# Предисловие

Данное учебное пособие состоит из двух независимых частей: «Программирование на языке Python. Основы структурного программирования» и «Программирование на языке Python. Сложные типы и конструкции».

Образовательный стандарт по укрупненному направлению подготовки 09.03.00 «Информатика и вычислительная техника» предусматривает, что в результате изучения общепрофессиональных дисциплин, в том числе дисциплины «Алгоритмические языки и программирование», обучаемый должен:

**знать:** основные языки программирования и работы с базами данных, операционные системы и оболочки, современные программные среды разработки информационных систем и технологий;

**уметь:** применять языки программирования и работы с базами данных, современные программные среды разработки информационных систем и технологий для автоматизации бизнес-процессов, решения прикладных задач различных классов, ведения баз данных и информационных хранилищ;

**владеть:** навыками программирования, отладки и тестирования прототипов программно-технических комплексов задач.

В первой части учебного пособия рассмотрены следующие вопросы.

1. Этапы решения задачи на ЭВМ.
2. Структурное программирование.
3. Основные правила работы в среде PyCharm.
4. Программы с линейной структурой.
5. Оператор условного перехода.
6. Многоальтернативное ветвление.
7. Простейшие циклические программы. Оператор цикла с предусловием.
8. Оператор цикла с заголовком. Вычисление конечных сумм и произведений.
9. Алгоритмы численного интегрирования.

10. Итерационный цикл. Вычисление суммы бесконечного ряда.

11. Итерационный цикл. Численные алгоритмы уточнения корней трансцендентных алгебраических уравнений.

12. Проектирование алгоритмов и программ со структурой вложенных циклов.

13. Отладка программ в среде PyCharm.

14. Работа с коллекциями.

Первая часть учебного пособия предназначения для изучения и освоения основных правил работы с простейшими типами данных в языке программирования Питон. Освоение конструкций языка ориентировано на решение основных типовых алгоритмов, используемых в структурном программировании. Приведенный материал подготавливает обучаемого к более сложным типам и конструкциям, используемым в языке Питон. Во второй части учебного пособия будут рассмотрены следующие темы.

1. Простые списки и кортежи. Обработка одномерных массивов.

2. Вложенные списки. Обработка двумерных массивов (матриц).

3. Обработка текстовой информации.

4. Словари.

5. Организация данных в множестве.

6. Ввод и вывод данных.

7. Подпрограммы и функции.

8. Рекурсивные алгоритмы.

9. Модули.

10. Работа с текстовыми файлами.

# 1 Этапы решения задачи на ЭВМ

Процесс решения задачи на компьютере условно можно представить в виде последовательности нескольких взаимосвязанных этапов совместной деятельности человека (программиста) и ЭВМ. Программист реализует, в первую очередь, творческие этапы, связанные с постановкой, алгоритмизацией, программированием задачи, анализом и трактовкой полученных результатов. Компьютер, в свою очередь, осуществляет обработку информации в соответствии с разработанным алгоритмом.

Основными этапами решения задачи на ЭВМ можно считать следующие.

1. Постановка задачи (разработка технического задания).
2. Формализация задачи (построение математической или информационной модели).
3. Выбор (или разработка) метода решения.
4. Разработка алгоритма (алгоритмизация задачи).
5. Выбор языка программирования и написание программного кода (программирование).
6. Отладка и тестирование программы.
7. Исполнение отлаженной программы.
8. Анализ и представление полученных результатов.
9. Документирование программы.

Последовательное выполнение перечисленных этапов составляет полный цикл разработки, отладки и выполнения программы. Рассмотрим подробнее наиболее важные из них.

**Постановка задачи.** Постановка задачи представляет собой *содержательный анализ* существа задачи, изучение общих свойств рассматриваемого физического явления или объекта, определение конечной цели и результатов работы, анализ известной информации и определение исходных данных, *выработку общего подхода* к исследуемой проблеме: выяснению, существует ли решение поставленной задачи и единственно ли оно, и, наконец, анализ возможностей используемой вычислительной среды. На этом этапе тре-

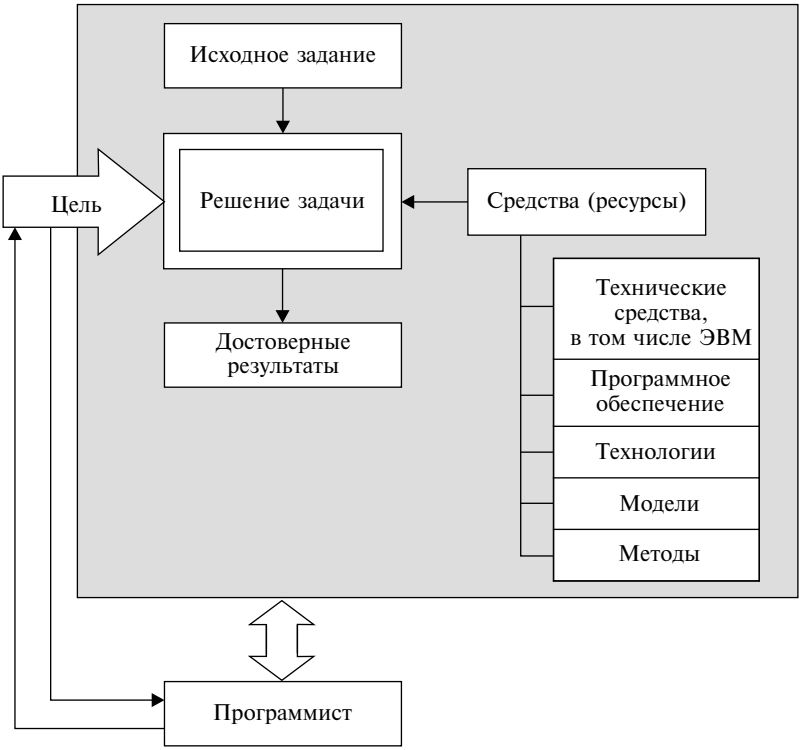


**Рис. 1.1.** Общая схема решения задачи на ЭВМ

буется знание исследуемой предметной области и глубокое понимание существа поставленной задачи.

Первостепенное значение при постановке задачи должно быть уделено конечной (глобальной) цели, решаемой программистом. Что дает решение задачи? Зачем нужно решение задачи? Все остальные этапы направлены (ориентированы) на достижение глобальной цели, хотя на каждом из последующих этапов определяются и решаются отдельные подцели и, соответственно, отдельные подзадачи (рис. 1.1). Основная цель, как правило, формулируется следующим образом:

**получение надежного (достоверного) результата для некоторого набора исходных данных.**

На первом этапе рассматриваются и анализируются основные подходы решения задачи, определяются существующие аналоги, изучаются общие свойства рассматриваемого физического явления. В обязательном порядке проводится анализ возможностей конкретного компьютера и используемой системы программирования.

Правильное формулирование задачи, глобальной цели и отдельных подцелей определяет до 50 % успешного решения задачи (в том числе и с помощью ЭВМ). Поэтому требуйте от заказчика четкой и конкретной постановки задачи с указанием доступных ресурсов для ее решения.

**Формализация задачи.** Как правило, формализация задачи сводится к построению *математической модели* рассматриваемого явления или процесса, когда в результате предыдущего анализа существа решаемой задачи устанавливается ее принадлежность к одному из известных классов задач и выбирается соответствующий математический аппарат, определяется формат исходных данных и результатов работы, вводится определенная система условных обозначений. При этом нужно уметь сформулировать на языке математики конкретные задачи физики, механики, экономики, технологии и т. д. Для успешного преодоления этого этапа требуются не только знание предметной области, но и определенное знание вычислительной математики, тех ее разделов и методов, которые могут быть использованы при решении данной задачи.

Как известно, любая система может быть представлена в виде «черного ящика» (рис. 1.2). При использовании системы $S$ требуется определить зависимость выходных характеристик $Y = (y_1, y_2, \ldots, y_m)$ от входных параметров $X = (x_1, x_2, \ldots, x_n)$ с учетом внешних воздействий $V = (v_1, v_2, \ldots, v_k)$.

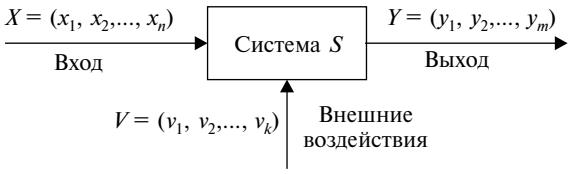


**Рис. 1.2.** Представление системы в виде «черного ящика»

Для исследования характеристик системы $S$ используется/разрабатывается математическая модель. *Математическая модель* — это система математических соотношений, учитывающих наиболее существенные взаимосвязи в изучаемом классе объектов/явлений и их свойства в совокупности с определенной областью допустимых значений исходных данных и областью допустимых значений искомых результатов. Известно, что 20 % свойств системы определяют 80 % характеристик системы (рис. 1.3). Используя этот закон, создается математическая модель $M$ исследуемой системы $S$. Сама модель также является системой $S'$, что позволяет реализовать за-

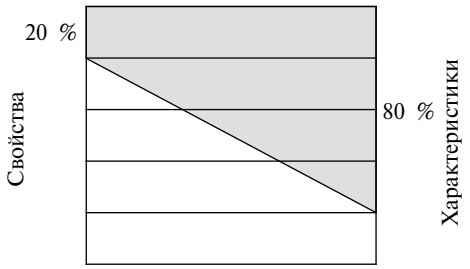


**Рис. 1.3.** Взаимосвязь свойств и характеристик системы

дачу исследования: оценить характеристики исследуемой системы $S$ с помощью математической модели $M = S'$.

Построение любой модели представляет собой грамотное упрощение поставленной задачи и требует знания того, какие факторы и параметры наиболее существенны для изучаемой задачи и должны быть учтены при построении модели. При этом следует учитывать два противоречивых требования: требуемую простоту (математических зависимостей, вычислительных методов и расчетных схем, способов получения или измерения требуемых исходных данных, возможных способов представления результатов работы) и адекватность модели (полноту и точность) исследуемому явлению или процессу.

Если известных средств недостаточно, тогда необходимо попытаться разработать новых подход, новые методы исследования.

**Выбор метода решения задачи.** После математической постановки задачи мы отвлекаемся от ее предметной сущности и далее оперируем с абстрактными математическими понятиями, величинами, формулами.

В общем случае метод решения представляет собой ряд формул или математических соотношений и правила, определяющие связи между формулами. Метод решения формулируется набором следующих математических конструкций:

- уравнения;
- функционалы;
- равенства;
- системы уравнений (алгебраических, дифференциальных, интегральных);
- логические правила и др.

Следующий шаг — анализ и оценка известных методов решения поставленной *математической задачи* и выбор наиболее приемлемого. При выборе метода надо учитывать: во-первых, характеристики самого метода, сложность формул и соотношений, связанных

с этим численным методом; во-вторых, необходимую точность вычислений.

На выбор метода большое влияние оказывают вкусы и знания самого пользователя. А между тем этот этап — важнейший в процессе решения задачи, так как оказывает существенное влияние на трудоемкость всей последующей процедуры разработки и реализации алгоритма и программы, их качество, на точность получаемых результатов.

При решении задачи на ЭВМ необходимо помнить, что любой получаемый результат является приближенным! Если известен алгоритм точного решения, то возможны ошибки, связанные с ограниченной точностью представления вещественных чисел в памяти ЭВМ. При вычислениях с заданной степенью точности (при использовании численных методов) возникает дополнительная погрешность, определяемая выбранным методом и которую, по возможности, оценивают на данном этапе.

**Разработка алгоритма.** *Процесс алгоритмизации* — это отдельный этап, и мы придаем ему особую значимость. Именно на этом этапе осуществляется процесс разработки структуры алгоритма, реализующего выбранный метод решения, и производится запись «придуманной» структуры на языке, «понятном» самому разработчику.

*Алгоритм* — последовательность точных и понятных предписаний о содержании и последовательности выполнения конечного числа действий, необходимых для решения любой задачи данного типа/класса (получение для конкретных исходных данных достоверного результата), что отражено на рис. 1.1. Алгоритм составляется в расчете на исполнителя, коим является компьютер, чьё поведение основано на реализации алгоритма. В свою очередь использование персонального компьютера (ПК) оказывает воздействие на развитие теории алгоритмов, одной из областей дискретной математики. Следует придерживаться правила: алгоритм создается безотносительно к языку программирования (не рекомендуется в описании алгоритма использовать нотации конкретного языка программирования).

В процессе составления/проектирования алгоритма необходимо обеспечить, чтобы он обладал рядом свойств.

*Однозначность алгоритма* — единственность толкования исполнителем правил выполнения действий и порядка их выполнения.

*Конечность алгоритма* — обязательность завершения каждого из действий, составляющих алгоритм, и завершимость алгоритма в целом.

*Результативность алгоритма* — выполнение предписанных действий должно завершиться получением достоверных результатов.

*Массовость* — возможность применения разработанного алгоритма для решения класса задач, отвечающих общей постановке задачи.

*Правильность алгоритма* — способность алгоритма приводить к правильным и достоверным результатам решения поставленной задачи.

Можно использовать различные способы описания алгоритмов, отличающие по простоте и наглядности. В практике программирования наибольшее распространение получили словесная запись алгоритмов, схемы алгоритмов (блок-схемы) и структурограммы (диаграммы Насси–Шнейдермана).

При использовании схем алгоритмов реализуется наглядное двухмерное изображение с применением специальных графических символов блоков (поэтому такой способ описания называется блок-схемой алгоритма). Схемы алгоритмов исполняются с учетом Единой системы программной документации:

*ГОСТ 19.701-90 (ИСО 5807-85) Единая система программной документации (ЕСПД). Схемы алгоритмов, программ, данных и систем. Обозначения условные и правила выполнения.*

Данный ГОСТ пришел на смену ранее используемому:

*ГОСТ 19.003-80 Единая система программной документации (ЕСПД). Схемы алгоритмов и программ. Обозначения условные графические.*

Построение алгоритма заключается в разложении вычислительного процесса решения задачи на возможные составные части, установлении порядка их следования, описании содержания каждой такой части в той или иной форме и последующей проверке, которая должна показать, обеспечивается ли реализация выбранного метода. Разработать структуру алгоритма порой непросто и получить «правильный», а тем более «хороший» алгоритм в большинстве случаев удается только после *многих шагов анализа и коррекции* его структуры.

Как правило, в процессе разработки алгоритм проходит несколько этапов детализации. Первоначально составляется укрупненная схема алгоритма, в которой отражаются наиболее важные и существенные связи между исследуемыми процессами (или частями процесса). На последующих этапах раскрываются (детализируются) выделенные на предыдущих этапах части вычислительного процес-

са, имеющие некоторое самостоятельное значение. Кроме того, на каждом этапе детализации выполняется многократная проверка и исправление (отработка) схемы алгоритма. Подобный подход позволяет во многом избежать возможных ошибочных решений.

Ориентируясь на крупноблочную структуру алгоритма, можно быстрее и проще разработать несколько различных его вариантов, провести их анализ, оценку и выбрать наилучший (оптимальный).

Эффект *поэтапной детализации алгоритма* во многом зависит от того, как осуществляется его структуризация: разделение общего алгоритмического процесса на составные части, что должно определяться не произволом пользователя (программиста), а внутренней логикой самого процесса. Каждый элемент крупноблочной схемы алгоритма должен быть максимально самостоятельным и логически завершенным в такой степени, чтобы дальнейшую его детализацию можно было выполнять независимо от детализации остальных элементов. Это упрощает процесс проектирования алгоритма и позволяет осуществлять его разработку по частям одновременно нескольким людям.

Разработка алгоритмов является в значительной степени творческим, эвристическим процессом, как правило, требует большой эрудиции, изобретательности, нестандартных и нетрадиционных подходов к решению задачи. Но даже этот творческий процесс включает с себя чисто технологические вопросы, и, придерживаясь определенной дисциплины (технологии) в разработке алгоритма, можно облегчить себе и сам процесс его разработки, и дальнейшую работу с этим алгоритмом.

**Разработка программы.** *Процесс программирования* — это запись разработанного алгоритма на специальном языке (*языке программирования*) — представление алгоритма на языке, «понятном» исполнителю (вычислительной машине), т. е. в форме, допускающей ввод в машину и последующий перевод на машинный язык (в коды машины).

*Язык программирования* — это строго формализованный язык для описания процесса решения задачи на ЭВМ, представляет собой совокупность ограниченного набора символов и строгих правил их использования. Составленная программа вводится в ЭВМ и затем автоматически переводится на язык машины с помощью специальных программных средств, позволяющих автоматизировать этот процесс. Перевод — *трансляция* исходного текста программы выполняется служебной программой — *транслятором*, который осуществляет синтаксический контроль текста программы и последующий его перевод.

Трансляторы могут быть компилирующего типа — компиляторы и интерпретирующего типа — интерпретаторы.

*Компилятор* анализирует и преобразует исходный текст в так называемый объектный код (промежуточное состояние программы в относительных адресах и с неразрешенными внешними ссылками) с использованием всей логической структуры программы. Затем программа, представленная в объектном коде, обрабатывается служебной программой — *компоновщиком*, который осуществляет подключение внешних подпрограмм/разрешение внешних ссылок и выполняет дальнейший перевод программы пользователя в коды машины (в абсолютный/загрузочный код — с абсолютной адресацией машинных команд). Программа в абсолютном коде может быть сохранена (в *.exe*-файле) и выполнена на компьютере. Загрузка программы из *.exe*-файла в память машины для ее выполнения осуществляется служебной программой *загрузчиком*.

*Интерпретатор* сразу производит анализ, перевод (в машинный код) и выполнение программы строка за строкой. Поэтому интерпретатор должен находиться в оперативной памяти в течение всего времени выполнения программы пользователя. При интерпретации скорость выполнения программы существенно снижается, однако весь процесс прохождения программы на ЭВМ упрощается и имеется возможность организации диалогового (*интерактивного*) режима отладки и выполнения программы.

Выбор языка программирования определяется многими факторами: типом решаемой задачи, располагаемыми вычислительными средствами, вкусами и знаниями заказчика и разработчика.

В данной книге рассмотрены основы программирования на языке Питон (английское название — Python). Начало разработки языка Питон относится к концу 1980-х годов. Основоположником разработки является сотрудник голландского института Гвидо Ван Россум. Название языка произошло от популярного британского комедийного телешоу 1970-х «Летающий цирк Монти Пайтона», которое любил смотреть Гвидо.

Язык Питон создан под влиянием множества языков программирования:

*ABC, Паскаль* — отступы для группировки данных операторов, высокоуровневые структуры данных, блочная структура;

*C*, *C++* — синтаксические конструкторы, которые нашли широкое применение и признание у программистов;

*Smalltalk* — объектно-ориентированное программирование;

*ЛИСП* — отдельные черты функционального программирования;

*Фортран* — срезы массивов, комплексная арифметика;
*Java* — модули.

Питон является высокоуровневым языком программирования общего назначения, ориентированным на повышение производительности программиста и читаемость программного кода. Отличительной особенностью Питона следует отметить возможность использования стандартных библиотек, которые включают большой объем полезных функций. Питон поддерживает динамическую типизацию (когда тип переменной определяется только во время исполнения программы). Основной недостаток — медлительность при выполнении алгоритма (например, по сравнению с C++ и Java). Широкое распространение языка, его современных диалектов свидетельствует о его практической ценности в различных сферах применения и прежде всего в сфере начального обучения программированию и формирования профессиональных навыков будущего специалиста в области IT-технологий.

**Отладка и тестирование программы.** *Отладка программы* — это процесс поиска и устранения ошибок. Часть ошибок формального характера, связанных с нарушением правил записи конструкций языка или отсутствием необходимых описаний, обнаруживает транслятор, производя синтаксический анализ текста программы. Транслятор выявляет ошибки и сообщает о них, указывая их тип и место в программе. Такие ошибки называются *ошибками времени трансляции* или *синтаксическими ошибками*.

Ошибочные ситуации могут возникнуть и при выполнении программы, например деление на нуль или извлечение корня квадратного из отрицательного числа. Такие ошибки называются *ошибками времени выполнения*.

Программа, не имеющая ошибок трансляции и выполнения, может и не дать верных результатов из-за логических ошибок в алгоритме, т. е. *алгоритмических, или семантических, ошибок*. Ошибки подобного рода могут возникнуть на любом этапе разработки программы: постановки задачи, разработки математической модели или алгоритма.

Необходим действенный контроль над процессом вычислений, позволяющий предотвращать или своевременно обнаруживать ошибки подобного рода. Для этого используются как качественный анализ задачи, основанный на различного рода интуитивных соображениях и правдоподобных рассуждениях, так и контрольный просчет, тестирование программы. Практические навыки отладки программы на языке Питон с использованием средств среды программирования PyCharm рассмотрены в разделе 13.

*Тестирование программы* — это выполнение программы на наборах исходных данных (*тестах*), для которых известны результаты, полученные другим методом. Система тестов подбирается таким образом, чтобы:

- проверить все возможные режимы работы программы;
- по возможности локализовать ошибку.

При тестировании программы простой и действенный метод дополнительного контроля над ходом ее выполнения — получение *контрольных точек*, т. е. контрольный вывод промежуточных результатов.

Для проверки правильности работы программы иногда полезно также выполнить проверку выполнения условий задачи (например, для алгебраического уравнения найденные корни подставляются в исходное уравнение и проверяются расхождения левой и правой частей).

Для сложных по структуре программ плохо спланированные процессы алгоритмизации и программирования приводят к ошибкам, которые могут быть обнаружены лишь после многократных проверок, и процесс отладки и тестирования может потребовать значительно больше машинного времени, чем, собственно, само решение задачи на ЭВМ.

**Документирование программы.** Разработка программной документации является важным аспектом процесса решения задач на ЭВМ. Одним из показателей квалификации программиста принято считать умение и навыки документирования программных продуктов. Главная цель документации состоит в том, чтобы помочь стороннему пользователю понять программу. Наличие документации помогает сэкономить значительное время и позволяет избежать многих недоразумений. В общем случае текст готовой программы снабжается внешней и программной документацией.

*Внешняя документация* представляет собой сведения о программе, не содержащиеся в самой программе. В зависимости от размеров и сложности программы внешняя документация может принимать различные формы:

- схемы алгоритмов;
- инструкции для пользователей;
- образцы входных и выходных данных;
- полное описание процесса построения программы;
- словесное описание алгоритма;
- ссылки на источники информации и др.

*Программная документация* реализуется в основном с помощью соответствующих комментариев (в начале программы в виде

Таблица 1.1

Виды программных документов на этапе проектирования

| Вид программного документа | Содержание программного документа |
|---|---|
| Спецификация | Состав программы и документации на нее |
| Ведомость держателей подлинников | Перечень предприятий, на которых хранят подлинники программных документов |
| Текст программы | Запись программы с необходимыми комментариями |
| Описание программы | Сведения о логической структуре и функционировании программы |
| Программа и методика испытаний | Требования, подлежащие проверке при испытании программы, а также порядок и методы их контроля |
| Техническое задание | Назначение и область применения программы, технические, технико-экономические и специальные требования, предъявляемые к программе, необходимые стадии и сроки разработки, виды испытаний |
| Пояснительная записка | Схема алгоритма, общее описание алгоритма и/или функционирования программы, а также обоснование принятых технических и технико-экономических решений |
| Эксплуатационные документы | Сведения для обеспечения функционирования и эксплуатации программы |

заголовка и вводных комментариев, и поясняющих — внутри текста программы), а также рационального выбора имен, применения стандартных методов структурирования программ.

Виды программ и программных документов для ЭВМ независимого от их назначения установлены ГОСТ 19.101-77 и ГОСТ 19781-90. В ГОСТах определены следующие элементы.

*Программа* — это данные, предназначенные для управления конкретными компонентами системы обработки информации в целях реализации определенного алгоритма. Программа может быть реализована в виде компоненты или комплекса.

*Компонент* — программа, рассматриваемая как единое целое, выполняющая законченную функцию и применяемая самостоятельно или в составе комплекса.

*Комплекс* — программа, состоящая из двух или более компонент и/или комплексов, выполняющих взаимосвязанные функции, и применяемая самостоятельно или в составе другого комплекса.

Совокупность программ системы обработки информации и программных документов, необходимых для эксплуатации этих программ, называется *программным обеспечением* (ПО).

Документирование программ позволяет решить следующие задачи:

- сокращает количество ошибок и облегчает отладку;