МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Основы проектирования машин»

АЛГОРИТМИЧЕСКИЕ ОСНОВЫ В ПРОЕКТИРОВАНИИ

Методические рекомендации к лабораторным работам для студентов направления подготовки 15.03.03 «Прикладная механика» очной формы обучения

Могилев 2024

УДК 621.01
ББК 36.4
А87

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Основы проектирования машин» «29» мая 2024 г., протокол № 11

Составитель канд. техн. наук, доц. А. П. Прудников

Рецензент канд. техн. наук, доц. М. Н. Миронова

Изложены цели, содержание и порядок выполнения лабораторных работ.

Учебное издание

АЛГОРИТМИЧЕСКИЕ ОСНОВЫ В ПРОЕКТИРОВАНИИ

Ответственный за выпуск	А. П. Прудников
Корректор	А. Т. Червинская
Компьютерная верстка	Е. В. Ковалевская

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 26 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

Содержание

Введение

Методические рекомендации составлены в соответствии с рабочей программой по курсу «Алгоритмические основы в проектировании» для студентов направления подготовки 15.03.03 «Прикладная механика» очной формы обучения.

Информационные технологии играют важную роль при создании новых продуктов и услуг, улучшении коммуникации и доступа к информации. Обладание навыками программирования позволяет решать сложные задачи и проводить научные исследования, автоматизировать и оптимизировать различные процессы, упрощая работу и повышая ее эффективность. С ростом компьютеризированности всех областей человеческой деятельности востребованность навыков создания собственных программ увеличивается и вместе с тем повышается актуальность проблемы обучения и самообучения навыкам программирования.

Целью изучения дисциплины «Алгоритмические основы в проектировании» является формирование у студентов базовых знаний алгоритмических основ в проектировании, умений и навыков программирования.

Задачи изучения дисциплины «Алгоритмические основы в проектировании»:

– обучиться программировать на языке Python;

– научиться использовать современные методы и средства разработки алгоритмов и программ;

– освоить приемы визуального программирования, изучить способы отладки, испытания программ.

Объектом изучения дисциплины «Алгоритмические основы в проектировании» является язык программирования Python, предметом изучения – структура, принципы и синтаксис языка.

Python является скриптовым языком программирования. Он универсален и поэтому подходит для решения разнообразных задач и для многих платформ: iOS, Android, серверные операционные системы. Он используется для разработки приложений, веб-сайтов, искусственного интеллекта [1–3].

В методических рекомендациях в краткой форме изложены цель, содержание и порядок выполнения лабораторных работ.

Лабораторная работа выполняется в среде IDLE.

Отчет по лабораторной работе оформляется в текстовом редакторе Word и должен содержать:

– цель лабораторной работы;

– описание задания;

– код приложения с комментариями, описывающими код.

Методические рекомендации предназначены для самостоятельной подготовки студентов к лабораторным занятиям по дисциплине «Алгоритмические основы в проектировании».

1 Лабораторная работа № 1. Введение в Python

Цель работы: познакомиться со средой разработки Python.

Теоретические основы

Python – это объектно-ориентированный, интерпретируемый, переносимый язык сверхвысокого уровня.

Программирование на Python позволяет получать быстро и качественно необходимые программные модули. В комплекте вместе с интерпретатором Python идет IDLE (интегрированная среда разработки). По своей сути она подобна интерпретатору, запущенному в интерактивном режиме с расширенным набором возможностей (подсветка синтаксиса, просмотр объектов, отладка).

Python можно рассматривать как:

– интерпретируемый язык: исходный код на Python не компилируется в машинный код, а выполняется непосредственно с помощью специальной программы-интерпретатора;

– интерактивный язык: можно писать код прямо в оболочке интерпретатора и вводить новые команды по мере выполнения предыдущих;

– объектно-ориентированный язык: Python поддерживает принципы объектно-ориентированного программирования, которые подразумевают инкапсуляцию кода в особые структуры, именуемые объектами.

Скорость выполнения программ несколько ниже по сравнению с компилируемыми языками (C или C++), поскольку Python транслирует инструкции исходного программного кода в байт-код, а затем интерпретирует этот байт-код. Байт-код обеспечивает переносимость программ, поскольку это платформонезависимый формат. Как результат, имеет место преимущество в скорости разработки с потерей скорости выполнения.

Любая программа состоит из последовательности допустимых символов, записанных в определенном порядке и по определенным правилам. Программа включает в себя: комментарии, команды, знаки пунктуации, идентификаторы, ключевые слова.

Комментарии в Python обозначаются предваряющим их символом # и продолжаются до конца строки (т. е. в Python все комментарии являются однострочными), при этом не допускается использование перед символом # кавычек.

Идентификаторы в Python это имена, используемые для обозначения переменной, функции, класса, модуля или другого объекта. Некоторые слова имеют в Python специальное назначение и представляют собой управляющие конструкции языка.

Порядок выполнения работы

После загрузки и установки Python необходимо открыть IDLE – среду разработки на языке Python.

Рассмотрим создание программы, выводящей на экран приветственное сообщение «Hello, students! :))». Для этого необходимо использовать инструкцию print("Hello, students!:))"). Код вводится в IDLE, результат отображается после нажатия Enter. Результат работы программы:

```
print("Hello, students! :))")
Hello, students! :))
```

Изначально IDLE запускается в интерактивном режиме. Интерактивный режим удобен, но крайне ограничен по своим возможностям. В основном, программный код создается в сценарном режиме, сохраняется в файл и запускается после сохранения этого файла. Для того чтобы создать новое окно и перейти в сценарный режим, в интерактивном режиме IDLE необходимо выбрать File – New File или нажать Ctrl + N.

Рассмотрим программу, которая спрашивает имя пользователя и здоровается, обращаясь к нему по имени. Первая строка печатает вопрос («What is your name?»), ожидает, пока пользователь не напечатает что-нибудь и не нажмет Enter, и сохраняет введенное значение в переменной name:

```
name input("What is your name? ")
print("Hi,", name,"!")
```

Для запуска программы необходимо нажать F5 или выбрать в меню IDLE Run – Run Module. Перед запуском IDLE предложит сохранить файл, после чего программа запустится.

Измените приложение, созданное в ходе выполнения данной лабораторной работы таким образом, чтобы программа выводила на экран следующую информацию (каждый студент должен использовать информацию о себе):

– ФИО студента;

– группа студента;

– дата рождения студента.

Вопросы для самоконтроля

1 Почему можно рассматривать Python как интерпретируемый, интерактивный и объектно-ориентированный язык программирования?

2 В чем особенности Python?

3 С чем связана скорость выполнения программ на Python?

4 Что такое IDLE?

5 В чем отличия работы в интерактивном и сценарном режимах среды разработки IDLE?

6 Зачем необходимо использовать комментарии в программе?

2 Лабораторная работа № 2. Типы данных

Цель работы: изучить основные типы данных.

Теоретические основы

Информация, сохраненная в памяти компьютера, может быть разных типов данных. К стандартным типам данных, используемым в Python, относятся: число (Number); строка (String); список (List); кортеж (Tuple); словарь (Dictionary); множество (Set).

Числовой тип данных в Python предназначен для хранения числовых значений и представляет собой неизменяемый тип данных, т. е. изменение значения числового типа приведет к созданию нового объекта в памяти (и удалению старого).

Для хранения и обработки текстовой информации используется строковый тип данных. В Python строки могут задаваться следующими способами:

– строка в одинарных кавычках (апострофах);

– строка в двойных кавычках;

– строка в тройных кавычках.

Особенность строки в тройных кавычках состоит в том, что она может занимать в коде несколько строк и при этом выводиться на экран точно в таком же виде, как и вводится.

Типы переменных – это типы значений, на которые они ссылаются. Для переменных, как правило, используют имена, которые раскрывают их назначение. Имена переменных должны начинаться с буквы, могут содержать буквы, цифры, символы подчеркивания; нежелательно использовать буквы верхнего регистра и нельзя использовать зарезервированные слова.

В Python имеет место динамическая типизация, где любой объект является ссылкой, а типом объекта является то, на что он ссылается. Тип может динамически меняться в процессе выполнения программы, когда ссылка начинает указывать на объект другого типа.

В Python существуют следующие типы операторов: арифметические операторы, операторы сравнения, операторы присваивания, побитовые операторы, логические операторы, операторы членства и тождественности.

Иногда может возникнуть необходимость преобразовать один тип данных в другой. Для этого существуют специальные встроенные функции Python, приведенные в таблице 2.1.

Таблица 2.1 – Функции для преобразования типа данных

Функция	Описание	Пример
int(x [,base])	Преобразование x в целое число	int(12.4)→12
long(x [,base])	Преобразование x в тип long	long(20)→20L
float(x)	Преобразование x в вещественное число	float(10)→10.0
str(x)	Преобразование x в строку	str(10) → "10"

Порядок выполнения работы

Программа должна запрашивать имя пользователя (строка) и почтовый индекс (целое число) и записывать соответственно в переменные *SName*1 и *PIndex*2. Вывести значения и типы переменных на экран.

Преобразовать переменную *PIndex*2 к строковому типу с использованием функции str(). Умножить полученную строку на год рождения пользователя и вывести результат.

Вопросы для самоконтроля

1 Что такое константа?

2 Описать стандартные простые типы данных в Python?

3 Что такое инициализация?

4 Что произойдет, если присвоить строковой переменной целочисленное значение (или наоборот)?

3 Лабораторная работа № 3. Основы ввода-вывода

Цель работы: научиться использовать стандартные функции ввода-вывода Python для написания интерактивных программ, организующих диалог между пользователем и компьютером.

Теоретические основы

Для достижения большей гибкости в отображении текста на экране в строки можно вставлять специальные escape-последовательности, которые не отображаются на экране и всегда начинаются с символа правого слеша ('\').

Если требуется вывести строку точно в таком же виде, как она описана в коде, без всевозможных преобразований символов escape-последовательностей, достаточно перед написанием самой строки поставить префикс в виде символа r (raw – запись/строка). Данный тип строки не преобразует слеши.

Функция для вывода информации на дисплей print(). Стандартная функция print() – это базовая функция для организации вывода одного или нескольких значений по умолчанию на консоль или в любой другой указанный поток.

Функция для ввода данных с клавиатуры input() . Стандартная функция input() – базовая функция для организации ввода данных. В качестве необязательного параметра она принимает строку-приглашение и возвращает строку, вводимую пользователем.

Основные правила использования функции input():

– строка-приглашение не начинается и не заканчивается с символа новой строки, т. е. пользователь должен заблаговременно учесть организацию удобного запроса на ввод данных;

– функция input() часто используется для того, чтобы организовать паузу перед закрытием программы для просмотра результата ее выполнения.

Синтаксис функции input():

```
answer=input("Computer: Hi! How are you?\n You: ")
print("Computer: I'm " + answer + # too :)")
input("\n Press enter for Exit...")
```

Порядок выполнения работы

Написать интерактивную программу, которая будет запрашивать соответствующие данные о студенте (ФИО, дата рождения, адрес (можно только город), любимое хобби, любимый фильм и т. д.), а затем выводить их на экран монитора.

Пример программы:

```
a=input('Введите ваши фамилию, имя, отчество ')
b=input('Сколько вам лет? ')
c=input('Где вы живёте?')
print('Ваше имя,a)
print('Ваш возраст ',b)
print('Вы живете в',c)
>>>
Введите ваши фамилию, имя, отчество Иванов Иван Иванович
Сколько вам лет? 15
Где вы живёте? Уссурийск
Ваше имя Иванов Иван Иванович
Ваш возраст 15
Вы живете в Уссурийск
```

Вопросы для самоконтроля

1 Зачем нужны escape-последовательности?

2 Зачем нужны raw-строки и как их записать в коде?

3 Опишите синтаксис функции print().

4 Опишите синтаксис функции input().

4 Лабораторная работа № 4. Работа со строками

Цель работы: приобрести навыки работы в Python со строками и закрепить их на примере разработки интерактивных приложений.

Теоретические основы

Строка – это неизменяемая последовательность символов.

Доступ к одному символу можно получить с помощью оператора квадратных скобок. Выражение в скобках называется индексом (index). Индекс – это смещение от начала строки; смещение для первого символа – нуль. Таким образом, нумерация в строках начинается с нуля. В качестве индекса можно использовать любое целочисленное выражение, включая переменные или операторы. Можно использовать отрицательные индексы, которые считаются с конца. fruit[-1] выдаст последний символ, fruit[-2] – второй с конца строки и т. д.

Множество алгоритмов включают посимвольную обработку строк, которая, как правило, начинается с начала строки. Такая обработка называется обходом (traversal). Обход может осуществляться с помощью различных циклов:

```
fruit='banana'
ndex=0
while index<len (fruit):
letter-fruit [index]
print (letter)
index+=1
>>>
b
a
n
a
n
a
```

Срезом (slice) называется часть строки. Оператор [n:m] возвращает часть строки от n-го символа до m-го, включая первый, но исключая последний. Если опустить первый индекс, то срез будет начинаться с начала строки. Если опустить второй – срез завершится в конце строки:

```
s = 'Monty Python'
print (s[0:5])
Monty
print (s[6:13])
Python
```

Если первый индекс не меньше второго, то результатом будет пустая строка, если первый и второй индексы отсутствуют – выводится вся строка.

Логический оператор in принимает две строки и возвращает True, если первая является подстрокой второй.

В Python можно сравнивать строки. Все буквы верхнего регистра имеют приоритет перед буквами нижнего, буквы – перед цифрами. При проверке равенства строк можно преобразовать строку в стандартный формат, например, нижний регистр.

Строка является объектом в Python. Объекты содержат данные (фактически саму строку) и методы, которые являются функциями, встроенными в объект и доступными для любого экземпляра (instance) объекта. Функция dir() выводит список доступных методов для объекта, вводимого в качестве параметра.

Метод вызывается аналогично функции, принимает аргументы и возвращает значение. Однако метод и функция различаются синтаксисом: метод вызывается путем добавления имени метода к имени переменной с использованием точки в качестве разделителя.

Порядок выполнения работы

Проверить, будет ли строка читаться одинаково справа налево и слева направо (т. е. является ли она палиндромом).

Сначала введем строку командой: s=input('Введите строку '). Затем определим логическую переменную flag и присвоим ей значение 1: flag=1.

Для начала в введенной строке нужно удалить пробелы. Для этого воспользуемся циклической конструкцией for, которая выполнится столько раз, какую имеет длину строка. Длину строки определим функцией len(s). В теле цикла будем проверять следующее условие: s[i]!=' '. Данное логическое выражение будет истинно в том случае, если i-й элемент строки не будет равен пробелу, тогда выполнится команда, следующая после двоеточия: string+=s[i]. К сроке string, которая была объявлена в начале программы, будет добавляться посимвольно строка s, но уже без пробелов.

Для проверки строки на "палиндром" воспользуемся циклической конструкцией for. Длина половины строки находится делением нацело на 2. Если количество символов нечетно, то стоящий в середине не учитывается, т. к. его сравниваемая пара – он сам. Количество повторов цикла равно длине половины строки. Длину строки определим функцией len(s), где аргумент – введенная нами строка s. Зная длину строки, можно вычислить количество повторов цикла. Для этого целочисленно разделим длину строки на 2: len(s)//2.

Для задания диапазона для цикла используем функцию range(), в которой аргументом будет являться половина длины строки: range(len(s//2)). for i in range(len(s//2)). Если символ с индексом i не равен "симметричному" символу с конца строки (который находится путем индексации с конца) if s[i] != s[–1–i], то переменной flag присваивается значение 0 и происходит выход из цикла командой break.

Далее, при помощи условной конструкции if-else в зависимости от значения flag либо 0, либо 1 выводится сообщение, что строка палиндром, либо нет:

```
s=input ("Введите строку \n")
flag=1
string=' '
```

```
for i in range (len(s)):
if s[i]!=' ':
string+=s[i]
print (string)
for i in range (len(s)//2):
if string[i]!=string[-i-1]:
flag=0
break
if flag: print ('Палиндром')
else: print ('не палиндром")
Пример программы на Python
Введите строку
а роза упала на лапу азора
арозаупаланалапуазора
Палиндром
```

Вопросы для самоконтроля

1 Чем характеризуется строковый тип данных в Python?
2 Зачем нужна индексация строк и как ее использовать?
3 Зачем и как используются срезы строк?
4 Какие основные методы используются для работы со строками?

5 Лабораторная работа № 5. Операторы ветвлений и циклов

Цель работы: изучить синтаксис условных конструкций и циклов языка Python, продемонстрировать возможности конструкций ветвления и циклов на примере разработки интерактивных приложений.

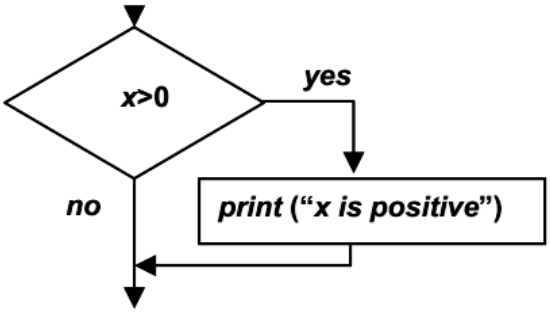
Теоретические основы

Условные конструкции.

Логическими (boolean expression) называются выражения, которые могут принимать одно из двух значений – истину или ложь. Операнды логических операторов должны быть логическими выражениями.

True и False – специальные значения, которые принадлежат к типу bool, они не являются строками.

Условные инструкции (conditional statements) позволяют изменять ход выполнения программы в зависимости от условий. На рисунке 5.1 представлена блок-схема и код для проверки положительности числа x с выводом соответствующего сообщения.

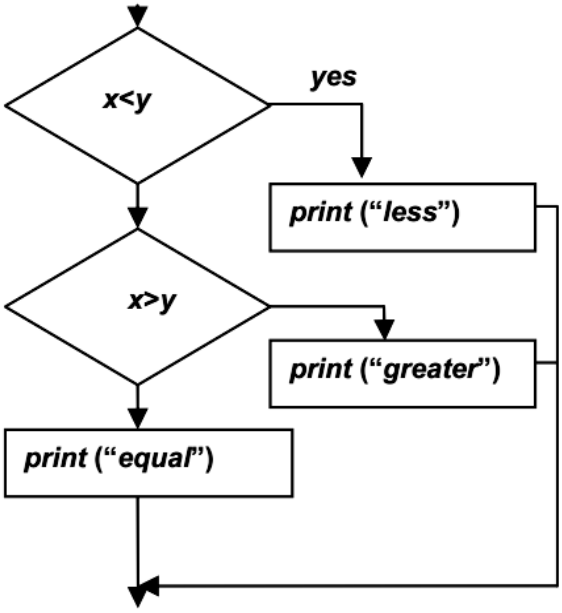



```
if x>0 :
print('x is positive')
```

Рисунок 5.1 – Блок-схема условной конструкции

Логическое выражение после инструкции if называется условием, далее следует символ двоеточия (:) и строка (строки) с отступом. Если логическое условие истинно, то управление получает выражение, записанное с отступами, иначе – выражение пропускается. Не существует ограничения на число инструкций, которые могут встречаться в теле if, но хотя бы одна инструкция там должна быть.

Когда имеется больше двух вариантов выполнения, то необходимо больше двух ветвей. В этом случае, можно воспользоваться сцепленными условиями (chained conditional). На рисунке 5.2 приведен алгоритм сравнения двух чисел x и y.




```
if x<y:
print('x is less than y')
elif x>y:
print('x is greater than y')
else:
print('x and y are equal')
```

Рисунок 5.2 – Сцепленные условия

Инструкция elif – аббревиатура от «else if». Оператор else должен быть в конце инструкции, но может и отсутствовать. Каждое условие проверяется в порядке расположения. Если первое условие ложно, то проверяется следующее и т. д. Если одно из условий истинно, то выполняется соответствующая ветка, и инструкция завершается.

Условия могут быть вложенными. Пример трихотомии, иллюстрирующий сравнение чисел x и y:

```
if x = = y:
print ('x and y are equal')
else:
if x < y:
print ('x is less than y')
else:
print ('x is greater than y')
```

Итерационные алгоритмы.

Поток выполнения для инструкции while имеет вид:

1) проверка условия (condition) (True или False);

2) если условие ложно, происходит выход из инструкции while, и выполнение продолжается со следующей инструкции;

3) если условие истинно, выполняется тело цикла (instruction 1, … , instruction n) и происходит возврат к шагу 1.

Такой поток называется циклом (loop), т. к. шаг 3 возвращает к началу алгоритма. Выполнение тела цикла называется итерацией (iteration). Итерации выполняются, пока условие истинно.

Тело цикла должно изменять одну или более переменных, что в конечном итоге должно привести к ложности условия и завершению цикла. Переменные, которые изменяются каждый раз при выполнении цикла и контролируют завершение цикла, называются итерационными переменными (iteration variable). Если итерационная переменная в цикле отсутствует, то такой цикл будет бесконечным (infinite loop).

Инструкция break используется, когда нужно выйти из бесконечного цикла при наступлении заданного условия.

Инструкция continue пропускает текущую итерацию в цикле и переходит к следующей без завершения тела цикла.

Пример использования конструкции while:

```
while True:
line = input ('Введите строку\n')
if line [0] == '#':
continue
if line =='done':
break
print (line)
print ('Done!')
```

Циклические конструкции.

Цикл for проходит через известное множество элементов столько раз, сколько элементов содержится во множестве.

Конструкция in range(n) используется для цикла прохождения определенного числа шагов (n). Итерационная переменная меняется от 0 до $n - 1$.

Пример цикла for:

```
friends = ['Joseph', 'Glenn', 'Sally']
for friend in friends: print ('Happy New Year', friend) print ('Done!')
>>>
Happy New Year: Joseph
Happy New Year: Glenn
Happy New Year: Sally
Done!
```

Порядок выполнения работы

1 Разработать интерактивную программу «Квадратное уравнение» для решения квадратных уравнений вида: $ax^2 + bx + c = 0$. Программа должна запрашивать соответствующие параметры a, b и c, проверять параметры и выдавать результат.

2 Разработать интерактивную программу «Попробуй угадать число», которая эмулирует игру на отгадывание числа. Суть игры сводится к следующему: компьютер генерирует случайное число из диапазона, к примеру, от 1 до 100, а пользователь пытается отгадать число. При каждой попытке компьютер «подсказывает» игроку, как соизмеряется вариант игрока с загаданным компьютером числом: загаданное число больше или меньше указанного. Как только игрок отгадывает число, компьютер должен «поздравить» его с выводом на экран угаданного числа и количества затраченных игроком попыток. Далее компьютер может «предложить» повторно сыграть в игру или выйти. Для универсальности можно добавить возможность выбора диапазона генерирования компьютером случайных чисел, а также задание ограничения на количество попыток. В случае, если игрок не укладывается в заданное количество попыток, программа должна выводить надпись «Game Over».

3 Найти сумму n элементов следующего ряда чисел: 1; –0,5; 0,25; –0,125 ... n. Количество элементов (n) вводится с клавиатуры. Вывести на экран каждый член ряда и его сумму. В данном случае ряд чисел состоит из элементов, где каждый следующий меньше предыдущего в два раза по модулю и имеет обратный знак. Значит, чтобы получить следующий элемент, надо предыдущий разделить на –2. Какой-либо переменной надо присвоить значение первого элемента ряда (в данном случае это 1). Далее в цикле добавлять ее значение к переменной, в которой накапливается сумма, после чего присваивать ей значение следующего элемента ряда, разделив текущее значение на 2. Цикл должен выполняться n раз. Результат выполнения программы:

```
n=int (input (Введите количество элементов последовательности:)
x=1
s=0
```

```
print(x)
for i in range(n):
s+=x
x/=-2
print (x)
print('Сумма ряда:', s)
>>>
Введите количество элементов последовательности: 5
1
-0.5
0.25
-0.125
0.0625
-0.03125
Сумма ряда: 0.6875
```

Вопросы для самоконтроля

1 Опишите синтаксис простой условной конструкции if.
2 Опишите синтаксис условной конструкции if-else.
3 Опишите синтаксис условной конструкции elif.
4 Для чего используются циклы? Что такое итерация?
5 Какие разновидности циклов существуют?
6 Описать синтаксис цикла с предусловием while.
7 Какова роль оператора break в теле цикла?
8 Какова роль оператора continue в теле цикла?
9 Что такое бесконечный цикл?
10 Опишите синтаксис и принцип работы цикла for.
11 Для чего используется стандартная функция range(..)?

6 Лабораторная работа № 6. Работа с массивами

Цель работы: приобрести навыки работы с массивами в Python.

Теоретические основы

Список / массив (list) – изменяемая последовательность значений.

В отличие от массивов, включающих в себя лишь однотипные элементы, списки не привязаны к определенной разновидности данных, а также не имеют жестких ограничений, связанных с их размером.

Элементы списка заключаются в квадратные скобки. Список внутри другого списка является вложенным (nested). Список без элементов называется пустым. Пустой список создается с помощью пустых квадратных скобок [ ].

Синтаксис для доступа к элементам списка – операторы квадратных скобок. Выражение внутри скобок определяет индекс. Индексация начинается с нуля. Любое целочисленное выражение можно использовать в качестве индекса. Если индекс имеет отрицательное значение, то отсчет элементов ведется в обратном направлении от конца списка.

Оператор «+» объединяет списки. Оператор «*» повторяет список несколько раз. Оператор среза для списков работает похожим образом, как для строк. Оператор in возвращает результат вхождения одного списка в другой список. Для обхода списка удобно использовать цикл for. Для удаления значений списков можно воспользоваться оператором del.

Существуют встроенные функции для работы со списками:

– len() – возвращает длину;

– max() и min() – поиск экстремальных элементов;

– sum() – суммирование элементов;

– list() – преобразует строку в список символов;

– split() – разбивает строку на слова (при вызове split() можно передать аргумент, который задает разделитель, вместо пробела по умолчанию);

– join() – объединяет элементы списков.

Порядок выполнения работы

Задание 1

Дан одномерный массив, состоящий из N целочисленных элементов. Ввести массив с клавиатуры. Найти максимальный элемент. Вывести массив на экран в обратном порядке.

Задание 2

Дан одномерный массив из 8 элементов. Заменить все элементы массива, меньшие 15, их удвоенными значениями. Вывести на экран монитора преобразованный массив.

Задание 3

Дан одномерный массив из 10 целых чисел. Найти максимальный элемент и сравнить с ним остальные элементы. Вывести количество меньших максимального и больших максимального элементов.

Задание 4

Дан одномерный массив, состоящий из N вещественных элементов. Ввести массив с клавиатуры. Найти и вывести минимальный по модулю элемент. Вывести массив на экран в обратном порядке.

Задание 5

Дан одномерный массив целых чисел. Проверить, есть ли в нем одинаковые элементы. Вывести эти элементы и их индексы.

Задание 6

Программа заполняет одномерный массив из 10 целых чисел числами, считанными с клавиатуры. Определить среднее арифметическое всех чисел массива. Заменить элементы массива, большие среднего арифметического на 1.

Задание 7

В массиве R, содержащем 25 элементов, заменить значения отрицательных элементов квадратами значений, значения положительных увеличить на 7, а нулевые значения оставить без изменения. Вывести массив R.

Задание 8

Дан массив, содержащий 10 элементов. Вычислить произведение элементов, стоящих после первого отрицательного элемента. Вывести исходный массив и результат вычислений.

Задание 9

Последовательность a1, a2, …, a12 состоит из нулей и единиц. Поставить в начало этой последовательности нули, а затем единицы.

Задание 10

Дан список целых чисел, содержащий 10 элементов, записать в этот же список сначала все положительные числа, а затем все отрицательные, сохраняя порядок их следования.

Вопросы для самоконтроля

1 Дайте определение массива.
2 Что называется размерностью массива?
3 Что такое индекс массива?
4 Напишите способы объявления и инициализации массивов.

7 Лабораторная работа № 7. Работа со списками и словарями

Цель работы: приобрести навыки работы со словарями в Python.

Теоретические основы

Очередь в Python – это линейный тип структуры данных, используемый для последовательного хранения данных. Ее концепция основана на FIFO (“First in First Out”), что означает: «первым пришел – первый вышел».

Можно выполнять следующие операции с очередью:

– Enqueue – операция, которая добавляет элементы в очередь;

– Dequeue – операция, которая удаляет элемент из очереди (элемент удаляется в том же порядке, в котором был вставлен).

Следующие методы обычно используются для выполнения операции в очереди:

– put(item) – используется для вставки элемента;

– get() – используется для извлечения элемента;

– empty() – функция для проверки, пуста ли очередь или нет (если пуста – она возвращает истину);

– qsize – возвращает длину очереди;

– full() – при условии, если очередь заполнена, возвращает истинное значение; в противном случае – ложное.

Обычный список можно использовать в качестве очереди, но это не очень эффективно с точки зрения производительности. Списки слишком медленные для этой задачи, т. к. вставка и удаление элемента сначала требует сдвига всех прочих элементов по одному.

Список Python можно использовать в качестве очереди FIFO:

```
q = []
q.append('eat')
q.append('sleep')
q.append('code')
print(q)
# ['eat', 'sleep', 'code']
# Осторожнее: медленно работает!
print(q.pop(0)) # 'eat'
```

Словарь (dictionary) похож на список, но имеет более широкие возможности. В списке индекс имеет целочисленное значение, в словаре – может быть любого типа. Вывод словаря на разных компьютерах может дать разный результат, т. к. порядок пар «ключ-значение» не всегда совпадает. Функция len() возвращает число пар «ключ-значение». Оператор in сообщает о похожих ключах. Метод keys() возвращает множество ключей словаря в виде списка, а метод values() возвращает в виде списка множество значений словаря.

Для обхода словаря можно использовать for:

```
eng2sp = {'one': 'uno', 'two': 'dos', 'three': 'tres'}
for element in eng2sp:
print (element)
>>>
two
one
three
```

Порядок выполнения работы

Пусть дана последовательность действительных чисел объема n. Необходимо:

– поменять местами наибольший и наименьший (экстремальные) элементы (если их несколько, то поменять первые из найденных);

– найти сумму и произведение всех положительных элементов.

Код программы с пояснениями:

```
def main():
import os
import sys
import module_for_task_9 as mdl
Lst = []
n = int(input("Input the size of the sequence: "))
```

```
print("Input real elements of the sequence:")
for in range(n):
Lst.append(float(input()))
if sys.platform == 'win32':
os.system('cls')
else:
os.system('clear')
print("Result of task 9:")
print("\toriginal sequence: ", end="\t")
print(Lst)
mdl.change_extreme_elements(lst)
print("\tfinal sequence: ", end="\t")
print(lst)
print("\tmax element: " + str(mdl.get_max_element_of_list(lst)))
print("\tmin element:" + str(mdl.get_min_element_of_list(lst)))
print("\tsum = "+ str(mdl.get_sum_of_positive_elements(lst)))
print("\tmultiplication = " + str(mdl.get_multiplication_of_positive_elements
(Lst)))
main()
input("\n\nPlease, press Enter for exit...")
```

Вопросы для самоконтроля

1 Зачем нужна индексация элементов высокоуровневых типов? Как ее использовать?

2 Как получить доступ к элементам словаря? Можно ли использовать индексацию для словарей?

3 Какие операторы и встроенные функции используются для работы со списками и словарями?

4 Приведите примеры объявления списков и словарей.

8 Лабораторная работа № 8. Функции и модули

Цель работы: изучить процедуры и функции в Python.

Теоретические основы

В контексте программирования функцией (function) называется хранимая последовательность инструкций, предназначенная для решения определенной задачи.

Программу рекомендуется разбивать на функции, поскольку:

– создание новой функции предоставляет возможность присвоить имя группе инструкций, что позволит упростить чтение, понимание и отладку программы;

– функции позволяют сократить код программы, благодаря ликвидации повторяющихся участков кода;

– разбиение длинной программы на функции позволяет одновременно отлаживать отдельные части, а затем собрать их в единое целое;

– хорошо спроектированная функция может использоваться в других программах.

Основными параметрами функции являются:

– имя функции;

– тело функции;

– передаваемые параметры (аргументы);

– возвращаемые параметры (результат).

В Python предоставляется возможность создавать свои собственные (пользовательские) функции, а также работать со встроенными стандартными функциями (функции ввода/вывода данных, функции преобразования типов, математические функции модуля math, функции генерации псевдослучайных функций модуля random).

Правила наименования функций такие же, как для переменных, например, нельзя использовать зарезервированные слова в качестве имен функций. Имена функций и переменных не должны совпадать.

Первая строка определения функции называется заголовком (header), оставшаяся часть – телом (body) функции. Заголовок заканчивается двоеточием, тело функции имеет отступ. Тело функции может содержать любое количество инструкций. Инструкции внутри функции не получат управления, пока функция не будет вызвана. Для возврата результата функции, используется инструкция return. Вызвать функцию (function call) можно, обратившись к ней по имени.

Например, ниже представлена функция addtwo(), имеющая два аргумента и обращение к ней. Указанная функция складывает два числа и возвращает результат. Для вызова этой функции ей нужно передать два параметра. Возвращаемый параметр функции – переменная added, в которую записывается сумма аргументов:

```
def addtwo(a, b):
added = a + b
return added
sum-addtwo (5, 3) #function call
```

Пустые скобки после имени функции указывают на то, что функция не требует аргументов, например, функция random(). Если попытаться присвоить результат выполнения такой функции переменной, вернется специальное значение, называемое None.

Функции, отвечающие за решение задач, принадлежащих к одной области, объединяют в модули, а модули – в пакеты. Модули могут быть как стандартными (математический модуль math, модуль случайных чисел random), так и созданными пользователем. В обоих случаях работа с функциями таких моду-

лей организуется по одинаковым принципам: модуль (или функции модуля) необходимо импортировать.

Примеры импортирования, вызова функций модуля math с присвоением псевдонимов:

```
#first variant
from math import sqrt, sin
y = 25
x=sqrt(y)
z = sin(y)
#second variant
from math import log as ln
y = 25
x = ln(y)
#third variant
import math
y = 9
x=math.sin(9)
#forth variant
import math as m
x=m.sin(7)
```

Порядок выполнения работы

1 Определить, являются ли три треугольника равновеликими. Длины сторон вводить с клавиатуры. Для подсчета площади треугольника использовать формулу Герона. Вычисление площади оформить в виде функции с тремя параметрами.

Пример решения задачи:

```
import math
def s(x, y, z):
p=(x+y+z)/2
s=math.sqrt(p* (p-x)*(p-y) (p-z))
return s
A=[]
for i in range (3):
print('Ваедите стороны ',i, '-го треугольника:')
a=int(input('a:'))
b=int(input('b:'))
c=int(input('c:'))
A.append(s(a,b,c))
for i in range (3):
print ('Площадь ',i, '-го треугольника {:.2f)'.format(A[i]))
if A[0]==A[1]:
if A[0]=A[2]:
print('Треугольники равновеликие)
```

```
else: print('Треугольники не равновеликие')
>>>
Введите стороны 0-го треугольника:
a:3
b:4
c:5
Введите стороны 1-го треугольника:
a:6
b:7
c:8
Введите стороны 2-го треугольника:
a:9
b:10
c:11
Площадь 0-го треугольника 6.00
Площадь 1-го треугольника 20.33
Площадь 2-го треугольника 42.43
Треугольники не равновеликие
```

2 Ввести одномерный массив A длиной m. Поменять в нем местами первый и последний элементы. Длину массива и его элементы ввести с клавиатуры. В программе описать процедуру для замены элементов массива. Вывести исходные и полученные массивы.

Пример решения задачи:

```
def zam (X):
tmp=X [0]
X[0]=X [len (X)-1]
X[len (X)-1]=tmp
A= []
m=int(input('Введите длину массива:'))
for i in range(m):
print('Введите ',i,'элемент массива')
A.append(int(input()))
print (A)
zam (A)
print (A)
>>>
Введите длину массива: 5
Введите
0 элемент массива
0
Введите
1 элемент массива
1
Введите
2 элемент массива
```

2
Введите
3 элемент массива
3
Введите
4 элемент массива
4
[0, 1, 2, 3, 4]
[4, 1, 2, 3, 0]

3 Необходимо разработать программу, которая вычисляет факториал числа N с использованием двух способов: итеративного и рекурсивного.

Вычисление факториала числа N рекурсивным методом:

```
def factorial (n):
if n==0:
return 1
return n factorial (n-1)
def main():
print("<< The program calculates and prints factorial of number >>\n")
nint (input("Input the number: "))
print(str(n)+"! = + str(factorial(n)))
main()
```

Вычисление факториала числа N итерационным методом:

```
def factorial (n):
p = 1
for i in range (1, n + 1):
p *= 1
return p
def main(): print("<<The program calculates and prints factorial of number >>\n")
nint (input("Input the number: "))
print(str(n)+"! = "+ str(factorial(n)))
main()
```

4 Модернизировать программы и оптимизировать алгоритмы решения приведенных выше задач 1–3, группируя основные функции в отдельных модулях.

Вопросы для самоконтроля

1 Что такое функция? Как описывается функция в Python?
2 Для чего используется оператор return в функциях?
3 Чем отличается глобальная переменная от локальной?
4 Что такое модуль?
5 Как создавать и использовать модули?
6 Для чего нужен оператор import?

Список литературы

1 **Гагарина, Л. Г.** Технология разработки программного обеспечения : учебное пособие / Л. Г. Гагарина, Е. В. Кокорева, Б. Д. Сидорова-Виснадул ; под ред. Л. Г. Гагариной. – Москва : ФОРУМ ; ИНФРА-М, 2019. – 400 с.

2 **Доусон, М.** Программируем на Python / М. Доусон. – Санкт-Петербург : Питер, 2014. – 416 с.

3 **Иванченко, В. В.** Языки программирования. Python : учебно-методическое пособие для студентов специальности 1-40 01 01 «Программное обеспечение информационных технологий» : в 2 ч. / В. В. Иванченко [и др.]. – Минск: БНТУ, 2021. – Ч. 1. – 91 с.