элементы списка, очищая его. Никакого результата при этом не возвращается.

Укажем еще ряд функций и методов, которые будут полезны при работе со списками.

Чтобы узнать индекс элемента внутри списка, следует воспользоваться методом **index()**. Формат метода:

```
index(<Значение>[, <Начало>[, <Конец>]])
```

Метод возвращает индекс элемента, имеющего указанное значение. Если значение не входит в список, то возбуждается исключение **ValueError**. Если второй и третий параметры не указаны, то поиск будет производиться с начала и до конца списка.

Узнать общее количество элементов с указанным значением позволяет метод **count(<*Значение*>)**. Если элемент не входит в список, то возвращается значение 0.

С помощью функций **max()** и **min()** можно узнать максимальное и минимальное значение списка, соответственно.

Метод **reverse()** изменяет порядок следования элементов списка на противоположный. Метод изменяет текущий список и ничего не возвращает.

Отсортировать список позволяет метод **sort()**. Он имеет следующий формат:

```
sort([key=None][, reverse=False])
```

Все параметры не являются обязательными. Метод изменяет текущий список и ничего не возвращает.

Чтобы отсортировать список по убыванию, следует в параметре **reverse** указать значение **True.**

В параметре **key** можно указать функцию, выполняющую какое-либо действие над каждым элементом списка. В качестве единственного параметра она должна принимать значение очередного элемента списка, а в качестве результата – возвращать результат действий над ним. Этот результат будет участвовать в процессе сортировки, но значения самих элементов списка не изменятся.

2 Задачи для самостоятельного решения

3.1 Найдите модуль разности первого и последнего элементов числового списка.

3.2 Определите индекс элемента числового списка, значение которого равно сумме первого и последнего элемента того же списка.

3.3 Определите, равен ли «центральный» элемент числового списка произведению крайних элементов.

3.4 Определите, является ли произведение элементов числового списка факториалом числа, равного длине списка.

3.5 Определите, все ли элементы числового списка являются трёхзначными числами.

3.6 Определите, все ли элементы числового списка являются чётными числами.

3.7 По заданному списку целых чисел создайте список, состоящий из их остатков от деления на 2.

3.8 Определите, встречается ли указанный своим индексом элемент первого заданного списка во втором заданном списке.

3.9 Составьте программу, которая по двум спискам, имеющим одинаковую длину, возвращает:

(1) первый список, если сумма элементов первого списка совпадает с суммой элементов второго списка;

(2) сумму элементов второго списка в противном случае.

3.10 Напишите программу, определяющую в списке, содержащем списки целых чисел, количество таких списков, содержащих заданное число.

3.11 (Покер) Задан список из пяти чисел. Если одинаковы 5 чисел, то напечатайте число 1, иначе если одинаковы 4 числа, то напечатайте число 2, иначе ес-

ли одинаковы 3 и 2 числа, то выведите число 3, если одинаковы 3 числа, то результатом работы программы будет число 4, если одинаковы 2 и 2 числа, то выведите число 5, если одинаковы 2 числа, то результатом работы программы будет число 6, иначе – число 7.

3.12 Напишите программу, удаляющую «крайние» элементы списка.

3.13 Напишите программу, меняющую местами «крайние» элементы списка.

3.14 Напишите программу, меняющую местами пару элементов списка по их указанным индексам.

3.15 Напишите программу, добавляющую в начало списка копию его предпоследнего элемента.

3.16 Напишите программу, удваивающую первый элемент списка (путём создания копии), если его значение меньше значения второго элемента.

3.17 Напишите программу, объединяющую первую «половину» элементов первого списка со второй «половиной» элементов второго списка.

3.18 Напишите программу, объединяющую три заданных списка в один, после предварительного удаления из них последних элементов.

3.19 Напишите программу, конструирующую список из «центральных» элементов трёх заданных списков.

3.20 Напишите программу, объединяющую два заданных списка, если сумма максимального элемента первого списка и минимального элемента второго списка равна заданному числу; в противном случае возвращается пустой список.

3.21 Напишите программу, конструирующую список из заданных списков $[A_1,A_2,...,A_n]$ и $[B_1,B_2,...,B_n]$ следующим образом:

(1) если $A_1=B_n$ и $A_n=B_1$, то $[A_2,...,A_{n-1},B_2,...,B_{n-1}]$;

(2) если $A_1=B_n$ и $A_n\neq B_1$, то $[A_2,...,A_n,B_1,...,B_{n-1}]$;

(3) если $A_1\neq B_n$ и $A_n=B_1$, то $[A_1,...,A_{n-1},B_2,...,B_n]$;

(4) в противном случае $[A_1,...,A_n,B_1,...,B_n]$.

3.22 Напишите программу, конструирующую список из двух заданных таким образом, что из первого списка берутся только отрицательные числа, а из второго – только положительные.

3.23 Напишите программу, конструирующую список путём утраивания его каждого элемента.

3.24 Дан список целых чисел. Упорядочьте по возрастанию только положительные числа.

3.25 Дан список целых чисел. Упорядочьте по возрастанию только элементы с четными порядковыми номерами в списке.

3.26 Даны два упорядоченных по не возрастанию списка. Объедините их в новый упорядоченный по не возрастанию список.

3.27 Если в списке есть отрицательные элементы, найти наибольший из них.

3.28 Найти все числа, каждое из которых встречается в двух заданных списках.

3.29 В заданном списке определить максимальное количество подряд идущих положительных элементов, не прерываемых ни нулями, ни отрицательными элементами.

3.30 Задан список A, состоящий из N целых чисел. Написать программу определения значения k, при котором величина $|A_1+...+A_k-A_{k+1} -... -A_N|$ минимальна.

3.31 Дан список A из k элементов, каждый из которых является списком из m целых чисел. Построить список из m элементов, каждый элемент которого равен сумме элементов, стоящих на соответствующих местах в каждом из k элементов списка A.

3.32 Дан список из k элементов, каждый из которых является списком из целых чисел. Получить список, составленный из средних арифметических каждого элемента исходного списка.

3.33 Дан список из k элементов, каждый из которых является списком из целых чисел. В каждом из k элементов исходного списка найти наибольший элемент и поменять его местами с элементом, номер которого совпадает с номером элемента в исходном списке.

3.34 Дан список из k элементов, каждый из которых является списком из целых чисел. Построить список из чисел, каждое из которых встречается в каждом элементе исходного списка.

3.35 Дан список $A=[A_1,A_2,...,A_{21}]$. Получить список списков:

1-й элемент:	A_1	0	0	0	0	0
2-й элемент:	A_2	A_3	0	0	0	0
3-й элемент:	A_4	A_5	A_6	0	0	0
4-й элемент:	A_7	A_8	A_9	A_{10}	0	0
5-й элемент:	A_{11}	A_{12}	A_{13}	A_{14}	A_{15}	0
6-й элемент:	A_{16}	A_{17}	A_{18}	A_{19}	A_{20}	A_{21}

3.36 Построить список списков по схеме:

1-й элемент:	0	0	0	1	0	0	0
2-й элемент:	0	0	1	1	1	0	0
3-й элемент:	0	1	1	1	1	1	0
4-й элемент:	1	1	1	1	1	1	1
5-й элемент:	0	1	1	1	1	1	0
6-й элемент:	0	0	1	1	1	0	0
7-й элемент:	0	0	0	1	0	0	0

3.37 Дан список списков, состоящий из t элементов, каждый из которых содержит t целых чисел. Найти наибольшие элементы среди элементов, имеющих одинаковые порядковые номера. Вычислить и напечатать сумму найденных наибольших элементов.

3.38 Дан список списков, состоящий из t элементов, каждый из которых содержит 2t целых чисел. Для каждого целого числа с соответствующим четным порядковым номером вычислить сумму квадратов элементов. Для каждого целого числа с соответствующим нечетным порядковым номером вычислить произведение элементов этого столбца. Результат оформить в виде списка, в котором будет находиться 2t чисел.

3.39 Дан список списков, состоящий из t элементов, каждый из которых содержит t целых чисел. Вычислить разности между квадратами суммы и суммами квадратов элементов, имеющих одинаковые порядковые номера.

3.40 Дан список списков, состоящий из t элементов, каждый из которых содержит t целых чисел. Найти наименьшие элементы среди элементов, имеющих одинаковые порядковые номера. Вычислить и напечатать произведение найденных наименьших элементов.

3.41 Из элементов списка A, состоящего из 25 целых чисел, сформировать список из 5 списков, каждый из которых состоит из 5 чисел.

3.42 Даны два целочисленных списка, содержащих одинаковое количество чисел. Написать программу, которая печатает «Да», если эти списки состоят из одинаковых элементов, и «Нет» – в противном случае.

3.43 Дан список списков, состоящий из целых чисел. Напечатать номер элемента списка и номер элемента в списке, который имеет максимальное значение.

3.44 Найти сумму элементов списка списков, имеющих заданную разность: <номер элемента списка>-<номер числа в элементе списка>=k. Число k целое, но не обязательно положительное.

3.45 Дано натуральное число n>=2, список списков, состоящий из n элементов по n чисел в элементе. Построить список $[b_1,..., b_n]$ из нулей и единиц, в котором очередной элемент равен 1 тогда и только тогда, когда элементы i-го списка образуют возрастающую или убывающую последовательность.

3.46 Даны целые числа $a_1,..., a_{10}$, список списков, состоящий из n элементов, каждый из которых содержит n целых чисел. Заменить нулями в исходном списке элементы, у которых сумма <номер элемента списка>+<номер числа в элементе списка> является четной, и для этого элемента имеется равный среди $a_1,..., a_{10}$.

3.47 Дано натуральное число n>=2, список списков, состоящий из n элементов по n чисел в элементе. Построить список, состоящий из n элементов, каждый из которых является максимальным среди элементов списков с соответствующим номером. Пример: дан список [[1,2,3], [4,7,2],[0, –7,1]]. Результат: [4,7,3].

3.48 Дано натуральное число n>=2, список списков arr, состоящий из n элементов по n чисел в элементе. Найти количество положительных элементов среди элементов списка arr[i][i], где 0<=i<n.

3.49 Дано натуральное число n>=2, список списков, состоящий из n элементов по n чисел в элементе. Список заполняется случайным образом числами из промежутка от 0 до 199. Найти количество всех двухзначных чисел, у которых сумма цифр кратна 2.

3.50 Даны натуральные числа n, m>=2, список списков, состоящий из n элементов по m чисел в элементе. Преобразовать этот список так, чтобы последний элемент каждого списка был заменен суммой предыдущих элементов того же списка.

3.51 Даны натуральные числа n, m>=2, список списков, состоящий из n элементов по m чисел в элементе. Найти: максимум из наименьших элементов каждого списка.

3.52 Составить программу получения следующего списка списков: [[1,2,3,4,5], [2,3,4,5,1], [3,4,5,1,2], [4,5,1,2,3], [5,1,2,3,4]].

3.53 Даны натуральные числа n, m>=2, список списков, состоящий из n элементов по m чисел в элементе. Отсортировать в списке списков элементы по убыванию значений элементов в самом первом списке.

3.54 Даны натуральные числа n, m>=2, список списков, состоящий из n элементов по m чисел в элементе. Вычислить сумму чисел, порядковые номера которых являются числами Фибоначчи.

3.5 В заданном списке, состоящем из целых чисел, вычислить сумму модулей элементов списка, расположенных после первого отрицательного элемента.

3.56 В заданном списке, состоящем из целых чисел, найдите сумму и количество элементов списка, попавших в интервал [a; b].

3.57 Вывести в порядке возрастания цифры, входящие в десятичную запись натурального числа N.

3.58 Из заданного списка удалить все повторяющиеся элементы (дубликаты) так, чтобы каждое значение встречалось в списке только один раз.

3.59 В заданном списке целых чисел найти (посчитать) количество положительных и количество отрицательных элементов.

3.60 Определить количество элементов в заданном списке целых чисел, отличающихся от минимального на 5.

3.61 Написать программу, которая сжимает серии списка, состоящего из единиц и нулей по следующему принципу:

- например, список [0,0,0,0,1,1,1,1,1,1,1,1,0,0,1,1,1,1] преобразуется в [4,7,2,4] (т. к. начинается с нуля, то сразу записывается количество элементов первой серии);
- а список [1,1,1,0,0,0,0,0,0,0] преобразуется в [0,3,7] (т. к. первая серия – это единицы, то первый элемент преобразованного списка 0).

3.62 Задан упорядоченный по возрастанию целочисленный список. Сформировать второй список из случайных целых чисел, которые не встречаются в первом списке, но имеют величину больше минимального и меньше максимального из чисел первого списка.

3.63 Заданы три числа, которые обозначают число, месяц, год. Найти порядковый номер даты, начиная отсчет с начала года.

3.64 Дан список целых чисел. Сравнить по модулю сумму элементов, стоящих на четных местах списка, с суммой элементов, стоящих на нечетных местах.

3.65 Найдите наименьший четный элемент списка. Если такого нет, то выведите первый элемент.

3.66 Дан список, состоящий только из 0 и 1. Определить самое большое количество подряд идущих единиц и вывести на экран индексы начала и конца этого диапазона.

3.67 Для списка целых чисел вычислить произведение первого, третьего и шестого положительных элементов и определить их номера в списке.

3.68 Дан список из 10 элементов. Первые 4 упорядочить по возрастанию, последние 4 – по убыванию.

3.69 Пользователь вводит десять целых чисел, представляющих собой элементы списка. Требуется его оценить:

- является ли список возрастающей последовательностью;
- есть ли в списке одинаковые элементы;
- является ли список знакочередующимся (положительные и отрицательные числа чередуются).

3.70 Дан список из N элементов. Требуется определить количество элементов, значение которых больше, чем у соседних элементов списка.

3.71 Даны два целочисленных списка с различным количеством элементов. Перераспределить их элементы так, чтобы в первом списке были наименьшие элементы из двух списков, а во втором – наибольшие.

3.72 Даны два пятизначных числа, необходимо найти количество совпадений по две одинаковые цифры в равносильных разрядах чисел, а также количество совпадений по две одинаковые цифры в различных разрядах этих чисел. Цифра, которая уже участвовала в одной паре совпадения, не учитывается повторно. Например, даны числа 12345 и 27376. Количество совпадений одинаковых цифр в равносильном разряде равно 1 (это цифра 3), количество совпадений одинаковых цифр в различных разрядах равно 1 (это цифра 2).

Лабораторная работа № 4. Структуры данных. Множества

Цель работы – познакомиться с основными конструкциями, используемыми для работы со списками.

1 Фрагмент теории

Множество – это неупорядоченная последовательность уникальных элементов, с которой можно сравнивать другие элементы, чтобы определить, принадлежат ли они этому множеству. Объявить множество можно с помощью функции **set()**.

Для работы с множествами предназначены следующие операторы и соответствующие им методы:

- **|** и **union()** – объединяет два множества;
- **a |= b** и **a.update(b)** – добавляют элементы множества b во множество a;
- **-** и d**ifference()** – вычисляет разницу множеств;
- **a -= b** и **a.difference_update(b)** – удаляют элементы из множества a, которые существуют и во множестве a, и во множестве b;
- **&** и **intersection()** – пересечение множеств. Позволяет получить элементы, которые существуют в обоих множествах;
- **a &= b** и **a.intersection_update(b)** – во множестве a останутся элементы, которые существуют и во множестве a, и во множестве b;
- **^** и **symmetric_difference()** – возвращают все элементы обоих множеств, исключая элементы, которые присутствуют в обоих этих множествах;
- **a ^= b** и a**.symmetric_difference_update(b)** – во множестве, a будут все элементы обоих множеств, исключая те, что присутствуют в обоих этих множествах.

Перечислим основные методы, применяемые при обработке множеств:

- **copy()** – создает копию множества;
- **add(<*Элемент*>)** – добавляет *Элемент* во множество;
- **remove(<*Элемент*>)** – удаляет *Элемент* из множества. Если элемент не найден, то возбуждается исключение **KeyError;**
- **discard(<*Элемент*>)** – удаляет *Элемент* из множества, если он присутствует. Если указанный элемент не существует, никакого исключения не возбуждается;
- **pop()** – удаляет произвольный элемент из множества и возвращает его. Если элементов нет, то возбуждается исключение **KeyError;**
- **clear()** – удаляет все элементы из множества.

2 Задачи для самостоятельного решения

4.1 Из множества целых чисел 1..20 выделить:

a) множество чисел, делящихся на 6 без остатка;

b) множество чисел, делящихся без остатка или на 2, или на 3.

4.2 Определить количество различных гласных букв латинского алфавита в некотором тексте.

4.3 В заданной последовательности литер, состоящей из букв латинского алфавита, определить общее число вхождений в нее букв «a», «e», «c», «h». Пример: дана последовательность букв «areaecc»; ответ – 3.

4.4 Переменные x, y, z – множества. Переменной x присвоить множество всех целых чисел от 8 до 22, переменной y – множество всех простых чисел от 8 до 22, а переменной z – множество всех составных чисел из этого же диапазона. Примечание: где возможно, использовать операторы, предназначенные для работы с множествами.

4.5 Дан текст. В алфавитном порядке напечатать (по разу) все строчные русские гласные буквы (а, е, о, у, ы, э, ю, я, и, ё), входящие в этот текст.

4.6 Дан текст из строчных латинских букв. Напечатать:

а) первые вхождения букв в текст, по возможности, сохраняя их исходный взаимный порядок;

б) все буквы, входящие в текст не менее двух раз;

в) все буквы, входящие в текст по одному разу.

4.7 В возрастающем порядке напечатать все целые числа из диапазона 1..10000, представимые в виде n*n+m*m, где n, m>=0. Для представления диапазона использовать множество.

4.8 Подсчитать общее количество цифр и знаков «+», «– », и «*», входящих в строку s.

4.9 Даны целые числа от 1 до 50. Определить, сколько среди них чисел Фибоначчи и сколько чисел, первая значащая цифра в десятичной записи которых 2 или 3.

4.10 Дано множество M, содержащее числа от 0 до 99. Подсчитать количество элементов во множестве A из M.

4.11 Подсчитать количество различных (значащих) цифр в десятичной записи натурального числа n.

4.12 Напечатать, желательно, в возрастающем порядке все цифры, не входящие в десятичную запись натурального числа n.

4.13 В порядке убывания напечатать все целые числа из диапазона 1..10000, которые представимы в виде n^2+2k^2, но не представимы в виде 7ij+j+3 (n, k, i, j >= 0).

4.14 Дано множество продуктов product = {«хлеб», «масло», «молоко», «мясо», «рыба», «соль», «сыр», «колбаса», «сахар», «чай», «кофе»}. Есть список магазинов, каждый элемент которого – множество продуктов в этом магазине. Определить значения множеств A, B, C: A – множество продуктов, которые есть во всех магазинах; B – множество продуктов, каждый из которых есть хотя бы в одном магазине; C – множество продуктов, которых нет ни в одном магазине.

4.15 Дано множество имен = {«Вася», «Володя», «Ира», «Лида», «Марина», «Миша», «Наташа», «Олег», «Оля», «Света», «Юля»}. Есть список, каждый элемент которого – множество людей, побывавших у какого-либо члена группы. Определить, есть ли хотя бы один человек, побывавший в гостях у всех остальных из групы.

4.16 Дано множество городов Town = {«a», «b», «c», «d», «e», «f», «g», «h»}. Есть список, каждый элемент которого – множество городов, в которые можно попасть из города с заданным номером из множества Town. Определить K – множество городов, в которые можно попасть автобусом (за один рейс или через другие города) из города H.

4.17 Дано целое n от 2 до 1000. Используя метод «решета Эратосфена», напечатать в убывающем порядке все простые числа из диапазона n..2n.

4.18 Дан текст из цифр и строчных латинских букв. Определить, каких букв – гласных или согласных – больше в этом тексте.

4.19 Из первой сотни натуральных чисел найти все простые с помощью решета Эратосфена. Для представления решета используйте множество!

4.20 Проверить введенную пользователем строку на наличие недопустимых символов. В качестве первого символа допустимы только буквы и знак подчеркивания. Остальные символы могут быть буквами, цифрами и знаком подчеркивания.

4.21 Известны марки машин, изготовляемые в данной стране и импортируемые за рубеж. Даны некоторые N стран. Определить для каждой из марок, какие из них были: a) доставлены во все страны; b) доставлены в некоторые из стран; c) не доставлены ни в одну страну.

4.22 В классе учится n учеников. Известны их имена. Известно так же, кто был в гостях у Кати, кто был у Васи и т. д. Определить, есть ли в классе хотя бы один человек, который не был в гостях ни у кого из своих одноклассников. Если есть такие ученики, то вывести их имена, если нет, то сообщить об этом.

4.23 Дана строка символов. Определить количество различных символов, которые являются буквами или цифрами, вывести их на печать, используя множества.

4.24 Дана последовательность символов. Требуется построить и напечатать множество, элементами которого являются встречающиеся в последовательности знаки препинания.

4.25 Дана последовательность символов. Требуется построить и напечатать множество, элементами которого являются встречающиеся в последовательности цифры от 1 до 3 и числа от 17 до 99 включительно.

Лабораторная работа № 5. Использование функций

Цель работы – получить первоначальные навыки создания и использования функций.

1 Фрагмент теории

Функция создается (определяется) с помощью ключевого слова **def** по следующей схеме:

```
def <Имя функции> ([<Параметры>]):
["" Строка документирования ""]
<Тело функции>
[return <Результат>]
```

Имя функции должно быть уникальным идентификатором, состоящим из латинских букв, цифр и знаков подчеркивания, причем имя функции не может начинаться с цифры. В качестве имени нельзя использовать ключевые слова, кроме того, следует избегать совпадений с названиями встроенных идентификаторов. Регистр символов в названии функции также имеет значение.

После имени функции в круглых скобках можно указать один или несколько параметров через запятую, а если функция не принимает параметры, указываются только круглые скобки. После круглых скобок ставится двоеточие.

Тело функции представляет собой составную конструкцию. Как и в любой составной конструкции, инструкции внутри функции выделяются одинаковым количеством пробелов слева. Концом функции считается инструкция, перед которой находится меньшее количество пробелов. Если тело функции не содержит инструкций, то внутри ее необходимо разместить оператор **pass**, который не выполняет никаких действий. Этот оператор удобно использовать на этапе отладки программы, когда мы определили функцию, а тело решили дописать позже. Пример функции, которая ничего не делает:

```
def func(): pass
```

Необязательная инструкция **return** позволяет вернуть из функции какое-либо значение в качестве результата. После исполнения этой инструкции выполнение функции будет остановлено. Это означает, что инструкции, следующие после оператора **return**, никогда не будут выполнены.

Инструкции **return** может не быть вообще. В этом случае выполняются все инструкции внутри функции, и в качестве результата возвращается значение **None**.

Пример определения функции:

```
def summa(x, y):
 """ Пример функции с параметрами,
```

```
    возвращающей сумму двух переменных """
    return x + y
```

При вызове функции значения передаются внутри круглых скобок через запятую. Если функция не принимает параметров, то указываются только круглые скобки. Необходимо также заметить, что *количество параметров в определении функции должно совпадать с количеством параметров при вызове*, иначе будет выведено сообщение об ошибке.

Примеры вызовов функции:

```
x = summa(5, 2) # Переменной x будет присвоено значение 7
a, b = 10, 50
y = summa(a, b) # Переменной y будет присвоено значение 60
```

Как видно из последнего примера, имя переменной в вызове функции может не совпадать с именем переменной в определении функции. Кроме того, *глобальные переменные* x и y не конфликтуют с одноименными переменными в определении функции, т. к. они расположены в разных областях видимости. Переменные, указанные в определении функции, являются *локальными* и *доступны только внутри функции*.

Определение функции должно быть расположено перед вызовом функции. Желательно располагать функции в начале программы.

Глобальные переменные – это переменные, объявленные в программе вне функции. В Python глобальные переменные *видны в любой части модуля*, включая функции.

Локальные переменные – это переменные, объявляемые внутри функций. Если имя локальной переменной совпадает с именем глобальной переменной, то все операции внутри функции осуществляются с локальной переменной, а значение глобальной переменной не изменяется. Локальные переменные видны только внутри тела функции.

Поиск идентификатора, используемого внутри функции, будет производиться в следующем порядке.

1 Поиск объявления идентификатора внутри функции (в локальной области видимости).

2 Поиск объявления идентификатора в глобальной области.

3 Поиск во встроенной области видимости (встроенные функции, классы и т. д.).

2 Задачи для самостоятельного решения

5.1 Даны действительные числа a_0, ..., a_6. Получить для x = 1, 2, 3, 4 значения p(x+1)-p(x), где: $p(y) = a[1]*y^6 + a[2]*y^5 + a[3]*y^4 + a[4]*y^3 + ... + a[0]$.

5.2 Даны действительные числа s, t, a_0 , ... , a_{12}. Получить p(1) –p(t)+p(s –t)*p(s –t) –p(1)*p(1), где $p(x) = a_{12}\, x^{12} + a_{11}\, x^{11} + ... + a_0$.

5.3 Дано натуральное число n. Среди чисел 1, 2, ... n найти все те, которые можно представить в виде суммы квадратов двух натуральных чисел, определив функцию, позволяющую распознавать полные квадраты.

5.4 Найти значение переменной z, заданной суммой функций:

$$z = f(a,b) + f(a^2,b^2) + f(a-1,b) + f(a-b,b) + f(a^2+b^2,b^2-1),$$

где: $f(u,t)=\begin{cases} u^2+t^2, & \text{если } u>0, t>0; \\ u+t^2, & \text{если } u\le 0, t\le 0; \\ u-t, & \text{если } u>0, t\le 0; \\ u+t, & \text{если } u\le 0, t>0. \end{cases}$

5.5 Найти значение переменной z, заданной суммой функций: z = f(sinb,a) + f(cosb,a) + f(sin b,a–1) + f(sinb –cosb,a*a –1) + f(sinb*sinb –1,cosa+a),

где: $f(u,t)=\begin{cases} u+\sin t, & \text{если } u>0; \\ u+t, & \text{если } u\le 0. \end{cases}$

5.6 Найти значение переменной z, заданной суммой функций: z=f(|x|,y) + f(a,b) + f(|x|+1, –y) + f(|x| –|y|,x)+f(x+y,a+b), где:

$$f(u,t)=\begin{cases} u+2t, & \text{если } u\ge 0; \\ u+t, & \text{если } u\le -1; \\ u^2-2t+1, & \text{если } -1<u<0. \end{cases}$$

5.7 Найти значение переменной z, заданной суммой функций: z = f(sinx+cosy,x+y)+ + f(sinx,cosy) + f(x –y,x) + f(sin x –2,a) + f(a+3,b+1), где:

$$f(u,t)=\begin{cases} u+t, & \text{если } u>1; \\ u-t, & \text{если } 0\le u\le 1; \\ t-u, & \text{если } u<0. \end{cases}$$

5.8 Натуральное число называется палиндромом, если оно читается одинаково с обеих сторон (например, 171). Возьмем произвольное натуральное число X. Если оно не палиндром, то перевернем его и сложим с исходным числом. Если сумма не является палиндромом, то проделаем с ней указанные операции. Работу продолжать до тех пор, пока не получится палиндром. На экран вывести полученное число и количество шагов. Для получения перевернутого числа составить функцию.

5.9 Составить программу вывода разложения бинома Ньютона: $(a+b)^n = C(0,n)a^n b^0 + C(1,n)a^{n-1} b^1 + ... + C(n,n)a^0 b^n$, где

$$C(0,n)=C(n,n)=1, \qquad C(m,n)=\frac{n!}{m!(n-m)!}.$$

5.10 Переменной t присвоить значение true, если уравнения $x^2 + 6.2x + a^2 = 0$ и $x^2 + ax + b - 1 = 0$ имеют вещественные корни и при этом оба корня первого уравнения лежат между корнями второго, и присвоить значение false во всех остальных случаях.

5.11 По заданным вещественным числам $a_0,...,a_{30}$, $b_0,...,b_{30}$, c_0, ..., c_{30}, x, y, z вычислить величину:

$$\frac{(a_0x^{30}+a_1x^{29}+\ldots+a_{30})^2-(b_0y^{30}+b_1y^{29}+\ldots+b_{30})^2}{c_0(x+z)^{30}+c_1(x+z)^{29}+\ldots+c_{30}}.$$

5.12 Даны два списка списков по 10 элементов. Каждый элемент списка – это список, состоящий из целых чисел. Вывести тот список, в котором находится список (элемент), содержащий минимальную сумму чисел.

5.13 Дан список списков A, содержащий M элементов, в каждом из которых N вещественных чисел. Найти величину $x_1x_n + x_2 x_{n-1} + ... + x_nx_1$, где x_i – максимальный элемент i-го элемента списка A.

5.14 Даны три списка списков, состоящих из 5 элементов, в каждом из которых 4 целых числа. Вывести тот из них, где больше элементов вида [0, 0, 0, 0] (если таких списков несколько, то вывести их все).

5.15 Даны три списка списков A, B, C, состоящих из 4 элементов, каждый из которых состоит из 5 вещественных чисел. Вычислить величину:

$$\frac{< A > + < B > + < C >}{< A + B + C >}, \text{ где:}$$

$$< D >= \max_{0 \le j \le 4} | D_{0,j} | + \max_{0 \le j \le 4} | D_{1,j} | + \max_{0 \le j \le 4} | D_{2,j} | + \max_{0 \le j \le 4} | D_{3,j} |.$$

5.16 Сгенерировать десять списков из случайных чисел. Вывести их и сумму их элементов на экран. Найти среди них один с максимальной суммой элементов. Указать какой он по счету, повторно вывести этот список и сумму его элементов. Заполнение списка и подсчет суммы его элементов оформить в виде отдельных функций (стандартную функцию подсчета суммы элементов списка использовать нельзя).

5.17 Найти наибольшие общие делители (НОД) для списка кортежей, составленных из пар чисел. Для нахождения НОД использовать функцию.

5.18 Дан список из N элементов, состоящих из N целых чисел. Вычислить сумму элементов главной или побочной диагонали, полученной из этого списка матрицы, в зависимости от выбора пользователя. Сумма элементов любой диагонали должна вычисляться в одной и той же функции.

5.19 Дан список, состоящий из натуральных чисел. Выполнить сортировку данного списка по возрастанию суммы цифр чисел. Например, дан список чисел [14, 30, 103]. После сортировки он будет таким: [30, 103, 14], так как сумма цифр числа 30 составляет 3, числа 103 равна 4, числа 14 равна 5.

Вывести на экран исходный список, отсортированный список, а также для контроля сумму цифр каждого числа отсортированного списка. Для нахождения суммы цифр использовать функцию.

5.20 Найти средние арифметические пяти списков, состоящих их десяти целых чисел. Для нахождения среднего арифметического использовать функцию.

5.21 Дан список, состоящий из целых чисел. Написать и протестировать функцию, которая из заданного списка формирует новый список, состоящий только из элементов, дважды входящих в первый список.

5.22 Дан список, состоящий из целых чисел. С клавиатуры вводится число N. Написать и протестировать функцию, которая сжимает список, удаляя из него элементы, равные числу N.

5.23 Написать функцию, которая циклически сдвигает список вправо или влево на указанное число позиций. Сдвиг также должен быть кольцевым, то

есть те элементы, которые уходят вправо или влево за пределы списка, должны помещаться с другого его конца.

Например, дан список: [1,2,3,4,5,6]. Кольцевой сдвиг вправо на 2 единицы: [5,6,1,2,3,4].

5.24 Написать функцию, заменяющую подстроку, которая начинается с первого вхождения в строку s открывающей квадратной скобки и заканчивается соответствующей ей закрывающей квадратной скобкой, на строку s1 и возвращающую подстроку, заключенную между скобками в качестве своего значения.

Лабораторная работа № 6. Структуры данных. Словари

Цель работы – получить первоначальные навыки использования словарей при решении практических задач.

1 Фрагмент теории

Словари – это наборы объектов, доступ к которым осуществляется не по индексу, а по ключу. В качестве ключа можно указать неизменяемый объект, например: число, строку или кортеж. Элементы словаря могут содержать объекты произвольного типа данных и иметь неограниченную степень вложенности. Следует также заметить, что элементы в словарях располагаются в произвольном порядке. Чтобы получить элемент, необходимо указать ключ, который использовался при сохранении значения.

Словари относятся к отображениям, а не к последовательностям. По этой причине функции, предназначенные для работы с последовательностями, а также операции извлечения среза, конкатенации, повторения и др., к словарям не применимы. Равно как и списки, словари относятся к изменяемым типам данных. Иными словами, мы можем не только получить значение по ключу, но и изменить его.

Укажем некоторые методы, которые будут полезны при работе со словарями:

- **keys()** – возвращает объект **dict_keys**, содержащий все ключи словаря. Этот объект поддерживает итерации, а также операции над множествами;
- **values()** – возвращает объект **diet_values**, содержащий все значения словаря. Этот объект поддерживает итерации;
- **items()** – возвращает объект **dict_items**, содержащий все ключи и значения в виде кортежей. Этот объект поддерживает итерации;
- ***Ключ* in *Словарь*** – проверяет существование указанного ключа в словаре. Если ключ найден, то возвращается значение **True**, в противном случае – **False**;
- ***Ключ* not in *Словарь*** – проверяет отсутствие указанного ключа в словаре. Если такового ключа нет, то возвращается значение **True**, в противном случае – **False**;
- **get(<*Ключ*>[, <*Значение по умолчанию*>])** – если ключ присутствует в словаре, то метод возвращает значение, соответствующее этому ключу. Если ключ отсутствует, то возвращается **None** или значение, указанное во втором параметре;

- **setdefault(<*Ключ*>[, <*Значение по умолчанию*>])** – если ключ присутствует в словаре, то метод возвращает значение, соответствующее этому ключу. Если ключ отсутствует, то создает в словаре новый элемент со значением, указанным во втором параметре. Если второй параметр не указан, значением нового элемента будет **None**;
- **pop(<*Ключ*>[, <*Значение по умолчанию*>])** – удаляет элемент с указанным ключом и возвращает его значение. Если ключ отсутствует, то возвращается значение из второго параметра. Если ключ отсутствует, и второй параметр не указан, возбуждается исключение **KeyError**;
- **popitem()** – удаляет произвольный элемент и возвращает кортеж из ключа и значения. Если словарь пустой, возбуждается исключение **KeyError**;
- **clear()** – удаляет все элементы словаря. Метод ничего не возвращает в качестве значения;
- **update()** – добавляет элементы в словарь. Метод изменяет текущий словарь и ничего не возвращает. Форматы метода:

```
update(<Ключ1>=<Значение1>[, ..., <КлючN>=<ЗначениеN>])
update(<Словарь>)
update(<Список кортежей с двумя элементами>)
update(<Список списков с двумя элементами>)
```

Если элемент с указанным ключом уже присутствует в словаре, то его значение будет перезаписано.

2 Задачи для самостоятельного решения

6.1 Опишите, используя словарь, телефонную книгу. Составьте программу, выдающую список абонентов, имеющих телефонный номер, начинающийся на 33.

6.2 Опишите, используя словарь, каталог книг в библиотеке. Составьте программу, выдающую список книг В. Пикуля, хранящихся в библиотеке.

6.3 Опишите, используя словарь, таблицу дат и событий русской истории. Составьте программу, выдающую список событий XIX века.

6.4 Опишите, используя словарь, школьную нагрузку (фамилия преподавателя, класс, часы). Составьте программу, определяющую нагрузку каждого преподавателя. Определить, у какого преподавателя самая большая нагрузка и у кого самая низкая.

6.5 Опишите, используя словарь, таблицу соревнований (название команды, количество набранных очков). Выбрать команду, занявшую первое место. Упорядочить список команд, в зависимости от занятого места.

6.6 При сдаче норм ГТО были получены результаты забега на 100 метров и прыжков в длину. Задайте нормы ГТО по этим видам, определите списки учеников, не выполнивших нормативы, количество учеников, сдавших нормативы, а также список 3 лучших.

6.7 Опишите, используя словарь, товар (наименование товара, старая цена, новая цена). Составьте программу, определяющую, на какие товары повысятся цены и на сколько процентов.

6.8 В анкетных данных обозначены фамилия, пол, рост. Определите средний рост женщин, фамилию самого высокого мужчины, есть ли в группе хотя бы два человека одного роста.

6.9 Опишите, используя словарь, записную книжку (фамилия, номер телефона). Составьте программу, определяющую:

1) есть ли в записной книжке сведения о знакомом с фамилией на букву «Ф», если есть – напечатайте его фамилию и телефон;

2) есть ли в записной книжке сведения о знакомом с телефоном 47-67-35, если есть – напечатайте его фамилию.

6.10 Вам дан словарь, состоящий из слов, расположенных парами. Каждое слово является синонимом к парному ему слову. Все слова в словаре различны. Для последнего слова из словаря определите его синоним.

6.11 Дан список стран и городов каждой страны. Затем даны названия городов. Для каждого города укажите, в какой стране он находится.

6.12 В сводке об экспортируемых товарах указывается: наименование товара, страна, импортирующая товар, объем поставляемой партии в штуках. Напечатайте списки стран, в которые экспортируется данный товар, и общий объем его экспорта.

6.13 Хранятся сведения о лесе: вид дерева, общая численность, численность здоровых деревьев. Составьте программу вычисления: 1) суммарного числа деревьев на контрольном участке; 2) суммарного числа здоровых деревьев; 3) относительную численность (%) больных деревьев; 4) относительную численность (%) различных видов, в том числе больных (%) для каждого вида.

6.14 Приняв способ изображения рационального числа в виде записи с двумя полями [числитель, знаменатель] целого типа написать программу, позволяющую:

а) определить, есть ли среди 10 рациональных чисел равные;

б) вычислить наибольшее из данных рациональных чисел (числа не обязательно имеют несократимую форму).

Для хранения рациональных чисел использовать словарь.

6.15 При поступлении на музыкально-педагогический факультет на абитуриентов собирают информацию: фамилия, музыкальный инструмент. Для поступления необходимо сдать экзамен по специальности. Составьте списки для данного экзамена, в зависимости от специальности.

6.16 Опишите, используя словарь, школьный класс (фамилия и инициалы, дата рождения, месяц рождения, год рождения). Напечатайте список учеников, рожденных в мае.

6.17 Опишите, используя словарь, записную книжку. Напечатайте список друзей, кому в этом году исполняется 25 лет (фамилия и инициалы, год рождения, дата рождения, месяц рождения).

6.18 Опишите, используя словарь, школьный класс (фамилия и инициалы, дата рождения, месяц рождения, год рождения). Вычислите день рождения класса (среднее арифметическое дат, месяцев, годов).

6.19 Опишите, используя словарь, данные на учеников (фамилия, улица, дом, квартира). Составьте программу, определяющую, сколько учеников живет на улице Гоголя, списки учеников, живущих в доме номер 45.

6.20 Опишите, используя словарь, выборы (фамилия кандидата и количество набранных голосов). Всего избирателей 2000. Определить кто из кандидатов прошел, или необходимо проводить повторные выборы (должно быть набрано не менее 1/3 голосов от общего количества).

6.21 Выберите самую высокую вершину из заданного словаря.

6.22 Опишите, используя словарь, анкету школьника (фамилия, возраст). Определите возрастные группы в классе и напечатайте их списки.

6.23 Опишите, используя словарь, оценки за год. Посчитайте процент и качество успеваемости в классе за год, составьте списки неуспевающих и отличников.

6.24 О поступивших в вуз студентах собрана информация: фамилия, нуждается ли в общежитии, стаж работы (если есть), что окончил, какой язык изучал. Определите: a) сколько человек нуждаются в общежитии; b) списки студентов, имеющих стаж работы более 2 лет; c) списки окончивших техникум; d) списки языковых групп.

Лабораторная работа № 7. Создание приложений на использование основных виджетов

Цель работы – получить первоначальные навыки использования основных виджетов при создании приложений с графическим интерфейсом.

1 Фрагмент теории

Современный пользователь в основном взаимодействует с программой с помощью различных кнопок, меню, значков, вводя информацию в специальные поля, выбирая определенные значения в списках и т. д. Все эти элементы формируют графический интерфейс пользователя (GUI), в дальнейшем мы их будем называть виджетами (от англ. widget – «штучка»).

Для языка программирования Python все виджеты включены в специальную библиотеку – tkinter. Если ее импортировать в программу (скрипт), то можно пользоваться ее компонентами, создавая графический интерфейс.

Каждый виджет представляет собой объект, создаваемый на основе класса, который содержит в себе описание свойств, характеризующий виджет, и методов, которые реализуют его функциональность (то, что может «делать» этот виджет). Таким образом, можно выделить две фазы использования виджета:

- создать объект виджета (в дальнейшем понятия «виджет» и «объект виджета» будем считать синонимами) и, при необходимости, задать значения его свойств (надпись на кнопке, размеры виджета и т. п.);
- реализовать реакцию на возникающие в программе события (нажатие на кнопку, ввод каких-то значений и т. п.), то есть описать «поведение» виджета при возникновении указанных событий.

Приведем небольшой пример программы, реализующей GUI:

```
from tkinter import *

def printer(event):
 print("Привет, мир")

root = Tk() # Создаем главное окно приложения (root)
but = Button(root) # Создаем кнопку
but["text"] = "Печать" # Меняем надпись на кнопке
but.bind("<Button-1>",printer) # Связываем нажатие кнопки с
 # вызываемой функцией

but.pack() # «Упаковываем» кнопку в окно
root.mainloop() # Выполняем приложение
```

Результат работы приложения приведен на рисунке 1:

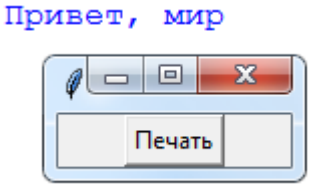

Рисунок 1 – Приложение с графическим интерфейсом (GUI)

Кратко прокомментируем приведенный текст приложения.

Как и любой модуль, **tkinter** в **Python** можно импортировать двумя способами: командами **import tkinter** или **from tkinter import ***. В дальнейшем мы будем пользоваться только вторым способом, т. к. это позволит не указывать каждый раз имя модуля при обращении к объектам, которые в нем содержатся. Итак, первая строка программы будет выглядеть так:

```
from tkinter import *
```

В современных операционных системах любое пользовательское приложение заключено в окно, которое можно назвать *главным*, т. к. в нем располагаются все остальные виджеты. Объект окна верхнего уровня создается при обращении к классу **Tk** модуля **tkinter**. Переменную, связанную с объектом-окном, принято называть **root** (хотя понятно, что можно назвать как угодно). Вторая строка кода:

```
root = Tk()
```

Пусть в окне будет располагаться всего одна кнопка. Кнопка создается при обращении к классу **Button** модуля **tkinter**. Объект кнопка связывается с какой-нибудь переменной. У класса **Button** (как и всех остальных классов, за исключением **Tk**) есть обязательный параметр – объект, которому кнопка принадлежит (кнопка не может «быть ничейной»). Пока у нас есть единственное окно (**root**), оно и будет аргументом, передаваемым в класс при создании объекта-кнопки:

```
but = Button(root)
```

У кнопки много свойств: размер, цвет фона и надписи и др. Мы перечислим их позже. Пока же установим всего одно свойство – текст надписи (**text**):

```
but["text"] = "Печать"
```

Многообразие событий и способов их обработки будет рассмотрено позднее. Здесь же просто коснемся данного вопроса в связи с потребностью.

Что же будет делать кнопка, и в какой момент она это будет делать? Предположим, что задача кнопки: вывести какое-нибудь сообщение в поток вывода, используя функцию **print()**. Делать она это будет при нажатии на нее левой кнопкой мыши.

Действия (алгоритм), которые происходят при том или ином событии, могут быть достаточно сложным. Поэтому часто их оформляют в виде функции, а затем вызывают, когда они понадобятся. Пусть у нас печать на экран будет оформлена в виде функции **printer()**:

```
def printer(event):
 print("Привет, мир")
```

Не забывайте, что функцию желательно (почти обязательно) размещать в начале кода. Параметр **event** – это какое-либо событие.

Событие нажатия левой кнопкой мыши выглядит так: **<Button-1>**. Требуется связать это событие с обработчиком (функцией **printer()**). Для связи предназначен метод **bind()**. Синтаксис связывания события с обработчиком выглядит так:

```
but.bind("<Button-1>", printer)
```

В любом приложении виджеты не разбросаны по окну как попало, а хорошо организованы, интерфейс продуман до мелочей и обычно подчинен определенным стандартам. В нашем случае нужно просто кнопку как-то отобразить в окне. Самый простой способ – это использование метода **pack().**

```
but.pack()
```

Если не вставить эту строку кода, то кнопка в окне так и не появится, хотя она есть в программе.

Ну и, наконец, главное окно тоже не появится, пока не будет вызван специальный метод **mainloop()**:

```
root.mainloop()
```

Данная строка кода должна быть всегда в конце скрипта!

Кратко охарактеризуем наиболее употребительные виджеты/

1.1 Кнопка (Button)

Задается следующим образом:

```
<переменная> = Button(<родит_виджет>,
    [<свойство>=<значение>, ... ])
```

Основные свойства кнопки приведены в таблице 1.

Таблица 1 – Основные свойства кнопки (класс Button)

Свойство	Описание
text	Текст надписи для кнопки
foreground (fg)	Цвет текста на кнопке
background (bg)	Цвет фона кнопки
font	Шрифт, используемый для надписи
borderwidth (bd)	Толщина рамки у кнопки в виде целого числа в пикселях
width	Ширина кнопки (измеряется в знакоместах)
height	Высота кнопки (измеряется в знакоместах)
command	Задание действия, выполняемого по щелчку на кнопке. Как правило, здесь указывается имя функции

Большинство этих свойств также имеется у других виджетов.

Пример задания свойств кнопки при ее создании:

```
root = Tk()

but = Button(root,
 text="Это кнопка", #надпись на кнопке
 width=30,height=5, #ширина и высота
 bg="white",fg="blue") #цвет фона и надписи
```

1.2 Метки (Label)

Метки (или надписи) – это достаточно простые виджеты, содержащие строку (или несколько строк) текста и служащие, в основном, для информирования пользователя.

```
lab = Label(root, text="Это метка! \n Из двух строк.",
            font="Arial 18")
```

1.3 Однострочное текстовое поле (Entry)

Такое поле создается при обращении к классу **Entry**. В него пользователь может ввести только одну строку текста.

```
ent = Entry(root, width=20, bd=3)
```

1.4 Многострочное текстовое поле (Text)

Объект класса **Text** предназначен для предоставления пользователю возможности ввода не одной строки текста, а существенно больше.

```
tex = Text(root, width=40,font="Verdana 12",wrap=WORD)
```

Последнее свойство (**wrap**) в зависимости от своего значения позволяет переносить текст, вводимый пользователем либо по символам (**CHAR**), либо по словам (**WORD**), либо вообще не переносить (**NONE**), пока пользователь не нажмет клавишу **Enter**.

1.5 Радиокнопки (Radiobutton)

Объект-радиокнопка никогда не используется по одному. Их используют группами, при этом в одной группе может быть «включена» лишь одна кнопка.

```
Var = IntVar()
var.set(1)
rad0 = Radiobutton(root,text="Первая", variable=var, value=0)
rad1 = Radiobutton(root,text="Вторая", variable=var, value=1)
rad2 = Radiobutton(root,text="Третья", variable=var, value=2)
```

Одна группа определяет значение одной переменной, т. е. если в примере будет выбрана радиокнопка **rad2**, то значение переменной будет **var** будет 2. Изначально также требуется установить значение переменной (выражение `var.set(1)` задает значение переменной **var** равное 1).

1.6 Флажки (Checkbutton)

Объект **checkbutton** предназначен для выбора не взаимоисключающих пунктов в окне (в группе можно активировать один, два или более флажков или не одного). В отличие от радиокнопок, значение каждого флажка привязывается к

своей переменной, значение которой определяется опциями **onvalue (*включено*)** и **offvalue (*выключено*)** в описании флажка.

```
c1 = IntVar()
c2 = IntVar()
che1 = Checkbutton(root, text="Первый флажок",
 variable=c1, onvalue=1, offvalue=0)
che2 = Checkbutton(root, text="Второй флажок",
 variable=c2, onvalue=2, offvalue=0)
```

1.7 Списки (Listbox)

Вызов класса **Listbox** создает объект, в котором пользователь может выбрать один или несколько пунктов в зависимости от значения опции **selectmode**. В примере ниже значение **SINGLE** позволяет выбирать лишь один пункт из списка.

```
r = ['Linux','Python','Tk','Tkinter']
lis = Listbox(root, selectmode=SINGLE, height=4)
for i in r:
 lis.insert(END,i)
```

Изначально список (**Listbox**) пуст. С помощью цикла **for** в него добавляются пункты из списка (тип данных) **r**. Добавление происходит с помощью специального метода класса **Listbox** – **insert()**. Данный метод принимает два параметра: куда добавить и что добавить.

В таблице 2 приведены наиболее употребительные свойства и методы этого виджета.

Таблица 2 – Основные свойства списка (класс Listbox)

Свойство	Описание
selectmode	Режим выбора пунктов. Возможные значения: **SINGLE** – выбор только одного элемента в списке, **MULTIPLE** – возможность выбора нескольких пунктов списка
foreground (fg)	Цвет текста
background (bg)	Цвет фона
font	Шрифт, используемый для надписи
selectforeground	Цвет текста у выбранного пункта
selectbackground	Цвет фона у выбранного пункта
width	Ширина компонента в символах текста. Значение по умолчанию – 20
height	Высота списка в пунктах. Значение по умолчанию – 10
command	Задание действия, выполняемого по щелчку на кнопке. Как правило, здесь указывается имя функции

Перечислим наиболее употребительные методы этого виджета.

1 insert(<*Индекс пункта*>, <*Текст вставляемого пункта*>) – вставляет новый пункт, текст которого задан в виде строки вторым параметром, перед пунктом, индекс которого указан первым параметром. В качестве индекса пункта можно использовать:

- целочисленный номер пункта. Нумерация пунктов в списке начинается с 0;

- **END** – конец списка;
- **ACTIVE** – выбранный пункт списка. Если список позволяет выбирать несколько пунктов, указывает на пункт, выбранный последним;
- строку формата **"@<*Горизонтальная координата*>,<*Вертикальная координата*>"** – пункт, на который приходится точка с указанными координатами, или ближайший к этой точке пункт. Координаты задаются в виде целых чисел в пикселях относительно самого списка.

2 delete(<*Начальный индекс*>[, <*Конечный индекс*>]) – удаляет все пункты, расположенные между указанными в параметрах индексами, включая и сами эти пункты. Если второй параметр отсутствует, удаляет пункт с индексом, заданным первым параметром.

3 selection_clear (<*Начальный индекс*>[, <*Конечный индекс*>]) – убирает из числа выбранных все пункты, расположенные между указанными в параметрах индексами, включая и сами эти пункты. Если второй параметр отсутствует, делает невыбранным только пункт с индексом, заданным первым параметром.

4 size() – возвращает количество пунктов в списке.

5 get(<*Начальный индекс*>[, <*Конечный индекс*>]) – возвращает кортеж с текстовыми надписями всех пунктов, расположенных между указанными в параметрах индексами, включая и сами эти пункты. Если второй параметр отсутствует, возвращает текст пункта с индексом, заданным первым параметром, в виде строки.

6 index(<*Индекс пункта*>) – выполняет прокрутку списка таким образом, чтобы пункт с заданным индексом находился в его верхней части.

7 activate(<*Индекс пункта*>) – делает пункт с указанным индексом выбранным.

2 Задачи для самостоятельного решения

Выполнить задачу 1.31 из первого раздела, взяв формулу, соответствующую заданному варианту. Полученный результат задачи 1.31(б) вывести в виджет **Listbox**.

Выполнить любую задачу заданного варианта из 2-го раздела, разработав приложение с GUI.

Лабораторная работа № 8. Обработка событий в tkinter

Цель работы – познакомить с основными понятиями, связанными с обработкой событий.

1 Фрагмент теории

При создании GUI-приложения не обойтись без ***обработки событий***, то есть выполнения определенных действий в ответ на возникновение некоторого события (нажатия кнопки, заполнения поля ввода и т. п.). В зависимости от виджета в tkinter у него могут быть одни события, и не быть других. Существует два способа связывания событий с обработчиками:

- первый – это задание виджету обработчика на стандартное для данного виджета событие (например, на нажатие кнопки) через передачу имени функции в аргументе **command** этого виджета при создании виджета и перед размещением его в контейнере. При этом не у всех виджетов есть такой параметр. Например, у виджета **Button** аргумент **command** есть, а у **Label** – нет;
- второй – использование метода **bind()**. Этот метод есть у всех виджетов, унаследованных от класса **Widget**.

Примеры:

1-й способ:

```
. . . .
def click():
 print("Привет")

btn = Button(text="Привет",
 command=click)
. . . .
```

2-й способ:

```
. . . .
def click(event):
 print("Привет")

btn = Button(text="Привет")
btn.bind("<Button-1>", click)
....
```

Если есть возможность использовать первый способ, то лучше воспользоваться им. Второй способ используется в том случае, когда нужно обрабатывать другие события виджета.

Как видно из приведенного примера метод **bind()** имеет два параметра: событие и имя функции, вызываемое в ответ на его возникновение. Остановимся на правилах задания события.

Наименование события записывается в следующем формате:

```
<[<Префиксы, разделенные дефисами>-]<Тип события>[-<Дополнение>]>
```

Обязательным компонентом является только тип события. Все поддерживаемые библиотекой события вместе с их типами приведены в таблице 3. Помимо этого, для каждого события там указан числовой код типа, который может пригодиться при обработке событий.

Таблица 3 – События, поддерживаемые библиотекой Tkinter

Тип	Условие возникновения	Код
Button	Нажатие кнопки мыши	4
ButtonRelease	Отпускание ранее нажатой кнопки мыши	5
MouseWheel	Вращение колесика мыши на компоненте	38
Enter	Наведение курсора мыши на компонент	7
Motion	Перемещение курсора мыши внутри компонента	6
Leave	Увод курсора мыши с компонента	8
KeyPress	Нажатие клавиши	2
KeyRelease	Отпускание ранее нажатой клавиши	3
FocusIn	Получение компонентом фокуса ввода	9
FocusOut	Потеря компонентом фокуса ввода	10
Activate	Изменение состояния компонента с недоступного для ввода (такой компонент закрашен серым) на доступное	36

Окончание таблицы 3

Deactivate	Изменение состояния компонента с доступного для ввода на недоступное	37
Map	Помещение компонента в контейнер с применением одного из диспетчеров компоновки	19
Unmap	Удаление компонента из контейнера	18
Expose	Компонент или окно, в котором он находится (или их части), стали видимыми	12
Visibility	Окно (или его часть), в котором находится компонент, стало видимым	15
Configure	Изменение размеров компонента (например, вследствие изменения размеров окна)	22
Destroy	Уничтожение компонента	17

Префиксы указывают на клавиши-модификаторы, которые должны удерживаться, чтобы событие возникло, и количество повторений этого события. Список поддерживаемых модификаторов приведен в таблице 4.

Таблица 4 – Модификаторы, поддерживаемые библиотекой Tkinter

Название	Описание
Double	Событие должно возникнуть дважды в течение короткого промежутка времени
Triple	Событие должно возникнуть трижды в течение короткого промежутка времени
Shift	Должна удерживаться клавиша **<Shift>**
Control	Должна удерживаться клавиша **<Ctrl>**
Alt	Должна удерживаться клавиша **<Alt>**
Any	Отсутствие любых дополнительных условий

Дополнения поддерживаются только двумя событиями:

- **Button** – дополнение указывает номер кнопки мыши, которая была нажата:
  - 1 – левая,
  - 2 – средняя (колесико),
  - 3 – правая,
  - **Key** – любая.

  Можно использовать сокращенную запись вида «номер кнопки». Так, вместо записи `<Button-3>` можно записать `<3>`;
- **KeyPress** – дополнение указывает наименование нажатой клавиши. Здесь также доступна сокращенная запись вида ***<<Наименование клавиши>>***. Например, вместо **<KeyPress-F1>** можно записать просто **<F1>**. Есть только два исключения: клавиша ***Пробел*** обозначается **<space>**, а клавиша ***Символ «меньше»*** – **<less>**. Напомним, что **<Return>** – нажатие клавиши **Enter**.

Рассмотрим несколько примеров написания наименований событий:

- **<Button>** – нажатие кнопки мыши;
- **<Button-1>** или **<1>** – нажатие левой кнопки мыши;
- **<Shift-Button-1>** – нажатие левой кнопки мыши при удерживании клавиши **<Shift>**;

- **<Ctrl-Shift-Button-1>** – нажатие левой кнопки мыши при удерживании клавиш **<Ctrl>** и **<Shift>**;
- **<Double-Button-1>** – двойной щелчок левой кнопкой мыши;
- **<KeyPress-Return>** или **<Return>** – нажатие клавиши **<Enter>**;
- **<Shift-KeyPress-Return>** – нажатие клавиши **<Enter>** при удерживании клавиши **<Shift>**.

В заключение обратим внимание на то, что функция, используемая в качестве второго параметра метода **bind()**, в своем описании обязательно должна иметь один аргумент (в примере он имеет имя **event**), который хранит дополнительные сведения о возникшем событии. В зависимости от возникшего события заполняются соответствующие свойства этого объекта. Укажем некоторые из них (полный список можно получить по адресу: http://it.kgsu.ru/Python_Tk/pytk_008.html).

✓ **x** – горизонтальная координата курсора мыши, вычисленная относительно левого верхнего угла компонента, в виде целого числа в пикселях;

✓ **y** – вертикальная координата курсора мыши, вычисленная относительно левого верхнего угла компонента, в виде целого числа в пикселях;

Пример:

```
def motion_handler(evt):
        x = evt.x
        y = evt.y
```

✓ **num** – целочисленное обозначение нажатой кнопки мыши: 1 – левая, 2 – средняя (колесико), 3 – правая. Используется при обработке событий мыши;

✓ **char** – строка с символом, введенным с клавиатуры, или пустая строка, если была нажата не алфавитно-цифровая клавиша. Используется при обработке событий **KeyPress** и **KeyRelease**;

✓ **keycode** – целочисленный код нажатой клавиши. Используется при обработке событий **KeyPress** и **KeyRelease**;

✓ **keysym** – строковое наименование нажатой клавиши. Используется при обработке событий **KeyPress** и **KeyRelease**:

```
def key_press_handler(evt):
if evt.keysym == "Return":
print("Нажата клавиша <Enter>")
```

✓ **keysym_num** – целочисленное обозначение нажатой клавиши в другом формате. Используется при обработке событий **KeyPress** и **KeyRelease**.

✓ **state** – целое число, указывающее состояние нажатых клавиш-модификаторов или кнопок мыши.

✓ **widget** – компонент, в котором возникло событие;

✓ **type** – целочисленный числовой код возникшего события (см. таблицу 3).

2 Задачи для самостоятельного решения

Выполнить все задачи третьего раздела, реализовав приложения с графическим интерфейсом пользователя.

Лабораторная работа № 9. Использование меню при разработке приложений

Цель работы – познакомить с особенностями создания и использования меню.

1 Фрагмент теории

Меню – это объект, который присутствует во многих пользовательских приложениях. Находится оно под строкой заголовка и представляет собой выпадающие списки под словами; каждый такой список может содержать другой вложенный в него список. Каждый пункт списка представляет собой команду, запускающую какое-либо действие или открывающую диалоговое окно.

Приведем текст приложения, иллюстрирующего создание и использование меню, и прокомментируем его.

```
from tkinter import *

def new_win():
 win = Toplevel(root)


def close_win():
 root.destroy()


def about():
 win = Toplevel(root)
 lab = Label(win, text="Это просто программа-тест \n меню в
                  Tkinter")
 lab.pack()


root = Tk()

m = Menu(root) # создается объект Меню на главном окне
root.config(menu=m) # окно конфигурируется с указанием
 # меню для него

fm = Menu(m, tearoff=0) # создается пункт меню с
 # размещением в основном меню (m)
m.add_cascade(label="Файл", menu=fm) # пункт располагается в
 # основном меню (m)
fm.add_command(label="Открыть...") # формируется список команд
 # пункта меню
fm.add_command(label="Новый", command=new_win)
fm.add_command(label="Сохранить...")
fm.add_separator() # добавляем в меню "Файл" разделитель
fm.add_command(label="Выход", command=close_win)
fm.add_separator() # добавляем в меню "Файл" разделитель

nfm = Menu(fm, tearoff=0)
```

```
fm.add_cascade(label="Импорт",menu=nfm)
nfm.add_command(label="Рисунок")
nfm.add_command(label="Текст")

hm = Menu(m, tearoff=0) # второй пункт меню
m.add_cascade(label="Помощь", menu=hm)
hm.add_command(label="Справка")
hm.add_command(label="О программе", command=about)


root.mainloop()
```

Результат работы приложения приведен на рисунке 2.

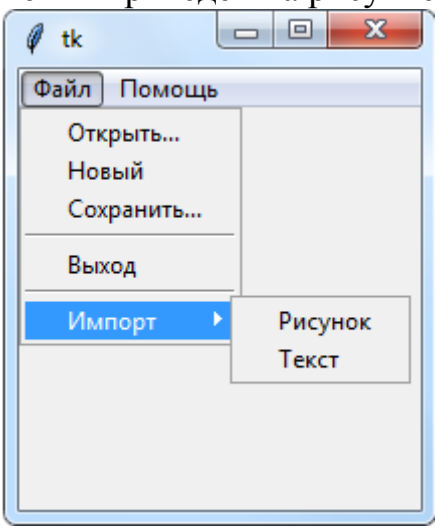

Рисунок 2 – Один из результатов использования меню

Метод **add_cascade()** добавляет новый пункт в меню, который указывается как значение опции **menu**.

Метод **add_command()** добавляет новую команду в пункт меню. Одна из опций данного метода – **command** – связывает данную команду с функцией - обработчиком.

Можно создать вложенное меню. Для этого создается еще одно меню и с помощью метода **add_cascade()** привязывается к родительскому пункту.

Метод **add_separator()** добавляет разделитель – горизонтальную линию (см. рисунок 2), с помощью которой можно визуально сгруппировать команды пункта меню.

Значение 0 опции **tearoff** отключает возможность откреплення подменю, иначе его можно было бы делать плавающим, щелкнув мышью по специальной линии. В случае **tearoff=0** эта линия отсутствует.

Полный список опций виджета **Menu**, а также сведения по настройки его пунктов можно получить здесь: http://it.kgsu.ru/Python_Tk/pytk_063.html.

2 Задачи для самостоятельного решения

Выполнить задания из 4 и 6 разделов, реализовав GUI-приложения с использованием меню. Например, в меню может быть два пункта: «Ввод данных» и «Решение задачи», а все виджеты, относящиеся к соответствующему пункту меню, располагаются в компонентах **Frame (*Панель*)**, которые по очереди ста-

новятся видимыми. Про компонент **Frame** можно прочитать, например, здесь: http://it.kgsu.ru/Python_Tk/pytk_033.html.

Лабораторная работа № 10. Создание приложений, использующих диалоговые окна

Цель работы – познакомить с особенностями использования диалоговых окон при разработке приложений.

1 Фрагмент теории

Библиотека **Tkinter** позволяет использовать в приложениях стандартные диалоговые окна: окна-сообщения различного типа, диалоговые окна открытия и сохранения файла.

1.1 Вывод окон-сообщений

Функциональность вывода стандартных окон-сообщений реализована в модуле **tkinter.messagebox**. Поэтому его обязательно следует импортировать:

```
import tkinter.messagebox
```

Для вывода окон-сообщений различных типов применяются следующие функции:

- **showinfo()** – выводит окно-сообщение со значком в виде синей буквы «**i**» на фоне белого «облачка». Всегда возвращает строку «**ok**». Применяется для вывода оповещений о завершении выполнения какой-либо операции и т. п.;
- **showwarning()** – выводит окно-сообщение со значком в виде черного восклицательного знака на фоне желтого треугольника. Всегда возвращает строку «**ok**». Применяется для оповещения о возможных нештатных ситуациях;
- **showerror()** – выводит окно-сообщение со значком в виде белого крестика в красном кружке. Всегда возвращает строку «**ok**». Применяется для вывода сообщений о критических ошибках;
- **askokcancel()** – выводит окно-предупреждение с кнопками **ОК** и ***Отмена***. Возвращает **True**, если была нажата кнопка **ОК**, и **False** – в противном случае;
- **askyesno()** – выводит окно-предупреждение с кнопками ***Да*** и ***Нет***. Возвращает **True**, если была нажата кнопка ***Да***, и **False** – в противном случае;
- **askyesnocancel()** – выводит окно-предупреждение с кнопками ***Да, Нет*** и ***Отмена***. Возвращает **True**, если была нажата кнопка ***Да,*** **False**, если была нажат кнопка **Нет**, и **None** – в остальных случаях;
- **askquestion()** – выводит окно-предупреждение с кнопками ***Да*** и ***Нет***. Возвращает «**yes**», если была нажата кнопка ***Да***, и «**no**» – в противном случае;

- **askretrycancel ()** – выводит окно-предупреждение с кнопками ***Повтор*** и ***Отмена***. Возвращает **True**, если была нажата кнопка ***Повтор***, и **False** – в противном случае.

Все эти функции имеют одинаковый формат вызова:

```
<Функция>(<Текст заголовка>, <Текст сообщения>[, <Опции окна>])
```

Текст заголовка окна и текст выводящегося в нем сообщения задаются в виде строк. Опции указываются так же, как у компонентов, – путем сопоставления параметров, чьи имена совпадают с опциями, по ключам. Перечень опций можно посмотреть здесь: http://it.kgsu.ru/Python_Tk/pytk_080.html.

Приведем пример использования некоторых из перечисленных функций:

```
from tkinter import *
from tkinter.messagebox import *
from tkinter.ttk import *

def but1():
 # Выводим обычное окно-сообщение
 showinfo("Test", "Сообщение")

def but2():
 # Выводим окно-предупреждение с кнопками "Да" и "Нет"
 # и обрабатываем нажатия на эти кнопки
 if askyesno("Test", "Сообщение", icon=QUESTION):
 print("Была нажата кнопка Да")
 else:
 print("Была нажата кнопка Нет")

def but3():
 # Выводим окно-предупреждение с кнопками "Да" и "Нет",
 # в котором изначально фокус ввода будет иметь кнопка "Нет"
 if askyesno("Test", "Сообщение", default=NO):
 print("Была нажата кнопка Да")
 else:
 print("Была нажата кнопка Нет")

root = Tk()
root.title("Окна-сообщения")

btn1 = Button(root, text="Обычное окно-сообщение", command=but1)
btn1.pack()
btn2 = Button(root, text="Окно-предупреждение_1", command=but2)
btn2.pack()
btn3 = Button(root, text="Окно-предупреждение_2", command=but3)
btn3.pack()

root.mainloop()
```

Результат работы приложения приведен на рисунке 3.

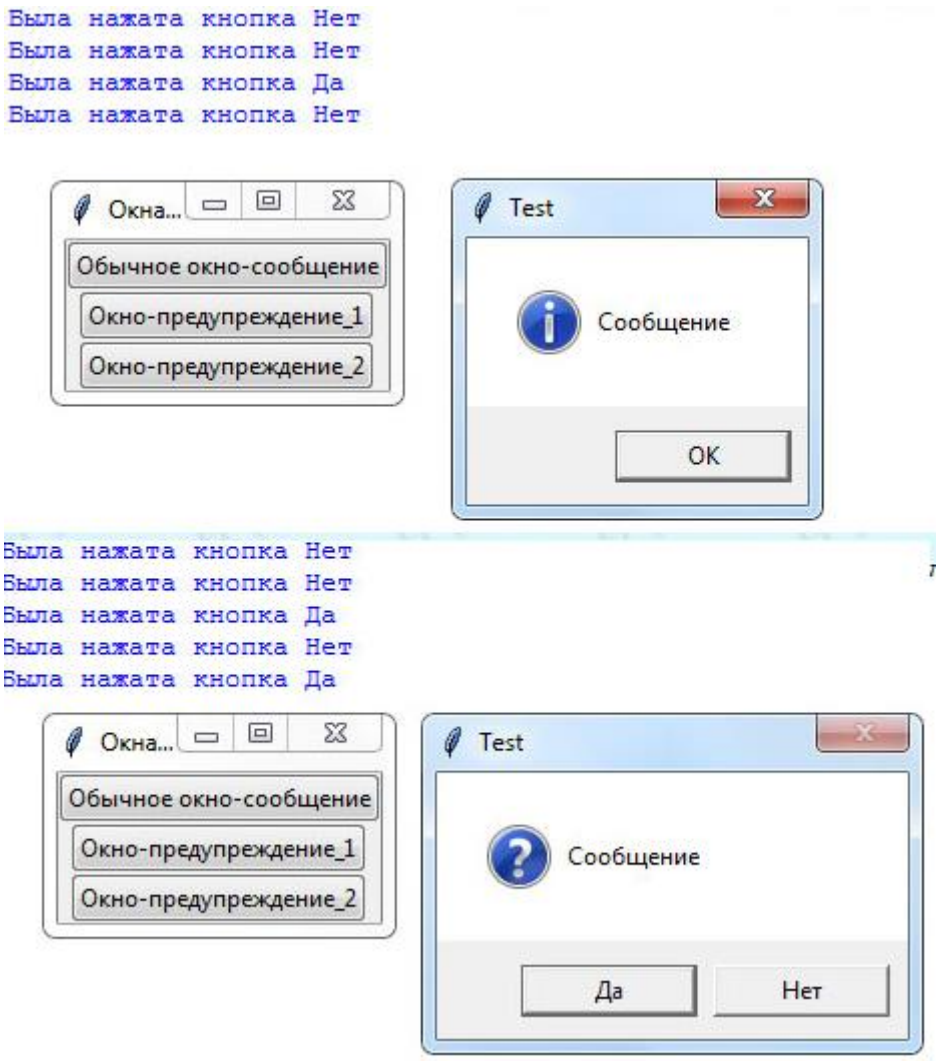

Рисунок 3 – Демонстрация использования различных окон-сообщений

Попробуйте разобраться с принципом работы этого приложения самостоятельно.

1.2 Диалоговые окна работы с файлами

Функциональность вывода стандартных диалоговых окон открытия и сохранения файла реализована в модуле **tkinter.filedialog**. Его обязательно следует импортировать:

```
import tkinter.filedialog
```

В этом модуле определены следующие две функции:

- **askopenfilename ([<*Опции окна*>])** – выводит диалоговое окно открытия файла;
- **asksaveasfilename ([<*Опции окна*>])** – выводит диалоговое окно сохранения файла.

Обе функции возвращают полный путь к указанному файлу или пустую строку, если была нажата кнопка ***Отмена***.

Поддерживаемые диалоговыми окнами опции:

- **title** – задает текст заголовка;
- **filetypes** – задает набор поддерживаемых типов файлов. Значение опции должно представлять собой последовательность, каждый элемент которой зада-

ет один тип файлов и, в свою очередь, должен представлять собой последовательность из двух строк: текстового описания типа и расширения, соответствующего типу. Расширение должно быть указано без начальной точки символами в верхнем регистре;

- **initialdir** – задает начальный каталог, содержимое которого будет выведено в диалоговом окне. Если опция не указана, в качестве начального используется рабочий каталог приложения;
- **defaultextension** – указывает расширение сохраняемого файла по умолчанию. Это расширение добавляется к введенному пользователем имени файла, если последнее не содержит расширения. Задается в виде строки, обязательно с начальной точкой. Значение по умолчанию – «.» (точка). Указывается только для диалогового окна сохранения файла. В диалоговом окне открытия файла игнорируется;
- **initialfile** – указывает начальное имя файла, выбранное в диалоговом окне;
- **parent** – задает ссылку на окно, над которым должно выводиться диалоговое окно. Если опция не указана, диалоговое окно будет выводиться над главным окном приложения.

Приведем пример использования указанных окон.

```
from tkinter.messagebox import *
from tkinter.ttk import *
from tkinter.filedialog import *

def open_file():
 # Выводим диалоговое окно открытия файла
 filename = askopenfilename(title="Открыть файл",
 filetypes=(("Текстовые файлы", "TXT"),))
 if filename:
 # == Пользователь выбрал файл. Открываем его.
 text_delete() # Предварительно очищаем поле
 # Открываем файл
 file = open(filename, "r", encoding="cp1251")
 tmp = file.readline() # Читаем очередную строку
 while tmp != '': # Если есть, что читать...
 text.insert(END, tmp) # ... помещаем в конец текста
 tmp = file.readline()
 file.close() # Закрываем файл
 else:
 # Пользователь отказался открывать файл
 pass

def save_file():
 # Выводим диалоговое окно сохранения файла
 filename = asksaveasfilename(title="Сохранить файл",
 filetypes=(("Текстовые файлы", "TXT"),
 ("Файлы CSV", "CSV")),
 defaultextension=".txt", initialdir="d:\\")
 context = text.get(1.0, END) # Читаем все содержимое поля
 file = open(filename, "w") # Открываем файл на запись
```

```
  file.write(context) # Сохраняем прочитанное
  # содержимое
  file.close() # Закрываем файл

def text_delete():
  # Очишаем текстовое поле
  text.delete('1.0', END)

root = Tk()
root.title("Окна открытия и сохранения файла")

btn1 = Button(root, text="Открытие файла", command=open_file)
btn1.pack()
btn2 = Button(root, text="Сохранение файла", command=save_file)
btn2.pack()
btn3 = Button(root, text="Удаление текста", command=text_delete)
btn3.pack()
# Поле для ввода текста
text = Text(root, width=40, height=10, font="Verdana 12")
text.pack()

root.mainloop()
```

Результат использования диалогового окна сохранения файла приведен на рисунке 4. Надеемся, что приведенных комментариев будет достаточно, чтобы разобраться с принципами работы приложения.

1.3 Основные принципы работы с файлами

Прежде чем работать с файлом, необходимо создать объект файла с помощью функции **open()**. Функция имеет следующий формат:

```
open(<Путь к файлу>[, mode = 'r'] [, buffering=-1] [, encoding=None] [, errors=None][, newline=None][, closefd=True])
```

В первом параметре указывается имя файла, возможно, вместе с путем доступа. Остальные параметры не являются обязательными, и их назначение можно посмотреть, например в [1].

Перечислим основные методы работы с файлами:

- **close()** – закрывает файл;
- **write(<*Данные*>)** – записывает строку или последовательность байтов в файл. Если в качестве параметра указана строка, то файл должен быть открыт в текстовом режиме. Для записи последовательности байтов необходимо открыть файл в бинарном режиме. Метод возвращает количество записанных символов или байтов;
- **writelines(<*Последовательность*>)** – записывает последовательность в файл. Если все элементы последовательности являются строками, то файл должен быть открыт в текстовом режиме. Если все элементы являются последовательностями байтов, то файл должен быть открыт в бинарном режиме;
- **writable()** – возвращает **True**, если файл поддерживает запись, и **False** – в противном случае;

- **read([<*Количество*>])** – считывает данные из файла. Если файл открыт в текстовом режиме, то возвращается строка, а если в бинарном – последовательность байтов. Если параметр не указан, возвращается содержимое файла от текущей позиции указателя до конца файла;
- **readline([<*Количество*>])** – считывает из файла одну строку при каждом вызове. Если файл открыт в текстовом режиме, то возвращается строка, а если в бинарном – последовательность байтов. Возвращаемая строка включает символ перевода строки. Исключением является последняя строка – если она не завершается символом перевода строки, то таковой добавлен не будет. При достижении конца файла возвращается пустая строка;
- **readlines()** – считывает все содержимое файла в список. Каждый элемент списка будет содержать одну строку, включая символ перевода строки. Исключением является последняя строка. Если она не завершается символом перевода строки, то символ перевода строки добавлен не будет. Если файл открыт в текстовом режиме, то возвращается список строк, а если в бинарном – список объектов типа **bytes**;
- **flush()** – принудительно записывает данные из буфера на диск;
- **fileno()** – возвращает целочисленный дескриптор файла. Возвращаемое значение всегда будет больше числа 2, т. к. число 0 закреплено за стандартным вводом **stdin**, 1 – за стандартным выводом **stdout**, а 2 – за стандартным выводом сообщений об ошибках **stderr**.
- **truncate([<*Количество*>])** – обрезает файл до указанного количества символов (если задан текстовый режим) или байтов (в случае бинарного режима). Метод возвращает новый размер файла;
- **tell()** – возвращает позицию указателя относительно начала файла в виде целого числа. Обратите внимание на то, что в **Windows** метод **tell()** считает символ «\r» как дополнительный байт, хотя этот символ удаляется при открытии файла в текстовом режиме;
- **seek(<*Смещение*>[, <*Позиция*>])** – устанавливает указатель в позицию, имеющую смещение *Смещение* относительно позиции *Позиция*. В параметре *Позиция* могут быть указаны следующие атрибуты из модуля **io** или соответствующие им значения:
  - **io.SEEK_SET** или 0 – начало файла (значение по умолчанию);
  - **io.SEEK_CUR** или 1 – текущая позиция указателя. Положительное значение смещения вызывает перемещение к концу файла, отрицательное – к его началу;
  - **io.SEEK_END** или 2 – конец файла;
- **seekable()** – возвращает **True**, если указатель файла можно сдвинуть в другую позицию, и **False** – в противном случае.

2 Задачи для самостоятельного решения

Во всех задачах нужно организовать ввод данных с последующим сохранением в выбранный пользователем файл, очистку поля ввода, загрузку данных

из файла. Эти операции, а также само решение задачи, можно реализовать, например, через выбор соответствующих пунктов меню.

10.1 Дан файл f, компоненты которого являются действительными числами. Найти: а) сумму компонент файла f; б) произведение компонент файла f; в) сумму квадратов компонент файла f; г) модуль суммы и квадрат произведения компонент файла f; д) последнюю компоненту файла.

10.2 Дан файл f, компоненты которого являются действительными числами. Найти: а) наибольшее из значений компонент; б) наименьшее из значений компонент с четными номерами; в) наибольшее из значений модулей компонент с нечетными номерами; г) разность первой и последней компонент файла.

10.3 Дан файл f, компоненты которого являются целыми числами. Найти: а) количество четных чисел среди компонент; б) количество удвоенных нечетных чисел среди компонент; в) количество квадратов нечетных чисел среди компонент.

10.4. Дано натуральное n. Записать в файл g целые числа b_1, ..., b_n. При значениях i=1, 2 ,..., n значение b_i равно: а) i; б) i*i ; в) i!; г) 2^{i+1}; д) $2^i+ 3^{i+1}$.

10.5 Последовательность x_1, x_2,... образована по закону:

$$x_i = \frac{i - 0.1}{i^3 + |tg2i|} \qquad (i = 1,2,3,\ldots).$$

Дано действительное E>0. Записать в файл h члены последовательности x_1, x_2,..., остановившись после первого члена, для которого выполнено $|x_i|<E$.

10.6 Даны файлы f1, f2, f3, f4, f5, компоненты которых являются действительными числами. Организовать обмен компонентами между файлами в соответствии со следующей схемой:

f1->f3 f2->f4 f3->f5 f4->f2 f5->f1,

т. е. компоненты файла f1 переписываются в файл f3, компоненты файла f2 – в f4 и т. д. Разрешается использовать только один вспомогательный файл h.

10.7 Дан символьный файл f. В файле не менее двух компонент. Определить, являются ли два первых символа файла цифрами. Если да, то установить, является ли число, образованное этими цифрами, четным.

10.8 Дан файл f, компоненты которого являются целыми числами. Получить в файле g все компоненты файла f: а) являющиеся четными числами; б) делящиеся на 3 и не делящиеся на 7; в) являющиеся точными квадратами.

10.9 Дан файл f, компоненты u_0, u_1, ..., u_n которого являются последовательными числами Фибоначчи. Получить в файле f последовательно числа Фибоначчи: u_0, u_1, ..., u_{n+1}.

10.10 Дан символьный файл f. Получить файл g, образованный из файла f заменой всех его прописных (больших) букв одноименными строчными (малыми).

10.11 Дан файл f, компоненты которого являются целыми числами. Записать в файл g все четные числа файла f, а в файл h – все нечетные. Порядок следования чисел сохраняется.

10.12 Дан символьный файл f. Записать в файл g компоненты файла f в обратном порядке.

10.13 Дан файл f, компоненты которого являются целыми числами. Получить файл g, образованный из файла f исключением повторных вхождений одного и того же числа.

10.14 Дан файл f, компоненты которого являются целыми числами. Никакая из компонент файла не равна нулю. Файл f содержит столько же отрицательных чисел, сколько и положительных. Используя вспомогательный файл h, переписать компоненты файла f в файл g так, чтобы в файле g: а) не было двух соседних чисел с одним знаком; б) сначала шли положительные, потом отрицательные числа; в) числа шли в следующем порядке: два положительных, два отрицательных, два положительных, два отрицательных и т. д. (предполагается, что число компонент в файле f делится на 4).

10.15 Дан символьный файл f, содержащий сведения о сотрудниках учреждения, записанные по следующему образцу: фамилия_имя_отчество, фамилия_имя_отчество, ... Записать эти сведения в файле g, используя образцы: а) имя_отчество_фамилия, имя_отчество_фамилия, ...; б) фамилия_и.о., фамилия_и.о., ...

10.16 Дан символьный файл f. Необходимо: а) подсчитать число вхождений в файл сочетаний «ab»; б) определить, входит ли в файл сочетание «abcdefgh»; в) подсчитать число вхождений в файл каждой из букв a, b, c, d, e, f и вывести результат в виде таблицы:

a – Na	b – Nb	c – Nc
d – Nd	e – Ne	f – Nf

где Na, Nb, Nc, Nd, Ne, Nf – числа вхождений соответствующих букв.

10.17 Даны символьные файлы f и g. Определить, совпадают ли компоненты файла f с компонентами файла g. Если нет, то получить номер первой компоненты, в которой файлы f и g отличаются между собой. В случае, когда один из файлов имеет n компонент (n>=0) и повторяет начало другого (более длинного) файла, ответом должно быть число n+1.

10.18 Дан символьный файл f. Группы слов, разделенных пробелами (одним или несколькими) и не содержащие пробелов внутри себя, будем называть словами. Удалить из файла все однобуквенные слова и лишние пробелы. Результат записать в файл g.

10.19 Дан символьный файл f. Считая, что количество символов в слове не превосходит двадцати: а) определить, сколько в файле f имеется слов, состоящих из одного, двух, трех и т. д. символов; б) получить гистограмму (столбчатую диаграмму) длин всех слов файла f; в) определить количество слов в файле f.

10.20 Даны два символьных файла f1 и f2. Файл f1 содержит произвольный текст. Слова в тексте разделены пробелами и знаками препинания. Файл f2 содержит не более 40 слов, которые разделены запятыми. Эти слова образуют пары: каждое первое слово считается заменяемым, каждое второе слово – заменяющим. Найти в файле f1 все заменяемые слова и заменить их на соответствующие заменяющие. Результат поместить в файле g.

10.21 Дан текстовый файл f. Переписать в файл g все компоненты файла f с заменой в них символа «0» на символ «1» и наоборот.

10.22 Дан файл f, содержащий сведения об экспортируемых товарах: указывается наименование товара, страна, импортирующая товар, и объем поставляемой партии в штуках. Найти страны, в которые экспортируется данный товар и общий объем его экспорта.

10.23 Сведения об автомобиле состоят из его марки, номера и фамилии владельца. Дан файл f, содержащий сведения о нескольких автомобилях. Найти: а) фамилии владельцев и номера автомобилей данной марки; б) количество автомобилей каждой марки.

10.24 Дан файл f, содержащий различные даты. Каждая дата – это число, месяц и год. Найти: а) год с наименьшим номером; б) все весенние даты; в) самую позднюю дату.

Лабораторная работа № 11. Графика. Виджет Canvas

Цель работы – познакомиться с возможностями этого виджета.

1 Фрагмент теории

Виджет **Canvas** позволяет располагать другие виджеты, но, в основном, используется для отображения векторной графики.

Для того чтобы создать объект холста необходимо вызвать его конструктор и установить значения его свойств. Например:

```
canv = Canvas(root, width=480, height=360, bg='#faffff',
                  cursor="pencil")
canv.pack(anchor=CENTER, expand=1)
```

Здесь, используя один из диспетчеров компоновки, холст размещается в главном окне. После этого можно приступить к рисованию геометрических фигур, форма и положение которых обычно управляется координатами точек. Начало координат (0, 0) объекта **Canvas** располагается в верхнем левом углу; направление осей **X** и **Y** – вправо и вниз.

Перечислим некоторые параметры этого виджета (параметры, которые можно задавать при создании холста):

- **background (bg)** – фоновый цвет;
- **border (bd)** – граница;
- **borderwidth** – толщина границы;
- **cursor** – курсор;
- **height** – высота виджета;
- **width** – ширина виджета.

Для двухмерного рисования **Canvas** предоставляет ряд методов:

- **create_line()** – рисует линию.
- **create_rectangle()** – рисует прямоугольник;
- **create_oval()** – рисует овал;
- **create_arc()** – рисует дугу;
- **create_polygon()** – рисует многоугольник;
- **create_text()** – добавляет текст;

- **create_image()** – добавляет изображение;
- **create_window()** – добавляет виджет.

В качестве результата все эти методы возвращают идентифтикатор добавленного элемента. Этот идентификатор в дальнейшем может использоваться для управления элементом. Рассмотрим применение этих методов.

Рисование простейшей линии:

```
canv.create_line(10, 10, 200, 50, activefill="red",
                 fill="blue", dash=2)
```

Здесь достаточно задать четыре числа, которые определяют, соответственно, координаты начала и конца отрезка.

Кроме того, у данного метода можно выделить ряд дополнительных параметров:

- **arrow** – помещает стрелку в начале линии (значение **first**), в конце (**last**) или на обоих концах (**both**);
- **arrowshape** – позволяет изменить форму стрелки;
- **capstyle** – если линия не имеет стрелки, то устанавливает, как завершается линия. Принимает значения: **butt** (по умолчанию), **projecting** и **round**;
- **joinstyle** – управляет соединением сегментов линии. Принимает значения: **round** (по умолчанию), **bevel** и **miter**;
- **smooth** – если значение **«true»** или **«bezier»**, сглаживает сегменты линии;
- **splinesteps** – управляет сглаживанием изогнутых линий.

Методы отрисовки имеют ряд параметров, которые позволяют настроить стилизацию фигур. Некоторые из этих параметров:

- **fill** – цвет заполнения фигуры;
- **width –** ширина линий;
- **outline** – для заполненных фигур цвет контура;
- **dash –** устанавливает пунктирную линию;
- **stipple** – устанавливает шаблон для заполнения фигуры (например, **gray75**, **gray50**, **gray25**, **gray12**);
- **activefill** – цвет заполнения фигуры при наведении курсора;
- **activewidth** – ширина линий при наведении курсора;
- **activestipple** – шаблон заполнения фигуры при наведении курсора.

Таким образом, в примере нарисована синим цветом пунктирная линия. При наведении на нее указателя мыши, она окрашивается в красный цвет.

При *отрисовке прямоугольника* также задаются четыре числа, определяющие координаты левого верхнего и правого нижнего углов требуемого прямоугольника.

При *изображении овала*, назначение этих четырех чисел такое же: они определяют прямоугольник, в который будет *вписан* требуемый овал (или окружность, если задан квадрат). В отличие от предыдущих случаев, прямоугольник не отображается.

В отличие от предыдущих случаев, при *построении многоугольника* методом **create_polygon()** здесь можно задавать больше двух пар чисел, которые определяют координаты соединяемых точек. Первая и последняя точка будут соеди-

нены. Поэкспериментируйте со свойством **smooth**; постройте многоугольник по одному и тому же множеству точек. Но в одном случае используйте опцию **smooth=0** (0 – значение по умолчанию), а во втором случае – **smooth=1**. В первом случае будет построена ломаная, проходящая через точки (x_i, y_i). Если **smooth=1**, то будет построена гладкая кривая. Ее форма только управляется положением этих точек, и обычно кривая через них не проходит.

Сектор, сегмент и *дуга* создаются методом **canvas.create_arc()**. Управляет формой опция **style**. Если она не задана, то рисуется сектор. При **style=CHORD** будет нарисован сегмент, а при **style=ARC** – дуга:

```
canv.create_arc([200,230], [270,330], start=0, extent=135,
 fill="lightgreen")
canv.create_arc([210,120], [280,220], start=0, extent=135,
 style=CHORD, fill="green")
canv.create_arc([340,230], [410,330], start=45, extent=270,
 style=ARC, outline="darkgreen", width=2)
```

Здесь опция **start** определяет начальный угол (0 – совпадает с осью **X**), а **extend** задает угол поворота (положительное значение – поворот против часовой стрелки, отрицательное – по часовой).

Если опция **fill** не задана, то будет нарисован контур соответствующей фигуры.

На холсте можно *разместить текст*. Делается это с помощью метода **create_text()**:

```
canv.create_text(100, 100, text="Hello World,\nPython\nand Tk",
 justify=CENTER, font="Verdana 14")
canv.create_text(200, 200, text="About this",
                  anchor=SE, fill="grey")
```

По умолчанию в заданной координате располагается центр текстовой надписи. Чтобы изменить это и, например, разместить по указанной координате левую границу текста, используется якорь со значением **W**. Другие значения: **N, NE, E, SE, S, SW, W, NW**. Если букв, задающих сторону привязки, две, то вторая определяет вертикальную привязку (вверх или вниз «уйдет» текст от заданной координаты). Опция **justify** определяет лишь выравнивание текста относительно себя самого.

Для *вывода изображения* применяется метод **create_image()**, который в качестве обязательного параметра принимает координаты изображения. Для установки самого изображения в метод через опцию **image** передается ссылка на изображение:

```
python_image = PhotoImage(file="python.png")
canv.create_image(10, 10, anchor=NW, image=python_image)
```

В данном случае координаты представлены точкой с **x=10** и **y=10**, а изображение представляет объект **PhotoImage** (здесь предполагается, что в одной папке с файлом программы находится файл **«python.png»**). Но, как и в случае с выводом текста, следует учитывать, что по умолчанию координаты представляют центр изображения. Чтобы настроить положение изображения относительно координат, применяется параметр **anchor**. Так, в данном случае значение **"NW"** означает, что координата представляет верхний левый угол изображения.

Одной из замечательных особенностей **Canvas** является то, что он позволяет добавлять другие виджеты и, таким образом, создавать сложные по композиции интерфейсы. Для этого применяется метод **create_window()**. Пример:

```
btn = Button(text="Нажми меня")
canv.create_window(10, 20, anchor=NW, window=btn,
                   width=100, height=50)
```

В данном случае верхний левый угол кнопки будет иметь координаты (10, 20), а сама кнопка имеет ширину 100 и высоту 50 пикселей. Если ширина и высота явным образом не указаны, то они имеют значения по умолчанию.

Более полную информацию по этому виджету можно получить, например, здесь: http://it.kgsu.ru/Python_Tk/pytk_069.html.

2 Задачи для самостоятельного решения

Во всех задачах, помимо построения, нужно ввести с клавиатуры значения параметров. Требуется закрасить полученную фигуру и подписать ее. Дополнительное задание: заставьте фигуру перемещаться при нажатии на клавиши управления курсором.

11.1 Построить равнобедренный треугольник с основанием a и высотой h.

11.2 Построить ромб, диагонали которого равны a и b. Ромб расположить так, чтобы одна из его диагоналей была горизонтальной.

11.3 Построить равнобедренную трапецию, высота которой равна h, а основания – a и b.

11.4 Построить треугольник по сторонам a, b и углу между ними C (в градусах).

11.5 Построить треугольник по стороне a и двум прилежащим к ней углам B и C (в градусах).

11.6 Построить ромб по стороне a и острому углу R (в градусах).

11.7 Построить правильный шестиугольник со стороной a.

11.8 Построить правильный пятиугольник со стороной a.

11.9 Построить правильный n-угольник со стороной a.

11.10 Построить треугольник с основанием a, высотой h и углом при вершине X (в градусах).

11.11 Построить квадрат, на одной стороне которого, как на основании, построен равносторонний треугольник со стороной a.

11.12 Построить равнобедренный треугольник с основанием a и высотой h.

11.13 Построить треугольник со сторонами a, b, c.

11.14 Построить квадрат, описанный около окружности с радиусом R.

11.15 Построить квадрат, вписанный в окружность радиуса R.

11.16 Построить треугольник, описанный около окружности с радиусом R.

11.17 Построить треугольник, вписанный в окружность радиуса R.

11.18 Построить равносторонний треугольник с длиной стороны a.

11.19 Построить равнобедренную трапецию, высота которой равна h, одно основание равно a и средняя линия равна b.

11.20 Построить параллелограмм по сторонам a, b и углу между ними A (в градусах).

11.21 Построить параллелограмм по диагоналям d1, d2 и углу между ними X (в градусах).

11.22 Построить ромб по стороне a и тупому углу R (в градусах).

11.23 Построить равнобедренную трапецию по основанию a, боковой стороне b и углу между ними Y (в градусах).

11.24 Построить равнобедренную трапецию по основанию a, прилежащему углу X (в градусах) и средней линии b.

РАСПРЕДЕЛЕНИЕ ЗАДАЧ ПО ВАРИАНТАМ

Задача 1.31 решается при выполнении ***лабораторной работы № 7***!

Вместо выполнения лабораторных работ студент, по согласованию с преподавателем, может взять на реализацию проект, который в конце семестра должен защитить. По итогам защиты определяется возможность выставления зачета. Примеры проектов: калькулятор, графический редактор, текстовый редактор и т. п.

ВАРИАНТ 1: 1.1, 1.22, 1.31(1), 2.1, 2.10, 3.2, 3.3, 3.31, 4.2, 5.1, 6.1, 10.1, 11.15.

ВАРИАНТ 2: 1.2, 1.23, 1.31(2), 2.2, 2.11, 3.1, 3.5, 3.32, 4.3, 5.2, 6.2, 10.2, 11.14.

ВАРИАНТ 3: 1.5, 1.21, 1.31(3), 2.3, 2.14, 3.7, 3.8, 3.33, 4.1, 5.4, 6.10, 10.3, 11.12.

ВАРИАНТ 4: 1.12, 1.20, 1.31(4), 2.4, 2.15, 3.10, 3.34, 3.16, 4.4, 5.5, 6.3, 10.4, 11.11.

ВАРИАНТ 5: 1.11, 1.19, 1.31(5), 2.5, 2.16, 3.11, 3.14, 3.35, 4.5, 5.6, 6.11, 10.5, 11.4.

ВАРИАНТ 6: 1.10, 1.18, 1.31(6), 2.6, 2.17, 3.12, 3.15, 3.36, 4.6, 5.7, 6.7, 10.6, 11.2.

ВАРИАНТ 7: 1.9, 1.17, 1.31(7), 2.7, 2.22, 3.4, 3.9, 3.37, 4.7, 5.3, 6.4, 10.8, 11.1.

ВАРИАНТ 8: 1.8, 1.16, 1.31(8), 2.8, 2.23, 3.20, 3.22, 3.38, 4.8, 5.10, 6.6, 10.7, 11.5.

ВАРИАНТ 9: 1.7, 1.24, 1.31(9), 2.9, 2.24, 3.24, 3.26, 3.39, 4.9, 5.11, 6.8, 10.9, 11.3.

ВАРИАНТ 10: 1.6, 1.25, 1.31(10), 2.12, 2.25, 3.28, 3.30, 3.40, 4.11, 6.9, 10.10, 11.13.

ВАРИАНТ 11: 1.13, 1.26, 1.31(11), 2.13, 2.26, 3.16, 3.30, 3.41, 4.10, 5.13, 6.5, 10.11, 11.10.

ВАРИАНТ 12: 1.4, 1.27, 1.31(12), 2.18, 2.27, 3.17 3.29, 3.42, 4.19, 5.14, 6.12, 10.15, 11.6.

ВАРИАНТ 13: 1.3, 1.28, 1.31(13), 2.19, 2.28, 3.18, 3.28, 3.44, 4.16, 5.15, 6.15, 10.14, 11.7.

ВАРИАНТ 14: 1.14, 1.29, 1.31(14), 2.20, 2.29, 3.19, 3.27, 3.43, 4.13, 5.9, 6.14, 10.13, 11.10.

ВАРИАНТ 15: 1.15, 1.30, 1.31(15), 2.21, 2.30, 3.21, 3.23, 3.45, 4.12, 5.8, 6.13, 10.12, 11.11.

ВАРИАНТ 16: 1.33, 1.44, 1.31(16), 2.31, 2.41, 3.55, 3,67, 3.47, 4.20, 5.16, 6.21, 10.17, 11.16.

ВАРИАНТ 17: 1.34, 1.42, 1.31(17), 2.32, 2.42, 3.56, 3.68, 3.49, 4.22, 5.17, 6.22, 10.18, 11.17.

ВАРИАНТ 18: 1.35, 1.45, 1.31(18), 2.33, 2.40, 3.57, 3.69, 3.48, 4.23, 5.20, 6.16, 10.21, 11.18.

ВАРИАНТ 19: 1.36, 1.43, 1.31(19), 2.34, 2.43, 3.58, 3.70, 3.50, 4.14, 5.22, 6.17, 10.22, 11.19.

ВАРИАНТ 20: 1.37, 1.47, 1.31(20), 2.35, 2.47, 3.59, 3.71, 3.51, 4.15, 5.23, 6.18, 10.23, 11.20.

ВАРИАНТ 21: 1.38, 1.46, 1.31(21), 2.36, 2.44, 3.60, 3.61, 3.52, 4.21, 5.21, 6.10, 10.24, 11.21.

ВАРИАНТ 22: 1.39, 1.50, 1.31(22), 2.37, 2.45, 3.64, 3.62, 3.53, 4.18, 5.19, 6.20, 10.19, 11.22.

ВАРИАНТ 23: 1.40, 1.49, 1.31(23), 2.38, 2.46, 3.65, 3.63, 3.54, 4.24, 5.18, 6.23, 10.16, 11.23.

ВАРИАНТ 24: 1.41, 1.32, 1.31(24), 2.39, 2.48, 3.66, 3.72, 3.46, 4.25, 5.24, 6.24, 10.20, 11.24.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 Язык программирования Python. Начала. – URL: http://it.kgsu.ru/ Python/oglav.html (дата обращения: 11.07.23).

2 Библиотека Tkinter. Основы разработки оконных приложений. – URL: http://it.kgsu.ru/Python_Tk/oglav.html (дата обращения: 11.07.23)

3 Прохоренок Н. А. Python 3. Самое необходимое / Н. А. Прохоренок, В. А. Дронов. – Санкт-Петербург : 2-е изд., перераб. и доп, 2019. – 610 с.

Медведев Аркадий Андреевич

ИЗУЧЕНИЕ ЯЗЫКА ПРОГРАММИРОВАНИЯ PYTHON

Методические рекомендации к выполнению лабораторных работ
для студентов направлений
01.03.01, 01.05.01 «Математика»,
«Фундаментальная математика и механика»

Редактор М. А. Лаврентьева

……
Подписано в печать __.__.___ Формат 60x84 1/16 Бумага 65 г/м^2
Печать цифровая Усл. печ. л. 3,5 Уч.-изд. л. 3,5
Заказ № Тираж э/в
……
БИЦ Курганского государственного университета.
640020, г. Курган, ул. Советская, 63/4.
Курганский государственный университет.