Оглавление

.Раздел 1. Основные положения информатики 2

Раздел 1. Основные положения информатики 3

Лекция 1. Информатика и информационные ресурсы 3

Раздел 2. Основы языка Python для работы с данными 27

Лекция 2. Основы работы с языком программирования. 27

Лекция 3. Основные структуры данных. Списки. Работа со списками. 53

Раздел 3. Базовые алгоритмы обработки данных 56

Лекция 4. Системы линейных алгебраических уравнений. Численное интегрирование Решение нелинейных уравнений. 56

4.1 Основная запись СЛАУ. Понятие решения СЛАУ и некоторые определения. 56

4.2. Общая схема метода Гаусса 58

4.3. Обратная матрица A^{-1}. 61

4.4. Понятие об итерационных методах решения СЛАУ 62

4.4.1. Общая схема итерационного процесса 64

4.4.2. Метод Зейделя и метод простой итерации 64

4.5. Методы численного интегрирования 66

4.5.1. Понятие о формулах численного интегрирования 66

4.5.2 Формула прямоугольников (формула средних) 67

4.5.3 Формула трапеций 69

4.5.4. Формула Симпсона 71

4.5.5. Точность квадратурных формул 73

4.6. Методы решения нелинейных уравнений 75

4.6.1. Уравнения с одним неизвестным 75

4.6.2. Метод половинного деления..... 77

4.6.3. Метод Ньютона 79

.

Раздел 1. Основные положения информатики

Лекция 1. Информатика и информационные ресурсы

Информационные ресурсы - продукт интеллектуальной деятельности наиболее квалифицированной и творчески активной части трудоспособного населения.

Сегодня информационное невежество - прямой путь к банкротству. Вместе с тем информационный поток уже опережает возможности человека по его обработке и использованию, так как мозг человека не беспределен. Необходимо изыскание и применение принципиально новых методов и средств восприятия, передачи, обработки, хранения и распространения информации, способных оперировать с большими массивами информации в реальном времени.

В настоящее время это реализовано - на базе компьютеров создана информационная индустрия, определившая переход к безбумажным технологиям обмена информацией на основе электронной почты, телеконференций, локальных и глобальных сетей передачи данных, спутниковой связи, баз и банков данных, информационно-поисковых систем.

Феномен информации инициировал новую науку об информации - информатику. Ее задачами являются: изучение свойств информации, методов управления информационными потоками и их использования, методов обработки информации для оптимального хранения, поиска и распространения. Можно сказать, что в настоящее время мировое сообщество все более превращается в гигантскую информационную систему.

В соответствии с документами ЮНЕСКО для информатики введено следующее определение:

ИНФОРМАТИКА - это крупное научное направление, которое включает методы и средства сбора, анализа и обработки информации на основе достижений микропроцессорной, компьютерной техники и

технологий, средств и систем коммуникаций в целях научно-технического прогресса и социального развития мирового сообщества.

Методы и средства информатики, материализуясь, доходят до потребителя в виде новых информационных технологий, т. е. современных видов информационного обслуживания на базе компьютерных средств и коммуникаций. Компьютер открыл эру безбумажной технологии. По мнению академика Н.Н. Моисеева, создание компьютера столь же крупная веха в становлении человечества, как и использование огня.

Алгоритм, машина Тьюринга, последовательная архитектура ЭВМ

Эра информатики берет начало в 30-х гг. XX в., когда стало возможным создание вычислительных машин на основе электронных устройств. До этого существовали лишь механические устройства, впервые построенные в XVII в.:

- суммирующие часы Вильгельма Шикарда, 1623 г. (кстати, в этом же году родился Блез Паскаль);
- счётное устройство Блеза Паскаля, 1642 г.;
- ступенчатый вычислитель Готфрида Лейбница, 1673 г., впервые предложившего двоичную систему счисления.

В XIX в. Чарльз Бэббидж изобретает “Аналитическую машину” (1834 г.), в основе которой заложен принцип разделения информации на команды и данные.

В 1843 г. 28-летняя графиня Августа Ада Лавлейс пишет научную работу об аналитической машине Бэббиджа, заложившую научные основы программирования на вычислительных машинах. В её работе была приведена программа, предназначенная для решения уравнения Бернулли.

В России в 70-80-х гг. XIX в. появляются механические арифмометры шведско-русского изобретателя Однера В.Т. и русского математика Чебышёва П.Л. В конце XIX в. Холлерит строит счётную машину на перфокартах, которая затем участвует в переписи населения США.

В 1936 г. Аланом Тьюрингом была предложена абстрактная вычислительная машина для формализации понятия «алгоритм», которое, пожалуй, является важнейшим понятием императивного программирования.

Гипотетическая машина Тьюринга положила начало теории алгоритмов, теории вычислений и программированию.

Машина Тьюринга имеет управляющее устройство, способное находиться в одном из множества состояний и бесконечную в обе стороны ленту, разделенную на ячейки. В ячейках ленты могут быть записаны символы некоторого конечного алфавита. Управляющее устройство может перемещаться влево и вправо по ленте, считывать и записывать символы в ячейки. Перемещение управляющего устройства основано на *правилах перехода*, которые представляют алгоритм, реализуемый машиной Тьюринга. Правила перехода применяются последовательно, путём циклического просмотра набора этих правил и применения подходящего на данном шаге правила до тех пор, пока в текущей ячейке ленты не будет встречен специальный символ останова.

Именно идея последовательного исполнения правил (команд, инструкций) и легла в основу архитектуры первых ЭВМ 1940-х гг.

Здесь следует отметить, что первые ЭВМ не были программируемыми, а набор команд и программ ЭВМ определялся их составом и способом коммутации составляющих их блоков. Первую ЭВМ разработали в начале 1940-х гг. в Университете Пенсильвании Эккерт и Мочли, она называлась ENIAC (Electronic Numerical Integrator And Computer) – электронный численный интегратор и вычислитель. Перепрограммирование такой машины заключалось в перестройке её блоков и устройств и их перекоммутации.

В 1945-м был опубликован отчет по архитектуре новой ЭВМ EDVAC, разработанной тем же коллективом, к которому присоединился Джон фон Нейман и др. В новой архитектуре процессорное устройство отделялось от памяти, а основным принципом являлось хранение и данных и программ в

одном виде. Один и тот же подход к рассмотрению данных и команд стал настоящим прорывом в области вычислений.

Строительство EDVAC, однако, затянулось, и первой ЭВМ, в которой эта архитектура реализована, стала ЭВМ «Марк I», разработанная в 1948 г. в Университете Манчестера (Великобритания). ЭВМ EDVAC разработана годом позже и введена в эксплуатацию ещё через 2 года. Лишь тогда термин «программирование» конкретизировался до термина «программирование ЭВМ». Первая в СССР ЭВМ с хранимой в памяти программой построена под руководством С.А. Лебедева и запущена в эксплуатацию в 1950-м.

До сих пор в основе подавляющего большинства современных вычислительных машин, гораздо более сложных, чем EDVAC или «Марк I», всё также лежит последовательная архитектура, предложенная коллективом из Университета Пенсильвании, не совсем справедливо называемая архитектурой фон Неймана. Интересно, что первая в СССР ЭВМ с производительностью 1 млн операций в секунду (БЭСМ-6) построена в 1965-м.

Следующим мощным толчком к развитию ЭВМ стало создание процессоров и наборов команд, но это был чисто технологический прорыв в уже проложенном архитектурном русле.

Интересно, что любая функция, которая может быть вычислена физическим устройством (ЭВМ), вычисляется и машиной Тьюринга (тезис Чёрча–Тьюринга). Таким образом, все подобные вычислительные машины оказываются эквивалентными гипотетической машине Тьюринга. Все задачи, разрешимые на машине Тьюринга, можно решить на современной ЭВМ, и наоборот.

Примеры программируемых устройств – компьютеры, мобильные телефоны, калькуляторы, бытовая техника, цифровая аппаратура, бортовые системы управления. Сегодня можно программировать не только аппаратные устройства, но и программы. Примеры программируемых программ – игровые персонажи, программные агенты.

Общее устройство ЭВМ

С позиции функционального назначения компьютер – это система, состоящая из 4-х основных устройств, выполняющих определенные функции: запоминающего устройства или памяти, которая разделяется на оперативную и постоянную, арифметико-логического устройства (АЛУ), устройства управления (УУ) и устройства ввода-вывода (УВВ) (рис.1.1). Рассмотрим их роль и назначение.

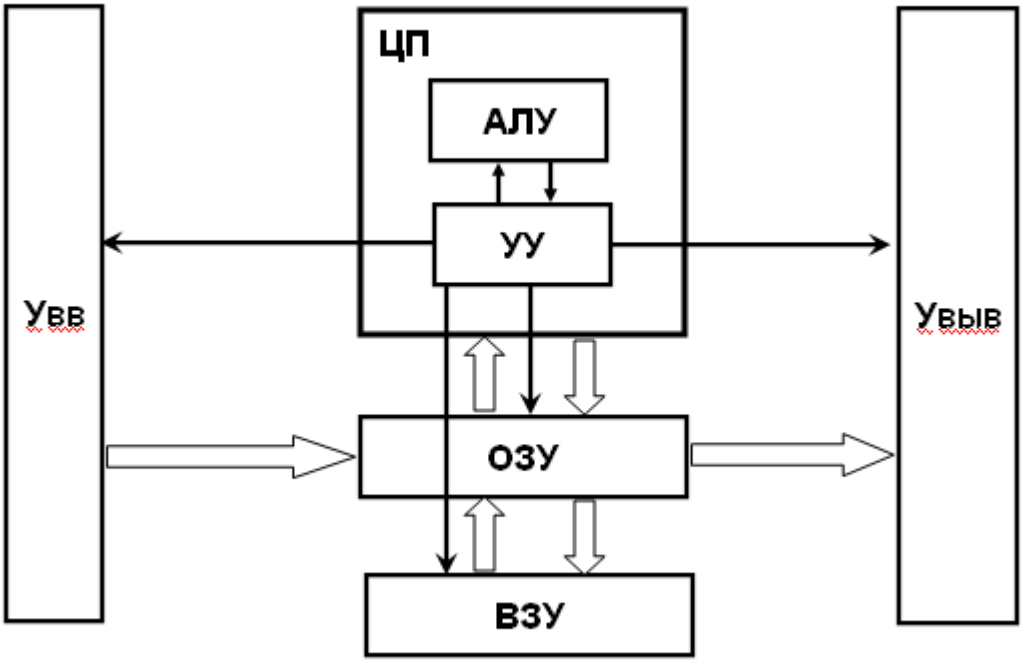

Рис 1.1. Общее устройство ЭВМ

Запоминающее устройство (память) предназначается для хранения информации и команд программы в ЭВМ. Информация, которая хранится в памяти, представляет собой закодированные с помощью 0 и 1 числа, символы, слова, команды, адреса и т.д.

Под записью числа в память понимают размещение этого числа в ячейке по указанному адресу и хранение его там до выборки по команде программы. Предыдущая информация, находившаяся в данной ячейке, перезаписывается. При программировании, например, на языке Паскаль или

Си, адрес ячейки связан с именем переменной, которое представляется комбинацией букв и цифр, выбираемых программистом.

Под считыванием числа из памяти понимают выборку числа из ячейки с указанным адресом. При этом копия числа передается из памяти в требуемое устройство, а само число остается в ячейке.

Пересылка информации означает, что информация читается из одной ячейки и записывается в другую.

Адрес ячейки формируется в устройстве управления (УУ), затем поступает в устройство выборки адреса, которое открывает информационный канал и подключает нужную ячейку.

Числа, символы, команды хранятся в памяти на равноправных началах и имеют один и тот же формат. Ни для памяти, ни для самого компьютера не имеет значения тип данных. Типы различаются только при обработке данных программой. Длину, или разрядность, ячейки определяет количество двоичных разрядов (битов). Каждый бит может содержать 1 или 0. В современных компьютерах длина ячейки кратна 8 битам и измеряется в байтах. Минимальная длина ячейки, для которой можно сформировать адрес, равна 1 байту, состоящему из 8 бит.

Для характеристики памяти используются следующие параметры:

1. Ёмкость памяти – максимальное количество хранимой информации в байтах;
2. Быстродействие памяти – время обращения к памяти, определяемое временем считывания или временем записи информации.

Арифметико-логическое устройство (АЛУ) производит арифметические и логические действия.

Следует отметить, что любую арифметическую операцию можно реализовать с использованием операции сложения.

Сложная логическая задача раскладывается на более простые задачи, где достаточно анализировать только два уровня: ДА и НЕТ.

Устройство управления (УУ) управляет всем ходом вычислительного и логического процесса в компьютере, т.е. выполняет функции "регулировщика движения" информации. УУ читает команду, расшифровывает ее и подключает необходимые цепи для ее выполнения. Считывание следующей команды происходит автоматически.

Фактически УУ выполняет следующий цикл действий:

1. Формирование адреса очередной команды;
2. Чтение команды из памяти и ее расшифровка;
3. Выполнение команды.

В современных компьютерах функции УУ и АЛУ выполняет одно устройство, называемое центральным процессором.

Устройство процессора и его назначение

На самом деле то, что мы сегодня называем процессором, правильно называть микропроцессором. Разница есть и определяется видом устройства и его историческим развитием.

Первый микропроцессор (Intel 4004) появился в 1971 году.

Внешне представляет собой кремневую пластинку с миллионами и миллиардами (на сегодняшний день) транзисторов и каналов для прохождения сигналов.

Назначение процессора – это автоматическое выполнение программы. Другими словами, он является основным компонентом любого компьютера.

Устройство процессора

Ключевыми компонентами процессора являются арифметико-логическое устройство (АЛУ), регистры и устройство управления. АЛУ выполнят основные математические и логические операции. Все вычисления производятся в двоичной системе счисления. От устройства управления зависит согласованность работы частей самого процессора и его связь с другими (внешними для него) устройствами. В регистрах временно хранятся текущая команда, исходные, промежуточные и конечные данные (результат вычислений АЛУ). Разрядность всех регистров одинакова.

Кэш данных и команд хранит часто используемые данные и команды. Обращение в кэш происходит намного быстрее, чем в оперативную память, поэтому, чем он больше, тем лучше.

Информация и информатика

Развитие производства компьютеров при снижении цен на них привело к информационной революции конца XX века. Эта революция заключается в проникновении компьютерных технологий в подавляющее большинство сфер деятельности людей (как производственных сфер, так и непроизводственных).

Появление таких тенденций как:

- увеличение темпов роста объема информации,
- возрастание удельного веса проблем, порожденных в процессе передачи информации,
- разнородность информации и ее рассеяние и в следствии этого трудность ее использования

привело к необходимости выделения как отдельной научной дисциплины предмета информатика.

Информатика – это наука об описании, представлении, интерпретации, формализации и применении знаний, накопленных с помощью вычислительной техники.

В США используется следующее определение предмета информатика – Computer Sciences – компьютерные науки.

Процессы, связанные с получением, хранением, обработкой и передачей информации, называются информационными процессами.

Информационный процесс можно описать как процесс взаимодействия объективных данных и субъективных методов. Все компоненты взаимодействия, такие как сигналы, данные и методы обработки имеют большое влияние на результат. Особенностью информационного процесса в информатике является автоматическое протекание некоторых этапов (аппаратные и программные методы обработки данных). Результатом

информационного процесса является сама информация. Так как информация не является статическим объектом, то важен процесс постоянного обновления данных и подбора соответствующих им методов.

Рис.1.2. Задачи информатики

Методы и технологии ориентированные на сбор, обработку, хранение, передачу распространение информации оможно коротко назвать информационными технологиями (IT). IT вошли в нашу жизнь повсеместно, ни один вид современного бизнеса не может быть успешным в долгосрочной перспективе без грамотного применения IT.

Схема цикла информационного процесса представлена на рис. 1.3. На ем показаны различные виды экономической информации и их взаимосвязи.

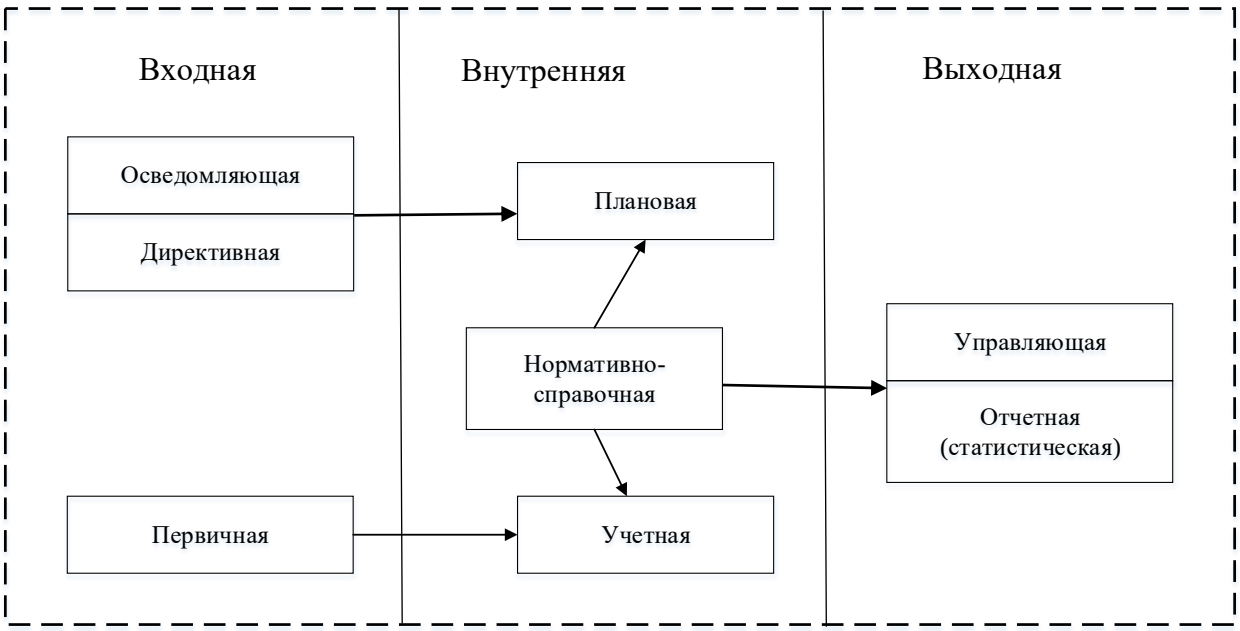

Рис.1.3. Общая структурная схема цикла информационного процесса

Содержательная классификация циркулирующей в разных объектах информации зависит от отраслевой принадлежности и уровня управления. Тем не менее, в процессе обработки информация проходит аналогичные стадии, общие в управлении разными экономическими объектами. Виды информации различаются:

- формой представления,
- организацией хранения,
- характером обработки.

Среди наиболее важных характеристик экономической информации, отражающих предъявляемые к ней требования, могут быть названы корректность, ценность, достоверность, точность, актуальность, полнота.

Схема управления бизнес процессом в рамках некоторого абстрактного экономического объекта следующая:

1. Информационные потоки, поступающие от внешних (управляющих, регулирующих и т.п.) органов.
2. Информация об условиях хозяйственной деятельности (наличных ресурсах, сроках поставок и др.).

3. Собственно управляющие воздействия – доведение принятых решений до объекта управления.

4. Информация о реализации управляющих воздействий.

5. Информация о результатах производства (например, выпущенная продукция, объем продаж и т.п.).

Входная информация поступает в орган управления извне.

Первичная информация – поступает непосредственно от объекта управления и получается в результате непосредственного измерения или подсчета. Первичная информация непосредственно соприкасается с конкретной стороной деятельности управляемых экономических объектов, при этом включает в себя как медленно изменяющиеся (условно постоянные), так и оперативные данные.

Директивная информация – исходит из вышестоящих органов, и в зависимости от характера подчиненности может включать различные параметры и условия формируемого задания. Директивные данные непосредственно влияют на цели функционирования объекта.

Осведомляющая информация – в основном поступает от вышестоящих органов, а также от других организаций связанных с объектом управления. Также осведомляющие данные определяют условия работы объекта.

К внутренней информации относится: учетная, плановая, а также нормативно-справочная информация.

Учетная информация – описывает уже совершившиеся процессы и реально существующие условия. Она является определенной и не зависит от последующих действий или принимаемых решений.

Плановая (прогнозная) информация – может корректироваться при изменении условий или целей.

Учетная и плановая информация является основой всего процесса управления, в том числе: анализ, прогнозирование, регулирование и другие функции.

Вместе с нормативно-справочной информацией, эти виды информации являются внутренними для органов управления и должны хранится в информационной базе. С ее помощью проводятся сложные виды обработки, которые позволяют обеспечивать решение управленческих задач.

Окончательным итогом обработки являются полученные выходные данные.

Выходные данные (информация) используются для управления или контроля ими со стороны вышестоящих или внешних органов и участвуют в последующих циклах обработки.

Деление выходной информации на управляющую и отчетную носит сугубо символический характер, т.к. данная информация может пересекаться, а также содержать одинаковые данные.

К выходным данным относятся также те данные, которые поступают в другие подразделения и являются для них осведомляющими.

Информация (от латинского “informatio” – изложение, разъяснение) – как научная категория составляет предмет изучения для различных областей знания. Понятие информация можно определить как «совокупность сигналов, воспринимаемых нашим сознанием, которые отражают те или иные свойства объектов и явлений окружающей нас действительности. Формулировка Шеннона: «Информация- это снятая неопределенность».

Информация – это совокупность сигналов, воспринимаемых нашим сознанием, которые отражают те или иные свойства объектов и явлений окружающей нас действительности. Природа данных сигналов подразумевает наличие принципиальных возможностей по их сохранению, передаче и трансформации (обработке).

Информация – это продукт взаимодействия данных и адекватных им методов. Содержательную часть информации определяют свойства данных. На свойства информации влияют свойства самих данных, а также свойства методов, взаимодействующих в ходе информационного процесса.

Более строгий подход к определению информации предполагает выделение двух фундаментальных элементов: источник (передатчик) и приемник (потребитель, клиент). При их взаимодействии и возникает информация – некоторое сообщение, которое, так или иначе, снимает неопределенность знания потребителя в отношении некоторого объекта, факта или явления. Информация – это знания полученные путем логической обработки данных.

Структура информации –то, что определяет взаимосвязи между ее составными элементами. Главным свойством информации является системность, т.е. совокупность взаимосвязанных элементов, в которой каждый отдельный элемент не обладает свойствами присущими всей системы.

Основными формами представления информации являются:

- символьно-знаковая (информация представленная совокупностью букв, цифр, знаков и т.п.);
- графическая (изображения и т.п.);
- звуковая.

Единицы меры измерения информации

За единицу количества информации принимается такое количество информации, которое содержит сообщение, уменьшающее неопределенность знаний в два раза. Такая единица названа бит.

Следующей по величине единицей измерения количества информации является байт (1 байт = 2^3 бит = 8 бит).

Кратные байту единицы измерения количества информации вводятся следующим образом:

1 Кбайт = 2^{10} байт = 1024 байт;

1 Мбайт = 2^{10} Кбайт = 1024 Кбайт;

1 Гбайт = 2^{10} Мбайт = 1024 Мбайт.

Экономическая информация. Примеры на уровне организации

Формулировка В.Д. Бройдо: *«Экономическая информация – это та информация, которая возникает при подготовке и в процессе производственно-хозяйственной деятельности и используется для управления этой деятельностью».*

Требования, предъявляемые к экономической информации

- Требования определяются самими характеристиками экономической информации.
- Корректность – форма и содержание должны предполагать однозначное восприятие
- Ценность – степень в которой информация способствует достижению поставленных целей и задач.
- Достоверность – как отражение объективной реальности
- Точность – мера близости к объективной реальности
- Актуальность – категория адекватности информации применительно к возможному изменению во времени изучаемого объекта
- Полнота – мера достаточности для принятия управленческого решения.
- На уровне фирмы к такой информации относятся:
- технические характеристики средств производства;
- описание технологий и условий производства;
- рыночная конъюнктура (цены, объемы спроса);
- сведения об оборотных средствах;
- сведения о кадровом составе;
- сведения о наличии и потребностях в ресурсах;
- нормативы и плановые задания;
- совокупность расчетных показателей (фондоемкость, рентабельность, себестоимость);
- различные приказы, инструкции, методики.

Программное обеспечение управления информационными ресурсами

Программные средства современных информационных технологий в целом подразделяются на системные и прикладные.

Рис. 1.4. Программные средства обеспечения управления информационными ресурсами

Системные программные средства предназначены для обеспечения деятельности компьютерных систем как таковых. В их составе выделяют:

- тестовые и диагностические программы;
- антивирусные программы;
- операционные системы;
- командно-файловые процессоры (оболочки).

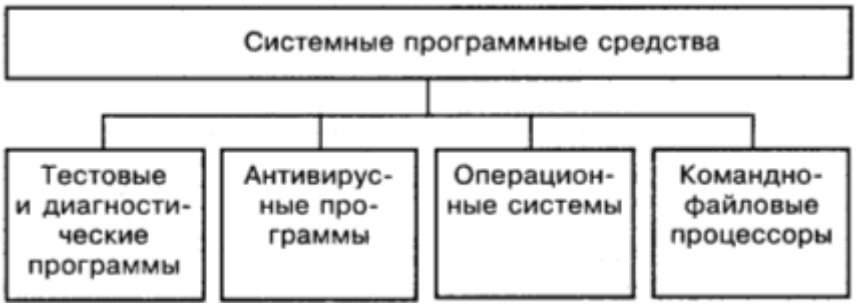

Рис. 1.5. Системные программные средства

Операционные системы являются основными системными программными комплексами, выполняющими следующие основные функции.

- тестирование работоспособности вычислительной системы и ее настройку при первоначальном включении;
- обеспечение синхронного и эффективного взаимодействия всех аппаратных и программных компонентов вычислительной системы в процессе ее функционирования;
- обеспечение эффективного взаимодействия пользователя с вычислительной системой.

Операционные системы классифицируются следующим образом:

- однопользовательские однозадачные системы (MS-DOS, DR-DOS);
- однопользовательские многозадачные системы (OS/2, Windows);
- многопользовательские системы (системы семейства UNIX)

Рис.1.6. Операционные системы

Командно-файловые процессоры (оболочки) предназначены для организации системы взаимодействия пользователя с вычислительной системой на принципах, отличных от реализуемых операционной системой, с целью облегчения его работы или предоставления дополнительных возможностей (например, Norton Commander или Windows версий до 3.11).

Тестовые и диагностические программы предназначены для проверки работоспособности отдельных узлов компьютера и компонентов программно-файловых систем и, возможно, устранения выявленных неисправностей.

Антивирусные программы необходимы для профилактики, выявления и, воз-можно, устранения вирусных программ, нарушающих нормальную работу вычисли-тельной системы.

В России широкое распространение получили несколько антивирусных про-грамм, среди которых: Антивирус Касперского (Kaspersky Anti-Virus, сокращенно: AVP), Dr. Web, Norton Antivirus, McAfee.

Прикладные программные средства классифицируются следующим образом:

- системы подготовки текстовых, табличных и др. документов;
- системы подготовки презентации;
- системы обработки финансово-экономической информации;
- системы управления базами данных;
- личные информационные системы;
- системы управления проектами;
- экспертные системы и системы поддержки принятия решения;
- системы интеллектуального проектирования и совершенствования управления;
- прочие системы.

Рис.1.7. Прикладные программные средства обеспечения управленческой деятельности

Системы подготовки текстовых документов предназначены для создания управленческих документов и различных информационных материалов текстового характера. К таким системам относятся:

- текстовые редакторы;
- текстовые процессоры;
- настольные издательские системы.

Системы обработки финансово-экономической информации предназначены для обработки числовых данных, характеризующих различные производственно-экономические и финансовые явления и объекты, и для составления соответствующих управленческих документов и информационно-аналитических материалов. Этот класс систем составляют:

- универсальные табличные процессоры;
- специализированные бухгалтерские программы;
- специализированные банковские программы (для внутри-банковских и межбанковских расчетов);
- специализированные программы финансово-экономического анализа и планирования.

Системы управления базами данных предназначены для создания, хранения и манипулирования массивами данных большого объема. Разные системы этого класса различаются способами организации хранения данных и обработки запросов на поиск информации, а также характером хранящихся в базе данных.

Личные информационные системы предназначены для информационного обслуживания рабочего места управленческого работника и, по существу, выполняют функции секретаря. Они, в частности, позволяют:

- планировать личное время на различных временных уровнях, при этом система может своевременно напоминать о наступлении запланированных мероприятии;
- вести персональные или иные картотеки и автоматически выбирать из них необходимую информацию;

- вести журнал телефонных переговоров и использовать функции, характерные для многофункциональных телефонных аппаратов;
- вести персональные информационные блокноты для хранения разнообразной личной информации.

Системы подготовки презентаций предназначены для квалифицированной подготовки графических и текстовых материалов, используемых в целях демонстрации на презентациях, деловых переговорах, конференциях. Для современных технологий подготовки презентаций характерно дополнение традиционных графики и текста такими формами информации, как видео- и аудиоинформация, что позволяет говорить о реализации гипер-медиа технологий.

Системы управления проектами предназначены для управления ресурсами различных видов (материальными, техническими, финансовыми, кадровыми, информационными) при реализации сложных научно-исследовательских и проектно-строительных работ.

Экспертные системы и системы поддержки принятия решений предназначены для реализации технологий информационного обеспечения процессов принятия управленческих решении на основе применения экономико-математического моделирования и принципов искусственного интеллекта.

Системы интеллектуального проектирования и совершенствования управления предназначены для использования так называемых CASE-технологии (Computer Aid System Engineering), ориентированных на автоматизированную разработку проектных решений по созданию и совершенствованию систем организационного управления.

Сетевые технологии в решении экономических задач

Локальные и глобальные компьютерные сети, их структура и функциональные возможности

Одним из самых значительных инновационно-технологических успехов двух последних десятилетий можно назвать рост продуктивности использования возможностей цифровой обработки данных в системе коммуникаций. Другими словами, в области компьютерных технологий в последние два десятилетия не было, наверное, более активно развивающегося направления, чем становление и развитие вычислительных сетей, составивших основу так называемых сетевых технологий. Компьютерные информационно-вычислительные сети явились закономерным результатом развития идей, техники и технологий в области создания и применения вычислительных машин и систем. Наблюдавшийся все эти годы бурный технологический прогресс микроэлектроники проявился не только в чисто компьютерной сфере, но и в производстве средств связи, с помощью которых распределенные в пространстве компьютеры объединяются в единую систему — вычислительную сеть.

Компьютерная сеть представляет собой совокупность компьютеров, связанных каналами передачи информации, необходимого программного обеспечения и технических средств, предназначенных для организации распределенной обработки информации. В такой системе любое из подключенных устройств может использовать сетевые возможности для передачи или получения информации. Современная информационно-вычислительная сеть обеспечивает обмен данными между ее абонентами и ориентирована на предоставление информационного обслуживания по запросам пользователей. В зависимости от масштабов (как по размеру территории, которую покрывает сеть, так и по числу узлов) различают локальные и глобальные компьютерные сети.

Локальные компьютерные сети (локальные вычислительные сети — ЛВС или LAN – Local Area Network) действуют на протяженности от нескольких метров до нескольких километров. Обычно они охватывают компьютеры одной организации или предприятия и не выходят за пределы одного здания. Из-за коротких расстояний в локальных сетях имеется

возможность использования относительно дорогих высококачественных линий связи, которые позволяют с помощью простых методов передачи данных достигать высоких скоростей обмена данными – порядка 100 Мбит/с. В связи с этим услуги, предоставляемые локальными сетями, отличаются широким разнообразием и обычно предусматривают реализацию в режиме непосредственного диалога (on-line).

Глобальные компьютерные сети Глобальные компьютерные сети (WAN – Wide Area Network) обеспечивают соединение большого числа компьютеров на огромных территориях, охватывающих целые регионы, страны и континенты, и используют для передачи информации оптоволоконные магистрали, спутниковые системы связи и коммутируемую телефонную сеть. Как правило, в глобальных сетях часто применяются уже существующие системы связи, изначально предназначенные совсем для других целей, поскольку прокладка высококачественных линий связи на большие расстояния обходится очень дорого. Например, многие глобальные сети строятся на основе телефонных и телеграфных каналов общего назначения. Кроме значительных масштабов по площади и по числу узлов для глобальных сетей характерна неоднородность, обусловленная различными типами архитектуры и программного обеспечения компьютеров-узлов.

Компоненты компьютерной сети

Очевидно, что информационно-вычислительная сеть — это сложный комплекс взаимосвязанных и согласованно функционирующих программных и аппаратных компонентов:

- компьютеров;
- коммуникационного оборудования;
- операционных систем;
- сетевых приложений.

Всемирная компьютерная сеть – Интернет

Временем зарождения Интернета принято считать осень 1969 года. Эпохальным событием, положившим начало глобальному информационному обществу, стал опыт обмена сообщениями между двумя удаленными друг от друга компьютерами, установленными в Калифорнийском университете (Лос-Анджелес): на расстояние в 5 м было передано слово «login» (удалось передать всего две буквы).

После первых удачных экспериментов довольно быстро был создан прообраз Интернета – опытная сеть Министерства обороны США, – ARPAnet (Advanced Research Projects Agency, первоначально объединявшая всего лишь четыре исследовательские лаборатории: в Калифорнийском Университете (Лос-Анджелес); в Стэндфордском Исследовательском Институте (Стэнфорд); в Калифорнийском Университете (Санта-Барбара); в Университете Юты (Солт-Лейк Сити)). 7 апреля по ARPAnet переданы первые биты...

На основе ARPAnet разрабатывалась методика создания устойчивых сетей, сохраняющих работоспособность при частичных повреждениях, т.е. децентрализованных (в то время в министерстве обороны США (US Defense Department) задались вопросом, как американские власти смогут общаться после ядерной войны). Тогда ставилась задача организовать большую многосвязную и надежную систему, состоящую из заведомо ненадежных компонентов. «Подход, основанный на предположении, что любой сегмент сети может вдруг исчезнуть, породил концепцию – каждый компьютер, подключенный к сети, должен иметь возможность связаться как равный с равным с любым другим компьютером. Это ключевой момент для понимания идеи Интернета». Практическое воплощение этой идейной посылки (стимулировавшее, в конечном счете, предоставление доступа к сетевым коммуникациям всем желающим) доказало: чем больше участников, тем надежнее связь. ARPAnet быстро развивалась, и к 1972 году в ней было уже 32 узла, ее главной функцией был обмен сообщениями при совместной работе над исследовательскими проектами. Пользователями сети

становились десятки университетов и корпораций, которым требовалась e-mail и передача файлов. В 1983 году, вследствие резкого увеличения количества пользователей ARPAnet, руководство ARPA приняло решение о выделении всех ресурсов военного значения, в особую, закрытую сеть MILNet. Собственно Arpanet оставалась открытой и общедоступной сетью. В 1986 году несколько небольших научных сетей, входящих в Arpanet, создали новую сеть – NFSNet (National Science Foundation – сеть Национального научного фонда США, хребет которой составляли 13 научных центров США, соединенных высокоскоростными линиями связи ([2], стр. 88)) – прямую предшественницу Интернет. Постепенно, в 87-90 гг. XX века, ARPAnet была заменена на NSFnet, в основе которой лежал internet (интернет). Параллельно в других странах мира создавались национальные информационно-вычислительные сети.

Считается, что начало сети Интернет в России положило создание в начале 1990 года компьютерной сети Relcom на базе Института атомной энергии И.В. Курчатова. К концу 1990 года в сеть интегрировалось более 30 локальных сетей разных организаций, что позволило осуществить ее официальную регистрацию и подключение к мировой глобальной сети.

Впоследствии возникшая метасеть, объединившая в начале 90-х гг. XX века национальные сети всех стран мира, стала называться Internet (Интернет).

К началу 90-х годов XX века сформировался общий принцип развития Интернета – объединение различных сетей на базе единой системы протоколов (правил обмена сообщениями) и «плетение сети сетей» посредством подключения через шлюзы новых сегментов. Несмотря на то, что формирование Интернета как единого образования уже долгие годы носит черты самоорганизующегося процесса, установление правил и законов общения, стандартизация протоколов и регулирование адресной системы производится разными международными организациями и объединениями

В настоящее время Интернет может выполнять целый ряд функций, необходимых человеку, занимающемуся бизнесом. Среди этих функций]:

- ***информационная,*** так как по сети можно получить любую интересующую специалиста: биржевую, коммерческую, научную и политическую информацию и т.п. (Интернет используют как справочник, как инструмент исследования, как источник новостей);
- ***коммуникационная,*** поскольку сетевые технологии позволяют пользователю поговорить по телефону со своим партнером в любом городе и стране, причем обойдется это дешевле обычной телефонной связи, а также послать ему факс или письмо с затратами, меньшими, нежели при использовании обычной почты, и к тому же существенно более оперативно;
- ***совещательная,*** ибо сеть Интернет — это место, где специалисты и пользователи компьютеров могут «встретиться» и обсудить интересующие их проблемы, в интерактивном режиме обменяться полезной информацией;
- ***коммерческая,*** обусловленная интенсивным развитием торговли по сети. Потенциальный покупатель просматривает на экране своего компьютера товары, заказывает и по кредитной карточке оплачивает их, а поступает товар к нему из ближайшего торгового пункта, естественно, уже не по сети;
- ***рекламная,*** связанная с тем, что реклама по Интернету весьма эффективна, в первую очередь из-за ее массовости и оперативности;
- ***развлекательная***, поскольку можно почитать и просмотреть огромное количество развлекательной литературы и фильмов; поиграть в самые увлекательные компьютерные игры, «путешествовать» и наслаждаться красотами разных музеев и стран и многое другое;
- специфично ***компьютерная*** функция, в соответствии с которой пользователи имеют возможность получить, причем чаще всего

бесплатно, самые новые программные средства, инструкции и рекомендации по работе в сети.

Раздел 2. Основы языка Python для работы с данными

Лекция 2. Основы работы с языком программирования.

Алгоритм

Итак, что же такое алгоритм? **Алгоритм** – точный набор инструкций, описывающий по- рядок действий исполнителя для достижения результата. В частности, алгоритм требует для работы некоторые исходные данные и выдает некоторый результат.

Алгоритм имеет следующие признаки:

- **определенность (детерминированность)** – следующий

шаг однозначно определяется текущим состоянием, что гарантирует постоянство результата для одних и тех же входных данных;

- **понятность исполнителю** – алгоритм состоит только из команд, входящих в систему команд исполнителя;
- **конечность** – способность завершить работу за конечное число шагов при корректных входных данных;
- **массовость (общность)** – применимость для разных входных данных.

Основная задача начинающих программистов – развитие абстрактного мышления. Нужно учиться создавать алгоритмы, соответствующие перечисленным выше признакам. И если о первых двух признаках позаботится компилятор, то за последние два придётся отвечать самим. Алгоритм является важнейшим понятием императивного программирования.

Императивное программирование описывает процесс вычисления в виде инструкций, изменяющих состояние программы. Такое название было получено из-за повелительного наклонения в естественном языке, с помощью которого выражаются приказы, команды исполнителю. Главный вопрос императивного программирования – «Как вычислять?»

По сути, императивное программирование – это процедура записи алгоритма на языке, понятном ЭВМ. Противоположное по духу направление программирования – декларативное программирование.

Декларативное программирование описывает результат вычислений и его свойства без предписывания последовательности действий. Ведь на самом деле важен результат, а не способ его достижения. Главный вопрос декларативного программирования – «Что должно получиться?»

Ветвями декларативного программирования являются функциональное и логическое программирование. Навыки декларативного программирования обычно приобретаются на старших курсах.

Первое поколение операторных языков

В определении алгоритма фигурирует некий исполнитель алгоритма. Если исполнитель – ЭВМ, то допустимые команды – система команд процессора ЭВМ, и здесь уместно назвать такой алгоритм **программой** или **программным кодом**. **Программа** – алгоритм, описанный в терминах конкретной системы команд ЭВМ.

Команды предназначены для проведения вычислений, работы с ячейками памяти и аппаратными устройствами ЭВМ, управления порядком выполнения команд и многого другого. Но что такое ко- манда для ЭВМ, которая понимает только язык чисел в своей системе счисления? Это специальный числовой код, который, будучи записанным в определённую область памяти в какой-либо момент времени, предлагает ЭВМ выполнить соответствующее действие, а результат (при его наличии) записать в указанное другим числом место.

Поскольку данные и программы в ЭВМ хранятся в одном виде, то данные, в том числе знакомые нам символы, также кодируются числами. Для этого систему команд дополняет таблица кодов, где каждому символу соответствует числовое представление. Таким образом, каждая команда программы и все необходимые данные оказываются закодированными в виде

последовательности чисел. Получается программа, представляющая собой последовательность кодов. Будем называть ее машинным кодом.

Исполнение машинного кода – последовательная интерпретация команд применительно к конкретным данным в соответствии с системой команд ЭВМ.

Важно понять, что для ЭВМ машинный код – текст на том единственном языке команд, который она понимает. Поскольку первые программы писались именно в кодах, то языки команд по праву считают первым поколением языков программирования.

Языки программирования второго поколения

Первые программисты наизусть помнили коды всех команд и символов, как азбуку Морзе. Но с возрастанием числа доступных команд в процессорах и усложнением решаемых задач становилось всё более трудоёмким не только программирование, но и развитие существующих программ. Всё это послужило предпосылками появления языков второго поколения – ассемблеров. Отныне каждой команде приписывался свой символьный мнемонический код, свои имена получили также регистры процессора, стало возможным за- писывать данные не только в числовой, но и символьной форме. Программа теперь представляла собой удобочитаемый текст, форматированный так, что каждая команда располагалась на отдельной строке.

В результате человеку стало гораздо удобнее, а вот ЭВМ текст на языке ассемблера уже не понимала. Для исполнения такой про- граммы теперь требовалась трансляция (перевод) исходного текста программы с языка ассемблера в машинные коды. Трансляцию можно было осуществить двумя способами:

- с помощью интерпретатора ассемблера;
- с помощью компилятора ассемблера с последующей интерпретацией.

В чём различие способов трансляции?

Интерпретатор – исполняемая программа, входными данными для которой является другая программа, записанная на языке ассемблера. Задача такого интерпретатора состоит в последовательном преобразовании каждой строки программы на языке ассемблера в машинный код и его немедленном исполнении с дальнейшим переходом к следующей строке программы. Таким образом, для исполнения программы таким способом необходимо сначала запустить программу-интерпретатор.

Другой подход состоит в однократном преобразовании текста на ассемблере в машинный код (компиляция) с помощью другой про- граммы (компилятора). Этот машинный код далее можно исполнять произвольное число раз, уже не пользуясь посредником (программой-интерпретатором), так как машинный код способна интерпретировать сама ЭВМ. Сегодня применяются оба способа трансляции.

С появлением ассемблеров немного изменилась и терминология: текст на ассемблере стал называться **исходным кодом** программы, а машинный код, полученный в результате компиляции, – **исполняемым кодом**. С той поры под программой понимают и то и другое.

Языки программирования третьего поколения

С помощью простейших команд приходится долго объяснять, а чтобы заставить ЭВМ сделать что-либо более-менее полезное, требовалось написать много строк ассемблерного кода. Средства операционных систем, конечно, сильно облегчали задачу, автоматизируя работу с устройствами и реализуя некоторые типовые операции. Однако, глядя на программу, едва ли можно было сказать, что она делает, а для того чтобы разобраться в чужом коде или понять хотя бы структуру программы, требовалась масса времени. Так возникла необходимость в более развитых языковых средствах. И они появились.

Третье поколение языков программирования характеризуется уходом от аппаратного уровня. Теперь программистам уже не нужно было думать о командах процессора, регистрах, ячейках памяти, прерываниях

операционной системы – это стало уделом компиляторов. Произошел большой качественный скачок: место команд процессора заняли **операторы**, место ячеек памяти и регистров – переменные, место прерываний операционной системы – стандартная библиотека процедур и функций, данные стали типизированными, появились структуры данных.

Что это дало? Программа так и осталась последовательностью, только уже не команд процессора, а операторов, но добавилось важнейшее свойство – структурированность. Вообще, операторы (так же как и команды процессора) описывают некоторые алгоритмические действия, но только более сложные. Например, операторы ветвления и операторы циклов делают явной структуру программы, допуская вложенность операторов друг в друга, а составной оператор позволяет обособить последовательность операторов в теле программы в рамках процедуры или функции.

Арифметические, логические и битовые операции и операции сравнения стало возможным записывать в привычной со школьной скамьи записи со скобками, в отличие от префиксной формы записи в ассемблерах, когда после команды перечислялись её аргументы. Если раньше для вычисления сложной формулы требовалось написать множество команд в нужном порядке, каждая из которых выполняла лишь одно действие, то теперь в одной строке можно было записывать сложные выражения и сразу же присваивать результат соответствующей переменной. Порядок вычислений определял компилятор на основе приоритетов операций и расставленных программистом скобок.

Встроенные атомарные типы данных и конструкторы типов данных дали возможность создания сложных статических и динамических структур данных. Процедуры и функции позволили отделять часто используемые фрагменты кода от основной программы. Таким образом, программный код на языке третьего поколения стал в несколько раз компактней и наглядней, а алгоритмы и структуры данных вышли на первый план. Было положено начало структурному программированию.

Позднее в языки вводились новые понятия – объекты, классы, интерфейсы, компоненты, события и обработчики событий. Это дало возможность оперировать классами объектов и отношений между ними. Выделялись методы программирования, ориентированные на события, объекты, компоненты и др.

За несколько десятилетий в мире возникло множество языков императивного программирования третьего поколения, сильно отличавшихся друг от друга как синтаксически, так и своими новшествами и особенностями: Фортран, Алгол, Кобол, ПЛ-1, Паскаль, Си, Ада. Однако все они принципиально схожи, так как основываются на концепции «Как вычислять?»

Какие проблемы возникли при использовании языков третьего поколения? Во-первых, задача преобразования программы на языке третьего поколения в машинный код перестала быть тривиальной: если в ассемблере каждому мнемоническому коду соответствовал один машинный код команды, то здесь это однозначное соответствие исчезло и появилось множество способов преобразования структуры операторов. Современный компилятор стал гораздо более сложной программой, способной оптимизировать генерируемый машинный код по скорости или расходу памяти.

Во-вторых, абстрагирование от аппаратного устройства ЭВМ привело к возникновению новых типов программных ошибок и неоптимальному расходу ресурсов ЭВМ. От ошибок можно изба- виться с помощью средств компиляции, отладки и тестирования.

Ошибки можно разделить на два класса по времени их возникновения или обнаружения: ошибки проектирования (разработки), выявляемые при компиляции, и ошибки времени выполнения, выявляемые лишь во время работы программы. Как правило, чем позже обнаруживается ошибка, тем сложнее найти её источник и дороже её исправить, поэтому разработчики языков программирования и компиляторов стремятся минимизировать

ошибки времени выполнения. Например, язык Си очень гибок, и для получения работоспособной программы, как шутят программисты, требуется 1 раз её скомпилировать и 100 раз запустить. Язык Ada предельно строг, и для получения работоспособной программы требуется 100 раз её скомпилировать и 1 раз запустить.

Зависимость от платформы, байт-код и псевдокод

На протяжении эволюции языков третьего поколения их разработчики пытались так снизить зависимость от аппаратной платформы, чтобы исходный код не менялся при смене платформы, а менялся лишь компилятор. К сожалению, в полной мере эту проблему решить не удалось в силу большого различия программно- аппаратных архитектур. Для переноса программы на языке третьего поколения, скажем, с IBM PC на Apple, приходится менять исходный код, чтобы учесть особенности архитектуры, и перекомпилировать программу с помощью соответствующего компилятора.

Технология Java решает проблему переносимости с помощью виртуальной машины, имеющей свою систему команд.

Байт-код – машинный код, состоящий из команд виртуальной машины java.

Под каждую аппаратную платформу разрабатывается программа-интерпретатор байт-кода, которая называется виртуальной java- машиной (jvm). Она учитывает особенности каждой конкретной программно-аппаратной архитектуры. Компилятору Java остаётся преобразовать исходный код на языке Java в этот байт-код. Таким образом, Java-программу достаточно один раз скомпилировать в байт-код, и она будет работать на любой платформе с java- машиной.

Чтобы не зависеть от конкретного языка при представлении алгоритмов широкой общественности (пусть даже такого независимого от платформы) и в то же время исключить различное толкование инструкций (шагов алгоритма) в периодической литературе по информатике часто используется псевдокод.

Псевдокод – полуформальное описание алгоритма в терминах ограниченного естественного языка с элементами математики и теории множеств.

Сейчас для исследования таких характеристик алгоритмов, как временная сложность и требовательность к ресурсам (памяти) используются системы команд или языки абстрактных машин. Например, Дональд Кнут в своей книге «Искусство программирования» применяет систему команд абстрактной машины MIX, близ- кой по своей архитектуре к настоящим процессорам (с точки зрения системы команд). Исследователи в области лямбда- исчисления, функционального программирования пользуются категориальной абстрактной машиной (КАМ). Также применяются абстрактная машина Тьюринга и нормальные алгоритмы Маркова. Известны их программные реализации на ЭВМ – своего рода виртуальные машины-интерпретаторы.

Языки четвёртого поколения

С усложнением программных систем появилась потребность в специализации и разделении труда. Универсальные языки становились всё менее удобными для решения узких задач. Начало меняться мышление: программисты стали больше думать о самой задаче, а не о том, как приспособить задачу для её решения на каком-либо языке программирования.

Языкам четвёртого поколения свойственен высокий уровень абстракции, они не универсальны, так как нацелены на определённую нишу. Например, задача проведения вычислений (то, ради чего создавались ЭВМ) уступила место задаче организации доступа к хранилищам данных. Появились структурированный язык запросов SQL, встроенные языки разработки приложений с базами данных от компаний SAP, Oracle, Software AG, IBM, Lotus. В области автоматизированного проектирования (CAD) можно отметить встроенные языки визуальных инженерных сред, таких как

AutoCAD, MathCAD и т.п. В области искусственного интеллекта – языки в инструментальных средствах разработки интеллектуальных систем.

Термин «программа» снова изменил своё значение – чаще всего программа на языке четвёртого поколения уже не является самостоятельной, а исполняется под управлением своей программной среды (интерпретируется). Такая программа совсем не обязательно описывает какой-либо алгоритм, т.е. не диктует последовательность действий, это берёт на себя среда выполнения. Грань между императивным и декларативным подходами здесь истончается.

В целях увеличения скорости выполнения программа на языке четвёртого поколения может компилироваться в некоторый промежуточный код наподобие байт-кода java и храниться как в текстовом, так и скомпилированном виде.

Язык программирования Python

В феврале 1991 года сотрудник голландского института CWI Гвидо ван Россумом опубликовал исходный текст Python в группе новостей alt.sources. Гвидо ван Россум назвал язык в честь популярного британского комедийного телешоу 1970-х «Летающий цирк Монти Пайтона», поскольку автор был поклонником этого телешоу, как и многие другие разработчики того времени, а в самом шоу прослеживалась некая параллель с миром компьютерной техники.

Наличие дружелюбного, отзывчивого сообщества пользователей считается, наряду с дизайнерской интуицией Гвидо, одним из факторов успеха Python. Развитие языка происходит согласно чётко регламентированному процессу создания, обсуждения, отбора и реализации документов PEP (англ. *Python Enhancement Proposal*) – предложений по развитию Python.

Основные особенности языка:

- Python - интерпретируемый язык программирования.
- Python характеризуется ясным синтаксисом.

• Python используется в различных сферах.

• Интерпретаторы Python распространяются свободно на основании лицензии подобной GNU General Public License.

На сегодняшний день согласно рейтингу TIOBE, который составляется из всех актуальных языков программирования (около 100), Python на сегодняшний день занимает первое место. Такой успех можно объяснить возможностью выполнения широкого спектра задач и удобством языка. Удобство заключается в том, что Python - высокоуровневый язык. Это означает, что сложные описания структур машинного кода выполнены в удобно читаемом для человека виде. Стоит отметить, что при изучении языка необходимо уделять больше времени пониманию того, как работают стандартные функции, поскольку это позволит быстрее прокачивать свой навык программирования.

Mar 2023	Programming Language	Ratings
1	Python	14.83%
2	C	14.73%
3	Java	13.56%
4	C++	13.29%
5	C#	7.17%
6	Visual Basic	4.75%
7	JavaScript	2.17%
8	SQL	1.95%
9	PHP	1.61%
10	Go	1.24%
11	Assembly language	1.11%

Рис.2.1. Рейтинг популярных языков программирования Рейтинг популярности языков программирования по данным индекса TIOBE на март 2023 года.

Разработчики языка Python придерживаются определённой философии программирования, называемой «The Zen of Python» (рис.2.2). Её текст выдаётся интерпретатором Python по команде import this (работает один раз за сессию). Автором этой философии считается Тим Петерс.

Явное лучше неявного.
Простое лучше сложного.
Сложное лучше усложнённого.
Плоское лучше вложенного.
Разрежённое лучше плотного.
Удобочитаемость важна.
Частные случаи не настолько существенны, чтобы нарушать правила. Однако практичность важнее чистоты.
Ошибки никогда не должны замалчиваться.
За исключением замалчивания, которое задано явно.
В случае неоднозначности сопротивляйтесь искушению угадать.
Должен существовать один - и, желательно, только один - очевидный способ сделать это.
Хотя он может быть с первого взгляда не очевиден, если ты не голландец.
Сейчас лучше, чем никогда.
Однако, никогда чаще лучше, чем прямо сейчас.
Если реализацию сложно объяснить – это плохая идея.
Если реализацию легко объяснить - это может быть хорошая идея.
Пространства имён - прекрасная идея, давайте делать их больше!

Рис.2.2. Дзен Python

На основе вышеизложенного, можно выделить следующие сильные стороны языка программирования Python:

- Объектно-ориентированный
- Мощный
- Динамическая типизация
- Автоматическое управление памятью
- Модульное программирование
- Встроенные типы объектов
- Встроенные инструменты
- Библиотеки утилит
- Утилиты сторонних разработчиков
- Прост в изучении

Синтаксис Python

Синтаксис языка Python очень прост, главное помнить основные моменты:

1. Конец строки выражения является окончанием инструкции, никаких знаков на конце не требуется, исключением являются вложенные инструкции, где Инструкция – это элемент языка, определяющий действие, которое требуется выполнить.

Примеры инструкций:

```
>>> print("Python is awesome!")
Python is awesome!
>>> var = "first string"
>>> print(var)
first string
>>> var = 2 + 9
>>> print(var)
11
```

Функция print() позволяет нам выводить различную информацию на экран.

2. Строковые выражения могут заключаться в одинарные, двойные и тройные кавычки.

Также может встречаться комбинация из одинарных и двойных кавычек. Строковое выражение обрамленное в тройные кавычки может состоять из нескольких строчек. В случае, когда используются комбинации из двойных и одинарных кавычек, одна пара должна быть внешней - она обозначает начало и конец строкового выражения, а внутренняя пара является частью этого выражения.

Примеры использования кавычек для определения строковых выражений:

```
>>> print("String")
String
>>> print('String')
String
>>> print('''Str
... ing''')
Str
ing
>>> print('Str"ing')
Str"ing
```

3. Комментарии - вспомогательные строки, не обрабатываемые программой, обозначаются знаком # перед началом строки и действуют до конца строки. Зачастую бывает полезно записать комментарий, чтобы оставить напоминание о том, что выполняется в программе в данный момент.

Пример:

```
>>> print("комментарий справа") # Функция print()
позволяет выводить результат на экран комментарий
справа
```

4. Вложенная инструкция - часть общей инструкции выполняющаяся при определенных условиях, в этом случае условие заканчивается двоеточием, а само вложенное выражение должно отступать на 4 пробела от места, откуда начинается условие.

Пример:

```
#инструкция объявляющая переменную var и присваивающая
ей значение равное 5
var = 5
#основная инструкция 1
if var < 3:
#вложенная инструкция, отступает от основной инструкции
1
    print("less")
#основная инструкция 2
else:
#вложенная инструкция, отступает от основной инструкции
2
    print("more")
```

Отступ в 4 пробела необходим для того, чтобы программа понимала, где начинается вложенная инструкция и к какой основной она относится. Можно использовать и табуляцию, но в руководстве по написанию кода на Python отступ в 4 пробела стоит на первом месте, так как он считается наиболее распространенным среди программистов. Главное не мешать разные варианты отступов в одном коде.

5. В программе может быть несколько уровней вложенных инструкций, в таком случае надо на каждом уровне делать отступы от начала предыдущей вложенной инструкции.

Пример:

```
# объявляем переменную var равную 5
var = 5
# основная инструкция
if var > 3:
# вложенная инструкция уровня 1
    if var > 4:
# вложенная инструкция уровня 2, отступает на 4 пробела
от
# вложенной инструкции уровня 1
        print("more than 4")
```

Допустимо писать одноуровневые инструкции в одну строчку, если в итоге получается не очень длинная строка:

```
if var < 3: print("like this")
```

Математические действия

Арифметические операторы

Оператором является элемент выражения, который указывает на то, какое действие необходимо произвести между элементами. То есть, в выражении "21 - 4" символ "-" является оператором, указывающим на то, что нужно произвести вычитание. "21" и "4" при этом называются операндами.

Символ	Описание	Пример
+	Оператор суммы	`>>> print(5 + 8)` `13`
-	Оператор разности	`>>> print(31 - 2)` `29`
*	Оператор произведения	`>>> print(12 * 9)` `108`
/	Оператор деления	`>>> print(6 / 4)` `1.5`
%	Оператор получения остатка от деления	`>>> print(6 % 4)` `2`
**	Оператор возведения в степень	`>>> print(9 ** 2)` `81`
//	Оператор целочисленного деления	`>>> print(6 // 4)` `1`

Операторы присваивания

Символ	Описание	Пример

=	Присваивает значение правого операнда левому	`>>> var = 5`
		`>>> print(var)`
		`5`

+=	Прибавляет значение правого операнда к левому и присваивает левому. a += b эквивалентно записи a=a+b	`>>> var = 5`
		`>>> var += 4`
		`>>> print(var)`
		`9`

-=	Отнимает значение у левого операнда правое и присваивает левому. a -= b эквивалентно записи a = a -b	`>>> var = 5`
		`>>> var -= 2`
		`>>> print(var)`
		`3`

*=	Умножает значение левого операнда на правое и присваивает левому. a *= b эквивалентно записи a=a*b	`>>> var = 5`
		`>>> var *= 10`
		`>>> print(var)`
		`50`

/=	Делит значение левого операнда на правое и присваивает левому. a /= b эквивалентно записи a=a/b	`>>> var = 5`
		`>>> var /= 4`
		`>>> print(var)`
		`1.25`

%=	Делит с остатком значение левого операнда на правое и присваивает	`>>> var = 5`
		`>>> var %= 10`
		`>>>print(var)`
		`5`

	остаток левому. a %= b эквивалентно записи a = a % b	

**=	Возводит значение левого операнда в степень правого и присваивает левому. a **= b эквивалентно записи a = a ** b	>>> var = 5
		>>> var **= 8
		>>>print(var)
		390625

//=	Целочисленно делит значение левого операнда на правое и присваивает левому. a //= b эквивалентно записи a = a // b	>>> var = 5
		>>> var //= 30
		>>>print(var)
		0

Логические операции. Структура ветвления.

Операторы сравнения (реляционные)

Символ	Описание	Пример

==	Проверяет, равны ли операнды между собой. Если они равны, то выражение становится истинным.	>>> print(5 == 5)
		True
		>>> print(6 == 44)
		False

!=	Проверяет, равны ли операнды между собой. Если они не равны, то выражение становится истинным.	>>> print(12 != 12)
		False
		>>> print(1231 != 0.4)
		True

>	Проверяет, больше ли левый операнд чем правый, если больше, то выражение становится истинным.	`>>> print(53 > 23)`
		`True`
		`>>> print(432 >500)`
		`False`

<	Проверяет, меньше ли левый операнд чем правый, если меньше, то выражение становится истинным.	`>>> print(5 < 51)`
		`True`
		`>>> print(6 < 4)`
		`False`

>=	Проверяет, больше левый операнд, чем правый, или равен ему. Если больше или равен, то выражение становится истинным.	`>>> print(5 >= 5)`
		`True`
		`>>> print(6 >=44)`
		`False`

<=	Проверяет, меньше левый операнд, чем правый, или равен ему. Если меньше или равен, то выражение становится истинным.	`>>> print(32 <= 232)`
		`True`
		`>>> print(65 <= 9)`
		`False`

Логические операторы

Символ	Описание	Пример

and	Логический оператор "И". Условие будет истинным если оба операнда истина	`True and True = True.`
		`True and False = False.`
		`False and True = False.`
		`False and False = False.`

or	Логический оператор "ИЛИ". Если хотя бы один из операндов истинный, то и все выражение будет истинным	`True or True = True.`
		`True or False = True.`
		`False or True = True.`
		`False or False = False.`

not	Логический оператор "НЕ". Изменяет логическое значение операнда на противоположное	`not True = False.` `not False = True.`

Оператор if

Условные операторы нужны для проверки условий и, в зависимости от результата, вести логику выполнения программы в нужном направлении.

Условный оператор **if** ("если") является основным оператором проверки выполнения условия. Для того, чтобы выполнить простую вложенную инструкцию, необходимо проверить условие на соответствие, использовав оператор **if** и прописав после него соответсвующее условие:

```
# объявляем переменную
a = 5
# выполняем проверку условия
if a < 10:
# если условие выполняется, то выполняется вложенная инструкция
    print("a less than 10")
```

В данном случае мы объявляем переменную **a** и присваиваем ей значение равное **5**, далее выполняем проверку условия - если переменная меньше **10**, то вывести соответсвующее сообщение.

Оператор **if** производит проверку истинности выражения, т.е. является ли результат выражения логической истиной (**True**) или же ложью (**False**). Далее выполняется вложенная инструкция, если результат выражения

является **True**. Если результат выражения является **False**, тогда вложенная инструкция игнорируется. В предыдущем разделе приводились примеры того, что выводит Python в качестве результата выражения с оператором сравнения.

Оператор else

Оператор **if** позволяет выполнить вложенную инструкцию, если результат выражения True, а что, если нам необходимо, чтобы программа могла выполнять действия и в случае False результата? Для этого мы можем использовать оператор else. Условный оператор else ("иначе") является продолжением основной инструкции.

```
# объявляем переменные
var_1 = 10
var_2 = 9
# выполняем проверку условия
if var_1 == var_2:
# если True
    print("var_1 equal var_2")
# иначе
else:
# выполняется другая вложенная инструкция
    print("var_1 not equal var_2")
```

Как показано в примере выше, мы задаем переменной **var_1** значение равное **10**, а переменной **var_2** значение **9**, затем производим сравнение наших переменных на равенство, если **var_1** равно **var_2**, тогда следует вывести сообщение, что они равны, иначе, вывести сообщение, что переменные не равны. Таким образом, оператор **else** позволяет выполнить инструкцию **print("var_1 not equal var_2")** в случаях, когда результат проверки на равенство является **False**.

Оператор elif

В предыдущем примере мы рассматривали ситуацию, когда нас интересовало всего 2 возможные ситуации - равно, не равно. А что, если для решения определенной задачи нас интересует более двух возможных ситуации? Для

этого предусмотрен оператор **elif** (сокращение от конструкции else if), который позволяет добавить дополнительные условия в логику выполнения программы:

```
# объявляем переменные
var_1 = 10
var_2 = -10
# выполняем проверку условия
if var_1 == var_2:
# если True
    print("var_1 equal var_2")
# иначе если выполняется другое условие
elif var_1 < var_2:
# если True для elif
    print("var_1 less than var_2")
# если False для всех
else:
    print("var_1 more than var_2")
```

Здесь мы объявляем переменные **var_1** и **var_2** и присваиваем значения **10** и **-10** соответственно, после этого выполняем проверку первого условия, равны ли эти переменные, если они равны, то выводим соответствующее сообщение. Если они не равны, то пытаемся понять, какая из этих переменных больше. Выполняется проверка второго условия (**elif**), если **var_1** меньше **var_2**, тогда выводим соответствующее сообщение, и наконец, остается последний вариант, который и выводит сообщение о том, что **var_1** больше чем **var_2**.

Главное, на что стоит обратить внимание - после использования конструкций **if** и **elif** всегда необходимо записывать условие проверки (в нашем случае **var_1 == var_2** и **var_1 < var_2** соответственно), а **else** всегда используется без условий, потому что означает выполнение инструкции при любых других вариантах, которые не были рассмотрены операторами **if** и **elif**.

Количество конструкций **elif** может быть множество, тогда, как **if** и **else** используются по 1 разу в рамках одной инструкции:

```
if (условие):
```

```
    (выполнение условия)
elif (другое условие):
    (выполнение другого условия)
elif (третье условие):
    (выполнение третьего условия)
elif (четвертое условие):
...
...
...
else:
    (выполнение при всех других не рассмотренных ранее
случаях)
```

Стоит обратить внимание, что в случае, когда вы используете оператор if несколько раз на одном уровне вложенности инструкций, то они работают независимо друг от друга и не образуют одну общую инструкцию:

```
var = 10
if var == 10:
    print("var equal 10")
if var < 10:
    print("var less than 10")
else:
    print("var more than 10")
```

Результат выполнения такого кода будет следующим:

```
var equal 10
var more than 10
```

Оператор **if** всегда задает начало новой инструкции. В примере выше переменная **var** попадает на проверку в несколько инструкций, сперва мы получаем результат **True** в выражении **var == 10**, после чего выводится первое сообщение. Далее **var** опять проверяется следующей инструкцией, получается результат **False** и программа выводит второе сообщение, после оператора **else**. Давайте для наглядности построим блок-схему данной программы:

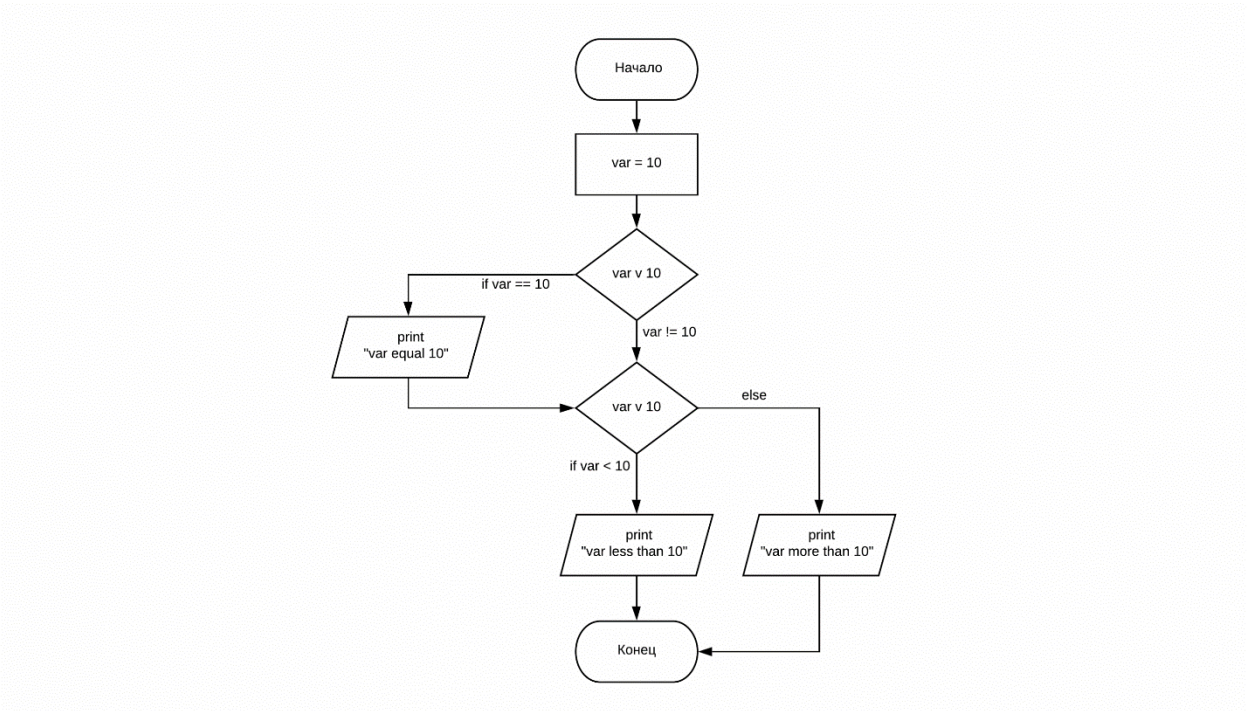

Рис.2.3. Схема алгоритма

В данном случае будет корректно заменить второй **if** на **elif**, тогда мы получим единую инструкцию, которая выведет только одно верное сообщение:

```
var = 10
if var == 10:
    print("var equal 10")
elif var < 10:
    print("var less than 10")
else:
    print("var more than 10")
```

И блок-схема данной программы будет выглядеть следующим образом:

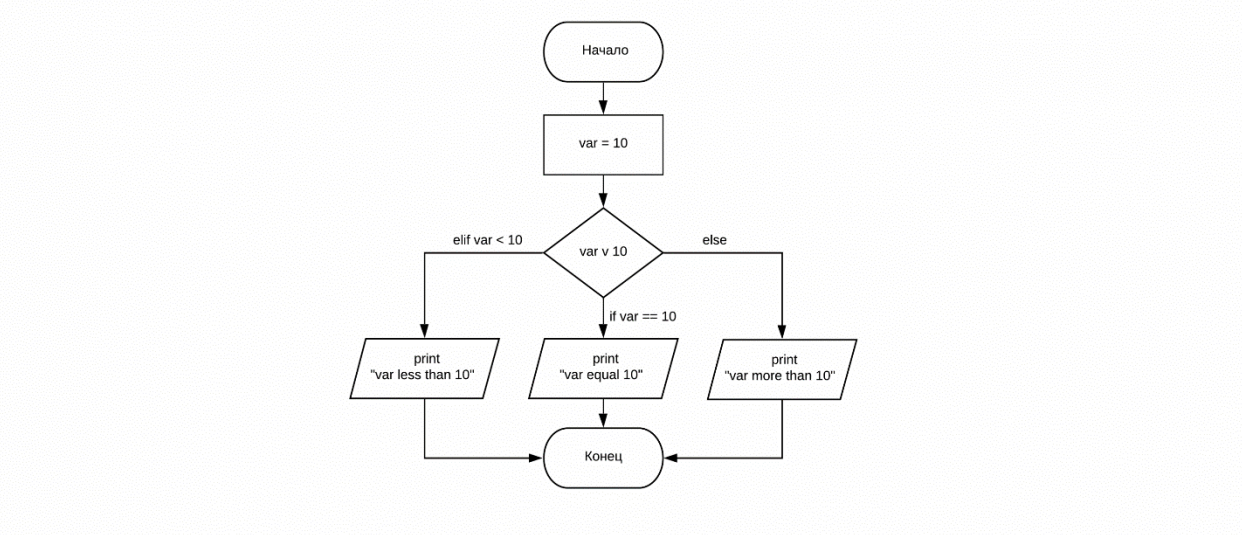

Рис.2.4. Схема алгоритма

Конструкции **if-elif-else** можно использовать на разных уровнях вложенности для более сложных программ:

```
if (условие):
    if (дополнительное условие):
        (выполнение дополнительного условия)
    elif (другое дополнительное условие):
        (выполнение другого дополнительного условия)
    elif ...
        ...
    ...
    else:
        ...
elif (другое условие):
    (выполнение другого условия)
elif ...
    ...
else:
    ...
```

Цикл for

Цикл **for** способен проходить по любому итерируемому объекту, будь то списки, словари, кортежи, строки и не требует ручного увеличения счетчика итераций:

```
for i in [1, 2, 3, 4, 5 ,6 ,7 ,8 ,9 , 10, 11, 12, 13]:
    print(i)
```

Для того, чтобы задать счетчик итераций используют функцию **range()**, которая позволяет автоматически сгенерировать последовательность чисел.

Особенность range() состоит в том, что последнее число последовательности не включается в нее, то есть если надо сформировать последовательность чисел от 1 до 14, не включая 14, то необходимо записать:

```
for i in range(1,14):
    print(i)
```

Функция **range()** работает так, потому что чаще всего используется для работы с индексамио, отсчет которых начинается с **0**.

Примечание. Если необходимо выводить элементы цикла без перехода на следующую строку, то можно использовать параметр end=' ' для функции print(). Этому параметру можно ничего не передавать, можно передавать пробелы или другие знаки:

```
for i in range(1,14):
    print(i, end=' ')
1 2 3 4 5 6 7 8 9 10 11 12 13
```

Цикл while

Самый простой вид циклов, выполняет инструкцию в блоке до тех пор, пока условие не станет ложным.

```
# создаем переменную, равную 1
var = 1
# прописываем цикл с условием - выполнять до тех пор,
пока переменная
# меньше или равна 13
while var <= 13:
# выводим значение переменной
    print(var)
# увеличиваем переменную на 1
    var += 1
```

Необходимо всегда помнить, что при использовании цикла **while**, нужно следить за тем, чтобы было прописано условие выхода из цикла. Если бы в примере выше мы не увеличивали переменную **var** на **1**, тогда цикл бы стал бесконечным.

Оператор continue

Оператор **continue** позволяет начать следующий проход цикла, минуя оставшиеся инструкции:

```
for var in 'Python':
    if var == 'h':
        continue
    print(var)
```

На примере выше мы перебираем последовательность символов и когда наша переменная хранит в себе символ 'h', мы используем оператор **continue**, чтобы пропустить дальнейшую инструкцию **print(var)**.

Оператор break

Оператор **break** досрочно прерывает цикл. Повторите пример ниже и посмотрите, что получится.

```
for var in 'Python':
    if var == 'h':
        break
    print(var)
```

В данном случае оператор break ведет нас сразу к окончанию работы цикла. При этом получается, что выход совершается экстренный, т.к. мы не переберем все элементы эталона, о чем говорит пунктирная стрелка.

Оператор else

Оператор **else** проверяет цикл на экстренный выход (**break**). Если экстренного выхода не было, т.е. оператор **break** не был выполнен, блок инструкций вложенный в оператор **else** - выполняется.

```
for var in 'Python':
    if var == 'a':
        break
else:
    print('Символа a нет в слове Python')
```

Оператор else обслуживает цикл for. Если оператор else сместить вправо на 4 пробела, то он уже будет работать в паре с оператором if.
Операторы continue, break и **else** работают с циклами **for** и **while**.

Лекция 3. Основные структуры данных. Списки. Работа со списками.

Списками в Python называются массивы. Они могут содержать данные различных типов.

Для создания списка автоматически можно использовать метод **list()**:

```
>>> list('Python')
['P','y','t','h','o','n']
```

Также можно это сделать напрямую, присвоив переменной значение типа **list**:

```
# Пустой список
>>> s = []
# список с данными разных типов
>>> l = ['s', 'p', ['isok'], 2]
>>> s
[]
>>> l
['s', 'p', ['isok'], 2]
```

также можно использовать генераторы списков:

```
>>> a = ['P', 'y', 't', 'h', 'o', 'n']
>>> b = [i * 3 for i in a]
>>> b
['PPP', 'yyy', 'ttt', 'hhh', 'ooo', 'nnn']
```

Количество элементов в списке можно узнать с помощью функции len():

```
>>> a = ['P', 'y', 't', 'h', 'o', 'n']
>>> len(a)
6
```

В списках, так же как и в строках, можно обратиться к элементу через индекс s[2] или s[3], для сравнение или вывода на печать:

```
>>> s = ['P', 'y', 't', 'h', 'o', 'n']
>>> print(s[2], s[3])
t h
```

Генераторы списков позволяют создавать и заполнять списки. Генератор списков предполагает наличие итерируемого объекта или итератора, на основании которого будет создаваться новый список, а также выражения, которое будет производить манипуляции с извлеченными из последовательности элементами перед тем как добавить их в формируемый список.

Методы для работы со списками

Методы списков вызываются по схеме: **list.method()**.

Рассмотрим основные методы работы со списками

append(a) - добавляет элемент a в конец списка

```
>>> var = ['l', 'i', 's', 't']
>>> var.append('a')
>>> print(var)
['l', 'i', 's', 't', 'a']
```

extend(L) - расширяет список, добавляя к концу все элементы списка L

```
>>> var = ['l', 'i', 's', 't']
>>> var.extend(['l', 'i', 's', 't'])
>>> print(var)
['l', 'i', 's', 't', 'l', 'i', 's', 't']
```

insert(i, a) - вставляет на **i** позицию элемент **a**

```
>>> var = ['l', 'i', 's', 't']
>>> var.insert(2,'a')
>>> print(var)
['l', 'i', 'a', 's', 't']
```

remove(a) - удаляет первое найденное значение элемента в списке со значением **a**, возвращает ошибку, если такого элемента не существует

```
>>> var = ['l', 'i', 's', 't', 't']
>>> var.remove('t')
>>> print(var)
['l', 'i', 's', 't']
```

pop(i) - удаляет **i**-ый элемент и возвращает его, если индекс не указан, удаляет последний элемент

```
>>> var = ['l', 'i', 's', 't']
>>> var.pop(0)
'l'
>>> print(var)
['i', 's', 't']
```

index(a) - возвращает индекс элемента **a** (индексация начинается с 0)

```
>>> var = ['l', 'i', 's', 't']
>>> var.index('t')
3
```

count(a) - возвращает количество элементов со значением **a**

```
>>> var = ['l', 'i', 's', 't']
>>> var.count('t')
1
```

sort([key = функция]) - сортирует список на основе функции, можно не прописывать функцию, тогда сортировка будет происходить по встроенному алгоритму

```
>>> var = ['l', 'i', 's', 't']
>>> var.sort()
>>> print(var)
['i', 'l', 's', 't']
```

reverse() - разворачивает список

```
>>> var = ['l', 'i', 's', 't']
>>> var.reverse()
>>> print(var)
['t', 's', 'i', 'l']
```

copy() - поверхностная копия списка, при присвоении переменной копии списка, значение данного списка не изменяется в случае изменения первого. Если переменной присвоить список через "=", тогда значение этой переменной будет меняться при изменении оригинала

```
>>> var = ['l', 'i', 's', 't']
>>> asd = var.copy()
```

```
>>> print(asd)
['l', 'i', 's', 't']
>>> var = ['l', 'i', 's', 't']
>>> asd = var
>>> print(asd)
['l', 'i', 's', 't']
>>> print(var)
['l', 'i', 's', 't']
>>> var.reverse()
>>> print(asd)
['t', 's', 'i', 'l']
>>> print(var)
['t', 's', 'i', 'l']
>>> var = ['l', 'i', 's', 't']
>>> asd = var.copy()
>>> print(asd)
['l', 'i', 's', 't']
>>> print(var)
['l', 'i', 's', 't']
>>> var.reverse()
>>> print(asd)
['l', 'i', 's', 't']
>>> print(var)
['t', 's', 'i', 'l']
```

clear() - очищает список

```
>>> var = ['l', 'i', 's', 't']
>>> var.clear()
>>> print(var)
[]
```

Раздел 3. Базовые алгоритмы обработки данных

Лекция 4. Системы линейных алгебраических уравнений. Численное интегрирование Решение нелинейных уравнений.

4.1 Основная запись СЛАУ. Понятие решения СЛАУ и некоторые определения.

В общем случае основная запись системы *n* линейных алгебраических уравнений с *n* неизвестными имеет следующий вид:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1j}x_j + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \ldots + a_{2j}x_j + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{i1}x_1 + a_{i2}x_2 + \ldots + a_{ij}x_j + \ldots + a_{in}x_n = b_i \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + \ldots + a_{nj}x_j + \ldots + a_{nn}x_n = b_n \end{cases} \quad (4.1)$$

При этом через $x_1, x_2, \ldots, x_n$ обозначены неизвестные, подлежащие определению; величины $a_{11}, a_{12}, \ldots, a_{nn}$ называемые **коэффициентами системы**, и величины $b_1, b_2, \ldots, b_n$, называемые **свободными членами,** предполагаются известными. Каждый коэффициент системы a_{ij} имеет два индекса, первый из которых i указывает номер уравнения, а второй j номер неизвестного, при котором стоит этот коэффициент.

Система (4.1) называется **однородной**, если все ее свободные члены $b_1, b_2, \ldots, b_n$ равны нулю. Если хотя бы один из свободных членов $b_1, b_2, \ldots, b_n$ отличен от нуля, то система (4.1) называется **неоднородной**.

Решением системы (4.1) называется такая совокупность n чисел $c_1, c_2, \ldots, c_n$, которая при подстановке в систему (4.1) на место неизвестных $x_1, x_2, \ldots, x_n$ обращает все уравнения этой системы в тождества.

Система уравнений вида (4.1) называется **совместной**, если она имеет хотя бы одно решение, и **несовместной**, если у нее не существует ни одного решения.

Совместная система вида (4.1) называется **определенной**, если она имеет единственное решение. Совместная система вида (4.1) называется **неопределенной**, если у нее существуют, по крайней мере, два различных решения. Рассмотренные понятия можно представить в виде схемы (рис.4.1)

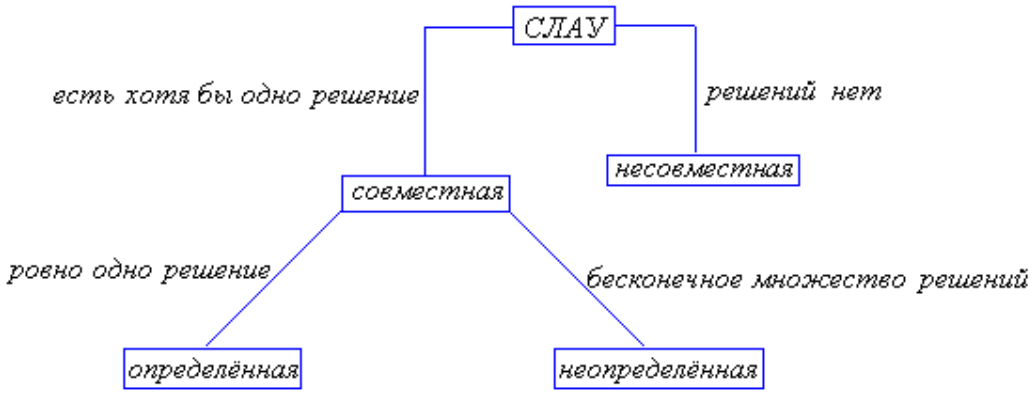

Рис. 4.1 Классификация СЛАУ

4.2. Общая схема метода Гаусса

Решение СЛАУ методом **Гаусса** состоит из двух частей:

- **1-я часть** – приведение к эквивалентной системе уравнений более простой структуры (треугольной) за счет вычитания уравнений друг из друга, при умножении вычитаемого уравнения на специально подобранное число (**Прямой ход**).
- **2-я часть** – решение равносильной системы уравнений с треугольной матрицей (**Обратный ход**).

Прямой ход (приведение системы уравнений к треугольному виду):

1-й шаг. Из *i-ой* строки (i=2, 3, ... , n) вычитаем *1-ую*, умноженную на величину $c_{i1} = a_{i1} / a_{11}$, получим новые коэффициенты по формуле

$$a_{ij}^{1} = a_{ij} - c_{i1} \cdot a_{1j}, \quad j = 1, 2, \ldots, n+1,$$

для всех i=2, 3, ... , n .

$$\left[\begin{array}{cccccc|c} a_{11} & a_{12} & \cdots & \cdots & \cdots & a_{1n} & a_{1,n+1} \\ 0 & a_{22}^1 & \cdots & \cdots & \cdots & a_{2n}^1 & a_{2,n+1}^1 \\ \vdots & \vdots & \ddots & & & \vdots & \vdots \\ \vdots & \vdots & & \ddots & & \vdots & \vdots \\ \vdots & \vdots & & & \ddots & \vdots & \vdots \\ 0 & a_{n2}^1 & \cdots & \cdots & \cdots & a_{nn}^1 & a_{n,n+1}^1 \end{array}\right]$$

2-й шаг. Из *i-ой* строки (*i*=3, 4, ... , *n*) вычитаем *2-ую*, умноженную на величину $c_{i2} = a_{i2}^1 / a_{22}^1$, получим новые коэффициенты по формуле

$$a_{ij}^2 = a_{ij}^1 - c_{i2} \cdot a_{2j}^1, \quad j = 2,3,...,n+1,$$

для всех *i*=3, 4, ... , *n*

$$\left[\begin{array}{cccccc|c} a_{11} & a_{12} & a_{13} & \cdots & \cdots & a_{1n} & a_{1,n+1} \\ 0 & a_{22}^1 & a_{23}^1 & \cdots & \cdots & a_{2n}^1 & a_{2,n+1}^1 \\ \vdots & 0 & a_{33}^2 & \cdots & \cdots & a_{3n}^2 & a_{3,n+1}^2 \\ \vdots & \vdots & \vdots & \ddots & & \vdots & \vdots \\ \vdots & \vdots & \vdots & & \ddots & \vdots & \vdots \\ 0 & 0 & a_{n3}^2 & \cdots & \cdots & a_{nn}^2 & a_{n,n+1}^2 \end{array}\right]$$

k-й шаг. Из *i-ой* строки (*i=k+1, k+2, ..., n*) вычитаем *k-ую*, умноженную на величину $c_{ik} = a_{ik}^{k-1} / a_{kk}^{k-1}$, получим новые коэффициенты по формуле

$$a_{ij}^k = a_{ij}^{k-1} - c_{ik} \cdot a_{kj}^{k-1}, \; j = k, k+1, \ldots, n+1,$$

для всех *i=k*+1, ... , *n*

$$\left[\begin{array}{ccccccc} a_{11} & \cdots & a_{1k} & a_{1k+1} & \cdots & a_{1n} & a_{1,n+1} \\ 0 & \ddots & \vdots & \vdots & & \vdots & \vdots \\ \vdots & \ddots & a_{kk}^{k-1} & a_{k,k+1}^{k-1} & \cdots & a_{kn}^{k-1} & a_{k,n+1}^{k-1} \\ \vdots & & 0 & a_{k+1,k+1}^{k} & \cdots & a_{k+1,n}^{k} & a_{k+1,n+1}^{k} \\ \vdots & & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & a_{n,k+1}^{k} & \cdots & a_{nn}^{k} & a_{n,n+1}^{k} \end{array}\right]$$

Продолжая процесс, после (n-1)- шагов получим расширенную матрицу вида:

$$\left[\begin{array}{ccccccc} a_{11} & \cdots & a_{1k} & \cdots & \cdots & a_{1n} & a_{1,n+1} \\ 0 & \ddots & \vdots & & & \vdots & \vdots \\ \vdots & \ddots & a_{kk}^{k-1} & \cdots & \cdots & a_{kn}^{k-1} & a_{k,n+1}^{k-1} \\ \vdots & & 0 & \ddots & & \vdots & \vdots \\ \vdots & & \vdots & \ddots & \ddots & \vdots & \vdots \\ 0 & \cdots & 0 & \cdots & 0 & a_{nn}^{n-1} & a_{n,n+1}^{n-1} \end{array}\right],$$

т.е. систему с треугольной матрицей, эквивалентную исходной (4.1), (4.3) – (4.5).

Запишем общие формулы прямого хода:

$$\begin{cases} c_{ik} = \dfrac{a_{ik}^{k-1}}{a_{kk}^{k-1}} \\ a_{ij}^{k} = a_{ij}^{k-1} - c_{ik} \cdot a_{kj}^{k-1} \end{cases}, \quad \begin{array}{c} k = 1,2,...,n-1, \\ i = k+1, k+2, ..., n, \\ j = k, k+1, ..., n+1 \end{array} \qquad (4.6)$$

Замечание**.** Элемент a_{kk}^{k-1} - называется **ведущим элементом**.

Обратный ход (решение системы с треугольной матрицей):

В результате прямого хода получена следующая система с треугольной матрицей, равносильная исходной системе (4.1):

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \ldots + a_{1i}x_i + \ldots + a_{1n}x_n = a_{1,n+1} \\ a_{22}^1 x_2 + \ldots + a_{2i}^1 x_i + \ldots + a_{2n}^1 x_n = a_{2,n+1}^1 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{ii}^{i-1} x_i + \ldots + a_{in}^{i-1} x_n = a_{i,n+1}^{i-1} \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{nn}^{n-1} x_n = a_{n,n+1}^{n-1} \end{cases}$$

Из треугольной системы последовательно (в обратном порядке) вычисляются значения неизвестных по формуле:

$$x_i = \frac{1}{a_{ii}^{i-1}}(a_{i,n+1}^{i-1} - \sum_{j=i+1}^{n} a_{ij}^{i-1} x_j) \qquad (4.7)$$

для $i=n, n\text{-}1, n\text{-}2, \ldots, 2, 1$ (при $i=n$ сумма в формуле отсутствует).

Пример. Решим СЛАУ 4.2 методом Гаусса. Эта СЛАУ является совместной определенной системой согласно теореме Кронекера-Капелли.

Расширенная матрица имеет вид

$$\left[\begin{array}{ccc|c} 2 & -3 & 1 & -1 \\ 1 & 2 & -6 & -10 \\ 5 & 1 & 1 & 3 \end{array}\right]$$

4.3. Обратная матрица A^{-1}.

Если А является квадратной матрицей, то **обратной** по отношению к А называется матрицы, которая при умножении на А как слева, так и справа дает единичную матрицу:

$$A \cdot A^{-1} = A^{-1} \cdot A = E$$

где $E = \begin{bmatrix} 1 & & & 0 \\ & 1 & & \\ & & \ddots & \\ 0 & & & 1 \end{bmatrix}$ – **единичная матрица.**

Если обратная матрица A^{-1} существует, то матрица А называется обратимой. Для того чтобы квадратная матрица имела обратную, необходимо и достаточно, чтобы матрица А была **невырожденной**, т.е. ее определитель был отличен от нуля $\Delta = \det A \neq 0$.

Для вычисления обратной матрицы в Python используется функция **inv().**

4.4. Понятие об итерационных методах решения СЛАУ

Для решения системы линейных алгебраических уравнений существуют **прямые** и **итерационные**.

Методы, вычисляющие точное решение (в отсутствии ошибок округлений) за конечное число шагов, называются **прямыми**. К прямым методам решения СЛАУ относят: метод последовательных исключений (метод Гаусса), метод Гаусса с частичным выбором главного элемента, метод Гаусса-Жордана, матричный метод, метод Крамера, и т.д.

Итерационные методы (приближенные) используются для решения систем уравнений

$$A \cdot \bar{x} = \bar{b}$$

тогда, когда методы типа Гаусса требует слишком много времени или памяти. В самом деле, для реализации метода Гаусса решения системы n уравнений с n неизвестными требуется

$$N \sim \frac{2}{3} n^3$$

арифметических операций, а при реализации метода Гаусса с выбором главного элемента по всей матрицы число действий и вовсе возрастает до

$$N \sim \frac{3}{2} n^3.$$

В этой связи становится очевидным, что метод Гаусса не является удобным с практической точки зрения для решения больших СЛАУ.

В отличие от прямых методов, итерационные методы обычно не дают точного ответа за конечное число шагов, однако на каждом шаге уменьшают ошибку на какую-то долю. Итерации прекращают, когда ошибка становится меньше допуска, заданного вычислителем (пользователем). Величина финальной ошибки зависит от количества итераций, а также от свойств метода и СЛАУ. Другими словам, итерационные методы дают решение СЛАУ в виде предела последовательности некоторых векторов, построение которых осуществляется посредством единообразного процесса, называемого **итерационным процессом**.

Существенную часть работы по исследованию и повышению эффективности итерационных методов составляет использование связи решаемой СЛАУ с математической или физической постановкой задачи, которая ее породила.

В современной литературе описано большое количество итерационных методов, основанных на разных принципах. Многие методы удобны при использовании вычислительной техники, некоторые еще и достаточно просты. Однако каждый итерационный метод имеет свою ограниченную область применимости, так как, во-первых, итерационный процесс может оказаться расходящимся для данной системы и, во-вторых, сходимость процесса может быть настолько медленной, что практически оказывается невозможным достигнуть удовлетворительной близости к решению. Одним из наиболее предпочтительных современных итерационных методов является **метод сопряженных градиентов.**

Отметим, что хотя задача обращения матрицы эквивалентна решению n СЛАУ с одной и той же матрицей, итерационные методы относительно редко используются для этой цели.

Настоящая лекция посвящена краткому описанию двух простейших итерационных методов – метода простой итерации и метода **Зейделя**.

4.4.1. Общая схема итерационного процесса

Итак, итерационные методы решения системы уравнений

$$A\bar{\mathbf{x}} = \bar{\mathbf{b}}$$

состоят в построении последовательности векторов

$$\bar{\mathbf{x}}^0, \bar{\mathbf{x}}^1, \bar{\mathbf{x}}^2, \ldots, \bar{\mathbf{x}}^k, \bar{\mathbf{x}}^{k+1}$$

по какому-либо алгоритму, такому, что из $\bar{\mathbf{x}}^k$ следует $\bar{\mathbf{x}}^{k+1}$. При этом

$$\begin{cases} \bar{\mathrm{x}}^0 - \textit{задается} \\ \bar{\mathrm{x}}^k \xrightarrow[k\to\infty]{} \bar{\mathbf{x}} \end{cases},$$

где $\bar{\mathbf{x}}$ – точное решение системы.

Вычисления ведутся до тех пор, пока не станет $z_k = \left\|\bar{\mathbf{x}}^{k+1} - \bar{\mathbf{x}}^k\right\| < \varepsilon$, где ε – малое положительное число (заданная точность). С точностью до ε решение принимается равным $\bar{\mathbf{x}}^{k+1}$.

4.4.2. Метод Зейделя и метод простой итерации

Пусть задана система уравнений:

$$\sum_{j=1}^{N} a_{ij} x_j = b_i, \quad i = 1,2,\ldots,N. .$$

Выразим x_i через остальные члены i-го уравнения:

$$x_i = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} x_j - \sum_{j=i+1}^{N} a_{ij} x_j) \quad , \quad i = 1,2,\ldots,N .$$

Полученная запись СЛАУ приводит к двум итерационным процессам:

Метод простой итерации

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{\substack{j=1 \\ j \neq i}}^{N} a_{ij} x_j^k), \quad i = 1,2,\ldots,N. \qquad (4.9)$$

Метод Зейделя

$$x_i^{k+1} = \frac{1}{a_{ii}}(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{N} a_{ij} x_j^k), \quad i = 1,2,\ldots,N. \qquad (4.10)$$

При этом x_i^0 задается ($i = 1, 2, ..., N$), k – номер итерации.

Процесс ведется до выполнения условия $z_k = \left\| \overline{\mathbf{x}}^{k+1} - \overline{\mathbf{x}}^k \right\| < \varepsilon$. Норму z_k вектора $\left(\overline{\mathbf{x}}^{k+1} - \overline{\mathbf{x}}^k \right)$ можно вычислять по формулам, приведенным в пункте 1.5 введения.

Разница методов состоит в том, что в методе простой итерации новые значения x_i учитываются лишь после вычисления их для всех $\boldsymbol{i}$, а в методе Зейделя они учитываются сразу же после вычисления их для каждого $\boldsymbol{i}$.

Достаточный признак сходимости обоих методов состоит в выполнении условия "**диагонального преобладания**":

$$\sum_{\substack{j=1 \\ j\neq i}}^{N} |a_{ij}| \le q|a_{ii}| \quad i=1,2,\ldots,N, \quad \text{где } 0<\text{q}<1. \tag{4.11}$$

4.5. Методы численного интегрирования

4.5.1. Понятие о формулах численного интегрирования

Пусть требуется вычислить определенный интеграл вида

$$S=\int_a^b f(x)dx.$$

Для многих функций $f(x)$ первообразные представляют собой достаточно сложные комбинации элементарных функций, либо вовсе не выражаются через них. В таких случаях использование формулы Ньютона-Лейбница на практике не представляется возможным. Во многих практических случаях достаточно получить значение интеграла с заданной точностью ε. Для вычисления приближенного значения интеграла существуют формулы численного интегрирования. Суть построения формул численного интегрирования состоит в следующем и представлена на рисунке 14.1

Разобьем отрезок $[a,b]$ на N частей. Для простоты изложения положим эти части одинаковой длины $h:\ h=(b-a)/N$.

Пронумеруем точки разбиения, как показано на рисунке 6.1:

$$x_0,\ldots,x_{i-1},\ x_i,\ x_{i+1},\ldots,x_N,$$

при этом

$$x_0=a,\ x_N=b,\ f(x_0)=f(a),\ f(x_N)=f(b).$$

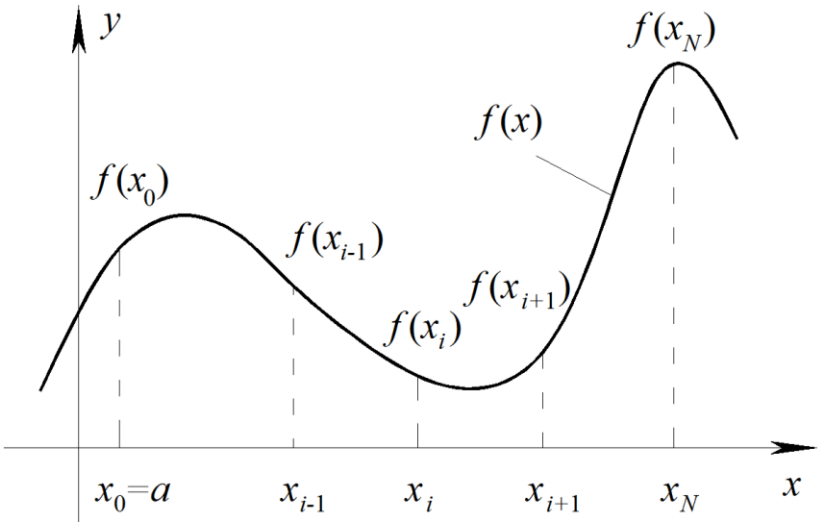

Рис. 4.2. Разбиение функции

Исходный интеграл может быть представлен в виде суммы интегралов по отрезкам:

$$S = \int_a^b f(x)dx = \sum_{i=0}^{N-1} \int_{x_i}^{x_{i+1}} f(x)dx .$$

Интегралы $\int_{x_i}^{x_{i+1}} f(x)dx$ вычисляются по приближенным формулам.

Простейшие формулы для приближенного вычисления интегралов по отрезку называются **квадратурными формулами.** Рассмотрим некоторые из них ниже, а также изучим вопросы их точности. Порядок точности квадратурной формулы определяется степенью полинома (многочлена), для которой эта квадратурная формула точна.

4.5.2 Формула прямоугольников (формула средних)

Заменим на *i*-ом участке интегрируемую функцию постоянной величиной, например, равной ее значению в средней точке (рис. 6.2):

$y(x) = f(x_i^*) = const$, где $x_i^* = \frac{x_{i+1} + x_i}{2} = x_i + \frac{h}{2}$.

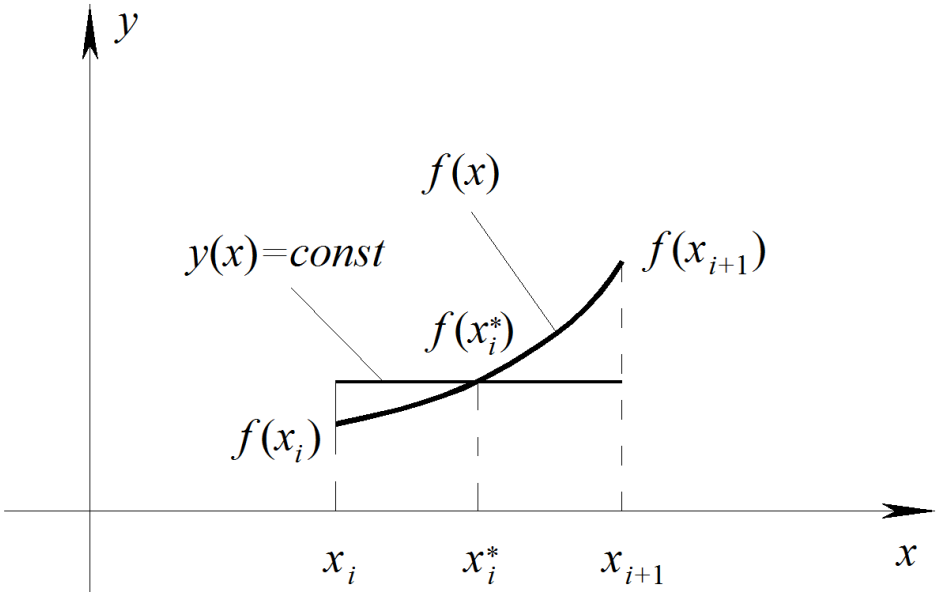

Рис. 4.3. Метод прямоугольников (формула средних)

Тогда интеграл на отрезке заменяется площадью прямоугольника, т.е.

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx hf\left(x_i + \frac{h}{2}\right) \qquad (6.1)$$

и вычисление исходного интеграла сводится к вычислению суммы

$$\int_a^b f(x)dx \approx h\sum_{i=0}^{N-1} f\left(x_i + \frac{h}{2}\right) \qquad (6.2)$$

Замечание. Часто из практических соображений в качестве x_i^* берется x_i (формула «левых» прямоугольников), либо x_{i+1} (формула «правых» прямоугольников). Эти формулы менее точные, но иногда более удобные, например, при численном решении дифференциальных уравнений:

квадратурная формула «левых» прямоугольников:

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx hf(x_i)$$

квадратурная формула «правых» прямоугольников:

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx hf(x_{i+1})$$

Точность вычисления. Как следует из построения квадратурные формулы прямоугольников дают точный результат интегрирования для функций, **постоянных** на *i*-ом участке ($i = 0, 1, \ldots, N-1$). Квадратурная формула «средних» прямоугольников дает точный результат также и для **линейных** на *i*-ом отрезке функций. Это утверждение достаточно проверить для $f(x) = x$:

1.) точное интегрирование

$$\int_{x_i}^{x_{i+1}} x\,dx = \frac{1}{2}x^2\Big|_{x_i}^{x_{i+1}} = \frac{1}{2}\left(x_{i+1}^2 - x_i^2\right) = \frac{1}{2}\underbrace{(x_{i+1} - x_i)}_{h}(x_i + x_{i+1}) = h\frac{x_{i+1} + x_i}{2};$$

2.) интегрирование по формуле «средних» прямоугольников

$$\int_{x_i}^{x_{i+1}} x\,dx = h \cdot x_i^* = h\frac{x_{i+1} + x_i}{2}$$

Результаты точного и численного интегрирования совпадают.

4.5.3 Формула трапеций

Заменяем на *i*-ом участке интегрируемую функцию линейной функцией $y(x)$, принимающей в точках x_i и x_{i+1} значения (рис. 6.3):

$$y(x_i) = f(x_i) = f_i, \quad y(x_{i+1}) = f(x_{i+1}) = f_{i+1}$$

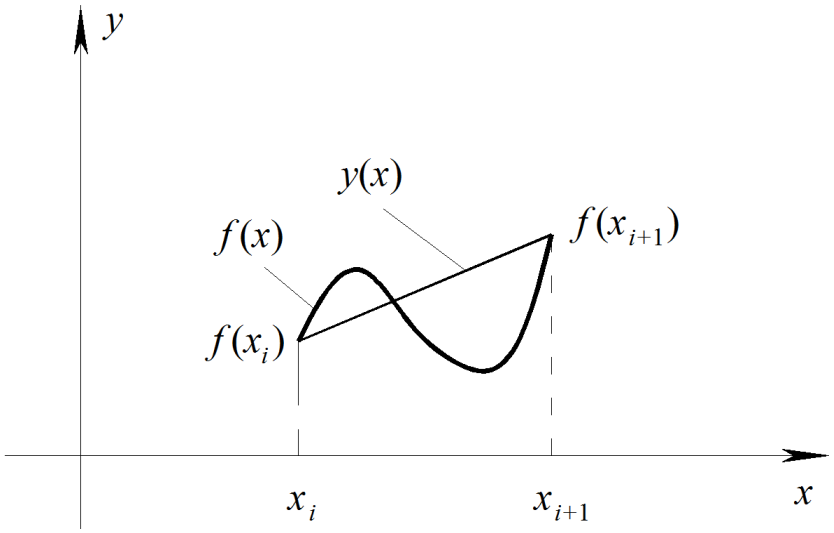

Рис. 4.4. Метод трапеций

Тогда интеграл на отрезке заменятся площадью трапеции (f_i **и** f_{i+1} **–** основания, h – высота)

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx \int_{x_i}^{x_{i+1}} y(x)dx = h\frac{f_i + f_{i+1}}{2} \qquad (6.3)$$

Вычисление исходного интеграла сводится к вычислению суммы

$$S = \int_a^b f(x)dx \approx h\sum_{i=0}^{N-1}\frac{f_i + f_{i+1}}{2},$$

или

$$S = \int_a^b f(x)dx \approx h\left(\frac{f_0 + f_N}{2} + \sum_{i=1}^{N-1} f_i\right) \qquad (6.4)$$

Точность вычисления. Как следует из построения квадратурная формула трапеций дает точный результат интегрирования для функций, **линейных** на *i*-ом участке ($i = 0, 1, \ldots, N-1$).

4.5.4. Формула Симпсона

Разобьем интервал интегрирования на четное число отрезков. Рассмотрим сдвоенный участок $[x_{i-1}, x_{i+1}]$. Построим параболу $y(x) = Ax^2 + Bx + C$, принимающую в точках x_{i-1}, x_i, x_{i+1} значения $f_{i-1} = f(x_{i-1})$, $f_i = f(x_i)$, $f_{i+1} = f(x_{i+1})$ (рис. 6.4).

Такая парабола может быть представлена формулой

$$y(t) = \alpha_0 + \alpha_1 t + \alpha_2 t^2,$$

где $t = \dfrac{x - x_i}{h}$, при этом $t = \begin{cases} -1, & x = x_{i-1} \\ 0, & x = x_i \\ 1, & x = x_{i+1} \end{cases}$, $dt = \dfrac{1}{h}dx$ или $dx = h \cdot dt$

.

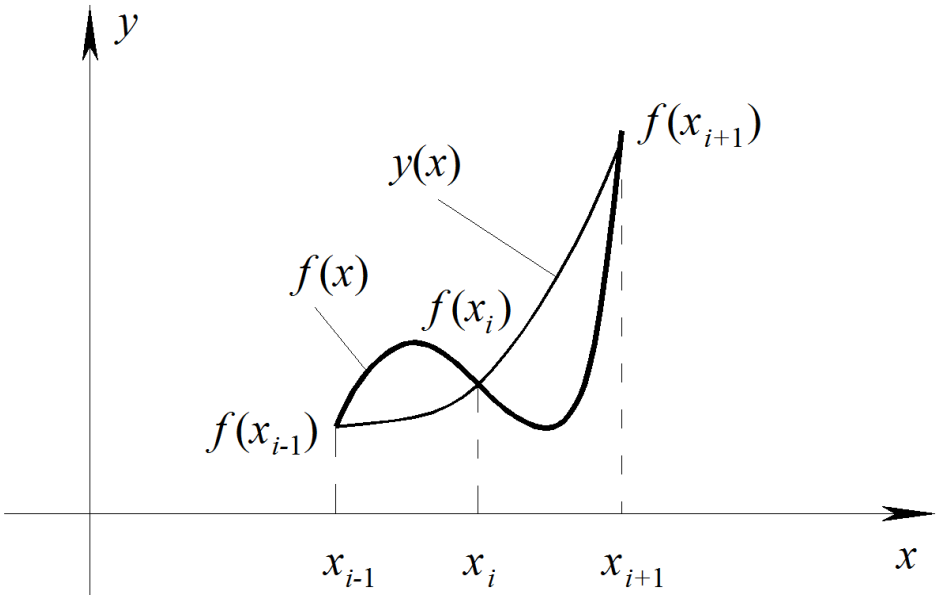

Рис. 4.5. Метод Симпсона

Делая замену переменных, вычислим приближенное значение интеграла

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx \approx \int_{x_{i-1}}^{x_{i+1}} y(x)dx = h\int_{-1}^{1}(\alpha_0 + \alpha_1 t + \alpha_2 t^2)dt = h\left(2\alpha_0 + \frac{2}{3}\alpha_2\right) = \frac{h}{3}(6\alpha_0 + 2\alpha_2).$$

Параметры α_0, α_2 определим из условий

$$\begin{cases} y(-1) = f_{i-1} \\ y(0) = f_i \\ y(1) = f_{i+1} \end{cases} \Rightarrow \begin{cases} \alpha_0 - \alpha_1 + \alpha_2 = f_{i-1} \\ \alpha_0 = f_i \\ \alpha_0 + \alpha_1 + \alpha_2 = f_{i+1} \end{cases} \Rightarrow \begin{cases} \alpha_0 = f_i \\ 2\alpha_2 = f_{i-1} + f_{i+1} - 2\alpha_0 = f_{i-1} - 2f + f_{i+1} \end{cases}$$

В итоге получим квадратурную формулу **Симпсона**

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx \approx \frac{h}{3}(f_{i-1} + 4f_i + f_{i+1}) \qquad (6.5)$$

Общая формула для вычисления приближенного значения интеграла примет вид

$$S = \int_a^b f(x)dx \approx \frac{h}{3} \sum_{\substack{i=1 \\ i-нечетное}}^{N-1} (f_{i-1} + 4f_i + f_{i+1}).$$

Суммирование ведется только по нечетным i. Если перегруппировать члены суммы, получим

$$S = \int_a^b f(x)dx \approx \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + 4f_5 + \ldots + 2f_{N-2} + f_{N-1} + f_N) = \\ = \frac{h}{3}\left(f(a) + f(b) + \sum_{i=1}^{N-1}[3 + (-1)^{i+1}]f_i\right) \qquad (6.6)$$

Точность вычисления. Как следует из построения квадратурные формулы Симпсона дают точный результат интегрирования для функций, имеющих вид **квадратичной параболы** на сдвоенном участке $[x_{i-1}, x_{i+1}]$ (i – нечетное). Заметим, что одинаковая длина сдвоенных участков вовсе не обязательна и использовалась здесь исключительно для упрощения промежуточных выкладок и вида результирующих формул (6.5)–(6.6). В том случае, когда сдвоенные участки имеют одинаковую длину , т.е.

$$x_i = \frac{x_{i-1} + x_{i+1}}{2}$$

– средняя точка сдвоенного участка, формула Симпсона точна для функций, имеющих вид **кубической параболы** на этих участках. Это утверждение достаточно проверить для $f(x)=x^3$:

1.) точное интегрирование

$$\int_{x_{i-1}}^{x_{i+1}} x^3\,dx = \frac{1}{4}x^4\Big|_{x_{i-1}}^{x_{i+1}} = \frac{1}{4}\left(x_{i+1}^4 - x_{i-1}^4\right)$$

2.) интегрирование по формуле Симпсона

$$\int_{x_{i-1}}^{x_{i+1}} x^3 dx = \frac{h}{3}\cdot(x_{i-1}^3 + 4x_i^3 + x_{i+1}^3) = \frac{h}{3}\cdot(x_{i-1}^3 + 4(\frac{x_{i+1}+x_{i-1}}{2})^3 + x_{i+1}^3) =$$

$$= \frac{x_{i+1}-x_{i-1}}{3\cdot 2}\left((x_{i+1}+x_{i-1})(x_{i-1}^2 - x_{i-1}x_{i+1} + x_{i+1}^2) + \frac{1}{2}(x_{i-1}-x_{i+1})^3\right) =$$

$$= \frac{(x_{i+1}-x_{i-1})(x_{i+1}+x_{i-1})}{3\cdot 2}\left(x_{i-1}^2 - x_{i-1}x_{i+1} + x_{i+1}^2 + \frac{1}{2}(x_{i-1}^2 + 2x_{i-1}x_{i+1} + x_{i+1}^2)\right) =$$

$$= \frac{(x_{i+1}^2 - x_{i-1}^2)}{3\cdot 2}\cdot\frac{3(x_{i-1}^2 + x_{i+1}^2)}{2} = \frac{1}{4}\left(x_{i+1}^4 - x_{i-1}^4\right)$$

Как видно, результаты точного и численного интегрирования совпадают.

4.5.5. Точность квадратурных формул

Оценка точности квадратурных формул основывается на использовании разложения подынтегральной функции в ряд. Приведем такую оценку для формулы трапеций. Запишем представление функции $f(x)$ рядом **Тейлора** относительно центра отрезка $\tilde{x}$: $\tilde{x} = \frac{x_{i+1}+x_i}{2} = x_i + \frac{h}{2}$:

$$f(x) = f(\tilde{x}) + (x-\tilde{x})f'(\tilde{x}) + \frac{(x-\tilde{x})^2}{2}f''(\tilde{x}) + \frac{(x-\tilde{x})^3}{6}f'''(\tilde{x}) + \ldots .$$

Тогда

$$\int_{x_i}^{x_{i+1}} f(x)dx \approx hf(\tilde{x}) + \frac{h^3}{24}f''(\tilde{x}) + \ldots$$

(интегралы при членах с нечетными производными равны 0) .

Используя разложение в ряд Тейлора относительно средней точки для $f(x_i)$ и $f(x_{i+1})$, получим

$$h\frac{f(x_i)+f(x_{i+1})}{2}=\frac{h}{2}\left(f(\tilde{x}-\frac{h}{2})+f(\tilde{x}+\frac{h}{2})\right)=hf(\tilde{x})+\frac{h^3}{8}f''(\tilde{x})+\ldots$$

.

Погрешность интегрирования оценивается разностью

$$\int_{x_i}^{x_{i+1}} f(x)dx-h\frac{f(x_i)+f(x_{i+1})}{2}\approx\left(\frac{h^3}{24}-\frac{h^3}{8}\right)f''(\tilde{x})+\ldots=-\frac{h^3}{12}f''(\tilde{x})+O(h^4)\approx-\frac{h^3}{12}f''(\tilde{x})$$

Сумма таких погрешностей на всем интервале интегрирования может быть оценена формулой

$$\Delta_{mp}\approx(N-1)\frac{h^3}{12}\max\left|f''(x)\right|$$

.

Поскольку $N-1=(b-a)/h$, окончательная оценка погрешности имеет вид:

$$\Delta_{mp}=(b-a)\frac{h^2}{12}\max\left|f''(x)\right| \qquad (6.7)$$

Иными словами, погрешность пропорциональна h^2(второй порядок точности).

Следует отметить, что оценка справедлива лишь при существовании $f''(x)$

Для формулы прямоугольников оценка получается аналогичным способом и сводится к виду

$$\Delta_{пр} \approx (b-a)\frac{h^2}{12}\max\left|f''(x)\right| \qquad (6.8)$$

т.е. имеет ту же порядковую точность, что и формула трапеций.

Погрешность вычисления интеграла по методу Симпсона, оцениваемая таким же образом, как и в случае метода трапеций, приводит к формуле

$$\Delta_{Симпс} \approx (b-a)\frac{h^4}{180}\max\left|f^{(4)}(x)\right| \qquad (6.9)$$

т.е. имеет 4-й порядок точности. Если четвертая производная существует и невелика, то формула Симпсона очень точная, что выгодно при ручном счете. В общем случае, например для кусочно-линейных функций, формула Симпсона не всегда обеспечивает высокую точность.

4.6. Методы решения нелинейных уравнений

4.6.1. Уравнения с одним неизвестным

Пусть на участке (a,b) задана непрерывная функция $f(x)$ (см. рис. 7.1).

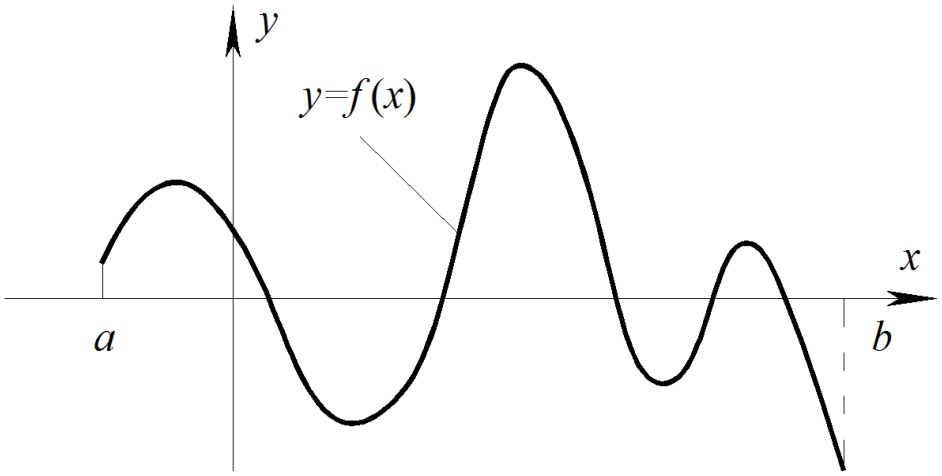

Рис.4.6. График уравнения $f(x)$

Требуется найти корни уравнения

$$f(x) = 0 \qquad (7.1)$$

Решение такой задачи, как правило, распадается на два этапа:

1) определение интервалов, в которых находится только один корень (если корень существует);

2) вычисление этого корня с заданной точностью.

Для решения задачи на 1-ом этапе существуют различные аналитические подходы, связанные с определенными типами уравнений (многочлены, тригонометрические уравнения и т.д.). Однако, наиболее эффективным численным подходом является **метод перебора**. Он реализуется следующим алгоритмом. Задается точность, определяемая шагом h: $h = \frac{b-a}{N}$. Затем последовательно вычисляются значения функции f:

$$f(x_0) = f(a),\ f(x_1) = f(a+h),\ \ldots,\ f(x_i) = f(a+ih),\ \ldots,\ f(x_N) = f(b)$$

В интервалах, на концах которых функция меняет знак:

$$f(x_i) \cdot f(x_{i+1}) < 0$$

находится корень уравнения. Если $h < \varepsilon$, где ε – заданная точность, то величина

$$\tilde{x}_i = \frac{x_i + x_{i+1}}{2}$$

является приближенным значением корня. Таким способом можно найти все значения корней одновременно. Недостатком метода является относительно большое количество вычислений. Число вычислений значений функции $f(x)$ имеет оценку

$$N \geq \frac{b-a}{\varepsilon}.$$

Точность метода может быть существенно увеличена, если этим же методом воспользоваться дополнительно на каждом отрезке, на котором было получено приближенное значение корня, но при более мелком разбиении (при меньшей величине шага). При возможном наличии нескольких корней метод перебора является достаточно эффективным при

решении практических задач. Для других, более эффективных методов, существенным является допущение, что на отрезке (a,b) существует только один корень.

Замечания.

1. Достаточным признаком того, что на отрезке (a,b) существует корень, является выполнение условия $f(a)\cdot f(b)<0$ (т.е. функция меняет знак на рассматриваемом отрезке).

2. Достаточным признаком единственности корня на отрезке (a,b) является постоянство знака производной $f'(x)$ на этом отрезке (т.е. монотонность функции *f*).

4.6.2. Метод половинного деления

Метод половинного деления (дихотомии) заключается в следующем. Отрезок, на котором существует корень уравнения $\tilde{x}$, делится пополам (рис. 7.2). Если знак функции в точке деления отличен от знака функции в начальной точке, то корень находится в первой половине отрезка и вторая половина отбрасывается. Если знаки совпадают, то корень находится во второй половине и первая половина отбрасывается.

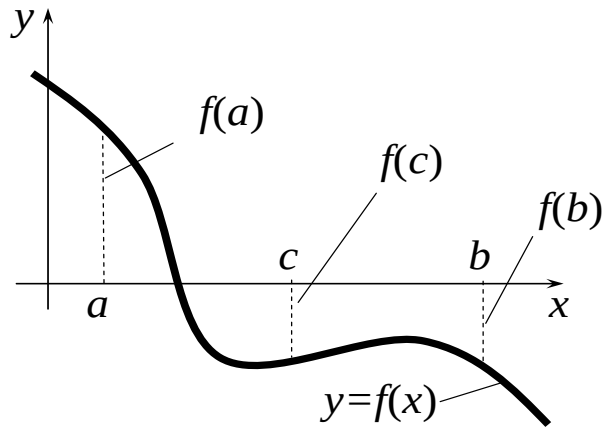

Рис.4.7. Метод половинного деления

Затем аналогичные действия (шаг приближенного решения) повторяются с оставшимся уменьшенным вдвое отрезком. Это происходит до тех пор, пока длина отрезка, оставшегося после *N*-го шага решения, не станет меньше ε. Тогда любая точка этого отрезка (например, его середина) может быть принята в качестве приближенного решения уравнения с заданной точностью ε.

Алгоритм метода половинного деления кратко представлен ниже.

$a_0 = a$, $b_0 = b$ начальные значения;

0–ой шаг:

$$c_0 = \frac{a_0 + b_0}{2}, \quad t_0 = f(a_0) \cdot f(c_0), \quad \begin{cases} a_1 = a_0, b_1 = c_0 \ \textit{если}\ t_0 < 0 \\ a_1 = c_0, b_1 = b_0 \ \textit{если}\ t_0 > 0 \end{cases};$$

k–ый шаг:

$$c_k = \frac{a_k + b_k}{2}, \quad t_k = f(a_k) \cdot f(c_k), \quad \begin{cases} a_{k+1} = a_k, b_{k+1} = c_k \ \textit{если}\ t_k < 0 \\ a_{k+1} = c_k, b_{k+1} = b_k \ \textit{если}\ t_k > 0 \end{cases};$$

$$k = 0, 1, \ldots \qquad (7.2)$$

Окончание вычислений происходит при достижении заданной точности (условие окончания счета):

$$b_k - a_k < \varepsilon \qquad (7.3)$$

тогда приближенное значение корня определяется в виде

$$\tilde{x} \approx c_k \qquad (7.4)$$

Оценка количества шагов *n*, необходимого для достижения заданной точности:

Из выполнения условия окончания счета (7.3)

$$b_n - a_n = \frac{b_0 - a_0}{2^n} < \varepsilon$$

следует, что

$$n > \frac{\ln[(b-a)/\varepsilon]}{\ln 2} \qquad (7.5)$$

4.6.3. Метод Ньютона

Во многих случаях наиболее эффективным методом вычисления корня нелинейного уравнения является метод **Ньютона**. Он представляется частным случаем обобщенного метода итераций, если принять

$$\psi(x) = 1/f'(x) \,. \qquad (7.17)$$

Тогда

$$\varphi(x) = x - f(x)/f'(x) \qquad (7.18)$$

алгоритм пересчета по методу Ньютона имеет вид

$$x_{k+1} = x_k - f(x_k)/f'(x_k) \,, \quad k = 0, 1, \ldots \qquad (7.19)$$

В этом случае

$$|\varphi'(x)| = \left|1 - \frac{f'(x)f'(x) - f(x)f''(x)}{(f'(x))^2}\right| = \left|\frac{f(x)f''(x)}{(f'(x))^2}\right| .$$

Таким образом, что достаточное условие сходимости имеет вид

$$\left|\frac{f(x)f''(x)}{(f'(x))^2}\right| < 1 \,, \quad \text{или} \quad |f(x)f''(x)| < (f'(x))^2 \qquad (7.20)$$

Замечание. При достаточной близости начального приближения x_0 к корню уравнения метод Ньютона всегда сходится.

Геометрическая интерпретация. Метод Ньютона известен также, как **метод касательных.** Рис. 7.3 дает геометрическую интерпретацию итераций по методу Ньютона при заданном начальном приближении корня x_0.

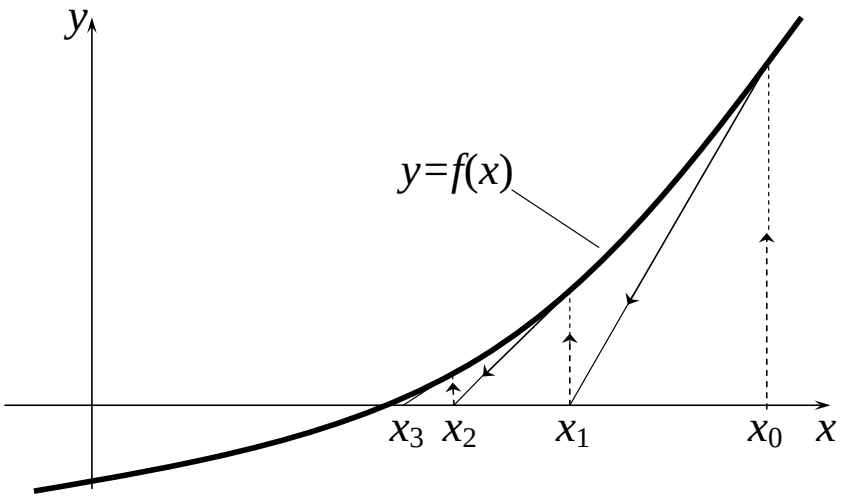

Рис. 4.8. Метод касательных

Выведем формулу для вычисления x_{k+1} исходя из геометрических соображений. Из рис. 7.4 следует, что $x_k - x_{k+1} = f(x_k)/\mathrm{tg}\,\alpha_k$, где $\mathrm{tg}\,\alpha_k = f'(x_k)$. Отсюда получается основная формула метода Ньютона (см. формулу (7.19)):

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}, \quad k = 0, 1, \ldots .$$

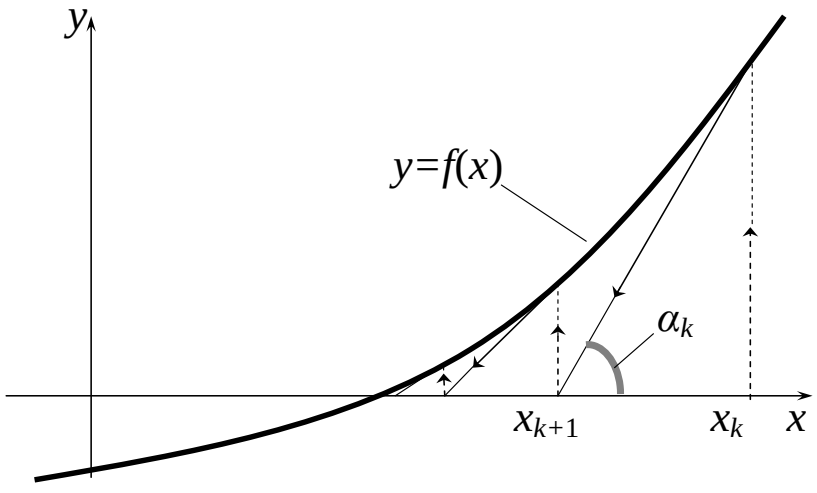

Рис. 4.9. Решение методом касательных

Заметим, что в практических задачах критерием окончания счета часто является условие

$$|f(x_k)|<\varepsilon. \qquad (7.21)$$

Величина

$$v_k = f(x_k) \qquad (7.22)$$

называется **невязкой**. Она свидетельствует, насколько точно удовлетворяется исходное уравнение.