

Павел Добряк

# Python

*12 уроков для начинающих*

Санкт-Петербург

«БХВ-Петербург»

2023

УДК 004.43  
ББК 32.973.26-018.1  
Д57

**Добряк П. В.**

Д57 Python. 12 уроков для начинающих. — СПб.: БХВ-Петербург, 2023. — 272 с.: ил. — (Для начинающих)

ISBN 978-5-9775-1799-7

В 12 уроках показаны основы программирования и базовые конструкции языка Python. Изложены принципы различных стилей программирования. Даны понятия ввода-вывода, переменных, условий, потока чисел, циклов и списков, массивов, функций и рекурсий. Рассмотрены особенности структурного, объектно-ориентированного и функционального программирования. В каждой главе предложены практические задачи и дано их пошаговое решение с подробным описанием алгоритма.

*Для начинающих программистов*

УДК 004.43  
ББК 32.973.26-018.1

**Группа подготовки издания:**

Руководитель проекта	<i>Павел Шалин</i>
Зав. редакцией	<i>Людмила Гауль</i>
Редактор	<i>Григорий Добин</i>
Компьютерная верстка	<i>Ольги Сергиенко</i>
Дизайн серии	<i>Марины Дамбиевой</i>
Оформление обложки	<i>Зои Канторович</i>

"БХВ-Петербург", 191036, Санкт-Петербург, Гончарная ул., 20

ISBN 978-5-9775-1799-7

© Добряк П. В., 2023  
© Оформление. ООО "БХВ-Петербург",  
ООО "БХВ", 2023

# Оглавление

<b>Введение</b> .....	<b>5</b>
Как обучают языкам программирования? .....	5
И вот появился язык Python .....	7
Структура книги .....	7
Благодарности .....	8
Об авторе .....	9
<b>Урок 1. Ввод/вывод, переменные, условия</b> .....	<b>10</b>
1.1. Привет, мир! .....	10
1.2. Как тебя зовут? .....	13
1.3. Чему равно $12 + 34$ ? .....	15
1.4. Линейное уравнение .....	18
1.5. Тип треугольника .....	26
1.6. стакан чая и кружка кофе .....	29
<b>Урок 2. Поток чисел, циклы и списки</b> .....	<b>34</b>
2.1. Поток чисел, рекуррентные формулы .....	34
2.2. Поток чисел, списки .....	40
2.3. Векторы: длина, сумма, скалярное произведение .....	46
<b>Урок 3. Флаги. Структурное программирование и стиль Python</b> .....	<b>51</b>
3.1. Эпидемия на корабле .....	51
3.2. Является ли слово палиндромом? .....	55
3.3. Поиск и замена подстроки в строке .....	59
3.4. Сравнение чисел между собой. Множества .....	62
<b>Урок 4. Словари, рекуррентный индекс в списке</b> .....	<b>76</b>
4.1. Палиндром путем перестановки букв .....	76
4.2. Подстановки .....	83
<b>Урок 5. Двумерные списки</b> .....	<b>88</b>
5.1. Сложение, транспонирование и умножение матриц .....	88
5.2. Магический квадрат .....	98
Итоги уроков 1–5 .....	105

<b>Урок 6. Декомпозиция программы в функции .....</b>	<b>106</b>
6.1. Математические формулы как функции.....	106
6.2. Функция факториал с циклом.....	108
6.3. Библиотека формул комбинаторики .....	110
6.4. Декомпозиция магического квадрата в функции.....	114
<b>Урок 7. Рекурсии .....</b>	<b>117</b>
7.1. Рекурсивный факториал.....	117
7.2. Числа Фибоначчи без списка, списком, с рекурсией.....	119
7.3. Быстрое возведение в степень .....	125
7.4. Мемоизация чисел Фибоначчи .....	128
7.5. Генерация слов и перестановок .....	132
<b>Урок 8. Динамика по подотрезкам .....</b>	<b>139</b>
8.1. Палиндром максимальной длины вычеркиванием букв .....	139
8.2. Максимальный квадрат в матрице .....	155
<b>Урок 9. Функциональное программирование.....</b>	<b>163</b>
9.1. Сумма факториалов в функциональном стиле .....	163
9.2. Стандартные функционалы Python .....	170
9.3. Стандартные функционалы для «Эпидемии на корабле».....	173
9.4. Стандартные функционалы Python для суммы факториалов.....	175
9.5. Частичное применение функции на примере степени.....	178
9.6. Универсальный мемоизатор .....	184
9.7. Декораторы .....	191
9.8. Генераторы.....	199
Итоги уроков 6–9 .....	202
<b>Урок 10. Объектно-ориентированное программирование предметной области «Геометрия».....</b>	<b>204</b>
10.1. Класс «точка».....	204
10.2. Предметная область «Геометрия».....	211
10.3. Геометрическая фигура «многоугольник» .....	222
10.4. Составные фигуры.....	227
<b>Урок 11. Матрица в объектно-ориентированном стиле.....</b>	<b>231</b>
11.1. Конструктор, индексатор .....	231
11.2. Транспонирование, сложение, умножение.....	233
11.3. Определитель, обратная матрица, возведение в степень .....	235
<b>Урок 12. Программирование сложных коллекций .....</b>	<b>246</b>
12.1. Функторы .....	246
12.2. Коллекция «кольцо» и задача Иосифа Флавия.....	253
12.3. Мемоизация максимального квадрата матрицы в словаре .....	260
Итоги уроков 10–12 .....	268
<b>Заключение.....</b>	<b>270</b>
<b>Предметный указатель .....</b>	<b>271</b>

# Введение

## Как обучают языкам программирования?

Программированию обычно учат в четыре этапа:

1. Сначала изучают собственно язык программирования. То есть знакомятся с его языковыми конструкциями и тем, как пользоваться средой разработки. На это может уйти год.
2. На втором этапе ученик понимает, что язык он выучил, а программировать не умеет. У него не сформировалось алгоритмическое мышление. Он не может решить простые задачи вида «среди списка чисел найти сумму четных чисел». На хороших ИТ-специальностях изучают курс «Алгоритмы и структуры данных». В него можно погрузиться очень надолго.

Далее — «развилка» — можно выделить еще два этапа, но их порядок следования может быть различен:

3. Ученик узнает, что, оказывается, существуют разные стили программирования (как говорят программисты, *парадигмы*). И что мало овладеть алгоритмическим мышлением — нужно уметь переключаться. Это как учиться думать на принципиально разных языках — например: русском, татарском, арабском и китайском. А современные языки программирования высокого уровня мультипарадигменные, поэтому, даже если вы выучите тот или иной язык программирования, это не означает, что вы поймете программу, написанную на этом же самом языке. Это как англичанину выучить литературный русский язык и оказаться в среде, где «ботают по фене». В университетах традиционно этому посвящены отдельные курсы — например, «Объектно-ориентированное программирование», «Функциональное программирование».
4. А есть еще практические задачи — например, написать библиотеку обработки изображений, написать мобильное приложение. И человек, прошедший первые три этапа, понимает, что до практического программирования ему еще «как до Луны».

Получается, что надо поступать в вуз на хорошую ИТ-специальность. Но это — высокий конкурс, время и деньги. Однако ведь все-таки есть хорошие программи-

сты-самоучки, которые умеют решать практические задачи. И если поговорить с хорошо образованным программистом, то с высокой вероятностью обнаружится, что он научился программировать, в общем-то, самостоятельно. А университет лишь помог ему сформировать кругозор и погрузил его в нужную среду. Так что же делать?

Считаю, что должен быть некий средний путь. Бесполезно учить языковые конструкции, не решая задачи для формирования алгоритмического мышления. Алгоритмическое мышление бесполезно без выбора стиля программирования. А если надолго откладывать решение практических задач, то не будет мотивации учиться.

Помимо «срединного пути», реальное обучение циклично. Нет идеальных программ и методов обучения. Думаю, что нет программистов, которые бы обучились на одном курсе или по одной книге. Скучно слишком долго застревать на одном и том же этапе, изучая, например, алгоритмы динамического программирования. Лучше пройти все программирование в несколько циклов, уточняя и углубляя свои знания.

Есть методы обучения погружением, когда программированию обучают, решая одну большую практическую задачу. Теоретически я — за. Но практически... Как правило, такое обучение превращается в мастер-класс, когда преподаватель делает всю работу, а посетители курсов наблюдают за тем, как он ее делает. Далее — в зависимости от способностей преподавателя. Слушатели либо постепенно теряют нить рассуждений и погружаются в глубокий, но, увы, нездоровый сон, либо развлекаются спецэффектами на экране, которые мастерски программирует лектор. И пытаться так войти в ИТ — это все равно что сказать: «Мы будем строить небо-скреб, а все, что нужно, освоим по ходу дела».

Возможно, я огорчу читателя, но в программировании, как и в математике, нет царских путей. (Когда царь Птолемей спросил математика Евклида, как можно полегче и побыстрее овладеть геометрией, то Евклид ответил ему, что царских путей к геометрии нет.) И даже если следовать концепциям «обучение как игра» и «наука как развлечение», то это не отменит того, что вам придется думать, развивать свой мозг, менять свое мышление.

Когда я начинал обучать программированию в университете, то находился в идеальном положении. Я вел различные программистские дисциплины, начиная с первого курса по пятый. И мог обстоятельно преподавать программирование, за несколько лет формируя профессионалов.

Но жизнь не стоит на месте. И передо мной встала задача обучить программированию за срок в несколько месяцев: сначала школьников в рамках подготовки к ЕГЭ по информатике, потом магистров ИТ-специальностей (они пришли в магистратуру, окончив бакалавриат, и очень часто, владея всей ИТ-терминологией, не могли решить простейших задач, — мне пришлось их обучать программированию с нуля), потом слушателей курсов по программированию.

# И вот появился язык Python

И вот появился язык Python<sup>1</sup>...

Как оказалось, языковые конструкции, которым надо было посвящать целые уроки, можно рассказать между делом в течение нескольких минут. Например, типы данных (числа, строки и т. п.) — как они хранятся, какая память под них отводится, какие есть предельные значения чисел и как этим всем управлять. Впрочем, об этом можно, по крайней мере на начальном этапе, вообще не рассказывать!

Алгоритмы, которым раньше посвящалось несколько уроков на разных уровнях обучения, — например, алгоритмы сортировки списков, вообще потеряло смысл преподавать. В программе на Python, чтобы отсортировать список чисел по возрастанию, нужно написать два слова. Кстати, по этой причине я сперва не стал обучать языку Python своих детей. Я хотел сформировать у них алгоритмическое мышление, а на чем же еще его формировать, как не на алгоритмах сортировки? Но переменил решение, когда дошел до программирования на Python сложных алгоритмов. Оказалось, что программы с ними выглядят очень просто, т. к. всю подготовительную работу, разные вспомогательные и технические действия можно было порекомендовать Python. И мои ученики, занимаясь сложными алгоритмами, сразу видели суть этих алгоритмов, не отвлекаясь на всякие побочные технические действия. На других же языках «за деревьями было не видно леса».

Что касается разных стилей (парадигм) программирования, Python поддерживает их несколько. И они очень гармонично соединяются друг с другом. Python — язык лаконичный. Рассматривая написанные на нем программы, я ловил себя на мысли, что в них нет ни одного лишнего слова, а все технические подробности спрятаны. И если раньше мне требовался отдельный курс, чтобы познакомить студентов с функциональным программированием, то теперь знакомство укладывается в 1–2 урока.

В общем, с Python сбылась моя мечта: соединить в одном курсе преподавание языка, алгоритмов, структур данных и разных парадигм программирования.

Мой опыт преподавания воплотился в этой книге. Она представляет собой первый круг обучения, в котором мы пройдемся по всем основным языковым конструкциям и решим задачи, от простых до сложных, в разных стилях программирования.

## Структура книги

Книга разделена на 12 уроков, примерно соответствующих трехчасовым занятиям в том виде, как я преподаю Python на курсах. Урок состоит из нескольких разделов — отдельных задач и их модификаций. Вы можете заниматься медленнее и вдумчивее, проходя один раздел за одно занятие продолжительностью примерно

---

<sup>1</sup> Правильно его читать как «Пайтон» — с ударением на первом слоге.

1,5 часа — это составит 34 полуторачасовых занятия. Таким образом, на освоение материала у вас уйдет от 36 до 50 часов.

Основу книги составляют именно задачи, а языковые конструкции вводятся по мере необходимости. По ходу дела разъясняются и приемы, и стили программирования. Обычно бывает наоборот: излагается теоретический материал, а потом в конце главы идут задачи. Но именно решение все более сложных задач поддерживает мотивацию к учебе.

А вот как расположить задачи в нужном порядке, чтобы они шли от простых к сложным, и разделить их на группы, чтобы они примерно соответствовали изучаемым темам, — здесь пригодился мой разнообразный преподавательский опыт.

Еще тяжелее мне было следовать принципу минимализма. За время обучения языкам программирования я собрал большую коллекцию красивых задач. В эту книгу вошло ровно столько из них, чтобы хватило на то, чтобы обучить вас Python.

Каждый раздел начинается с формулировки задачи, потом идет перечень языковых конструкций и приемов программирования, потом прописан ход программирования (решения задачи).

Обычно в книгах по программированию описывается идея алгоритма, рисуется его блок-схема алгоритма, а потом идет код программы. Я задумался над вопросом: почему я не знаю ни одного человека, который занимается программированием и рисует блок-схемы? Да потому что они хороши только в том случае, когда программист уже решил задачу и хочет описать свой алгоритм, чтобы объяснить его другим людям.

В то же время обучение программированию не сводится к тому, чтобы рассказать идею алгоритма и показать блок-схему. Нужно показать, как рождался алгоритм, как программист постепенно, по шагам пишет программу, какие типичные ошибки его подстерегают и какие есть ложные пути. Поэтому я отказался от рисования блок-схем и вместо этого показываю пошаговое решение задачи. То есть фактически книга представляет собой стенограмму моих занятий, в которых я вместе с моими учениками постепенно пишу программы.

## Благодарности

Мне сложно вспомнить поименно всех моих учеников, студентов и слушателей курсов, с которыми я решал задачи из этой книги, и перечислить их всех, чтобы никого не забыть.

Отдельной благодарности заслуживают мои сыновья Андрей и Артем Добряки, которые в достаточно юном возрасте выдержали занятия со мной. Нужно поблагодарить Виктора Запорожца, который не только решал ряд задач из моей подборки на разных языках программирования, но еще и читал мои объяснения и дал ряд ценных советов по изложению материала. Другой мой ученик, Александр Ефимов, начал изучать программирование, уже будучи взрослым человеком. Я благодарен

ему за то, что он заставил меня обстоятельно и не торопясь объяснять материал еще доходчивее, чем я делал до этого.

Ошибки в моих программах, которые иногда случались, находил сам Python, а вот для исправления ошибок в человеческой речи по-прежнему нужен человек. Я благодарен моей жене Евгении Хрущевой, которая терпеливо редактировала мою книгу, исправляя ошибки.

## Об авторе

Павел Вадимович Добряк — кандидат технических наук, университетский преподаватель, ведущий занятия по различным языкам программирования, базам данных, искусственному интеллекту и проектированию информационных систем. Репетитор математики и информатики. Область научных интересов: сложные модели данных и алгоритмы, мультипарадигменное программирование.



# УРОК 1

## Ввод/вывод, переменные, условия

В этом уроке вы научитесь вводить данные с консоли и выводить сообщения на экран, делать вычисления по формулам, управлять ходом программы, чтобы выполнялись разные ее блоки в зависимости от истинности или ложности условий.

### 1.1. Привет, мир!

#### Задача

Программа должна вывести на экран сообщение «Привет, мир!».

**Языковая конструкция:** функция `print`.

#### Ход программирования

Традиционно первое, что пишут изучающие программирование, — это программа, выводящая на экран сообщение «Привет, мир!».

Если вы полный новичок в программировании, вам нужно сначала познакомиться со *средой разработки*, которой вы будете пользоваться, а именно — понять, где находятся два окна, или поля ввода:

1. В первом окне вы, как программист, будете писать и редактировать программу.
2. Во втором окне программа будет выводить сообщения и запрашивать у вас, как пользователя, данные для обработки.

Я рекомендую начать изучать программирование на Python со среды разработки IDLE Python. Это свободно распространяемое программное обеспечение, которое можно скачать с официального сайта: <https://www.python.org/downloads/>. Пройдя по этой ссылке, нажмите в открывшемся окне на кнопку **Download Python** (рис. 1.1).

После загрузки с сайта запустите скачанный файл, и среда разработки IDLE установится на ваш компьютер.

Когда вы запускаете IDLE, то открывается окно № 2, в котором вы будете вводить данные для обработки (рис. 1.2).

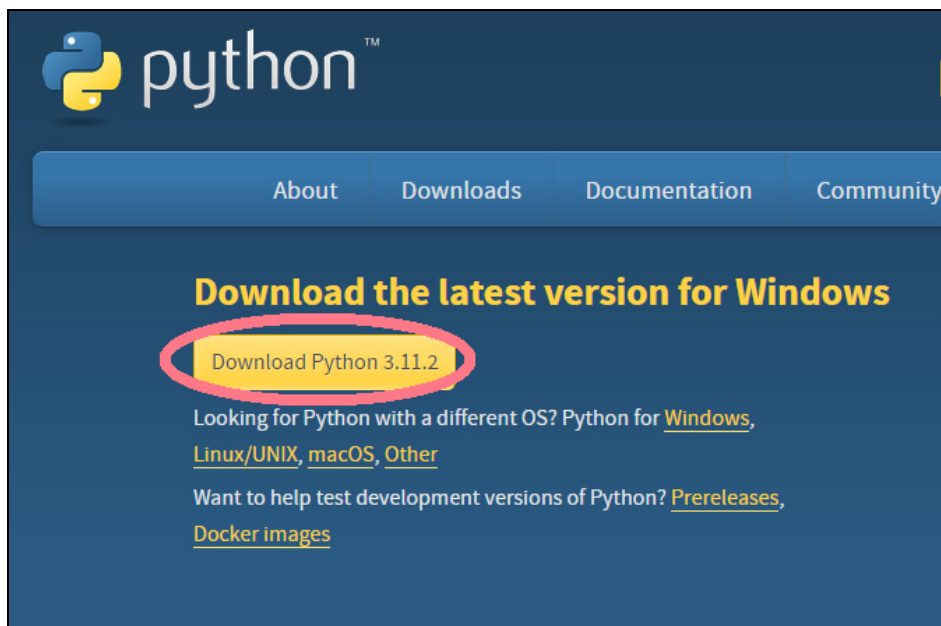


Рис. 1.1. Скачивание Python с официального сайта

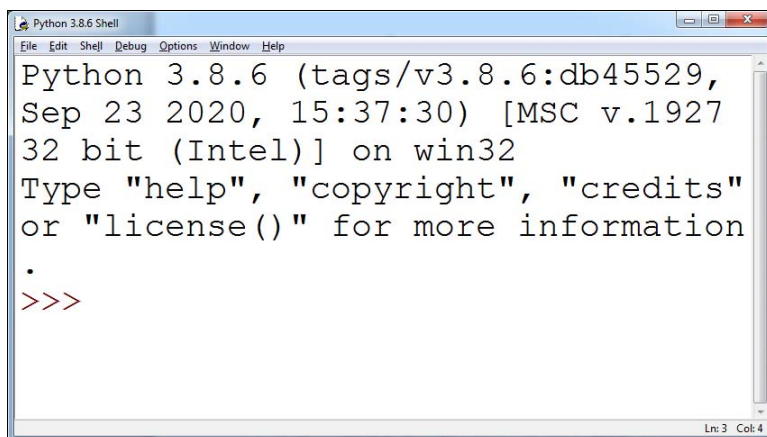


Рис. 1.2. Окно ввода данных для обработки

Чтобы начать писать программу, вам нужно открыть первое окно, создав файл с программой. Для этого выберите пункт меню **File | New File**, и откроется редактор программы (рис. 1.3).

И именно здесь вы будете писать программу, а результат увидите в предыдущем окне.

Особенность Python в том, что он очень лаконичен. Программа на Python «Привет, мир!», в отличие от программ на других языках высокого уровня (Фортрана, Паскаля, C++, Java, C# и др.), занимает одну строчку. Ее основа — это функция `print`.

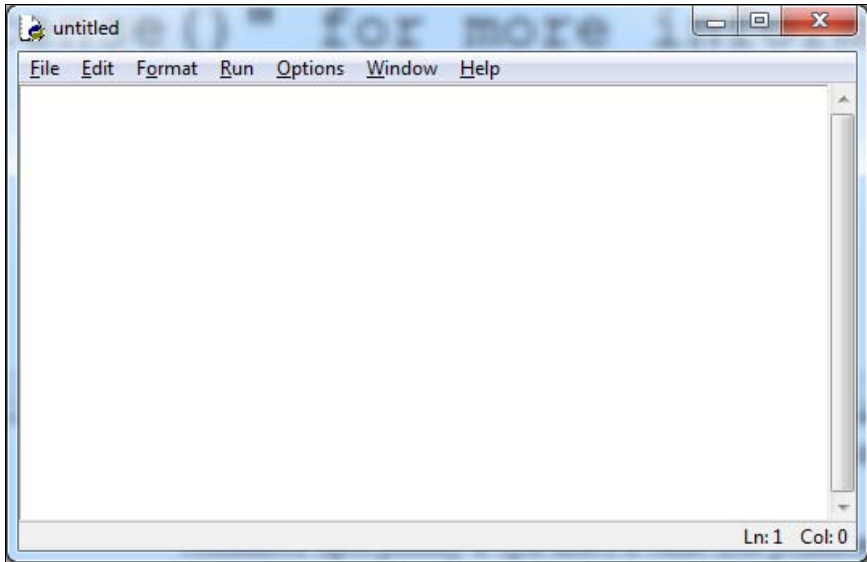


Рис. 1.3. Окно редактирования программы

Напишем программу в три шага в окне для редактирования программы:

**Шаг 1.** Пишем:

```
print()
```

**Шаг 2.** В круглых скобках пишем двойные кавычки:

```
print("")
```

или апострофы:

```
print('')
```

Я предпочитаю двойные кавычки, но это — дело вкуса.

**Шаг 3.** В кавычках напишем текст для вывода: `Привет, мир!` — и получим готовую программу (листинг 1.1.1).

#### Листинг 1.1.1. «Привет, мир!»

```
print("Привет, мир!")
```

Привыкайте писать сначала обе круглые скобки, потом обе пары кавычек внутри круглых скобок и только затем — текст внутри. Так делают все программисты. Разных скобок и кавычек придется писать очень много, и если вы какую-нибудь из них забудете, то программа не станет работать. У меня с этим связана история. На одной вечеринке возникла необходимость срочно набрать текст. Чем я и занялся. Человек, который видел меня в первый раз, сказал: «Да ты программист!» — «Как ты это понял?» — «Очень просто: ты открывающие и закрывающие скобки ставишь одновременно!» Такая вот у программистов профдеформация.

**Шаг 4.** Теперь нужно нашу программу запустить. Разберитесь, где находится кнопка запуска программы или соответствующий пункт меню. В IDLE Python это пункт меню **Run | Run Module**.

**Шаг 5.** Если не вышло никаких сообщений об ошибках, то смотрим в окне вывода нашу фразу:

```
Привет, мир!
```

## 1.2. Как тебя зовут?

### Задача

Добавим в предыдущую программу интерактивности. Теперь программа должна спросить пользователя, как его зовут, и поприветствовать его по имени — вывести сообщение: «Привет, введенное пользователем имя!»

**Языковые конструкции:** функции `print` и `input`, переменные.

### Ход программирования

**Шаг 1.** Для начала выведем сообщение: «Как тебя зовут?» — так же, как это мы делали в программе «Привет, мир!»:

```
print("Как тебя зовут?")
```

**Шаг 2.** Для ввода информации нам понадобится вызвать *функцию ввода* `input`:

```
input()
```

Обратите внимание на пустые круглые скобки. Поскольку `input`, как и `print`, является функцией, то после названия функции круглые скобки *обязательны* — они сигнализируют Python, что нужно вызвать функцию. Но этого мало, и поскольку результат ввода (имя человека) нам понадобится дальше для вывода, сохраним его в *переменной* `name`:

```
name=input()
```

Думаю, что у читателя есть интуитивное понимание того, что такое *переменная*, которую можно воспринимать так же, как и в математике или физике, — т. е. как некоторую величину, которая может принимать различные значения (например, температура, обозначаемая обычно буквой *T*). Переменные могут использоваться в математических формулах. Переменной соответствует некоторая область памяти компьютера, где хранится ее значение. Переменным вы можете давать любые названия, состоящие из строчных и заглавных латинских букв, арабских цифр и нижнего подчеркивания. Переменная должна начинаться на букву.

**Шаг 3.** Для вывода опять пользуемся функцией `print`:

```
print("Привет, ")
```

Но нам нужно еще вывести имя. Мы можем вывести несколько текстов с помощью одного вызова `print`. Для этого запишем нужные текстовые блоки через запятую:

```
print ("Привет", name)
```

Обратите внимание, что `name` мы написали без кавычек. Если мы хотим, чтобы вместо *слова* `name` (название переменной) подставилось его *значение*, то пишем переменную без кавычек. Мы получили готовую программу (листинг 1.2.1).

#### Листинг 1.2.1. Программа «Как тебя зовут?». Версия 1

```
print ("Как тебя зовут?")
name=input ()
print ("Привет, ", name)
```

Здесь мы уже получили *алгоритм* — последовательность действий, в нашем случае состоящую из трех команд, записанных в разных строках программы, которые выполняются по порядку. В скором будущем в наших алгоритмах появятся *ветвления* — выполнение разных последовательностей команд в зависимости от выполнения условий и *циклы* — многократное повторение одной и той же последовательности команд.

**Шаг 4.** Запускаем программу. В диалоговом окне пользователя видим вопрос:

Как тебя зовут?

Вводим Паша и получаем:

Как тебя зовут?

Паша

Привет, Паша

**Шаг 5.** Если вы до конца не поняли, какая разница между переменной и ее значением и когда нужно использовать кавычки, а когда — нет, введите программу (листинг 1.2.2) и посмотрите, к какому выводу на экран приводит каждая строчка программы.

#### Листинг 1.2.2. Эксперименты с кавычками

```
print ("Как тебя зовут?")
name =input ()
print (name)
print ("name")
print ("name", name)
print ("Название переменной", "name")
print ("значение переменной", name)
print ("Название переменной", "name", "значение переменной", name)
print ("Название переменной - name, а ее значение -", name)
```

#### Результат работы программы

Как тебя зовут?

Паша

```
Паша
name
name Паша
Название переменной name
значение переменной Паша
Название переменной name значение переменной Паша
Название переменной - name, а ее значение - Паша
```

**Шаг 6.** Программу можно сделать короче. Функция `input`, как и `print`, тоже может выводить сообщения на экран (листинг 1.2.3).

#### Листинг 1.2.3. Программа «Как тебя зовут?». Версия 2

```
name=input("Как тебя зовут?")
print("Привет, ",name)
```

**Шаг 7.** Программу можно сделать еще короче. Поскольку переменная `name` используется только один раз, просто копируем ее код туда, где она сработает, — т. е. `input` копируем внутрь `print` на место `name` и получаем еще одну версию программы (листинг 1.2.4).

#### Листинг 1.2.4. Программа «Как тебя зовут?». Версия 3

```
print("Привет, ",input("Как тебя зовут?"))
```

## 1.3. Чему равно $12 + 34$ ?

### Задача

Пользователь вводит значения двух переменных. Надо подсчитать их сумму. Например, в диалоговом окне пользователь вводит: 12 и 34, программа вычисляет их сумму и выводит ответ: 46.

**Языковые конструкции:** математические формулы, преобразования типов.

### Ход программирования

**Шаг 1.** Спрашиваем у пользователя значения двух переменных с именами `a` и `b`:

```
a=input("a=")
b=input("b=")
```

**Шаг 2.** Вводим третью переменную `c`, приравненную к формуле сложения:

```
c=a+b
```

**Шаг 3.** Выводим результат на экран:

```
print("a+b=",c)
```

и получаем готовую программу (листинг 1.3.1).

**Листинг 1.3.1. Сумма двух переменных строкового типа**

```
a=input("a=")
b=input("b=")
c=a+b
print("a+b=",c)
```

**Шаг 4.** Запускаем программу и получаем неожиданный результат:

```
a=12
b=34
a+b= 1234
```

Дело в том, что `input` возвращает *строковый* тип переменных. То есть переменные `a` и `b` хранят *строки*. А оператор «плюс», когда видит, что надо складывать строки, соединяет их вместе.

Здесь мы впервые столкнулись с тем, что у переменной есть не только имя и значение, но и *тип* — множество значений, которые она может принимать, и *операции*, которые можно над ней совершать. В написанной нами программе *тип данных* — это строки, и оператор сложения для них работает как соединение строк. А нам нужен *числовой* тип данных, которых в Python три: целые числа, числа с дробной частью (плавающей точкой) и комплексные числа.

**Шаг 5.** Нам необходимо преобразовать вводимые строки в числа, чтобы получить результат математического сложения. Для этого преобразования воспользуемся функцией `int()` (листинг 1.3.2)

**Листинг 1.3.2. Сумма двух переменных целого типа**

```
a=int(input("a="))
b=int(input("b="))
c=a+b
print("a+b=",c)
```

**Шаг 6.** Запускаем программу и видим результат:

```
a=12
b=34
a+b= 46
```

**Шаг 7.** Если мы хотим работать не только с целыми числами, но и с числами с дробной частью (действительными числами), то надо сделать преобразование не с помощью `int`, а с помощью `float` (листинг 1.3.3).

**Листинг 1.3.3. Сумма двух переменных типа «число с плавающей запятой»**

```
a=float(input("a="))
b=float(input("b="))
c=a+b
print("a+b=",c)
```

Результат выполнения:

```
a=12.3
b=45.6
a+b= 57.9
```

Обратите внимание, что дробная часть вводится в «западном» варианте — через точку, а не в российском, через запятую.

Еще раз посмотрите на программы (табл. 1.1) и запомните разницу между ними.

**Таблица 1.1.** Сложение переменных разных типов

Программа 1	Программа 2	Программа 3
<pre>a=input("a=") b=input("b=") c=a+b print("a+b=",c)</pre>	<pre>a=int(input("a=")) b=int(input("b=")) c=a+b print("a+b=",c)</pre>	<pre>a=float(input("a=")) b=float(input("b=")) c=a+b print("a+b=",c)</pre>
Результат 1	Результат 2	Результат 3
<pre>a=12 b=34 a+b= 1234</pre>	<pre>a=12 b=34 a+b= 46</pre>	<pre>a=12.3 b=45.6 a+b= 57.9</pre>

**Шаг 8.** Сделаем программу в минималистическом варианте — без задания вопросов и введения переменной *c*. Складывать переменные будем прямо в функции `print` (листинг 1.3.4).

**Листинг 1.3.4.** Минималистический вариант суммы двух чисел

```
a=float(input())
b=float(input())
print(a+b)
```

**Шаг 9.** Разберемся, какие математические операторы мы можем использовать. Запустите код из листинга 1.3.5 и посмотрите на результат.

Листинг 1.3.5. Математические операторы	Результат
<code>a=7</code>	
<code>b=2</code>	
<code>c=5</code>	
<code>print(a+b)</code>	9
<code>print(a-b)</code>	5
<code>print(a*b)</code>	14
<code>print(a**b)</code>	49
<code>print(a/b)</code>	3.5
<code>print(a//b)</code>	3
<code>print(a%b)</code>	1
<code>print(a/b+c)</code>	8.5

```
print (a/(b+c))           | 1.0
print (a/b*c)             | 17.5
print (a/(b*c))           | 0.7
print (int (a/b))         | 3
```

Обратите внимание на неочевидные операторы `**`, `//`, `%` и на то, как скобки влияют на результат.

## 1.4. Линейное уравнение

### Задача

Решить линейное уравнение  $kx + b = 0$ .

**Языковые конструкции:** математические формулы, условия `if ... elif ... else`, операторы для составных условий: `and`, `or`, `not`.

### Ход программирования

**Шаг 1.** Разбираемся, какие переменные вводятся, а какие вычисляются. Вводятся  $k$  и  $b$ , а  $x$  — вычисляется. Для ввода используем `input`. Обратите внимание:  $x$  с помощью `input` вводить не нужно (это ошибка № 1 начинающих программистов):

```
print ('Введите числа:')
k=int(input ())
b=int(input ())
```

**Шаг 2.** Сам Python решать уравнения не умеет (это могут делать только математические программы — например, Mathcad, в которых есть элементы программирования), поэтому вводить формулу  $kx + b = 0$  в программу не нужно (это ошибка № 2 начинающих программистов). Формулу для вычисления  $x$  выводим сами:

```
x=-b/k
```

**Шаг 3.** Организуем вывод на экран с помощью `print` и получаем готовую программу (листинг 1.4.1).

#### Листинг 1.4.1. Линейное уравнение. Версия 1

```
print ("Введите числа:")
k=int(input ())
b=int(input ())
x=-b/k
print ("x=",x)
```

**Шаг 4.** Запускаем и проверяем результат:

**Введите числа:**

```
2
3
x= -1.5
```