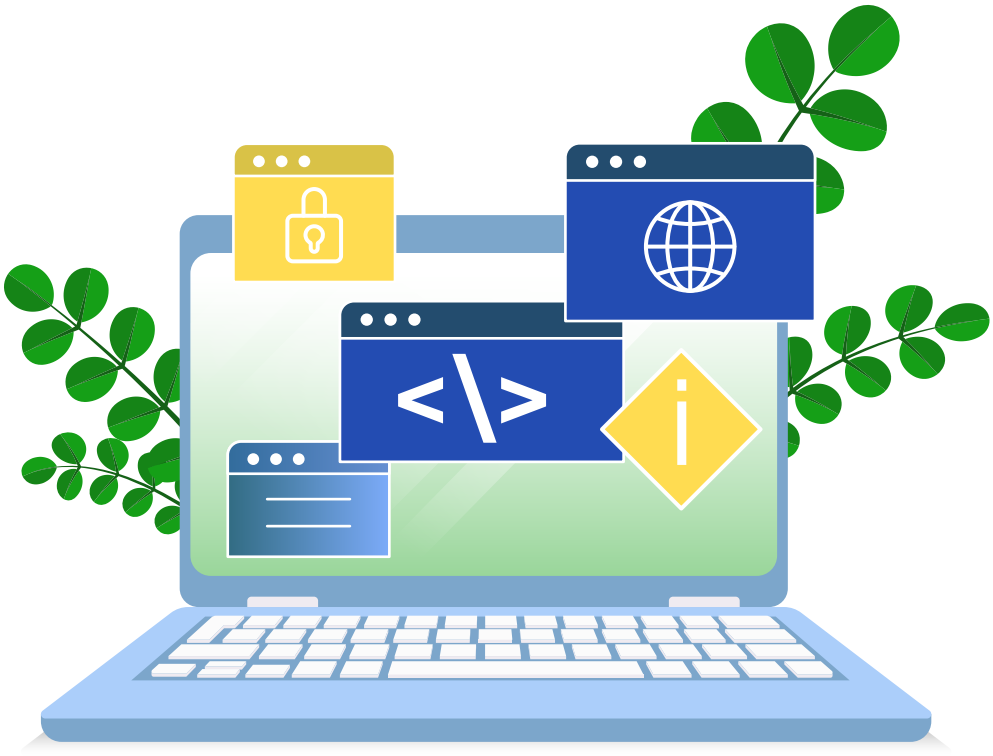
Министерство образования и науки Кыргызской Республики
Некоммерческое образовательное учреждение
Учебно-научно-производственный комплекс
«Международный Университет Кыргызстана»

САВЧЕНКО Е. Ю.
МУСАКУЛОВА Ж. А.

ПРОГРАММИРОВАНИЕ НА PYTHON:

ОТ БАЗОВЫХ КОНЦЕПЦИЙ К СОЗДАНИЮ GUI



УЧЕБНОЕ ПОСОБИЕ

Министерство образования и науки Кыргызской Республики
Некоммерческое образовательное учреждение
Учебно-научно-производственный комплекс
«Международный Университет Кыргызстана»

САВЧЕНКО Е. Ю.
МУСАКУЛОВА Ж. А.

ПРОГРАММИРОВАНИЕ НА PYTHON:

ОТ БАЗОВЫХ КОНЦЕПЦИЙ К СОЗДАНИЮ GUI

УЧЕБНОЕ ПОСОБИЕ

Бишкек • 2024

УДК 004.43
ББК 32.973.2
С-13

Рекомендовано к изданию ученым советом УНПК МУК

Допущено Министерством образования и науки
Кыргызской Республики в качестве учебного пособия
для студентов высших учебных заведений.

Приказ МОиН КР № 671/1, от 26.04.2024 г.

Рецензенты:
Мусина И. Р. — к. т. н., доцент кафедры «Программное обеспечение компьютерных систем» КГТУ им. И. Раззакова.
Нежинских С. С. — к. т. н., и. о. доцента кафедры «Компьютерных информационных систем и управления» УНПК МУК.

ISBN 978-9967-????-?-?

Савченко Е. Ю., Мусакулова Ж. А.
С-13 **Программирование на Python: от базовых концепций к созданию GUI** Учеб. пособие / Е. Ю. Савченко, Ж. А. Мусакулова – Б.: Нео Принт, 2024. – ???? с.: ил.

Учебное пособие «Программирование на Python: от базовых концепций к созданию GUI» представляет собой комплексный материал, охватывающий ключевые аспекты языка Python.

В пособии рассмотрены основы языка, включая условные операторы и циклы, работу со структурами данных (списки, словари, кортежи), обработку исключений и работу с файлами, функции и классы, а также использование библиотеки Tkinter для создания графического интерфейса пользователя. Каждый раздел содержит теоретический материал, практические примеры и контрольные вопросы для закрепления полученных знаний. Пособие предназначено как для самостоятельного изучения, так и для использования в учебном процессе при подготовке специалистов в области информационных технологий.

УДК 004.43
ББК 32.973.2

ISBN 978-9967-????-?-?

СОДЕРЖАНИЕ или ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Данное учебное пособие по программированию на языке Python представляет собой структурированный и систематизированный курс, разработанный для студентов, стремящихся овладеть основами языка программирования. Python, считающийся одним из наиболее распространенных и востребованных в индустрии информационных технологий, обеспечивает эффективное решение широкого спектра задач.

Пособие включает обширный материал, начиная с фундаментальных элементов, таких как переменные, условия, циклы и функции, включая работу с объектами, обработку ошибок и структуры данных.

Язык программирования Python выделяется своей простотой и выразительностью синтаксиса, что делает его доступным для понимания и использования. Язык разработан с учетом читаемости кода, что способствует быстрой и эффективной разработке программ. Эта характеристика делает Python идеальным инструментом как для начинающих, так и для профессионалов, стремящихся улучшить свои навыки.

Программирование на Python охватывает различные парадигмы, включая объектно-ориентированное и процедурное программирование. Этот язык обладает портативностью и кроссплатформенностью, позволяя разрабатывать приложения, которые легко могут быть запущены на различных операционных системах.

Раздел 1

ОСНОВЫ ЯЗЫКА PYTHON

Введение в Python

Python — это интерпретируемый объектно-ориентированный язык программирования высокого уровня, является кроссплатформенным языком.

Программа на языке Python представляет собой текстовый файл с расширением py (консольная программа) или pyw (программа с графическим интерфейсом). Для установки интерпретатора Python можно скачать дистрибутив с сайта https://www.python.org/downloads.

Программа на языке Python представляет собой текстовый файл с инструкциями. Каждая инструкция располагается на отдельной строке. Если инструкция не является вложенной, то она должна начинаться с начала строки, иначе будет выведено сообщение об ошибке.

В языке Python концом инструкции является конец строки, точку с запятой ставить не рекомендуется (если только необходимо разместить несколько инструкции на одной строке).

В языке Python отсутствуют ограничительные символы для выделения инструкций внутри блока (**Например:** фигурные скобки или команды **begin/end**).

В языке Python принято использовать четыре пробела для выделения инструкций внутри блока. Инструкции, перед которыми расположено одинаковое количество пробелов, являются телом блока.

В программном коде можно использовать комментарии, которые служат для пояснения кода, улучшения его читаемости и облегчения понимания функциональности. В Python однострочный комментарий

выделяется с помощью символа «решётка» **#**. Специального многострочного комментария нет, но можно расположить фрагмент кода внутри утроенных кавычек (или утроенных апострофов).

Вывод данных из программы осуществляется с помощью команды (функции) **print()**. Ввод данных в программу осуществляется с помощью команды (функции) **input()**.

Например:

```
name = input("Введите ваше имя: ")
print("Привет, " + name + "!")
```

Результат:

```
Введите ваше имя: Бакыт
Привет, Бакыт!
```

Функция **input()** используется для ввода данных с клавиатуры. Она считывает введенную строку, и ее можно сохранить в переменную.

Функция **input()** всегда возвращает строку, поэтому, если необходимо ввести числовое значение, надо явно преобразовать его в соответствующий тип данных.

Функция **print()** используется для вывода данных на экран, которой можно передавать несколько аргументов, разделяя их запятой.

Переменные

Все данные в языке Python представлены объектами. Каждый объект имеет тип данных и значение. Для доступа к объекту предназначены переменные.

Переменная в программировании на Python — это именованное хранилище, которое содержит какое-то значение данных. В Python переменные создаются путем присваивания значения определенному имени. В отличие от некоторых других языков программирования, в Python переменные не требуют объявления типа данных. Тип данных переменной определяется автоматически в момент присваивания значения.

При инициализации в переменной сохраняется ссылка на объект. Благодаря этой ссылке можно в дальнейшем изменять объект из программы.

Каждая переменная должна иметь уникальное имя. В качестве переменной нельзя использовать ключевые слова.

Получить список всех ключевых слов можно, используя команды:

```
import keyword
keyword.kwlist
```

Имена переменных также должны отличатся от встроенных идентификаторов (они выделяются фиолетовым цветом).

Цвет переменной (имени переменной) выделяется черным цветом.

В языке учитывается регистр букв. То есть переменные **X** и **x** разные.

Типы данных

- `bool` — логический тип данных.
- `int` — целые числа.
- `Nonetype` — объект со значением `None` (обозначает отсутствие значения).
- `float` — вещественные числа.
- `complex` — комплексные числа.
- `str` — строки (Unicode).
- `bytes` — неизменяемая последовательность байтов.
- `bytearray` — изменяемая последовательность байтов.
- `list` — списки (аналогичны массивам).
- `tuple` — кортежи.
- `dict` — словари.
- `set` — множества (коллекции уникальных объектов).
- `frozenset` — неизменяемые множества.
- `function` — функции.
- `module` — модули.
- `type` — классы и типы данных.

В языке Python используется динамическая типизация. Это означает, что при присваивании переменной значения интерпретатор автоматически относит переменную к одному из типов данных.

Значение переменной присваивается с помощью оператора **=**, таким образом:

```
x=7                 # тип int
y=7.8               # тип float
s1='Строка '        # тип строки
s1="Строка "        # тип строки
b=True              # тип логический
x,y,z="123"         #строка
x,y,z=[1,2,3]       #список
x,y,z=(1,2,3)       #кортеж
```

Python в любой момент времени может изменить тип переменной в соответствии с данными, хранящимися в ней:

```
f="Строка"          # тип str
f=5                 #тип int
```

Определить, на какой тип данных ссылается переменная, позволяет функция **type()**:

```
type(f)
```

Для удаления переменной используется команда **del**:

```
del f
```

Явное преобразование типов

Для явного преобразования типов данных используются следующие функции:

- `bool()` — преобразует объект в логический тип данных.
- `int()` — преобразует объект в число, вторым необязательным аргументом идет система счисления (по умолчанию 10 — десятичная).
- `float()` — преобразует целое число или строку в вещественное число.
- `str()` — преобразует объект в строку.
- `list()` — преобразует элементы последовательности в список.

Например:

```
x=input("x=")
y=input("y=")
z=x+y
print(z)     #результат – 59 (склеивание символов)
#--------------------
x=int(input("x="))
y=int(input("y="))
z=x+y
print(z)     #результат – 14 (сложение целых чисел)
```

Операторы

Математические операторы:

- Сложение **+**
- Вычитание **-**
- Умножение *****
- Деление **/**
- Деление с округлением вниз **//**
- Остаток от деления **%**
- Возведение в степень ******

Операторы сравнения:

- Больше **>**
- Меньше **<**
- Больше или равно **>=**
- Меньше или равно **<=**
- Не равно **!=**
- Сравнение **==**

Логические операторы:

- И **and**
- ИЛИ **or**
- НЕ **not**

Задания для закрепления изученного материала

1. Написать программу для вычисления площади круга. Значение радиуса вводится во время выполнения программы.
2. Написать программу для вычисления площади прямоугольного треугольника. Значения катетов вводятся во время выполнения программы.
3. Написать программу для вычисления площади поверхности куба. Значение стороны вводится во время выполнения программы.
4. Дано двузначное число, найти сумму его цифр.
   Например: дано: 65. Ответ: 11.
5. Дано трехзначное число, найти число, полученное при перестановке первой и второй цифры.
   Например: дано: 615. Ответ: 165.

Задания для самостоятельной работы

1. Изучить возможности онлайн интерпретатора (компилятора) Python. **Например:** https://www.programiz.com/
2. Изучить возможности работы с интерпретатором Python для мобильных устройств.

Условный оператор (оператор ветвления)

Оператор ветвления позволяет в зависимости от значения логического выражения выполнить отдельный участок программы или, наоборот, не выполнять его.

Условный оператор имеет следующий формат:

```
if логическое_выражение:
```

Блок, выполняемый при истинности условия

```
elif логическое_выражение:
```

Блок, выполняемый при истинности условия

```
else:
```

Блок, выполняемый, если все условия ложны

Блоки внутри составной инструкции выделяются одинаковым количеством пробелов (четыре пробела). Концом блока является

инструкция, перед которой расположено меньшее количество пробелов. На *рисунке 1.* изображена блок-схема работы условного оператора.

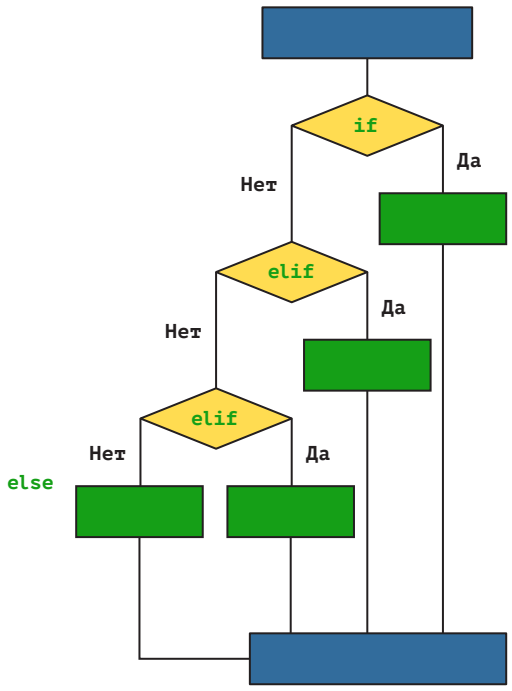


Рисунок 1. Блок-схема работы условного оператора

Рассмотрим несколько примеров применения условного оператора.

Пример 1. Проверка числа на четность

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

Пример 2. Определение возрастной группы

```
age = int(input("Введите ваш возраст: "))
if age < 0:
    print("Ошибка: Возраст не может быть отрицательным.")
```



```
elif age <= 12:
    print("Ребенок")
elif age <= 18:
    print("Подросток")
else:
    print("Взрослый")
```

Пример 3. Расчет площади круга и длины окружности

```
radius = int(input("Введите радиус: "))
if radius < 0:
    print("Ошибка: Радиус не может быть отрицательным.")
else:
    print("Длина окружности равна: ", 2*3.14*radius)
    print("Площадь круга равна: ", 3.14*radius**2)
```

Пример 4. Расчет итоговой оценки

```
score = float(input("Введите количество баллов: "))
if score>= 85:
    print("Отлично. Ваша оценка - 5!")
elif score>= 70:
    print("Хорошо. Ваша оценка - 4!")
elif score>= 55:
    print("Ваша оценка - 3. Стоит повторить материал!")
else:
    print("Вы не сдали экзамен!!!")
```

Задания для закрепления изученного материала

1. Написать программу вычисления стоимости покупки с учетом скидки. Скидка в 7% предоставляется, если сумма покупки больше 500 сомов, в 50 % — если сумма больше 21 000 сомов.
2. Дано трехзначное число, определить, входит ли в него цифра 3.
3. Определить, является ли заданное шестизначное число — счастливым. (Счастливым называют такое шестизначное число, у которого сумма первых трех цифр равна сумме его последних трех цифр).

Задания для самостоятельной работы

1. Написать программу. Дано четырехзначное целое число. Определить: является ли число палиндромом (перевертышем) с учетом четырех цифр (**например:** 2222,6116, 2002 и т. д.);
2. Написать программу. Дано четырехзначное целое число. Определить: верно ли, что это число содержит ровно три одинаковые цифры (**например:** 6676,4544 и т. д.);
3. Написать программу. Дано четырехзначное целое число. **Определить:** верно ли, что все четыре цифры числа различны?

Циклические конструкции. Цикл for

Цикл **for** применяется для перебора элементов последовательности.

Имеет следующий формат:

```
for Текущий_элемент in Последовательность:
Тело_цикла
```

Где:

- Последовательность — объект, поддерживающий механизм итерации. **Например**, строка, список, кортеж, словарь.
- Текущий_элемент — на каждой итерации через этот параметр доступен текущий элемент последовательности или ключ словаря
- Тело_цикла — блок, который будет многократно выполнятся.

Для создания последовательности чисел в определенном диапазоне можно использовать функцию **range()**. Функция **range()** генерирует последовательность чисел. Имеет три аргумента range(старт, стоп, шаг), при этом первый и третий аргумент может отсутствовать.

- start — целое число, начиная с которого должна быть возвращена последовательность целых чисел (по умолчанию равен 0);
- stop — целое число, до которого должна быть возвращена последовательность целых чисел. Диапазон целых чисел заканчивается на stop — 1;

- step (необязательно) — это целочисленное значение, определяющее шаг между каждым целым числом в последовательности (по умолчанию равен 1).

Например следующий программный код, выведет все целые числа от 1 до 100:

```
for x in range(1,101):
    print(x)
```

Пример 1. Вывод числовых последовательностей

```
print("Вывод 5 чисел, начиная с 0.")
for i in range(5):
    print(i)
#----------
print("Вывод чисел от 3 до 6 (не включая 6).")
for i in range(3,6):
    print(i)
#----------
print("Вывод чисел от 4 до 10 (не включая 10), с шагом 2.")
for i in range(4,10,2):
    print(i)
#----------
print("Вывод чисел от 0 до -10 (не включая -10), с шагом -2.")
for i in range(0,-10,-2):
    print(i)
```

Пример 2. Перебор в цикле

```
print ("Перебор списка")
my_list=['один','два','три','четыре','пять']
length=len(my_list)
#функция len() определяет длину списка (размер массива)
for i in range(0,length):
    print(my_list[i])
#--------------
```

```
print("Перебор букв в слове")
for s in "Hello":
    print(s)
#--------------
print("Перебор элементов списка (массива) со сложением")
spisok = [10, 40, 20, 30]
for element in spisok:
    print(element + 2)
```

Пример 3. Сумма четных целых чисел от 10 до 30

```
print("Сумма четных целых чисел от 10 до 30")
s=0
for x in range(10,30,2):
    s+=x;
print("Сумма четных целых чисел от 10 до 30 = ",s)
```

Пример 4. Нахождение факториала числа.

```
print("3. Нахождение факториала числа")
num = int(input("Введи число: "))
factorial = 1
if num < 0:
    print("Для отрицательных чисел факториал не определен")
elif num == 0:
    print("Факториал 0 равен 1")
else:
    for i in range(1,num + 1):
        factorial = factorial*i
    print("Факториал",num,"равен",factorial)
```

Задания для закрепления изученного материала

1. Напишите программу, которая вычисляет сумму первых **n** целых положительных чисел. Количество суммируемых чисел должно вводиться во время работы программы.
2. Напишите программу, которая находит количество и сумму целых положительных чисел, кратных 4 и меньших 100.

3. Напишите программу, которая находит произведение всех целых чисел от 8 до 15.

Задания для самостоятельной работы

1. Написать программу. Вводится любое произвольное количество чисел. Необходимо посчитать сколько среди них четных чисел.
2. Написать программу. Вводится любое произвольное количество чисел. Необходимо посчитать их сумму и количество нечетных чисел.

Циклические конструкции. Цикл while

Выполнение инструкций в цикле while продолжается до тех пор, пока логическое выражение истинно.

Имеет следующий формат:

```
Начальное_значение
while Условие:
    Инструкции
    Приращение
```

Последовательность работы цикла:

- Переменной-счетчику присваивается начальное значение.
- Проверяется условие, если оно истинно, выполняются инструкции внутри цикла, иначе выполнение цикла завершается.
- Переменная-счетчик изменяется на величину, указанную в параметре Приращение. (Если Приращение не указано, то цикл будет бесконечным.
- Переход в начало цикла

Примеры. Вывод чисел и перебор элементов

```
print('Вывод чисел от 1 до 5')
i=1             # Начальное значение
while i<6:      # Условие
    print(i)    # Инструкции
```

```
    i+=1       # Приращение (увеличение на один)
#----------------------
print('Вывод чисел от 5 до 1')
i=5
while i:
    print(i)
    i-=1       # Приращение (уменьшение на один)
#--------------------
print('Перебор элементов списка (массива)')
x=[1,2,3]
i=0
c=len(x)  #длина списка (массива)
while i<c:
    x[i]*=2    #умножение каждого элемента на 2
    i+=1
print(x)
#-------------------
print('Вывод чисел от 1 до 5. Использование ПРЕРЫВАНИЯ')
i=1
while True: # вариант создания бесконечного цикла
    if i>5:
        break #оператор прерывания (останавливает работу цикла)
    print(i)
    i+=1
```

Задания для закрепления изученного материала

1. Необходимо, чтоб программа выводила на экран следующую последовательность: 7 14 21 28 35 42 49 56 63 70 77 84 91 98
2. Организовать беспрерывный ввод чисел с клавиатуры, пока пользователь не введёт 0. После ввода нуля, показать на экран количество чисел, которые были введены, их общую сумму и среднее арифметическое.
3. Необходимо суммировать все нечётные целые числа в диапазоне, который введёт пользователь с клавиатуры.

Задания для самостоятельной работы.

1. Напишите программу. Дано число меньше одного миллиона. Определить количество цифр в нем.
2. Дан массив числами, например: [10, 20, 30, 50, 235, 3000]. Выведите на экран только те числа из массива, которые начинаются на цифру 1, 2 или 5.
3. Дано число **n=1000**. Делите его на 2 столько раз, пока результат деления не станет меньше 50. Какое число получится? Посчитайте количество итераций, необходимых для этого (итерация — это проход цикла), и запишите его в переменную num.

Бесконечные циклы и прерывание

Бесконечный цикл — это цикл, который продолжает выполняться, пока не будет принудительно прерван. Они могут быть полезны в различных сценариях программирования, но также могут привести к нежелательным последствиям, если их использовать без должного контроля.

Пример бесконечного цикла:

```
while True:
    # код, который будет выполняться бесконечно
```

В Python также могут быть использованы бесконечные циклы с использованием других условий, например:

```
while 1 > 0:
    # код, который будет выполняться бесконечно
```

Для прерывания бесконечных циклов используется оператор **break**. Он позволяет выйти из цикла при выполнении определенного условия.

Рассмотрим пример использования прерывания:

```
while True:
    answer = input("Хотите выйти из цикла? (да/нет): ")
    if answer.lower() == "да":
        break
```

```
    else:
        print("Цикл продолжается...")
print("Цикл завершен.")
```

Оператор continue используется для перехода к следующей итерации цикла, минуя оставшуюся часть кода в текущей итерации.

Рассмотрим пример использования оператора continue:

```
for i in range(1, 11):
    if i % 2 == 0:
        continue  # Пропускаем четные числа
    print(i)
```

Этот код выведет на экран числа от 1 до 10, пропуская при этом четные числа.

Рассмотрим пример программы, которая будет выводить на экран все простые числа от 1 до 100. (Простое число — это число, которое делится только на себя и на 1).

```
# Выводим простые числа от 1 до 100
print("Простые числа от 1 до 100:")
for num in range(2, 101):  # Начинаем с 2, так как 1 не является простым числом
    is_prime = True     # Предполагаем, что число простое
    for i in range(2, num):     # Проверяем деление числа на числа от 2 до (num - 1)
if num % i == 0:  # Если число делится на какое-то число кроме 1 и самого себя, то оно не простое
            is_prime = False
            break     # Прерываем внутренний цикл, так как уже известно, что число не простое
if is_prime:     # Если после проверки число осталось простым, выводим его
        print(num)
```

Эта программа использует вложенные циклы: внешний цикл проходит по числам от 2 до 100, а внутренний цикл проверяет деление каждого числа на числа от 2 до числа — 1. Если число делится

без остатка хотя бы на одно число, кроме 1 и самого себя, то оно не является простым, и цикл прерывается с помощью оператора break. Если же число проходит все проверки, оно считается простым и выводится.

Задания для закрепления изученного материала

1. Напишите программу, которая будет считывать числа с клавиатуры и выводить их сумму. Программа должна останавливаться при вводе числа 0.
2. Напишите программу, которая будет выводить на экран все числа от 1 до 100, кроме чисел, делящихся на 5.
3. Напишите программу, которая будет запрашивать у пользователя ввод чисел до тех пор, пока сумма введенных чисел не достигнет 100. При этом программа должна выводить текущую сумму после каждого ввода числа.

Задания для самостоятельной работы

1. Напишите программу, которая будет запрашивать у пользователя ввод чисел до тех пор, пока сумма введенных чисел не станет отрицательной. При этом программа должна выводить текущую сумму после каждого ввода числа.
2. Напишите программу, которая будет выводить на экран таблицу умножения от 1 до 10, используя вложенные циклы. Однако, программа должна пропускать умножение, если результат превышает 50.
3. Напишите программу, которая будет запрашивать у пользователя ввод чисел до тех пор, пока он не введет число, кратное 7. После этого программа должна вывести сообщение «Поздравляем, вы ввели число, кратное 7: {число}» и завершиться.

КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 1

1. Что такое Python и чем он отличается от других языков программирования?
2. Каковы основные преимущества использования Python?
3. Каковы основные особенности синтаксиса Python?
4. Что такое интепретируемый язык программирования?
5. Как создать переменную в Python и какие правила именования переменных следует соблюдать?
6. Каковы основные типы данных в Python?

7. Чем отличаются списки **(list)**, кортежи **(tuple)** и множества **(set)** друг от друга?
8. Как создать и использовать словарь **(dictionary)** в Python?
9. Что такое инструкция **print()** и как использовать её для вывода данных?
10. Какая структура у условного оператора в Python?
11. Какие ключевые слова используются для условного оператора в Python?
12. Как создать условие с несколькими вариантами **(elif)** в Python?
13. Какова разница между операторами **==** и **=**?
14. Как использовать условный оператор для проверки условий в Python?
15. Какая структура у цикла **for** в Python?
16. Как использовать цикл **for** для перебора элементов в списке?
17. Какая функция используется для создания последовательности чисел в Python?
18. Как использовать цикл **for** для работы с диапазонами чисел?
19. Какие встроенные функции могут помочь при работе с циклом **for**?
20. Какая структура у цикла **while** в Python?
21. Как использовать цикл **while** для выполнения действий до тех пор, пока условие истинно?
22. Чем отличается цикл **while** от цикла **for**?
23. Как избежать бесконечного выполнения цикла **while**?
24. Как использовать операторы **break** и **continue** в цикле **while**?

Раздел 2

РАБОТА СО СПИСКАМИ, СЛОВАРЯМИ И КОРТЕЖАМИ

Работа со строками

Строки являются упорядоченными последовательностями символов.

Длина строки ограничена лишь объемом оперативной памяти. Как и все последовательности, строки поддерживают обращение к элементу по индексу, конкатенацию (**оператор +**), повторение (**оператор ***), проверку на вхождение (**оператор in**).

Строки относятся к неизменяемым типам данных. Практически все строковые методы в качестве значения возвращают новую строку.

Можно получить символ по индексу, но изменить его нельзя!

Например:

```
s="Строка"      #это строка
print(s)        #вывод всей строки
print(s[0])     #вывод первого символа
s[0]="Л"        #изменить строку нельзя. Это ОШИБКА!
```

Узнать количество символов в строке можно с помощью функции **len()**:

```
s="Строка"       #это строка
print(s)        #вывод всей строки
print(len(s))   #вывод длины строки
```

Рассмотрим перебор символов строки с помощью цикла **for**:

```
s="Строка"     # это строка
```

```
for i in range(len(s)):  # перебор символов (первый способ)
    print(s[i],end=",")
print("\n") # символ \n означает переход на новую строку
#------------
for i in s:     # перебор символов (второй способ)
    print(i,end=",")
```

Срезы строк (Извлечение среза).

Оператор извлечения среза: **[X:Y]**.

X — начало среза,

Y — кончание;

символ с номером **Y** в срез не входит.

По умолчанию первый индекс равен 0, а второй — длине строки.

Например:

```
s='Программист'
print(s[3:5])      # гр
print(s[2:-2])     # ограмми
print(s[:6])       # Програ
print(s[1:])       # рограммист
print(s[:])        # Программист
```

Строки являются упорядоченными последовательностями символов.

Некоторые из методов строк представлены в *таблице 1.*

Таблица 1

МЕТОДЫ РАБОТЫ СО СТРОКАМИ

S.find(str,[start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или -1
S.rfind(str [start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или -1
S.index(str,[start],[end])	Поиск подстроки в строке. Возвращает номер первого вхождения или вызывает **ValueError**

S.rindex(str,[start],[end])	Поиск подстроки в строке. Возвращает номер последнего вхождения или вызывает **ValueError**
S.replace(шаблон,замена, [maxcount])	Замена шаблона на замену. **maxcount** ограничивает количество замен
S.split(символ)	Разбиение строки по разделителю
S.isdigit()	Состоит ли строка из цифр
S.isalpha()	Состоит ли строка из букв
S.isalnum()	Состоит ли строка из цифр или букв
S.islower()	Состоит ли строка из символов в нижнем регистре
S.isupper()	Состоит ли строка из символов в верхнем регистре
S.count()	Возвращает число вхождений подстроки в строку. Если подстрока в строку не входит, то возвращается значение 0
S.istitle()	Начинаются ли слова в строке с заглавной буквы
S.upper()	Преобразование строки к верхнему регистру
S.lower()	Преобразование строки к нижнему регистру
S.lstrip([chars])	Удаление пробельных или указанных символов в начале строки
S.rstrip([chars])	Удаление пробельных или указанных символов в конце строки
S.strip([chars])	Удаление пробельных или указанных символов в начале и в конце строки

Рассмотрим примеры работы со строками, результат команды дан в комментариях к программному коду.

```
s=' Я лучший Программист C++'
s1=s.find('и')     # s1=7
s1=s.rfind('и')    # s1=18
s1=s.rfind('Б')    # s1=-1
```

```
s1=s.index('и')     # s1=7
s1=s.index('Б')     # ValueError: substring not found
s1=s.find('C++')    # s1=22
s1=s.replace('и','е')     # s1='Я лучшей Программест C++'
s1=s.replace('и','е',1)     # s1='Я лучшей Программист C++'
s1=s.replace('C++','ПИТОН')     # s1='Я лучший Программист ПИТОН'
s1=s.split(' ')     # ['', 'Я', 'лучший', 'Программист', 'C++']
#-------------------------
s=' Я лучший Программист C++'
s1=s.isdigit()     #False
s='5663'
s1=s.isdigit() #True
s='Василий Алибабаевич'
s1=s.isalpha() #False
s='Я лучший Программист питон'
s1=s.count('п') # 1
s='Я лучший Программист питон'
s1=s.count('м') # 2
s=' Я лучший Программист питон '
s1=s.strip() # 'Я лучший Программист питон'
s=' Я лучший Программист питон '
s1=s.strip() # 'Я лучший Программист питон'
```

Задания для закрепления изученного материала

1. Вставить в строку **S** три символа # начиная с 5-ой позиции и результат записать в **S1**.
2. Найти индекс позиции буквы **E** в строке **S** и результат записать в **S1**.
3. Удалить из строки **S 5** символов начиная с 8-й позиции и результат записать в **S1**.
4. Заменить в строк **S** все буквы **T** на знак **?** и результат записать в **S1**.
5. Выделить подстроку из строки **S** с 8-й позиции 2 символа и результат записать в **S1**.

Задания для самостоятельной работы

1. Найти индекс позиции последнего вхождения буквы **E** в строке **S** и результат записать в **S1**.
2. Вставить свое имя в строку **S** начиная с 3-ей позиции и результат записать в **S1**.
3. Заменить в строке **S** все пробелы на знак нижнего подчеркивания _ и результат записать в **S1**.
4. Посчитать сколько букв **A** в строке **S** и результат записать в **S1**.
5. Перевернуть текст из строки **S** и результат записать в **S1**.

Списки и кортежи

Списки и кортежи – это нумерованные наборы объектов. Каждый элемент набора содержит ссылку на объект. По этой причине списки и кортежи могут содержать объекты произвольного типа данных и иметь не ограниченную степень вложенности.

Позиция элемента в наборе задается индексом. Нумерация элементов начинается с нуля.

Списки и кортежи являются упорядоченными последовательностями элементов. Как и все последовательности, они поддерживают обращение к элементу по индексу, получение среза, конкатенацию **(+)**, повторение **(*)**, проверку на вхождение **(in)**.

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

```
x=[1,2,3]     #Создаем список
print(x[0])    #Получаем и выводим первый элемент списка 1
x[0]=50     #Изменяем элемент по индексу
print(x)     #Выводим список [50, 2, 3]
```

Кортежи относятся к неизменяемым типам данных. Иными словами, можно получить элемент по индексу, но изменить нельзя.

```
x=(1,2,3)     #Создаем кортеж
print(x[0])    #Получаем и выводим первый элемент кортежа 1
x[0]=50     #Изменить элемент по индексу НЕЛЬЗЯ!
```

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

Функция **list()** — преобразует любую последовательность в список. Если параметр не указан, то создается пустой список.

```
print(list())    #Пустой список  []
print(list(«Строка»))    #Преобразование строки в список
                 # ['С', 'т', 'р', 'о', 'к', 'а']
print(list((1,2,3,4,5))) #Преобразование кортежа в список
                 # [1, 2, 3, 4, 5]
```

Любой элемент списка может содержать объект произвольного типа. Например, элемент списка может быть числом, строкой, списком, кортежем, словарем… Создать вложенный список можно следующими способами:

```
x=[[1,2,3],[4,5,6],[7,8,9]]    #Многомерный список
```

Выражение внутри скобок может располагаться на нескольких строках:

```
x=[
    [1,2,3],
    [4,5,6],
    [7,8,9]
    ]
```

Что бы получить значение элемента во вложенном списке, следует указать два индекса:

```
x=[[1,2,3],[4,5,6],[7,8,9]]      #Многомерный список
print(x[1][1])    #Вывод элемента списка  5
```

Задания для закрепления изученного материала

Дан список любой размерности:

1. Дан список любой размерности. Найти сумму элементов списка.
2. Дан список любой размерности. Найти произведение элементов списка.
3. Дан список любой размерности. Найти среднеарифметическое значение элементов списка.

4. Дан список любой размерности. Найти количество нулевых элементов списка.
5. Дан список любой размерности. Найти количество отрицательных элементов списка.

Задания для самостоятельной работы

1. Дан двумерный список любой размерности:
2. Дан двумерный список любой размерности. Заменить все элементы кратные десяти на цифру 0.
3. Дан двумерный список любой размерности. Вывести все элементы, являющиеся двузначными числами
4. Дан двумерный список любой размерности. Найти сумму тех элементов, оба индекса которых четные числа и их количество.
5. Дан двумерный список любой размерности. Найти количество нулевых элементов, расположенных в нечетных столбцах
6. Дан двумерный список любой размерности. Заменить все отрицательные четные числа на положительные.

Методы работы со списками

Для добавления и удаления элементов списка используются следующие методы:

- `append()` — добавляет один объект в конце списка. Метод изменяет текущий список и ничего не возвращает.
- `extend()` — добавляет элементы последовательности в конец списка. Метод изменяет текущий список и ничего не возвращает.
- `insert()` — добавляет один объект в указанную позицию. Остальные элементы смещаются. Метод изменяет текущий список и ничего не возвращает.
- `pop()` — удаляет элемент, расположенный по указанному индексу, и возвращает его. Если индекс не указан, то удаляется и возвращается последний элемент списка. Если элемента с указанным индексом нет или список пустой, возбуждается исключение IndexError.
- `remove()` — удаляет первый элемент, содержащий указанное значение. Если элемент не найден, возбуждается исключение

ValueError. Метод изменяет текущий список и ничего не возвращает.

Рассмотрим примеры работы с методами списка, результат команды дан в комментариях к программному коду.

```
x=[1,2,3]
x.append(4)     # x=[1, 2, 3, 4]
x.extend([5,6,7])     # x=[1, 2, 3, 4, 5, 6, 7]
x.insert(2,53)     # x=[1, 2, 53, 3, 4, 5, 6, 7]
y=x.pop()     # y=7  x=[1, 2, 53, 3, 4, 5, 6]
y=x.pop(2)     # y=53  x=[1, 2, 3, 4, 5, 6]
x.remove(7)     # ValueError: list.remove(x): x not in list
  x.remove(3)     # x=[1, 2, 4, 5, 6]
```

Для поиска элементов в списке можно использовать следующие методы:

- index() — 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 списка.
- count() — возвращает общее количество элементов с указанным значением. Если элемент не входит в список, возвращается 0.
- max() — возвращает максимальное значение списка.
- min() — возвращает минимальное значение списка.

Рассмотрим примеры работы с методами поиска элементов в списке, результат команды дан в комментариях к программному коду.

```
x=[1,2,3,2,5,6,4,7,5]
y=x.index(2)     # y=1
y=x.index(2,3,8)     # y=3
y=x.count(2)     # y=2
```

Для перемешивания элементов в списке используются следующие методы:

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

- shuffle() — перемешивает список случайным образом. Использует модуль random.

Рассмотрим примеры работы с методами перемешивания элементов в списке, результат команды дан в комментариях к программному коду.

```
import random  #подключение модуля random  (генератор случайных чисел)
          x=[1,2,3,2,5,6,4,7,5]
          x.reverse()     # x=[5, 7, 4, 6, 5, 2, 3, 2, 1]
          random.shuffle(x)     # x=[1, 5, 2, 5, 6, 3, 2, 7, 4]
```

Для выбора элемента случайным образом из списка, можно воспользоваться следующими методами:

- choise() — возвращает случайный элемент из любой последовательности.
- sample() — возвращает список из указанного количества элементов. В этот список попадут элементы из последовательности, выбранные случайным образом.

Примеры работы с методами случайного выбора элемента, результат команды дан в комментариях к программному коду:

```
import random    #подключение модуля random (генератор случайных чисел)
    x=[1,2,3,2,5,6,4,7,5]
    y=random.choice(x)     # y=4
    y=random.sample(x,3)      # y=[3, 2, 6]
```

Для сортировки списка используется метод sort().

- sort() — сортирует список . Изменяет текущий список. Аргумент reverse=True сортирует список по убыванию.

Примеры:

```
x=[1,9,552,3,2,5,6,4,7,5]
x.sort()     #x=[1, 2, 3, 4, 5, 5, 6, 7, 9, 552]
x.sort(reverse=True)     #x=[552, 9, 7, 6, 5, 5, 4, 3, 2, 1]
```

Рассмотрим примеры заполнения списка:

```
import random
x=list(range(11))     # x=[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
x=list(range(1,16))     # x=[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
x=list(range(0,10,2))     # x=[0, 2, 4, 6, 8]
x=list(range(15,0,-1))     # x=[15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
x=random.sample(range(10),10)     # x=[6, 0, 2, 8, 5, 9, 3, 4, 1, 7]
x=random.sample(range(100),5)     # x=[23, 41, 92, 33, 5]
```

Задания для закрепления изученного материала

1. Дан список любой размерности. Выведите все элементы списка с четными индексами.
2. Дан список любой размерности. Необходимо удалить все вхождения числа 20 из него.
3. Дан список любой размерности. Найдите количество положительных элементов в данном списке.

Задания для самостоятельной работы

1. Дан список любой размерности. Определите, сколько в этом списке элементов, которые больше двух своих соседей, и выведите количество таких элементов.
2. Дан список любой размерности. Определите, какое число в этом списке встречается чаще всего.
3. Дан список любой размерности. Требуется «сжать» его, переместив все ненулевые элементы в левую часть списка, не меняя их порядок, а все нули — в правую часть.

Кортежи и словари

Кортежи (tuple) в Python – это те же списки за одним исключением.

Кортежи неизменяемые структуры данных

Можно сказать, что кортеж – это список доступный «только для чтения». Так же как списки они могут состоять из элементов

разных типов, перечисленных через запятую. Кортежи заключаются в круглые, а не квадратные скобки.

В кортежах удобно хранить свойства объектов, например, имя, возраст, дату рождения. Если вдруг кортеж состоит из одного элемента, то после единственного элемента кортежа необходимо поставить запятую.

Создать кортеж можно одним из следующих способов:

```
x=tuple() #Создание пустого кортежа
    # ()
x=tuple(«string») #Преобразование строки в кортеж
    # ('s', 't', 'r', 'i', 'n', 'g')
x=tuple([1,2,3,4,5]) #Преобразование списка в кортеж
    # (1, 2, 3, 4, 5)
x=(2,4,1,2,5) # Создание кортежа
    # (2, 4, 1, 2, 5)
x=(1,»str», (2.2,5))# Создание кортежа из разнотипных данных
    # (1, ‘str’, (2.2, 5))
```

В Python есть много встроенных структур данных, используемых для хранения разных типов информации.

Словарь (dict) — одна из таких структур, которая хранит данные в формате пар ключ-значение. Получить доступ к значениям словаря Python можно с помощью ключей.

Для создания словаря в Python необходимо передать последовательность элементов внутри фигурных скобок **{}**, разделив их запятыми **(,)**. Каждый элемент имеет ключ и значение, выраженное парой «ключ: значение».

Значения могут быть представлять собой любые типы данных и повторяться, но ключи обязаны быть уникальными.

Следующие примеры показывают, как создавать словари Python:

```
x={} #Создание пустого словаря
x={1: 'яблоко', 2: 'груша'} #Словарь, где ключи являются целыми числами
x= {'фрукт': 'яблоко', 1: [4, 6, 8]} #Создание словаря с  ключами разных типов
```

```
x={'ИВТ-1':'Илья, Соня, Акмарал','ИВТ-2':'Бакыт, Катя, Канукей'}
```

Чтобы получить доступ к элементам словаря, нужно передать ключ в квадратных скобках [].

```
x={'ИВТ-1':'Илья, Соня, Акмарал','ИВТ-2':'Бакыт, Катя, Канукей'}
print(x['ИВТ-2'])    # Бакыт, Катя, Канукей
```

Добавление элемента:

```
x['ИВТ-3']= 'Самат, Наташа, Максим'
print(x)
        # {'ИВТ-1': 'Илья, Соня, Акмарал',
        # 'ИВТ-2': 'Бакыт, Катя, Канукей',
        # 'ИВТ-3': 'Самат, Наташа, Максим'}
```

Используя цикл for, можно вывести на экран только ключи словаря:

```
for i in x:
    print(i)
        #ИВТ-1
        #ИВТ-2
        #ИВТ-3
```

Задания для закрепления изученного материала

1. Даны ящики, которые вмещают 5 кг, 10 кг и 15 кг яблок. Необходимо выяснить, сколько ящиков разного размера понадобится для того, чтобы распределить 100 кг яблок. (должны быть использованы, все три вида ящиков)
2. В кортеже хранятся: имя и оценки студентов вашей группы. Напишите программу, которая выводит имена студентов, у которых оценка равна введенной с клавиатуры оценке.
3. Создайте словарь «Рыбы», а его элементы разделите на 3 вида: «речные», «озерные» и «морские рыбы». Выведите на экран сначала только ключи, а потом элементы словаря.

Задания для самостоятельной работы.

1. Дан кортеж **(3, 's', 1, 5, 's')**. Выведите на экран: количество всех элементов кортежа, количество строк 's', индекс первого вхождения **'s'** в кортеж.
2. Дан словарь **d = {'1': 1.29, '2': 0.43}**. Используя доступ к элементам словаря по ключу, найдите произведение **1.29*0.43**, после чего добавьте результат в словарь, а затем выведите значение нового элемента на экран.
3. Создайте словарь, связав его с переменной school, и наполните данными, которые бы отражали количество учащихся в разных классах **(1а, 1б, 2б, 6а, 7в и т. п.)**. Внесите изменения в словарь согласно следующему: а) в одном из классов изменилось количество учащихся, б) в школе появился новый класс, с) в школе был расформирован (удален) другой класс. Вычислите общее количество учащихся в школе.

КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 2

1. Каковы основные различия между списками, словарями и кортежами в Python?
2. Как создать пустой список, словарь и кортеж в Python?
3. Как добавить элемент в конец списка и как удалить элемент из списка по индексу?
4. Как получить значение из словаря по ключу? Как добавить новую пару ключ-значение в словарь?
5. В чем отличие между изменяемыми и неизменяемыми типами данных в Python? К каким из них относятся списки, словари и кортежи?
6. Как объединить несколько строк в одну в Python?
7. Как разделить строку на подстроки по определенному разделителю?
8. Как получить длину строки в Python?
9. Как проверить, содержит ли строка определенный подстрок?
10. Как заменить подстроку в строке другой подстрокой?
11. Как получить доступ к элементу списка и кортежа по индексу?
12. Что такое срезы в Python и как их использовать для извлечения подпоследовательностей из списка или кортежа?
13. Как добавить новый элемент в середину списка?
14. Как объединить два списка в один в Python?
15. Как создать кортеж из списка и наоборот?
16. Какие методы доступны для работы со списками в Python?
17. Как сортировать список? Как отсортировать его в обратном порядке?
18. Как удалить элемент из списка по значению, а не по индексу?
19. Как получить индекс элемента в списке по его значению?
20. Как создать копию списка?

21. Как создать пустой кортеж и словарь в Python?
22. Как получить список ключей и значений из словаря?
23. Как удалить элемент из словаря по ключу?
24. Как объединить два словаря в Python?

Раздел 3

ОБРАБОТКА ИСКЛЮЧЕНИЙ И РАБОТА С ФАЙЛАМИ

Обработка исключений с помощью конструкции try-except

Исключения — это события, которые возникают во время выполнения программы и нарушают нормальный поток исполнения. В Python обработка исключений — это мощный механизм, который позволяет программистам обрабатывать ошибки, управлять ошибочными ситуациями и предотвращать прерывание выполнения программы.

Рассмотрим некоторые из наиболее распространенных исключений в Python:

- `SyntaxError` — возникает при нарушении синтаксиса языка Python.
- `IndentationError` — возникает, когда отступы в коде не соответствуют стандартам Python.
- `NameError` — возникает, когда пытаетесь использовать переменную или функцию, которая не была определена.
- `TypeError` — возникает, когда операция применяется к объекту несовместимого типа.
- `ZeroDivisionError` — возникает при попытке деления на ноль.
- `ValueError` — возникает, когда функция получает аргумент правильного типа, но с недопустимым значением.
- `IndexError` — возникает, когда индекс последовательности находится за её пределами.

- KeyError — возникает при попытке доступа к ключу словаря, которого не существует.
- FileNotFoundError — возникает, когда пытаетесь открыть файл, который не существует.
- IOError — возникает при ошибках ввода-вывода.
- AttributeError — возникает, когда объект не имеет запрашиваемого атрибута.
- ImportError — возникает, когда не удается импортировать модуль или библиотеку.
- KeyboardInterrupt — возникает при нажатии комбинации клавиш прерывания выполнения программы (обычно Ctrl+C).
- OverflowError — возникает, когда результат арифметической операции слишком велик для хранения.
- MemoryError — возникает, когда недостаточно памяти для выполнения операции.

Основной механизм обработки исключений в Python — это конструкция try-except. Она позволяет программистам написать блок кода, в котором могут возникнуть исключения, и предоставить обработчики для этих исключений.

Конструкция try-except в Python используется для обработки исключений. Она позволяет попытаться выполнить определенный блок кода и перехватывать исключения, которые могут возникнуть внутри этого блока. Синтаксис конструкции try-except выглядит следующим образом:

```
try:
    # Блок кода, в котором может произойти исключение
    # Код, который нужно проверить на исключение
except ExceptionType:
    # Блок кода, который выполнится в случае возникновения исключения типа ExceptionType
    # Обработка исключения
```

Пример 1. Рассмотрим пример, в котором мы попытаемся поделить число на ноль

```
try:
    result = 10 / 0
except ZeroDivisionError:
    print("Ошибка: деление на ноль!")
```


В блоке except можно указать несколько типов исключений через кортеж. Это позволяет обрабатывать разные типы исключений по-разному.

Пример 2: Обработка разных исключений

```
try:
    value = int(input("Введите число: "))
    result = 10 / value
except ValueError:
    print("Ошибка: введено некорректное значение!")
except ZeroDivisionError:
    print("Ошибка: деление на ноль!")
```

Кроме того, в конструкции try-except можно использовать блок finally, который будет выполняться в любом случае, независимо от того, произошло исключение или нет.

Пример 3: Использование блока finally

```
try:
    file = open("файл.txt", "r")
     # работаем с файлом
except FileNotFoundError:
    print("Ошибка: файл не найден!")
finally:
    file.close()    # закрываем файл независимо от результата
```

Таким образом, Конструкция try-except является мощным инструментом для обработки исключений в Python. Она позволяет писать более надежные программы, предотвращая их аварийное завершение при возникновении ошибок.

Рассмотрим пример программы нахождения факториала введенного числа, сделаем проверку числа на отрицательность, так как факториала отрицательного числа не существует:

```
try:
    num = int(input("Введите число: "))
     if num < 0:
        raise ValueError("Факториал отрицательного числа не
определен")
    factorial = 1
    for i in range(1, num + 1):
        factorial *= i
    print(f"Факториал числа {num} равен {factorial}")
except ValueError as ve:
    print(ve)
```

В этой программе мы сначала запрашиваем у пользователя ввод числа. Затем мы проверяем, является ли число отрицательным. Если число отрицательное, генерируется исключение ValueError. В противном случае мы вычисляем факториал числа, используя цикл for, и выводим результат.

Задания для закрепления изученного материала

1. Пользователь вводит два числа. Напишите программу, которая делит первое число на второе. Обработайте исключение, которое возникает при делении на ноль.
2. Пользователь вводит имя файла. Напишите программу, для открытия этого файла для чтения. Обработайте исключение, которое возникает, если файл не найден.
3. Пользователь вводит список чисел через запятую. Напишите программу, которая делит каждое число на 10 и выведите результат. Обработайте исключение, которое возникает при делении на ноль.

Задания для самостоятельной работы.

1. Создайте список. Попросите пользователя ввести индекс элемента списка. Попробуйте вывести элемент списка с этим индексом. Обработайте исключение, которое возникает при вводе некорректного индекса.
2. Попросите пользователя ввести число. Попробуйте вычислить факториал этого числа (произведение всех целых чисел от 1 до введенного числа). Обработайте исключение, которое возникает, если введенное число отрицательное.

3. Создайте словарь с несколькими парами ключ-значение. Попросите пользователя ввести ключ. Попробуйте получить значение из словаря по этому ключу и вывести его. Обработайте исключение, которое возникает, если введенный ключ отсутствует в словаре.

Работа с файлами (открытие, чтение, запись)

Текстовые файлы представляют собой файлы, содержащие текст в виде последовательности символов, обычно в кодировке ASCII или Unicode. В них каждый символ представлен в текстовой форме, что делает их удобными для чтения человеком. Примерами текстовых файлов могут быть файлы .txt, .csv, .html и многие другие.

В отличие от текстовых файлов, бинарные файлы содержат данные в бинарном формате, что означает, что они могут содержать произвольные данные, включая изображения, аудио, видео и т. д. Бинарные файлы не имеют прямой текстовой интерпретации и могут быть сложными для чтения человеком. Однако они обычно более компактны и могут содержать более сложные данные.

Перед тем как работать с файлом, его необходимо открыть. Для этого в Python используется функция open(). Синтаксис этой функции:

```
open(file, mode='r', buffering=-1, encoding=None, errors=None,
newline=None, closefd=True, opener=None)
```

Рассмотрим назначение каждого аргумента функции **open()**:

- `file` — имя файла или путь к нему.
- `mode` — режим доступа к файлу. По умолчанию файл открывается в режиме чтения (`'r'`). Другие возможные режимы:
  - `'w'` — открыть файл для записи. Если файл уже существует, его содержимое будет заменено. Если файла не существует, он будет создан,
  - `'r'` — открыть файл для чтения (по умолчанию),
  - `'a'` — открыть файл для добавления. Новая информация будет добавлена в конец файла,

- 'b' — открыть файл в режиме двоичного чтения или записи:
  - 'rb' — бинарное чтение,
  - 'wb' — бинарная запись.

- ❑ buffering — управляет буферизацией. По умолчанию -1, что означает использование системной настройки.
- ❑ encoding — кодировка файла.
- ❑ errors — способ обработки ошибок при декодировании файла.
- ❑ newline — как обрабатывать перенос строки. По умолчанию None, что означает использование системной настройки.
- ❑ closefd — если True, то файловый дескриптор будет закрыт при закрытии файла. По умолчанию True.
- ❑ opener — пользовательская функция открытия файла.

Пример открытия файла для чтения

```
file = open("example.txt", "r")
```

После открытия файла можно считывать его содержимое. Для этого используются методы объекта файла.

- ❑ read(size) — считывает size символов из файла, если аргумент не указан, то считывается весь файл.
- ❑ readline() — считывает одну строку из файла.
- ❑ readlines() — считывает все строки из файла и возвращает список строк.

Рассмотрим пример чтения из файла:

```
file = open("example.txt", "r")
content = file.read()
print(content)
file.close()
```

В этом примере мы открываем файл **"example.txt"** для чтения **("r")**, считываем его содержимое с помощью метода **read()** и затем выводим содержимое файла на экран. Не забывайте закрывать файл после завершения работы с ним с помощью метода **close()**.

Метод **close()** закрывает файл после завершения операций чтения.

Когда вы открываете файл, операционная система выделяет ресурсы для его обработки. После завершения работы с файлом необходимо освободить эти ресурсы, чтобы другие программы или процессы могли ими воспользоваться. Если файл не закрыт, это может привести к утечке ресурсов и нежелательным эффектам на производительность.

При чтении или записи файлов операционная система кэширует данные в памяти для оптимизации производительности. Закрытие файла гарантирует, что все изменения будут фактически записаны на диск. Если файл не закрыт, изменения могут остаться в кэше и не сохраниться.

Закрытие файла после его использования помогает избежать неожиданных ошибок при последующих операциях с файлом или другими файлами. Например, если файл остается открытым, и вы пытаетесь изменить его имя или переместить, это может привести к ошибке доступа к файлу.

Для записи в файл также используется открытие файла, но уже в режиме записи **('w')** или дозаписи **('a')**.

- ❑ write(string) — записывает строку string в файл.

Рассмотрим пример записи в файл:

```
file = open("example.txt", "w")
file.write("Это текст, который мы записываем в файл.")
file.close()
```

В этом примере мы открываем файл **"example.txt"** для записи **("w")**, записываем строку в файл с помощью метода **write()** и закрываем файл.

Еще один пример, добавления информации в файл:

```
file = open("example.txt", "a")
file.write("\nЭто еще одна строка, добавленная в конец файла.")
file.close()
```

В этом примере мы открываем файл **"example.txt"** для добавления **("a")**, записываем новую строку.

Для более удобной работы с файлами рекомендуется использовать контекстный менеджер with. Он автоматически закрывает файл после завершения блока кода.

Пример использования контекстного менеджера:

```
with open("example.txt", "r") as file:
    content = file.read()
    print(content)
```

Рассмотрим пример, создадим новый файл **output.txt** и запишем в него строку **"Hello, world!"**:

```
# Открываем файл output.txt в режиме записи ('w')
file = open("output.txt", "w")
# Записываем строку "Hello, world!" в файл
file.write("Hello, world!")
# Закрываем файл
file.close()
```

Этот код открывает файл **output.txt** в режиме записи **('w')**, что означает, что, если файл уже существует, он будет перезаписан. Затем он записывает строку **"Hello, world!"** в файл и закрывает его.

Задания для закрепления изученного материала

1. Откройте файл **example.txt** и выведите его содержимое на экран.
2. Найдите количество строк в файле **text.txt** и выведите результат.
3. Напишите программу, которая открывает файл **words.txt** и проверяет, содержится ли в нем слово **«apple»**.

Задания для самостоятельной работы

1. Прочитайте файл **numbers.txt**, содержащий числа, и найдите их сумму, среднее значение и максимальное/минимальное число.
2. Откройте файл **input.txt**, замените в нем все вхождения слова **«old»** на «new» и сохраните изменения в новый файл **output.txt**.
3. Напишите программу для шифрования текстового файла. Прочитайте содержимое файла, зашифруйте его (например, сдвиньте каждую букву на определенное количество позиций в алфавите) и сохраните результат в новом файле.

Обработка исключений при работе с файлами

Открытие и закрытие файлов – это стандартная операция во многих приложениях. Однако, важно помнить, что эти операции могут вызывать ошибки, особенно если файл не существует или нет прав на его открытие. Для обработки таких ситуаций в Python используется механизм исключений.

Для безопасного открытия файла можно использовать конструкцию **try-except**. В блоке try открывается файл, а в блоке except обрабатываются возможные исключения, такие как **FileNotFoundError**.

```
try:
    file = open("example.txt", "r")
    # выполнение операций с файлом
except FileNotFoundError:
    print("Файл не найден или не существует")
```

Для автоматического закрытия файла после работы с ним, рекомендуется использовать контекстный менеджер **with**. В этом случае не нужно явно вызывать метод **close()**.

```
try:
    with open("example.txt", "r") as file:
        # выполнение операций с файлом
except FileNotFoundError:
    print("Файл не найден или не существует")
```

При чтении и записи данных также могут возникать различные ошибки, например, из-за неправильного формата файла или недостаточных прав на запись в файл.

В Python исключение **IOError** может возникнуть при ошибках ввода-вывода, таких как недостаток места на диске или неверный формат файла. Для обработки таких ошибок используется конструкция **try-except**.

```
try:
    with open("example.txt", "r") as file:
        data = file.read()
```

```
except IOError:
    print("Ошибка ввода-вывода")
```

При записи данных в файл также необходимо учитывать возможные ошибки, например, если файл доступен только для чтения или если диск заполнен.

```
try:
    with open("example.txt", "w") as file:
        file.write("Пример данных для записи")
except IOError:
    print("Ошибка записи в файл")
```

Обработка исключений при работе с файлами является важной частью разработки приложений на Python. Правильное использование конструкций **try-except** и контекстных менеджеров with помогает обеспечить безопасность и надежность операций с файлами, предотвращая возможные ошибки и сбои программы.

Рассмотрим пример программы, которая запрашивает у пользователя имя файла для чтения и выводит его первые 15 строк. Обрабатывает исключение, если файл содержит меньше 15 строк:

```
filename = input("Введите имя файла для чтения: ")
try:
        with open(filename, 'r') as file:
            lines = file.readlines()
            for i in range(10):
            if i < len(lines):
            print(lines[i], end='')
            else:
            break
except FileNotFoundError:
    print("Указанный файл не найден.")
except Exception as e:
    print("Произошла ошибка при чтении файла:", e)
```

Этот программный код запрашивает у пользователя имя файла для чтения, затем открывает файл и считывает его содержимое.

Затем он выводит первые 15 строк файла или все строки, если их меньше 15. Если файл не найден, программа выведет сообщение об ошибке «Указанный файл не найден.», а если произойдет любая другая ошибка при чтении файла, программа выведет сообщение об этой ошибке.

Задания для закрепления изученного материала

1. Напишите программу, которая открывает файл для чтения и выводит его содержимое на экран. Обработайте исключение, если файл не существует.
2. Напишите программу для записи текстовой строки в файл. Обработайте исключение, если файл недоступен для записи.
3. Создайте программу, которая запрашивает у пользователя имя файла для чтения и выводит его первые 10 строк. Обработайте исключение, если файл содержит меньше 10 строк.

Задания для самостоятельной работы

1. Напишите скрипт, который запрашивает у пользователя путь к файлу для чтения и выводит количество слов в этом файле. Обработайте исключение, если файл не найден.
2. Напишите функцию для создания нового файла с указанным именем. Обработайте исключение, если файл с таким именем уже существует.
3. Создайте программу для переименования файла. Обработайте исключение, если файл не существует или если не удалось изменить его имя.

Работа с бинарными файлами

Работа с бинарными файлами в Python позволяет осуществлять чтение и запись данных в двоичном формате. Это часто необходимо, когда мы имеем дело с файлами, содержащими не текстовую, а бинарную информацию, такую как изображения, аудио, видео и другие бинарные данные.

Перед тем как работать с бинарным файлом, необходимо его открыть. Для этого используется функция open(), указывая вторым аргументом режим открытия файла. Для работы с бинарными файлами используются режимы **«rb»** для чтения и **«wb»** для записи.

Пример открытия бинарного файла для чтения:

```
file = open("binary_file.bin", "rb")
```

Пример открытия бинарного файла для записи:

```
file = open("binary_file.bin", "wb")
```

После того как файл открыт для чтения, мы можем считывать данные из него. Для этого используется метод **read()**, который читает указанное количество байтов из файла.

Пример чтения из бинарного файла:

```
with open("binary_file.bin", "rb") as file:
    data = file.read(10)  # Читаем 10 байт
    print(data)
```

Для записи данных в бинарный файл используется метод **write()**. Этот метод принимает строку байтов и записывает её в файл.

Пример записи в бинарный файл:

```
with open("binary_file.bin", "wb") as file:
    data = b"Hello, World!"     # Строка байтов для записи
    file.write(data)
```

Рассмотрим пример, создадим бинарный файл и запишем в него целые числа от 1 до 10:

```
# Открываем файл для записи в бинарном режиме ('wb')
with open('binary_numbers.bin', 'wb') as f:
    # Цикл для записи чисел от 1 до 10
    for i in range(1, 11):
        # Преобразуем целое число в бинарное представление
и записываем его в файл
        f.write(i.to_bytes(4, byteorder='little', signed=True))
```

В данной программе открывается файл с именем **'binary_numbers.bin'** для записи в бинарном режиме, при этом используется оператор **with**, который гарантирует автоматическое закрытие файла после выполнения операций.

Далее запускается цикл for, который проходит от 1 до 10 включительно. Внутри цикла каждое целое число преобразуется

в бинарное представление с помощью метода **to_bytes()**. В данном случае используется 4 байта для представления каждого числа, задаются параметры **length=4, byteorder='little'** (порядок байтов) и **signed=True** (знаковое число). Каждое бинарное представление записывается в файл с помощью метода **write()**.

Таким образом, этот программный код создает бинарный файл **'binary_numbers.bin'** и записывает в него целые числа от 1 до 10 в бинарном формате. Этот файл можно использовать для выполнения других операций, например, чтения данных из него или добавления новых данных.

Задания для закрепления изученного материала

1. Создайте программу, которая читает бинарный файл с целыми числами и выводит их на экран.
2. Создайте программу, которая считывает бинарный файл, содержащий строки текста в кодировке UTF-8, и выводит их содержимое на экран.
3. Напишите программу для копирования содержимого одного бинарного файла в другой.
4. Реализуйте программу, которая ищет определенное значение в бинарном файле и выводит его позицию (если оно есть).

Задания для самостоятельной работы

1. Напишите программу для удаления определенного значения из бинарного файла.
2. Напишите программу для перезаписи определенного байта в бинарном файле.
3. Напишите программу, которая считывает бинарный файл, содержащий массив чисел с плавающей запятой (float), и вычисляет их среднее значение.

КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 3

1. Что такое конструкция try-except и для чего она используется в обработке исключений?
2. Какая роль у блока except в конструкции try-except?
3. Какие типы исключений можно обрабатывать с помощью конструкции try-except?
4. Какие действия рекомендуется выполнять в блоке except для более информативной обработки исключений?

5. Какие практические примеры можно привести для иллюстрации работы с конструкцией try-except?
6. Какие шаги нужно выполнить для открытия файла в Python?
7. Какие режимы открытия файла существуют, и что они означают?
8. Каким образом осуществляется чтение данных из файла в Python?
9. Как производится запись данных в файл в Python?
10. Какие методы необходимо использовать для корректного закрытия файла после его использования?
11. Какие исключения могут возникнуть при работе с файлами?
12. Какие стратегии обработки исключений при работе с файлами рекомендуется применять?
13. Какие методы и функции Python могут помочь в обработке исключений при работе с файлами?
14. Каким образом можно реализовать проверку наличия файла перед его открытием?
15. Какие меры безопасности рекомендуется предпринимать при работе с файлами для предотвращения потенциальных ошибок и утечек данных?

Раздел 4

ФУНКЦИИ И КЛАССЫ

Математические функции. Модуль math

Модуль math представляет дополнительный функции для работы с числами, а также стандартные константы. Прежде чем использовать модуль, необходимо подключить его с помощью инструкции:

```
import math
```

Стандартные константы:

- `pi` – возвращает число ПИ

```
import math
print(math.pi) # 3.141592653589793
```

- `e` – возвращает значение константы e

```
import math
print(math.e) # 2.718281828459045
```

Модуль `math` включает в себя набор тригонометрических функций, рассмотри некоторые из них.

Стандартные тригонометрические функции (значение указывается в радианах):

- `sin()`
- `cos()`
- `tan()`

Обратные тригонометрические функции (значение возвращается в радианах)

- ❑ asin()
- ❑ acos()
- ❑ atan()

```
import math
radian=1.5708  #90 градусов
print(math.sin(radian)) # 0.9999
```

Также модуль math включает в себя набор стандартных математических функций для работы с числами:

- ❑ round — возвращает число, округленное до ближайшего меньшего целого для чисел с дробной частью меньше 0,5, или значение, округленное для ближайшего большего целого для чисел с дробной частью больше 0,5. Если дробная часть равна 0,5, то округление производится до ближайшего четного числа.
- ❑ abs — возвращает абсолютное значение.
- ❑ pow — возводит число в степень.
- ❑ max — максимальное значение списка (список чисел через запятую)
- ❑ min — минимальное значение списка (список чисел через запятую)
- ❑ sum — возвращает сумму значений элементов последовательности (список, кортеж)

Примеры работы с математическими функциями:

```
print(pow(5,2)) #25
print(max(5,9,5,2,66,2,3)) #66
print(min(5,-9,5,-2,66,12,3)) #-9
print(sum((5,-9,5,-2,66,12,3))) #80
print(sum([5,-9,5,-2,66,12,3])) #80
```

Задания для закрепления изученного материала

1. Дано трехзначное число, определить, входит ли в него цифра 5.
2. Напишите программу, которая вычисляет сумму первых **n** целых положительных чисел. Количество суммируемых чисел должно вводиться во время работы программы.

3. Напишите программу, которая находит количество и сумму целых положительных чисел, кратных 3 и меньших 100.
4. Организовать беспрерывный ввод чисел с клавиатуры, пока пользователь не введёт 0. После ввода нуля, показать на экран количество чисел, которые были введены, их общую сумму и среднее арифметическое.
5. Необходимо суммировать все нечётные целые числа в диапазоне, который введёт пользователь с клавиатуры

Задания для самостоятельной работы

6. Дано четырехзначное целое число. Определить входит ли в него цифра 0.
7. Дан список чисел, найти максимум
8. Напишите программу. Дано число меньше одного миллиона. Определить количество цифр в нем.
9. Дан массив числами, например: [10, 20, 30, 50, 235, 3000]. Выведите на экран только те числа из массива, которые начинаются на цифру 1, 2 или 5.
10. Дано число n=1000. Делите его на 2 столько раз, пока результат деления не станет меньше 50. Какое число получится? Посчитайте количество итераций, необходимых для этого (итерация — это проход цикла), и запишите его в переменную num.

Создание собственных функций

Функция в Python — это блок кода, который может выполнять определенную операцию и возвращать результат (return).

Функции облегчают повторное использование кода и позволяют разбить программу на более мелкие и управляемые части.

В Python функция объявляется с помощью ключевого слова "**def**".

Вот пример простой функции, которая возвращает сумму двух чисел:

```
def my_sum(x, y):
    result = x + y
    return result
```

Вызов функции my_sum:

```
print(my_sum(6, -9))    # -3
```

Функция может принимать любое количество аргументов.

Пример функции, которая принимает неопределенное количество чисел и возвращает их среднее значение:

```
def average(*args):
    total = sum(args)
    return total / len(args)
```

Используется специальный оператор * перед аргументом «args», чтобы указать, что функция может принимать любое количество аргументов.

Затем вычисляется сумма всех аргументов с помощью функции **"sum"**, а затем возвращается среднее значение (сумму делим на количество аргументов).

Вызов функции:

```
print(average(6,9,6,1,2))     # 4.8
```

Функции могут возвращать несколько значений.

Рассмотрим пример функции, которая возвращает минимальное и максимальное значение из списка чисел:

```
def find_min_max(numbers):
    min_number = min(numbers)
    max_number = max(numbers)
    return min_number, max_number
```

Здесь используется функция **"min"** и **"max"** для поиска минимального и максимального значения в списке. Затем возвращаются эти значения в виде кортежа.

Вызов функции:

```
print(find_min_max((5,3,5,1,66,0,-9))) # (-9, 66
```

Функции могут быть также рекурсивными, то есть они могут вызывать сами себя.

Рассмотрим пример функции, которая вычисляет факториал числа:

```
def factorial(n):
    if n == 0:
        return 1
    else:
        return n * factorial(n-1)
```

В функции проверяется, равно ли число **"n"** нулю. Если это так, то функция возвращает 1. В противном случае функция вызывает функцию **"factorial"** (саму себя) с аргументом **n-1** и умножает результат на **n**.

Вызов:

```
print(factorial(5))  #120
```

Задания для закрепления изученного материала

1. Напишите функцию, которая принимает на вход число **n** и возвращает список всех делителей этого числа.
2. Напишите функцию, которая принимает на вход число и возвращает True, если оно является числом Армстронга, и **False** в противном случае. (Число Армстронга — это число, которое равно сумме своих цифр, возведенных в степень, равную количеству цифр в числе).
3. Напишите функцию, которая принимает на вход список чисел и возвращает список, содержащий только четные числа.

Задания для самостоятельной работы.

1. Напишите функцию, которая принимает на вход список строк и возвращает список строк, в которых содержится подстрока "python".
2. Напишите функцию, которая принимает на вход список стран и возвращает список, содержащий только страны, которые начинаются на букву **K**.
3. Вирусная клетка, выделяет 4 новых бактерии в минуту. каждые 100 бактерий объединяются в новую вирусную клетку, которая также выделяет 4 новых бактерии в минуту. сколько вирусных клеток будет через час.

Классы

Класс включает набор переменных и функций для управления этими переменными. Переменные называют атрибутами, а функции — методами.

В языке программирования Python класс — это шаблон для создания объектов, который определяет состояние и поведение объекта.

Класс определяет атрибуты и методы объекта, которые могут быть использованы для управления состоянием объекта и выполнения операций.

Создание класса в Python начинается с ключевого слова **class**, за которым следует имя класса, например:

```
class people:
    pass
```

Здесь people — это имя класса. Ключевое слово pass указывает, что тело класса пустое, то есть он не определяет никаких атрибутов или методов.

Атрибуты класса определяются как переменные внутри тела класса.

Например:

```
class people:
    height=175
```

Здесь height — это атрибут класса people, который имеет значение 175

Методы класса определяются как функции внутри тела класса. Методы могут получать доступ к атрибутам класса и изменять их значение, а также выполнять другие операции. Например:

```
class people:
    height =175
      def say(self):
         print("Hello, world!")
```

Здесь **say()** — это метод класса people, который выводит строку "Hello, world!" в стандартный вывод.

Здесь **self** — это ссылка на создаваемый объект класса.

Чтобы создать объект класса, необходимо вызвать его конструктор, который имеет такое же имя, как имя класса.

Например:

```
class people:
    height=175
    def say(self):
        print(“Hello, world!”)
Bakyt=people()
```

Здесь Bakyt – это объект класса people, созданный с помощью вызова конструктора people().

Для обращения к атрибутам и методам класса в Python используется оператор «точка» (.).

Для доступа к атрибутам класса необходимо написать имя объекта, за которым следует оператор «точка» и имя атрибута. Например:

```
class people:
    height =175
    def say(self):
        print(“Hello, world!”)
Bakyt=people()
x=Bakyt.height
print(x)
```

Здесь мы создали объект класса people и обратились к его атрибуту **height** с помощью оператора "точка".

Для вызова методов класса также необходимо использовать оператор **"точка" (.)**.

Например:

```
class people:
    height =175
    def say(self):
        print("Hello, world!")
```

```
Bakyt=people()
x=Bakyt.height
print(x)
Bakyt.say()
```

Здесь мы создали объект класса people и вызвали его метод **say()** с помощью оператора "**точка**". Обратите внимание, что метод должен принимать аргумент **self**, который ссылается на сам объект класса.

В данном случае мы не передаем никаких аргументов, так как метод не требует их.

Задания для закрепления изученного материала

1. Создать класс «окружность», который включает два метода: для нахождения длины окружности, для нахождения объема шара.
2. Создать класс «параллелепипед», который включает два метода: для нахождения объема параллелепипеда, для нахождения площади поверхности параллелепипеда.
3. Создать класс «объемы», который включает три метода: для нахождения объема цилиндра, для нахождения объема конуса, для нахождения объема шара.

Задания для самостоятельной работы

4. Создать класс Автомобиль. И реализовать функции для изменения атрибутов класса (Марка, цвет, год выпуска, объем двигателя).
5. Создать класс Робот-пылесос. И реализовать функции для изменения атрибутов класса (Уровень заряда, список комнат).
6. Создайте класс Прямоугольник, который имеет атрибуты «ширина» и «высота», и методы для возвращения площади прямоугольника и возвращения периметра прямоугольника.

Конструкторы и деструкторы

Конструктор класса

В Python конструктор класса — это метод с именем **__init__**. Он вызывается при создании нового объекта класса и позволяет инициализировать его атрибуты. Общий формат конструктора класса выглядит так:


```
class MyClass:
    def __init__(self, arg1, arg2, ...):
        self.attr1 = arg1
        self.attr2 = arg2
```

Здесь self — это ссылка на создаваемый объект класса, а **arg1**, **arg2**, ... — это аргументы, которые передаются при создании объекта.

Внутри конструктора можно производить различные операции, например, вычислять значения атрибутов на основе переданных аргументов, создавать другие объекты классов и т. д.

Рассмотрим пример работы с конструктором.

```
#класс "Человек" с атрибутами "имя" (name) и "возраст" (age).
class people:
    def __init__(self, name, age):    #конструктор
        self.name = name
        self.age = age
    def output(self):
        print("Меня зовут ",self.name, "\nМой возраст ",self.
age )
#объект класса "Человек" с именем "Bakyt" и возрастом 25.
man1 = people("Бакыт", 25)
man1.output()    #вызов функции вывода
```

Результат:

```
Меня зовут  Бакыт
Мой возраст  25
```

Деструктор класса

Деструктор класса вызывается автоматически при удалении. В Python деструктор класса — это метод **__del__** объекта класса (сборщик мусора Python).

Деструкторы используются для выполнения операций по очистке ресурсов, которые были выделены объекту во время его жизненного цикла. Например, если объект использует некоторые внешние

ресурсы, такие как файлы, базы данных или сетевые соединения, то деструктор может быть использован для закрытия этих ресурсов перед удалением объекта.

Вот пример простого класса, который определяет деструктор:

```
class MyClass:
    def __init__(self):
        print("Создание объекта")
    def __del__(self):
        print("Удаление объекта")
obj = MyClass()
del obj
```

Задания для закрепления изученного материала

1. Написать программу, которая содержит класс. Класс назван вашей фамилией. Класс содержит компоненты для хранения вашего имени, года рождения и массив оценок по программированию. Инициализацию данных произвести в конструкторе.
2. Написать программу, которая содержит класс «число» Класс, включает методы для нахождения суммы и количества делителей числа. Инициализацию данных произвести в конструкторе.
3. Написать программу, которая содержит класс «массив». Класс, включает методы для нахождения суммы и количества элементов массива, которые являются простыми числами. Инициализацию данных произвести в конструкторе.

Задания для самостоятельной работы

1. Создать класс Млекопитающие. Использовать атрибуты и методы по смыслу.
2. Создать класс Бобовые. Использовать атрибуты и методы по смыслу.
3. Создать класс Жилой_Дом. Использовать атрибуты и методы по смыслу.

Наследование, инкапсуляция и полиморфизм

Наследование — это механизм, который позволяет создавать новые классы на основе уже существующих. В Python наследование реа-

лизуется очень просто с использованием ключевого слова class и указания базового класса в круглых скобках после имени нового класса.

Рассмотрим простой пример наследования:

```
class Animal:
    def __init__(self, name):
        self.name = name
    def speak(self):
        pass
class Dog(Animal): # Dog наследует от Animal
    def speak(self):
        return “Woof!”
class Cat(Animal): # Cat также наследует от Animal
    def speak(self):
        return “Meow!”
```

Определение базового класса **Animal:** В этом классе определяется базовое поведение для всех животных. У него есть метод **__init__**, который инициализирует атрибут name, представляющий имя животного, и метод speak, который пока не имеет реализации, так как разные животные издают разные звуки.

Определение производных классов **Dog** и **Cat:** Эти классы наследуют от класса **Animal**. Это означает, что они наследуют его атрибуты и методы, такие как name и speak. Однако, они также переопределяют метод speak, чтобы предоставить специфичное для своего вида животного поведение. В случае с классом **Dog**, метод speak возвращает строку **"Woof!"**, а в случае с классом **Cat – "Meow!"**.

```
dog = Dog("Buddy")
cat = Cat("Fluffy")
print(dog.speak())  # Выводит "Woof!"
print(cat.speak())  # Выводит "Meow!"
```

Теперь, когда мы создаем объекты типа **Dog** и **Cat**, они имеют доступ к атрибутам и методам класса **Animal**, так как он является их базовым классом, но при вызове метода speak они вместо

родительской реализации используют свои собственные переопределенные версии этого метода.

Инкапсуляция – это механизм, который позволяет скрыть детали реализации объекта и предоставить только интерфейс для работы с ним. В Python инкапсуляция обеспечивается за счет использования защищенных **(_)** и приватных **(__)** атрибутов и методов класса.

```
class Car:
    def __init__(self):
        self._speed = 0    # защищенный атрибут
    def _accelerate(self):    # защищенный метод
        self._speed += 10
    def drive(self):
        self._accelerate()
        print("Driving at speed:", self._speed)
class ElectricCar(Car):
    def __init__(self):
        super().__init__()
        self.__battery_level = 100    # приватный атрибут
    def charge(self):
        self.__battery_level = 100
    def drive(self):
        if self.__battery_level > 0:
            super().drive()
            self.__battery_level -= 10
            print("Battery level:", self.__battery_level)
        else:
            print("Battery is empty, please charge!")
```

Данный пример демонстрирует концепцию наследования и инкапсуляции в Python, а также некоторые аспекты полиморфизма. Давайте разберем каждую часть классов **Car** и **ElectricCar:**

Класс Car

Определяет основные атрибуты и методы для всех машин.

Имеет защищенный атрибут **_speed**, который представляет текущую скорость машины. Обычно защищенные атрибуты предназначены для внутреннего использования внутри класса или его наследников.

Метод **_accelerate** увеличивает скорость машины на 10 единиц.

Метод drive вызывает **_accelerate** для увеличения скорости и выводит текущую скорость на экран.

Класс ElectricCar

Наследует от класса **Car**, что означает, что он наследует его атрибуты и методы.

В конструкторе **__init__** вызывается конструктор родительского класса Car с помощью функции **super().__init__()**, чтобы установить начальное состояние.

Определяет приватный атрибут **__battery_level**, который представляет уровень заряда батареи. Приватные атрибуты обычно не должны изменяться или использоваться извне класса.

Метод **charge** используется для зарядки батареи.

Переопределяет метод **drive**, чтобы учесть заряд батареи. Если батарея не пуста, он вызывает метод drive родительского класса **Car**, уменьшает уровень заряда батареи и выводит информацию о уровне заряда. Если батарея пуста, выводится сообщение о необходимости зарядки.

Пример использования класса:

```
car = ElectricCar()
car.drive()  # Выведет "Driving at speed: 10" и "Battery level: 90"
```

Этот пример демонстрирует, как класс **ElectricCar** наследует функциональность от класса **Car**, но также имеет собственные уникальные атрибуты и методы, такие как **__battery_level** и **charge**, а также переопределенный метод **drive**.

Полиморфизмм — это способность объектов с одинаковым интерфейсом иметь различную реализацию. В Python полиморфизм достигается за счет использования переопределения методов базовых классов в производных классах.

Пример использования полиморфизма:

```
class Shape:
    def area(self):
        pass
class Rectangle(Shape):
    def __init__(self, width, height):
        self.width = width
        self.height = height
    def area(self):
        return self.width * self.height

class Circle(Shape):
    def __init__(self, radius):
        self.radius = radius
    def area(self):
        return 3.14 * self.radius ** 2
# Функция для расчета площади любой фигуры
def calculate_area(shape):
    return shape.area()
# Пример использования полиморфизма
rectangle = Rectangle(5, 4)
circle = Circle(3)
print("Area of rectangle:", calculate_area(rectangle))  # Выводит площадь прямоугольника
print("Area of circle:", calculate_area(circle))  # Выводит площадь круга
```

Приведенный пример демонстрирует использование полиморфизма в Python с помощью наследования и переопределения методов. Рассмотрим подробнее каждую часть:

Определение базового класса Shape

В этом классе определяется метод area, который должен быть реализован в производных классах для вычисления площади

конкретной фигуры. В данной реализации метод area просто возвращает pass, что означает, что его нужно переопределить в производных классах.

Определение производных классов Rectangle и Circle

Оба класса наследуют от класса **Shape**, что означает, что они должны предоставить реализацию метода **area**.

Класс **Rectangle** принимает ширину и высоту в качестве параметров и определяет метод area, возвращающий площадь прямоугольника, вычисленную как произведение ширины на высоту.

Класс **Circle** принимает радиус в качестве параметра и определяет метод **area**, возвращающий площадь круга, вычисленную как **pi * radius^2**.

Функция calculate_area

Принимает объект фигуры в качестве параметра и вызывает метод area этого объекта для вычисления его площади.

Это пример полиморфизма, поскольку функция работает с объектами разных классов, но использует одинаковый интерфейс **(area)** для получения результатов.

Пример использования:

Создаются объекты **rectangle** (прямоугольник) и **circle** (круг).

Для каждого объекта вызывается функция **calculate_area**, которая автоматически выбирает правильную реализацию метода **area** в зависимости от типа объекта (прямоугольник или круг).

Выводится площадь прямоугольника и круга.

Наследование, инкапсуляция и полиморфизм являются ключевыми концепциями объектно-ориентированного программирования, которые позволяют создавать более гибкий и модульный код. Используя эти концепции, вы можете легко создавать иерархии классов, скрывать реализацию объектов и обеспечивать их различное поведение при вызове одинаковых методов.

Задания для закрепления изученного материала

1. Создайте базовый класс **Vehicle**, содержащий метод **drive()**. Создайте производные классы **Car** и **Motorcycle**, переопределите

метод **drive()** в каждом из них, чтобы он выводил разные сообщения.
2. Создайте класс **Employee**, содержащий атрибуты name и salary. Создайте классы **Manager** и **Developer**, которые наследуют от **Employee** и добавляют свои собственные атрибуты, такие как **department** и **programming_language**.
3. Создайте класс **BankAccount**, содержащий приватный атрибут **balance**. Реализуйте методы **deposit()** и **withdraw()**, чтобы можно было изменять баланс, соблюдая правила инкапсуляции.

Задания для самостоятельной работы

1. Создайте класс **Bank**, содержащий список банковских счетов. Реализуйте методы **add_account()** и **remove_account()** для управления счетами, соблюдая принципы инкапсуляции.
2. Создайте класс **Vehicle**, содержащий метод **move()**. Создайте производные классы **Car**, **Bicycle** и **Boat**, переопределите метод **move()** для каждого из них.
3. Создайте класс **Employee**, содержащий атрибуты **name** и **position**. Создайте класс **Manager**, который наследует от **Employee** и добавляет атрибут **department**.
4. Создайте класс **BankAccount**, содержащий приватные атрибуты **balance** и **account_number**. Реализуйте методы **deposit()** и **withdraw()**, а также метод **get_account_number()**, чтобы получить номер счета.

КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 4

1. Какие основные математические функции предоставляет модуль math в Python?
2. Как можно импортировать модуль math в свою программу?
3. Какова цель создания собственных функций в Python?
4. Как объявить функцию в Python?
5. Чем отличается глобальная переменная от локальной в контексте функции?
6. Какие основные преимущества использования классов в Python?
7. Что такое конструктор класса и зачем он нужен?
8. Как создать конструктор в Python?
9. Что такое деструктор класса и как он работает?
10. В чем разница между конструктором и деструктором класса?
11. Какие атрибуты и методы могут быть определены внутри класса?

12. Как вызвать метод класса извне?
13. Какова основная цель использования классов в объектно-ориентированном программировании?
14. Какова основная разница между функцией и методом?
15. Что такое наследование в объектно-ориентированном программировании? Какие преимущества оно предоставляет?
16. Что такое инкапсуляция? Какие уровни доступа к атрибутам и методам класса существуют в Python?
17. Что такое полиморфизм? Как он реализуется в Python?
18. Какие основные принципы объектно-ориентированного программирования реализуются с помощью наследования, инкапсуляции и полиморфизма?
19. Какие функции выполняют базовый класс и производные классы в иерархии наследования?

Раздел 5

БИБЛИОТЕКА TKINTER

Создание пользовательского интерфейса

Tkinter — это стандартная библиотека графического интерфейса пользователя (GUI, graphical user interface) для языка программирования Python.

(Библиотека Tkinter установлена в Python в качестве стандартного модуля, поэтому не нужно устанавливать что-либо для его использования.)

Она предоставляет разработчикам возможность создавать приложения с графическим интерфейсом, которые могут работать на различных операционных системах, включая Windows, macOS и Linux.

В Tkinter реализованы различные элементы интерфейса, такие как кнопки, текстовые поля, метки и многое другое.

Вы можете настраивать их параметры, располагать на форме, задавать обработчики событий и многое другое.

Создание окна

Для того чтобы создать окно *(рис. 5.1.)*, необходимо прописать следующий программный код:

```
from tkinter import *
myWindow = Tk()
myWindow.title("Добро пожаловать!")
myWindow.geometry('600x400')
myWindow.mainloop()
----------------------------------------------------------------
```


`from tkinter import *` — ***импортирует библиотеку***

`Tk()` — создает объект (myWindow) нового окна

Метод `title()` — создает заголовок окна

Метод `geometry()` — устанавливает размер окна

Метод `mainloop()` — запускает главный цикл приложения, который отвечает за обработку событий и отрисовку элементов интерфейса.

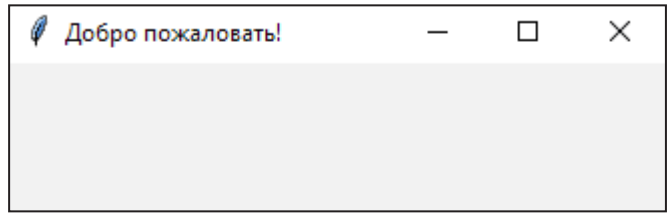


Рисунок 5.1. Пользовательское окно

Менеджеры геометрии

В Tkinter существует три менеджера геометрии — упаковщик, сетка и размещение по координатам.

`Pack` — располагает виджеты (элементы) в ряд, который может быть вертикальным или горизонтальным.

`Grid` — располагает виджеты (элементы) в виде таблицы, используя ряды и столбцы.

`Place` — позволяет располагать виджеты (элементы) в произвольном порядке на окне или фрейме, указывая их координаты **`x`** и **`y`**.

Создание элемента «Надпись» (Label)

Элемент Label (виджет) позволяет выводить статический текст без возможности редактирования.

Label отображает текст в окне и служит в основном для информационных целей (вывод сообщений, подпись других элементов интерфейса).

Некоторые свойства (параметры):

- background – фоновый цвет
- font – шрифт и размер текста

- ❑ foreground – цвет текста
- ❑ text – устанавливает текст метки

Пример создания надписи:

```
x= Label(myWindow, text="Привет", font=("Arial Bold", 50))
```

Рассмотрим пример создания окна, с добавлением двух надписей (рис. 5.2).

```
from tkinter import *
myWindow = Tk()
myWindow.title("Добро пожаловать!")
myWindow.geometry('600x400')
x= Label(myWindow, text="Привет", font=("Arial Bold", 50))
x.grid(column=0, row=0)
y= Label(myWindow, text="Добро пожаловать", font=("Arial Bold",
50),foreground ="red")
y.grid(column=0, row=1)
myWindow.mainloop()
```

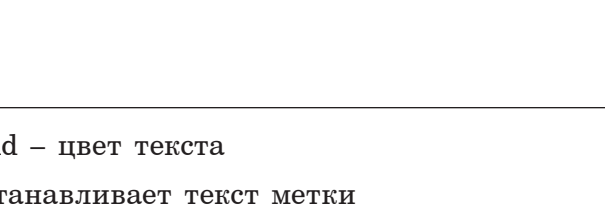


Рисунок 5.2. Использование label

Создание элемента «Текстовое поле» (Entry)

Текстовые поля предназначены для ввода информации пользователем. Однако нередко также для вывода, если предполагается, что текст из них будет скопирован. Текстовые поля как элементы графического интерфейса бывают однострочными Entry и многострочными **Text**.

Методы:

- ❑ **insert(index, str)** – вставляет в текстовое поле строку по определенному индексу.

- ❑ **get()** – возвращает введенный в текстовое поле текст.
- ❑ **delete(first, last=None)** – удаляет символ по индексу first. Если указан параметр last, то удаление производится до индекса last. Чтобы удалить до конца, в качестве второго параметра можно использовать значение END.
- ❑ **focus()** – установить фокус на текстовое поле.

Создание элемента «Кнопка» (Button)

Одним из наиболее используемых компонентов в графических программах является кнопка.

В tkinter кнопки представлены классом Button.

Основные параметры:

- ❑ **command** – функция, которая вызывается при нажатии на кнопку
- ❑ **text** – устанавливает текст метки
- ❑ **width** – ширина виджета

Пример создания кнопки:

```
z= Button(myWindow, text="Нажми меня")
z.grid(column=0, row=2)
```

Пример. Создать программу для сложения двух чисел *(рис. 5.3.)*.

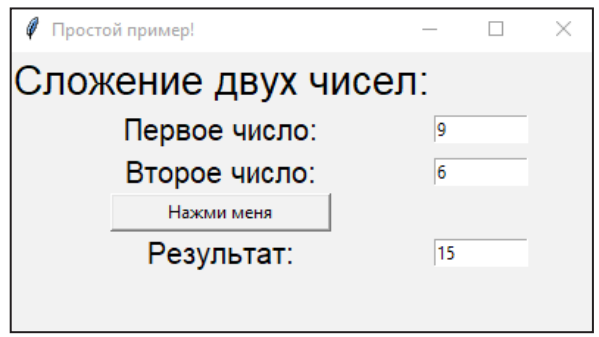


Рисунок 5.3. Программа для сложения двух чисел

Программный код:

```
from tkinter import *

def  sum():
     va11=int(x1.get())
     va12=int(y1.get())
     va13=va11+va12
     z1.insert(0,va13)

myWindow = Tk()
myWindow.title("Простой пример!")
myWindow.geometry('400x200')
t= Label (myWindow, text="Сложение двух чисел:", font=("Arial", 20))
t.grid(column=0, row=0)

x= Label(myWindow, text="IIepBoe число:", font=("Arial", 15))
x.grid(column=0, row=1)
x1= Entry(myWindow, width=10)
x1.grid(column=1, row=1)

y= Label(myWindow, text="Второе число:", font=("Arial", 15))
y.grid(column=0, row=2)
y1= Entry(myWindow, width=10)
y1.grid(column=1, row=2)

z= Label(myWindow, text="Результат:", font=("Arial", 15))
z.grid(column=0, row=4)
z1= Entry(myWindow, width=10)
z1.grid(column=1, row=4)

b= Button(myWindow, text="Нажми меня", width=20, command=sum)
b.grid(column=0, row=3)

myWindow.mainloop()
```

Задания для закрепления изученного материала

1. Создать окно. В текстовое поле ввести номер месяца. При нажатии на кнопку вывести в другое текстовое поле название времени года, в зависимости от введенного номера месяца.

2. Создать окно. В текстовое поле ввести целое число. При нажатии на кнопку вывести в другое текстовое поле все делители введенного числа через запятую.
3. Создать окно. Ввести в текстовое поле любой текст. При нажатии на кнопку вывести в другое текстовое поле Ваше имя если оно было найдено в веденном тексте (в первом текстовом поле).

Задания для самостоятельной работы.

1. Создать окно. В первое текстовое поле ввести число. Во второе текстовое поле ввести число. При нажатии на кнопку вывести в третье текстовое поле остаток от деления двух целых чисел.
2. Создать окно. В первое текстовое поле ввести значение катета. Во второе текстовое поле ввести значение катета. При нажатии на кнопку вывести в третье текстовое поле значение гипотенузы прямоугольного треугольника.
3. Создать окно. В текстовое поле ввести список из целых чисел. При нажатии на кнопку вывести во второе текстовое поле максимальное число из введенного списка.

Окна сообщений

Всплывающее окно с помощью Tkinter, создает класс messagebox.

Подключение класса:

```
from tkinter import messagebox
```

Методы вывода окна сообщения:

- showinfo() — показывает информационное сообщение *(рис. 5.4.)*.

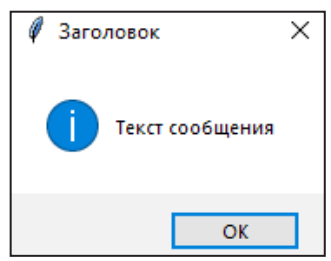


Рисунок 5.4. Окно информационного сообщения

- `showwarning()` — показывает предупреждающее сообщение *(рис. 5.5.)*.

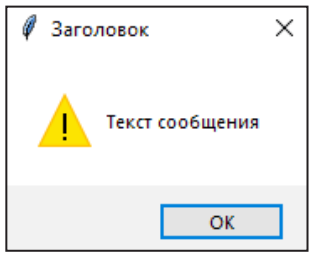


Рисунок 5.5. Окно предупреждающего сообщения

- `showerror()` — показывает сообщение об ошибке *(рис. 5.6.)*.

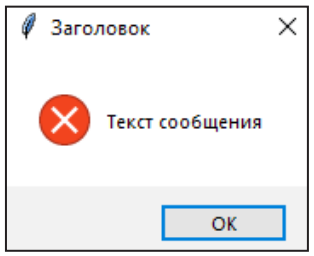


Рисунок 5.6. Окно сообщения об ошибке

Создание информационного окна сообщения:

```
messagebox.showinfo('Заголовок', 'Текст сообщения') #(рис. 5.4.).
```

Рассмотрим пример создания окна информационного сообщения *(рис. 5.7)*.

```
from tkinter import *
from tkinter import messagebox
def clicked():
    messagebox.showinfo('Заголовок', 'Текст сообщения')
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
btn = Button(myWindow, text='Нажми ', command=clicked)
btn.grid(column=0, row=0)
myWindow.mainloop()
```

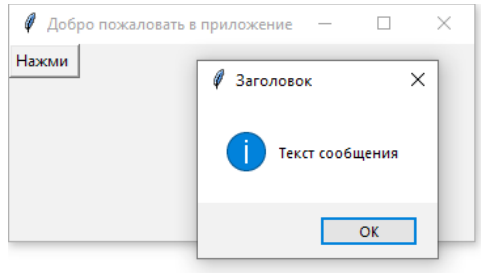



Рисунок 5.7. Окно сообщения

Для создания окон сообщения с возможностью выбора кнопки ДА/НЕТ необходимо использовать следующие методы:

- `res = messagebox.askquestion('Заголовок', 'Текст')`
- `res = messagebox.askyesno('Заголовок', 'Текст')` *(рис. 5.8.)*.

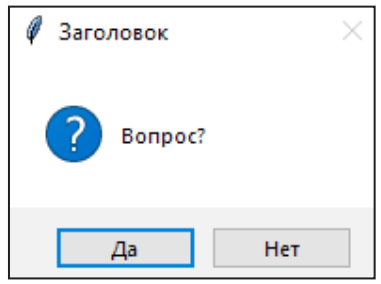


Рисунок 5.8. Метод askyesno()

- `res = messagebox.askyesnocancel('Заголовок', 'Текст')` *(рис. 5.9.)*.

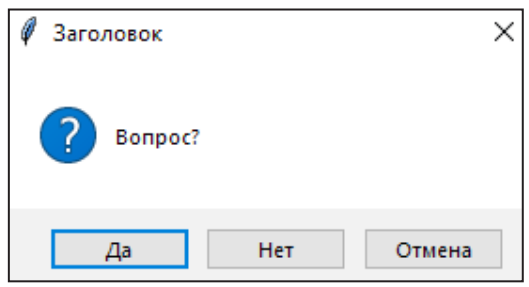


Рисунок 5.9. Метод askyesnocancel()

- `res = messagebox.askokcancel('Заголовок', 'Текст')` (рис. 5.10.).

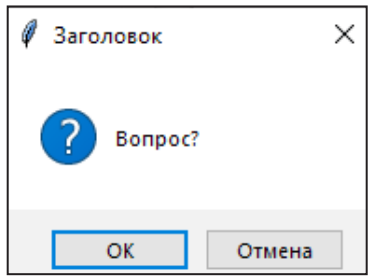


Рисунок 5.10. Метод askokcancel()

- `res = messagebox.askretrycancel('Заголовок', 'Текст')` *(рис. 5.11.)*.

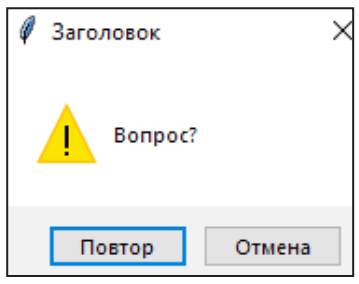


Рисунок 5.11. Метод askretrycancel()

Рассмотрим пример, который выводит окно приветствия пользователю, имя пользователя вводится в текстовое поле, расположенное на форме *(рис. 5.12.)*.

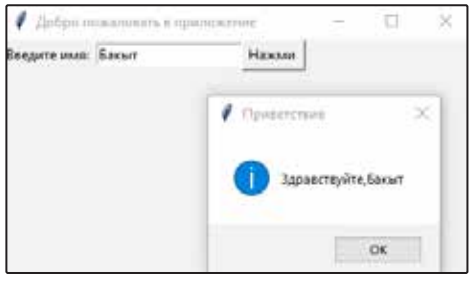


Рисунок 5.12. Пример работы с оном сообщения.

Листинг программы:

```
from tkinter import *
from tkinter import messagebox
def clicked():
```



```
    name=txt.get();
    messagebox.showinfo('Приветствие', 'Здравствуйте,'+str(name))
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
lab=Label(myWindow, text="Введите имя: ")
lab.grid(column=0, row=0)
txt=Entry(myWindow, width=20)
txt.grid(column=1, row=0)
btn = Button(myWindow, text='Нажми ', command=clicked)
btn.grid(column=2, row=0)
myWindow.mainloop()
```

Добавим, в вышеописанный пример, еще одну кнопку, для закрытия окна. При это добавим функцию проверки нажатия кнопки «Да», при подтверждении закрытия, в окне сообщения *(рис. 5.13.)*.

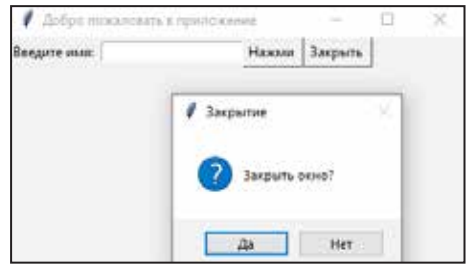


Рисунок 5.13. Пример обработки кнопки «Да»

Листинг программы:

```
from tkinter import *
from tkinter import messagebox
def clicked():
    name=txt.get();
    messagebox.showinfo('Приветствие', 'Здравствуйте,'+str(name))
def close():
    x=messagebox.askyesno('Закрытие', 'Закрыть окно?')
    if x==True: #нажатие кнопки "Да"
```

```
        myWindow.destroy() #закрывает окно
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
lab=Label(myWindow, text="Введите имя: ")
lab.grid(column=0, row=0)
txt=Entry(myWindow, width=20)
txt.grid(column=1, row=0)
btn = Button(myWindow, text='Нажми ', command=clicked)
btn.grid(column=2, row=0)
btn1 = Button(myWindow, text='Закрыть ', command=close)
btn1.grid(column=3, row=0)
myWindow.mainloop()
```

Задания для закрепления изученного материала

1. Создать окно. В текстовое поле ввести номер дня недели. При нажатии на кнопку вывести информационное окно сообщения с названием дня недели.
2. Создать окно. В текстовые поля ввести фамилию, имя, год рождения. При нажатии на кнопку в информационном окне сообщения вывести имя и время года, когда родился этот человек.
3. Создать окно. В текстовое поле ввести любое предложение. При нажатии на кнопку в информационном окне сообщения вывести длину его самого короткого слова.

Задания для самостоятельной работы

Реализовать форму регистрации:

В форме необходимо предусмотреть возможность ввода: фамилии, имени, индекса, почтового адреса (E-mail), и телефона.

Организовать проверку введенных данных (если **не верно**, вывести соответствующие сообщения об ошибках):

- ❑ проверить все поля на наличие введенных символов (не должно быть пустых полей).
- ❑ проверить поле E-mail на наличие символов «@» и «.».
- ❑ проверить поле индекс, должно быть не более 6 символов.

- ❑ проверить поле телефон, должны быть только цифровые символы.

Если **все верно**, вывести сообщение об успешной регистрации.

Флажки и радиокнопки

Элемент Checkbutton

Элемент Checkbutton представляет собой флажок, который может находиться в двух состояниях: отмеченном и неотмеченном.

Checkbutton в основном используется для предоставления пользователю множества вариантов выбора, из которых пользователю необходимо выбрать лишь один.

Некоторые свойства (параметры):

- ❑ `command` — ссылка на функцию, которая вызывается при нажатии на флажок
- ❑ `offvalue` — значение флажка в неотмеченном состоянии, по умолчанию равно 0
- ❑ `onvalue` — значение флажка в отмеченном состоянии, по умолчанию равно 1
- ❑ `text` — текст элемента
- ❑ `textvariable` — привязанный к тексту объект `StringVar`
- ❑ `variable` — ссылка на переменную, как правило, типа `IntVar`, которая хранит состояние флажка

Checkbutton позволяет привязать переменную через параметр variable, который представляет значение флажка.

Например, можно привязать параметр variable к любой переменной типа **IntVar**.

- ❑ `stringVar` — хранит строку (тип `str`);
- ❑ `intVar` — хранит целое число (тип `int`);
- ❑ `DoubleVar` — хранит вещественное число (тип `float`);
- ❑ `BooleanVar` — хранит логическую величину (тип `boolean`).

В отмеченном состоянии привязанный объект **IntVar** имеет значение 1, а в неотмеченном – 0.

В итоге через **IntVar** мы можем получать значение, указанное пользователем.

Рассмотрим пример обработки флажка *(рис. 5.14.)*.

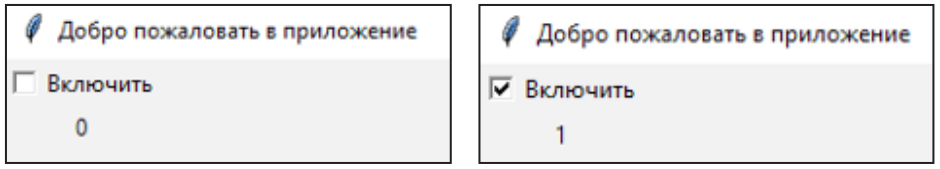


Рисунок 5.14. Обработка флажка (включено/выключено).

Листинг программы:

```
from tkinter import *
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
ch= IntVar()
ch1=Checkbutton(text="Включить", variable=ch)
ch1.grid(column=0, row=0)
lab=Label(myWindow,textvariable=ch)
lab.grid(column=0, row=1)
myWindow.mainloop()
```

Рассмотрим обработчик изменения флажка. С помощью параметра command можно установить функцию, которая будет вызываться при изменении состояния флажка *(рис. 5.15.)*.

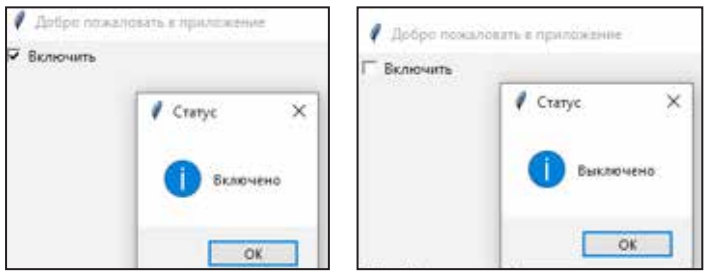


Рисунок 5.15. Обработка флажка (использование функции)

Листинг программы:

```
from tkinter import *
from tkinter import messagebox
def changed():
    if ch.get()==1:
        messagebox.showinfo('Статус', 'Включено')
    else:
        messagebox.showinfo('Статус', 'Выключено')
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
ch= IntVar()
        ch1=Checkbutton(text='Включить", variable=ch,
command=changed)
ch1.grid(column=0, row=0)
myWindow.mainloop()
```

Также обрабатывая флажок, можно изменять текст надписи флажка. Для установки текста флажка можно использовать параметры **text** и **textvariable**. Можно привязать текст флажка к его значению с помощью **textvariable**.

Например, для хранения текста в отмеченном и неотмеченном состояниях определены две переменные: **ch_on** и **ch_off**. Переменная ch инициализируется тем же значением **(ch_on)**, что и параметр onvalue, поэтому по умолчанию флажок будет отмечен. Поскольку его параметры **textvariable** и **variable** привязаны к одной и той же переменной **ch**, то они будет изменяться синхронно *(рис. 5.16.)*.

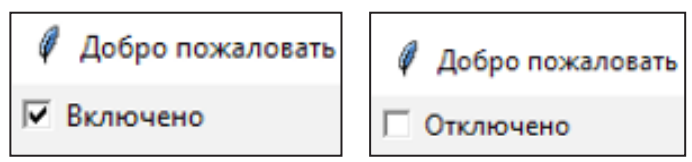


Рисунок 5.16. Обработка флажка (изменение текста флажка)

Листинг программы:

```
from tkinter import *
```

```
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
ch_on="Включено"
ch_off="Отключено"
ch=StringVar(value=ch_on)
ch1=Checkbutton(textvariable=ch, variable=ch, offvalue=ch_off,
onvalue=ch_on)
ch1.grid(column=0, row=0)
myWindow.mainloop()
```

Очень часто, в программах, обрабатываются группы флажков, рассмотрим пример обработки нескольких флажков *(рис. 5.17.)*.

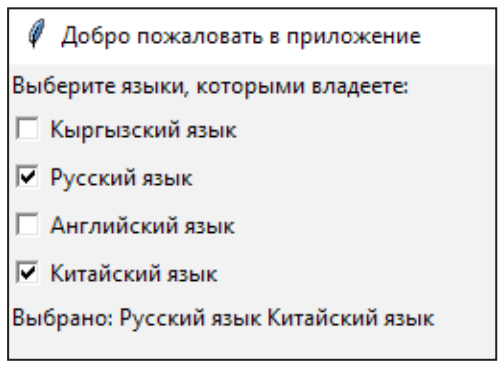


Рисунок 5.17. Обработка нескольких флажков.

Листинг программы:

```
from tkinter import *
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
def select():
    result = "Выбрано: "
    if kyrg.get() == 1: result += "Кыргызский язык "
    if rus.get() == 1: result += "Русский язык "
    if eng.get() == 1: result += "Английский язык "
```



```
    if chin.get() == 1: result += "Китайский язык "
    languages.set(result)
lb1=Label(myWindow, text="Выберите языки, которыми владеете: ")
lb1.grid(column=0, row=0, sticky=SW)
kyrg=IntVar()
ch_kyrg=Checkbutton(text="Кыргызский язык", variable=kyrg,
command=select)
ch_kyrg.grid(column=0, row=1, sticky=SW)
rus=IntVar()
ch_rus=Checkbutton(text="Русский язык", variable=rus,
command=select)
ch_rus.grid(column=0, row=2, sticky=SW)
eng=IntVar()
ch_eng=Checkbutton(text="Английский язык", variable=eng,
command=select)
ch_eng.grid(column=0, row=3, sticky=SW)
chin=IntVar()
ch_chin=Checkbutton(text="Китайский язык", variable=chin,
command=select)
ch_chin.grid(column=0, row=4, sticky=SW)
languages = StringVar()
lb2=Label(myWindow, textvariable=languages)
lb2.grid(column=0, row=5, sticky=SW)
myWindow.mainloop()
```

Элемент Radiobutton

Radiobutton представляет переключатель, который может находиться в двух состояниях: отмеченном или неотмеченном. Но в отличие от **Checkbutton** переключатели могут создавать группу, из которой одномоментно можно выбрать только один переключатель.

Некоторые свойства (параметры)

- `command` — ссылка на функцию, которая вызывается при нажатии на флажок
- `image` — графическое изображение, отображаемое элементом

- text — текст элемента
- textvariable — привязанный к тексту объект StringVar
- variable — ссылка на переменную, как правило, типа IntVar, которая хранит состояние переключателя
- value — значение переключателя

Рассмотрим пример реализации группы радиокнопок, для выбора страны проживания *(рис. 5.18.)*.

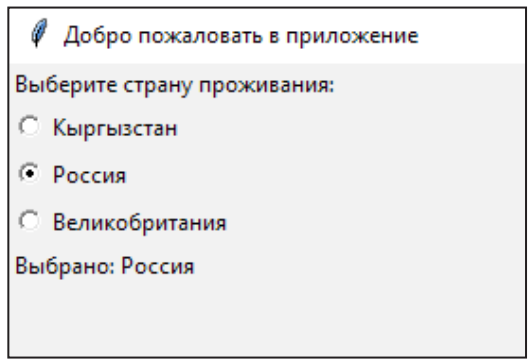


Рисунок 5.18. Пример работы с радиокнопками.

Листинг программы:

```
from tkinter import *
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
def select():
    result = "Выбрано: "+country.get()+""
    res.set(result)
lb1=Label(myWindow, text="Выберите страну проживания: ")
lb1.grid(column=0, row=0, sticky=SW)
res = StringVar()
country = StringVar(value="Кыргызстан")
ch_kg=Radiobutton(text="Кыргызстан", value="Кыргызстан",
variable=country, command=select)
ch_kg.grid(column=0, row=1, sticky=SW)
```



```
ch_ru=Radiobutton(text="Россия", value="Россия",variable=
country, command=select)
ch_ru.grid(column=0, row=2, sticky=SW)
ch_en=Radiobutton(text="Великобритания",value=
"Великобритания",variable=country, command=select)
ch_en.grid(column=0, row=3, sticky=SW)
lb2=Label(myWindow, textvariable=res)
lb2.grid(column=0, row=5, sticky=SW)
myWindow.mainloop()
```

Задания для закрепления изученного материала

1. Напишите программу для обработки «Анкеты переводчика». Использовать флажки. В анкете переводчик должен указать языки, которыми он владеет, в результате сценарий выдает сумму вознаграждения. (За знание каждого языка назначается определенная сумма).
2. Написать программу для нахождения длины окружности, площади круга, объема шара по заданному радиусу. Радиус вводится пользователем. И в зависимости от выбранной радиокнопки, производится расчет длины, площади или объема, и соответственно выводится ответ в текстовом окне.
3. Написать программу для нахождения объемов тел (конус, цилиндр). Радиус и высота вводятся пользователем. И в зависимости от выбранной радиокнопки, производится расчет объема конуса или цилиндра и соответственно выводится ответ в текстовом окне.

Задания для самостоятельной работы

Напишите программу обработки анкеты слушателя курсов. Заполните анкету слушателя курсов. Пользователь может выбрать курс, его продолжительность, язык. В зависимости от этих параметров определяется стоимость обучения.

Работа со списком объектов

Элемент Listbox

Элемент Listbox представляет собой список объектов. При этом можно выбирать один или множество элементов списка.

Некоторые свойства (параметры):

- `listvariable` — список элементов, которые добавляются в `ListBox`
- `bg` — фоновый цвет
- `bd` — толщина границы вокруг элемента
- `ont` — настройки шрифта
- `fg` — цвет текста
- `selectbackground` — фоновый цвет для выделенного элемента
- `selectmode` — определяет, сколько элементов могут быть выделены. Может принимать следующие значения: `BROWSE`, `SINGLE`, `MULTIPLE`, `EXTENDED`.
- `height` — высота элемента в строках. По умолчанию отображает 10 строк
- `width` — устанавливает ширину элемента в символах. По умолчанию ширина 20 символов

`Listbox` имеет ряд методов для управления поведением элемента и его содержимым.

Некоторые из них:

- `curselection()` — возвращает набор индексов выделенных элементов
- `delete(first, last = None)` — удаляет элементы с индексами из диапазона `[first, last]`. Если второй параметр опущен, то удаляет только один элемент по индексу first.
- `get(first, last = None)` — возвращает кортеж, который содержит текст элементов с индексами из дипазона `[first, last]`. Если второй параметр опущен, возвращается только текст элемента с индексом first.
- `insert(index, element)` — вставляет элемент по определенному индексу
- `size()` — возвращает количество элементов

Рассмотрим пример, реализации списка *(рис. 5.19.)*.

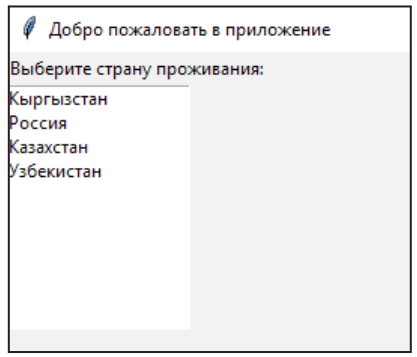


Рисунок 5.19. Пример реализации списка

Листинг программы:

```
from tkinter import *
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
country=["Кыргызстан","Россия","Казахстан","Узбекистан"]
x=Variable(value=country)
lb1=Label(myWindow, text="Выберите страну проживания: ")
lb1.grid(column=0, row=0, sticky=SW)
sp=Listbox(listvariable=x)
sp.grid(column=0, row=2, sticky=SW)
myWindow.mainloop()
```

Добавим функцию обработки списка в предыдущий пример *(рис. 5.20.)*.

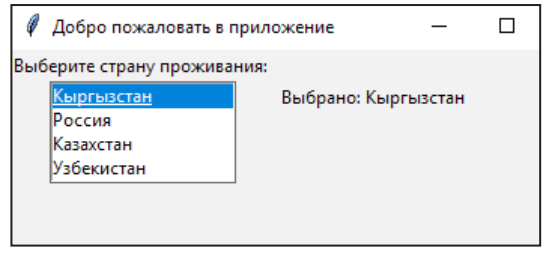


Рисунок 5.20. Пример реализации списка, с функцией обработки

Листинг программы:

```
from tkinter import *
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
def select(event):
    w = event.widget #Создание виджета для списка
    i = int(w.curselection()[0]) #индекс преобразовали в число
    result = "Выбрано: "+sp.get(i)
    res.set(result)
res = StringVar()
country=["Кыргызстан","Россия","Казахстан","Узбекистан"]
x=Variable(value=country)
lb1=Label(myWindow, text="Выберите страну проживания: ")
lb1.grid(column=0, row=0, sticky=SW)
lb2=Label(myWindow, textvariable=res)
lb2.grid(column=1, row=1, sticky=NW)
sp=Listbox(listvariable=x, height=4)
sp.bind("<<ListboxSelect>>", select) #привязка функции к событию
sp.grid(column=0, row=1)
myWindow.mainloop()
```

Элемент Combobox

Элемент **Combobox** представляет выпадающий список, из которого пользователь может выбрать один элемент. Фактически он представляет комбинацию **Entry** и **Listbox**.

Некоторые свойства (параметры):

- `values` — список строк для отображения в Combobox
- `background` — фоновый цвет
- `cursor` — курсор указателя мыши при наведении на текстовое поле
- `foreground` — цвет текста

- `font` — шрифт текста
- `justify` — устанавливает выравнивание текста. Значение `LEFT` выравнивает текст по левому краю, `CENTER` — по центру, `RIGHT` — по правому краю
- `state` — состояние элемента, может принимать значения `NORMAL` (по умолчанию) и `DISABLED`
- `textvariable` — устанавливает привязку к элементу `StringVar`
- `height` — высота элемента
- `width` — ширина элемента

Рассмотрим предыдущий пример, но уже с использованием поля со списком **Combobox** *(рис. 5.21.)*.

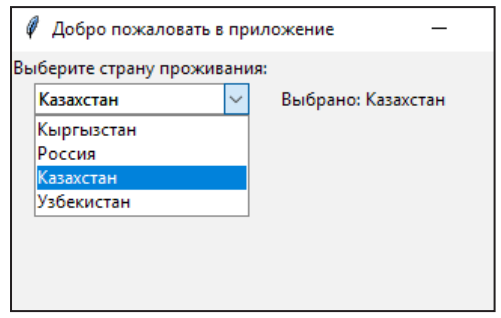


Рисунок 5.21. Пример реализации поля со списком

Листинг программы:

```
from tkinter import *
from tkinter import ttk
myWindow = Tk()
myWindow.title("Добро пожаловать в приложение")
myWindow.geometry('400x200')
def selected(event):
    result = "Выбрано: "+sp.get()
    res.set(result)
res = StringVar()
country=["Кыргызстан","Россия","Казахстан","Узбекистан"]
lb1=Label(myWindow, text="Выберите страну проживания: ")
```

```
lb1.grid(column=0, row=0, sticky=SW)
lb2=Label(myWindow, textvariable=res)
lb2.grid(column=1, row=1, sticky=NW)
sp=ttk.Combobox(values=country)
sp.bind("<<ComboboxSelected>>", selected) #привязка функции
к событию
sp.grid(column=0, row=1)
myWindow.mainloop()
```

Задания для закрепления изученного материала

1. Создать список с наименованием автомобилей. При выборе автомобиля из списка, в текстовом поле вывести наименование выбранного автомобиля (только одно).
2. Создать список с названием овощей и фруктов. При выборе из списка (список можно сделать с мультивыбором), добавить выбранный овощ или фрукт в текстовое поле. В результате в текстовом поле отображаются все выбранные овощи и фрукты.
3. Создать список с наименованием городов. При выборе города из списка, вывести наименование города и стоимость билета на самолет.

Задания для самостоятельной работы

Создать форму регистрации пользователя, которая содержит:

- ❑ поле для ввода фамилии,
- ❑ поле для ввода имени,
- ❑ поле со списком с названиями областей КР,
- ❑ кнопку.

После нажатия на кнопку, все введенные и выбранные данные должны выводиться в одном поле «Данные пользователя».

КОНТРОЛЬНЫЕ ВОПРОСЫ К РАЗДЕЛУ 5

1. Что такое библиотека Tkinter?
2. Какие возможности предоставляет Tkinter для создания пользовательского интерфейса?
3. Как создать окно сообщения в Tkinter?

4. Какие методы используются для управления содержимым окон сообщений?
5. Чем отличаются флажки от радиокнопок в Tkinter?
6. Как создать флажок (чекбокс) в Tkinter?
7. Как создать радиокнопку в Tkinter?
8. Какая разница между флажками и радиокнопками в контексте использования?
9. Как можно управлять списком объектов в Tkinter?
10. Как создать список объектов (listbox) в Tkinter?
11. Как добавить элемент в список объектов в Tkinter?
12. Как удалить элемент из списка объектов в Tkinter?
13. Как получить выбранный элемент из списка объектов в Tkinter?
14. Как настроить прокрутку списка объектов в Tkinter?
15. Как реализовать множественный выбор в списке объектов Tkinter?
16. Как изменить порядок элементов в списке объектов Tkinter?
17. Как привязать функцию к выбору элемента в списке объектов Tkinter?
18. Можно ли использовать изображения в элементах интерфейса Tkinter?
19. Как добавить изображение к окну сообщения в Tkinter?
20. Как настроить внешний вид элементов интерфейса в Tkinter?
21. Как реализовать валидацию ввода данных в Tkinter?
22. Как обработать событие закрытия окна в Tkinter?

ЗАКЛЮЧЕНИЕ

В данном учебном пособии представлен обширный обзор основ языка программирования Python, начиная с его базовых конструкций и заканчивая более сложными темами, такими как работа с файлами и создание пользовательского интерфейса с помощью библиотеки Tkinter.

Каждый раздел пособия содержит не только теоретические материалы, но и практические примеры, что помогает студентам лучше усвоить материал и применить его на практике. Материал представлен доступным и понятным способом, чтобы дать читателям возможность освоить основы программирования на Python даже без предварительного опыта в этой области.

В первом разделе рассмотрены основы языка Python, начиная с его введения и переходя к условным операторам и циклическим конструкциям. Во втором разделе предоставлена информация по работе со списками, словарями и кортежами, рассмотрены методы работы с ними и их особенности. Третий раздел посвящен обработке исключений и работе с файлами, что является важной частью программирования, особенно при разработке приложений. В четвертом разделе рассмотрены функции и классы, предоставлены материалы по созданию собственных функций и использованию классов в Python. В пятом разделе представлена библиотека Tkinter для создания пользовательского интерфейса, что позволяет разработчикам создавать графические приложения на Python.

Каждый раздел пособия содержит как теоретические материалы, так и практические примеры, что помогает студентам получить не только теоретическое, но и практическое понимание изучаемых тем. Мы уверены, что данное пособие будет полезным как для начинающих программистов, так и для опытных разработчиков, желающих расширить свои знания в области Python.

ГЛОССАРИЙ

Python. Это мощный и простой в использовании язык программирования общего назначения. Он имеет простой и понятный синтаксис, что делает его привлекательным для начинающих и опытных программистов.

Интерпретатор. Программа, которая читает и исполняет исходный код на Python. Он интерпретирует код построчно, что позволяет пользователям запускать код на Python без предварительной компиляции.

Строка (String). Это последовательность символов в Python, заключенная в одинарные, двойные или тройные кавычки. Строки могут содержать текст и используются для хранения информации.

Список (List). Это упорядоченная изменяемая коллекция объектов в Python. Элементы списка могут быть любого типа данных и доступны по индексу.

Кортеж (Tuple). Это упорядоченная неизменяемая коллекция объектов в Python. Кортежи подобны спискам, но их содержимое нельзя изменить после создания.

Словарь (Dictionary). Это неупорядоченная коллекция пар ключ-значение. Ключи должны быть уникальными и неизменяемыми, а значения могут быть изменяемыми.

Метод (Method). Функция, связанная с объектом и вызываемая с использованием синтаксиса объект.метод(). Методы являются атрибутами объектов, предоставляющими доступ к поведению этого объекта.

Функция (Function). Блок кода, который выполняет определенную задачу, когда он вызывается. Функции в Python могут принимать аргументы и возвращать значения.

Модуль (Module). Это файл с расширением .py, содержащий код на Python. Модуль может содержать переменные, функции и классы, которые могут быть использованы в других программах.

Исключение (Exception). Это событие, которое происходит во время выполнения программы, и которое нарушает нормальный поток

выполнения кода. Исключения могут быть обработаны с помощью конструкции try-except.

Цикл (Loop). Это структура управления потоком, которая позволяет повторять выполнение определенного блока кода несколько раз. В Python есть два основных типа циклов: for и while.

Условие (Condition). Это выражение, которое оценивается как истина или ложь. Условия используются в конструкциях управления потоком, таких как if, elif и else.

Класс (Class). Это шаблон или чертеж, определяющий атрибуты и методы объекта. Классы используются для создания объектов (экземпляров класса) с определенным поведением и состоянием.

Объект (Object). Это экземпляр класса, который имеет состояние (атрибуты) и поведение (методы). В Python почти все является объектом.

Инкапсуляция. Это механизм, который ограничивает доступ к методам и переменным объекта, обеспечивая скрытость данных и предотвращая их непосредственное изменение.

Наследование. Это механизм, который позволяет новому классу (подклассу) наследовать атрибуты и методы от другого класса (суперкласса). Подкласс может использовать методы суперкласса, а также определять свои собственные методы и атрибуты.

Полиморфизм. Это концепция, позволяющая объектам с одинаковым интерфейсом иметь различную реализацию. Это означает, что объекты разных классов могут обладать одинаковыми методами или атрибутами, но выполнять их по-разному в зависимости от своей конкретной реализации.

СПИСОК ЛИТЕРАТУРЫ

1. *Мур А. Д.* Python GUI Programming with Tkinter [Текст] / А. Д. Мур. — Великобритания: Packt Publishing Ltd, 2018. — 644 с.
2. *Грейсон М.* Python and Tkinter Programming [Текст] / М. Грейсон. — США: Manning Publications, 2024. — 688 с.
3. *Бэрри П.* Изучаем программирование на Python [Текст] / П. Бэрри. — Москва: Эксмо, 2022. — 624 с.
4. *Васильев А. Н.* Программирование на Python в примерах и задачах [Текст] / А. Н. Васильев. — Москва: Эксмо, 2021. — 616 с.
5. *Дауни А. Б.* Think Python. Основы Python [Текст] / А. Б. Дауни. — Москва: Манн, Иванов и Фербер, 2021. — 304 с.
6. *Иванов С. С.* Python. Просто о сложном [Текст] / С. С. Иванов. — Санкт-Петербург: Наука и Техника, 2023. — 368 с.
7. Изучаем Python: программирование игр, визуализация данных, веб-приложения [Текст]. — Санкт-Петербург: Питер, 2022. — 512 с.
8. *Левашов П. Ю.* Python с нуля [Текст] / П. Ю. Левашов. — Санкт-Петербург: Питер, 2024. — 448 с.
9. *Лутц М.* Изучаем Python [Текст] / М. Лутц. — Москва: Диалектика, 2020. — 832 с.
10. *Нисчал Н.* Python — это просто. Пошаговое руководство по программированию и анализу данных [Текст] / Н. Нисчал. — Санкт-Петербург: БХВ-Петербург, 2022. — 416 с.
11. *Хайнеман Д.* Алгоритмы. С примерами на Python [Текст] / Д. Хайнеман. — Санкт-Петербург: Питер, 2024. — 304 с.

Савченко Е. Ю., Мусакулова Ж. А.

ПРОГРАММИРОВАНИЕ НА PYTHON: ОТ БАЗОВЫХ КОНЦЕПЦИЙ К СОЗДАНИЮ GUI

Учебное пособие

Дизайн и компьютерная верстка: *В. Горнушкин*

Подписано к печати ??.??.2024.

Заказ № 57.

Формат бумаги 98 × 72 $^{1}/_{8}$. Объем 12 п.л.

Тираж ??? экз.

ОсОО «НЕО принт»