РОСЖЕЛДОР
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Ростовский государственный университет путей сообщения»
(ФГБОУ ВО РГУПС)

Загика К.Н., Щербакова К.С.

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ДЛЯ ЛАБОРАТОРНЫХ РАБОТ И САМОСТОЯТЕЛЬНОЙ РАБОТЫ СТУДЕНТОВ ПО ДИСЦИПЛИНЕ

ОП.03 «Основы алгоритмизации и программирования»

для специальности
09.02.09 Веб-разработка

Ростов-на-Дону
2025

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Методические указания к лабораторным работам и по выполнению самостоятельной работы студентов составлены в соответствии с ФГОС СПО и рабочей программой общепрофессионального модуля ОП.03 «Основы алгоритмизации и программирования» которые являются частью программы подготовки специалистов среднего звена специальности 09.02.09 Веб-разработка.

Рабочей программой дисциплины ОП.03 «Основы алгоритмизации и программирования» предусмотрено на выполнение лабораторных работ – 72 часа и самостоятельной работы студентов – 12 часов.

При выполнении лабораторных работ и при организации самостоятельной работы студентов используются активные и интерактивные формы обучения - просмотр и обсуждение учебных видеофильмов, групповая дискуссия, лекция - консультация, моделирование производственных процессов и ситуаций, обсуждение в группах, тренинг, кейс-метод, защита практических и лабораторных работ и другие.

Цель методических рекомендаций - оказание методической помощи студентам в выполнении лабораторных работ и в организации их самостоятельной работы по изучению учебного материала, для расширения, углубления и закрепления знаний и умений, а также формирования профессиональных (ПК) компетенций.

Код и содержание компетенции	**Уметь**	**Знать**
ОК 01. Выбирать способы решения задач профессиональной деятельности применительно к различным контекстам;	Анализировать постановку вычислительной задачи и определять наиболее подходящий тип алгоритма; Выбирать оптимальные структуры данных для хранения и обработки информации в зависимости от условий конкретной задачи; Сравнивать эффективность различных алгоритмов решения одной задачи и обосновывать выбор; Выбирать парадигму программирования в зависимости от сложности и масштаба разрабатываемой программы.	Классификации алгоритмов и их свойств; Базовых принципов оценки сложности алгоритмов; Особенностей и областей применения различных парадигм программирования.

ОК 02. Использовать современные средства поиска, анализа и интерпретации информации и информационные технологии для выполнения задач профессиональной деятельности;	Использовать официальную документацию Python, справочные системы IDE и ресурсы для поиска информации о синтаксисе, стандартных библиотеках и методах решения типовых проблем программирования; Применять интегрированные среды разработки со всеми их инструментами для повышения эффективности разработки; Анализировать и интерпретировать сообщения об ошибках для локализации и устранения дефектов в коде.	Структуры и правил работы с официальной технической документацией по языку программирования; Основных возможностей современных сред разработки для python; Базовых принципов визуализации данных в контексте программной разработки.
ОК 09. Пользоваться профессиональной документацией на государственном и иностранном языках.	Читать и понимать фрагменты технических заданий, описаний алгоритмов и спецификаций на русском языке для корректной реализации программы; Использовать англоязычную документацию к стандартным библиотекам python; Понимать ключевую профессиональную лексику на английском языке, сообщениях об ошибках, названиях функций и методов.	Базовой профессиональной терминологии в области алгоритмизации и программирования на русском и английском языках; Структуры и условных обозначений, принятых в технической документации по программированию.

ПК 1.1. Проектировать информационные ресурсы.	Анализировать постановку задачи и выделять основные сущности и процессы предметной области; Разрабатывать алгоритмы решения типовых вычислительных задач с использованием основных алгоритмических конструкций; Выбирать оптимальные структуры данных для хранения и обработки информации в соответствии с условиями задачи; Проектировать структуру программы, выделяя логические блоки и определяя взаимосвязи между ними.	Основных свойств алгоритмов (дискретность, понятность, определенность, результативность, массовость) и способов их описания; Основных алгоритмических конструкций и принципов структурного программирования; Характеристик, областей применения и ограничений базовых структур данных языка python; Этапов жизненного цикла программного обеспечения.
ПК 1.2. Разрабатывать интерфейсы пользователя.	Организовывать консольный ввод данных от пользователя с использованием функции input() и преобразованием типов; Форматировать вывод результатов работы программы для удобства восприятия; Реализовывать простое интерактивное консольное меню выбора действий; Обеспечивать «дружественность» интерфейса путем обработки некорректного ввода пользователя с использованием механизма исключений;	Принципов организации ввода-вывода в консольных приложениях; Методов форматирования строк в python; Базовых принципов создания удобного интерфейса и обработки ошибок ввода.

ПК 1.3. Интегрировать программный код в соответствующую инфраструктуру.	Создавать пользовательские функции и классы для организации многофайлового проекта; Импортировать и использовать собственные и стандартные модули python; Интегрировать отдельные программные модули в единую исполняемую программу; Организовывать чтение исходных данных из внешних файлов и запись итоговых результатов в файлы; Использовать виртуальные окружения для управления зависимостями проекта.	Принципов модульности и повторного использования кода; Синтаксиса и семантики оператора импорта модулей в python; Основных форматов файлов для обмена данными и способов работы с ними; Назначения и основ использования виртуальных окружений.
ПК 1.4. Использовать систему контроля версий в процессе коллективной (параллельной) разработки.	Создавать локальный и удаленный репозиторий для учебного программного проекта; Выполнять основные операции с репозиторием; Организовывать работу с удаленным репозиторием; Выполнять слияние веток и разрешать простые конфликты слияния; Анализировать историю коммитов.	Назначения, преимуществ и базовых принципов работы распределенных систем контроля версий; Инициировать репозиторий для учебного проекта и подключать удаленный репозиторий; Корректно фиксировать изменения в коде с информативными комментариями; Создавать отдельные ветки для разработки новых функций и сливать их с основной веткой.
ПК 1.5. Выполнять процедуры тестирования программного кода.	Идентифицировать типы ошибок в программе; Применять основные методы отладки; Разрабатывать и применять простые тестовые наборы данных; Проводить ручное функциональное тестирование программы на соответствие техническому заданию.	Классификации ошибок программирования; Основных стратегий и методов тестирования программного обеспечения (отладка, модульное тестирование); Основных типов исключений в python и принципов их обработки.

2. ПЛАН РАСПРЕДЕЛЕНИЯ УЧЕБНОЙ НАГРУЗКИ

Объем дисциплины в академических часах с указанием количества академических часов, выделенных на контактную работу обучающихся с преподавателем (по видам учебных занятий) и на самостоятельную работу обучающихся.

Вид учебной работы	**Всего часов**	**Число часов в семестре**	
		3	**4**
Объем образовательной программы учебной дисциплины	**144**	**72**	**72**
в том числе:			
Лекции (теоретическое обучение)	52	32	20
Практические занятия			
Лабораторные работы	72	32	40
Самостоятельная работа	12	6	6
Промежуточная аттестация (в форме зачета в 3 семестре и экзамена в 4 семестре)	**8**	**2**	**6**

3. ОТВЕДЕННОЕ КОЛИЧЕСТВО ЧАСОВ ПО ВИДАМ УЧЕБНЫХ ЗАНЯТИЙ И РАБОТЫ

Лекционные занятия. Семестр № 3.

№ п/п	**Наименование лекционных занятий**	**Трудоемкость аудиторной работы, часы**
	Раздел № 1	
1	Введение. Понятие алгоритма и его свойства. Типы алгоритмов. Способы описания алгоритмов	2
2	Схемы алгоритмов. Основные алгоритмические конструкции.	2
3	Введение. Элементы технологии программирования.	2
4	Понятие структурного, модульного, объектно-ориентированного программирования.	2
	Раздел № 2	
5	Идентификаторы. Ключевые слова и имена. Символы операций и разделители. Литералы	2
6	Типы данных и объявления переменных.	2
7	Операции и выражения. Операторы присваивания. Операторы ввода-вывода.	2
8	Организация ветвлений и циклов. Составные и пустые операторы.	2
9	Условные операторы. Оператор-переключатель.	2
10	Организация циклических вычислений. Операторы цикла. Вложенные циклы. Операторы перехода и возврата.	2

№ п/п	Наименование лекционных занятий	Трудоемкость аудиторной работы, часы
11	Массивы как структурированный тип данных. Объявление массивов. Ввод-вывод одномерных массивов. Обработка одномерных массивов.	2
12	Двумерные массивы. Ввод-вывод двумерных массивов. Обработка двумерных массивов	2
13	Списки, кортежи, множества, словари. Создание списков, кортежей, множеств и словарей операции над списками и кортежами.	2
14	Строки. Объявление строковых типов данных. Стандартные функции для работы со строками.	2
15	Поиск, удаление, замена символа в строке	2
16	Регулярные выражения	2

Лекционные занятия. Семестр № 4.

№ п/п	Наименование лекционных занятий	Трудоемкость аудиторной работы, часы
	Раздел № 2	
1	Пользовательские типы данных. Действия над пользовательскими типами данных.	2
2	Понятие функции, их сущность и назначение. Организация функций.	2
3	Функции, определенные пользователем, передача аргументов. Декораторы.	2
4	Рекурсия. Примеры классических рекурсивных задач.	2
5	Типы файлов. Открытие и закрытие файла. Запись в файл, чтение данных из файла. Функции работы с файлами.	2
6	Стеки и очереди. Программирование алгоритмов с использованием стеков и очередей.	2
7	Двунаправленные списки. Программирование алгоритмов с использованием двунаправленных списков.	2
8	Основы объектно-ориентированного программирования: объекты, классы и инкапсуляция.	2
9	Полиморфизм и наследование: расширение и специализация поведения классов	2
10	Деревья. Программирование алгоритмов с использованием деревьев.	2

Лабораторный практикум. Семестр 3.

№ п/п	Наименование лабораторных работ	Трудоемкость аудиторной работы, часы

№ п/п	Наименование лабораторных работ	Трудоемкость аудиторной работы, часы
	Раздел № 1	
1	Лабораторная работа № 1. Разработка и анализ простых алгоритмов с использованием словесного описания и пошаговой детализации. Исследование свойств алгоритмов (детерминированность, конечность, массовость и др.) на примере конкретных задач.	2
2	Лабораторная работа № 2. Разработка алгоритмов линейной, разветвляющейся, циклической и рекурсивной структуры. Сравнение различных типов алгоритмов (линейный, разветвляющийся, циклический, рекурсивный) на основе решения одной задачи разными способами.	4
3	Лабораторная работа № 3. Установка интерпретатора Python. Знакомство со штатной средой разработки IDLE.	2
4	Лабораторная работа № 4. Сравнительный анализ: решение одной и той же задачи средствами структурного, модульного и объектно-ориентированного программирования.	2
	Раздел № 2	
5	Лабораторная работа № 5. Правила именования и использование идентификаторов. Работа с литералами и основными символами языка.	2
6	Лабораторная работа № 6. Правила объявления и использование переменных с различными типами данных.	2
7	Лабораторная работа № 7. Вычисления с использованием арифметических, логических и сравнительных операций. Ввод данных, присваивание и форматированный вывод.	2
8	Лабораторная работа № 8. Разработка программ линейной структуры.	2
9	Лабораторная работа № 9. Разработка программ разветвляющей структуры.	2
10	Лабораторная работа № 10. Разработка программ циклической структуры.	2
11	Лабораторная работа № 11. Разработка программ с использованием одномерных массивов.	2
12	Лабораторная работа № 12. Разработка программ с использованием двумерных массивов.	2
13	Лабораторная работа № 13. Разработка программ с использованием списков, кортежей, множеств и словарей.	2
14	Лабораторная работа № 14. Разработка программ с использованием стандартных функций для работы со строками.	2
15	Лабораторная работа № 15. Разработка программ с использованием регулярных выражений для работы со строками.	2

Лабораторный практикум. Семестр 4.

№ п/п	Наименование лабораторных работ	Трудоемкость аудиторной работы, часы
	Раздел № 2	
1	Лабораторная работа № 16. Разработка программ с использованием пользовательских типов данных, действия над пользовательскими типами данных.	4
2	Лабораторная работа № 17. Разделение программы на функции: выделение логических блоков и оформление их как отдельных функций.	2
3	Лабораторная работа № 18. Разработка функций с передачей аргументов разных типов, с использованием изменяемых и неизменяемых аргументов.	4
4	Лабораторная работа № 19. Разработка функций с передачей произвольного числа аргументов.	2
5	Лабораторная работа № 20. Разработка функций с реализацией классических рекурсивных алгоритмов: факториал, числа Фибоначчи, НОД (алгоритм Евклида).	2
6	Лабораторная работа № 21. Разработка программ с использованием рекурсивной обработки структур данных: обход вложенных списков или словарей, вычисление суммы/глубины/количества элементов.	2
7	Лабораторная работа № 22. Создание текстового и двоичного файла. Чтение из файла. Изменение данных в файле	2
8	Лабораторная работа № 23. Создание форматированного файла (CSV, JSON). Чтение из файла. Изменение данных в файле.	2
9	Лабораторная работа № 24. Разработка программ с использованием однонаправленных списков типа стек и очередь.	4
10	Лабораторная работа № 25. Разработка программ с использованием двусвязных списков.	4
11	Лабораторная работа № 26. Создание классов и объектов. Инкапсуляция и управление доступом к данным.	4
12	Лабораторная работа № 27. Разработка программы с реализацией иерархии классов с использованием наследования.	4
13	Лабораторная работа № 28. Разработка программы с применением полиморфизма и переопределением методов.	2
14	Лабораторная работа № 29. Разработка программ с использованием деревьев и бинарных деревьев.	4

Самостоятельное изучение учебного материала.

Номер раздела данной дисциплины	Наименование тем, вопросов, вынесенных для самостоятельного изучения	Трудоемкость внеаудиторной работы, часы
Семестр № 2		

Номер раздела данной дисциплины	Наименование тем, вопросов, вынесенных для самостоятельного изучения	Трудоемкость внеаудиторной работы, часы
1	Работа с внешними библиотеками: установка и использование. Установка пакетов через `pip`, виртуальные окружения (`venv`), примеры использования популярных библиотек (например, `requests` для HTTP-запросов или `tabulate` для красивого вывода таблиц).	2
2	Работа с числами: точные вычисления и специальные типы. Использование модулей `decimal` и `fractions` для точной арифметики, сравнение с `float`, применение в финансовых или научных задачах, где важна точность.	2
Семестр № 4		
2	Работа с датой и временем в Python. Использование модулей `datetime`, `time`, `calendar`: получение текущей даты/времени, форматирование, вычисление разницы между датами, работа с часовыми поясами.	2
2	Обработка исключений и отладка программ. Использование конструкций `try`, `except`, `else`, `finally`; создание пользовательских исключений; применение отладчика (например, в IDLE или VS Code); логирование с помощью модуля `logging`.	2
2	Тестирование программ: написание unit-тестов. Использование модуля `unittest` или `pytest` для проверки корректности функций, написание тест-кейсов, проверка граничных значений и ошибочных входных данных.	2
2	Создание документации и аннотаций типов. Написание docstring в стиле Google или NumPy, использование type hints (`typing`), проверка типов с помощью `mypy`, улучшение читаемости и поддержки кода.	2

Лабораторная работа №1. Разработка и анализ простых алгоритмов с использованием словесного описания и пошаговой детализации. Исследование свойств алгоритмов (детерминированность, конечность, массовость и др.) на примере конкретных задач.

Цель работы.

Освоить принципы разработки алгоритмов с использованием словесного описания и пошаговой детализации. Изучить и экспериментально проверить основные свойства алгоритмов: детерминированность, конечность, массовость, понятность, результативность. Научиться анализировать корректность алгоритмов на примере конкретных задач.

Краткие теоретические сведения.

Алгоритм – это четкая последовательность действий, выполнение которой дает какой-то заранее известный результат.

Свойством алгоритма является дискретность, детерминированность, результативность, массовость и понятность и конечность. Ниже в таблице 1 представлен разбор свойств на сущности и пример их нарушения.

Таблица 1 – Свойства алгоритмов

Свойство	**Сущность**	**Пример нарушения**
Дискретность	Разбиение на последовательные шаги.	«Объяснить, как приготовить пасту» – процесс приготовления не разбит на четкие шаги
Детерминированность	Однозначность каждого шага, отсутствие разночтений.	«Выбрать любое число» — не определено какое
Конечность	Завершение работы за конечное число шагов.	Бесконечный цикл без условия выхода
Массовость	Способность решать не одну частную задачу, а целый класс.	Алгоритм только для конкретных чисел 5 и 3
Понятность	Использование только известных команд.	Использование неизвестных команд
Результативность	Получение конечного результата.	Алгоритм, который ничего не вычисляет

Пример выполнения задания

Пример 1. Вычисление площади и периметра прямоугольника
Шаг 1:
1. Словесное описание алгоритма:
1. Начало.
2. Получить значение длины прямоугольника (a).

3. Получить значение ширины прямоугольника (b).
4. Вычислить периметр по формуле: P = (a + b) * 2.
5. Вычислить площадь по формуле: S = a * b.
6. Вывести на экран значения периметра P и площади S.
7. Конец.
8. Шаг 2:

Анализ и разбор алгоритма по свойствам:

1. ДИСКРЕТНОСТЬ (Разбиение на шаги)
Что проверяем: Разделен ли алгоритм на отдельные, четкие шаги?
Вопрос для проверки:
Можно ли выделить отдельные действия?
Анализ алгоритма:
Шаг 1: Начало – отдельный шаг
Шаг 2: Получить длину – отдельный шаг
Шаг 3: Получить ширину – отдельный шаг
Шаг 4: Вычислить периметр – отдельный шаг
Шаг 5: Вычислить площадь – отдельный шаг
Шаг 6: Вывести результат – отдельный шаг
Шаг 7: Конец – отдельный шаг
Свойство выполнено: алгоритм четко разбит на 7 отдельных шагов.

2. ДЕТЕРМИНИРОВАННОСТЬ (Определенность)
Что проверяем: Точно ли определен каждый шаг?
Вопрос для проверки:
Все ли действия описаны точно?
В данном случае неизвестны строки 2, 3 и 6. Разберем подробно:
Шаг 2: «Получить значение длины» – НЕ ОПРЕДЕЛЕНО:
Откуда получить (с клавиатуры, из файла, случайно)? В каких единицах (метры, сантиметры)? Какой диапазон допустим?
Шаг 3: «Получить ширину» – НЕ ОПРЕДЕЛЕНО
Откуда получить (с клавиатуры, из файла, случайно)? В каких единицах (метры, сантиметры)?
Шаг 6: «Вывести на экран» – НЕ ОПРЕДЕЛЕНО:
В каком формате выводить? С каким текстом?
Следовательно свойство детерминированности нарушено.

3. КОНЕЧНОСТЬ
Что проверяем: Закончится ли алгоритм?
Вопрос для проверки:
Есть ли в алгоритме циклы или повторения?
Анализ алгоритма:
Шаг 1: Выполняется 1 раз
Шаг 2: Выполняется 1 раз
Шаг 3: Выполняется 1 раз
Шаг 4: Выполняется 1 раз
Шаг 5: Выполняется 1 раз
Шаг 6: Выполняется 1 раз
Шаг 7: Выполняется 1 раз
Свойство выполнено: Алгоритм выполнит ровно 7 шагов и завершится.

4. МАССОВОСТЬ
Что проверяем: Подойдет ли алгоритм для многих задач?
Вопрос для проверки:
Работает ли для разных входных данных?
Анализ алгоритма:
Работает для ЛЮБОЙ длины: a = 1, a = 100, a = 3.14
Работает для ЛЮБОЙ ширины: b = 2, b = 50, b = 7.5
Подходит для любого прямоугольника
Свойство выполнено: Алгоритм работает для всех прямоугольников.

5. ПОНЯТНОСТЬ
Что проверяем: Понятны ли все команды?
Вопрос для проверки:
Все ли слова и команды известны?
Анализ алгоритма:
«Начало» – понятно; «Получить значение» – понятно; «Вычислить по формуле» – понятно ; «Вывести на экран» – понятно; «Конец» – понятно; Формулы: P = (a + b) * 2 – математическая формула; Формулы: S = a * b – математическая формула.
Свойство выполнено: Все команды понятны.

6. РЕЗУЛЬТАТИВНОСТЬ
Что проверяем: Приводит ли алгоритм к полезному результату?
Вопрос для проверки:
Есть ли в алгоритме полезные вычисления?
Анализ алгоритма:
Шаг 4: Вычисляет периметр – полезный результат
Шаг 5: Вычисляет площадь – полезный результат
Шаг 6: Показывает результаты – полезное действие
Свойство выполнено: Алгоритм вычисляет и показывает нужные значения.
Шаг 3: Занесем данные в таблицу 2.
Таблица 2 – Сводная таблица анализа

Свойство	**Выполнено?**	**Комментарий**	**Как проверить (вопрос)**
Дискретность	Да	7 четких шагов	"Можно ли выделить отдельные действия?"
Детерминированность	Нет	Не сказано откуда брать данные	"Точно ли описано, как получить данные?"
Конечность	Да	Всего 7 шагов, нет циклов	"Закончится ли когда-нибудь?"
Массовость	Да	Для любого прямоугольника	"Подойдет ли для других размеров?"
Понятность	Да	Все слова известны	"Понятен ли алгоритм?"

Свойство	Выполнено?	Комментарий	Как проверить (вопрос)
Результативность	Да	Вычисляет нужные значения	"Будет ли полезный результат?"

Итог: 5 из 6 свойств выполнены. Нужно исправить **детерминированность.**

Исправленный алгоритм:

1. Начало
2. Ввести с клавиатуры значение длины прямоугольника в сантиметрах.
   Сохранить в переменную a.
3. Ввести с клавиатуры значение ширины прямоугольника в сантиметрах.
   Сохранить в переменную b.
4. Проверить: если $a \leq 0$ или $b \leq 0$, вывести «Ошибка: размеры должны быть положительными».
5. Вычислить периметр по формуле: $P = (a + b) \times 2$.
6. Вычислить площадь по формуле: $S = a \times b$.
7. Вывести на экран: «Периметр прямоугольника:» P «см» и «Площадь прямоугольника:» S «см²».
8. Конец.

Задания к лабораторной работе.

Задание 1.
Вариант задания назначит преподаватель.

Вариант 1: Решение квадратного уравнения $ax^2 + bx + c = 0$.
Вариант 2: Нахождение НОД (наибольшего общего делителя) двух чисел.
Вариант 3: Определение принадлежности точки кругу.
Вариант 4: Вычисление площади треугольника по трем сторонам.
Вариант 5: Определение правильности хода шахматной фигуры.
Вариант 6: Определение победителя в игре «Камень-Ножницы-Бумага»ю
Вариант 7: Определить, кратно ли число 3.
Вариант 8: Определить, какое из трех чисел наибольшее.
Вариант 9: Проверить, одинаковые ли два числа.
Вариант 10: Проверить, меньше ли число 0.
Вариант 11: Найти сумму двух чисел.
Вариант 12: Проверить, делится ли число на 10.
Вариант 13: Приготовить чай.
Вариант 14: Отправить письмо по электронной почте.
Вариант 15: Заказать такси.
Вариант 16: Записаться на курсы.

Задание 2.
Вариант задания назначит преподаватель.

Вариант 1. Вычисление стоимости покупки
Вариант 2. Перевод градусов Цельсия в Фаренгейты
Вариант 3. Проверка четности числа
Вариант 4. Нахождение большего из двух чисел
Вариант 5. Расчет времени пути

Вариант 6. Конвертация валюты
Вариант 7. Расчет среднего балла
Вариант 8. Определение високосного года
Вариант 9. Расчет скидки
Вариант 10. Проверка пароля
Вариант 11. Расчет чаевых
Вариант 12. Определение времени суток
Вариант 13. Расчет количества дней в месяце
Вариант 14. Конвертация метров в сантиметры
Вариант 15. Расчет возраста
Вариант 16: Определение, является ли число четным

Контрольные вопросы.

1. Что такое алгоритм?
2. Назови 2 любых свойства алгоритма.
3. В чем разница между свойствами «детерминированность» и «понятность»?
4. Как проявляется свойство детерминированности (определённости) в алгоритме? Приведите пример нарушения этого свойства.
5. В чём заключается свойство конечности алгоритма и почему оно обязательно?
6. Что означает свойство массовости алгоритма и как оно связано с возможностью его применения к разным входным данным?
7. Почему результативность является необходимым свойством любого алгоритма?
8. Можно ли считать алгоритмом инструкцию, содержащую неоднозначные формулировки? Обоснуйте ответ.
9. Как проверить, обладает ли предложенный способ решения задачи свойством дискретности?
10. Приведите пример задачи, для которой можно составить алгоритм, и укажите, какие его свойства при этом выполняются.

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 2. Разработка алгоритмов линейной, разветвляющейся, циклической и рекурсивной структуры. Сравнение различных типов алгоритмов (линейный, разветвляющийся, циклический, рекурсивный) на основе решения одной задачи разными способами.

Цель работы.

Освоить принцип линейного выполнения команд. Понять ограниченность линейных алгоритмов для решения задач, требующих обработки переменного количества данных.

Краткие теоретические сведения.

Линейный алгоритм — это последовательность команд, выполняемых одна за другой в том порядке, в котором они записаны. В таком алгоритме отсутствуют условия и повторения. Пример: вычисление площади прямоугольника по заданным длине и ширине.

Разветвляющийся (ветвящийся) алгоритм содержит условие, в зависимости от которого выбирается один из возможных путей выполнения. Такой алгоритм использует конструкции «если — то — иначе» и позволяет обрабатывать разные ситуации по-разному. Пример: определение, является ли число положительным, отрицательным или нулём.

Циклический алгоритм предполагает многократное повторение одной или нескольких операций до тех пор, пока не будет выполнено заданное условие. Циклы бывают с предусловием, постусловием или с заданным числом повторений. Пример: вычисление суммы первых N натуральных чисел.

Рекурсивный алгоритм — это алгоритм, который вызывает сам себя для решения подзадачи того же типа. Рекурсия обязательно должна содержать условие завершения, чтобы избежать бесконечного вызова. Пример: вычисление факториала числа через вызов функции от меньшего значения.

Эти типы алгоритмов могут комбинироваться. Одну и ту же задачу часто можно решить разными способами — например, вычисление факториала возможно как циклическим, так и рекурсивным методом. Сравнение таких подходов помогает понять их преимущества и недостатки с точки зрения читаемости, эффективности и использования памяти.

Линейный алгоритм. Пример выполнения задания.

Разработайте линейный алгоритм для вычисления суммы чисел от 1 до N для конкретного, заранее известного N. Рассмотрите случай N = 5.

Словесное описание алгоритма (линейный способ):

1. Начало.
2. Задать значение N = 5.
3. Вычислить сумму S по формуле: S = 1 + 2 + 3 + 4 + 5.
4. Вывести результат S.
5. Конец.

Блок-схема линейного алгоритма показана на рисунке 1:

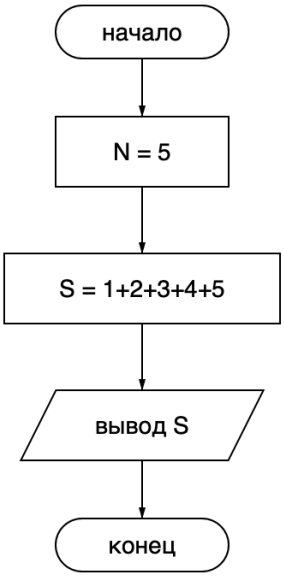

Рисунок 1 – Блок-схема линейного алгоритма

Анализ линейного алгоритма представлен в таблице 1.

Таблица 1 – Анализ линейного алгоритма

Критерий	**Анализ линейного алгоритма**	
Корректность решения	Даёт верный результат для N=5? (Да/Нет)	Да
Гибкость	Можно ли легко изменить N на 6? Что для этого нужно сделать?	Нет, изменить сложно. Чтобы изменить N на 6, нужно полностью переписать алгоритм
Эффективность	Удобно ли писать алгоритм для N=100? Почему?	Неудобно. Для N=100 пришлось бы вручную перечислить все 100 слагаемых в одной строке кода, что нерационально, громоздко и чревато ошибками
Область применения	Для каких задач подходят линейные алгоритмы? (Привести 2 примера)	1. Вычисление площади прямоугольника по известным сторонам: S = a * b 2. Расчет стоимости покупки: Сумма = Цена * Количество

Таким образом, линейный алгоритм прост для понимания, но не является универсальным решением данной задачи. Он не обладает свойством массовости в полной мере: для каждого нового N нужен новый алгоритм. Для решения задачи с переменным N необходимы другие структуры.

Разветляющийся алгоритм. Пример выполнения задания.

Пользователь вводит число N. Если N > 0, вычисляется сумма чисел от 1 до N. Если N <= 0, выводится сообщение об ошибке «Число должно быть положительным!».

Словесное описание алгоритма (разветвляющийся способ):

1. Начало.

2. Ввести число N.
3. Если N > 0, то перейти к шагу 4, иначе перейти к шагу 7.
4. Вычислить сумму S по формуле: S = N * (N + 1) / 2.
5. Вывести результат S.
6. Перейти к шагу 8 (конец).
7. Вывести сообщение: «Число должно быть положительным!».
8. Конец.

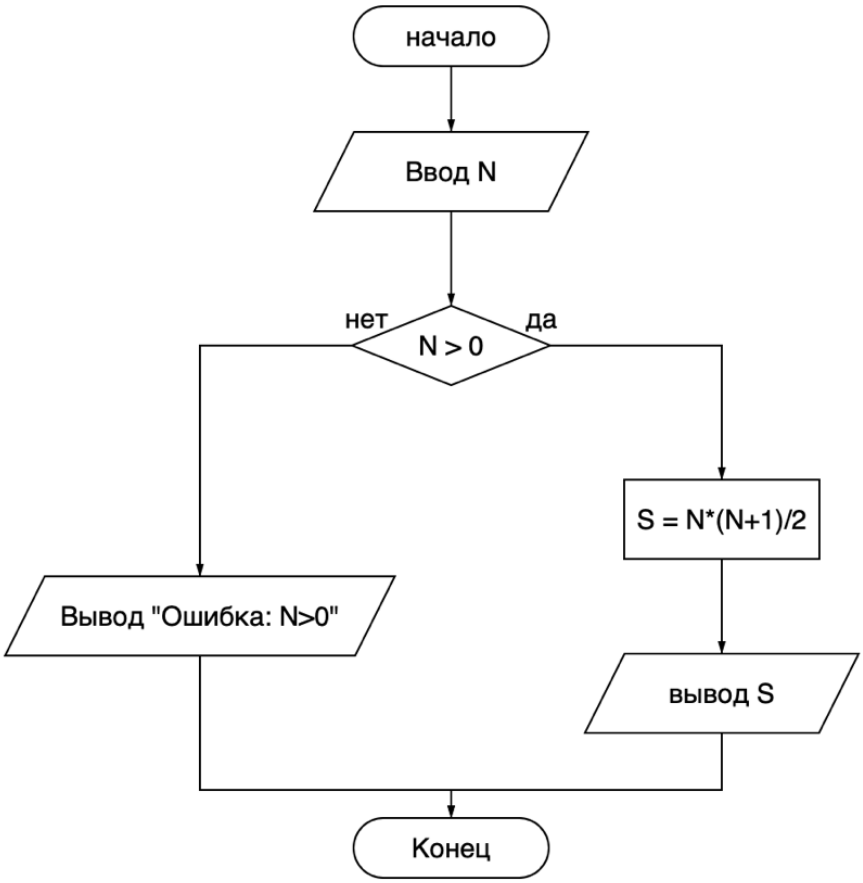

Рисунок 2 – Блок-схема разветвляющегося алгоритма

Анализ разветвляющегося алгоритма представлен в таблице 2.

Таблица 2 – Анализ разветвляющегося алгоритма

Критерий	Анализ разветвляющегося алгоритма	
Корректность решения	Даёт верный результат для N=5? (Да/Нет)	Да

Критерий	Анализ разветвляющегося алгоритма	
Гибкость	Можно ли легко изменить N на 6? Что для этого нужно сделать?	Да. Достаточно ввести новое значение при запуске программы, алгоритм автоматически применит формулу
Эффективность	Удобно ли писать алгоритм для N=100? Почему?	Да, удобно. Для любого N алгоритм выполняет всего 3 арифметические операции (N*(N+1)/2) вместо N операций сложения
Область применения	Для каких задач подходят разветвляющиеся алгоритмы? (Привести 2 примера)	1. Проверка пароля: если введён правильный пароль - доступ разрешён, иначе - доступ запрещён 2. Определение скидки: если сумма покупки >5000 руб, скидка 10%, иначе скидка 5%

Таким образом, разветвляющийся алгоритм делает программу интеллектуальнее, позволяя ей реагировать на разные условия. Он обрабатывает ошибочные ситуации, что критически важно для создания надежных программ. Однако основное действие (вычисление суммы) по-прежнему выполняется по готовой формуле. Для реализации последовательного сложения потребуется цикл.

Циклический алгоритм. Пример выполнения задания.

Пользователь вводит число N. Вычислите сумму всех натуральных положительных чисел от 1 до N.

Словесное описание алгоритма (циклический способ, цикл for):

1. Начало.
2. Ввести число N.
3. Присвоить переменной-счётчику i значение 1.
4. Присвоить переменной суммы S значение 0.
5. Если i > N, перейти к шагу 9 (выход из цикла).
6. Увеличить сумму S на текущее значение i: S = S + i.
7. Увеличить счётчик i на 1.
8. Перейти к шагу 5.
9. Вывести результат S.
10. Конец.

Блок-схема циклического алгоритма представлена на рисунке 1.

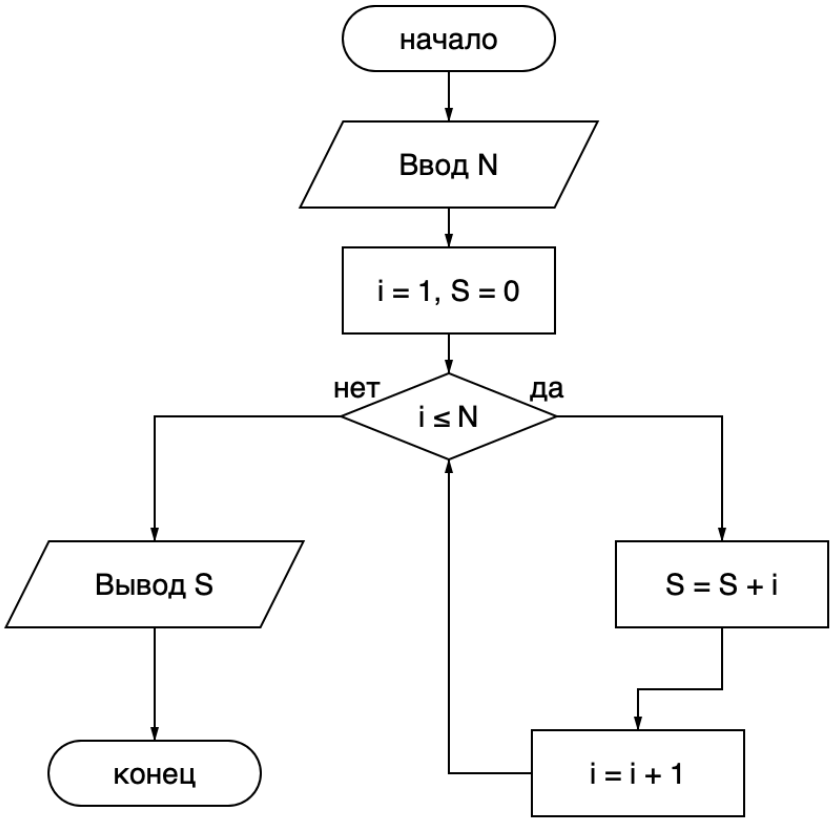

Рисунок 3 – Блок-схема циклического алгоритма

Анализ циклического алгоритма представлен в таблице 3.

Таблица 3 – Анализ циклического алгоритма

Критерий	Анализ циклического алгоритма	
Корректность решения	Даёт верный результат для N=5? (Да/Нет)	Да
Гибкость	Можно ли легко изменить N на 6? Что для этого нужно сделать?	Да. Для изменения N достаточно ввести новое значение, алгоритм автоматически выполнит нужное количество итераций цикла
Эффективность	Удобно ли писать алгоритм для N=100? Почему?	Менее эффективен для больших N. Алгоритм выполняет N операций сложения и N операций инкремента счётчика, что даёт сложность O(N). Для N=100 выполняется ~200 операций

Критерий	Анализ циклического алгоритма	
Область применения	Для каких задач подходят разветвляющиеся алгоритмы? (Привести 2 примера)	1. Последовательная обработка элементов массива (поиск, фильтрация, преобразование) 2. Вычисление факториала числа: n! = 1×2×3×...×n

Таким образом, циклический алгоритм является наиболее наглядной и естественной реализацией процесса последовательного суммирования. Он универсален и легко адаптируется для изменения логики (например, суммирование только чётных чисел). Позволяет решать задачи, где количество повторений заранее известно или зависит от условия.

Рекурсивный алгоритм. Пример выполнения задания.

Разработать алгоритм и реализовать программу для вычисления суммы всех натуральных чисел от 1 до N с использованием рекурсивного подхода, где N – целое положительное число, вводимое пользователем.

Словесное описание алгоритма (рекурсивный способ):

Определение функции Sum(N):

базовый случай: Если N = 1, то Sum = 1;

рекурсивный шаг: Иначе Sum(N) = N + Sum(N-1).

Главный алгоритм:

1. Начало.
2. Ввести число N.
3. Если N < 1, вывести ошибку.
4. Иначе, вызвать функцию Result = Sum(N).
5. Вывести Result.
6. Конец.

Блок-схема рекурсивного алгоритма представлена на рисунке 4.

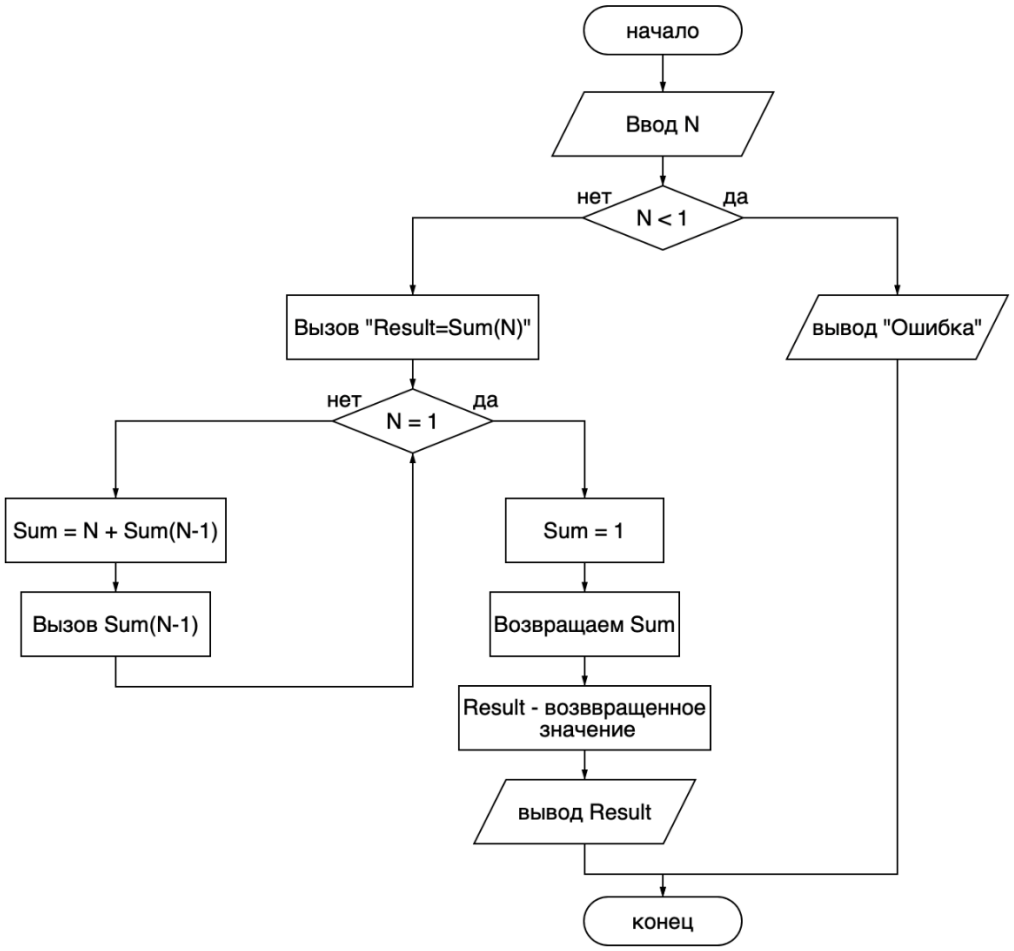

Рисунок 4 –Блок-схема рекурсивного алгоритма

Пошаговая трассировка рекурсивного алгоритма:
1. Начало: «Нужно сложить числа от 1 до 3»
2. Вызывает Sum(3)
3. Sum(3) получает задание: «Сложи числа 1..3», у Sum(3) есть 3 – это не 1
4. Вызывает Sum(2): «Сложи числа 1..2»
5. Sum(2) получает: «Сложи числа 1..2», у Sum(2) есть только 2 – это не 1
6. Вызывает Sum(1): «Сложи числа 1..1»
7.Sum(1) получает: «Сложи числа 1..1», у Sum(1) есть 1 – это «базовый случай»
8. Возвращает: «1»
9. Sum(2): «Получил от Sum(1) ответ 1»
10. Sum(2): «складывает число 2 + полученный 1 = 3»
11. Возвращает: «3»
12. Sum(3): «Получил от Sum(2) ответ 3»
13. Sum(3): «складывает число 3 + полученный 3 = 6»
14. Возвращает: «6»
15. Главная программа: «Получила от Sum(3) ответ 6»
16. Вывод: «Сумма = 6»

Таким образом, одна и та же задача может быть решена принципиально разными алгоритмическими способами. Выбор структуры зависит от требований к ясности, эффективности, универсальности и особенностей самой задачи. Линейная структура проста, но негибка. Разветвление добавляет логику выбора. Цикл – основной инструмент для обработки множеств данных. Рекурсия предлагает элегантное математическое решение, но требует аккуратного применения.

Задания к лабораторной работе.

Задание 1.

Вариант задания назначит преподаватель.

Составьте словесное описание линейного алгоритма, нарисуйте блок-схему и заполните таблицу анализа.

Вариант 1. Вычислить периметр прямоугольника со сторонами 8 см и 5 см.

Вариант 2. Вычислить стоимость покупки 3 тетрадей по 45 рублей.

Вариант 3. Найти среднее арифметическое чисел 12, 18 и 24.

Вариант 4. Вычислить площадь треугольника с основанием 10 см и высотой 6 см.

Вариант 5. Перевести 3 часа 25 минут в минуты.

Вариант 6. Рассчитать общий вес посылки: коробка 2 кг + книга 850 г + диск 120 г.

Вариант 7. Найти сумму, разность, произведение и частное чисел 20 и 4.

Вариант 8. Вычислить объём параллелепипеда с размерами: длина=15 см, ширина=8 см, высота=5 см.

Вариант 9. Перевести температуру 25°C в градусы Фаренгейта по формуле $F = C \times 9/5 + 32$.

Вариант 10. Рассчитать скидку 15% на товар стоимостью 1200 рублей.

Вариант 11. Вычислить площадь круга радиусом 7 см.

Вариант 12. Найти гипотенузу прямоугольного треугольника с катетами 6 см и 8 см.

Вариант 13. Рассчитать площадь квадрата со стороной 9 см.

Вариант 14. Найти сумму чисел 125, 230 и 95.

Задание 2.

Вариант задания назначит преподаватель.

Составьте словесное описание разветвляющегося алгоритма, нарисуйте блок-схему и заполните таблицу анализа.

Вариант 1. Определить, является ли число чётным или нечётным.

Вариант 2. Определить, положительное, отрицательное или нулевое число ввёл пользователь.

Вариант 3. Вычислить значение функции: $y = 2x$, если $x \geq 0$; $y = x^2$, если $x < 0$.

Вариант 4. Определить максимальное из двух чисел.

Вариант 5. Рассчитать скидку: если сумма покупки > 5000 руб, скидка 10%, иначе скидка 5%.

Вариант 6. Определить, попадает ли точка с координатой X в интервал [a, b].

Вариант 7. Вычислить значение функции: если $x \leq 0$, $y = \sin(x)$; если $0 < x \leq 10$, $y = x^2$; если $x > 10$, $y = \sqrt{x}$.

Вариант 8. Определить тип треугольника по углам: остроугольный, прямоугольный, тупоугольный.

Вариант 9. Определить, является ли последовательность чисел возрастающей.

Вариант 10. Определить категорию водителя по стажу: новичок (до 2 лет), опытный (2-10 лет), ветеран (более 10 лет)

Вариант 11. Определить тип фигуры по количеству сторон: 3 - треугольник, 4 - четырёхугольник, 5 - пятиугольник, другое – многоугольник.

Задание 3.

Вариант задания назначит преподаватель.

Составьте словесное описание циклического алгоритма, нарисуйте блок-схему и заполните таблицу анализа.

Вариант 1. Вывести все чётные числа от 2 до 40.

Вариант 2. Вычислить факториал числа N (N! = 1×2×3×...×N).
Вариант 3. Найти сумму цифр заданного числа (N=563).
Вариант 4. Вывести таблицу умножения на 7 (от 1 до 10).
Вариант 5. Вычислить среднее арифметическое N чисел, введённых пользователем.
Вариант 6. Вывести все числа от 1 до N, которые делятся на 3 или на 5.
Вариант 7. Вычислить произведение цифр числа.
Вариант 8. Найти максимальную цифру в числе.
Вариант 9. Найти количество чисел от 1 до N, которые заканчиваются на 7.
Вариант 10. Вывести все двузначные числа, кратные 7.
Вариант 11. Вычислить количество чётных цифр в числе.
Вариант 12. Найти все числа от 1 до N, которые делятся на сумму своих цифр.
Вариант 13. Вычислить произведение всех чётных чисел от 2 до N.
Вариант 14. Вывести все числа от 1 до N в виде: 1, 12, 123, 1234, ...
Вариант 15. Вычислить произведение всех нечётных цифр в числе.
Вариант 16. Вычислить произведение всех простых чисел до N.

Задание 4.

Вариант задания назначит преподаватель.

Составьте словесное описание рекурсивного алгоритма, нарисуйте блок-схему и выполните пошаговую трассировку.

Вариант 1. Вычислить факториал числа N.
Вариант 2. Найти максимальную цифру в числе.
Вариант 3. Вычислить сумму квадратов чисел от 1 до N.
Вариант 4. Найти минимальную цифру в числе.
Вариант 5. Вычислить произведение цифр числа.
Вариант 6. Вывести числа от N до 1 в обратном порядке.
Вариант 7. Вычислить сумму нечётных чисел от 1 до N.
Вариант 8. Найти сумму чётных цифр числа.
Вариант 9. Вычислить среднее арифметическое цифр числа.
Вариант 10. Вычислить сумму чётных чисел от 2 до N.
Вариант 11. Вычислить произведение чётных цифр числа.
Вариант 12. Определить, является ли число счастливым (сумма квадратов цифр в итоге даёт 1).
Вариант 13. Найти количество чисел от 1 до N, взаимно простых с N.
Вариант 14. Вычислить произведение чисел, делящихся на 3, от 1 до N.

Контрольные вопросы.

1. Что такое линейный алгоритм? Приведите пример из повседневной жизни.
2. Какие основные признаки характеризуют линейный алгоритм?
3. Может ли линейный алгоритм содержать условные операторы или циклы? Почему?
4. Приведите пример задачи, которая решается исключительно линейным алгоритмом.
5. В каких случаях недостаточно использовать только линейную структуру для решения задачи?
6. Разветвляющиеся алгоритмы
7. Что определяет выбор ветви выполнения в разветвляющемся алгоритме?
8. Какие виды ветвлений вы знаете (полное, неполное, множественный выбор)?
9. Может ли разветвляющийся алгоритм не содержать блока «иначе»? Приведите пример.

10. Как обеспечить корректность работы разветвляющегося алгоритма при всех возможных входных данных?
11. Что такое тело цикла и как оно связано с условием повторения?
12. В чём разница между циклом с предусловием и циклом с постусловием?
13. Как избежать зацикливания при написании циклического алгоритма?
14. Когда целесообразно использовать цикл с заданным числом повторений, а не с условием?
15. Что такое базовый случай в рекурсивном алгоритме и зачем он нужен?
16. Почему рекурсивный алгоритм может привести к переполнению стека вызовов?
17. Можно ли любой рекурсивный алгоритм преобразовать в итеративный (циклический)?
18. Какие преимущества и недостатки имеет рекурсивный подход по сравнению с циклическим?
19. Что такое блок-схема? Для чего её используют при разработке алгоритмов?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 3. Установка интерпретатора Python. Знакомство со штатной средой разработки IDLE.

Цель работы.

Ознакомиться с процессом установки интерпретатора языка программирования Python и убедиться в его корректной работе. Научиться загружать библиотеки. Научиться пользоваться интерактивным режимом и режимом редактирования программ.

Краткие теоретические сведения.

Python — высокоуровневый язык программирования общего назначения, который выполняется с помощью интерпретатора. Для запуска программ на Python необходимо установить соответствующий интерпретатор на компьютер. Официальная версия доступна на сайте python.org по адресу https://www.python.org/. При загрузке необходимо выбрать версию Python не ниже 3.9. Если установка производится на 32-х разрядную операционную систему то выбирается версия 3.9 как последняя версия с поддержкой 32-х разрядных ОС. Для 64-х разрядных операционных систем рекомендуется выбирать последнюю стабильную версию.

При установке рекомендуется добавлять Python в переменную среды PATH, чтобы обеспечить возможность запуска из командной строки.

Работа с библиотеками Python.

Библиотека — это набор готового кода (функций, классов, модулей), написанного другими разработчиками, который решает определённые задачи: работа с сетью, обработка изображений, математические вычисления, создание веб-приложений и т.д.

Зачем нужны библиотеки:

1. Экономия времени — не нужно писать всё с нуля.
2. Надёжность — популярные библиотеки хорошо протестированы сообществом.
3. Функциональность — дают доступ к сложным возможностям (например, машинное обучение через scikit-learn или tensorflow).
4. Совместимость — обеспечивают единый способ решения типовых задач.

Часть необходимых библиотек входят в стандартную поставку Python, но многие популярные и необходимые библиотеки не входят в стандартную поставку Python. Чтобы использовать встроенную библиотеку достаточно в коде программы указать в коде программы инструкцию `import`. Обычно эта инструкция указывается в начале программы на Python.

```
import math
```

Чтобы использовать внешнюю библиотеку в своём коде, её сначала нужно скачать и установить в систему с помощью менеджера пакетов, например стандартного в поставке pip. В командной строке или в терминале операционной системы необходимо выполнить команду

```
pip install numpy
```

Где `numpy` это имя библиотеки, в данном случае `numpy`. Для успешного выполнения этой команды компьютер должен быть подключен к интернету. Некоторые библиотеки могут потребовать наличия других библиотек. Такое требование называется зависимостью. Менеджер пакетов `pip` умеет разрешать зависимости и при необходимости скачает и установит дополнительные библиотеки. После установки библиотеки с помощью `pip` их

можно использовать в программе с помощью инструкции `import`. Какие библиотеки уже установлены в Python можно узнать, выполнив в командной строке или в терминале операционной системы команду:

```
pip list
```

Работа с IDLE (Integrated Development and Learning Environment).

IDLE — это официальная интегрированная среда разработки для языка Python, поставляемая вместе с интерпретатором. Она предназначена для обучения и написания простых программ.

Основные возможности IDLE:

1. Интерактивный режим (Shell): После запуска IDLE открывается окно Python Shell — это интерактивный режим, где можно вводить команды Python построчно и сразу видеть результат. Полезно для тестирования небольших фрагментов кода.
2. Редактор кода: Через меню File → New File открывается редактор, в котором можно писать и сохранять программы (файлы с расширением .py). Поддерживается подсветка синтаксиса, автоматическое форматирование отступов и базовая навигация.
3. Выполнение программ: Чтобы запустить написанную программу, используйте Run → Run Module (или клавишу F5). При первом запуске файл нужно сохранить. Результат выполнения отображается в окне Shell.
4. Отладка: IDLE предоставляет простой отладчик: можно включить его через Debug → Debugger, установить точки останова (breakpoints) и пошагово выполнять код.
5. Управление проектом: Хотя IDLE не поддерживает полноценные проекты, она позволяет удобно работать с отдельными файлами, быстро переключаться между ними и выполнять модульное тестирование.

Преимущества IDLE:

- Входит в стандартную поставку Python (не требует установки дополнительных программ).

- Простой и понятный интерфейс, идеален для начинающих.

Ограничения IDLE:

- Отсутствие продвинутых функций (автодополнение, управление зависимостями, интеграция с Git и т.д.).

- Не подходит для крупных проектов и командной работы.

Порядок выполнения работы.

1. Перейдите на официальный сайт Python (https://www.python.org/downloads/).
2. Скачайте последнюю стабильную версию для своей операционной системы.
3. Запустите установочный файл. В окне установки обязательно отметьте опцию «Add Python to PATH». Выберите «Install Now».
4. Дождитесь завершения установки.
5. Откройте командную строку (терминал) и введите команду:

   ```
   python --version
   ```

   или

   ```
   python3 –version
   ```

6. Убедитесь, что отображается установленная версия Python.

7. Проверьте, какие библиотеки установлены в системе, выполнив в терминале команду:

```
pip list
```

8. Запустите IDLE через меню ПУСК операционной системы.
9. Выполните в IDLE следующие команды

```
>>>16+94
>>> print(“Hello world”)
```

10. Откройте редактор через меню File → New File. Напишите в окне следующую программу

```
import os

print(45-34)
print(«Привет мир!»)
```

11. Сохраните получившийся код с помощью меню File или горячей клавиши Ctrl+S и затем запустите на выполнение с помощью меню Run или клавишей F5. Если программа выполнилась без ошибок вы увидите следующие строки:

```
11
Привет мир!
```

Если вывод не такой значит есть ошибки. Проверьте правильность написания программы.

Задания к лабораторной работе.

1. **Откройте IDLE и в окне Python Shell (интерактивный режим) попробуйте выполнить простые арифметические действия:**

```
5 + 3
7 * 6
20 / 4
12/0
```

Напишите, что появляется после нажатия Enter.

2. **В том же окне введите команду:**

```
print("Мой первый код на Python!")
```

Нажмите Enter. Что отобразилось? Попробуйте изменить текст внутри кавычек и запустить снова.

3. **Создайте новый файл через меню File → New File. Напишите в нём одну строку:**

```
print("Привет!")
```

Сохраните файл под именем «hello.py», затем запустите его с помощью Run → Run Module (или клавиши F5). Что появилось в окне Shell?

4. **В интерактивном режиме попробуйте написать:**

```
print(Привет)
```

Что будет выведено на экран и, как вы думаете, почему такой результат?

5. **В интерактивном режиме IDLE введите следующие строки по одной и смотрите результат:**

```
name = "Анна"
print(name)
```

Что делает эта программа? Попробуйте заменить "Анна" на своё имя.

6. **В новом файле напишите три строки:**

```
a = 10
b = 3
print(a + b)
```

Сохраните и запустите. Измените числа и проверьте, как меняется результат.

Контрольные вопросы.

1. Что такое IDLE и для чего он используется?
2. В чём разница между интерактивным режимом (Shell) и редактором кода в IDLE?
3. Как запустить написанную программу из файла в IDLE?
4. Зачем нужны кавычки при использовании функции `print("текст")`? Что произойдёт, если их убрать?
5. Как вы думаете, что делает команда `print()` в Python? Приведите пример её использования.
6. Как сохранить файл с программой на Python и какое расширение он должен иметь?
7. Почему при выполнении `print(Привет)` возникает ошибка, а `print("Привет")` — нет?
8. Что произойдет при выполнении команды `print(100/(12-12))`?
9. Как в IDLE можно быстро повторно выполнить команду, которую уже вводили ранее?
10. Как проверить, что программа работает правильно, если она выводит число или текст?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 4. Сравнительный анализ: решение одной и той же задачи средствами структурного, модульного и объектно-ориентированного программирования.

Цель работы.

Изучить и сравнить основные подходы к организации программного кода — структурный, модульный и объектно-ориентированный; реализовать одно и то же решение задачи тремя разными способами; проанализировать различия в читаемости, повторном использовании кода, поддерживаемости и логической структуре программы при каждом подходе.

Краткие теоретические сведения.

Программирование может осуществляться с использованием разных парадигм, каждая из которых определяет принципы организации кода и взаимодействия его частей.

Структурное программирование основывается на последовательном выполнении команд, использовании ветвлений и циклов, а также разбиении программы на функции (процедуры). Основной акцент делается на логическую структуру алгоритма. Данные и функции разделены: функции оперируют над данными, передаваемыми им в качестве параметров.

Модульное программирование — это развитие структурного подхода, при котором программа разбивается на отдельные модули (файлы, функции, модули), каждый из которых решает определённую подзадачу и имеет чётко определённый интерфейс. Модули могут содержать функции и данные, но без объединения их в единые сущности. Это упрощает разработку, тестирование и повторное использование кода.

Объектно-ориентированное программирование (ООП) строится вокруг понятия объекта — единства данных (атрибутов) и методов, которые над этими данными работают. Основные принципы ООП: инкапсуляция (сокрытие внутреннего устройства объекта), наследование (создание новых классов на основе существующих) и полиморфизм (возможность использовать одинаковый интерфейс для разных типов объектов). ООП способствует моделированию реальных сущностей и повышает гибкость и масштабируемость программ.

Решение одной и той же задачи всеми тремя способами позволяет наглядно увидеть, как меняется структура программы, степень связности данных и логики, а также удобство сопровождения и расширения кода.

Например перед нами стоит задача выполнить симуляцию светофора

Вводные условия:

1. Светофор имеет три цвета: красный, жёлтый, зелёный.
2. Каждый цвет горит определённое время (красный - 30 сек, зелёный - 25 сек, жёлтый - 5 сек).
3. Нужно имитировать работу светофора в течение 2 минут.
4. Выводить текущее состояние и оставшееся время для каждого цвета.
5. Возможность сбросить светофор в начальное состояние (красный, 30 сек).

Подход 1: структурное (линейное) программирование

Мышление: «Что происходит в каждую секунду? Распишем по шагам.»

Алгоритмическое описание:

Объявляем переменные:

текущий_цвет = «красный»

осталось_секунд = 30

общее_время_работы = 120 (2 минуты)

прошедшее_время = 0

Создаём главный цикл на 120 секунд:

На каждой секунде:

Уменьшаем осталось_секунд на 1

Увеличиваем прошедшее_время на 1

Выводим: "Сейчас: [цвет]. Осталось: [N] сек"

Если осталось_секунд стало 0:

Если был красный → переключаем на зелёный (25 сек)

Если был зелёный → переключаем на жёлтый (5 сек)

Если был жёлтый → переключаем на красный (30 сек)

Сброс светофора:

Просто присваиваем: текущий_цвет = «красный», осталось_секунд = 30

Проблемы:

1. Вся логика в одной длинной последовательности.

2. Чтобы добавить второй светофор, нужно ДУБЛИРОВАТЬ весь код

3. Если изменить время жёлтого (скажем, на 3 сек), нужно искать ВСЕ места, где есть число 5.

4. Нет чёткого разделения: «данные светофора» и «правила переключения» перемешаны.

Когда подходит: Если нужно написать одноразовый скрипт для демонстрации работы одного светофора.

Подход 2: модульное программирование

Мышление: «Выделим логические части в отдельные функции.»

Структура решения:

1. Определяем структуру данных:

Создаём «запись» или «структуру» Светофор, которая содержит:

текущий_цвет

осталось_секунд

2. Создаём библиотеку функций:

Функции управления состоянием:

создать_светофор() → возвращает новый светофор (красный, 30 сек)

обновить_светофор(сф) → уменьшает время на 1 сек, переключает цвет при необходимости

сбросить_светофор(сф) → устанавливает красный, 30 сек

Функции отображения:

показать_состояние(сф) → выводит цвет и оставшееся время

получить_следующий_цвет(текущий_цвет) → говорит, какой цвет будет следующим

Функции времени:

получить_время_для_цвета(цвет) → возвращает 30, 25 или 5 секунд

Основная программа становится чище:

Начало программы:

```
  светофор1 = создать_светофор()
  светофор2 = создать_светофор()  // Теперь легко создать второй!
```

Для секунды от 1 до 120:
обновить_светофор(светофор1)
обновить_светофор(светофор2) // Обрабатываем оба одинаково
показать_состояние(светофор1)
показать_состояние(светофор2)
Если нужно сбросить:
сбросить_светофор(светофор1)

Преимущества:

1. Логика переключения инкапсулирована в обновить_светофор()
2. Изменить время работы цвета → меняем только получить_время_для_цвета()
3. Можно легко создать несколько светофоров
4. Функции можно тестировать отдельно

Ограничения:

1. Данные (структура Светофор) всё ещё отделены от функций
2. Ничто не мешает программисту напрямую изменить светофор.осталось_секунд = -10 (нарушение логики)
3. Нет естественного способа создать «особый светофор» с другим поведением

Когда подходит: Для системы управления несколькими светофорами, где нужна чёткая организация кода.

Подход 3: Объектно-ориентированное программирование

Мышление: «Светофор – это объект, который знает своё состояние и умеет себя обновлять.»

Шаг 1: Проектируем класс «Светофор»

Класс – это «чертёж» объекта. Он определяет:

Данные (поля, свойства):

текущий_цвет (приватное – доступно только самому светофору)

осталось_секунд (приватное)

Поведение (методы):

Конструктор() – создаёт светофор в начальном состоянии

обновить() – основная логика: уменьшает время, переключает цвет

показать_состояние() – выводит информацию

сбросить() – возвращает в начальное состояние

получить_текущий_цвет() – даёт прочитать цвет (но не изменить напрямую!)

Шаг 2: Как работает объект светофора

Создание объекта:

светофор1 = НОВЫЙ Светофор()

// Внутри объекта автоматически:

// текущий_цвет = "красный"

// осталось_секунд = 30

Работа с объектом:

светофор1.обновить() // Объект САМ решает:

// 1. Уменьшить своё время

// 2. Если время вышло — переключить свой цвет

// 3. Установить новое время для нового цвета

светофор1.показать_состояние() // Объект САМ знает, как показать себя

Инкапсуляция в действии:

Программист НЕ МОЖЕТ сделать светофор1.осталось_секунд = -5

Он МОЖЕТ только светофор1.сбросить() или светофор1.обновить()

Внутренняя логика защищена от неправильного использования

Шаг 3: Расширение системы

Ситуация 1: Нужен «ночной режим» (мигающий жёлтый):

Создаём НОВЫЙ класс НочнойСветофор, который НАСЛЕДУЕТ от Светофор:
Переопределяем метод обновить() – теперь он просто мигает жёлтым
ВСЁ остальное (показать_состояние, сбросить) работает как раньше
В программе:
дневной = Светофор()
ночной = НочнойСветофор()

список = [дневной, ночной] // Можем хранить вместе!

Для каждого в списке:
элемент.обновить() // Каждый обновляется ПО-СВОЕМУ
элемент.показать_состояние()

Ситуация 2: Нужен «пешеходный светофор» (только красный/зелёный):
Создаём класс ПешеходныйСветофор:
Своя логика переключения
Своё время (зелёный 15 сек, красный 45 сек)
Но интерфейс ТОТ ЖЕ: обновить(), показать_состояние(), сбросить()

Ситуация 3: Создаём «Умный перекрёсток»:
Класс Перекрёсток:
Поля: светофор_север, светофор_юг, светофор_запад, светофор_восток

Методы:
синхронизировать() // Управляет всеми светофорами вместе
аварийный_режим() // Включает всем мигающий жёлтый

Выбор подхода для светофора:
1. Демонстрация работы одного светофора на уроке физики → Структурный
2. Программа для моделирования работы 4 светофоров на перекрёстке → Модульный
3. Городская система управления сотнями светофоров с разными режимами, датчиками, удалённым управлением → ООП

Задания к лабораторной работе.

Задание 2.

Вариант задания назначит преподаватель. Каждую задачу нужно решить 3 способами, структурным, модульным и объектно-ориентированным.

1. Разработайте алгоритм для вычисления площади, периметра и проверки, является ли прямоугольник квадратом.
2. Разработайте алгоритм для добавления записи о студенте, вывода всех записей и поиска студента по имени.
3. Разработайте алгоритм выполнения арифметических операций над двумя числами: сложение, вычитание, умножение, деление.
4. Разработайте алгоритм вычисления длины окружности, площади круга и диаметра по заданному радиусу.
5. Разработайте алгоритм хранения и вывода информации о книге: название, автор, год издания.
6. Разработайте алгоритм управления банковским счётом: пополнение, снятие средств и проверка текущего баланса.
7. Разработайте алгоритм работы с точкой на плоскости: ввод координат, вычисление расстояния до начала координат, смещение на заданный вектор.
8. Разработайте алгоритм анализа набора чисел: нахождение суммы, среднего значения, максимального и минимального элементов.
9. Разработайте алгоритм расчёта расхода топлива автомобиля на заданное расстояние по известному пробегу и среднему расходу.

10. Разработайте алгоритм расчёта годового дохода сотрудника по его месячному окладу и дополнительным выплатам.
11. Разработайте алгоритм конвертации температуры между шкалами Цельсия, Фаренгейта и Кельвина.
12. Разработайте алгоритм приведения дроби к несократимому виду и её представления в виде строки «числитель/знаменатель».
13. Разработайте алгоритм установки времени, добавления заданного количества секунд и форматированного вывода времени в виде ЧЧ:ММ:СС.
14. Разработайте алгоритм расчёта общей стоимости товара по его цене и количеству, а также учёта нескольких товаров.
15. Разработайте алгоритм анализа текстовой строки: подсчёт количества символов, слов и проверка, является ли строка палиндромом.

Контрольные вопросы.

1. Какие основные управляющие конструкции используются в структурном программировании?
2. Как в структурном подходе организовано хранение данных о светофоре?
3. Опишите простыми словами алгоритм работы светофора в структурном подходе.
4. Что такое «функция» в контексте модульного программирования?
5. Как организовано хранение данных о светофоре в модульном подходе?
6. Что такое «класс» в ООП?
7. Чем отличается «класс» от «объекта» на примере светофора?
8. Почему поля текущий_цвет и осталось_секунд сделаны приватными?
9. Что такое «состояние объекта» и как оно меняется в течение жизненного цикла объекта?
10. В каком подходе легче всего добавить второй светофор? Обоснуйте
11. В каком подходе безопаснее всего работать (меньше вероятность ошибок)? Почему?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 5. Правила именования и использование идентификаторов. Работа с литералами и основными символами языка.

Цель работы.

Изучить правила именования идентификаторов, освоить использование литералов и основных символов языка Python на практике.

Краткие теоретические сведения.

Идентификаторы — это имена переменных, функций, классов и других объектов.
Правила именования:

1. Идентификаторы могут содержать буквы (A–Z, a–z), цифры (0–9) и символ подчёркивания «_».
2. С версии 3.9 Python начал понимать национальные символы в именах идентификаторов. Настоятельно НЕ РЕКОМЕНДУЕТСЯ использовать в названиях идентификаторов национальные символы. Будьте особенно внимательны с символом «с» который находится на одной клавише в русской и английской раскладке.
3. Идентификаторы должны начинаться с буквы или символа подчеркивания «_». Не должен начинаться с цифры.
4. Не рекомендуется без особой необходимости начинать название идентификатора с символа подчеркивания.
5. Python является регистрозависимым языком. Регистр символов имеет значение: "name" не эквивалентно "Name".
6. Не могут совпадать с ключевыми словами Python `['False', 'None', 'True', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']`
7. По сложившейся практике в программировании на Python переменные и функции называются строчными буквами. Классы начинаются с прописной буквы. Константы задаются всеми прописными буквами.

Литералы — значения, записанные прямо в коде:

1. Целые: 42, -7
2. Вещественные: 3.14, -2.5e3
3. Строковые обозначаются " или ': `"hello"` или `'world'`. Разницы между " и ' нет.
4. Логические: истина – `True, ложь - False`

Основные символы языка:

1. Буквы латинского алфавита, цифры, _.
2. Операторы и разделители: `+, -, *, /, =, ==, !=, <, >, and, or, not, :, ,, ., (), [], {}, #.`
3. Пробельные символы (пробел, \t, \n) используются для разделения токенов и форматирования.

Обратите внимание! В Python операторные скобки или операторные блоки не обозначаются символами {} (как в C, Java и др.) или begin/end (как в Pascal), а задаются отступами. Блок кода начинается с двоеточия : и выделяется отступом (обычно 4 пробела или одно нажатие на клавишу Tab). Будьте внимательны отсутствие отступа или неправильно использованный отступ может привести к сообщению об ошибке или даже изменить логику работы программы.

Порядок выполнения работы.

Запустите интерактивный режим окно Python Shell. Откройте редактор через меню File → New File.

В соответствии с вашим вариантом придумайте наименования для идентификаторов и впишите их в программу.

Например:

```
VUZ = "RGUPS "
name = "Иван "
age = 15
yavka = True
gruppa = "ТМСП-01-001 "
x = 45+12
f = 100/34
m = 343 > 162
print(VUZ, name, age, yavka, gruppa)
print(x, f, m)
#Это комментарий
```

Сохраните и выполните код. Если будут ошибки проверьте правильность написанных строк и исправьте ошибки.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом придумайте наименования идентификаторов присвойте им указанные значения и впишите их в программу. Ваш вариант – последняя цифра в номере зачетки или студенческого билета. Все значения идентификаторов вывести на экран командой `print`.

0. "МГУ ", "Анна ", 19, False, "ИСТ-202 ", 30-7, (30-7)*2.5, (30-7)*2.5<=100
1. "СПбГЭТУ ", "Дмитрий ", 21, True, "РТ-111 ", 82,(82)/4, (82)==(82)/4
2. "ВШЭ ", "Елена ", 17, False, "ЭКН-02-23 ", 100//6, (100//6)+3.7, ((100//6)+3.7)!=20
3. "МФТИ ", "Сергей ", 20, True, "ФИВТ-405 ", 15*4, (15*4)%7, (15*4)%7>0
4. "НИУ ВШЭ ", "Ольга ", 18, False, "БИ-01-24 ", 50+25, (50+25)/3, (50+25)/3>=25.0
5. "ИТМО ", "Максим ", 22, True, "КТ-303 ", 99-44, (99-44)0.5, (99-44)0.5<10
6. "РАНХиГС ", "Артём ", 16, False, "ГМУ-101 ", 12*5, (12*5)//8, (12*5)//8==7
7. "МГТУ ", "Полина ", 19, True, "МА-202 ", 200/10, (200/10)+(200/10), (200/10)+(200/10)!=40
8. "ТУСУР ", "Кирилл ", 23, False, "РК-505 ", 72, 72-10, (72-10)<=40
9. "УрФУ ", "Нина ", 20, True, "ФФ-101 ", 64//8, (64//8)*1.5, (64//8)*1.5>10

Задание 2.

Выберите строку в соответствии с вашим номером варианта и напишите, правильные ли это наименования идентификаторов и если нет, то почему.

```
0. 1class, for, user@name, 2var, valid_name, my-var, $amount, while, _price_ok, 345
1. new-value, try, __init__, 123abc, hello world, count, #tag, return, data_2025, temp$
2. global, name!, _my_var, 42, else, calc_result, user.name, @decor, index_i, 99cents
3. lambda, total-sum, balance, pass, email@domain, good_var, 7up, def, _counter, final result
4. import, speed_kmh, true, %percent, score, break, 1st_place, validInput, user#id, xyz
5. yield, max.value, _temp_ok, class1, with, item-list, valid_check, 2fast, output_file, &ref
6. as, price$usd, correct_var, finally, 3d_model, user name, _data_store, nonlocal, myVar, ==equal
7. from, avg-temp, result_ok, assert, 5star, login_id, email address, or, _valid_flag, !!!
8. and, file.name, counter, except, $money, goodName, 2nd_try, private_data, class, π
9. not, my-variable, _total_sum, elif, 100%, ok_value, user@home, async, calc#1, valid_name_2026
```

Задание 3.

С помощью оператора # создайте комментарий в коде, который вы выполнили в задании 1.

Контрольные вопросы.

1. Какие символы могут входить в идентификатор в Python?
2. Почему имя переменной "`2var`" является недопустимым?
3. Можно ли использовать ключевые слова Python (например, "`if`", "`for`") в качестве имён переменных? Почему?
4. Какой стиль именования рекомендуется для переменных и функций в Python? Приведите пример.
5. Что такое литерал? Приведите примеры литералов разных типов.
6. Чем отличаются строковые литералы, записанные в одинарных и двойных кавыках?
7. Как в Python обозначается логическое значение «истина»?
8. Какую роль играют отступы в Python при выделении блока операторов?
9. Какие символы используются в Python для начала комментария?
10. Можно ли начинать имя переменной с символа подчёркивания "_"? Если да — в каких случаях это уместно?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 6. Правила объявления и использование переменных с различными типами данных.

Цель работы.

Изучить правила объявления и использования переменных с различными типами данных.

Краткие теоретические сведения.

Переменная — это имя, связанное со значением определённого типа хранящимся в оперативной памяти. В Python объявление переменной происходит автоматически при присваивании значения: отдельного описания типа не требуется. Это называется динамическая типизация, тип переменной в ходе выполнения программы может быть изменен.

В некоторых других языках программирования переменные необходимо явно объявлять перед использованием и тип переменной в процессе выполнения программы не может быть изменен.

Основные правила объявления переменных в Python:

- Имя переменной должно быть корректным идентификатором (см. правила именования).
- Переменная создаётся в момент первого присваивания: "x = 10".
- Тип переменной определяется типом присвоенного значения и может меняться в ходе выполнения программы.

Распространённые типы данных:
`int` — целые числа (42, -7)
`float` — числа с плавающей точкой (3.14, -2.5e3)
`str` — строки ("hello", "Python")
`bool` — логический тип (True, False)
`NoneType` — специальное значение None (отсутствие значения)

Примеры:

```
age = 20              # int
price = 199.99        # float
name = "Алиса"        # str
is_student = True     # bool
result = None         # NoneType
```

Переменные используются для хранения, обработки и передачи данных в программе. Важно выбирать осмысленные имена и соблюдать правила именования.

Обратите внимание Python регистрозависимый язык. Это означает, что переменные, например, name и Name это разные переменные.

Порядок выполнения работы.

Запустите интерактивный режим окно Python Shell. Откройте редактор через меню File → New File. В соответствии с вашим вариантом придумайте наименования для идентификаторов переменных и присвойте им значения, в комментарии # впишите тип переменной, который она получает с этим значением. После этого с помощью встроенной функции `type().__name__` проверьте правильно ли вы определили типы переменных.

Например:

```
age = 20              # int
price = 199.99        # float
name = "Алиса"        # str
is_student = True     # bool
result = None         # NoneType
print(age, price, name, is_student, result)
print("age -", type(age).__name__)
print("price -", type(price).__name__)
print("name -", type(name).__name__)
print("is_student -", type(is_student).__name__)
print("result -", type(result).__name__)
```

Сохраните и выполните код. Если будут ошибки проверьте правильность написанных строк и исправьте ошибки.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом возьмите строку значений, придумайте наименования идентификаторов присвойте им указанные значения и впишите их в программу. Ваш вариант – последняя цифра в номере зачетки или студенческого билета. Все значения идентификаторов вывести на экран командой `print`. Типы идентификаторов вывести на экран с помощью команд `print и type().__name__`

0. 42, 3.14, Python, True, 100//3, 2.5*4, False, "2026", 7>5, (10+5)==15, 1.5e3
1. -17, 0.0, Hello, False, 82, 99/11, True, "Lab", 5<=3, (20-8)!=12, 2e-4
2. 0, -4.5, Data, True, 50%7, 12.0+3.5, False, "Type", 100>=100, (3*4)<10, 5.25e6
3. 1024, 1e-3, Code, False, 255//8, 7.7-2.2, True, "Task", 0==0, (100/5)>20, 9.81e1
4. 999, 2.718, Exam, True, 1000%9, 0.1+0.2, False, "Test", 5!=5, (50*2)<=100, 6.02e23
5. -50, 9.99, Work, False, 210, 15.5/2, True, "Done", 3>3, (7+8)==15, 1e-10
6. 1, -0.001, Var, True, 123//10, 3.1415*2, False, "Out", 10<0, (100-1)>=99, 3.0e8
7. 2025, 0.5, Input, False, 64//4, 10.02, True, "Output", 1==2, (30/3)!=10, 1.6e-19
8. 777, 1.414, Logic, True, 99%10, 5.5+4.5, False, "Value", 0<=-1, (8*8)==64, 7.7e2
9. -100, 2.0, End, False, 1000//100, 7.0-7.0, True, "Final", 5>=5, (123%10)<5, 5e-5

Сохраните код программы для дальнейшего использования.

Задание 2.

В окне редактирования кода IDLE создайте не менее 10 идентификаторов переменных и присвойте им значения различных типов, обязательно должны быть целые числа, числа с

плавающей запятой, числа в экспоненциальном представлении, строки, логические выражения и значение без типа. Сохраните и выполните. Сохраните код программы для дальнейшего использования.

Контрольные вопросы.

1. Как в Python объявляется переменная? Нужно ли указывать её тип при объявлении?
2. Какие основные типы данных используются в Python для хранения чисел, текста и логических значений?
3. Что такое динамическая типизация, и как она проявляется при работе с переменными в Python?
4. Может ли одна и та же переменная в разные моменты времени содержать значения разных типов? Приведите пример.
5. Как проверить тип переменной во время выполнения программы?
6. Что произойдёт, если присвоить переменной строковое значение без кавычек (например, `x = hello`)?
7. Чем отличают типы `int` и `foat`?
8. Какой тип данных будет у результата логического выражения, например 5 > 3?
9. Можно ли использовать экспоненциальную запись 1e5 для задания числовых значений в Python? К какому типу данных относится такое значение?
10. Почему важно давать переменным осмысленные имена? Как это влияет на читаемость и отладку кода?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 7. Вычисления с использованием арифметических, логических и сравнительных операций. Ввод данных, присваивание и форматированный вывод.

Цель работы.

Изучить вычисления с использованием различных операций. Ввод и вывод данных.

Краткие теоретические сведения.

В Python вычисления строятся на трёх основных группах операций:

1. Арифметические операции:
   - `+, -, *, /` - базовые арифметические действия;
   - `**` - возведение в степень, например `2**4;`
   - `//` - целочисленное деление, возвращает частное, например `15//7;`
   - `%` - целочисленное деление, возвращает остаток, например `15%7;`

Обратите внимание операции `/` и `//` возвращают разные типы данных. Операция обычного деления `/` всегда возвращает тип `float`, даже если оба операнда целые и деление нацело. Операция целочисленного деления `//` при делении целых операндов всегда возвращает тип `int`, но если хоть один операнд имеет тип `float` тогда возвращает `float`.

2. Операции сравнения:
   - `==` - сравнение на равенство;
   - `!=` - сравнение на неравенство;
   - `<, >, <=, >=` - операции сравнения больше, меньше, больше или равно, меньше или равно;

3. Логические операции (для bool и совместимых типов):
   - `and, or, not` – логические операции И, ИЛИ, НЕ;

Ввод данных осуществляется с помощью функции `input()`, которая ВСЕГДА возвращает строку. Для чисел требуется преобразование типа:

```
x = int(input("Введите число: "))
```

В данном случае, чтобы в переменной `x` у нас было целое число, необходимо выполнить явное приведение строкового значения, которое вернет `input()` к целочисленному типу `int`. Выполняется это вот так `x=int(…..)` где в скобках выражение которое нужно привести к `int`.

Присваивание связывает имя переменной со значением:

```
a = 5
b = a + 3
```

Эти строки читаются как "Присвоить переменной `a` значение 3. Присвоить переменной `b` значение переменной `a`, увеличенное на 3."

Форматированный вывод позволяет красиво отображать результаты. Основные способы:

f-строки это современный рекомендуемый метод:

```
print(f"Результат: {a + b:.2f}")
```

метод `.format()`:

```
print("Результат: {:.2f}".format(a + b))
```

F-строки поддерживают выражения внутри "{}", округление, выравнивание и другие возможности, что делает их удобным инструментом для вывода.

Порядок выполнения работы.

Запустите интерактивный режим окно Python Shell. Откройте редактор через меню File → New File. В соответствии с вашим вариантом задания составьте программу.

Например:

Напишите программу, которая запрашивает у пользователя два числа, приводит их к типу float, выполняет с ними основные арифметические операции (сложение, вычитание, умножение и деление) и выводит результаты на экран в удобочитаемом виде с помощью f-строк. Убедитесь, что программа корректно работает с целыми и дробными числами.

```
# Ввод данных
a = float(input("Введите первое число: "))
b = float(input("Введите второе число: "))

# Арифметические операции
s = a + b          # сумма
d = a - b          # разность
p = a * b          # произведение
q = a / b          # частное

# Вывод с использованием f-строк
print(f"Сумма: {s}")
print(f"Разность: {d}")
print(f"Произведение: {p}")
print(f"Частное: {q}")
```

Сохраните и выполните код. Если будут ошибки проверьте правильность написанных строк и исправьте ошибки.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание.

0. Напишите программу, которая запрашивает у пользователя два числа, приводит оба к типу int, вычисляет: сумму, разность, произведение, целочисленное деление ("//"), остаток от деления ("%"). Выведите результаты с помощью f-строк.

1. Напишите программу, которая запрашивает у пользователя два числа, приводит первое к int, второе к float, вычисляет: сумму, частное ("/"), возведение первого числа в степень второго ("**"), целочисленное деление ("//"). Выведите результаты с помощью f-строк.

2. Напишите программу, которая запрашивает у пользователя два числа, приводит оба к типу "float", вычисляет: сумму, разность, произведение, частное ("/"), возведение первого числа в степень второго ("**"). Выведите результаты с помощью f-строк.

3. Напишите программу, которая запрашивает у пользователя два числа, приводит первое к "float", второе к "int", вычисляет: произведение, частное ("/"), целочисленное деление ("//"), остаток от деления ("%"). Выведите результаты с помощью f-строк.

4. Напишите программу, которая запрашивает у пользователя два числа, приводит оба к типу "int", вычисляет: сумму, разность, возведение первого числа в степень второго ("**"), остаток от деления ("%"). Выведите результаты с помощью f-строк.

5. Напишите программу, которая запрашивает у пользователя два числа, приводит первое к "int", второе к "float", вычисляет: разность, произведение, частное ("/"), возведение второго числа в степень первого ("**"). Выведите результаты с помощью f-строк.

6. Напишите программу, которая запрашивает у пользователя два числа, приводит оба к типу "float", вычисляет: сумму, целочисленное деление ("//"), остаток от деления ("%"), возведение первого числа в степень второго ("**"). Выведите результаты с помощью f-строк.

7. Напишите программу, которая запрашивает у пользователя два числа, приводит первое к "float", второе к "int", вычисляет: сумму, разность, целочисленное деление ("//"), остаток от деления ("%"). Выведите результаты с помощью f-строк.

8. Напишите программу, которая запрашивает у пользователя два числа, приводит оба к типу "int", вычисляет: произведение, целочисленное деление ("//"), возведение второго числа в степень первого ("**"), остаток от деления ("%"). Выведите результаты с помощью f-строк.

9. Напишите программу, которая запрашивает у пользователя два числа, приводит **первое к "int", второе к "float"**, вычисляет: сумму, частное ("/"), целочисленное деление ("//"), возведение первого числа в степень второго ("**"). Выведите результаты с помощью f-строк.

Задание 2.

В соответствии с вашим вариантом выполните задание. Используйте знания о спецификаторах и полях в f-строках полученные на леции.

0. Присвойте переменным a = 1234, b = 56.789. Вычислите: сумму a + b, разность a - int(b), произведение a * 0.1, и остаток a % 100. Выведите все четыре значения в

одной f-строке, используя форматы: .2f для суммы, d для разности, , (разделитель тысяч) для произведения, 03d для остатка.

1. Присвойте переменным x = 987, y = 3.14159. Вычислите: x // 10, y ** 2, x * y, x % 7. Выведите в одной f-строке, используя: d для целочисленного деления, .3f для квадрата, e для произведения, 02d для остатка.

2. Присвойте p = 5000, q = 0.025. Вычислите: p * q (проценты), p + p*q, p // 100, int(q * 1000). Выведите в f-строке с форматами: .1f для процентов, , для итоговой суммы, d для деления, 04d для последнего значения.

3. Присвойте m = 256, n = 12. Вычислите: m / n, m // n, m % n, m ** 0.5. Выведите в f-строке, используя: .2f для обычного деления, d для целочисленного, 02d для остатка, .4f для корня.

4. Присвойте u = 7777, v = 0.99. Вычислите: u * v, u - u*v, u // 7, v * 100. Выведите в f-строке с форматами: , для скидки, .2f для разницы, d для деления, .1f для процента.

5. Присвойте r = 1024, s = 8. Вычислите: r / s, r // s, r % s, r ** 2. Выведите в f-строке, используя: .1f для деления, d для целочисленного, 03d для остатка, , для квадрата.

6. Присвойте c = 999, d = 0.123456. Вычислите: c + d, c * d, int(d * 1000), c % 10. Выведите в f-строке с форматами: .3f для суммы, e для произведения, 03d для целой части, d для последней цифры.

7. Присвойте k = 2000, l = 7.5. Вычислите: k / l, k // l, k % l, l ** 3. Выведите в f-строке, используя: .2f для деления, d для целочисленного, .1f для остатка, , для куба (округлённого до целого).

8. Присвойте g = 333, h = 0.004. Вычислите: g * h, g + 1000, g // 33, h * 1e6. Выведите в f-строке с форматами: e для первого, , для второго, d для третьего, .0f для четвёртого.

9. Присвойте t = 888, w = 1.618. Вычислите: t * w, t // w, t % 100, w ** 2. Выведите в f-строке, используя: .2f для произведения, d для целочисленного деления, 02d для остатка, .3f для квадрата.

Контрольные вопросы.

1. Как в Python осуществляется ввод данных с клавиатуры и почему часто требуется преобразование типа?

2. Чем отличается оператор обычного деления "/" от целочисленного деления "//"? Приведите примеры.

3. Какие типы данных возвращает результат логического выражения (например, "5 > 3") и операции сравнения?

4. Что такое динамическая типизация, и как она проявляется при присваивании значений переменным?

5. Как правильно вывести на экран текст и значение переменной с помощью f-строки?

6. Какие форматные спецификаторы используются для вывода числа с двумя знаками после запятой, в экспоненциальной форме и с разделителем тысяч?

7. Можно ли выполнять арифметические операции непосредственно внутри f-строки? Приведите пример.

8. Почему выражение `print(f"Результат: {10 / 3:.2f}")` выводит `3.33`, а не `3`?

9. Какой тип данных будет у переменной после выполнения `x = input("Число: ")` и как его преобразовать в число?

10. Как вывести целое число с ведущими нулями (например, 007) с помощью f-строки?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 8. Разработка программ линейной структуры.

Цель работы.

Развить навыки решения практических задач с использованием линейных алгоритмов, научиться работать со сложными формулами и преобразованиями, освоить профессиональное форматирование вывода данных, научиться валидации входных данных на уровне линейной логики.

Краткие теоретические сведения.

Программа линейной структуры – это программа, в которой все операторы выполняются строго последовательно, один за другим, от первого до последнего.

Проектирование линейных программ
Шаги разработки:
1. Анализ задачи – понять, что нужно вычислить
2. Определение входных данных – что нужно ввести
3. Определение формул – как вычислять результат
4. Определение выходных данных – что выводить
5. Реализация – написание кода
6. Тестирование – проверка на разных данных
Пример структуры:

```
# 1. Ввод данных
a = float(input("Введите a: "))
b = float(input("Введите b: "))

# 2. Вычисления
summa = a + b
proizv = a * b

# 3. Вывод результатов
print(f"Сумма: {summa}")
print(f"Произведение: {proizv}")
```

Порядок выполнения работы.

Задание 1. Рассчитайте аннуитетный платеж по кредиту. Платеж = (Сумма * Ставка * (1 + Ставка)^Срок) / ((1 + Ставка)^Срок - 1)

Решение:

```
# Входные данные
S = float(input("Сумма кредита (руб.): "))
annual_rate = float(input("Годовая ставка (%): ")) / 100
years = int(input("Срок кредита (лет): "))

# Расчетные параметры
n = years * 12  # Количество месяцев
i = annual_rate / 12  # Месячная ставка

# Расчет платежа
```

```
    coeff_pow = (1 + i) ** n
    coeff_num = i * coeff_pow
    coeff_den = coeff_pow - 1
    monthly_payment = S * coeff_num / coeff_den
    total_payment = monthly_payment * n
    overpayment = total_payment - S

    # Форматированный вывод
    print(f"{'Параметр':<30} {'Значение':>20}")
    print("-" * 50)
    print(f"{'Ежемесячный платеж':<30} {monthly_payment:>20.2f}
руб.")
    print(f"{'Общая сумма выплат':<30} {total_payment:>20.2f}
руб.")
    print(f"{'Переплата':<30} {overpayment:>20.2f} руб.")
    print(f"{'Переплата в %':<30} {overpayment/S*100:>19.1f} %")
```

Задания к лабораторной работе

В соответствии с вашим вариантом выполните задание. Ваш вариант – последняя цифра в номере зачетки или студенческого билета. Вариант 10 соответствует значению цифры 0.

Задание 1.

Вариант 1. Рассчитайте эффективную процентную ставку с учетом комиссий. ЭПС = ((Сумма_выплат / Сумма_кредита)^(1/Срок) - 1) × 100.

Вариант 2. Рассчитайте точки безубыточности. Точка = Постоянные_затраты / (Цена - Переменные_затраты_на_единицу).

Вариант 3. Рассчитайте NPV (чистой приведенной стоимости). NPV = ∑(Денежный_поток / (1 + Ставка_дисконтирования)^Период).

Вариант 4. Рассчитайте индекс рентабельности. PI = (NPV + Инвестиции) / Инвестиции.

Вариант 5. Рассчитайте дисконтированный срок окупаемости.

Вариант 6. Рассчитайте IRR (внутренней нормы доходности) упрощенно.

Вариант 7. Рассчитайте суммы со сложным процентом с ежемесячной капитализацией

Вариант 8. Рассчитайте амортизацию по методу уменьшаемого остатка

Вариант 9. Рассчитайте таможенную стоимость товара с учетом пошлин и НДС

Вариант 10. Рассчитайте 13% от заработной платы равной 67965 руб.

Задание 2.

Вариант 1: Рассчитайте теплопотерю через стену. $Q = (t_{внутр} - t_{наруж}) \times S \times k / d$

Вариант 2: Рассчитайте сечения кабеля по мощности и длине. $S = (2 \times P \times L \times \rho) / (U \times \Delta U)$

Вариант 3: Рассчитайте прочность балки на изгибе $\sigma = (M \times y) / I$

Вариант 4: Рассчитайте расход жидкости через трубу. $Q = (\pi \times d^2 \times v) / 4$

Вариант 5: Рассчитайте КПД тепловой машины (цикл Карно). $\eta = 1 - T_{холод} / T_{горяч}$

Вариант 6: Рассчитайте период колебаний математического маятника. $T = 2\pi \times \sqrt{(L/g)}$

Вариант 7: Рассчитайте скорость истечения жидкости из отверстия (Торричелли). $v = \sqrt{(2gh)}$

Вариант 8: Рассчитайте фокусное расстояние линзы. $1/F = 1/f1 + 1/f2$

Вариант 9: Рассчитайте электрическую мощность трехфазной цепи. $P = \sqrt{3} \times U \times I \times \cos\varphi$

Вариант 10: Рассчитайте количество теплоты при фазовом переходе. $Q = m \times L$

Задание 3.

Вариант 1: Рассчитайте основные статистические показатели для 5 чисел (среднее, дисперсия, СКО, min, max, размах)

Вариант 2: Рассчитайте коэффициент корреляции Пирсона для двух выборок

Вариант 3: Линейная регрессия (расчет коэффициентов a и b) $y = ax + b$

Вариант 4: Рассчитайте индекс массы тела с классификацией (без if - использовать логические выражения)

Вариант 5: Рассчитайте проценты и накопленные суммы для вклада с пополнением

Вариант 6: Рассчитайте скользящее среднее для 5 значений

Вариант 7: Рассчитайте моды и медианы для 5 чисел (без использования циклов и сортировки)

Вариант 8: Рассчитайте коэффициент вариации и анализ однородности данных

Вариант 9: Рассчитайте доверительный интервал для среднего

Вариант 10: Рассчитайте показатели динамики (темп роста, темп прироста)

Контрольные вопросы.

1. Что такое программа линейной структуры?
2. Какие основные элементы включает линейная программа?
3. Почему линейные программы часто называют «последовательными»?
4. Может ли программа линейной структуры содержать условные операторы или циклы? Почему?
5. Какие типы задач наиболее эффективно решаются с использованием линейных алгоритмов?
6. В чём заключается последовательность выполнения команд в линейной программе?
7. Приведите пример реальной задачи, решение которой можно реализовать линейной программой.
8. Как обеспечивается однозначность результата в программе линейной структуры?
9. Почему линейные программы считаются простейшими по структуре управления?
10. Какие операции обычно выполняются в линейной программе (ввод, вычисления, вывод)?
11. Можно ли считать программу линейной, если она вызывает встроенные функции (например, sqrt, input)? Обоснуйте ответ.
12. Какие ограничения накладывает линейная структура на обработку различных входных данных?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 9. Разработка программ разветвляющей структуры.

Цель работы.

Изучить принципы разработки программ разветвляющейся структуры на языке Python. Приобрести практические навыки использования операторов ветвления if, elif, else для реализации алгоритмов с принятием решений в зависимости от различных условий.

Краткие теоретические сведения.

Программа разветвляющейся структуры – это программа, в которой выполнение операторов зависит от определенных условий. В таких программах используются операторы ветвления, которые позволяют выбирать различные пути выполнения кода в зависимости от результатов проверки условий.

Основные операторы ветвления в Python:

Оператор if

```
if условие:
    # выполняется, если условие истинно
    оператор1
    оператор2
```

Оператор if-else

```
if условие:
    # выполняется, если условие истинно
    операторы
else:
    # выполняется, если условие ложно
    Операторы
```

Оператор if-elif-else

```
if условие1:
    операторы1
elif условие2:
    операторы2
elif условие3:
    операторы3
else:
    операторы4
```

Логические выражения и операторы

Операторы сравнения:

```
a == b  # равно
a != b  # не равно
a > b   # больше
a < b   # меньше
a >= b  # больше или равно
a <= b  # меньше или равно
```

Логические операторы:

```
a and b  # логическое И (истинно, если оба истинны)
a or b     # логическое ИЛИ (истинно, если хотя бы один истинен)
```

```
not a    # логическое НЕ (инверсия)
```

Вложенные условия

```
if условие1:
    if условие2:
        операторы
    else:
        операторы
else:
    операторы
```

Порядок выполнения работы.

Задание 1. Определить, является ли число четным.

Решение:

```
number = int(input("Введите целое число: "))

if number % 2 == 0:
    print(f"Число {number} является ЧЕТНЫМ")
else:
    print(f"Число {number} является НЕЧЕТНЫМ")
```

Задания к лабораторной работе.

В соответствии с вашим вариантом выполните задание. Ваш вариант – последняя цифра в номере зачетки или студенческого билета. Вариант 10 соответствует значению цифры 0.

Задание 1.

Вариант 1. Дано одно число. Определить, является ли оно положительным, отрицательным или равным нулю.

Вариант 2. Даны два числа. Определить, какое из них больше.

Вариант 3. Дано одно число. Проверить, делится ли оно одновременно на 3 и на 5.

Вариант 4. Дан год. Определить, является ли этот год високосным.

Вариант 5. Даны три числа — длины сторон. Проверить, может ли существовать треугольник с такими сторонами.

Вариант 6. Даны координаты точки (x, y). Определить, в какой четверти координатной плоскости находится точка.

Вариант 7. Дано целое число. Проверить, является ли оно трехзначным.

Вариант 8. Дано число и границы интервала a и b. Определить, принадлежит ли число отрезку от a до b включительно.

Вариант 9. Дана строка (набор символов). Проверить, является ли она палиндромом (читается одинаково слева направо и справа налево, без учета регистра).

Задание 2.

Вариант 1. Дан номер месяца (от 1 до 12). Определить, к какому времени года относится этот месяц.

Вариант 2. Создать простой калькулятор. Пользователь вводит два числа и знак операции (+, −, *, /). Необходимо выполнить выбранную операцию.

Вариант 3. Даны коэффициенты квадратного уравнения. Определить количество корней и найти их значения.

Вариант 4. Создать конвертер единиц измерения. Пользователь выбирает тип величины (длина, масса или температура) и выполняет перевод.

Вариант 5. Дано количество набранных баллов (от 0 до 100). Определить оценку по заданной шкале.

Вариант 6. Даны рост и вес человека. Рассчитать индекс массы тела (ИМТ) и определить категорию состояния здоровья.

Вариант 7. Даны длины трех сторон треугольника. Определить тип треугольника: равносторонний, равнобедренный или разносторонний.

Вариант 8. Дано число и указана система счисления. Выполнить перевод числа в другую систему счисления (2, 8, 10 или 16).

Вариант 9. Дан размер дохода и категория налогоплательщика. Рассчитать сумму налога в зависимости от условий.

Вариант 10. Дан номер дня в году. Определить день недели, считая, что 1 января — понедельник.

Задание 3.

Вариант 1. Даны сумма кредита, срок и процентная ставка. Определить ежемесячный платеж для аннуитетного и дифференцированного видов выплат.

Вариант 2. По заданным параметрам определить уровень сложности экзаменационного вопроса (низкий, средний или высокий).

Вариант 3. На основе заданных предпочтений пользователя (жанр, год, рейтинг) подобрать подходящие фильмы.

Вариант 4. Рассчитать стоимость доставки с учетом веса посылки, расстояния и срочности доставки.

Вариант 5. Определить размер скидки на товар в зависимости от суммы покупки, количества товаров и типа клиента.

Вариант 6. На основе ответов на тест определить рекомендуемую профессию.

Вариант 7. Рассчитать суточную норму калорий с учетом пола, возраста, веса и уровня физической активности.

Вариант 8. По заданным симптомам определить возможную причину неисправности компьютера.

Вариант 9. На основе уровня дохода и склонности к риску предложить подходящую инвестиционную стратегию.

Вариант 10. Сгенерировать пароль с учетом выбранной длины и уровня сложности.

Контрольные вопросы.

1. Что такое разветвляющаяся структура программы и в чём её основное отличие от линейной?
2. Какие конструкции языка программирования используются для реализации ветвлений?
3. Что такое оператор ветвления и для чего он используется в программировании?
4. Чем отличается if-else от просто if?
5. Как работает оператор elif и когда его следует использовать?
6. Можно ли использовать несколько операторов elif в одной конструкции? Приведите пример.
7. Что такое вложенные условия и как их правильно оформлять?
8. В чём разница между полной и неполной формой условного оператора?
9. Может ли ветвление содержать более двух направлений выполнения? Если да, то как это реализуется?
10. Как обеспечивается корректная обработка всех возможных значений входных данных в разветвляющейся программе?
11. Приведите пример задачи, которую невозможно решить без использования разветвления.
12. Что произойдёт, если в условном операторе не указать блок else при необходимости обработки обоих случаев?

13. Как избежать ошибок, связанных с неправильной вложенностью условных операторов?
14. Почему важно проверять граничные значения условий при разработке разветвляющихся программ?
15. Можно ли заменить сложное ветвление циклом? Обоснуйте ответ.

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 10. Разработка программ циклической структуры.

Цель работы.

Изучить принципы построения программ с циклической структурой; освоить использование различных видов циклов (с предусловием, постусловием и с заданным числом повторений); научиться правильно организовывать тело цикла, управлять параметрами цикла и избегать ошибок, таких как зацикливание или пропуск итераций.

Краткие теоретические сведения.

Циклы – это управляющие конструкции, позволяющие выполнять один и тот же блок кода несколько раз. Они используются для:

обработки последовательностей данных;
повторения операций до достижения условия;
автоматизации повторяющихся задач;
работы с коллекциями данных.

Типы циклов в Python:
Цикл `while` – выполняется, пока условие истинно
Цикл `for` – выполняется для каждого элемента последовательности

Управление циклами

`break` – немедленный выход из цикла
`continue` – переход к следующей итерации
`else` – блок выполняется после завершения цикла (если не было `break`)

Порядок выполнения работы.

Задание 1. Дано число N. Найти сумму всех целых чисел от 1 до N.

Решение:

```
N = int(input("Введите N: "))
summa = 0
i = 1

while i <= N:
    summa += i
    i += 1

print(f"Сумма чисел от 1 до {N} = {summa}")

# Альтернативное решение с for
summa_for = 0
for num in range(1, N + 1):
    summa_for += num

print(f"Сумма через for: {summa_for}")
```

Задания к лабораторной работе.

В соответствии с вашим вариантом выполните задание. Ваш вариант – последняя цифра в номере зачетки или студенческого билета. Вариант 10 соответствует значению цифры 0.

Задание 1.

Вариант 1. Дано число N. Вывести все четные числа от 1 до N.

Вариант 2. Дано число. Вывести таблицу умножения для этого числа от 1 до 10.

Вариант 3. Дано число N. Вычислить факториал этого числа (произведение всех чисел от 1 до N).

Вариант 4. Дано число N. Вывести первые N чисел последовательности Фибоначчи.

Вариант 5. Дано целое число. Проверить, является ли оно простым.

Вариант 6. Дано целое число. Вывести все его делители.

Вариант 7. Дано целое число. Получить число, записанное в обратном порядке цифр.

Вариант 8. Дано целое число. Определить количество цифр в этом числе.

Вариант 9. Дано целое число. Найти сумму всех его цифр.

Вариант 10. Дано целое число. Проверить, является ли оно палиндромом.

Задание 2.

Вариант 1. Дана последовательность чисел. Найти наибольшее значение в этой последовательности.

Вариант 2. Дана последовательность чисел. Найти наименьшее значение.

Вариант 3. Дана последовательность чисел. Вычислить среднее арифметическое всех элементов.

Вариант 4. Дана последовательность чисел. Подсчитать количество положительных элементов.

Вариант 5. Дана последовательность чисел. Найти сумму всех отрицательных элементов.

Вариант 6. Дана последовательность чисел. Определить, есть ли среди элементов нулевые значения.

Вариант 7. Дана последовательность чисел. Найти произведение элементов, которые делятся на 3 без остатка.

Вариант 8. Дана последовательность чисел. Вывести те элементы, которые больше предыдущего элемента.

Вариант 9. Дана последовательность чисел. Найти второй по величине элемент.

Вариант 10. Дана последовательность чисел. Проверить, упорядочена ли она по возрастанию.

Задание 3.

Вариант 1. Дано число N. Найти все простые числа от 1 до N с использованием решета Эратосфена.

Вариант 2. Дана последовательность чисел. Отсортировать ее по возрастанию методом пузырьковой сортировки.

Вариант 3. Дан отсортированный массив чисел и искомое значение. Выполнить бинарный поиск элемента.

Вариант 4. Дано число N. Вывести треугольник Паскаля из N строк.

Вариант 5. Даны два целых числа. Найти их наибольший общий делитель и наименьшее общее кратное с помощью алгоритма Евклида.

Вариант 6. Дано целое число. Разложить его на простые множители.

Вариант 7. Дано число и основание системы счисления. Перевести число в систему счисления с основанием от 2 до 16.

Вариант 8. Дано число N. Найти все пары простых чисел-близнецов, не превышающие N.

Вариант 9. Вычислить приближенное значение числа π с использованием метода Монте-Карло.

Вариант 10. Дано число N. Найти все совершенные числа, не превышающие N.

Контрольные вопросы.

1. Что такое цикл и зачем он нужен в программировании?
2. Чем отличается цикл while от цикла for?
3. Как работает функция range()? Какие параметры она принимает?
4. Что делают операторы break и continue?
5. Что такое бесконечный цикл и как его избежать?
6. Как работает вложенный цикл?
7. Что такое циклическая структура программы и в каких случаях она применяется?
8. В чём разница между циклом с предусловием и циклом с постусловием?
9. Какой тип цикла целесообразно использовать, если известно точное количество повторений?
10. Что такое тело цикла и какие операторы могут в него входить?
11. Как избежать зацикливания при написании программы с циклом?
12. Можно ли изменять переменную-счётчик цикла внутри его тела? К чему это может привести?
13. Зачем нужна инициализация переменных перед началом цикла?
14. Как организовать досрочный выход из цикла или пропуск текущей итерации?
15. Почему важно проверять корректность условия завершения цикла на граничных значениях?
16. Приведите пример задачи, которую невозможно решить без использования циклической структуры.

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 11. Разработка программ с использованием одномерных массивов.

Цель работы.

Освоить основные принципы работы с одномерными массивами (списками) в языке Python — включая создание, заполнение, обработку элементов и реализацию типовых алгоритмов, таких как поиск, сортировка, вычисление суммы, среднего значения и других характеристик.

Краткие теоретические сведения.

Классическое определение понятия "одномерный массив" — это упорядоченная совокупность элементов как правило одного типа, расположенных в памяти последовательно и объединённых общим именем.

Но необходимо понимать, что в Python нет встроенного типа «массив» в классическом смысле, но вместо него чаще всего используют списки (list) — упорядоченные изменяемые коллекции объектов произвольных типов.

Таким образом одномерный массив в Python фактически является структурой называющейся "список" и обладает всеми преимуществами как одномерных массивов, так и списков.

Массив (список) в Python можно задать несколькими способами

```
arr = [] #Пустой массив

arr = [0,0,0,0,0] #Массив из 5 элементов заполненный нулями

arr = [0] * 5 # Тоже массив из 5 элементов заполненный нулями
```

Обратиться к элементу массива можно указав имя массива и в квадратных скобках индекс элемента. Элементы массива индексируются от 0 до длины массива-1. Длину массива в Python можно узнать вызвав функцию len(). В Python есть проверка границ массива. Попытка обратиться к индексу за границами массива вызовет ошибку IndexError и остановку выполнения программы.

Особенность Python! Индексы элементов массива могут быть отрицательными. Такая запись означает обращение к элементам массива с конца. Например `arr[-1]` обратится к последнему элементу массива. Обращение `arr[-2]` означает обращение ко второму с конца элементу. Если модуль отрицательного индекса больше длины последовательности так же будет ошибка IndexError. Отрицательные индексы удобны для доступа к последним элементам без явного вычисления длины.

```
arr[3] = 5
print(arr[3])
print(len(arr))
```

Одномерные массивы применяются для хранения набора однотипных или связанных данных. Например:

Список оценок, температур за неделю, имен студентов.

Обработки последовательностей перебор, поиск, фильтрация, сортировка — всё это делается над элементами массива.

Передачи данных в функции. Иногда необходимо передавать весь набор как один аргумент, а не поэлементно.

Реализации алгоритмов. Большинство базовых алгоритмов (поиск максимума, сумма, среднее, линейный/бинарный поиск и т.д.) работают с одномерными массивами.

Ввода и вывода серии значений. Например, прочитать 10 чисел с клавиатуры в цикле и сохранить в список.

Подготовки данных для дальнейшего анализа. В научных и прикладных задачах одномерные массивы — основной способ хранения векторов, временных рядов, сигналов и т.п.

Задания к лабораторной работе

Задание 1.

В соответствии с вашим вариантом выполните задание:

0. Создайте массив из 7 целых чисел. Выведите первый, последний и средний элементы массива.

1. Создайте массив из 5 строк — названий фруктов. Замените второй элемент на "банан" и выведите весь массив.

2. Создайте массив из 10 нулей. Присвойте значение 1 каждому элементу с чётным индексом (0, 2, 4...). Выведите получившийся массив.

3. Создайте массив из 6 чисел: 10, 20, 30, 40, 50, 60. Найдите и выведите сумму всех элементов.

4. Создайте массив из 8 чисел: 1, 2, 3, 4, 5, 6, 7, 8. Удалите последний элемент и выведите длину массива до и после удаления.

5. Создайте массив из 5 чисел: 2, 4, 6, 8, 10. Добавьте в конец число 99, а в начало — число -1. Выведите итоговый массив.

6. Создайте массив из 9 элементов: 11, 22, 33, 44, 55, 66, 77, 88, 99. Выведите элементы с индексами -1, -3 и -5, используя отрицательную индексацию.

7. Создайте массив из 6 чисел: 15, -3, 0, 42, 7, 100. Увеличьте каждый элемент на 10 и выведите новый массив.

8. Создайте массив из 10 чисел: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10. Поменяйте местами первый и последний элементы и выведите результат.

9. Создайте массив из 7 целых чисел: 5, -10, 0, 100, 33, 7, 42. Выведите каждый элемент на отдельной строке с указанием его индекса (например: "0: 5", "1: -10", "2: 0", ...).

Задание 2.

В соответствии с вашим вариантом выполните задание:

0. Заполните массив значениями: [45, 12, 88, 3, 67, 21, 94, 50, 76, 5, 33, 81, 19, 72, 6]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

1. Заполните массив значениями: [101, 22, 5, 89, 44, 77, 13, 96, 30, 61, 8, 73, 55, 2, 68]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

2. Заполните массив значениями: [34, 99, 17, 62, 5, 83, 28, 71, 40, 9, 56, 100, 23, 47, 12]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

3. Заполните массив значениями: [7, 58, 92, 25, 84, 11, 69, 36, 102, 43, 20, 75, 2, 51, 64]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

4. Заполните массив значениями: [18, 85, 3, 72, 49, 96, 27, 60, 14, 81, 5, 38, 93, 22, 67]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

5. Заполните массив значениями: [53, 10, 78, 29, 90, 6, 41, 87, 16, 74, 31, 99, 8, 55, 24]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

6. Заполните массив значениями: [66, 13, 82, 35, 4, 79, 26, 95, 12, 68, 47, 103, 19, 54, 32]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

7. Заполните массив значениями: [21, 70, 5, 88, 37, 100, 14, 59, 28, 91, 9, 42, 76, 3, 63]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

8. Заполните массив значениями: [44, 97, 18, 61, 7, 84, 33, 72, 5, 99, 26, 50, 101, 15, 39]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

9. Заполните массив значениями: [30, 65, 12, 93, 24, 78, 8, 51, 19, 86, 4, 70, 102, 37, 55]. Найдите минимальное и максимальное значения, их индексы, поменяйте элементы местами и выведите получившийся массив.

Задание 3.

По заданному алгоритму напишите программу.

Создайте массив arr не менее чем из 10 элементов, заполните его случайными значениями от 0 до 100 с помощью функции `arr[i] = random.randint(0, 100)`. Не забудьте импортировать библиотеку `import random`.

Алгоритм сортировки массива методом перестановок (сортировка выбором):

1. Начните с первого элемента массива и предположите, что он минимальный.
2. Последовательно сравните его со всеми остальными элементами массива.
3. Если найдётся элемент меньше текущего минимального, запомните его значение и индекс.
4. После полного прохода по массиву поменяйте местами первый элемент и найденный минимальный.
5. Перейдите ко второму элементу и повторите поиск минимального среди оставшихся элементов (начиная со второго до конца).
6. Поменяйте второй элемент с найденным минимальным.
7. Продолжайте этот процесс для каждого следующего элемента: на i-м шаге находите минимальный элемент в подмассиве от i-го до конца и меняйте его местами с i-м элементом.
8. Повторяйте шаги, пока не дойдёте до предпоследнего элемента.
9. После завершения всех проходов массив будет упорядочен по возрастанию.

Контрольные вопросы

1. Как в Python создать массив (список) из 10 нулей?
2. Чем отличается положительная индексация от отрицательной при обращении к элементам массива?
3. Можно ли в одном массиве хранить числа и строки одновременно? Почему?
4. Как найти длину массива в Python?
5. Как заменить значение элемента массива по известному индексу?
6. Что произойдёт, если обратиться к элементу массива с индексом, выходящим за его границы?
7. Как вывести все элементы массива в одну строку через пробел?
8. Как найти индекс первого вхождения заданного значения в массиве?
9. Как проверить, содержится ли определённое значение в массиве?
10. Как поменять местами два элемента массива, зная их индексы?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 12. Разработка программ с использованием двумерных массивов.

Цель работы.

Освоить создание, обработку и вывод двумерных массивов в Python, научиться выполнять типовые операции над матрицами, такие как поиск элементов, вычисление сумм по строкам и столбцам, работа с диагоналями и изменение структуры массива.

Краткие теоретические сведения.

Двумерный массив представляет собой матрицу. В Python двумерный массив реализуется как список списков — основной список содержит другие списки, каждый из которых представляет строку таблицы.

Элемент матрицы обозначается как matrix[i][j], где i — номер строки, j — номер столбца (индексация с нуля).

Количество строк определяется как len(matrix), а количество столбцов (в прямоугольной матрице) — как len(matrix[0]).

Создать двумерный массив в Python можно вручную

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

или используя конструкцию называемую списковое включение:

```
# Массив 3×4, заполненный нулями
matrix = [[0 for j in range(4)] for i in range(3)]
```

Нельзя создавать двумерный массив как `matrix = [[0]*4]*3` — все строки будут ссылаться на один и тот же список!

Двумерные массивы применяются для хранения табличных данных, изображений, игровых полей и других структур, организованных в виде сетки. Основные операции включают ввод и вывод матрицы, вычисление сумм по строкам или столбцам, поиск минимального или максимального элемента, работу с диагоналями и транспонирование.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание.

0. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 150. Найдите сумму всех элементов главной диагонали (главная диагональ слева-направо) и выведите её.
1. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 55. Найдите сумму всех элементов побочной (побочная диагональ справа-налево) диагонали и выведите её.
2. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 550. Вычислите сумму элементов в каждой строке и выведите все суммы.
3. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 80. Вычислите сумму элементов в каждом столбце и выведите все суммы.

4. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 40. Найдите максимальный элемент во всём массиве и укажите его индексы (номер строки и столбца).
5. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 56. Найдите минимальный элемент в каждой строке и выведите эти значения.
6. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 965. Замените все чётные элементы на ноль и выведите получившийся массив.
7. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 65. Посчитайте количество нечётных чисел в массиве и выведите результат.
8. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 1001. Поменяйте местами первую и последнюю строку массива и выведите результат.
9. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 45. Выведите элементы массива построчно, разделяя их пробелами.

Задание 2.

В соответствии с вашим вариантом выполните задание.

0. Создайте двумерный массив размером 8×8, заполненный случайными целыми числами от 1 до 95. Найдите сумму элементов, расположенных выше главной диагонали, и сумму элементов, расположенных ниже неё. Выведите обе суммы.
1. Создайте двумерный массив размером 7×7, заполненный случайными целыми числами от 1 до 130. Найдите сумму элементов, расположенных выше побочной диагонали, и сумму элементов, расположенных ниже неё. Выведите обе суммы.
2. Создайте двумерный массив размером 9×9, заполненный случайными целыми числами от 1 до 275. Для каждой строки найдите среднее арифметическое её элементов и выведите все средние значения с округлением до двух знаков после запятой.
3. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 180. Найдите максимальный элемент в каждой чётной строке (0, 2, 4) и минимальный элемент в каждой нечётной строке (1, 3, 5). Выведите полученные значения.
4. Создайте двумерный массив размером 10×10, заполненный случайными целыми числами от 1 до 64. Замените все элементы на главной и побочной диагоналях на ноль и выведите получившийся массив.
5. Создайте двумерный массив размером 7×7, заполненный случайными целыми числами от 1 до 210. Посчитайте количество нечётных чисел во всём массиве и выведите результат.
6. Создайте двумерный массив размером 9×9, заполненный случайными целыми числами от 1 до 420. Умножьте каждый элемент матрицы на номер его строки (индексация с нуля) и выведите полученную матрицу.
7. Создайте двумерный массив размером 8×8, заполненный случайными целыми числами от 1 до 88. Найдите сумму элементов в четырёх угловых позициях: (0,0), (0,7), (7,0), (7,7).
8. Создайте двумерный массив размером 6×6, заполненный случайными целыми числами от 1 до 330. Удалите средние две строки (2 и 3) и средние два столбца (2 и 3), оставив только внешнюю "рамку". Выведите полученную матрицу размером 4×4.
9. Создайте двумерный массив размером 10×10, заполненный случайными целыми числами от 1 до 500. Поверните матрицу на 90 градусов по часовой стрелке и выведите результат.

Контрольные вопросы.

1. Как в Python создать двумерный массив размером N×M, заполненный нулями, без использования внешних библиотек?
2. Почему конструкция `matrix = [[0] * M] * N` не подходит для создания двумерного массива с независимыми строками?
3. Как обратиться к элементу в i-й строке и j-м столбце двумерного массива?
4. Как определить количество строк и количество столбцов в прямоугольном двумерном массиве?
5. Что такое главная диагональ матрицы и как проверить, лежит ли элемент на ней?
6. Что такое побочная диагональ и по какому условию можно определить, что элемент принадлежит ей?
7. Как вывести двумерный массив построчно так, чтобы элементы в строке были разделены пробелами?
8. Как найти сумму всех элементов в заданной строке двумерного массива?
9. Как заменить все элементы определённого столбца на заданное значение?
10. Как проверить, является ли двумерный массив квадратным (то есть количество строк равно количеству столбцов)?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 13. Разработка программ с использованием списков, кортежей, множеств и словарей.

Цель работы.

Освоить основные встроенные коллекции языка Python — списки, кортежи, множества и словари, научиться выбирать подходящий тип данных для решения задач, а также применять их методы для хранения, обработки и анализа данных.

Краткие теоретические сведения.

В Python существует четыре основных встроенных типа коллекций:

— Список (list) — упорядоченная изменяемая последовательность элементов, допускающая дубликаты. Элементы индексируются, начиная с нуля. Списки создаются с помощью квадратных скобок: [1, 2, 3]. Списки мы рассмотрели в предыдущих лабораторных работах посвященных массивам.

— Кортеж (tuple) — упорядоченная неизменяемая последовательность. После создания его элементы нельзя изменить. Это его основное отличие от списка (массива). Кортежи быстрее списков и подходят для хранения константных данных. Создаются с помощью круглых скобок: (1, 2, 3) или просто через запятую: 1, 2, 3.

— Множество (set) — неупорядоченная коллекция уникальных элементов. Не поддерживает индексацию, но позволяет быстро выполнять операции объединения, пересечения и разности. Создаётся с помощью фигурных скобок без ключей: {1, 2, 3} или функцией set().

— Словарь (dict) — неупорядоченная коллекция пар «ключ — значение». Ключи должны быть неизменяемыми и уникальными, значения могут быть любого типа. Создаётся как {ключ: значение, ...}.

Каждый тип коллекции решает свои задачи: списки — для последовательностей с возможностью изменения, кортежи — для защиты данных от случайного изменения, множества — для работы с уникальными элементами и логическими операциями над ними, словари — для хранения структурированной информации по именованным ключам.

```
#Кортежи
tuple1 = (1, 2, 3)
tuple2 = 4, 5, 6 # круглые скобки можно опустить
tuple3 = (7,)   # кортеж из одного элемента — запятая обязательна!
tuple4 = ()     # пустой кортеж

# Множества
set1 = {1, 2, 3, 2} # дубликаты автоматически удаляются {1, 2, 3}
set2 = set([4, 5, 6]) # создание из списка
set3 = set()    # пустое множество (только так! {} — это словарь)

# Словари
dict1 = {"name": "Алиса", "age": 25}
dict2 = {1: "один", 2: "два"}
dict3 = dict(name="Боб", age=30) # через функцию dict()
dict4 = {} # пустой словарь
```

```
# Вывод для проверки
print("Кортежи:", tuple1, tuple2, tuple3, tuple4)
print("Множества:", set1, set2, set3)
print("Словари:", dict1, dict2, dict3, dict4)
```

Примеры работы со структурами. Кортежи.

```
# === Кортежи (tuple) ===
t = (10, 20, 30)
print("Кортеж:", t)
print("Элемент по индексу 1:", t[1])
print("Длина кортежа:", len(t))
print("Сложение кортежей:", t + (40, 50))
print("Повторение:", t * 2)

# Кортеж нельзя изменить:
# t[0] = 100  # есл раскомментировать строка вызовет ошибку!
# Можно объединить два кортежа в новый кортеж
t2 = (101,102,105)
t3 = t + t2
print(t3)
# Распаковка кортежа
a, b, c = t
print("Распаковка:", a, b, c)
```

Распаковка кортежа – значения кортежа присваиваются переменным a,b,c

Множества.

```
# === Множества (set) ===
s1 = {1, 2, 3, 4}
s2 = {3, 4, 5, 6}
print("\nМножество s1:", s1)
print("Множество s2:", s2)

# Операции над множествами
print("Объединение:", s1 | s2)        # или s1.union(s2)
print("Пересечение:", s1 & s2)        # или s1.intersection(s2)
print("Разность s1 - s2:", s1 - s2)   # или s1.difference(s2)
print("Симметрическая разность:", s1 ^ s2)

# Добавление и удаление
s1.add(7)
s1.discard(1)  # не вызывает ошибку, если элемента нет
print("После add/discard:", s1)
```

Словари.

```
# === Словари (dict) ===
```

```
d = {"город1":"Москва", "страна":"Россия", "население":"12 млн."}
d = {"город2":"Ростов-на-Дону", "страна":"Россия",
"население":"1.2 млн."}

print("\nСловарь:", d)
print("Значение по ключу 'город1':", d["город1"])
print("Ключи:", list(d.keys()))
print("Значения:", list(d.values()))
print("Пары ключ-значение:", list(d.items()))

# Изменение и добавление
d["столица"] = True
d["население"] = 12_500_000
print("После изменения:", d)

# Безопасное получение значения
print("Регион:", d.get("регион", "не указан"))

# Удаление
removed = d.pop("страна")
print("Удалено значение:", removed)
print("Словарь после pop:", d)
```

Срезы

Срезы (slices) — это способ получения части последовательности (списка, кортежа, строки и т.д.) в Python.

Синтаксис среза:

```
последовательность[start:stop:step]
```

`start` — индекс начала среза (включается), по умолчанию 0.
`stop` — индекс конца среза (не включается), по умолчанию длина последовательности.
`step` — шаг (на сколько элементов переходить), по умолчанию 1.

```
a = (10, 20, 30, 40, 50)
# Элементы со 2-го по 4-й (индексы 2 и 3)
print(a[2:4])        # (30, 40)
# От начала до 3-го элемента
print(a[:3])         # (10, 20, 30)
# От 2-го элемента до конца
print(a[2:])         # (30, 40, 50)
# Каждый второй элемент
print(a[::2])        # (10, 30, 50)
# Весь кортеж в обратном порядке
print(a[::-1])       # (50, 40, 30, 20, 10)
```

Срез **всегда создаёт новый объект** того же типа (новый кортеж, новый список и т.д.), не изменяя исходный. Срезы работают со всеми последовательностями: строками, списками, кортежами.

Задания к лабораторной работе.

Задание 1. Кортежи.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Создайте кортеж из пяти целых чисел. Выведите первый, последний и средний элементы кортежа.
2. Создайте два кортежа: один с числами (1, 2, 3), другой — с буквами ('a', 'b', 'c'). Объедините их в один кортеж и выведите результат.
3. Создайте кортеж из семи строк — названий дней недели. Используя отрицательную индексацию, выведите последний, предпоследний и третий с конца элементы.
4. Создайте кортеж из шести чисел. Найдите сумму всех элементов, преобразовав кортеж в список или используя встроенную функцию `sum()`.
5. Создайте кортеж из четырёх элементов: имя, фамилия, возраст, город. Распакуйте его в отдельные переменные и выведите информацию в виде строки: «Имя: ..., Фамилия: ..., Возраст: ..., Город: ...».
6. Создайте кортеж из восьми чисел. Определите, сколько раз встречается заданное число (например, 5) в кортеже, используя метод `.count()`.
7. Создайте кортеж из пяти чисел. Проверьте, находится ли число 10 в кортеже, и выведите соответствующее сообщение.
8. Создайте кортеж из трёх координат точки в пространстве: (x, y, z). Используя распаковку, вычислите и выведите сумму квадратов координат.
9. Создайте кортеж из шести элементов. Получите срез, содержащий элементы со второго по пятый (включительно), и выведите его.
10. Создайте кортеж из девяти чисел. Получите срез, содержащий каждый второй элемент, начиная с первого, и выведите его.

Задание 2. Множества.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Создайте два множества: первое — из чисел от 1 до 10, второе — из чётных чисел от 1 до 15. Найдите их пересечение и выведите результат.
2. Создайте множество из букв слова «программирование». Выведите все уникальные буквы в алфавитном порядке.
3. Создайте два множества: A = {1, 2, 3, 4, 5}, B = {4, 5, 6, 7, 8}. Найдите разность A − B и B − A, выведите оба результата.
4. Создайте множество из пяти случайных целых чисел от 1 до 20. Добавьте в него число 100, затем удалите число 5 (если оно есть). Выведите итоговое множество.
5. Даны три множества: студенты, сдавшие математику; студенты, сдавшие физику; студенты, сдавшие информатику. Найдите тех, кто сдал **все три** предмета (пересечение трёх множеств).
6. Создайте множество A = {1, 2, 3, 4} и множество B = {3, 4, 5, 6}. Найдите симметрическую разность (элементы, которые есть в одном из множеств, но не в обоих) и выведите её.
7. Проверьте, является ли одно множество подмножеством другого. Используйте множества: X = {2, 4, 6}, Y = {1, 2, 3, 4, 5, 6, 7}. Выведите, является ли X подмножеством Y.
8. Создайте множество из цифр вашего года рождения. Создайте второе множество — из цифр текущего года. Найдите объединение этих множеств и выведите его.
9. Удалите из множества {10, 20, 30, 40, 50} все элементы, которые делятся на 10 и больше 25. Выведите оставшиеся элементы.

10. Создайте пустое множество. Добавьте в него пять строк: названия цветов. Затем попробуйте добавить дубликат одного из цветов. Выведите итоговое множество и объясните, почему дубликат не появился.

Задание 3. Словари.
В соответствии с вашим вариантом выполните задание.

1. Создайте словарь, содержащий информацию о студенте: имя, возраст, группа, средний балл. Выведите значение по ключу «средний балл».
2. Создайте словарь с пятью парами «страна — столица». Попросите пользователя ввести название страны и выведите соответствующую столицу. Если такой страны нет — выведите сообщение «Страна не найдена».
3. Дан словарь с ценами на товары: {"хлеб": 50, "молоко": 80, "сыр": 150}. Увеличьте все цены на 10 % и выведите обновлённый словарь.
4. Создайте словарь из трёх элементов. Добавьте в него два новых ключа с помощью присваивания и один — с помощью метода .update(). Выведите итоговый словарь.
5. Дан словарь: {"a": 1, "b": 2, "c": 3, "d": 4}. Найдите сумму всех значений и выведите её.
6. Создайте словарь, где ключи — это числа от 1 до 5, а значения — квадраты этих чисел. Выведите получившийся словарь.
7. Удалите из словаря {"x": 10, "y": 20, "z": 30} элемент с ключом "y" с помощью оператора del и элемент с ключом "z" с помощью метода .pop(). Выведите удалённое значение "z" и оставшийся словарь.
8. Проверьте, есть ли в словаре {"имя": "Анна", "возраст": 22} ключ "город". Если нет — добавьте его со значением "Москва". Выведите итоговый словарь.
9. Создайте словарь с пятью парами «слово — перевод». Выведите все ключи, все значения и все пары «ключ — значение» отдельно.
10. Создайте словарь, описывающий расписание на неделю: ключи — дни недели («понедельник», «вторник» и т.д.), значения — количество пар в этот день. Найдите и выведите день с максимальным количеством пар.

Контрольные вопросы.

1. Чем кортеж отличается от списка?
2. Как создать кортеж из одного элемента? Почему это требует особого синтаксиса?
3. Можно ли изменить элемент кортежа после его создания? Почему?
4. Как объединить два кортежа в один?
5. Что произойдёт, если попытаться использовать список как элемент множества?
6. Как создать пустое множество? Почему запись {} не подходит?
7. Как проверить, содержится ли элемент в множестве?
8. Как найти пересечение двух множеств? Назовите два способа.
9. Что такое симметрическая разность множеств и как её вычислить в Python?
10. Можно ли в множестве хранить дубликаты? Почему?
11. Как добавить элемент в множество?
12. Как удалить элемент из множества, если он может отсутствовать?
13. Что такое ключ в словаре и какие требования к нему предъявляются?
14. Как получить значение из словаря, если ключ может отсутствовать, не вызывая ошибку?
15. В чём разница между оператором del и методом pop() при работе со словарём?
16. Как обновить несколько значений в словаре за один раз?
17. Как получить список всех ключей словаря?
18. Можно ли использовать кортеж как ключ словаря? А список? Почему?

19. Как создать словарь на основе двух списков: один содержит ключи, другой — значения?
20. Как проверить, существует ли определённый ключ в словаре?
21. Как преобразовать список в множество и зачем это может быть полезно?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 14. Разработка программ с использованием стандартных функций для работы со строками.

Цель работы.

Освоить основные стандартные функции и методы Python для обработки строк, научиться решать практические задачи: анализ текста, преобразование регистра, поиск и замена подстрок, разбиение и объединение строк.

Краткие теоретические сведения.

Краткая теория:

В Python строки (тип str) — это неизменяемые последовательности символов. Над строками можно выполнять множество операций с помощью встроенных методов.

Основные методы работы со строками:

`len(s)` — длина строки;

`s.upper(), s.lower()` — преобразование к верхнему/нижнему регистру;

`s.strip()` — удаление пробелов (или других символов) с начала и конца строки;

`s.find(sub), s.index(sub)` — поиск подстроки (возвращают индекс или вызывают ошибку);

`s.replace(old, new)` — замена всех вхождений подстроки;

`s.split(sep)` — разбиение строки на список по разделителю;

`"".join(list)` — объединение списка строк в одну строку с указанным разделителем;

`s.startswith(pref), s.endswith(suff)` — проверка начала или конца строки;

`s.isalpha(), s.isdigit(), s.islower()` и др. — проверка типа символов.

`s – строковая переменная`

Строки поддерживают индексацию (`s[0]`), срезы (`s[1:5]`) и перебор в цикле. Все методы возвращают новую строку, так как исходная строка неизменяема. Эти инструменты позволяют эффективно обрабатывать текстовые данные без написания сложного кода вручную.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Объявите строку: `" Привет, Мир!!! "`. Определите её длину. Преобразуйте строку к верхнему регистру. Удалите пробелы в начале и конце, а также восклицательные знаки в конце строки.
2. Объявите строку: `"***Python***"`. Определите её длину. Преобразуйте строку к нижнему регистру. Удалите символы `*` с начала и конца строки.
3. Объявите строку: `"__Данные__"`. Определите её длину. Преобразуйте строку к верхнему регистру. Удалите символы подчёркивания с начала и конца строки.
4. Объявите строку: `"123текст123"`. Определите её длину. Преобразуйте строку к нижнему регистру. Удалите цифры `1`, `2`, `3` с начала и конца строки.
5. Объявите строку: `" Data Science $$$"`. Определите её длину. Преобразуйте строку к верхнему регистру. Удалите пробелы в начале и символы `$` в конце строки.
6. Объявите строку: `"###Отчёт###"`. Определите её длину. Преобразуйте строку к нижнему регистру. Удалите символы `#` с начала и конца строки.

7. Объявите строку: `" Hello World ???"`. Определите её длину. Преобразуйте строку к верхнему регистру. Удалите пробелы в начале и знаки вопроса в конце строки.
8. Объявите строку: `"xxxАлгоритмxxx"`. Определите её длину. Преобразуйте строку к нижнему регистру. Удалите символы `x` с начала и конца строки.
9. Объявите строку: `"%%%Результат%%%"`. Определите её длину. Преобразуйте строку к верхнему регистру. Удалите символы `%` с начала и конца строки.
10. Объявите строку: `" Код: ABC123 "`. Определите её длину. Преобразуйте строку к нижнему регистру. Удалите пробелы в начале и конце строки.

Задание 2.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Объявите строку: `"apple,banana,orange,grape"`. Определите её длину. Переберите все символы и подсчитайте количество букв `'a'`. Получите срез от 3-го до 10-го символа. Разбейте строку по запятой. Замените слово `"banana"` на `"kiwi"`.
2. Объявите строку: `"2024-05-15"`. Определите её длину. Переберите символы и найдите все цифры (подсчитайте их количество). Получите срез, содержащий только год (`"2024"`). Разбейте строку по символу `'-'`. Замените `"05"` на `"декабрь"`.
3. Объявите строку: `"Иванов;Петр;Сергеевич"`. Определите её длину. Переберите символы и подсчитайте количество заглавных букв. Получите срез первых 6 символов. Разбейте строку по точке с запятой. Замените `"Иванов"` на `"Сидоров"`.
4. Объявите строку: `"hello world python programming"`. Определите её длину. Переберите символы и подсчитайте количество пробелов. Получите срез от конца строки длиной 11 символов (`"programming"`). Разбейте строку по пробелу. Замените `"python"` на `"java"`.
5. Объявите строку: `"user@example.com"`. Определите её длину. Переберите символы и проверьте наличие символа `'@'`. Получите срез до `'@'` (имя пользователя) и после `'@'` (домен). Разбейте строку по `'@'`. Замените `"example.com"` на `"mail.ru"`.
6. Объявите строку: `"ID:1001 NAME:Alice STATUS:active"`. Определите её длину. Переберите символы и подсчитайте количество двоеточий. Получите срез от 4-го до 8-го символа (`"1001"`). Разбейте строку по пробелу. Замените `"active"` на `"inactive"`.
7. Объявите строку: `"red|green|blue|yellow"`. Определите её длину. Переберите символы и подсчитайте общее количество гласных букв (в английском варианте). Получите срез последних 6 символов (`"yellow"`). Разбейте строку по символу `'|'`. Замените `"green"` на `"purple"`.
8. Объявите строк punitive: `"Москва - столица России."`. Определите её длину. Переберите символы и подсчитайте количество пробелов и знаков препинания (точка, тире). Получите срез от начала до первого пробела. Разбейте строку по пробелам. Замените `"России"` на `"Франции"`.
9. Объявите строку: `"login=admin&password=12345&role=user"`. Определите её длину. Переберите символы и подсчитайте количество символов `'='`. Получите срез от 6-го до 11-го символа (`"admin"`). Разбейте строку по `'&'`. Замените `"user"` на `"admin"`.
10. Объявите строку: `"abc123def456ghi789"`. Определите её длину. Переберите символы и соберите все цифры в отдельную строку. Получите срез от 9-го до 15-го символа (`"def456"`). Разбейте строку на части по три символа (вручную или через срезы). Замените `"123"` на `"XXX"`.

Контрольные вопросы.

1. Как определить длину строки в Python?
2. Почему строки в Python называются неизменяемыми? Что происходит при вызове метода, изменяющего строку?
3. В чём разница между методами `find()` и `index()` при поиске подстроки?
4. Как удалить пробелы (или другие символы) с начала и конца строки?
5. Как преобразовать все буквы строки к верхнему или нижнему регистру?
6. Что делает метод `split()` и как указать свой разделитель?
7. Как объединить список строк в одну строку с заданным разделителем?
8. Как заменить все вхождения одной подстроки на другую в строке?
9. Как получить часть строки (срез) от 3-го до 8-го символа?
10. Как перебрать все символы строки в цикле и, например, подсчитать количество цифр?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 15. Разработка программ с использованием регулярных выражений для работы со строками.

Цель работы.

Освоить использование регулярных выражений в Python для поиска, проверки и извлечения информации из текстовых данных, научиться решать задачи обработки строк с помощью модуля re.

Краткие теоретические сведения.

Регулярные выражения — это шаблоны для описания множества строк. В Python они реализованы в стандартном модуле `re`. Для использования модуля в программе его необходимо импортировать командой `import re`. С их помощью можно выполнять сложный поиск, замену и проверку текста без написания громоздких условий.

Основные функции модуля `re`:

`re.search(pattern, string)` — ищет первое совпадение шаблона в строке;

`re.findall(pattern, string)` — находит все неперекрывающиеся совпадения и возвращает список;

`re.match(pattern, string)` — проверяет, совпадает ли начало строки с шаблоном;

`re.sub(pattern, repl, string)` — заменяет все совпадения на указанную строку;

`re.split(pattern, string)` — разбивает строку по шаблону.

Примеры часто используемых элементов регулярных выражений:

`\d` - любая цифра,

`\w` - буква, цифра или подчёркивание,

`\s` - пробельный символ,

`.` - любой символ (кроме перевода строки),

`*, +, ?` - квантификаторы (0 или более, 1 или более, 0 или 1),

`[...]` - набор символов,

`^, $` - начало и конец строки.

Регулярные выражения особенно полезны при обработке логов, форматированных данных, email-адресов, телефонных номеров, дат и других структурированных текстов.

Примеры использования регулярных выражений для решения разных задач.

```
import re

# 1. Поиск email-адреса в тексте
text = "Свяжитесь с нами: support@example.com или info@site.ru"
# \b — граница слова (чтобы не захватить часть более длинного
слова)
# [\w.-]+ — один или несколько символов: буквы, цифры,
подчёркивание, точка или дефис (имя до @)
# @ — обязательный символ @
# [\w.-]+ — доменное имя (может содержать точки и дефисы)
```

```
# \. — обязательная точка перед доменной зоной (экранирована, так
как . в регэкспах означает "любой символ")
# \w+ — доменная зона (например, com, ru) — только буквы/цифры/_
emails = re.findall(r'\b[\w.-]+@[\w.-]+\.\w+\b', text)
print("Emails:", emails)

# 2. Проверка простого формата телефона: "8-900-123-45-67"
phone = "8-900-123-45-67"
# ^ — начало строки
# \d{1} — ровно одна цифра (код страны или оператора)
# - — дефис как разделитель
# \d{3} — три цифры (код оператора)
# -\d{3} — дефис и ещё три цифры
# -\d{2} — дефис и две цифры
# -\d{2} — дефис и ещё две цифры
# $ — конец строки (гарантирует, что вся строка соответствует
шаблону)
if re.match(r'^\d{1}-\d{3}-\d{3}-\d{2}-\d{2}$', phone):
    print("Телефон корректный")

# 3. Извлечение всех целых чисел из строки
s = "Цены: 100 руб, 250 руб, 50 руб"
# \d+ — одна или более цифр подряд (находит любое целое число)
numbers = re.findall(r'\d+', s)
print("Числа:", numbers)

# 4. Замена всех цифр на '#'
text = "Пароль: secret123"
# \d — любая одна цифра
# re.sub заменяет каждую найденную цифру на '#'
masked = re.sub(r'\d', '#', text)
print(masked)

# 5. Разбиение строки по запятой, точке с запятой или пробелу
data = "яблоко,банан;апельсин груша"
# [,\s;]+ — один или несколько символов из набора: запятая, пробел
(\s), точка с запятой
# + гарантирует, что несколько идущих подряд разделителей
(например, ", ") считаются одним
parts = re.split(r'[,\s;]+', data)
print(parts)

# 6. Поиск слов, начинающихся с заглавной русской буквы
sentence = "Москва и Санкт-Петербург — города России."
# \b — граница слова
# [А-ЯЁ] — одна заглавная русская буква (включая Ё)
# [а-яё]* — ноль или более строчных русских букв (включая ё)
# Вместе: слово, начинающееся с заглавной русской буквы
capitalized = re.findall(r'\b[А-ЯЁ][а-яё]*\b', sentence)
print("Слова с заглавной буквы:", capitalized)
```

```
# 7. Проверка, что строка состоит только из русских букв и
пробелов
name = "Иван Петров"
# re.fullmatch — проверяет, соответствует ли ВЕСЬ текст шаблону
# [А-ЯЁа-яё\s]+ — один или более символов: русские буквы любого
регистра или пробелы (\s)
# Без ^ и $, потому что fullmatch и так проверяет всю строку
if re.fullmatch(r'[А-ЯЁа-яё\s]+', name):
    print("Имя корректное")
```

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Проверьте, является ли строка корректным **email-адресом** с доменом `.ru`, `.com` или `.рф`. Адрес должен содержать имя пользователя (латинские буквы, цифры, точки, подчёркивания), символ `@` и допустимое доменное имя (например: `user_123@example.ru`).
2. Найдите в строке все **URL-адреса**, начинающиеся с `http://` или `https://`, за которыми следует домен (например, `https://example.ru/page`). Извлеките полные URL.
3. Проверьте, соответствует ли строка формату **российского мобильного номера**: может начинаться с `+7`, `8` или без кода страны, содержит 10 цифр после кода, допускает пробелы, дефисы и скобки (например: `+7 (900) 123-45-67`).
4. Проверьте, что **логин пользователя** состоит только из латинских букв, цифр, дефиса и подчёркивания, начинается с буквы и имеет длину от 3 до 20 символов.
5. Проверьте, что **пароль** содержит не менее 8 символов, включает хотя бы одну заглавную букву, одну строчную, одну цифру и один специальный символ из набора `!@#$%^&*`.
6. Найдите в строке все **доменные имена**, заканчивающиеся на `.ru`, `.com` или `.рф` (например: `my-site.ru`, `test123.com`), но не как часть email или URL — только отдельно стоящие домены.
7. Проверьте, что **ФИО** состоит из 2–3 слов, каждое начинается с заглавной кириллической буквы, остальные — строчные, между словами ровно один пробел (например: «Иванов Петр» или «Сидорова Анна Владимировна»).
8. Извлеките из строки все **параметры запроса** из URL (часть после `?`), например из `"https://site.ru/search?q=python&lang=ru"` извлеките пары `q=python` и `lang=ru`.
9. Найдите в строке все **email-адреса** с доменами `.ru`, `.com` или `.рф` и выведите их в виде списка.
10. Проверьте, что строка представляет собой **корректный домен верхнего уровня**, состоящий из латинских букв, цифр, дефисов (но не в начале и не в конце), и заканчивается на `.ru`, `.com` или `.рф` (например: `my-domain123.ru`).

Задание 2.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Найдите все слова в тексте, состоящие **ровно из 5 букв** (только кириллица). Игнорируйте знаки препинания и цифры.

2. Извлеките из строки все **даты в формате ДД.ММ.ГГГГ** (например, «15.03.2024»). Убедитесь, что день — от 01 до 31, месяц — от 01 до 12.
3. Найдите все **цены в формате «1 250 руб.» или «999 руб.»**, где число может содержать пробел как разделитель тысяч. Извлеките только числовые значения (без «руб.» и пробелов).
4. Определите, содержит ли строка **хотя бы одно слово, написанное целиком заглавными буквами** (латиницей или кириллицей), длиной не менее 3 символов.
5. Найдите все **хештеги** в тексте (например, #python, #Лабораторная15). Хештег начинается с `#`, за которым идут буквы, цифры или подчёркивания, без пробелов и знаков препинания.
6. Извлеките из текста все **временные метки в формате ЧЧ:ММ** (например, «09:30», «23:59»). Убедитесь, что часы от 00 до 23, минуты — от 00 до 59.
7. Найдите все **слова, содержащие хотя бы одну цифру** (например, «пароль123», «тест2»), но состоящие в основном из букв.
8. Определите, есть ли в строке **последовательность из трёх и более одинаковых символов подряд** (например, «ааа», «!!!», «555»).
9. Найдите все **ссылки, начинающиеся с http:// или https://**, но не обязательно заканчивающиеся доменом верхнего уровня (просто всё, что идёт после протокола до пробела или конца строки).
10. Извлеките из текста все **слова, заканчивающиеся на «ing»** (латиницей), независимо от регистра (например, «Running», «jumping», «ING»).

Контрольные вопросы.

1. Для чего используется модуль `re` в Python?
2. В чём разница между функциями `re.search()` и `re.match()`?
3. Как найти все совпадения шаблона в строке?
4. Что означает символ `\d` в регулярном выражении? А `\w` и `\s`?
5. Как экранировать специальные символы (например, точку или звёздочку), чтобы они воспринимались как обычные символы?
6. Что делает квантификатор `+`, а что — `*`?
7. Как указать, что шаблон должен соответствовать всей строке, а не её части?
8. Как извлечь часть строки, соответствующую определённой группе в регулярном выражении?
9. Почему при работе с кириллицей в регулярных выражениях нельзя полагаться только на `\w`?
10. Как заменить все вхождения подстроки, соответствующей шаблону, на другую строку?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 16. Разработка программ с использованием пользовательских типов данных, действия над пользовательскими типами данных.

Цель работы.

Освоить создание и использование пользовательских типов данных в Python, реализовывать основные операции над объектами (инициализация, вывод, изменение состояния).

Краткие теоретические сведения.

В Python пользовательские типы данных создаются несколькими способами. Но, так или иначе все эти способы связаны с объектно-ориентированным программированием и созданием классов. Любой тип в Python является классом. Основной способ создания пользовательского типа это создание класса и реализации принципов объектно-ориентированного программирования. Этот способ будет подробно рассмотрен в лабораторных работах №№ 31, 32 и 33.

Существуют упрощенные методы для определения пользовательских типов с использованием именованных кортежей `namedtuple` и упрощенных классов для хранения данных т.н. `dataclass`. Эти способы мы рассмотрим сейчас.

Именованные кортежи.

Именованные кортежи или кортежи с именованными полями `namedtuple` — это функция из модуля `collections`, которая создаёт класс кортежа с именованными полями. Он сочетает в себе свойства обычного кортежа (неизменяемость, лёгкость, поддержка индексации) и удобство доступа к элементам по именам, как в объекте. Для использования `namedtuple` его необходимо импортировать командой:

```
from collections import namedtuple
```

Создание типа:

```
from collections import namedtuple

Point = namedtuple('Point', ['x', 'y'])
```

Здесь:
`Point` - имя нового типа,
`['x', 'y']` — список имён полей.

Создание объекта:

```
p = Point(10, 20)
# или
p = Point(x=10, y=20)
```

Доступ к данным:

```
print(p.x)        # 10 — по имени
print(p[0])       # 10 — по индексу
```

```
print(p.y)         # 20
print(p[1])        # 20
```

Но, поскольку в создании этого пользовательского типа используется кортеж, то значения полей данных в созданном объекте менять нельзя. Следующая команда вызовет ошибку

```
p.x = 5  # Ошибка! Объект неизменяем.
```

Поскольку объект namedtuple является кортежем, ему доступны все операции с кортежами, а так же доступно его использование там, где ожидается кортеж. Объект `namedtuple` идеально подходит для простых структур данных, где не нужны методы или изменяемое состояние.

Упрощенные классы для хранения данных.

`dataclass` — это декоратор из модуля `dataclasses` (доступен в Python с версии 3.7), который автоматически генерирует «стандартный» код для классов, предназначенных в первую очередь для хранения данных.
Пример использования:

```
from dataclasses import dataclass

@dataclass
class Book:
    title: str
    author: str
    pages: int = 0

b = Book("1984", "Рэй Бредбери", 328)
b.author = "Джордж Оруэлл"
print(b)  # Book(title='1984', author='Джордж Оруэлл',
pages=328)
print(b == Book("1984", "Джордж Оруэлл", 328))  # True
```

Объект `dataclass` так же можно сделать неизменяемым, если при объявлении класса указать пометку `(frozen=True)`

```
from dataclasses import dataclass

@dataclass(frozen=True)
class Book:
    title: str
    author: str
    pages: int = 0

b = Book("1984", "Рэй Бредбери", 328)
b.author = "Джордж Оруэлл" # Здесь будет ошибка!
print(b)  # Book(title='1984', author='Джордж Оруэлл',
pages=328)
print(b == Book("1984", "Джордж Оруэлл", 328))  # True
```

Задания к лабораторной работе

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Создайте `namedtuple` «Точка» с полями `x` и `y`. Создайте две точки: A(1, 2) и B(4, 6). Найдите расстояние между ними по формуле Евклида.
2. Создайте `namedtuple` «Студент» с полями: `имя`, `группа`, `средний_балл`. Создайте список из трёх студентов. Выведите данные самого успешного студента (с максимальным средним баллом).
3. Создайте `namedtuple` «Книга» с полями: `название`, `автор`, `год`. Создайте объект книги и выведите его в формате: «"Название", автор — Автор, год издания — Год».
4. Создайте `namedtuple` «Продукт» с полями: `наименование`, `цена`, `количество`. Создайте два продукта и вычислите общую стоимость (цена × количество) для каждого.
5. Создайте `namedtuple` «Дата» с полями: `день`, `месяц`, `год`. Создайте дату своего рождения. Используйте метод `_asdict()`, чтобы преобразовать её в словарь, и выведите результат.
6. Создайте `namedtuple` «Автомобиль» с полями: `марка`, `модель`, `год_выпуска`. Создайте объект автомобиля. С помощью `_replace()` измените год выпуска на 2025 и выведите новый объект.
7. Создайте `namedtuple` «Координата» с полями `широта` и `долгота`. Создайте координату Москвы (55.7558, 37.6176). Распакуйте её в две переменные и выведите отдельно широту и долготу.
8. Создайте `namedtuple` «Заказ» с полями: `id`, `товар`, `статус`. Создайте заказ со статусом «в обработке». Проверьте, равен ли статус строке «доставлен», и выведите соответствующее сообщение.
9. Создайте `namedtuple` «Работник» с полями: `фамилия`, `должность`, `зарплата`. Создайте список из четырёх работников. Найдите суммарную зарплату всех работников.
10. Создайте `namedtuple` «RGB» с полями `red`, `green`, `blue`. Создайте цвет (255, 128, 0). Проверьте, что все значения находятся в диапазоне от 0 до 255, и выведите корректность цвета.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Создайте `dataclass` «Сотрудник» с полями: `имя` (str), `должность` (str), `зарплата` (float). Создайте объект сотрудника и **измените его зарплату** на новое значение. Выведите обновлённые данные.
2. Создайте `dataclass` «Товар» с полями: `название`, `цена`, `в_наличии` (bool). Добавьте метод `apply_discount(self, percent)`, который уменьшает цену на указанный процент. Примените скидку 10 % к товару и выведите новую цену.
3. Создайте `dataclass` «Книга» с полями: `автор`, `название`, `страницы`. Создайте объект книги. **Измените название** и **увеличьте количество страниц** на 50. Выведите обновлённую информацию.
4. Создайте `dataclass` «Авто» с полями: `марка`, `модель`, `пробег` (int). Добавьте метод `drive(self, km)`, который увеличивает пробег на заданное число километров. Вызовите метод для добавления 150 км.
5. Создайте `dataclass` «Студент» с полями: `фамилия`, `курс`, `средний_балл`. Создайте список из трёх студентов. Найдите студента с самым высоким баллом и **повысьте его средний балл** на 0.5 (но не выше 5.0).

6. Создайте `dataclass` «Прямоугольник» с полями: `ширина`, `высота`. Добавьте метод `area(self)`, возвращающий площадь. Создайте объект, **измените ширину**, затем вычислите и выведите новую площадь.
7. Создайте `dataclass` «Пользователь» с полями: `логин`, `email`, `активен` (bool). Создайте пользователя. **Измените статус** `активен` на `True` и выведите сообщение: «Пользователь [логин] активирован».
8. Создайте `dataclass` «Счёт» с полями: `номер`, `баланс` (float). Добавьте метод `deposit(self, amount)`, увеличивающий баланс. Создайте счёт, внесите 1000 рублей, выведите новый баланс.
9. Создайте `dataclass` «Фильм» с полями: `название`, `год`, `рейтинг`. Создайте фильм. **Измените рейтинг** на основе пользовательского ввода (например, 8.5). Убедитесь, что рейтинг — число от 0 до 10.
10. Создайте `dataclass` «Точка» с параметром `frozen=True` и полями `x`, `y`. Попробуйте **изменить** `x` — объясните, почему это вызывает ошибку. Затем создайте **новый объект** с помощью `_replace()` (подсказка: для `dataclass` с `frozen=True` можно использовать `dataclasses.replace()`).

Контрольные вопросы.

1. Что такое `namedtuple` и для чего он используется в Python?
2. Почему объекты, созданные с помощью `namedtuple`, нельзя изменять?
3. Как получить доступ к полям `namedtuple` — по имени или по индексу? Можно ли использовать оба способа?
4. Как преобразовать объект `namedtuple` в словарь?
5. Как объявить пользовательский тип данных с помощью `dataclass` и указать типы его полей?
6. Как сделать объект, созданный с помощью `dataclass`, неизменяемым?
7. Можно ли изменять значения полей у объекта `dataclass`, если он не помечен как неизменяемый? Приведите пример.
8. Как преобразовать объект `dataclass` в словарь?
9. В чём основное различие между `namedtuple` и `dataclass` по возможностям и гибкости?
10. Как создать новый объект `dataclass` с изменённым значением одного поля, если исходный объект неизменяем (`frozen=True`)?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 17. Разделение программы на функции: выделение логических блоков и оформление их как отдельных функций.

Цель работы.

Научиться выделять логически завершённые фрагменты кода в отдельные функции, правильно оформлять их с учётом входных и выходных данных, а также использовать функции для повышения читаемости, структурированности и повторного использования кода.

Краткие теоретические сведения.

Функция в Python — это именованный блок кода, который выполняет определённую задачу. Функции позволяют разбить программу на независимые части, упрощают отладку и делают код более понятным.

Функция объявляется с помощью ключевого слова def, может принимать параметры (входные данные) и возвращать результат с помощью return. Если return не указан, функция возвращает None.

Пример:

```
def square(x):
     return x * x
```

Обратите внимание на отступ команды `return`. Таким образом обозначается принадлежность блока команд функции. Команда `return` возвращает результат работы функции для дальнейшего применения его в программе. Если не указать `return` функция вернет тип None.

Правильно оформленная функция:

- решает одну конкретную задачу,
- имеет понятное имя,
- не зависит от внешнего контекста (работает только со своими аргументами),
- возвращает результат, а не печатает его (если это не функция вывода).

При использовании функций необходимо учитывать области видимости переменных. Крайне нежелательно использовать глобальные переменные, в том числе и внутри функций.

Разделение программы на функции помогает избежать дублирования кода, упрощает тестирование и позволяет многократно использовать один и тот же блок в разных частях программы.

Пример правильной реализации функции:

```
import re

def is_valid_email(email):
#Проверяет, соответствует ли строка формату email-адреса.
     pattern = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
     return bool(re.match(pattern, email))

# Основная часть программы
email = input("Введите email: ").strip()

if is_valid_email(email):
    print("Email корректный.")
```

```
else:
    print("Email некорректный.")
```

Программа предлагает пользователю ввести адрес электронной почты и производит проверку на валидность адреса. Адрес должен соответветствовать формату `имя@домен.домен`. Проверка производится в функции с помощью регулярного выражения. Функция возвращает True если адрес валидный и False если нет. Функция проверки адреса решает одну конкретную задачу, имеет понятное имя, не зависит от внешнего контекста и возвращает результат.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

В алгоритме необходимо выделить ту часть, которую целесообразно оформить в виде функции. Записать словесный алгоритм.

1. Составить алгоритм программы, в котором необходимо запросить у пользователя ФИО, потом эти ФИО разделить на отдельные части (фамилию, имя, отчество), получить результат обработки и вывести ответ.
2. Составить алгоритм программы, в котором необходимо запросить у пользователя почтовый адрес, потом эти данные проверить на соответствие формату email, получить результат обработки и вывести ответ.
3. Составить алгоритм программы, в котором необходимо запросить у пользователя телефонный номер, потом эти данные нормализовать к стандартному формату (+7 XXX XXX-XX-XX), получить результат обработки и вывести ответ.
4. Составить алгоритм программы, в котором необходимо запросить у пользователя дату рождения в формате ДД.ММ.ГГГГ, потом эти данные извлечь год и определить, является ли он високосным, получить результат обработки и вывести ответ.
5. Составить алгоритм программы, в котором необходимо запросить у пользователя строку с оценками через пробел, потом эти данные вычислить средний балл, получить результат обработки и вывести ответ.
6. Составить алгоритм программы, в котором необходимо запросить у пользователя адрес сайта (URL), потом эти данные извлечь доменное имя, получить результат обработки и вывести ответ.
7. Составить алгоритм программы, в котором необходимо запросить у пользователя пароль, потом эти данные проверить на надёжность (длина, наличие цифр, букв и спецсимволов), получить результат обработки и вывести ответ.
8. Составить алгоритм программы, в котором необходимо запросить у пользователя предложение на русском языке, потом эти данные подсчитать количество слов, начинающихся с заглавной буквы, получить результат обработки и вывести ответ.
9. Составить алгоритм программы, в котором необходимо запросить у пользователя два целых числа через пробел, потом эти данные найти их наибольший общий делитель, получить результат обработки и вывести ответ.
10. Составить алгоритм программы, в котором необходимо запросить у пользователя название файла с расширением, потом эти данные извлечь расширение файла, получить результат обработки и вывести ответ.

Задание 2.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Напишите функцию, которая принимает целое число и проверяет, является ли оно простым. Основная программа должна запрашивать у пользователя число и выводить результат проверки.
2. Напишите функцию, которая принимает строку и подсчитывает количество гласных букв (включая кириллицу и латиницу). Программа запрашивает строку и выводит результат.
3. Напишите функцию, которая принимает вещественное число — радиус круга — и возвращает его площадь. Используйте константу π из модуля math. Программа запрашивает радиус и выводит площадь.
4. Напишите функцию, которая принимает строку — пароль — и проверяет его надёжность: длина не менее 8 символов, наличие хотя бы одной заглавной буквы, одной строчной, одной цифры и одного спецсимвола (!@#$%^&*). Программа запрашивает пароль и выводит, подходит он или нет.
5. Напишите функцию, которая принимает строку — предложение из слов, разделённых пробелами — и возвращает новую строку, в которой каждое слово перевёрнуто, а порядок слов сохранён. Программа запрашивает строку и выводит результат.
6. Напишите функцию, которая принимает строку в формате "месяц год" (например, "2 2024"), где месяц — целое число от 1 до 12, а год — целое число. Функция должна вернуть количество дней в указанном месяце с учётом високосного года. Программа запрашивает такую строку и выводит результат.
7. Напишите функцию, которая принимает строку, содержащую два целых числа, разделённых пробелом (например, "48 18"), и возвращает их наибольший общий делитель. Программа запрашивает строку и выводит результат.
8. Напишите функцию, которая принимает строку — телефонный номер, который может содержать цифры, пробелы, дефисы и начинаться с 8 или +7. Функция должна проверить корректность номера (должно быть 11 цифр после нормализации) и вернуть номер в формате +7 XXX XXX-XX-XX или None, если номер недопустим. Программа запрашивает номер и выводит результат.
9. Напишите функцию, которая принимает строку, содержащую целые числа от 2 до 5, разделённые пробелами (оценки), и возвращает средний балл, округлённый до двух знаков после запятой. Программа запрашивает строку и выводит результат.
10. Напишите функцию, которая принимает строку — email-адрес — и возвращает доменную часть (всё после символа @). Если в строке нет @ или после него ничего нет, функция должна вернуть None. Программа запрашивает email и выводит домен или сообщение об ошибке.

Контрольные вопросы.

1. Зачем выделять части программы в отдельные функции?
2. Какие преимущества даёт использование функций при написании кода?
3. Что такое параметр функции и чем он отличается от аргумента?
4. Может ли функция в Python не возвращать значение? Что она возвращает в этом случае?
5. Как правильно выбрать, какую часть кода оформить как функцию?
6. Почему хорошая функция должна решать только одну задачу?
7. Можно ли вызывать одну функцию внутри другой? Приведите пример ситуации, где это уместно.
8. Что происходит, если передать в функцию аргумент другого типа, чем ожидается?
9. Как избежать дублирования кода с помощью функций?
10. В чём разница между локальными и глобальными переменными, и почему лучше не использовать глобальные переменные внутри функций?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 18. Разработка функций с передачей аргументов разных типов.

Цель работы.

Освоить написание функций, принимающих аргументы различных типов (числа, строки, списки, кортежи, словари), научиться корректно обрабатывать такие аргументы внутри функции и возвращать результаты с учётом их типов.

Краткие теоретические сведения.

В Python функции могут принимать аргументы любых типов, так как язык является динамически типизированным. Это означает, что при объявлении параметров не требуется указывать их тип (хотя можно использовать аннотации для читаемости).

Функция может работать с:

- Числами — выполнять арифметические операции;
- Строками — конкатенация, поиск, замена и т.д.;
- Списками и кортежами — перебор, изменение (для списков), срезы;
- Словарями — доступ к значениям по ключам, итерация по парам;
- Смешанными типами — например, функция может принимать число и строку одновременно.

Важно учитывать особенности передачи. Изменяемые объекты (списки, словари) передаются по ссылке: изменения внутри функции повлияют на оригинал. Неизменяемые объекты (числа, строки, кортежи) передаются по значению: изменения внутри функции не затрагивают оригинал.

Пример:

```
def process_data(name, scores, multiplier=1):
    total = sum(scores) * multiplier
    return f"{name}: итог = {total}"
```

Еще один пример этой же функции:

```
def process_data(name: str, scores: list, multiplier: int = 1):
    total = sum(scores) * multiplier
    return f"{name}: итог = {total}"
```

Обратите внимание, во втором случае функция объявлена с указанием типов аргументов. Такое указание типов называется "аннотацией" и продиктовано оно не требованиями языка, а для повышения документированности и читаемости кода. В Python аннотации типов не влияют на выполнение программы — интерпретатор их игнорирует. Они не обеспечивают проверку типов во время выполнения и не ограничивают, какие значения можно передать в функцию.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0. В задании указано, что должна делать функция. Решение о содержании программы, способ получения исходных данных студент принимает и определяет сам.

1. Напишите программу, в которой объявлена функция, принимающая: название товара (строка), количество (целое число), цену за единицу (вещественное число). Функция должна возвращать общую стоимость покупки.
2. Напишите программу, в которой объявлена функция, принимающая: имя пользователя (строка), возраст (целое число), рост в метрах (вещественное число), флаг подписки на рассылку (булево значение). Функция должна возвращать приветственное сообщение с краткой информацией о пользователе.
3. Напишите программу, в которой объявлена функция, принимающая: название города (строка), текущая температура (вещественное число), влажность в процентах (целое число). Функция должна возвращать строку с кратким описанием погоды.
4. Напишите программу, в которой объявлена функция, принимающая: модель устройства (строка), год выпуска (целое число), цена (вещественное число), наличие гарантии (булево значение). Функция должна возвращать описание устройства с указанием, действует ли гарантия.
5. Напишите программу, в которой объявлена функция, принимающая: название курса (строка), количество часов (целое число), средняя оценка студентов (вещественное число). Функция должна возвращать рекомендацию: «Рекомендуется» (если оценка ≥ 4.0) или «Требует доработки».
6. Напишите программу, в которой объявлена функция, принимающая: имя персонажа (строка), уровень (целое число), здоровье (вещественное число), статус «в бою» (булево значение). Функция должна возвращать строку с текущим состоянием персонажа.
7. Напишите программу, в которой объявлена функция, принимающая: название ресторана (строка), средний чек (вещественное число), количество отзывов (целое число). Функция должна возвращать рейтинг в виде текста: «Популярный» (если отзывов ≥ 100), иначе «Новый».
8. Напишите программу, в которой объявлена функция, принимающая: название фильма (строка), длительность в минутах (целое число), IMDb-рейтинг (вещественное число), доступен ли для просмотра (булево значение). Функция должна возвращать карточку фильма с пометкой «Доступен» или «Недоступен».
9. Напишите программу, в которой объявлена функция, принимающая: название задачи (строка), приоритет (целое число), процент выполнения (вещественное число). Функция должна возвращать строку с оценкой прогресса: «Завершено», «В работе» или «Не начата».
10. Напишите программу, в которой объявлена функция, принимающая: название продукта (строка), срок годности в днях (целое число), текущая температура хранения (вещественное число), требует ли холодильника (булево значение). Функция должна возвращать рекомендацию: «Хранение в норме» или «Нарушены условия хранения».

Задание 2.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0. В задании указано, что должна делать функция. Решение о содержании программы, способ получения исходных данных студент принимает и определяет сам.

1. Напишите программу, в которой создаются список и кортеж из целых чисел (не более 10 элементов каждый). Объявите функцию, принимающую эти два аргумента. Внутри функции удалите все чётные числа из одного из аргументов. Функция должна вернуть количество удалённых элементов. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.
2. Напишите программу, в которой создаются список строк и кортеж строк (по 3–5 элементов). Объявите функцию, принимающую эти аргументы. Внутри функции

добавьте к каждому элементу одного из аргументов суффикс "_new". Функция должна вернуть длину самого длинного слова в обновлённом списке. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.

3. Напишите программу с небольшим списком оценок (целые от 2 до 5) и кортежем названий предметов. Объявите функцию, принимающую эти аргументы. Внутри функции замените все оценки ниже 3 на 3 в одном из аргументов. Функция должна вернуть средний балл после исправления. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.
4. Напишите программу, в которой заданы список имён и кортеж фамилий. Объявите функцию, принимающую эти аргументы. Внутри функции удалите из одного из аргументов все имена, начинающиеся на гласную букву. Функция должна вернуть количество оставшихся имён. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.
5. Напишите программу с списком цен (вещественные числа) и кортежем названий товаров. Объявите функцию, принимающую эти аргументы. Внутри функции примените скидку 10 % ко всем ценам в одном из аргументов. Функция должна вернуть сумму всех цен после скидки. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.
6. Напишите программу, в которой заданы список логических значений и кортеж целых чисел. Объявите функцию, принимающую эти аргументы. Внутри функции инвертируйте все значения в одном из аргументов (True → False, False → True). Функция должна вернуть количество True после инверсии. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.
7. Напишите программу с небольшим списком слов и кортежем-шаблоном (например, ("a", "b", "c")). Объявите функцию, принимающую эти аргументы. Внутри функции переверните каждое слово в одном из аргументов (например, "мир" → "рим"). Функция должна вернуть самое длинное перевёрнутое слово. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.
8. Напишите программу, в которой создаются список координат X (вещественные числа) и кортеж координат Y. Объявите функцию, принимающую эти аргументы. Внутри функции округлите все значения в одном из аргументов до целых чисел. Функция должна вернуть сумму новых значений. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.
9. Напишите программу с списком email-адресов и кортежем доменов. Объявите функцию, принимающую эти аргументы. Внутри функции удалите из одного из аргументов все адреса, не содержащие символ '@'. Функция должна вернуть количество оставшихся адресов. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.
10. Напишите программу, в которой заданы список дат (в виде строк, например "12.05.2024") и кортеж времён (например "09:30"). Объявите функцию, принимающую эти аргументы. Внутри функции замените в одном из аргументов все точки на дефисы (например, "12.05.2024" → "12-05-2024"). Функция должна вернуть первую изменённую дату. Программа выводит оба аргумента до вызова, после вызова и результат вызова функции.

Контрольные вопросы.

1. В чём разница между изменяемыми и неизменяемыми типами данных в Python? Приведите примеры каждого.
2. Что произойдёт с кортежем, если попытаться изменить его элемент внутри функции?
3. Почему изменения списка внутри функции влияют на исходный список вне функции?

4. Как передать в функцию несколько аргументов разных типов? Нужно ли указывать их типы при объявлении функции?
5. Может ли функция принимать одновременно список и кортеж? Как они будут обрабатываться внутри функции?
6. Что возвращает функция, если в ней отсутствует оператор `return`?
7. Как избежать неожиданного изменения исходного списка при передаче его в функцию?
8. Можно ли изменить содержимое словаря, переданного в функцию? Почему?
9. Как проверить тип аргумента внутри функции, если он может быть разным?
10. Почему важно понимать, какие объекты передаются по ссылке, а какие — по значению, при работе с функциями?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 19. Разработка функций с передачей произвольного числа аргументов.

Цель работы.

Освоить написание функций, принимающих произвольное количество аргументов, научиться использовать механизмы *args и **kwargs для обработки переменного числа позиционных и именованных аргументов.

Краткие теоретические сведения.

В Python функции могут принимать переменное число аргументов с помощью специальных конструкций

`*args` — собирает все лишние позиционные аргументы в кортеж. Имя `args` условно и может быть любым, главное — звёздочка.

Пример:

```
def sum_all(*args):
     return sum(args)

sum_all(1, 2, 3, 4)  # args = (1, 2, 3, 4)
```

`**kwargs` — собирает все лишние именованные аргументы в словарь. Имя `kwargs` также условно; главное — две звёздочки.

Пример:

```
def print_info(**kwargs):
     for key, value in kwargs.items():
          print(f"{key}: {value}")
print_info(name="Анна", age=25)
# kwargs = {"name": "Анна", "age": 25}
```

Эти механизмы позволяют создавать гибкие функции, которые можно вызывать с разным количеством параметров, не зная их точного числа заранее. Также возможна комбинация: обычные параметры, `*args`, `**kwargs` — в строго определённом порядке.

Пример полной сигнатуры:

```
def func(a, b, *args, **kwargs):
     ...
```

Такой подход широко используется в библиотеках, декораторах, конфигурационных функциях и других случаях, где требуется максимальная гибкость при вызове.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0. В задании указано, что должна делать функция. Решение о содержании программы, способ получения исходных данных студент принимает и определяет сам.

Вариант 1

1. Напишите функцию, принимающую произвольное количество чисел и возвращающую их сумму.
2. Напишите функцию, принимающую произвольное количество именованных параметров и выводящую их в формате «ключ: значение».
3. Напишите функцию с обязательным параметром title, затем *args (список авторов) и **kwargs (дополнительные данные, например год, жанр). Функция должна возвращать строку с описанием книги.

Вариант 2

1. Напишите функцию, принимающую произвольное количество строк и возвращающую самую длинную из них.
2. Напишите функцию, принимающую произвольные именованные параметры и возвращающую словарь только с теми парами, где значение — число.
3. Напишите функцию с обязательным параметром name, затем *args (оценки студента) и **kwargs (дополнительная информация: группа, email). Функция возвращает анкету студента.

Вариант 3

1. Напишите функцию, принимающую произвольное количество чисел и возвращающую их среднее арифметическое.
2. Напишите функцию, принимающую произвольные именованные параметры и возвращающую список ключей, значения которых — строки.
3. Напишите функцию с обязательным параметром product, затем *args (характеристики: цвет, размер и т.д.) и **kwargs (цена, наличие). Функция формирует описание товара.

Вариант 4

1. Напишите функцию, принимающую произвольное количество целых чисел и возвращающую количество чётных из них.
2. Напишите функцию, принимающую произвольные именованные параметры и возвращающую новый словарь, отсортированный по значениям.
3. Напишите функцию с обязательным параметром event, затем *args (участники) и **kwargs (дата, место, статус). Функция возвращает краткое описание события.

Вариант 5

1. Напишите функцию, принимающую произвольное количество строк и возвращающую их, объединённые через запятую.
2. Напишите функцию, принимающую произвольные именованные параметры и проверяющую, есть ли среди значений хотя бы одно None.
3. Напишите функцию с обязательным параметром command, затем *args (аргументы команды) и **kwargs (флаги, например verbose=True). Функция возвращает строку вызова команды.

Вариант 6

1. Напишите функцию, принимающую произвольное количество чисел и возвращающую максимальное и минимальное из них в виде кортежа.

2. Напишите функцию, принимающую произвольные именованные параметры и возвращающую сумму всех числовых значений.
3. Напишите функцию с обязательным параметром user_id, затем *args (роли пользователя) и **kwargs (настройки: theme, lang). Функция возвращает профиль пользователя.

Вариант 7

1. Напишите функцию, принимающую произвольное количество логических значений и возвращающую True, если все — True.
2. Напишите функцию, принимающую произвольные именованные параметры и возвращающую только те пары, где ключ начинается с буквы «a» или «A».
3. Напишите функцию с обязательным параметром filename, затем *args (дополнительные теги) и **kwargs (метаданные: author, created). Функция возвращает описание файла.

Вариант 8

1. Напишите функцию, принимающую произвольное количество чисел и возвращающую список только положительных.
2. Напишите функцию, принимающую произвольные именованные параметры и преобразующую все строковые значения в верхний регистр.
3. Напишите функцию с обязательным параметром url, затем *args (параметры запроса) и **kwargs (заголовки: user_agent, auth_token). Функция возвращает строку, имитирующую HTTP-запрос.

Вариант 9

1. Напишите функцию, принимающую произвольное количество строк и возвращающую количество строк, содержащих букву «a».
2. Напишите функцию, принимающую произвольные именованные параметры и возвращающую словарь без пар, где значение — пустая строка.
3. Напишите функцию с обязательным параметром project, затем *args (задачи) и **kwargs (статус, дедлайн, ответственный). Функция возвращает краткое резюме проекта.

Вариант 10

1. Напишите функцию, принимающую произвольное количество чисел и возвращающую их произведение.
2. Напишите функцию, принимающую произвольные именованные параметры и возвращающую булево значение: True, если все значения — строки.
3. Напишите функцию с обязательным параметром device, затем *args (подключённые модули) и **kwargs (настройки: ip, port, active). Функция возвращает конфигурацию устройства.

Контрольные вопросы.

1. Для чего используется конструкция *args в определении функции?
2. Какой тип данных получает параметр, обозначенный как *args, внутри функции?
3. Что такое **kwargs и когда его удобно применять?

4. Какой тип данных представляет собой переменная, обозначенная как **kwargs, внутри функции?
5. В каком порядке должны указываться параметры в сигнатуре функции: обычные, *args, **kwargs? Почему?
6. Можно ли вызвать функцию с *args, передав ей список? Как это сделать корректно?
7. Как передать словарь в функцию, использующую **kwargs, при вызове?
8. Чем отличается поведение функции при использовании *args от передачи обычного списка в качестве одного аргумента?
9. Может ли функция одновременно использовать и *args, и **kwargs? Приведите пример.
10. Как избежать ошибки, если в функции с **kwargs ожидается наличие определённого ключа, но он не был передан?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 20. Разработка функций с реализацией классических рекурсивных алгоритмов: факториал, числа Фибоначчи, НОД (алгоритм Евклида).

Цель работы.

Освоить принципы реализации рекурсивных алгоритмов на примере вычисления факториала, чисел Фибоначчи и наибольшего общего делителя (НОД) по алгоритму Евклида и других. Понять особенности рекурсивного подхода к решению задач.

Краткие теоретические сведения.

Рекурсия — это метод определения функции через саму себя. Классические рекурсивные алгоритмы (факториал, числа Фибоначчи, НОД) служат фундаментальными примерами, демонстрирующими как ясность и лаконичность рекурсивного кода, так и его потенциальные недостатки (например, избыточные вычисления или переполнение стека). Умение работать с такими алгоритмами развивает навыки декомпозиции задач, понимания базовых случаев и рекуррентных соотношений, а также закладывает основу для освоения более сложных рекурсивных структур и алгоритмов (деревья, динамическое программирование, разделяй-и-властвуй).

Но, необходимо понимать, что в рекурсии использется самовызов функции, иногда количество таких вызовов может доходить до нескольких десятков тысяч, например, при вычислении больших факториалов. Этот процесс потребляет много ресурсов в стеке вызовов и при определенной глубине рекурсии произойдет переполнение стека и среда выполнения аварийно завершиться. В Python есть механизм защиты, ограничивающий глубину рекурсии по умолчанию. Можно изменить допустимую глубину рекурсии, но нужно понимать последствия. По умолчанию в Python глубина рекурсии равна 1000. Изменить допустимую глубину рекурсии в Python можно следующими командами:

```
import sys

# Установить лимит глубины рекурсии в 5000
sys.setrecursionlimit(5000)
```

Альтернативой использованию рекурсии будет итерация, т.е. алгоритм когда повторение осуществляется с помощью циклов (`for, while`), без создания новых фреймов в стеке — используется один и тот же блок памяти. В большинстве случаев рекурсию можно заменить итерацией (и наоборот), но выбор зависит от контекста, требований к памяти, скорости и читаемости кода. Итерация менее требовательна к ресурсам, но чаще всего требует большего объема кода.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0. В задании указано, что должна делать функция. Решение о содержании программы, способ получения исходных данных студент принимает и определяет сам.

1. Реализуйте функцию для вычисления факториала натурального числа с помощью рекурсии. Добавьте проверку корректности входного значения (неотрицательное целое).
2. Напишите рекурсивную функцию, возвращающую n-е число Фибоначчи. Сравните время выполнения для n = 35 с итеративной версией.

3. Реализуйте рекурсивный алгоритм Евклида для нахождения НОД двух целых положительных чисел. Протестируйте на парах: (48, 18), (1071, 462).
4. Создайте рекурсивную функцию возведения числа a в целую неотрицательную степень n с использованием метода «разделяй и властвуй» (быстрое возведение).
5. Решите задачу о Ханойских башнях для n дисков. Функция должна выводить последовательность ходов в формате: "Переместить диск X из A в B."
6. Напишите рекурсивную функцию для вычисления биномиального коэффициента C(n, k). Убедитесь, что она корректно обрабатывает граничные случаи (k = 0, k = n).
7. Реализуйте рекурсивную функцию, подсчитывающую количество разбиений натурального числа n на сумму натуральных слагаемых (порядок слагаемых не важен).
8. Создайте рекурсивную функцию, проверяющую, является ли строка палиндромом (игнорируя регистр и пробелы — по желанию усложнить).
9. Напишите рекурсивную функцию, вычисляющую сумму всех элементов списка целых чисел. Не используйте встроенные функции sum() или циклы.
10. Реализуйте рекурсивную функцию, возвращающую сумму цифр заданного натурального числа. Например: digit_sum(1234) → 10.

Контрольные вопросы.

1. Что такое рекурсия и какие два обязательных условия должна содержать рекурсивная функция?
2. Почему при вычислении чисел Фибоначчи простой рекурсивный алгоритм неэффективен? Как можно улучшить его производительность?
3. В чём состоит основная идея рекурсивного алгоритма Евклида для нахождения НОД?
4. Почему при рекурсивном вычислении факториала или чисел Фибоначчи может возникнуть ошибка переполнения стека, и как глубина рекурсии связана с объёмом используемой памяти?
5. Как изменить максимальную глубину рекурсии в Python и к чему может привести её чрезмерное увеличение?
6. Почему задача о Ханойских башнях удобно решается с помощью рекурсии? Опишите базовый и рекуррентный случаи.
7. Можно ли любой рекурсивный алгоритм переписать в итеративной форме? Приведите пример и аргументируйте.
8. Как рекурсивно вычислить сумму элементов списка, не используя циклы и встроенные функции?
9. Как работает рекурсивная проверка строки на палиндром? Какие параметры целесообразно передавать в функцию?
10. Приведите пример задачи, где рекурсия даёт более понятное и лаконичное решение, чем итерация.

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 21. Разработка программ с использованием рекурсивной обработки структур данных: обход вложенных списков или словарей, вычисление суммы/глубины/количества элементов.

Цель работы.

Освоить методы рекурсивной обработки вложенных структур данных (списков и словарей), научиться реализовывать функции для вычисления суммы числовых элементов, определения глубины вложенности и подсчёта количества элементов в произвольно устроенных структурах.

Краткие теоретические сведения.

Рекурсия особенно эффективна при работе с иерархическими или древовидными структурами данных, такими как вложенные списки или словари, где заранее неизвестна глубина вложенности. В таких случаях итеративный подход требует явного управления стеком или очередью, тогда как рекурсивное решение естественно отражает структуру данных: функция обрабатывает текущий уровень и рекурсивно вызывает саму себя для каждого вложенного подэлемента.

Типичные задачи включают:

- подсчёт всех атомарных значений (например, чисел);
- вычисление общей суммы числовых значений;
- определение максимальной глубины вложенности;
- поиск элемента по ключу (в случае словарей).

Рекурсивная обработка позволяет писать компактный и легко читаемый код, адаптирующийся к произвольной структуре входных данных.

Пример вложенного списка в Python:

```
nested_list = [
    1,
    [2, 3],
    [4, [5, 6], 7],
    [[8, 9], 10, [11, [12]]]
]
```

Этот список содержит числа и другие списки на разных уровнях вложенности. Максимальная глубина вложенности — 4 (элемент 12 находится внутри четырёх уровней списков).

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0. Атомарные типы это типы которые не являются списками. Контейнерные типы это списки, словари и множества.

1. Напишите рекурсивную функцию `sum_numbers(data)`, которая вычисляет сумму всех числовых элементов (`int`, `float`) в произвольно вложенном списке. Игнорируйте нечисловые элементы.
2. Реализуйте функцию `count_elements(data)`, подсчитывающую общее количество атомарных (не являющихся списком) элементов во вложенном списке.

3. Создайте рекурсивную функцию `max_depth(data)`, возвращающую максимальную глубину вложенности списка. Пустой список имеет глубину 1, отсутствие вложенности — глубину 0.
4. Напишите функцию `flatten(data)`, которая рекурсивно преобразует вложенный список в одномерный («выравнивает» его), сохраняя порядок элементов.
5. Реализуйте функцию `all_strings(data)`, возвращающую список всех строковых значений, содержащихся во вложенном списке любого уровня вложенности.
6. Напишите рекурсивную функцию `contains_value(data, value)`, проверяющую, содержится ли заданное значение (не список) где-либо во вложенном списке.
7. Создайте функцию `replace_values(data, old, new)`, которая рекурсивно заменяет все вхождения значения `old` на `new` и возвращает новый список (исходный не изменяется).
8. Реализуйте функцию `get_numbers(data)`, возвращающую список всех числовых значений (`int`, `float`) из вложенного списка в порядке их следования.
9. Напишите функцию `is_homogeneous(data, types)`, проверяющую, что все атомарные элементы списка принадлежат только указанным типам (например, только `int` или только `str`). Списки как контейнеры не учитываются.
10. Создайте рекурсивную функцию `reverse_nested(data)`, которая «зеркально» разворачивает структуру вложенного списка: каждый список (на любом уровне) инвертируется по порядку элементов, включая вложенные подсписки.

Контрольные вопросы.

1. Почему при рекурсивной обработке вложенного списка в Python важно проверять тип элемента с помощью `isinstance(x, list)`?
2. Как рекурсивно определить, что элемент списка является «атомарным» (т.е. не списком)?
3. В чём разница между поверхностным и глубоким копированием при рекурсивной обработке вложенных списков? Нужно ли это учитывать при реализации функции замены значений?
4. Почему использование глобальных переменных в рекурсивных функциях для подсчёта или накопления данных считается плохой практикой? Как лучше организовать возврат результата?
5. Как изменится реализация функции `flatten`, если допустить, что список может содержать не только другие списки, но и кортежи?
6. Как рекурсивно вычислить глубину вложенности пустого списка? Обоснуйте принятое значение.
7. Почему в Python рекурсивные функции для обработки глубоко вложенных структур могут вызывать `RecursionError`, и как этого избежать?
8. Можно ли реализовать обход вложенного списка без явной рекурсии? Если да, то какими средствами (укажите подход)?
9. Как поведёт себя рекурсивная функция `sum_numbers`, если в списке окажутся значения типа `bool`? Почему и как этого избежать?
10. Приведите пример вложенного списка, для которого итеративный подход к выравниванию (`flatten`) сложнее, чем рекурсивный.

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 22. Создание текстового и двоичного файла. Чтение из файла. Изменение данных в файле.

Цель работы.

Освоить основные операции работы с файлами в Python: создание текстовых и двоичных файлов, запись и чтение данных, а также методы изменения содержимого файла.

Краткие теоретические сведения.

В Python работа с файлами осуществляется с помощью встроенной функции open(), которая возвращает объект файла. Файлы могут открываться в различных режимах:

'r' — чтение (по умолчанию),
'w' — запись (создаёт новый файл или перезаписывает существующий),
'a' — добавление данных в конец файла,
'r+' — чтение и запись,
'b' — двоичный режим (используется вместе с другими, например 'rb', 'wb').

Текстовые файлы хранят данные в виде последовательности символов, кодируемых (обычно UTF-8). Они удобны для хранения читаемой информации: логов, конфигураций, исходного кода и т.п.

Двоичные файлы хранят данные в «сыром» виде — как последовательность байтов. Используются для изображений, аудио, сериализованных объектов (например, через pickle) и других нетекстовых данных.

Изменение данных в файле напрямую («на месте») возможно только при открытии в режиме 'r+' и требует точного позиционирования курсора (seek()). Чаще всего для изменения содержимого читают данные в память, модифицируют их и перезаписывают файл заново.

Понимание различий между текстовым и двоичным вводом-выводом, а также корректное управление ресурсами (закрытие файлов, использование with) — ключевые навыки при работе с постоянными данными.

Пример работы с текстовым файлом:

```
# Запись текста в файл
with open('example.txt', 'w', encoding='utf-8') as f:
    f.write("Привет, мир!\n")
    f.write("Это текстовый файл.")

# Чтение текста из файла
with open('example.txt', 'r', encoding='utf-8') as f:
    content = f.read()
    print(content)
```

Файл открывается в текстовом режиме (`'w'`, `'r'`). Указана кодировка `utf-8` (рекомендуется явно). Конструкция `with` гарантирует автоматическое закрытие файла.

Пример работы с двоичным файлом:

```
# Запись байтов в файл
data = b"Hello, binary world!"  # bytes-объект
with open('example.bin', 'wb') as f:
    f.write(data)
# Чтение байтов из файла
with open('example.bin', 'rb') as f:
```

```
binary_content = f.read()
print(binary_content)          # b'Hello, binary world!'
print(binary_content.decode('utf-8')) #"Hello, binary world!"
```

Используются режимы `'wb'` (запись байтов) и `'rb'` (чтение байтов). Данные должны быть типа `bytes` при записи. Для получения строки из байтов применяется `.decode()`.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Создайте текстовый файл input.txt, содержащий несколько строк произвольного текста. Напишите программу, которая читает содержимое файла и записывает его в новый файл output.txt в обратном порядке (последняя строка — первой).
2. Создайте текстовый файл data.txt, заполнив его несколькими строками текста. Прочитайте файл, подсчитайте количество строк, слов и символов (без учёта пробелов) и запишите эти три числа в файл stats.txt.
3. Создайте текстовый файл numbers.txt, записав в него по одному целому числу в каждой строке. Напишите программу, которая читает числа из файла, находит их сумму и записывает результат в файл sum.txt.
4. Создайте текстовый файл text.txt, включив в него как обычные строки, так и пустые строки. Напишите программу, которая удаляет все пустые строки и сохраняет оставшиеся в файл cleaned.txt.
5. Создайте текстовый файл log.txt. Напишите программу, которая при каждом запуске дописывает в конец этого файла текущую дату и время в формате YYYY-MM-DD HH:MM:SS.
6. Создайте текстовый файл input.txt, содержащий предложения с повторяющимся словом «старый». Напишите программу, которая заменяет все вхождения «старый» на «новый» и сохраняет результат в файл replaced.txt.
7. Создайте текстовый файл lines.txt, заполнив его произвольными строками. Напишите программу, которая создаёт файл numbered.txt, где каждая строка из исходного файла предваряется её номером (начиная с 1) и точкой, например: 1. Первая строка.
8. Создайте текстовый файл source.txt, включив в него несколько строк, часть из которых повторяется. Напишите программу, которая формирует файл unique.txt, содержащий только уникальные строки в порядке их первого появления.
9. Создайте два текстовых файла: part1.txt и part2.txt, заполнив каждый своим текстом. Напишите программу, которая объединяет их содержимое в один файл combined.txt, разделяя части строкой ---.
10. Создайте текстовый файл data.txt, записав в него строки в произвольном порядке. Напишите программу, которая сортирует строки по алфавиту и записывает отсортированный результат в файл sorted.txt.

Задание 2.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Создайте текстовый файл info.txt. Откройте произвольный JPEG-файл в двоичном режиме, прочитайте его первые 10 байт и запишите их шестнадцатеричное представление в файл info.txt.

2. Скопируйте содержимое любого JPEG-файла в новый двоичный файл copy.jpg, используя побайтовое чтение и запись.
3. Определите размер любого JPEG-файла в байтах, открыв его в двоичном режиме и считав всю информацию в память (или с помощью метода seek/tell). Запишите размер в файл size.txt.
4. Создайте программу, которая разбивает JPEG-файл на две части: первую половину байтов записывает в part1.bin, вторую — в part2.bin.
5. Объедините два двоичных файла part1.bin и part2.bin (полученные из предыдущего задания) обратно в один JPEG-файл restored.jpg и убедитесь, что изображение открывается корректно.
6. Прочитайте JPEG-файл в виде байтов, замените каждый байт на его дополнение до 255 (255 - byte), и запишите результат в файл inverted.bin. (Полученный файл не будет корректным JPEG, но задание демонстрирует побайтовую обработку.)
7. Создайте программу, которая читает JPEG-файл и подсчитывает, сколько раз встречается байт со значением 0xFF. Результат запишите в файл count_ff.txt.
8. Откройте JPEG-файл в двоичном режиме и запишите его содержимое в обратном порядке (последний байт — первым) в файл reversed.bin. (Файл не будет валидным изображением — это упражнение на работу с байтами.)
9. Прочитайте JPEG-файл и запишите его содержимое в новый двоичный файл backup.jpg, добавив в начало файла 4 байта, представляющих длину исходного файла (в формате int, little-endian). Затем напишите программу, которая читает backup.jpg, извлекает эту длину и проверяет, совпадает ли она с реальным размером оставшейся части файла.
10. Создайте программу, которая добавляет в конец JPEG-файла произвольную текстовую «подпись» (например, ваше имя) в двоичном виде, а затем читает файл заново и извлекает эту подпись, записывая её в signature.txt. Убедитесь, что исходное изображение по-прежнему открывается (стандарт JPEG игнорирует данные после маркера конца файла).

Контрольные вопросы.

1. В чём основное различие между текстовым и двоичным режимами открытия файла в Python?
2. Почему при работе с двоичными файлами нельзя использовать метод `readline()` так же, как с текстовыми?
3. Какие режимы открытия файла позволяют одновременно читать и записывать данные? Приведите пример для текстового и двоичного файла.
4. Зачем используется конструкция `with open(...) as f:` вместо простого вызова `open()` и последующего `close()`?
5. Почему добавление произвольных данных в конец JPEG-файла обычно не нарушает его отображение, в отличие от изменения начала файла?
6. Как определить размер файла, не считывая всё его содержимое в память? Приведите фрагмент кода.
7. Что произойдёт, если открыть JPEG-файл в текстовом режиме и попытаться прочитать его содержимое?
8. Как правильно записать целое число (например, длину файла) в начало двоичного файла так, чтобы его можно было потом точно прочитать?
9. Почему при побайтовой обработке двоичного файла важно учитывать тип данных (например, `bytes` vs `int`) в Python?
10. Можно ли изменить произвольный байт в середине файла без перезаписи всего файла? Если да, то какими средствами это делается?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 23. Создание форматированного файла (CSV, JSON). Чтение из файла. Изменение данных в файле.

Цель работы.

Освоить создание, чтение и модификацию структурированных данных в форматах CSV и JSON с использованием стандартных средств Python; научиться преобразовывать данные между внутренними структурами программы и внешними файлами.

Краткие теоретические сведения.

CSV (Comma-Separated Values) — это текстовый формат хранения табличных данных, где каждая строка представляет запись, а поля внутри строки разделены запятыми (или другим разделителем, например точкой с запятой). CSV-файлы широко используются для обмена данными между программами (например, таблицами Excel, базами данных, аналитическими системами) благодаря простоте и универсальности.

JSON (JavaScript Object Notation) — это текстовый формат обмена данными, основанный на синтаксисе объектов JavaScript. Он поддерживает вложенные структуры: объекты (аналог словарей) и массивы (аналог списков), а также примитивные типы (строки, числа, логические значения, null). JSON стал де-факто стандартом для передачи структурированных данных в веб-API, конфигурационных файлах и промежуточном хранении.

Зачем нужно уметь работать с CSV и JSON? Эти форматы позволяют сохранять и передавать данные вне программы, обеспечивая совместимость между разными языками и системами. Они поддерживаются большинством приложений (таблицы, базы данных, веб-сервисы), что делает их незаменимыми в реальных проектах. Умение корректно читать, изменять и записывать такие файлы — базовый навык для анализа данных, автоматизации и интеграции ПО.

В Python для работы с этими форматами используются стандартные модули:

- `csv` — для чтения и записи CSV-файлов,
- `json` — для сериализации и десериализации JSON-данных.

Пример содержимого CSV-файла (например, `people.csv`) в текстовом виде:

```
Имя,Возраст,Город
Анна,25,Москва
Борис,30,Санкт-Петербург
Вера,22,Новосибирск
```

Каждая строка соответствует одной записи. Поля внутри строки разделены запятыми. Первая строка часто содержит заголовки — названия столбцов. Файл сохраняется в кодировке UTF-8 (особенно если есть неанглийские символы), и не содержит специальных управляющих символов — только текст.

Пример работы с csv-файлом:

Запись данных в CSV-файл:

```
import csv
```

```
data = [
    ["Имя", "Возраст", "Город"],
    ["Анна", 25, "Москва"],
    ["Борис", 30, "Санкт-Петербург"],
    ["Вера", 22, "Новосибирск"]
]

with open('people.csv', 'w', encoding='utf-8', newline='') as f:
    writer = csv.writer(f)
    writer.writerows(data)
```

Параметр newline='' предотвращает добавление лишних пустых строк в Windows. Параметр encoding='utf-8' обеспечивает корректную запись кириллицы.

Чтение данных из CSV-файла:

```
import csv

with open('people.csv', 'r', encoding='utf-8') as f:
    reader = csv.reader(f)
    for row in reader:
        print(row)
```

Вывод:

```
['Имя', 'Возраст', 'Город']
['Анна', '25', 'Москва']
['Борис', '30', 'Санкт-Петербург']
['Вера', '22', 'Новосибирск']
```

Чтение как словарей (если есть заголовок):

```
import csv

with open('people.csv', 'r', encoding='utf-8') as f:
    reader = csv.DictReader(f)
    for row in reader:
        print(f"{row['Имя']} — {row['Возраст']} лет,
{row['Город']}")
```

Вывод:

```
Анна — 25 лет, Москва
Борис — 30 лет, Санкт-Петербург
Вера — 22 лет, Новосибирск
```

Пример содержимого JSON-файла (например, `people.json`) в текстовом виде:

```
 [
  {
    "Имя": "Анна",
```

```
    "Возраст": 25,
    "Город": "Москва"
  },
  {
    "Имя": "Борис",
    "Возраст": 30,
    "Город": "Санкт-Петербург"
  },
  {
    "Имя": "Вера",
    "Возраст": 22,
    "Город": "Новосибирск"
  }
]
```

JSON-файл представляет собой текст в строгом формате:

- данные организованы в объекты (в фигурных скобках `{}`), которые похожи на словари,
- или в массивы (в квадратных скобках `[]`), аналогичные спискам,
- строки всегда заключаются в двойные кавычки,
- допустимы числа, логические значения (`true`, `false`) и `null`.

Файл легко читается как человеком, так и программой, и широко используется для хранения и передачи структурированных данных.

Примеры работы с JSON-файлами в Python: запись, чтение и обновление данных.

Запись данных в JSON-файл:

```
import json

data = [
    {"Имя": "Анна", "Возраст": 25, "Город": "Москва"},
    {"Имя": "Борис", "Возраст": 30, "Город": "Санкт-Петербург"},
    {"Имя": "Вера", "Возраст": 22, "Город": "Новосибирск"}
]

with open('people.json', 'w', encoding='utf-8') as f:
    json.dump(data, f, ensure_ascii=False, indent=2)
```

Параметр `ensure_ascii=False` позволяет сохранять кириллицу без экранирования. Параметр `indent=2` делает файл читаемым (с отступами).

Чтение данных из JSON-файла:

```
import json

with open('people.json', 'r', encoding='utf-8') as f:
    data = json.load(f)

for person in data:
    print(f"{person['Имя']} — {person['Возраст']} лет,
{person['Город']}")
```

Изменение данных и перезапись файла:

```
import json

# Чтение
with open('people.json', 'r', encoding='utf-8') as f:
    data = json.load(f)

# Изменение
data.append({"Имя": "Дмитрий", "Возраст": 28, "Город":
"Екатеринбург"})

# Запись обратно
with open('people.json', 'w', encoding='utf-8') as f:
    json.dump(data, f, ensure_ascii=False, indent=2)
```

Эти примеры показывают базовые операции: сериализацию (запись), десериализацию (чтение) и модификацию структурированных данных в формате JSON.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

Вариант 1

1. Создайте текстовый файл students.csv с полями: ФИО, группа, средний балл. Заполните его минимум тремя записями. Напишите программу, которая читает файл, увеличивает средний балл каждого студента на 0.5 (но не более 5.0), сохраняет изменения в тот же файл и выводит обновлённые данные на экран.
2. Создайте файл books.json, содержащий список книг с полями: название, автор, год издания. Заполните его минимум тремя записями. Напишите программу, которая добавляет новую книгу, перезаписывает файл и выводит весь список книг на экран.

Вариант 2

1. Создайте файл products.csv с колонками: товар, цена, количество. Заполните его. Программа должна прочитать файл, вычислить общую стоимость каждого товара (цена × количество), добавить эту информацию как новую колонку, сохранить обновлённый файл и вывести его содержимое.
2. Создайте файл employees.json со списком сотрудников: имя, должность, зарплата. Программа должна прочитать файл, повысить зарплату всех «менеджеров» на 10 %, записать изменения обратно и вывести обновлённый список.

Вариант 3

1. Создайте файл weather.csv с данными: дата, температура, осадки (да/нет). Заполните на 5–7 дней. Программа должна прочитать файл, найти день с максимальной температурой, вывести его на экран и записать в конец файла строку «Самый тёплый день: [дата]».

2. Создайте файл movies.json со списком фильмов: название, режиссёр, рейтинг. Программа должна удалить все фильмы с рейтингом ниже 7.0, перезаписать файл и вывести оставшиеся фильмы.

Вариант 4

1. Создайте файл tasks.csv с полями: задача, статус (выполнено/не выполнено). Заполните. Программа должна прочитать файл, пометить все задачи как «выполнено», сохранить изменения и вывести обновлённый список.
2. Создайте файл countries.json со странами: название, столица, население. Программа должна отсортировать страны по населению по убыванию, перезаписать файл и вывести результат.

Вариант 5

1. Создайте файл orders.csv: номер заказа, клиент, сумма. Заполните. Программа должна прочитать файл, найти общую сумму всех заказов, добавить в конец строки «ИТОГО: [сумма]» и вывести всё содержимое.
2. Создайте файл users.json с пользователями: логин, email, активен (true/false). Программа должна деактивировать пользователя с заданным логином (ввести с клавиатуры), сохранить изменения и вывести обновлённый список.

Вариант 6

1. Создайте файл inventory.csv: наименование, категория, остаток. Заполните. Программа должна прочитать файл, вывести только товары с остатком меньше 5, и записать этот список в новый файл low_stock.csv.
2. Создайте файл events.json с событиями: название, дата (в формате "YYYY-MM-DD"), место. Программа должна отфильтровать события, прошедшие до текущей даты, удалить их из списка, обновить файл и вывести актуальные события.

Вариант 7

1. Создайте файл grades.csv: предмет, студент, оценка. Заполните. Программа должна прочитать файл, вычислить среднюю оценку по каждому предмету, вывести результат на экран и добавить эти строки в конец файла.
2. Создайте файл projects.json со списком проектов: название, срок (в днях), завершён (true/false). Программа должна добавить поле «недель» = срок / 7 для каждого проекта, перезаписать файл и вывести обновлённые данные.

Вариант 8

1. Создайте файл contacts.csv: имя, телефон, email. Заполните. Программа должна прочитать файл, удалить контакт по имени (ввести с клавиатуры), сохранить изменения и вывести оставшиеся контакты.
2. Создайте файл recipes.json с рецептами: название, ингредиенты (список), время приготовления. Программа должна добавить новый рецепт, записать файл и вывести все рецепты.

Вариант 9

1. Создайте файл sales.csv: менеджер, месяц, сумма продаж. Заполните. Программа должна найти менеджера с наибольшей суммой продаж за указанный месяц (ввести с клавиатуры), вывести его имя и сохранить результат в файл top_manager.txt.
2. Создайте файл cars.json с автомобилями: марка, модель, год выпуска. Программа должна отфильтровать автомобили старше 2010 года, сохранить только их в файл и вывести на экран.

Вариант 10

1. Создайте файл library.csv: книга, автор, в наличии (да/нет). Заполните. Программа должна изменить статус «в наличии» на противоположный для указанной книги (ввести название с клавиатуры), сохранить файл и вывести обновлённый каталог.
2. Создайте файл scores.json с игроками: имя, очки, уровень. Программа должна увеличить уровень всем, у кого очков больше 1000, перезаписать файл и вывести обновлённую таблицу.

Контрольные вопросы.

1. В чём принципиальное различие между структурой данных в CSV и JSON? Какой из форматов поддерживает вложенность, а какой — нет?
2. Почему при записи CSV-файла в Windows рекомендуется указывать параметр newline=''? Что может произойти, если его не использовать?
3. Какие типы данных Python могут быть напрямую сериализованы в JSON, а какие требуют дополнительной обработки?
4. Зачем в функции json.dump() используется параметр ensure_ascii=False? Что будет, если его не указать при записи кириллицы?
5. Как прочитать CSV-файл так, чтобы каждая строка представлялась в виде словаря с ключами из заголовка? Какой класс для этого используется?
6. Можно ли хранить в одном CSV-файле данные разной структуры (например, строки с разным числом полей)? Как это повлияет на чтение?
7. Как обновить значение одного поля у конкретной записи в JSON-файле, не перезаписывая весь файл целиком? Опишите последовательность действий.
8. Почему JSON не поддерживает комментарии, а CSV — не имеет стандарта для метаданных? Как это влияет на их использование в реальных проектах?
9. Как обработать ситуацию, когда CSV-файл содержит запятые внутри текстовых полей (например, «Иванов, А.Б.»)? Как Python-модуль csv с этим справляется?
10. В каких случаях предпочтительнее использовать CSV, а в каких — JSON? Приведите по одному примеру задачи для каждого формата.

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 24. Разработка программ с использованием однонаправленных списков типа стек и очередь.

Цель работы.

Изучить принципы организации и функционирования линейных динамических структур данных — стека и очереди; реализовать их на основе однонаправленных списков; освоить основные операции (добавление, удаление, просмотр) и применить эти структуры для решения практических задач.

Краткие теоретические сведения.

В программировании однонаправленный список — это структура данных, состоящая из узлов, каждый из которых содержит данные и ссылку (указатель) на следующий узел. Такая организация позволяет эффективно добавлять и удалять элементы без необходимости предварительного выделения фиксированного объёма памяти (в отличие от массивов).

На основе однонаправленных списков можно реализовать две важные абстрактные структуры данных: стек и очередь.

Стек (stack) — это структура данных, работающая по принципу «последним пришёл — первым вышел» (LIFO, Last In – First Out).

Элементы добавляются и удаляются только с одного конца, называемого вершиной стека. Основные операции:

добавить элемент x на вершину стека;
удалить и вернуть элемент с вершины;
посмотреть верхний элемент без удаления;
проверить, пуст ли стек.

Примеры использования: обработка вложенных скобок, отмена действий (undo), рекурсивные алгоритмы, обход дерева в глубину.

В Python стек удобно реализуется с помощью обычного списка (list):

```
# Создание пустого стека
stack = []

# Добавление элемента (push)
stack.append(10)
stack.append(20)

# Удаление и получение верхнего элемента (pop)
top = stack.pop()  # вернёт 20

# Просмотр вершины без удаления
if stack:
    top = stack[-1]

# Проверка на пустоту
if not stack:
    print("Стек пуст")
```

Очередь (queue) — это структура данных, работающая по принципу «первым пришёл — первым вышел» (FIFO, First In – First Out).

Элементы добавляются в конец очереди, а извлекаются из начала.
Основные операции:

добавить элемент x в конец очереди;
удалить и вернуть элемент из начала;
посмотреть первый элемент без удаления;
проверить, пуста ли очередь.

Примеры использования: планирование задач, обработка запросов, поиск в ширину (BFS), буферизация данных.

При реализации на однонаправленном списке для эффективной работы необходимо хранить указатели как на голову (начало), так и на хвост (конец) списка.

Для очереди не рекомендуется использовать обычный список с pop(0), потому что удаление из начала списка потребует количества операций равного количеству элементов списка. При удалении первого элемента потребуется сдвинуть все оставшиеся элементы левее. Вместо этого используется специальный класс `collections.deque (double-ended queue)`, который поддерживает добавление и удаление с обоих концов за одну операцию.

```
from collections import deque

# Создание пустой очереди
queue = deque()

# Добавление в конец (enqueue)
queue.append(10)
queue.append(20)

# Удаление из начала (dequeue)
first = queue.popleft()  # вернёт 10

# Просмотр первого элемента без удаления
if queue:
    first = queue[0]

# Проверка на пустоту
if not queue:
    print("Очередь пуста")
```

Обе структуры не позволяют произвольного доступа к элементам по индексу — доступ возможен только через строго определённые концы. Это ограничение обеспечивает предсказуемое поведение и высокую эффективность базовых операций.

Задания к лабораторной работе.

Задание 1.

Это задание для всех вариантов одинаково.

1. Реализуйте стек на базе списков Python. Не используйте стандартные методы. Для реализации стека объявите в глобальной области видимости список с заданной размерностью. В этой же области видимости создайте указатель на вершину стека. Определите функции push() – поместить элемент в стек, pop() – извлечь элемент из

стека, peek() – просмотреть верхний элемент стека без извлечения. Обрабатывайте ситуацию когда стек заполнен или когда стек пуст.

2. Реализуйте однонаправленную очередь на базе списков Python. Не используйте `collections.deque (double-ended queue).` Для реализации очереди объявите в глобальной области видимости список с заданной размерностью. В этой же области видимости создайте указатель на начало очереди. Определите функции q_get() – извлечь первый [0] элемент очереди, q_push() – поместить элемент в конец очереди, q_top() – посмотреть первый элемент очереди без извлечения. При извлечении первого элемента очереди необходимо передвинуть все элементы списка на единицу индекса влево. Обрабатывайте ситуацию когда очередь заполнена или когда очередь пуста.

Задание 2.

Используя написанные вами в задании 1 функции выполните задание 2. В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

Вариант 1

1. Реализуйте стек и напишите функцию, проверяющую, является ли строка корректной последовательностью скобок (), [], {}). Используйте стек для отслеживания открывающих скобок.
2. Реализуйте очередь и смоделируйте очередь в кассу: добавьте 5 клиентов, затем обслужите трёх первых. Выведите оставшихся.

Вариант 2

1. С помощью стека реализуйте функцию, которая переворачивает строку (без использования срезов).
2. С помощью очереди реализуйте «круговую» замену: поместите в очередь числа от 1 до N, затем перенесите первые K элементов в конец. Выведите результат.

Вариант 3

1. Напишите программу, использующую стек для вычисления факториала числа N через эмуляцию рекурсии (сохраняя промежуточные состояния в стеке).
2. Смоделируйте очередь печати: добавьте 4 задания на печать, затем извлеките и выведите их в порядке поступления.

Вариант 4

1. Используя стек, реализуйте функцию, которая удаляет все соседние дубликаты в строке (например, "abbaca" → "ca").
2. Создайте очередь и реализуйте функцию, возвращающую количество элементов в ней, не разрушая саму очередь (без вызова len()).

Вариант 5

1. Реализуйте стек и напишите функцию, которая определяет, является ли слово палиндромом (без индексов — только через стек).
2. С помощью очереди реализуйте простой буфер: добавьте N сообщений, но ограничьте размер очереди M элементами (при превышении — удалять самые старые).

Вариант 6

1. Используя стек, преобразуйте десятичное число в двоичное (остатки от деления на 2 складывайте в стек, затем извлекайте).
2. Смоделируйте очередь задач: добавьте задачи с приоритетом (просто текст), выполните первую, затем добавьте новую и покажите текущее состояние очереди.

Вариант 7

1. Реализуйте стек и напишите функцию, которая проверяет, можно ли получить последовательность B из последовательности A с помощью операций push/pop.
2. Смоделируйте очередь обслуживания: добавьте клиентов с именами, затем реализуйте функцию, которая находит клиента по имени и перемещает его в конец очереди (остальные сохраняют порядок).

Вариант 8

1. Используя стек, реализуйте функцию, которая удаляет все элементы из стека, кроме максимального (максимум должен остаться на дне).
2. Создайте очередь и напишите функцию, которая перемещает все чётные числа в конец очереди, сохраняя относительный порядок как чётных, так и нечётных.

Вариант 9

1. С помощью стека реализуйте функцию, которая «отменяет» последние N действий (представьте действия как строки; отмена = извлечение из стека).
2. Смоделируйте очередь студентов на зачёт: добавьте 6 имён, затем вызовите троих по одному и выведите, кто остался ждать.

Вариант 10

1. Реализуйте стек и напишите функцию, которая находит и выводит минимальный элемент в стеке, просматривая все его элементы (разрешается использовать временный стек для восстановления исходного состояния).
2. С помощью очереди реализуйте циклический буфер фиксированного размера: при добавлении в полную очередь удаляется самый старый элемент.

Контрольные вопросы.

1. В чём принципиальное различие между стеком и очередью по способу добавления и извлечения элементов?
2. Почему для реализации очереди в Python рекомендуется использовать collections.deque, а не обычный список?
3. Как с помощью стека можно проверить правильность расстановки скобок в арифметическом выражении?
4. Приведите пример задачи, где использование стека естественно отражает логику решения.
5. Как реализовать обход дерева по уровням с использованием очереди?
6. Что произойдёт, если попытаться извлечь элемент из пустого стека или пустой очереди? Как этого избежать?
7. Можно ли с помощью одной очереди реализовать поведение стека? Если да, то какими операциями?

8. Как сохранить исходное содержимое стека, если нужно найти в нём минимальный или максимальный элемент?
9. Почему стек удобно использовать для эмуляции рекурсивных вызовов в итеративных алгоритмах?
10. Как с помощью очереди смоделировать систему, в которой новые заявки обрабатываются строго в порядке поступления?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 25. Разработка программ с использованием двусвязных списков.

Цель работы.

Изучить устройство и принципы работы двусвязного списка как динамической структуры данных; реализовать основные операции — добавление, удаление, поиск и обход — в обе стороны; закрепить навыки проектирования пользовательских структур данных на языке Python.

Краткие теоретические сведения.

Двусвязный список — это линейная структура данных, состоящая из узлов, каждый из которых содержит:

- поле с данными,
- ссылку на следующий узел (next),
- ссылку на предыдущий узел (prev).

Благодаря наличию двух ссылок в каждом узле, двусвязный список поддерживает обход в обоих направлениях: от начала к концу и от конца к началу. Это отличает его от однонаправленного списка, где движение возможно только вперёд.

Основные преимущества двусвязного списка:

- эффективное удаление элемента при наличии ссылки на него (не требуется поиск предыдущего узла);
- возможность итерации назад;
- гибкость при вставке/удалении в произвольной позиции (если известен соседний узел).

Типичные операции:

- добавление в начало / конец / заданную позицию;
- удаление по значению или по позиции;
- поиск элемента;
- прямой и обратный обход;
- получение элемента по индексу (менее эффективно, чем в массиве — O(n)).

В отличие от массивов, двусвязный список не обеспечивает прямого доступа к элементу по индексу, но позволяет эффективно изменять структуру без перемещения больших блоков памяти. В Python двусвязные списки не встроены в язык напрямую, поэтому их реализуют вручную с помощью классов, что развивает понимание указателей, ссылок и управления памятью на логическом уровне.

В стандартной библиотеке Python нет встроенной структуры данных «двусвязный список» в виде отдельного типа, как, например, list или dict. Однако функциональность двусвязного списка реализована внутри класса collections.deque (double-ended queue).

Как работает deque:

deque — это обобщённая очередь с эффективным добавлением и удалением элементов с обоих концов.

Внутренне она реализована как динамический массив фиксированного размера (блоков), но логически ведёт себя как двусвязный список: каждый блок содержит указатели на предыдущий и следующий, что позволяет быстро менять начало и конец.

Операции append(), appendleft(), pop(), popleft() выполняются за одну операцию.

Хотя deque не предоставляет прямого доступа к «узлам» или указателям (как в классическом двусвязном списке на C++), он обеспечивает те же алгоритмические свойства: эффективную работу с обоими концами, что делает его практической заменой двусвязного списка в большинстве задач.

В отличие от массивов, двусвязный список не обеспечивает прямого доступа к элементу по индексу, но позволяет эффективно изменять структуру без перемещения больших блоков памяти. В Python двусвязные списки не встроены в язык напрямую, поэтому их реализуют вручную с помощью классов, что развивает понимание указателей, ссылок и управления памятью на логическом уровне.

Задания к лабораторной работе.

Задание 1.

Это задание для всех вариантов одинаково.

Реализуйте двунаправленный список на базе стандартных списков Python. Для этого опишите пользовательский тип NodeList в нем должно быть 3 поля. Одно поле хранит индекс предыдущего элемента, второе поле хранит индекс следующего элемента, третье поле хранит значение элемента списка. Для реализации двунаправленного списка объявите в глобальной области видимости список состоящий из элементов типа NodeList с заданной размерностью. Определите функции push_start() – поместить элемент в начало списка, push_end() – поместить элемент в конец списка, push() – поместить элемент в заданную позицию в списке, если такой позиции нет, вернуть ошибку. Так же описать функции которые вернут элемент из начала, конца или произвольного места списка. Функцию которая вернет индекс элемента списка по значению. Функцию которая удалит элемент списка из начала, конца или из произвольного места. Функцию которая возвращает фактическую длину списка. Необходимо обрабатывать ситуацию когда список нулевой длины.

Задание 2.

Используя написанные вами в задании 1 функции выполните задание 2. В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

1. Создайте двусвязный список и заполните его числами от 1 до 10. Выведите содержимое списка сначала в прямом, затем в обратном порядке.
2. Заполните список строками: "январь", "февраль", ..., "декабрь". Удалите из списка все месяцы, содержащие букву «р». Выведите оставшиеся элементы.
3. Создайте список из 8 случайных целых чисел. Найдите и выведите минимальное и максимальное значения, обходя список только один раз в каждом направлении.
4. Заполните список символами слова "программирование". Удалите все гласные буквы. Выведите результат в прямом порядке.
5. Создайте два двусвязных списка: один — с чётными числами от 2 до 20, другой — с нечётными от 1 до 19. Объедините их в третий список так, чтобы сначала шли все чётные, затем все нечётные. Выведите итоговый список.
6. Заполните список числами: 5, 12, 7, 9, 3, 15. Вставьте число 10 после первого элемента, большего 8. Выведите получившийся список.
7. Создайте список из 6 элементов. Поменяйте местами первый и последний элементы, изменив только ссылки между узлами (без замены данных). Выведите список до и после обмена.

8. Заполните список строками: "кот", "ток", "мир", "рим", "пир". Удалите все палиндромы. Выведите оставшиеся элементы.
9. Создайте список из 10 чисел. Переместите все отрицательные числа в конец списка, сохранив относительный порядок как отрицательных, так и неотрицательных элементов. Выведите результат.
10. Заполните список последовательностью: 1, 2, 3, 4, 5, 4, 3, 2, 1. Проверьте, является ли список симметричным («зеркальным»), сравнивая элементы с начала и конца одновременно. Выведите «Да» или «Нет».

Контрольные вопросы.

1. Чем двусвязный список отличается от однонаправленного и как это влияет на операции вставки и удаления?
2. Почему при удалении узла из двусвязного списка необходимо обновлять ссылки как у предыдущего, так и у следующего узлов?
3. Как организован обход двусвязного списка в обратном направлении? Что для этого должно быть обязательно реализовано?
4. Можно ли эффективно получить элемент по индексу в двусвязном списке? Обоснуйте ответ.
5. Какие преимущества даёт наличие ссылки на предыдущий узел при реализации операции удаления?
6. Что происходит с памятью в Python при удалении узла из двусвязного списка? Нужно ли явно «освобождать» память?
7. Как реализовать вставку нового узла в начало двусвязного списка? Какие поля и у каких узлов нужно изменить?
8. Почему при работе с двусвязным списком важно корректно обрабатывать случаи пустого списка, одного элемента и границ (начало/конец)?
9. Можно ли с помощью двусвязного списка эффективно реализовать стек или очередь? Если да, то какие концы использовать для каждой структуры?
10. Как проверить, что двусвязный список действительно «связан» правильно в обоих направлениях? Предложите способ диагностики ошибок в связях.

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 26. Создание классов и объектов. Инкапсуляция и управление доступом к данным.

Цель работы.

Изучить основы объектно-ориентированного программирования (ООП) в Python: научиться создавать классы и объекты, понимать принцип инкапсуляции, освоить механизмы управления доступом к атрибутам и методам, а также применять эти знания для построения надёжных и поддерживаемых программ.

Краткие теоретические сведения.

Класс и объект

Класс — это шаблон или «чертёж» для создания объектов. Он определяет: атрибуты (данные, характеризующие состояние), методы (функции, определяющие поведение).

Объект — это экземпляр класса, то есть конкретная сущность, созданная на основе класса, обладающая собственным состоянием и способная выполнять действия, заданные методами.

Пример объявим класс Car и создадим объект my_car:

```
class Car:
    def __init__(self, brand):
        self.brand = brand

my_car = Car("Toyota")  # my_car — объект класса Car
```

Инкапсуляция

Инкапсуляция — один из фундаментальных принципов ООП, заключающийся в объединении данных и методов, работающих с этими данными, внутри одного объекта, а также в ограничении прямого доступа к внутренним деталям реализации. Это позволяет:

скрыть сложность,
защитить данные от некорректного использования,
упростить изменение внутренней реализации без влияния на внешний код.

Управление доступом в Python

В отличие от языков вроде Java или C++, Python не имеет строгих модификаторов доступа (private, protected, public). Вместо этого используется соглашение об именовании, основанное на символах подчёркивания:

Публичные (public) атрибуты/методы
Имя без подчёркиваний: name, run().
Доступны из любого места — как внутри класса, так и вне его.
«Защищённые» (protected)
Имя начинается с одного подчёркивания: _name, _calculate().
Это сигнал разработчику: «этот атрибут предназначен для внутреннего использования, не стоит обращаться к нему напрямую вне класса или его подклассов».
Однако технически доступ остаётся открытым.
«Приватные» (private)
Имя начинается с двух подчёркиваний: __name, __validate().
Python автоматически искажает имя (name mangling): атрибут __x в классе MyClass становится _MyClass__x.
Это затрудняет (но не делает невозможным) прямой доступ извне и предотвращает случайные коллизии имён в наследниках.

Пример:

```
class BankAccount:
    def __init__(self, balance):
        self.__balance = balance  # приватный атрибут

    def deposit(self, amount):
        if amount > 0:
            self.__balance += amount

    def get_balance(self):
        return self.__balance  # контролируемый доступ через метод
```

Здесь значение баланса нельзя изменить напрямую (`account.__balance = -1000` не сработает), но можно получить через `get_balance()` и изменить только через `deposit()` — с проверкой корректности.

Свойства (properties)

Для более гибкого контроля доступа часто используют свойства — специальные методы, которые выглядят как атрибуты, но позволяют выполнять логику при чтении, записи или удалении:

```
class Temperature:
    def __init__(self, celsius=0):
        self._celsius = celsius

    @property
    def celsius(self):
        return self._celsius

    @celsius.setter
    def celsius(self, value):
        if value < -273.15:
            raise ValueError("Температура не может быть ниже
абсолютного нуля")
        self._celsius = value
```

Теперь запись t.celsius = -300 вызовет ошибку, хотя синтаксис выглядит как простое присваивание.

Конструкторы и деструкторы в Python. Конструктор.

Конструктор — это специальный метод, который автоматически вызывается при создании нового объекта класса. Его основная задача — инициализировать начальное состояние объекта, то есть задать значения его атрибутов.

В Python конструктор реализуется с помощью метода __init__ (читается как «дандер-инит»).

Пример:

```
class Person:
    def __init__(self, name, age):
        self.name = name
```

```
        self.age = age
```

При создании объекта:

```
p = Person("Анна", 25)
```

— вызывается __init__, и объект p получает атрибуты name и age.

Важно:

- __init__ не создаёт объект — он только инициализирует уже созданный интерпретатором объект.
- Фактическое создание объекта происходит в методе __new__, но в большинстве случаев его не нужно переопределять.
- У одного класса может быть только один __init__, но с помощью значений по умолчанию или *args, **kwargs можно эмулировать перегрузку.

Деструктор

Деструктор — это метод, который вызывается при уничтожении объекта, то есть когда он больше не нужен и подлежит удалению из памяти. В Python деструктор реализуется методом __del__.

Пример:

```
class Logger:
    def __init__(self, filename):
        self.filename = filename
        print(f"Журнал {filename} открыт")

    def __del__(self):
        print(f"Журнал {self.filename} закрыт (объект уничтожен)")
```

Однако важно понимать особенности работы __del__ в Python:

- Время вызова не гарантировано. Объект удаляется тогда, когда сборщик мусора (garbage collector) определяет, что на него больше нет ссылок. Это может произойти не сразу после выхода из области видимости.
- __del__ не вызывается при аварийном завершении программы (например, при sys.exit() или ошибке), поэтому нельзя полагаться на него для критически важных операций, таких как закрытие файлов или сетевых соединений.
- Лучшая практика — использовать менеджеры контекста (`with`) или явные методы вроде close() для освобождения ресурсов.

Пример надёжного подхода:

```
class FileHandler:
    def __init__(self, filename):
        self.file = open(filename, 'w')

    def close(self):
        if self.file:
            self.file.close()
            self.file = None

    def __del__(self):
        self.close()  # на случай, если забыли вызвать close()
```

```
# Но лучше:
with open('file.txt', 'w') as f:
    f.write("данные")
# файл закроется автоматически
```

Конструктор __init__ — стандартный способ инициализации объекта; используется всегда.

Деструктор __del__ — существует, но не рекомендуется для управления ресурсами из-за неопределённого времени вызова.

- Для надёжного освобождения ресурсов следует использовать явные методы или протокол менеджера контекста.

Методы класса в Python

Метод класса в Python — это функция, определённая внутри класса и предназначенная для работы с объектами этого класса или с самим классом.

Метод экземпляра работает с конкретным объектом (экземпляром). Первый параметр — `self` (ссылка на текущий объект).

Пример:

```
class MyClass:
    def __init__(self, value):
        self.value = value

    def show(self):           # обычный метод
        print("Значение:", self.value)
```

Вызов:

```
obj = MyClass(10)
obj.show()  # → Значение: 10
```

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

Вариант 1

1. Создайте класс `Book` с атрибутами: название, автор, год издания. Добавьте метод `display()`, выводящий информацию о книге.
2. Сделайте атрибуты приватными. Реализуйте свойства (`@property`) для чтения и записи года издания с проверкой: год должен быть от 0 до текущего.
3. Создайте класс `Library`, хранящий список книг. Реализуйте методы: добавить книгу, найти все книги по автору, вывести все книги.

Вариант 2

1. Создайте класс `Point` с координатами x и y. Добавьте метод `show()`, выводящий точку в формате (x, y).
2. Сделайте координаты защищёнными. Реализуйте метод `distance_to(other)`, вычисляющий расстояние до другой точки.
3. Создайте класс `Polygon`, содержащий список точек. Реализуйте метод вычисления периметра фигуры.

Вариант 3

1. Создайте класс `Car` с атрибутами: марка, модель, год выпуска. Добавьте метод `start()`, выводящий сообщение «Машина заведена».
2. Сделайте год выпуска приватным. Разрешите его изменение только через сеттер с проверкой: год не может быть больше текущего.
3. Создайте класс `Garage`, который хранит список машин. Реализуйте методы: добавить машину, найти машины указанной марки, подсчитать количество машин.

Вариант 4

1. Создайте класс `Student` с атрибутами: имя, возраст, группа. Добавьте метод `introduce()`, выводящий краткую информацию.
2. Сделайте возраст приватным. При попытке установить возраст < 0 или > 150 — вызывайте исключение.
3. Создайте класс `Group`, хранящий список студентов. Реализуйте методы: добавить студента, найти самого старшего студента, вывести всех студентов группы.

Вариант 5

1. Создайте класс `Rectangle` с длиной и шириной. Добавьте метод `area()`, возвращающий площадь.
2. Сделайте длину и ширину защищёнными. Реализуйте свойства с проверкой: значения должны быть положительными.
3. Создайте класс `Room`, содержащий список прямоугольников (например, пол, стены). Реализуйте метод подсчёта общей площади всех элементов.

Вариант 6

1. Создайте класс `BankAccount` с номером счёта и балансом. Добавьте метод `deposit(amount)` для пополнения.
2. Сделайте баланс приватным. Запретите снятие средств, если их недостаточно (метод `withdraw`).
3. Создайте класс `Bank`, управляющий несколькими счетами. Реализуйте перевод денег между счетами и поиск счёта по номеру.

Вариант 7

1. Создайте класс `Product` с названием, ценой и количеством. Добавьте метод `total_price()`, возвращающий стоимость всего товара.
2. Сделайте цену и количество приватными. При установке количества < 0 — устанавливайте 0.
3. Создайте класс `Cart`, хранящий список товаров. Реализуйте метод подсчёта общей стоимости корзины и удаления товара по названию.

Вариант 8

1. Создайте класс `Circle` с радиусом. Добавьте метод `circumference()`, возвращающий длину окружности.
2. Сделайте радиус приватным. Разрешите его изменение только через сеттер с проверкой: радиус > 0.
3. Создайте класс `FigureCollection`, хранящий разные фигуры (круги, прямоугольники). Реализуйте метод подсчёта суммарного периметра всех фигур.

Вариант 9

1. Создайте класс `Employee` с именем, должностью и зарплатой. Добавьте метод `promote(new_position)`.
2. Сделайте зарплату приватной. Реализуйте повышение зарплаты на процент через метод `raise_salary(percent)`.
3. Создайте класс `Company`, хранящий список сотрудников. Реализуйте методы: добавить сотрудника, найти всех менеджеров, рассчитать фонд оплаты труда.

Вариант 10

1. Создайте класс `Temperature` с температурой в градусах Цельсия. Добавьте метод `to_fahrenheit()`.
2. Сделайте значение приватным. Используйте свойство `celsius` с валидацией: температура ≥ −273.15.
3. Создайте класс `WeatherStation`, хранящий историю измерений (список объектов `Temperature`). Реализуйте методы: добавить измерение, найти максимальную температуру, вывести все значения в Фаренгейтах.

Контрольные вопросы.

1. Что такое класс и чем он отличается от объекта в Python?
2. Как называется метод, автоматически вызываемый при создании объекта, и какова его основная задача?
3. Зачем в Python используются приватные атрибуты (с двумя подчёркиваниями) и как они реализуются на уровне интерпретатора?
4. В чём разница между публичным, защищённым и приватным атрибутом в Python? Приведите примеры имён для каждого типа.
5. Как с помощью свойств (@property) организовать контроль при чтении и записи значения атрибута?
6. Почему не рекомендуется полагаться на метод **del** для освобождения важных ресурсов, таких как файлы или сетевые соединения?
7. Как правильно реализовать инкапсуляцию данных в классе, чтобы внешний код не мог напрямую изменять внутреннее состояние объекта?
8. Можно ли в Python полностью запретить доступ к атрибуту извне класса? Обоснуйте ответ.
9. Какую роль играет параметр self в методах экземпляра и почему его нужно указывать явно в определении метода?
10. Как методы-свойства помогают соблюдать принцип инкапсуляции при сохранении удобного синтаксиса обращения к атрибутам?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 27. Разработка программы с реализацией иерархии классов с использованием наследования.

Цель работы.

Изучить принципы наследования в объектно-ориентированном программировании; научиться проектировать иерархии классов, использовать переопределение методов, применять базовые и производные классы для повторного использования кода и моделирования реальных отношений «общее — частное».

Краткие теоретические сведения.

Наследование — один из ключевых механизмов ООП, позволяющий создавать новый класс (дочерний, или подкласс) на основе уже существующего (родительского, или базового). Дочерний класс наследует все атрибуты и методы родительского класса и может:

- расширять их (добавлять новые),
- изменять (переопределять) поведение унаследованных методов,
- вызывать родительскую реализацию при необходимости.

В Python наследование задаётся в объявлении класса:

```
class Parent:
    def greet(self):
        print("Привет из родителя")

class Child(Parent):  # Child наследует Parent
    def greet(self):
        print("Привет из потомка")
```

Ключевые понятия:

- Базовый (родительский) класс — класс, от которого наследуются другие.
- Производный (дочерний) класс — класс, который наследует функциональность.
-Переопределение метода — определение в дочернем классе метода с тем же именем, что и в родительском.
- Вызов родительского метода — осуществляется с помощью `super()`:

```
  class Child(Parent):
      def greet(self):
          super().greet()  # вызов метода родителя
          print("Дополнительно из потомка")
```

Преимущества наследования:

- Повторное использование кода — общая логика выносится в базовый класс.
-Модульность и расширяемость — легко добавлять новые типы, не изменяя существующий код.
-Полиморфизм — возможность обрабатывать объекты разных классов единообразно через общий интерфейс.

Пример иерархии:

```
Животное (Animal)
├── Птица (Bird)
└── Млекопитающее (Mammal)
    ├── Кошка (Cat)
    └── Собака (Dog)
```

Все классы наследуют общие черты от Animal, но каждый может иметь специфическое поведение.

Важно помнить: наследование следует использовать, когда между классами существует семантическое отношение «является» (is-a): «Собака является Животным», но не «Автомобиль имеет Двигатель» (это композиция, а не наследование).

Таким образом, наследование — мощный инструмент для построения гибких, читаемых и поддерживаемых программных систем.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

Вариант 1

1. Создайте класс Vehicle с атрибутами brand и model. Создайте дочерний класс Car, добавив атрибут num_doors.
2. В классе Car переопределите метод **str**, чтобы он выводил информацию в формате «Марка Model, N дверей». Вызовите родительский конструктор через super().
3. Создайте ещё один дочерний класс Motorcycle (с атрибутом engine_size). Напишите функцию, принимающую список транспортных средств и выводящую информацию о каждом с помощью print(obj).

Вариант 2

1. Создайте класс Shape с методом area(), возвращающим 0. Создайте дочерний класс Rectangle с атрибутами width и height.
2. Реализуйте метод area() в Rectangle. Добавьте класс Circle с атрибутом radius и своим методом area().
3. Создайте список из нескольких фигур (Rectangle, Circle) и вычислите общую площадь всех фигур, вызывая у каждого объекта area().

Вариант 3

1. Создайте класс Animal с атрибутом name и методом speak(), выводящим «Животное издаёт звук». Создайте класс Dog, наследующий Animal.
2. В классе Dog переопределите speak(), чтобы он выводил «Гав!». Используйте super() для вывода имени перед звуком.
3. Добавьте классы Cat («Мяу!») и Bird («Чирик!»). Напишите функцию, которая принимает список животных и вызывает у каждого speak().

Вариант 4

1. Создайте класс Employee с атрибутами name и salary. Создайте дочерний класс Manager, добавив атрибут department.
2. В классе Manager переопределите метод get_info(), который возвращает строку с именем, зарплатой и отделом. Используйте данные из родительского класса.
3. Создайте класс Developer с атрибутом programming_language. Напишите функцию, выводящую информацию обо всех сотрудниках в компании, независимо от их типа.

Вариант 5

1. Создайте класс Book с атрибутами title и author. Создайте дочерний класс EBook с дополнительным атрибутом file_size.
2. В EBook переопределите метод **str**, добавив размер файла. Вызовите родительский **init** через super().
3. Создайте класс AudioBook с атрибутом duration. Напишите функцию, которая выводит все книги (обычные, электронные, аудио) из списка, используя единый интерфейс.

Вариант 6

1. Создайте класс Appliance с атрибутом brand и методом turn_on(), выводящим «Прибор включён». Создайте класс Kettle, наследующий Appliance.
2. В Kettle переопределите turn_on(), чтобы он выводил «Чайник включён. Грею воду…».
3. Добавьте класс Microwave с собственным поведением при включении. Создайте список приборов и включите каждый по очереди.

Вариант 7

1. Создайте класс Person с атрибутами name и age. Создайте класс Student, наследующий Person, с атрибутом student_id.
2. В Student реализуйте метод introduce(), который выводит имя, возраст и номер студенческого. Используйте данные из родительского класса.
3. Создайте класс Teacher с атрибутом subject. Напишите функцию, которая принимает список людей (Student, Teacher) и вызывает у каждого introduce().

Вариант 8

1. Создайте класс Product с атрибутами name и price. Создайте класс DiscountedProduct, наследующий Product, с атрибутом discount_percent.
2. В DiscountedProduct переопределите метод get_final_price(), учитывающий скидку.
3. Создайте список из обычных и скидочных товаров. Напишите функцию, вычисляющую общую стоимость корзины.

Вариант 9

1. Создайте класс Figure с методом perimeter(), возвращающим 0. Создайте класс Triangle с тремя сторонами.
2. Реализуйте perimeter() в Triangle. Добавьте проверку корректности сторон (сумма любых двух > третьей).
3. Создайте класс Square с одной стороной и своим perimeter(). Напишите функцию, выводящую периметр каждой фигуры из списка.

Вариант 10

1. Создайте класс Device с атрибутами model и is_on (булево). Метод power_on() устанавливает is_on = True. Создайте класс Phone, наследующий Device.
2. В Phone добавьте атрибут battery_level и измените power_on(): устройство включается только если заряд > 0.
3. Создайте класс Tablet с тем же поведением. Напишите функцию, пытающуюся включить все устройства в списке и сообщающую, какие удалось включить.

Контрольные вопросы.

1. Что такое наследование в программировании и как оно записывается в определении класса в Python?
2. Какие элементы (атрибуты и методы) дочерний класс получает от родительского?
3. Можно ли в дочернем классе создавать новые атрибуты и методы, которых нет в родительском классе?
4. Зачем в конструкторе дочернего класса часто вызывают конструктор родительского с помощью super()?
5. Что произойдёт, если в дочернем классе определить метод с тем же именем, что и в родительском?
6. Как правильно вызвать метод родительского класса из метода дочернего класса?
7. Может ли один класс наследовать сразу от нескольких родительских классов в Python?
8. Почему важно, чтобы при наследовании соблюдалось логическое отношение «является» (is-a) между классами?
9. Как проверить, является ли объект экземпляром определённого класса или его подкласса?
10. Приведите пример реальной ситуации, где использование наследования упрощает структуру программы.

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 28. Разработка программы с применением полиморфизма и переопределением методов.

Цель работы.

Изучить принципы полиморфизма и переопределения методов в объектно-ориентированном программировании; научиться создавать иерархии классов, в которых дочерние классы изменяют поведение унаследованных методов; освоить написание кода, который работает с объектами разных типов через единый интерфейс.

Краткие теоретические сведения.

Полиморфизм — это возможность использовать один и тот же интерфейс (например, вызов метода с одинаковым именем) для объектов разных классов, при этом каждый объект выполняет действия по-своему. В Python полиморфизм достигается за счёт динамической типизации и наследования.

Переопределение метода — это создание в дочернем классе метода с тем же именем, что и в родительском классе. При вызове этого метода у объекта дочернего класса будет выполнена именно новая реализация, а не та, что в родителе.

Для обеспечения корректного поведения часто используется базовый (родительский) класс, в котором метод может быть либо реализован обобщённо, либо оставлен как заглушка (например, с вызовом исключения или просто пустым). Дочерние классы обязаны предоставить конкретную реализацию.

Пример:

```
class Animal:
     def make_sound(self): pass
class Dog(Animal):
     def make_sound(self): return "Гав!"
class Cat(Animal):
     def make_sound(self): return "Мяу!"
```

Функция, использующая полиморфизм: def call_animal(animal): print(animal.make_sound())

Такая функция работает с любым объектом, имеющим метод make_sound(), независимо от его конкретного типа. Это делает код гибким, легко расширяемым и удобным для поддержки.

Ключевое преимущество полиморфизма — возможность писать обобщённый код, не зависящий от конкретных классов, а только от наличия определённого интерфейса.

Задания к лабораторной работе.

Задание 1.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

Вариант 1

1. Создайте базовый класс Animal с методом make_sound(), который выводит «Неизвестный звук».
2. Создайте два дочерних класса: Dog и Cat. Переопределите в них метод make_sound() так, чтобы он выводил «Гав!» и «Мяу!» соответственно.
3. Напишите функцию, которая принимает список животных и вызывает у каждого метод make_sound(). Протестируйте её на смешанном списке объектов Dog и Cat.

Вариант 2

1. Создайте класс Shape с методом area(), возвращающим 0.
2. Создайте классы Rectangle и Circle, наследующие Shape. Переопределите area() в каждом: для прямоугольника — ширина × высота, для круга — π × радиус².
3. Создайте список фигур разных типов и вычислите общую площадь, вызывая у каждой фигуры area().

Вариант 3

1. Создайте класс Vehicle с методом start_engine(), выводящим «Двигатель запущен».
2. Создайте классы Car и Motorcycle, наследующие Vehicle. Переопределите start_engine() так, чтобы сообщения отличались: «Автомобиль заведён» и «Мотоцикл заведён».
3. Напишите функцию, которая принимает список транспортных средств и запускает двигатель у каждого, используя общий интерфейс.

Вариант 4

1. Создайте класс Employee с методом get_salary(), возвращающим фиксированную сумму.
2. Создайте классы Manager и Developer, наследующие Employee. Переопределите get_salary(): у менеджера — базовая зарплата + бонус, у разработчика — почасовая оплата × часы.
3. Создайте список сотрудников разных типов и выведите их зарплаты, вызывая у каждого get_salary().

Вариант 5

1. Создайте класс Device с методом turn_on(), выводящим «Устройство включено».
2. Создайте классы Laptop и Phone, наследующие Device. Переопределите turn_on() так, чтобы выводились разные сообщения: «Ноутбук загружается…» и «Телефон разблокирован».
3. Напишите функцию, которая принимает список устройств и включает каждое, не проверяя его тип.

Вариант 6

1. Создайте класс Figure с методом draw(), выводящим «Рисую фигуру».
2. Создайте классы Triangle и Square, наследующие Figure. Переопределите draw() так, чтобы выводились сообщения: «Рисую треугольник» и «Рисую квадрат».
3. Создайте список фигур и вызовите у каждой draw(), используя цикл и единый интерфейс.

Вариант 7

1. Создайте класс Person с методом introduce(), выводящим «Привет, я человек».
2. Создайте классы Student и Teacher, наследующие Person. Переопределите introduce(): студент говорит «Я студент, учусь в …», учитель — «Я учитель, преподаю …».
3. Создайте список людей (студентов и учителей) и вызовите у каждого introduce() в цикле.

Вариант 8

1. Создайте класс Product с методом get_info(), возвращающим строку с названием и ценой.
2. Создайте классы Book и Electronics, наследующие Product. Переопределите get_info(), добавив специфичные детали: для книги — автор, для электроники — гарантия.
3. Напишите функцию, которая выводит информацию о каждом товаре из списка, не зная их конкретного типа.

Вариант 9

1. Создайте класс Transport с методом move(), выводящим «Транспорт движется».
2. Создайте классы Airplane и Ship, наследующие Transport. Переопределите move(): «Самолёт летит» и «Корабль плывёт».
3. Создайте список транспортных средств и вызовите у каждого move(), используя полиморфизм.

Вариант 10

1. Создайте класс GameCharacter с методом attack(), выводящим «Атакую!».
2. Создайте классы Warrior и Mage, наследующие GameCharacter. Переопределите attack(): воин — «Бью мечом!», маг — «Кастую заклинание!».
3. Напишите функцию, которая заставляет группу персонажей атаковать по очереди, вызывая attack() у каждого без проверки типа.

Контрольные вопросы.

1. Что такое полиморфизм и как он проявляется при вызове методов у объектов разных классов?
2. Как в Python обеспечивается возможность вызывать один и тот же метод у объектов разных типов без проверки их конкретного класса?
3. Зачем в базовом классе определять метод, если его всё равно будут переопределять в дочерних классах?
4. Что происходит, если в дочернем классе не переопределить метод, унаследованный от родителя?
5. Как правильно организовать иерархию классов, чтобы использовать полиморфизм эффективно?
6. Можно ли вызывать переопределённый метод родительского класса из дочернего? Если да, то как?
7. Почему важно, чтобы переопределяемые методы в дочерних классах сохраняли ту же сигнатуру (набор параметров), что и в родительском?
8. Какую роль играет базовый класс при реализации полиморфизма?
9. Приведите пример ситуации, когда полиморфизм упрощает расширение программы (например, добавление нового типа объекта).
10. В чём разница между простым переопределением метода и использованием этого механизма для достижения полиморфного поведения?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

Лабораторная работа № 29. Разработка программ с использованием деревьев и бинарных деревьев.

Цель работы.

Изучить основные понятия и свойства деревьев, в частности бинарных деревьев; реализовать базовые операции — вставку, поиск, обход (в прямом, симметричном и обратном порядке); научиться применять деревья для решения практических задач хранения и обработки иерархических данных.

Краткие теоретические сведения.

Дерево — это иерархическая структура данных, состоящая из узлов, где каждый узел (кроме корневого) имеет ровно одного родителя, а корень не имеет родителя. Узлы без потомков называются листьями. Деревья широко используются для представления иерархий: файловых систем, организационных структур, синтаксических конструкций и др.

Бинарное дерево — это дерево, в котором каждый узел имеет не более двух потомков: левого и правого. Бинарные деревья особенно полезны в алгоритмах поиска и сортировки.

Особый интерес представляет бинарное дерево поиска (BST) — бинарное дерево, в котором для каждого узла выполняется условие:

все значения в левом поддереве меньше значения узла,
все значения в правом поддереве больше или равны (или строго больше — в зависимости от реализации).

Основные операции с бинарным деревом:

Вставка — добавление нового узла с сохранением свойств дерева;
Поиск — проверка наличия значения;
Удаление — более сложная операция, требующая перестройки связей;
Обходы — три классических способа:

- Прямой (pre-order): узел → левое поддерево → правое поддерево;
- Симметричный (in-order): левое поддерево → узел → правое поддерево (для BST даёт отсортированную последовательность);
- Обратный (post-order): левое поддерево → правое поддерево → узел.

Глубина (высота) дерева — это длина самого длинного пути от корня до листа. В худшем случае (вырожденное дерево, похожее на список) высота равна количеству узлов, и операции становятся O(n). В сбалансированном дереве высота — O(log n), что обеспечивает эффективность.

В Python деревья реализуются с помощью классов, где каждый узел содержит данные и ссылки на левого и правого потомков.

Задания к лабораторной работе.

Задание 1.

Это задание для всех вариантов одинаково.

Создайте класс TreeNode для представления узла бинарного дерева, содержащий атрибуты:

- data — данные узла (целое число),
- left — ссылка на левый дочерний узел,
- right — ссылка на правый дочерний узел.

На основе этого класса создайте класс BinaryTree, реализующий следующие методы:

1. insert(value) — добавляет новый узел с указанным значением в дерево, соблюдая свойства бинарного дерева поиска (все значения в левом поддереве меньше значения корня, в правом — больше или равны).
2. search(value) — возвращает True, если значение присутствует в дереве, и False в противном случае.
3. inorder() — выполняет симметричный (in-order) обход дерева и возвращает список значений узлов в порядке возрастания.

Протестируйте работу класса: создайте дерево, добавьте в него несколько чисел, выполните поиск и выведите отсортированную последовательность с помощью обхода inorder.

Задание 2.

В соответствии с вашим вариантом выполните задание. Вариант 10 соответствует значению цифры 0.

Вариант 1

1. Создайте дерево и вставьте в него числа: 50, 30, 70, 20, 40, 60, 80.
2. Найдите и выведите все значения, находящиеся в диапазоне от 25 до 65.
3. Реализуйте метод count_nodes(), возвращающий общее количество узлов в дереве.

Вариант 2

1. Заполните дерево случайными числами от 1 до 100 (10 значений).
2. Выведите все значения дерева в порядке возрастания с помощью обхода inorder.
3. Реализуйте метод find_min(), возвращающий минимальное значение в дереве.

Вариант 3

1. Создайте дерево и добавьте в него последовательность: 10, 5, 15, 3, 7, 12, 18.
2. Проверьте наличие чисел 7, 9 и 15 в дереве и выведите результаты поиска.
3. Реализуйте метод find_max(), возвращающий максимальное значение в дереве.

Вариант 4

1. Постройте дерево из элементов списка: [45, 25, 65, 15, 35, 55, 75].
2. Реализуйте метод height(), возвращающий высоту дерева (количество уровней).
3. Выведите высоту и общее число узлов.

Вариант 5

1. Создайте дерево и вставьте в него 8 чисел по вашему выбору.
2. Реализуйте метод preorder(), выполняющий прямой обход (корень → левое поддерево → правое) и возвращающий список значений.
3. Сравните результаты preorder() и inorder().

Вариант 6

1. Заполните дерево числами: 100, 50, 150, 25, 75, 125, 175.
2. Реализуйте метод postorder(), выполняющий обратный обход (левое → правое → корень).
3. Выведите результаты всех трёх обходов: preorder, inorder, postorder.

Вариант 7

1. Создайте дерево и добавьте в него 10 уникальных чисел.
2. Реализуйте метод is_empty(), возвращающий True, если дерево не содержит узлов.
3. Проверьте, пусто ли дерево до и после вставки.

Вариант 8

1. Постройте дерево из последовательности: 20, 10, 30, 5, 15, 25, 35.
2. Реализуйте метод contains_all(values), проверяющий, присутствуют ли все указанные значения в дереве.
3. Проверьте наличие списков [10, 15, 40] и [5, 25, 30].

Вариант 9

1. Создайте дерево и вставьте в него числа от 1 до 7 в порядке: 4, 2, 6, 1, 3, 5, 7.
2. Реализуйте метод sum_values(), возвращающий сумму всех значений в дереве.
3. Выведите сумму и среднее арифметическое значений.

Вариант 10

1. Заполните дерево произвольными числами (не менее 9).
2. Реализуйте метод leaf_count(), подсчитывающий количество листьев (узлов без потомков).
3. Выведите общее число узлов и число листьев.

Контрольные вопросы.

1. Что такое бинарное дерево и чем оно отличается от других видов деревьев?
2. Какие свойства характерны для бинарного дерева поиска?
3. Почему при вставке элемента в бинарное дерево поиска важно соблюдать правило: левый потомок меньше родителя, правый — больше или равен?
4. Какой обход бинарного дерева поиска даёт отсортированную последовательность значений и почему?
5. Что такое лист дерева и как его определить в программе?
6. Как найти минимальное и максимальное значение в бинарном дереве поиска?
7. В чём разница между прямым, симметричным и обратным обходами бинарного дерева?
8. Как определить высоту бинарного дерева и зачем она нужна?
9. Почему в худшем случае (вырожденное дерево) операции поиска и вставки становятся неэффективными?
10. Можно ли в бинарном дереве поиска хранить дубликаты значений? Если да, то как это обычно реализуется?

Порядок оформления отчета.

В отчете кратко опишите, что вы сделали, какие умения приобрели. Укажите результаты выполнения заданий. Ответьте на контрольные вопросы. Сделайте развернутый вывод по работе.

Порядок сдачи работы преподавателю.

Для проверки, преподавателю необходимо представить все полученные в результате выполнения работы материалы, схемы, программы, результаты работы программ, а так же полностью оформленный отчет. Коды программ нужно представлять в электронном виде, готовом для выполнения.

4. ОБЩАЯ ХАРАКТЕРИСТИКА САМОСТОЯТЕЛЬНОЙ РАБОТЫ ОБУЧАЮЩИХСЯ

Самостоятельная работа - целенаправленная, планируемая в рамках учебного плана деятельность студентов, которая осуществляется по заданию, при методическом руководстве и контроле преподавателя, но без его непосредственного участия. Самостоятельная работа студентов является одной из важнейших составляющих образовательного процесса.

В учебном процессе учебного заведения выделяют два вида самостоятельной работы: аудиторная и внеаудиторная.

Аудиторная самостоятельная работа выполняется на учебных занятиях под непосредственным руководством преподавателя и по его заданию.

Внеаудиторная — планируемая учебная, учебно-исследовательская, научно-исследовательская работа студентов, выполняемая во внеаудиторное время по заданию и при методическом руководстве преподавателя, но без его непосредственного участия.

Целью самостоятельной работы студентов является:

- систематизация и закрепление полученных теоретических знаний и практических умений;
- углубление и расширение теоретических знаний;
- формирование умений использовать нормативную, правовую, справочную документацию и специальную литературу;
- развитие познавательных способностей и активности студентов, творческой инициативы, самостоятельности, ответственности, организованности;
- формирование самостоятельности мышления, способностей к саморазвитию, совершенствованию и самоорганизации;
- формирование общих и профессиональных компетенций.

Самостоятельная работа студентов должна быть хорошо спланирована и организована. При планировании такой работы необходимо учитывать условия, обеспечивающие её успешное выполнение:

- чёткое определение преподавателем объёма и содержания самостоятельной работы;
- определение видов консультативной помощи;
- постановка цели самостоятельной работы и критерии её оценки;
- виды и формы контроля её выполнения.

Выполняя самостоятельную работу под контролем преподавателя, студент должен:

- освоить минимум знаний;
- планировать свою самостоятельную работу в соответствии разработанным графиком;

- выполнять самостоятельную работу и отчитываться по ее результатам в соответствии с графиком представления результатов, видами и сроками отчетности по самостоятельной работе студентов.

В процессе самостоятельной работы студент приобретает навыки самоорганизации, самоконтроля, самоуправления, саморефлексии и становится активным самостоятельным субъектом учебной деятельности.

Таким образом, самостоятельная работа студентов оказывает важное влияние на формирование личности будущего специалиста.

Самостоятельная работа студентов является обязательной для каждого студента, объем ее определяется учебным планом в соответствии с требованиями Государственных образовательных стандартов.

При изучении тем дисциплины студенты выполняют следующие виды самостоятельной работы:

- проработка конспектов занятий, учебных изданий и специальной технической литературы;
- составление конспекта, тематических схем, таблиц;
- подготовка к лабораторным работам и практическим занятиям с использованием методических рекомендаций преподавателя;
- оформление отчетов по лабораторным работам и практическим занятиям, подготовка к их защите;
- моделирование и решение производственных процессов и ситуационных задач;
- подготовка презентаций;
- работа с электронными ресурсами в сети Интернет;
- подготовка к семинару;
- подготовка к зачетам, экзаменам.

Технология организации самостоятельной работы студентов включает использование информационных и материально-технических ресурсов образовательного учреждения. Материально-техническое и информационно - техническое обеспечение самостоятельной работы студентов включает в себя:

- библиотеку с читальным залом, укомплектованную в соответствии с существующими нормами;
- учебно-методическую базу учебных кабинетов, лабораторий и методического центра;
- компьютерные классы с возможностью работы в Интернет;
- базы практики в соответствии с заключенными договорами;
- аудитории для консультационной деятельности;
- учебную и учебно-методическую литературу, разработанную с учетом увеличения доли самостоятельной работы студентов, и иные методические материалы.

Перед выполнением внеаудиторной самостоятельной работы преподаватель проводит инструктаж по выполнению задания, в котором указывает цель задания, его содержание, сроки выполнения, ориентировочный объем работы, основные требования к результатам работы, критерии оценки. Во время выполнения студентами внеаудиторной самостоятельной работы и

при необходимости преподаватель может проводить консультации. Самостоятельная работа может осуществляться индивидуально или группами студентов в зависимости от цели, объема, конкретной тематики самостоятельной работы, уровня сложности, уровня умений обучающихся.

5. МЕТОДИЧЕСКИЕ УКАЗАНИЯ К ВЫПОЛНЕНИЮ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Общие методические рекомендации студенту при изучении тем дисциплины.

Большая часть самостоятельной работы выполняется студентом вне учебных занятий при подготовке домашних заданий. Общие требования к выполнению этого вида самостоятельной работы заключаются в следующем:

- активно работать на уроке, усваиваю основную часть нового материала;
- если что-то непонятно, не стесняться задавать вопросы преподавателю;
- большое задание необходимо разбивать на части и работать над каждой из них в отдельности;
- выполняя домашнее задание, надо не просто думать, что надо сделать, а еще и решать, с помощью каких средств и приемов этого можно добиться;
- в процессе приготовления домашнего задания необходимо делать перерывы;
- готовиться к докладам, рефератам, защите курсовых работ и проектов, практических и лабораторных занятий надо заранее, равномерно распределяя нагрузку, а не оставлять такую ответственную работу на последний день;
- изучая заданный материал, сначала надо его понять, а уже потом запомнить;
- научиться находить интересующую нужную информацию с помощью компьютера;
- не стесняться обращаться за помощью к взрослым и однокурсникам;
- надо составлять план устного ответа и проверять себя;
- на письменном столе должно лежать только то, что необходимо для выполнения одного задания. После его завершения со стола убираются уже использованные материалы, и кладутся те учебные принадлежности, которые необходимы для выполнения следующего задания;
- нужно решить, в какой последовательности лучше выполнять задания и сколько времени понадобится на каждое из них;
- трудный материал урока лучше повторить в тот же день, чтобы сразу закрепить его и запомнить;
- читая учебник, надо задавать самому себе вопросы по тексту.

Подготовка тематических сообщений, докладов, рефератов

Реферат доклад, сообщение (от латинского refero - передаю, сообщаю) - краткое письменное изложение материала по определенной теме с целью привития студентам навыков самостоятельного поиска и анализа информации, формирования умения подбора и изучения литературных источников,

используя при этом дополнительную научную, методическую и периодическую литературу.

Тема реферата выбирается по желанию студента из списка, предлагаемого преподавателем. Тема может быть сформулирована студентом самостоятельно.

Выбранная тема согласовывается с преподавателем.

После выбора темы требуется:

- составить план реферата;
- подобрать необходимую информацию;
- изучить подобранную информацию;
- составить текст реферата.

План реферата должен включать в себя введение, основной текст и заключение. Во введении аргументируется актуальность выбранной темы, указываются цели и задачи исследования. В нем также отражается методика исследования и структура работы. Основная часть работы предполагает освещение материала в соответствии с планом. В заключении излагаются основные выводы и рекомендации по теме исследования.

Реферат оформляется согласно требованиям, установленным в учебном заведении. Он должен содержать: титульный лист, оглавление и список использованной литературы. На титульном листе указываются: название учебного заведения, название профессионального модуля, междисциплинарного курса, тема работы, курс, группа, фамилии, имена, отчества студента и руководителя работы, название города, в котором находится учебное заведение, год написания данной работы. Реферат может содержать приложения в форме схем, образцов документов и другие изображения в соответствии с темой исследования. Все страницы работы, включая оглавление и список литературы, нумеруются по порядку с титульного листа (на нем цифра не ставится) до последней страницы без пропусков и повторений. Введение, заключение, новые главы, список использованных источников и литературы должны начинаться с нового листа. Подбор литературы производится студентом из предложенного преподавателем списка литературы. Текст реферата необходимо набирать на компьютере на одной стороне листа. Размер левого поля 30 мм, правого - 15 мм, верхнего - 20 мм, нижнего - 20 мм. Шрифт - Times New Roman, размер - 14, межстрочный интервал - 1,5. Фразы, начинающиеся с новой строки, печатаются с абзацным отступом от начала строки (1,25 см). Реферат, выполненный небрежно, неразборчиво, без соблюдения требований по оформлению, возвращается студенту без проверки с указанием причин возврата на титульном листе.

Критерии оценки:

- знание и понимание проблемы;
- умение систематизировать и анализировать материал, четко и обоснованно формулировать выводы;
- «трудозатратность» (объем изученной литературы, добросовестное отношение к анализу проблемы);

- самостоятельность, способность к определению собственной позиции по проблеме и к практической адаптации материала, недопустимость плагиата;
- выполнение необходимых формальностей (точность в цитировании и указании источника текстового фрагмента, аккуратность оформления).

Проработка занятый, учебных изданий и специальной технической литературы

Работа с конспектом лекций по темам междисциплинарных курсов заключается в том, что студент после рассмотрения темы на учебных занятиях в период между очередными лекциями изучает материал конспекта. При этом непонятные положения конспекта необходимо выяснять у преподавателя на консультациях или при чтении основной и дополнительной литературы.

При работе с книгой необходимо научиться правильно ее читать, вести записи. Для подбора литературы в библиотеке используются алфавитный и систематический каталоги. Правильный подбор учебников рекомендуется преподавателем. Необходимая литература может быть также указана в методических разработках. Изучая материал по учебнику, следует переходить к следующему вопросу только после правильного уяснения предыдущего, описывая на бумаге все выкладки и определения (в том числе те, которые в учебнике опущены или на лекции даны для самостоятельного вывода). Полезно составлять опорные конспекты. При изучении материала по учебнику, полезно в тетради (на специально отведенных полях) дополнять конспект лекций, написанный на учебных занятиях. Там же следует отмечать вопросы, выделенные студентом для консультации с преподавателем. Выводы, полученные в результате изучения, рекомендуется в конспекте выделять, чтобы они при пропитывании записей лучше запоминались. Различают два вида чтения; первичное и вторичное. Первичное - эти внимательное, неторопливое чтение, при котором можно остановиться на трудных местах. После него не должно остаться ни одного непонятного слова. Содержание не всегда может быть понятно после первичного чтения. Задача вторичного чтения - полное усвоение смысла целого (по счету это чтение может быть и не вторым, а третьим или четвертым).

Чтение научного текста является частью познавательной деятельности. Ее цель - извлечение из текста необходимой информации. От того на сколько осознанна читающим собственная внутренняя установка при обращении к печатному слову (найти нужные сведения, усвоить информацию полностью или частично, критически проанализировать материал и т.п.) во многом зависит эффективность осуществляемого действия. Выделяют четыре основные установки в чтении научного текста:

- информационно-поисковая, задача которой - найти, выделить искомую информацию;
- усваивающая, при которой усилия читателя направлены на то, чтобы как можно полнее осознать и запомнить как сами сведения, излагаемые автором, так и всю логику его рассуждений;

- аналитико-критическая - читатель стремится критически осмыслить материал, проанализировав его, определив свое отношение к нему;
- творческая, создающая у читателя готовность в том или ином виде использовать суждения автора, ход его мыслей, результат наблюдения, разработанную методику, дополнить их, подвергнуть новой проверке.

Самостоятельная работа при чтении учебной литературы начинается с изучения конспекта материала, полученного при слушании лекций преподавателя. Полученную информацию необходимо осмыслить. При необходимости, в конспект лекций могут быть внесены схемы, эскизы рисунков, другая дополнительная информация.

Составление конспекта, тематических схем, таблиц

При изучении нового материала, как правило, составляется конспект. Конспект - изложение текста, которому присущи краткость, связность и последовательность. При этом максимально точно записываются формулы, определения, схемы, трудные для запоминания места.

При оформлении конспекта необходимо стремиться к емкости каждого предложения. Мысли автора книги следует излагать кратко, заботясь о стиле и выразительности написанного. Число дополнительных элементов конспекта должно быть логически обоснованным, записи должны распределяться в определенной последовательности, отвечающей логической структуре текста. Для уточнения и дополнения необходимо оставлять поля. Овладение навыками конспектирования требует от студента целеустремленности, повседневной самостоятельной работы.

Классификация конспектов:

- плановый конспект, для чего сначала нужно написать план текста, а затем на пункты плана делаются комментарии: свободно изложенный текст либо цитаты;
- обзорный конспект - краткое изложение данной темы с использованием нескольких источников;
- текстуальный конспект состоит из цитат одного текста;
- свободный конспект предполагает цитаты текста и собственные формулировки прочитанного текста;
- сложный - конспект, в котором отражается определенная тема или вопрос;
- хронологический конспект отражает последовательность событий;
- опорный конспект, в котором излагается информация в виде опорных знаков, слов, сигналов.

Методические рекомендации по составлению конспекта:

- определить цель написания конспекта;
- внимательно прочитать текст, уточнить в справочной литературе непонятные слова;
- выделить основные смысловые части текста;
- определить главное, составить план;

- кратко сформулировать основные положения текста, отметить аргументацию автора;
- составить текст конспекта, изложив информацию кратко и своими словами, четко следуя пунктам плана, записи следует вести четко, ясно;
- грамотно записывать цитаты, учитывая лаконичность, значимость мысли;
- в тексте конспекта желательно приводить не только тезисные положения, но и их доказательства.

При составлении тематических схем, таблиц необходимо внимательно прочитать текст соответствующий параграф учебника. Продумать «конструкцию» таблицы или схемы, расположение порядковых номеров, терминов, примеров и пояснений (и прочего). Начертить схему или таблицу и заполнить ее графы необходимым содержимым.

Подготовка к лабораторным работам и практическим занятиям, оформление отчетов по лабораторным работам и практическим занятиям, подготовка к их защите

Программы профессиональных модулей предусматривают выполнение практических и лабораторных занятий.

Лабораторное занятие - форма учебного занятия, ведущей дидактической целью которого является экспериментальное подтверждение и проверка существующих теоретических положений (законов, зависимостей), формирование учебных и профессиональных практических умений и навыков.

Практическое занятие - это одна из форм учебной работы, которая ориентирована на закрепление изученного теоретического материала, его более глубокое усвоение и формирование умения применять теоретические знания в практических целях. Особое внимание на практических занятиях уделяется выработке учебных или профессиональных навыков. Такие навыки формируются в процессе выполнения конкретных заданий - упражнений, задач - под руководством и контролем преподавателя.

Подготовка к практическим и лабораторным занятиям заключается в работе с конспектом лекций по данной теме, в изучении соответствующего раздела учебника или учебного пособия, в просмотре дополнительной литературы. Этапы подготовки к практическому или лабораторному занятию заключаются в следующем: освежить в памяти теоретические сведения, полученные на лекциях и в процессе самостоятельной работы, подобрать необходимую учебную и справочную литературу. Отобрать те материалы, которые позволят в полной мере реализовать цели и задачи предстоящей работы. Еще раз проверить соответствие отобранного материала. Студент должен прийти на лабораторное или практическое занятие подготовленным по данной теме.

При выполнении заданий практического или лабораторного занятия студент должен быть ознакомлен преподавателем с целью и ходом выполнения задания и, по необходимости, с правилами техники безопасности. Если у студентов во время выполнения заданий возникают вопросы, то преподаватель

консультирует студентов. Порядок выполнения того или иного задания излагается в инструкционных картах или рабочих тетрадях.

После проведения занятия студент представляет письменный отчет, который оформляется в соответствие с принятыми в образовательном учреждении правилами. Отчеты оформляются на листах писчей бумаги формата А4 или в специальных рабочих тетрадях, разработанных преподавателем. Содержание отчета указано в инструкционных картах или рабочих тетрадях.

При подготовке к защите практических и лабораторных занятий студент должен ответить на контрольные вопросы, указанные также в инструкционных картах или рабочих тетрадях, проштудировав при этом конспект лекций, учебную литературу.

Моделирование и решение производственных процессов и ситуационных задач

При изучении дисциплины очень часто студенту приходится сталкиваться с профессиональными задачами и ситуациями, которые необходимо решить самостоятельно, как во время аудиторной работы, так и во время внеаудиторной. При решении таких задач необходимо:

- провести анализ ситуации для определения проблемы в целом; представить ситуацию и себя в качестве действующего в ней лица; проанализировать ошибочные или правильные действия всех участников ситуации;
- определить проблемные узлы - возможные причины и прогнозируемые последствия развития данной ситуации;
- рассмотреть условное прогнозирование развития ситуации: определить окончательную гипотезу, представить обоснованный и доказательный прогноз вероятностного развития ситуации; предложить варианты действий, обоснованные теоретически и, по возможности, подкрепленные практическим личным опытом, опираясь на принципы профессиональной этики; определить способы и методы воздействия на предлагаемую ситуацию;
- сформулировать итоговые выводы, используя профессиональные термины, доказательства правильности своего решения.

Подготовка презентаций

Подготовка презентации позволит студенту логически выстроить изучаемый материал, систематизировать его, сформировать коммуникативные компетенции. Материал презентации представляется в виде текста, схем, диаграмм, таблиц, которые призваны дополнить текстовую информацию или передать ее в более наглядном виде. Желательно избегать в презентации изображений, не несущих смысловой нагрузки, если они не являются частью стилевого оформления. Цвет графических изображений не должен резко контрастировать с общим стилевым оформлением слайдов, иллюстрации рекомендуется сопровождать пояснительным текстом.

Анимационные эффекты используются для привлечения внимания слушателей или для демонстрации динамики развития какого - либо процесса. В этих случаях использование анимации оправдано, но не стоит чрезмерно насыщать презентацию такими эффектами, иначе это вызовет негативную реакцию аудитории.

Звуковое сопровождение должно отражать суть или подчеркивать особенность темы слайда, презентации. Фоновая музыка не должна отвлекать внимание слушателей и заглушать слова докладчика.

Оптимальное количество слайдов, как правило, десять - пятнадцать. Для оформления слайдов презентации рекомендуется использовать несложные шаблоны, соблюдать единый стиль. Не рекомендуется на одном слайде использовать более трех цветов. Смену слайдов для управления презентацией докладчиком желательно устанавливать по щелчку без времени. Шрифт, выбираемый для презентации, должен обеспечивать читаемость информации на экране и соответствовать выбранному шаблону оформления. Не желательно использовать разные шрифты в одной презентации.

Алгоритм выстраивания презентации должен соответствовать логической структуре работы и отражать последовательность ее этапов. Независимо от алгоритма выстраивания презентации на первом слайде рекомендуется выносить следующие данные: полное наименование образовательной организации; тема презентации; фамилия, имя, отчество студента; специальность обучения; фамилия, имя, отчество руководителя. Последний слайд должен содержать фразу «Спасибо за внимание».

Работа с электронными ресурсами в сети Интернет

Для повышения эффективности самостоятельной работы студент должен учиться работать в поисковой системе сети Интернет, в электронно-библиотечной системе и использовать найденную информацию при подготовке к занятиям.

Интернет сегодня - правомерный источник научных статей, статистической и аналитической информации, и использование его наряду с книгами давно уже стало нормой. Однако, несмотря на то, что ресурсы Интернета позволяют достаточно быстро и эффективно осуществлять поиск необходимой информации, следует помнить о том, что эта информация может быть неточной или вовсе не соответствовать действительности. В связи с этим при поиске материала по заданной тематике следует обращать внимание на научные труды признанных авторов, которые посоветовали вам преподаватели.

Поиск информации можно вести по автору, заглавию, виду издания, году издания или издательству. Также в сети Интернет доступна услуга по скачиванию методических указаний и учебных пособий, подбору необходимой учебной и научно - технической литературы.

Подготовка к семинару

Семинар — это особая форма учебно-теоретических занятий, которая, как правило, служит дополнением к лекционному курсу. Семинар обычно посвящен детальному изучению отдельной темы.

Этапы подготовки к семинару:

- проанализировать тему семинара, подумать о цели и основных проблемах, вынесенных на обсуждение;
- внимательно прочитать материал, данный преподавателем по этой теме на лекции;
- изучить рекомендованную литературу, делая при этом конспекты прочитанного или выписки, которые понадобятся при обсуждении на семинаре;
- постараться сформулировать свое мнение по каждому вопросу и аргументированно его обосновать;
- записать возникшие во время самостоятельной работы с учебниками и научной литературой вопросы, чтобы затем на семинаре получить на них ответы.

При подготовке к семинарским занятиям следует руководствоваться указаниями и рекомендациями преподавателя, использовать основную и дополнительную литературу из представленного им списка.

При подготовке доклада на семинарское занятие желательно заранее обсудить с преподавателем перечень используемой литературы, за день до семинарского занятия предупредить его о необходимых для представления материала технических средствах. Напечатанный текст доклада представить преподавателю на рецензию.

Подготовка к зачетам, экзаменам

Изучение выше перечисленных тем дисциплины завершается зачетами или экзаменами.

Подготовка к зачету или экзамену способствует закреплению, углублению и обобщению знаний, получаемых в процессе обучения, а также применению их к решению практических задач. Готовясь к зачету или экзамену, студент ликвидирует имеющиеся пробелы в знаниях, углубляет, систематизирует и упорядочивает свои знания. На зачете или экзамене студент демонстрирует то, что он приобрел в процессе обучения конкретным темам междисциплинарных курсов или модулям в целом.

Экзаменационная сессия - это серия экзаменов, установленных учебным планом. Между экзаменами, согласно графику их проведения, дается интервал времени в несколько дней. Не следует думать, что их достаточно для успешной подготовки к экзаменам. В эти дни нужно систематизировать уже имеющиеся знания. На консультации перед экзаменом студентов познакомят с основными требованиями, ответят на возникшие у них вопросы. Поэтому посещение консультаций обязательно.

Требования к организации подготовки студента к экзаменам те же, что и при занятиях в течение семестра, но соблюдаться они должны более строго. Во-первых, очень важно соблюдение режима дня: сон не менее 8 часов в сутки, занятия должны заканчиваться не позднее, чем за 2-3 часа до сна.

Оптимальное время занятий - утренние и дневные часы. В перерывах между занятиями рекомендуются прогулки на свежем воздухе, неутомительные занятия спортом. Во-вторых, наличие хороших собственных конспектов лекций. Даже в том случае, если была пропущена какая- либо лекция, необходимо во время ее восстановить, обдумать, снять возникшие вопросы для того, чтобы запоминание материала было осознанным. В-третьих, при подготовке к зачету или экзамену у студента должен быть хороший учебник или конспект литературы, прочитанной по указанию преподавателя в течение семестра. Здесь можно эффективно использовать листы опорных конспектов. Вначале следует просмотреть весь материал по сдаваемой теме, отметить для себя трудные вопросы, обязательно в них разобраться. В заключение еще раз целесообразно повторить основные положения. Систематическая подготовка к занятиям в течение семестра позволит использовать время экзаменационной сессии для систематизации знаний.

Правила подготовки к экзамену:

- сориентироваться во всем материале и обязательно расположить его согласно экзаменационным вопросам или вопросам, обсуждаемым на семинарах, учебных занятиях. Эта работа может занять много времени, но все остальное - уже технические детали, главное - это ориентировка в материале;
- постараться максимально запомнить материал, переосмыслить его, рассмотреть альтернативные идеи;
- подготовить «шпаргалки», главный смысл которых систематизация и оптимизация знаний, однако пользоваться таким подспорьем не рекомендуется. Это очень сложная и важная для студента работа, более сложная и важная, чем простое поглощение массы учебной информации. Если студент самостоятельно подготовил такие «шпаргалки», то, скорее всего, он и экзамены сдавать будет более уверенно, так как у него уже сформирована общая ориентировка в сложном материале. Как это ни парадоксально, но использование «шпаргалок» часто позволяет отвечающему студенту лучше демонстрировать свои познания, точнее - ориентировку в знаниях, что намного важнее знания «запомненного» и «тут же забытого» после сдачи экзамена.

При ответе на экзамене студент сначала должен продемонстрировать преподавателю усвоенный по программе обучения материал, и лишь после этого высказать иную, желательно аргументированную точку зрения.

6. МЕТОДИКА ВЫПОЛНЕНИЯ ВНЕАУДИТОРНОЙ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

1. Получить у преподавателя задание и необходимую литературу.
2. Найти предложенную литературу на образовательном портале или в библиотеке.
3. Изучить имеющуюся литературу в электронном или печатном виде, прочитать материалы лекций, практических и (или) семинарских занятий по теме.
4. Изучить методические рекомендации.
5. Оформить работу в тетради или на компьютере в соответствии с требованиями преподавателя.
6. Сдать самостоятельную работу преподавателю, предварительно ответив на вопросы для самоконтроля.

7. МЕТОДЫ КОНТРОЛЯ И ОЦЕНКА ВНЕАУДИТОРНОЙ САМОСТОЯТЕЛЬНОЙ РАБОТЫ

Контроль результатов самостоятельной работы проводится преподавателем одновременно с текущим и промежуточным контролем знаний обучающихся. Для контроля самостоятельной работы обучающегося используются разнообразные формы и методы: фронтальный, индивидуальный, выборочный, самоконтроль, защита презентации, участие в семинарском занятии, ответы на контрольные вопросы и т. д. При контроле результатов самостоятельной работы используются следующие критерии:

- уровень освоения обучающимся учебного материала;
- умение обучающегося использовать теоретические знания при выполнении заданий;
- обоснованность и чёткость изложения ответа;
- оформления материала в соответствии с требованиями.

Критерии оценки выполненной обучающимися работы:

оценка «5» - работа выполнена без ошибок; чисто, без исправлений; тема раскрыта полностью;

оценка «4» - работа выполнена с незначительными ошибками; тема раскрыта не полностью;

оценка «3» - работа выполнена со значительными ошибками; тема практически не раскрыта;

оценка «2» - работа не выполнена.

8. СПИСОК РЕКОМЕНДУЕМЫХ ИСТОЧНИКОВ

1. Кудрявцева, И. А. Программирование: комбинаторная логика : учебник для среднего профессионального образования / И. А. Кудрявцева, М. В. Швецкий. — 2-е изд., перераб. и доп. — Москва: издательство Юрайт, 2025. — 524 с. — (профессиональное образование). — isbn 978-5-534-15128-2. — текст : электронный // образовательная платформа юрайт [сайт]. — url: https://urait.ru/bcode/565805 (дата обращения: 04.07.2025)..
2. Трофимов, В. В. Основы алгоритмизации и программирования: учебник для среднего профессионального образования / В. В. Трофимов, Т. А. Павловская. — 4-е изд. — Москва: Издательство Юрайт, 2025. — 108 с. — (Профессиональное образование). — ISBN 978-5-534-20429-2. — Текст: электронный // Образовательная платформа Юрайт [сайт]. — URL: https://urait.ru/bcode/563861 (дата обращения: 04.07.2025).
3. Чернышев, С. А. Основы программирования на Python : учебник для среднего профессионального образования / С. А. Чернышев. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2025. — 349 с. — (Профессиональное образование). — ISBN 978-5-534-17056-6. — Текст : электронный // Образовательная платформа Юрай