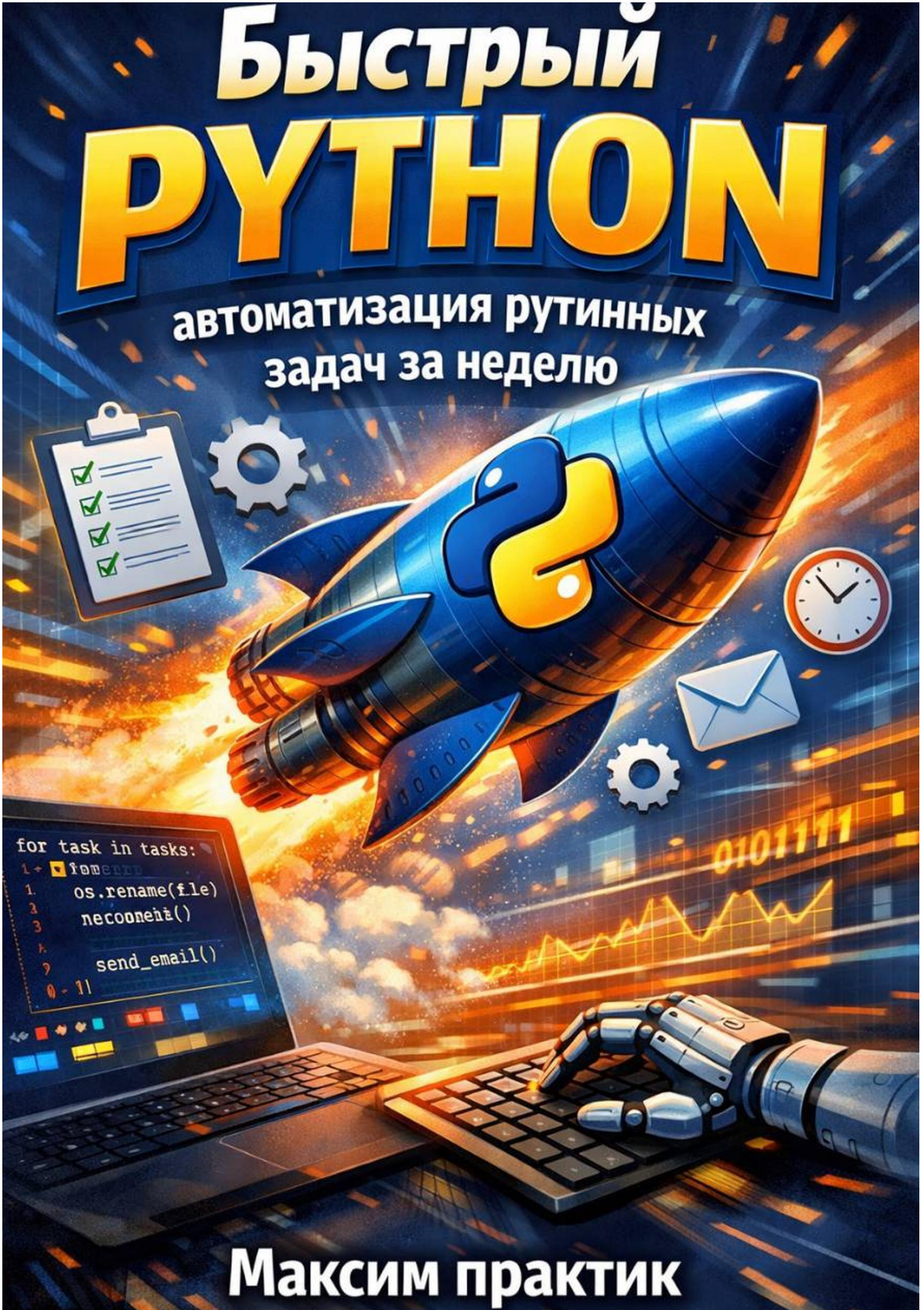
Быстрый PYTHON

автоматизация рутинных задач за неделю



Максим практик

Максим Практик

Быстрый Python. автоматизация рутинных задач за неделю

«Автор»
2026

Практик М.

Быстрый Python. автоматизация рутинных задач за неделю / М. Практик — «Автор», 2026

Практическое руководство по освоению Python для автоматизации повседневных компьютерных задач за одну неделю. Книга предназначена для новичков без опыта программирования и предоставляет пошаговые инструкции по созданию полезных скриптов. Читатель научится автоматизировать работу с файлами, обработку данных, взаимодействие с веб-страницами, отправку уведомлений и многое другое. Каждая глава — это конкретная задача и готовое решение, которое можно сразу применить. Цель книги — дать практический навык, который сэкономит часы ручного труда и откроет новые возможности в использовании компьютера.

Содержание

Максим Практик
Быстрый Python. автоматизация рутинных задач за неделю

Вступление

Приветствую вас, будущий автоматизатор!

Вы держите в руках (или на экране) практическое руководство, которое за семь дней превратит вас из новичка, возможно, лишь слышавшего о Python, в уверенного пользователя, способного заставить компьютер работать на себя. Эта книга – не про академическое программирование, не про сложные алгоритмы и теории. Это практикум по написанию простых, но чрезвычайно полезных скриптов, которые возьмут на себя рутину, сберегут ваше время и откроют новые возможности в работе с компьютером.

Каждый день мы сталкиваемся с однообразными, повторяющимися задачами: сотни файлов нужно переименовать, данные из разных источников – собрать в одну таблицу, регулярно проверять сайты на обновления или отправлять однотипные письма. Выполнять это вручную – скучно, долго и чревато ошибками. Именно здесь на помощь приходит Python – язык программирования, знаменитый своей простотой, читаемостью и огромным набором инструментов для автоматизации буквально всего.

Кому будет полезна эта книга? * **Офисным работникам**, уставшим от монотонного копирования данных между Excel, Word и почтой. * **Системным администраторам** и IT-специалистам, желающим автоматизировать обслуживание компьютеров и сетей. * **Студентам и исследователям**, которым нужно быстро обрабатывать большие объёмы данных или текстов. * **Всем любознательным пользователям ПК**, кто хочет выйти за рамки стандартного софта и научить компьютер подстраиваться под свои нужды.

Мы начнём с абсолютного нуля: установим Python, разберём базовый синтаксис и уже к концу первого дня напишем первые скрипты для работы с файлами. Каждая последующая глава – это новый шаг вперёд и новый набор конкретных задач, которые вы сможете решить сразу после прочтения. Вы не просто изучите команды, а поймёте логику автоматизации, научитесь «видеть» рутину вокруг себя и превращать её в несколько строк кода.

Наша цель – не сделать из вас профессионального разработчика (хотя это может стать отличным стартом), а дать вам в руки мощный и практический инструмент. Инструмент, который экономит самый ценный ресурс – время, освобождая его для творчества, анализа и решения действительно интересных задач.

Готовы за неделю изменить свой подход к работе за компьютером? Тогда вперёд – к первой главе!

Часть 1. Основы Python для автоматизации

Знакомство с Python и установка окружения

Представьте, что вы приехали на большую стройку. Повсюду лежат материалы, стоят машины, но главного – мастера, который знает, что с этим всем делать, – нет. Компьютер без программы – это примерно такая же картина. Много возможностей, но они спят. Python – это тот самый мастер-прораб, который просыпается, когда вы ему звоните, и говорит: ‘Я понял, что нужно сделать, давайте начнем’. Наша задача сегодня – не просто позвать его, а познакомиться, пригласить в гости и обустроить ему рабочее место на вашем компьютере. Это и называется ‘установить окружение’.

Почему именно Python? Давайте без заумных фраз про ‘высокоуровневый’ и ‘интерпретируемый’. Представьте языки программирования как реальные языки. Некоторые – как латынь, очень точные и строгие, но говорить на них в быту неудобно. Другие – как сленг подростков, быстро меняются и не всем понятны. Python – это как разговорный, современный и при этом очень понятный язык. Он создавался с идеей, что код должен легко читаться, будто это рассказ или инструкция на обычном языке. В мире автоматизации это золотой стандарт, потому что часто скрипт пишется один раз, а читается и правится потом много раз. И когда вы через месяц посмотрите на свой код, вы с большой вероятностью поймете, что же вы там написали.

Устанавливаем Python

Процесс установки Python на компьютер похож на установку любой другой программы, но с одним важным нюансом – нужно поставить галочку в одном волшебном месте. Сначала заходим на официальный сайт python.org. Не пугайтесь английского, мы найдем нужную кнопку. На главной странице вы увидите большую желтую кнопку с надписью ‘Downloads’. Нажимаем на нее, и сайт, скорее всего, сам предложит вам последнюю версию для вашей операционной системы – Windows, macOS или Linux. Скачиваем установщик и запускаем его.

Вот тут наступает ключевой момент. В начале установщика, в самом первом окне, будет маленький, но крайне важный пункт в самом низу: ‘Add Python to PATH’. Рядом с ним будет пустая галочка. Ваша святая обязанность – поставить эту галочку. Что такое PATH? Это как адресная книга вашей операционной системы. Когда вы ставите галочку, вы говорите системе: ‘Запомни, пожалуйста, где жит наш новый мастер-прораб Python, чтобы мы могли позвать его отовсюду, просто по имени’. Если не поставить галочку, вам придется каждый раз, когда вы захотите с ним поговорить, заходить к нему домой полным адресом, а это неудобно. Ставим галочку и жмем ‘Install Now’. Через пару минут мастер будет готов к работе.

Первый разговор

После установки нужно убедиться, что все прошло хорошо и система действительно узнает Python. Открываем командную строку. На Windows это делается через поиск в меню ‘Пуск’ – вводим ‘cmd’ и нажимаем Enter. На macOS открываем программу ‘Терминал’ через поиск Spotlight. Появится черное или белое окно с мигающим курсором. Это консоль, наш прямой проводник к душе компьютера. Теперь вводим туда магическую фразу: python –version. И нажимаем Enter. Если вы все сделали правильно, в следующей строке появится ответ что-то вроде ‘Python 3.11.4’. Это система здоровается с Python и спрашивает его версию, а он в ответ называет свой ‘титул’. Если же вы видите ошибку, что команда не найдена, значит, та

самая галочка про PATH не сработала. Не беда, это исправимо, но требует чуть больше шагов, которые легко гуглятся по запросу ‘добавить Python в PATH’ для вашей системы. Главное – не сдаваться на этом этапе.

Где писать код

Теперь, когда прораб на месте, нужно определиться, где мы будем с ним общаться. Можно, конечно, разговаривать прямо в командной строке, но это как обсуждать проект стройки, стоя посреди участка и крича друг другу. Удобнее сесть в бытовку, разложить чертежи. Для этого существуют специальные программы – редакторы кода и среды разработки (IDE).

Не стоит сразу лезть в самые мощные и сложные IDE. Они как огромный командный центр с сотнями кнопок, которые вам пока не нужны. Начните с простого текстового редактора, который понимает Python. Отличный и бесплатный вариант – Visual Studio Code (или просто VS Code). Его легко найти в интернете, скачать и установить. Он легкий, понятный и после установки сразу предложит поставить расширение для Python, что сделает его вашим лучшим другом. В нем вы будете писать скрипты – текстовые файлы с расширением .py. Это и есть те самые инструкции для нашего прораба Python.

Попробуйте создать прямо сейчас на рабочем столе файл с именем test.py. Откройте его в VS Code или даже в простом ‘Блокноте’. Напишите там одну строчку: print(‘Привет, я мастер Python!’). Сохраните файл. Теперь откройте командную строку, перейдите в папку ‘Рабочий стол’ (команда ‘cd Desktop’ на английской раскладке) и выполните команду python test.py. В окне консоли вы должны увидеть ваше приветствие. Поздравляю, вы только что запустили свою первую программу! Вы отдали прорабу первую устную команду, и он ее выполнил.

Мысли перед дорогой

На этом этапе многие откладывают все на потом. Срабатывает страх: ‘А вдруг я все сломаю?’, ‘Это слишком сложно для меня’. Давайте сделаем паузу. Закройте глаза и вспомните, как вы в первый раз устанавливали сложное приложение на телефон или настраивали новый телевизор. Было страшно нажать не ту кнопку? Скорее всего, да. Но вы же разобрались, методом тыка и подсказок. Здесь то же самое. Компьютер – очень терпеливый тренажер. Он не сломается от того, что вы установите Python или напишете не ту команду. В худшем случае вы получите понятное сообщение об ошибке, которое можно скопировать и найти в интернете. Миллионы людей прошли этот путь до вас. Теперь ваша очередь. Подумайте, сколько рутинных действий вы делаете за компьютером каждый день, которые, как вам кажется, нельзя автоматизировать. Возможно, уже через неделю вы посмотрите на них иначе.

Переменные и типы данных

Вспомните, как вы делаете покупки в магазине. У вас в руках корзина, в которую вы кладете разные предметы. Вы можете назвать эту корзину как угодно – «еда», «хозтовары» или даже «мои_покупки_на_вечер». Самое главное, что в корзину можно положить яблоко, пачку молока или упаковку батареек. Корзина – это переменная, а то, что вы в нее кладете – это данные. Python работает очень похоже.

Программа – это набор инструкций для компьютера, а чтобы давать инструкции, нужно где-то хранить информацию, с которой мы работаем. Вот для этого и нужны переменные. Переменная в Python – это просто имя, которое вы даете какому-то значению, чтобы потом его использовать. Создать переменную проще простого – придумайте имя и поставьте знак равенства. Слева от знака равенства – имя корзины, справа – то, что вы в нее кладете.

Коробка_с_числом = 42 Надпись_на_двери = “Вход воспрещен” Флаг_включения = True

Вы только что создали три переменные и положили в них три принципиально разных типа данных. И это ключевой момент – данные бывают разные, и Python это прекрасно понимает. Цифра 42 – это целое число, или, как говорят программисты, целочисленный тип – integer (сокращенно int). Текст в кавычках “Вход воспрещен” – это строка, или string (str). А слово True (имеющее пару – False) – это булевый тип, или boolean (bool), который может быть только либо истиной, либо ложью. Компьютер по-разному обрабатывает эти типы: с числами можно производить арифметические действия, строки можно склеивать или резать на части, а булевы значения помогают принимать решения.

Коробки с разными отделениями

Давайте присмотримся к этим типам поближе. С числами все более-менее понятно: 10, -5, 0, 100500. Python отлично справляется и с дробными числами, которые называются числами с плавающей точкой – float. Например, температура 36.6, курс доллара 75.43 или математическое число Пи 3.14159 – все это float. Главное отличие от целых чисел – наличие точки, разделяющей целую и дробную часть.

Строки – это все, что заключено в кавычки. Кавычки могут быть одинарными или двойными, главное, чтобы они совпадали в начале и в конце. Это ваш универсальный инструмент для работы с текстом: названия файлов, пути к папкам, содержимое писем, данные из веб-страниц. Вы будете использовать строки постоянно, и одно из первых заданий по автоматизации почти наверняка будет связано с обработкой текста.

А теперь представьте, что вам нужно сохранить не одно значение, а целый список покупок. Для этого в Python есть специальный тип данных – список (list). Создается он с помощью квадратных скобок, а элементы в нем разделяются запятыми. Список покупок = [“молоко”, “хлеб”, “яйца”, “сыр”]. В списке могут храниться элементы любых типов, даже другие списки. Список – это ваша рабочая лошадка, упорядоченная корзина, где каждый предмет лежит на своей позиции, и вы можете обратиться к нему по номеру.

Есть и другие, более специфические коробки – словари (dict), которые хранят данные в виде пар “ключ-значение” (как записная книжка с именами и телефонами), или кортежи (tuple) – неизменяемые списки. Но для старта нам вполне хватит чисел, строк, булевых значений и списков.

Как Python понимает, что в корзине

Одна из классных особенностей Python – это динамическая типизация. Это умное словосочетание означает простую вещь: вам не нужно заранее объявлять, какого типа данные вы собираетесь хранить в переменной. Вы просто кладете туда значение, а Python смотрит на него и сам понимает: “Ага, это число”, или “Так, это текст в кавычках, значит строка”. Более того, вы можете положить в одну и ту же переменную сначала число, потом строку, и Python не будет возражать. Хотя так делать не рекомендуется, чтобы не запутаться самому.

Но иногда вам нужно быть уверенным, с чем именно вы работаете. Для этого есть функция type(). Просто передайте ей вашу переменную, и она вам честно скажет, что внутри.

Вот вы написали: моя_переменная = 100. А потом спросили: print(type(моя_переменная)). И Python ответит вам: <class ‘int’>. Это значит, что внутри лежит целое число. Если бы там была строка, ответ был бы <class ‘str’>. Эта функция – ваш диагностический инструмент, когда что-то идет не так и вы не понимаете, почему скрипт не хочет складывать два “числа”, которые на самом деле являются текстом.

Превращения и волшебство

Жизнь была бы слишком простой, если бы типы данных всегда оставались сами собой. Часто их нужно преобразовывать. Допустим, вы получили от пользователя ввод, и это всегда строка. Даже если пользователь ввел цифру 10, для Python это текст “10”. И если вы попробуете прибавить к этому тексту число 5, вы получите ошибку. Нужно превратить строку в число. Для этого существуют функции-преобразователи: int(), float(), str().

Строка_с_числом = “10” Настоящее_число = int(Строка_с_числом) Результат = Настоящее_число + 5 # Получится 15

Или обратная ситуация: вам нужно вставить значение числа в готовое текстовое сообщение. Тогда вы преобразуете число в строку с помощью str() и спокойно склеиваете его с другим текстом. Эти преобразования будут встречаться вам на каждом шагу, особенно когда вы начнете читать данные из файлов или из интернета – оттуда почти всё приходит в виде текста.

Подумайте на минуту о вашей ежедневной работе за компьютером. Какие данные проходят через ваши руки? Наверняка это имена файлов (строки), размеры этих файлов (числа), даты их изменения (часто строка, которую потом нужно превратить в специальный тип для дат), списки задач или писем. Все эти сущности в программе будут представлены разными типами данных. Понимание, чем число отличается от строки, а строка от списка – это фундамент, на котором вы построите все свои скрипты. Без этого знания вы не сможете заставить компьютер правильно сложить два числа из разных отчетов или найти нужное слово в тысяче текстовых файлов.

Теперь, когда вы знаете про основные коробки для данных, можно переходить к следующему шагу – к действиям, которые мы можем совершать с содержимым этих корзин. Как из двух чисел получить третье, как из двух строк сделать одну или как вытащить нужный элемент из длинного списка. Но это уже тема для следующего разговора. А пока поэкспериментируйте в интерпретаторе: создавайте переменные с разными именами, кладите в них разные значения, проверяйте их типы и пробуйте преобразовывать. Помните, что лучший способ понять – это сделать самому.

Условия и циклы

Давайте представим самую обычную утреннюю рутину. Вы просыпаетесь, идете на кухню и смотрите на чайник. Ваш мозг мгновенно проводит молниеносную проверку: если чайник пустой, то нужно его наполнить водой и включить. Если же он полный, можно сразу нажимать кнопку. А потом, пока чай заваривается, вы, скорее всего, повторяете одно и то же действие несколько раз – помешиваете ложкой в кружке. Компьютер в своих задачах ведет себя точно так же. Он постоянно что-то проверяет и повторяет одни и те же действия сотни, тысячи раз. И сегодня мы научим его именно этому – принимать решения и работать в цикле.

Без этих двух концепций автоматизация просто невозможна. Любой полезный скрипт – это всегда комбинация условий и циклов. Условия – это развилки на дороге, которые говорят программе: «Если случилось вот это, иди налево, а если другое – тогда направо». Циклы – это та самая тропинка, по которой можно ходить кругами, пока не выполнится нужное условие, например, пока не переименуешь все файлы в папке.

Если бы да кабы – учим компьютер выбирать

В Python решение принимается с помощью оператора if, что в переводе с английского и означает «если». Конструкция выглядит очень похоже на обычную человеческую логику. Сначала мы задаем вопрос (условие), а потом говорим, что делать, если ответ «да» (True).

Вот простейший пример из жизни автоматизации. Допустим, у нас есть скрипт, который проверяет, есть ли в папке файлы с расширением .txt. Наш вопрос программе: «Если в имени файла встречается строка ".txt"?» Если да, то мы, к примеру, выводим сообщение, что текстовый файл найден. Код будет выглядеть как обычное предложение: если что-то верно, то сделай вот это. Красота в том, что условием может быть что угодно: проверка типа файла, сравнение размера, наличие определенного слова в тексте, текущее время или даже результат другой программы.

А что делать, если условие не выполнилось? Для этого есть продолжение – else (иначе). Это как план Б. Если файл не текстовый, то, может быть, его нужно переместить в другую папку или просто проигнорировать. Иногда вариантов больше двух – тогда на помощь приходит elif, что значит «иначе если». Это как длинная лестница проверок: если не первое, то проверь второе, если не второе, то третье и так далее. Например, скрипт, сортирующий файлы по типам, будет задавать целую цепочку вопросов: если расширение .jpg – в папку «Фото», иначе если .mp3 – в «Музыку», иначе если .pdf – в «Документы», а иначе – в «Разное». Попробуйте прямо сейчас мысленно пройтись по своей «Загрузкам» или «Рабочему столу». Сколько там файлов, которые можно рассортировать по таким простым правилам? Компьютер сделает это за секунду, не устанет и не ошибется.

Белка в колесе, или сила повторений

Теперь перейдем к самому мощному инструменту автоматизатора – циклам. Их главная суперсила – выполнять одно и то же действие много раз, не моргнув глазом. Представьте, что вам нужно отправить одинаковые письма ста разным людям или переименовать тысячу фотографий по единому шаблону. Вручную это – день тоски и однообразия. Для компьютера – три строчки кода и пара секунд работы.

В Python чаще всего используются два типа циклов. Первый – это цикл for. Его логику проще всего понять на примере. Допустим, у вас есть корзина с яблоками, и вам нужно каждое взвесить. Вы берете первое яблоко, кладете на весы, записываете вес. Потом второе, третье

и так далее, пока корзина не опустеет. Цикл for делает ровно это: он проходит по элементам какой-либо «коллекции» (например, списку имен файлов) и для каждого элемента выполняет указанный блок кода. Вам не нужно знать, сколько там всего файлов – десять или десять тысяч. Вы просто говорите: «Для каждого файла в этой папке сделай то-то». И он сделает. Это идеально подходит для обработки всего, что можно представить как набор элементов: строк в файле, писем в почтовом ящике, строк в таблице.

Второй тип – цикл while, что переводится как «пока». Он работает по другому принципу: не «для каждого», а «пока выполняется условие». Это как дежурный на вахте, который проверяет пропуска. Он будет выполнять свою работу (задавать вопросы) снова и снова, пока не сменится смена (пока условие не станет ложным). Например, ваш скрипт может мониторить папку, пока в нее не упадет новый файл, и тогда сразу его обработать. Или ждать, пока пользователь не введет правильный пароль, давая ему ограниченное количество попыток. Главное с этим циклом – не создать «вечный» вариант, то есть такое условие, которое никогда не станет ложным. Иначе ваш скрипт превратится в ту самую белку, которая будет бегать в колесе до скончания времен, пока вы не остановите его вручную.

Собираем пазл: условия внутри циклов

Настоящая магия начинается, когда мы соединяем эти два инструмента вместе. Это как если бы наша воображаемая белка в колесе не просто бежала, а на каждом круге смотрела по сторонам и принимала решение. Типичная и очень мощная конструкция в автоматизации выглядит так: цикл for проходит по всем файлам в папке, и для каждого файла оператор if проверяет какое-то условие (например, размер больше 10 МБ или дата создания старше месяца). И в зависимости от результата файл либо удаляется, либо копируется, либо переименовывается.

Давайте представим конкретную историю. Допустим, есть человек, который каждый день скачивает кучу отчетов. Файлы называются хаотично: «отчет_01.pdf», «data_final.xlsx», «итого_пятница.docx» и так далее. Ему нужно оставить только Excel-таблицы, созданные на прошлой неделе, и переместить их в отдельную папку для анализа. Вручную – это открыть папку, глазами искать .xlsx, смотреть свойства каждого, сравнивать даты, перетаскивать. Скучно и долго. Наш скрипт же сделает это в мгновение ока. Алгоритм будет простым: взять список всех файлов, для каждого файла проверить – если его расширение .xlsx И его дата создания попадает в диапазон прошлой недели, то скопировать его в целевую папку. Всего несколько строк кода, в которых for, if и несколько проверок работают как один слаженный механизм.

Подумайте на минуту, какая однообразная цифровая рутина есть в вашей жизни? Может, это ежемесячная сортировка фотографий из мессенджера? Или чистка почтового ящика от старых вложений? Или регулярная выгрузка данных из одной программы в другую? Практически в каждой такой задаче есть однотипные действия, повторяющиеся проверки. Там, где есть «если» и «повтори для каждого», – там живёт возможность для автоматизации.

Именно с этого момента ваши скрипты перестают быть простыми последовательностями команд. Они начинают «думать» – оценивать обстоятельства и методично, без устали выполнять задуманное. Вы даете им сценарий, а они играют свою роль. Это невероятно упрощает жизнь. Вам больше не нужно самому быть тем дежурным на вахте или той белкой в колесе. Вы становитесь тем, кто ставит задачу, нажимает кнопку «Пуск» и идет пить чай, пока компьютер послушно выполняет всю черновую работу. И в этом, пожалуй, и есть главный кайф автоматизации.

Функции

До сих пор мы писали скрипты как линейную последовательность команд: сверху вниз, строка за строкой. Это работает, пока задача маленькая и умещается на одном экране. Но представьте, что вы убираете квартиру. Вы можете ходить из комнаты в комнату и делать одно и то же: взять тряпку, намочить ее, протереть пол, прополоскать тряпку. А можно один раз описать этот процесс, назвать его, например, ‘помыть пол’, а потом просто говорить себе: ‘помыть пол в кухне’, ‘помыть пол в спальне’. Суть не меняется, но мыслей и повторений становится меньше. В программировании такие упакованные процессы называются функциями.

Функция – это не что иное, как именованный блок кода, который выполняет определенную задачу. Вы пишете этот блок один раз, даете ему понятное имя, а потом можете использовать (или, как говорят программисты, вызывать) его сколько угодно раз в разных частях вашей программы. Это как волшебный ящик. Вы кладете внутрь что-то (или ничего), нажимаете кнопку с именем ящика, а он выдает результат. Самое главное – вам не нужно каждый раз заглядывать внутрь и проверять, как там шестеренки крутятся. Вы просто пользуетесь им.

Как устроена функция

Создание функции в Python начинается с волшебного слова def. Это сокращение от define, то есть определить. После него пишется имя функции, круглые скобки, и в конце ставится двоеточие. Все, что относится к функции, пишется с отступом (обычно четыре пробела) ниже. Давайте создадим нашу первую, очень простую функцию, которая просто печатает приветствие.

def поздороваться(): print(‘Привет, мир автоматизации!’)

Обратите внимание на скобки после имени функции. Сейчас они пустые. Это значит, что наша функция не принимает никаких входных данных. Она как автомат, у которого только одна кнопка: нажал – получил стандартный результат. Чтобы заставить эту функцию работать, ее нужно вызвать. Для этого достаточно написать ее имя с теми же скобками. Вот так: поздороваться(). И в консоли появится наше сообщение. Вы можете написать эту строчку в коде десять раз подряд, и сообщение выведется десять раз, но сам код функции, ее описание, у вас будет записано всего один раз. Уже чувствуете, как это экономит силы и место?

А теперь представьте, что вам нужно не просто печатать одно и то же приветствие, а обращаться к разным людям или программам. Тут нам на помощь приходят параметры. Параметры – это те самые входные данные, которые мы кладем в наш волшебный ящик. Мы объявляем их в тех самых круглых скобках при создании функции. Давайте улучшим нашу функцию.

def поздороваться_по_имени(имя): print(‘Привет,’, имя, ‘! Добро пожаловать в мир автоматизации!’)

Теперь, когда мы будем вызывать эту функцию, мы обязаны указать в скобках то значение, которое будет подставлено вместо слова ‘имя’ внутри функции. Например, поздороваться_по_имени(‘Алексей’) или поздороваться_по_имени(‘Excel-таблица’). Функция станет гораздо более гибкой и полезной.

Функции, которые что-то возвращают

Часто нам нужно не просто что-то сделать (например, напечатать текст), а получить какой-то результат для дальнейшей работы. Представьте калькулятор. Вы вводите два числа и нажимаете кнопку ‘плюс’. Калькулятор не просто пишет результат на экране – он его вычис-

ляет и отдает вам. В мире функций за это отвечает ключевое слово return (что переводится как вернуть).

Создадим функцию, которая принимает два числа и возвращает их сумму.

def сложить(a, b): результат = a + b return результат

Что здесь происходит? Функция получает два числа (их значения будут подставлены вместо a и b). Внутри она создает переменную ‘результат’, куда записывает сумму. И затем команда return отправляет значение этой переменной туда, откуда функцию вызвали. Это важно понять. Сама по себе функция сложить(5, 3) в коде ничего не напечатает. Она просто вычислит число 8 и вернет его. А что с этим числом делать дальше – решаете вы. Можно сразу напечатать: print(сложить(5, 3)). Можно сохранить в переменную для будущих расчетов: мой_результат = сложить(5, 3). А можно даже передать это число в другую функцию. Возвращаемое значение – это и есть главный продукт, который производит наш волшебный ящик.

Зачем это нужно в автоматизации

Теперь давайте спустимся с небес теории на землю практики. Вспомните любую свою рутинную задачу. Например, ежемесячное составление отчета. В нем всегда одни и те же шаги: взять данные из одного файла, очистить их от лишних строк, посчитать итоги по столбцам, оформить в красивую таблицу и сохранить в новом файле. Каждый из этих шагов – идеальный кандидат в функцию. Вы пишете функцию ‘взять_данные_из_файла’, функцию ‘очистить_данные’, функцию ‘посчитать_итоги’. Ваш основной скрипт при этом превращается из длинного, запутанного текста в чистый и понятный план:

данные = взять_данные_из_файла(‘отчет_май.csv’) чистые_данные = очистить_данные(данные) итоги = посчитать_итоги(чистые_данные) сохранить_отчет(итоги, ‘итоговый_отчет_май.xlsx’)

Выглядит, как инструкция на человеческом языке, не правда ли? А самое прекрасное происходит, когда на следующий месяц вам нужно сделать то же самое, но с файлом ‘отчет_июнь.csv’. Вам не нужно переписывать весь скрипт! Вы просто меняете имя входного файла в первой строке. Вся сложная логика спрятана внутри функций, которые уже написаны и проверены. Если вы найдете ошибку в очистке данных, вам нужно будет исправить ее только в одном месте – внутри функции ‘очистить_данные’, и она автоматически поправится во всех ваших скриптах, где эта функция используется.

Подумайте на минутку о своих повторяющихся действиях за компьютером. Может, это переименование фотографий по шаблону ‘отпуск_2024_01.jpg’? Или еженедельная рассылка писем с одним и тем же текстом, но разными именами в адресной строке? Каждое такое действие – это готовая функция, которая только ждет, когда вы ее опишете. Вы уже чувствуете, как ваш будущий скрипт обретает структуру, становясь не грудой команд, а набором осмысленных блоков?

Немного магии: аргументы по умолчанию

Иногда вашей функции нужны не все данные каждый раз. Некоторые параметры могут быть стандартными, наиболее часто используемыми. Python позволяет задавать для параметров значения по умолчанию. Синтаксис прост: в объявлении функции вы пишете параметр со знаком равенства и значением. Например, функция создания папки. Обычно вы создаете ее в текущей директории, но иногда нужно указать другой путь.

def создать_папку(имя_папки, родительский_путь=‘.’): полный_путь = os.path.join(родительский_путь, имя_папки) os.mkdir(полный_путь)

Здесь '.' – это условное обозначение текущей папки. Теперь, если вы вызовете функцию как создать_папку('НоваяПапка'), она создаст папку 'НоваяПапка' прямо там, где находится ваш скрипт. Но если вам нужно создать ее где-то еще, вы можете явно указать второй аргумент: создать_папку('НоваяПапка', 'C:/Документы'). Это невероятно удобно, так как делает функции еще более гибкими и умными, избавляя вас от необходимости постоянно указывать очевидные вещи.

Функции – это краеугольный камень, на котором строится не только автоматизация, но и все программирование в целом. Они превращают хаос операций в порядок, повторяющееся – в однократное, сложное – в простое. Не стремитесь сразу написать идеальную, универсальную функцию на все случаи жизни. Начните с малого. Выделите в вашем следующем скрипте хотя бы один повторяющийся кусок кода, оберните его в def, дайте честное имя. Вы удивитесь, насколько понятнее и короче станет ваш код. И помните, каждая новая функция – это еще один кирпичик в стене, которая защищает ваше время от нашествия рутины.

Работа с файлами и исключениями

Представьте себе рабочий стол после большого проекта. Фотографии, документы, таблицы, архивы – всё свалено в кучу, как в шкафу после генеральной уборки, когда всё выбросил, а разбирать уже нет сил. Теперь представьте, что у вас есть помощник, который молча, быстро и без ошибок рассортирует всё по папкам: картинки – в одну, отчеты – в другую, старые версии – в архив. Этот помощник – ваш скрипт для работы с файлами, и сегодня мы его создадим. Работа с файлами – это основа основ автоматизации. Всё, с чем мы имеем дело на компьютере, – это файлы. Текст, картинки, настройки, базы данных. Научившись управлять ими с помощью кода, вы получите в руки волшебную палочку для наведения цифрового порядка.

Писать код, который работает в идеальных условиях, несложно. Сложно написать код, который не сломается, когда что-то пойдёт не так. А с файлами что-то идёт не так очень часто: нужный файл удалён, диск переполнен, у вас нет прав на запись, а кто-то забыл закрыть документ в редакторе. Если не подготовиться к таким сбоям, ваш красивый скрипт-помощник вместо порядка устроит настоящий цифровой потоп. Поэтому вторая половина нашей сегодняшней встречи посвящена исключениям – это механизм в Python, который позволяет предвидеть неприятности и обрабатывать их так, чтобы скрипт не падал с грохотом, а вежливо сообщал, что пошло не по плану, и шёл дальше.

Открытие и чтение: знакомство с содержимым

Всё начинается с открытия. В Python для работы с файлом нужно его открыть, указав путь к нему и режим работы. Режимы – это как ключи от разных дверей. ‘r’ – ключ только для чтения (read), ‘w’ – для записи (write, при этом старое содержимое файла стирается!), ‘a’ – для добавления (append) в конец файла. Просто представьте, что у вас есть блокнот. Режим ‘r’ – вы только смотрите в него. Режим ‘w’ – вы берёте чистый лист и пишете заново. Режим ‘a’ – вы перелистываете на последнюю страницу и дописываете новые мысли.

Когда файл открыт, можно читать его содержимое. Самый простой способ – прочитать всё сразу в одну строку. Но часто удобнее читать файл построчно, особенно если это лог-файл или конфигурация. После того как работа с файлом закончена, его нужно обязательно закрыть – сказать системе, что вы освободили его, и другие программы могут с ним работать. Это как вернуть книгу в библиотеку. Забыть закрыть файл – плохая привычка, которая может привести к ошибкам. Python помогает нам и тут: есть специальная конструкция ‘with’, которая автоматически закроет файл, как только вы выйдете из её блока. Это надёжнее и чище, поэтому мы будем использовать именно её.

Вспомните последний раз, когда вам приходилось искать что-то в большом текстовом отчёте или вытаскивать данные из лога. Как долго вы это делали? А теперь представьте, что три строки кода делают это за секунду, читая файл и проверяя каждую строку на нужное вам ключевое слово. Это и есть тот самый момент, когда рутина превращается в магию.

Запись и сохранение: оставляем след

Чтение – это хорошо, но настоящая сила – в изменении мира, то есть диска. Запись в файл позволяет вашему скрипту сохранять результаты работы: новый очищенный список, обработанные данные, отчёт об ошибках. Тут важно помнить про режим ‘w’. Он безжалостен. Если вы откроете существующий файл в режиме ‘w’ и что-то в него запишете, всё старое содержимое исчезнет навсегда. Это как взять тетрадь с годовым отчётом и начать писать на первой странице список покупок. Поэтому будьте осторожны: перезаписывайте только то, что не жалко,

или создавайте копии. Режим ‘a’ безопаснее – он только добавляет, но и он может раздуть файл до гигантских размеров, если добавлять в него без остановки.

Частая задача – прочитать один файл, обработать данные и записать результат в другой. Это классическая операция преобразования. Например, у вас есть CSV-файл с товарами, где цены указаны в долларах, а вам нужен файл с ценами в рублях. Скрипт читает исходный файл, для каждой строки пересчитывает цену по курсу и записывает новую строку в другой файл. Исходный файл остаётся нетронутым, а вы получаете нужный результат. Задумайтесь на минуту, какие данные в вашей работе или личной жизни нужно регулярно переводить из одного формата в другой? Возможно, это отчёты из одной программы, которые нужно подготовить для загрузки в другую. Именно такие задачи – идеальные кандидаты для автоматизации.

Что-то пошло не так: встречаем исключения

А теперь давайте смоделируем маленькую аварию. Ваш скрипт пытается открыть файл ‘secret_plans.txt’, но его нет на диске. Без подготовки интерпретатор Python остановится и выведет на экран красное пугающее сообщение ‘FileNotFoundError’. Для вас, сидящего за компьютером, это нестрашно. Но если этот скрипт запущен ночью на сервере и должен обработать сотни файлов, одна ошибка остановит всю работу. Вот для этого и нужна обработка исключений.

Исключение – это специальный сигнал в программе, который говорит: ‘Эй, тут случилось что-то, с чем я не знаю, как справиться!’. Наш задача – перехватить этот сигнал и сказать программе, что делать в такой ситуации. Конструкция для этого называется try-except. Вы ‘пытаетесь’ (try) выполнить рискованный кусок кода, например, открыть файл. Если всё идёт хорошо, блок ‘except’ игнорируется. Если возникает ошибка (то самое исключение), выполнение кода немедленно перепрыгивает в блок ‘except’, где вы можете обработать проблему: вывести понятное сообщение, создать отсутствующий файл, пропустить проблемное место и продолжить работу.

Конец ознакомительного фрагмента.