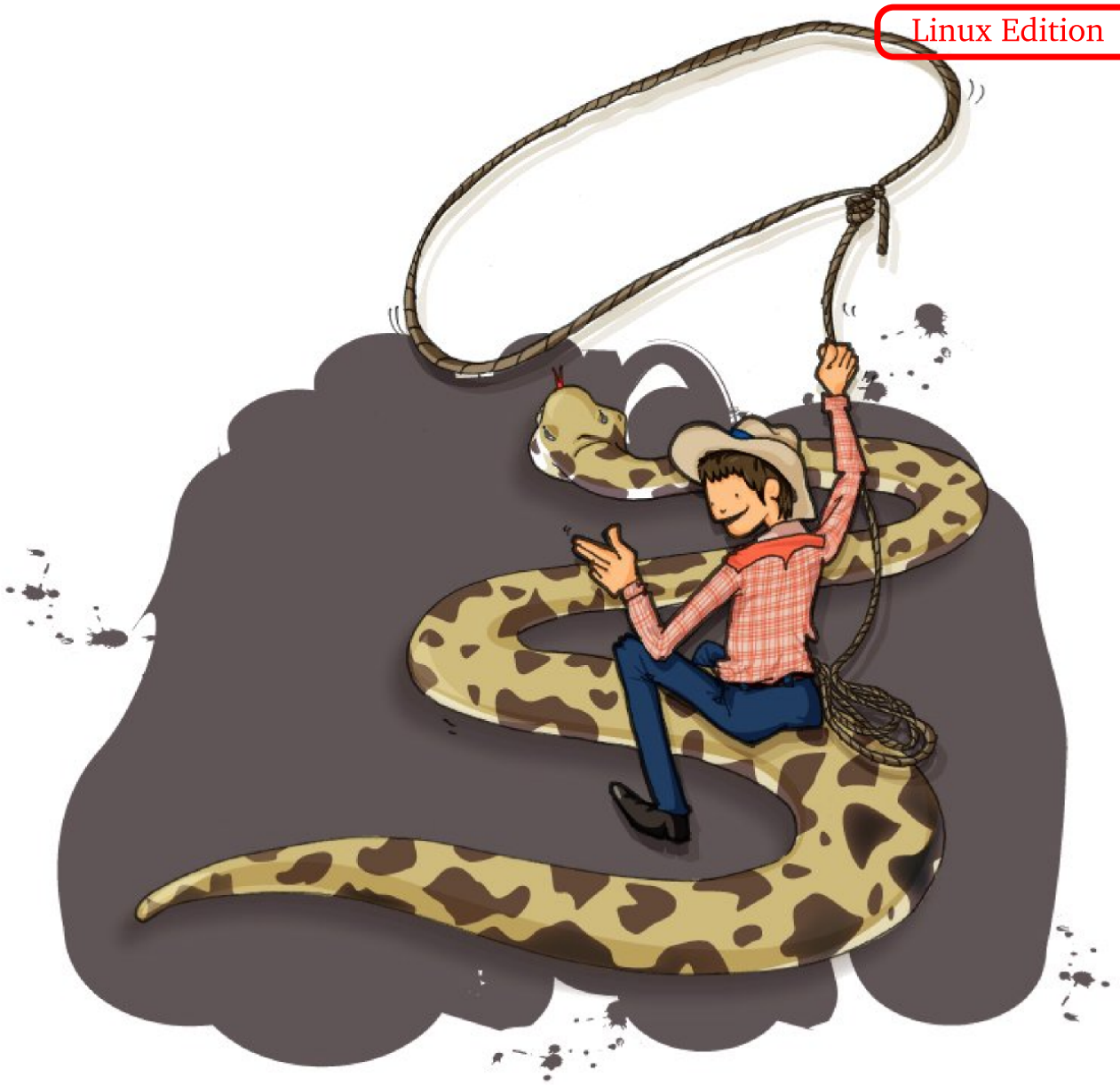
Snake Wrangling For Kids

Learning to Program with Python

Linux Edition



Written by Jason R. Briggs

Snake Wrangling for Kids, Learning to Program with Python
by Jason R. Briggs
перевод на русский язык:
Кочетов Е. М. <Egor.Kochetoff@gmail.com>

Version 0.7.7

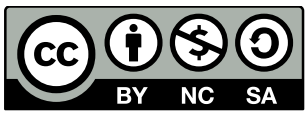
Cover art and illustrations by Nuthapitol C.

This book has been completely rewritten and updated, with new chapters (including developing graphical games), and new code examples. It also includes lots of fun programming puzzles to help cement the learning. Published by No Starch Press - available here: Python for Kids. Also find more info here.

Website:
http://www.briggs.net.nz/log/writing/snake-wrangling-for-kids
Github с переводом книги: `https://github.com/gluk47/swfk/tree/master/ru`

Thanks To:
Guido van Rossum (for benevolent dictatorship of the Python language), the members of the Edu-Sig mailing list (for helpful advice and commentary), author David Brin (the original instigator of this book), Michel Weinachter (for providing better quality versions of the illustrations), and various people for providing feedback and errata, including: Paulo J. S. Silva, Tom Pohl, Janet Lathan, Martin Schimmels, and Mike Cariaso (among others). Anyone left off this list, who shouldn't have been, is entirely due to premature senility on the part of the author.

License:



Ниже приведены основные положения лицензии.

Лицензия позволяет:

- **Делиться** — копировать, распространять и передавать эту работу,
- **Изменять** — адаптировать эту работу.

На следующих условиях:

Attribution. Обязательно явно указать авторство этой работы таким способом, как этого просит автор (и так, чтобы не казалось, что автор прямо поддерживает вас или вашу работу, основанную на его).

Noncommercial. Нельзя использовать эту работу и основанные на ней в коммерческих целях.

Share Alike. Если вы изменяете, перерабатываете, адаптируете и распространяете эту работу, обязательно распространять её на условиях этой же лицензии.

При любом использовании и распространении необходимо донести до получателей условия лицензии на эту работу.

Any of the above conditions can be waived if you get permission from the copyright holder.

Nothing in this license impairs or restricts the author's moral rights.

Оглавление

Вступление

Пара слов для родителей...

Уважаемый родитель или иной управляющий компьютером!

Чтобы ваш ребёнок смог начать знакомиться с программированием, вам нужно установить Python на компьютер. Эта книга была недавно обновлена до версии Python 3.0, самой новой и несовместимой с предыдущими, так что если у вас установлена более старая версия Pyhton, вам стоит скачать и более старую версию этой книги.

Установка Python — достаточно простая задача, но есть несколько тонкостей — в зависимости от используемой операционной системы. Если вы только что купили сверкающий новый компьютер и не имеете никаких идей, что с ним делать, а предыдущее предложение начало вызывать у вас нервную дрожь или холодный пот, то, пожалуй, лучше вам найти кого-то, кто сделает это за вас. Установка Python может занять от 15 минут до пары часов в зависимости от скорости интернета и компьютера.

Прежде всего, скачайте и установите последнюю версию Python 3 для вашего дистрибутива. Дистрибутивов очень много, так что инструкции для всех тут привести не получится... да и скорее всего, если вы используете Linux, то уже знаете, как это сделать. Наверное, вы даже возмущены самой идеей того чтобы рассказывать вам, как что-либо устанавливать, так что тут я остановлюсь.

После установки...

...Вам, возможно, придётся в течение первых пары глав посидеть с ребёнком рядом, но после нескольких примеров ему будет только приятнее читать книгу самому (с компьютером вместе). Нужно рассказать ребёнку, как вводить команды в консоль, как пользоваться текстовым редактором (наподобие блокнота; Microsoft Word никак не подойдёт), открывать и сохранять файлы в этом редакторе. Всё остальное расскажет эта книга.

Спасибо за уделённое время; с наилучшими пожеланиями,
КНИГА.

Глава 1

Не все змеи будут шипеть на тебя

Возможно, тебе подарили эту книгу на день рожденья. А может, на рождество. Например, так: тётя Агата (у всех есть тётя Агата, но не все об этом знают) хотела подарить носки, хотя и не парные, но оба красивые, на два размера больше — на вырост (и всё равно бы эти носки не пригодились потом). А потом вместо этого услышала про эту книгу (которую можно взять и напечатать), вспомнила твои вопросы про всякие компьютерные штуки, и твои непонятные объяснения, как пользоваться компьютером, оборвавшиеся в момент, когда она начала разговаривать с компьютерной мышью, и решила подарить эту книгу. Во всяком случае эта книга уж точно лучше пары разных носков.

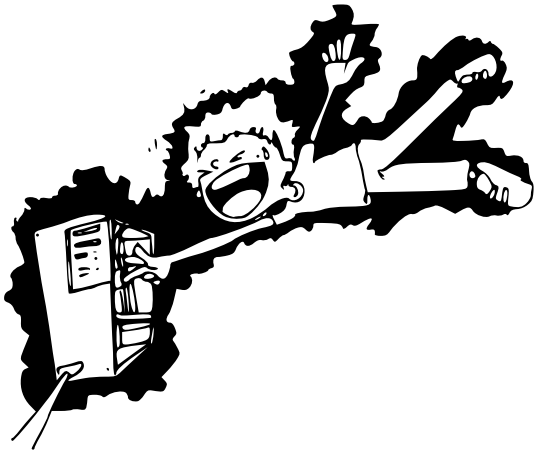
Надеюсь, я не слишком тебя разочаровываю тем, что я — возможно, напечатанная на какой-нибудь старой обёрточной бумаге (хотя если повезло, то и нет) — не слишком разговорчивая (прямо, скажем, совсем молчаливая) книга, с пугающим словом «изучение» в названии... Но представь на минутку и мои ощущения. Если бы вот ты был персонажем из какой-нибудь книги про волшебников, одна из которых наверняка есть у тебя в спальне на книжной полке, — у меня бы могли быть зубы... или даже глаза! А ещё какие-нибудь движущиеся картинки, таинственные звуки... ладно, чего я. В общем, я просто бумажная книжка, хотя могло бы быть и лучше.

Ах, много бы я дала за пару хороших острых челюстей...

Но вообще, быть конкретно такой книжкой тоже не слишком печально. Ну не могу я говорить... пальцы покусывать не могу; зато могу рассказать немного о том, что заставляет компьютеры работать. Не про разные аппаратные штуки — все эти провода, платы, чипы — они меня немного пугают. Электричеством, например, могут ударить (так что не стоит и пытаться ту-

да лезть, как по мне). Я могу рассказать о том, что удивительным образом скрыто внутри всех этих проводов, микросхем и что делает компьютер по-настоящему полезным.

Вообще, это здорово похоже на мысли, например, в твоей голове. Если бы мыслей у тебя не было — сидел бы ты, скажем, на полу в спальне и бессмысленно смотрел в пространство перед собой. Без *программ* компьютеры бы могли приносить пользу, пожалуй, разве что как стопор для двери. Да и то посредственный: вечно все бы об него спотыкались по ночам. А что может быть хуже, чем удариться ночью в темноте пальцем ноги с размаху о железный угол...



Итак, я всего лишь книга. И мне это хорошо известно.

Вообще, у тебя в семье могут быть разные устройства вроде Playstation, Xbox, Wii — игровые консоли, — а ещё DVD-проигрыватель, может, даже современный холодильник и игрушечная машинка. В них во всех есть программы, которые делают их намного полезнее, чем если бы эти штуки были без программ. В DVD-проигрывателе есть программа для чтения и воспроизведения дисков. В холодильнике — какая-то простая программа для поддержания температуры при минимальных затратах электричества. В машинке — программа для приёма команд с пульта управления и для езды в ответ на эти команды. А в настоящих машинах программы показывают маршруты в объезд пробок и сигналят водителю, когда он паркуется, чтоб он никуда не въехал (в стену или соседнюю машину).

Зная, как писать программы, ты сможешь сделать множество самых разных полезных вещей. Можно свою игру написать. Можно писать страницы в интернете, которые что-нибудь делают, а не просто показывают текст и картинки. Можно упрощать себе выполнение домашней работы.

Так вот, пора приступить к чему-то чуть более интересному, чем эти рассуждения.

1.1. Пара слов про язык

Так же как и у людей, определённо как у китов, возможно, и у дельфинов и, возможно, у родителей (тут, конечно, спорно), у компьютеров есть свой собственный язык. Вообще, как и у людей, у компьютеров много языков. Какую букву английского алфавита ни возьми, она называет какой-нибудь язык. Вот, например, буквы A, B, C, D, E — не только буквы, но и названия языков программирования (что ещё раз доказывает, что у взрослых никакого воображения и хорошо бы им давать почитать хотя бы словарь перед тем, как названия придумывать).

Есть ещё языки программирования, названные в честь людей[1], есть языки-сокращения из заглавных букв (SQL, например), есть немножко названных в честь телевизионных шоу. А, да, ещё если дописать к этим буквам всяких значков типа плюсиков, решёточек (+, #), то тоже получатся названия разных языков программирования. И ещё впридачу некоторые языки очень похожи и отличаются только в каких-то мелочах.

Что я говорила? Никакого воображения!

К счастью, многие из языков уже почти не используются или совсем исчезли; но однако же список способов «говорить» с компьютером всё ещё пугающе велик. Я буду обсуждать только один из них, потому что иначе всё так и закончится их перечислением, не успеем мы приступить к чему-то действительно интересному.

1.2. Орден Неядовитых Удушающих Змей...

... или просто питонов.

Вообще, питон — не только змея[2], но и язык программирования. Многие называют его «пайтон», как принято за рубежом, где его и придумали (и пишут его название как Python). Язык, правда, был назван не в честь змеи — это один из немногих языков программирования, названных в честь телевизионного шоу. Монти Пайтон (Monty Python) — британское телешоу, популярное с 1970х годов. Требуется достичь некоторого возраста и иметь определённый склад ума, чтобы счесть его забавным, но многим нравится. Хотя лет до 12 смотреть вообще смысла нет, будет скучно и непонятно.

Есть несколько особенностей Питона (языка программирования, а не змеи), делающих его очень полезным, чтобы учиться программировать. Для нас сейчас важнее всего то, что используя его, можно быстро сесть и начать писать

[1] например, язык Ада — в честь Ады Лавлейс. А есть ещё язык РАЯ.

[2] которая может не есть полтора года кряду!

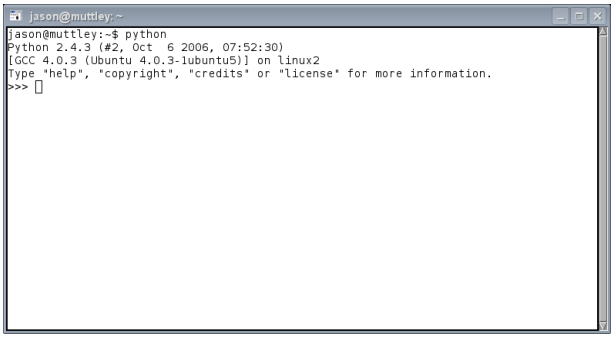


Рис. 1.1: Консоль Питона в Линуксе.

какие-то программки, пусть и очень простые, без долгих разговоров и объяснений.

Это тот момент, когда надо убедиться, что твоя мама, папа или кто там управляет компьютером прочли часть «Заметка для мам и пап». Есть хороший способ проверить это. Спроси у них, как называется программа-*эмулятор терминала* — это может быть `yakuake`, `konsole`, `rxvt`, `xterm` или ещё какая-нибудь, их очень много бывает — именно поэтому придётся спросить. Запусти эту программу, напиши *в командной строке* 'python3' (без кавычек) и нажми на клавиатуре клавишу Enter. На экране должно появиться что-то похожее на рисунок 1.1.

Если ты обнаружишь, что они не прочитали инструкции в начале книги...

... и из-за этого у тебя не получилось что-то сделать, то перелистни эту книгу на начало, подсунь им под нос введение, пока они читают утреннюю газету, и умоляюще посмотри на них. Иногда помогает говорить «пожалуйста-пожалуйста-пожалуйста» до тех пор, пока они не встанут и не сделают всё, что надо. Ну и конечно, можно попробовать сделать всё самостоятельно, это может оказаться даже проще.

1.3. Первая программа на Питоне

Так или иначе, если ты добрался досюда, у тебя уже открыта *консоль*, или командная строка Питона — это один из способов запускать команды и целые программы на Питоне. После запуска консоли или ввода любой команды ты увидишь *приглашение командной строки*, которое в Питоне выглядит вот так:

```
>>>
```

Если записать несколько команд на Питоне одну за другой, получится программа, которую можно запускать и не через консоль, но пока на минутку остановимся на простых командах, которые можно вводить прямиком в командную строку (после «приглашения»). Например, можно ввести туда следующую команду:

```
print("Всем привет!")
```

Чтобы всё получилось, нужно ввести и скобки и кавычки (вот эти: "") так, как написано выше. Тогда на экране должно появиться что-то вроде такого:

```
>>> print("Всем привет!")
Всем привет!
```

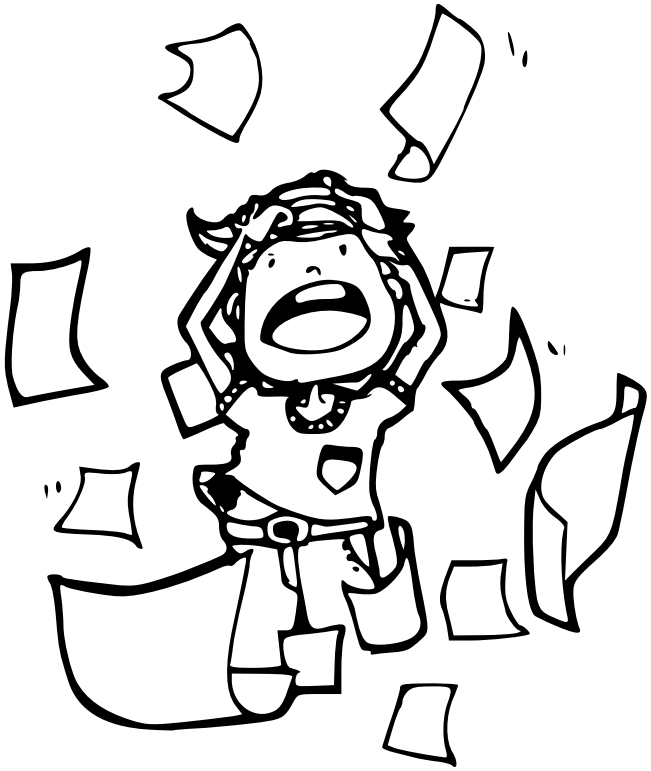
После этого приглашение командной строки появится снова, чтобы показать, что Питон готов принимать новые команды. Поздравляю! Ты только что создал и запустил свою первую программу на Питоне — пусть пока и всего из одной команды: print — функции, которая просто печатает всё, что написано в скобках. Потом мы много будет использовать эту команду.

1.4. Вторая программа на питоне... опять то же самое?

Программы на Питоне были бы не слишком полезными, если бы их приходилось каждый раз вводить заново в командную строку или если бы ты написал программу для кого-то, а ему бы пришлось её перепечатывать, чтобы запустить.

Программа для редактирования текстов (Microsoft Word, Libreoffice Writer или другая подобная), которую ты, вероятно, используешь для выполнения каких-нибудь домашних заданий, получена из исходного кода размером примерно от 10 до 100 миллионов строк. Если печатать это на бумаге с двух сторон не очень крупно, это может занять, например, 400 000 страниц. Это стопка бумаги высотой 40 метров, с десятиэтажный дом. Такое количество бумаги нести из магазина в дом, чтобы перепечатать, пришлось бы долго... очень долго...

...а если бы ещё и ветер подул в подходящий момент... за бумагой пришлось бы долго бегать. Так вот, хорошая новость: всем этим заниматься не обязательно.



Открой текстовый редактор (можешь опять спросить у родителей, как он называется: например, kate, gedit, kdevelop, но никак не Microsoft Word, он не подойдёт) и напиши туда точно ту же самую команду, что ты до этого вводил в консоль:

```
print("Всем привет!")
```

Теперь сохрани этот файл в своей домашней папке. Наверху в программе должен быть значок сохранения, а когда тебя спросят, куда сохранять, нажми на какую-нибудь кнопку типа домика. В качестве имени файла введи «hello.py». Теперь опять открой терминал и напиши:

```
python hello.py
```

В консоли должно появиться приветствие от программы, точно так же, как в прошлый раз (примерно как на рисунке 1.2).

Вот. Теперь ты видишь, что мудрые люди, создавшие Питон, спасли тебя от ввода одних и тех же программ много-много-много раз для выполнения одних и тех же действий. Как они делали в 1980х. Я серьёзно, им приходилось вводить каждый раз кучу команд для выполнения одной и той же программы. Можешь спросить у папы — вдруг у него был ZX81 в молодости — так там приходилось так делать. Теперь можно просто написать имя программы, и она целиком исполнится от начала до конца.

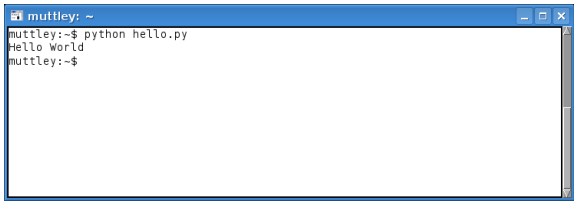


Рис. 1.2: Запуск программы на Питоне, сохранённой в текстовый файл

Конец начала

Добро пожаловать в удивительный мир программирования! Мы начали с простой программы, которая печатает «Всем привет» («Hello world») — все с этого начинают, когда учатся программировать. В следующей главе мы займёмся чуть более полезными вещами в консоли Питона, а потом изучим, как написать программу посложнее.

Глава 2

8 умножить на 3.57 равняется...

Чему равно 8 умножить на 3.57? Пришлось бы использовать калькулятор, чтобы посчитать? Ладно, этот пример можно вычислить и в уме, но не в том дело. То же самое можно сделать в консоли Питона. Запусти её опять (как описано в предыдущей главе), если она ещё не запущена, и введи туда такую команду: `8*3.57`. Потом нажми Enter.

```
Python 3.4.0 (default, Apr 11 2014, 13:05:11)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> 8 * 3.57
28.56
```

Звёздочка (*) (обычно это shift + 8) используется для умножения вместо привычного символа × (или ·), потому что не везде их можно просто так ввести с клавиатуры, а буква X путалась бы с собственно буквой, если бы её использовали как знак умножения. Ладно, как насчёт чего-нибудь более полезного?

Представь, что тебе приходится заниматься делами по хозяйству раз в неделю, за что каждый раз ты получаешь по 5 рублей и ещё у тебя есть курьерская подработка по доставке газет за 30 рублей в неделю. Сколько денег такими темпами накопится за год?

Если бы мы решали эту задачу на бумаге, то написали бы что-то вроде:

```
(5 + 30) × 52
```

Что значит: (5 руб + 30 руб), умноженное на 52 недели в году. Можно, конечно, и сразу сократить эту запись до такой:

```
35 × 52
```

И это уже совсем просто посчитать что на калькуляторе, что в столбик (в уме сложнее). А можно всё то же самое сделать в консоли:

Питон сломался!?!?

Если ты возьмёшь калькулятор и введёшь туда `8×3.57`, ответ на экране будет такой:

28.56

В Питоне ответ может быть такой, а может быть такой:

28.55999999

Чем Питон отличается? Уж не поломан ли он??
Да нет, вообще-то. Просто так дробные числа (числа с десятичной запятой, или *числа с плавающей запятой*) представляются в компьютере: приближённо. Результаты всегда почти точные, далеко справа после точки могут набегать небольшие ошибки вычисления после выполнения последовательности действий. Дробные числа представляются в компьютере не очень просто, мы не будем сейчас на этом останавливаться. Я хочу сказать, не удивляйся, что *иногда* результаты вычислений не в точности равны тому, что ты ожидаешь, это верно для умножения, деления, сложения и вычитания.
Целые числа в компьютере представляются точно и всегда вычисляются без ошибок в Питоне (в других языках программирования есть свои хитрости).

```
>>> (5 + 30) * 52
1820
>>> 35 * 52
1820
```

Как быть, если ты тратишь 10 рублей в неделю? Что теперь останется к концу года? Можно было бы разными способами записать это на бумаге, но давай опять обратимся к консоли:

```
>>> (5 + 30 - 10) * 52
1300
```

Это значит: 5 рублей и ещё 30 рублей минус 10 потраченных рублей, и всё умножается на 52 недели в году. В конце года будет 1300 накопленных рублей. Ладно, я понимаю, что выглядит это всё не слишком полезно, и без Питона

тут было бы легко обойтись. Мы потом опять вспомним про этот пример, и я покажу, как сделать из него куда более полезный.

В питоньей консоли можно умножать, складывать, вычитать и делить. Можно выполнять и другие арифметические действия, но мы пока их пропустим. Вот так выглядят символы математических операций:

$+$	Сложение
$-$	Вычитание
$*$	Умножение
$/$	Деление

Для деления используется косая черта («прямой слэш» или просто «слэш», как его ещё называют), потому что рисовать дроби с клавиатуры не так-то просто, а символа деления ($\div$) обычно тоже нет.

Если бы тебя попросили сказать, сколько коробок, в которые влезает по 20 яиц, нужно для 100 яиц, ты мог бы написать что-то такое:

$$\frac{100}{20}$$

или такое:

$$\begin{array}{r|l} 100 & 20 \\ \hline 100 & 5 \\ \hline 0 & \end{array}$$

ну или, может, такое (хотя так чаще за рубежом пишут)

$$100 \div 20$$

Так вот, в Питоне надо написать вот так: `100 / 20`.

Что, на мой взгляд, гораздо проще. Впрочем, что я в этом понимаю, я всего лишь книга.

2.1. Использование скобок и «приоритет операций»

В языках программирования (да и просто в математике) мы используем скобки, чтобы управлять тем, что называется «приоритет операций». Операции не ограничиваются четырьмя арифметическими, бывают и другие, но смысл одинаковый. Из этих четырёх операций умножение и деление, как обычно, выполняются перед сложением и вычитанием, то есть у них выше приоритет. В примере ниже у всех операций одинаковый приоритет, поэтому они все выполняются слева направо:

```
>>> print(5 + 30 + 20)
55
```

В примере ниже тоже у всех одинаковый приоритет, и тоже все операции выполняются по порядку слева направо:

```
>>> print(5 + 30 - 20)
15
```

А вот в следующем примере есть умножение. Тут сначала умножаются числа 20 и 30, а потом к ним прибавляется 5: у умножения выше приоритет.

```
>>> print(5 + 30 * 20)
605
```

Что будет, если добавить скобок? Вот что:

```
>>> print((5 + 30) * 20)
700
```

Почему поменялся результат? Потому что скобки меняют порядок операций. Сначала вычисляется то, что в скобках, а потом всё, что снаружи. Тут сначала к 5 прибавляется 30, а результат всего Питон умножает на 20.

Можно использовать скобки более сложным образом. Можно внутри скобок писать ещё скобки:

```
>>> print(((5 + 30) * 20) / 10)
70
```

Тут Питон сначала вычисляет то, что внутри самых вложенных скобок, потом то, что во внешних скобках, и потом выполняет операцию деления, которая вообще не в скобках (самые внешние скобки нужны слову print, а не чтобы указывать порядок операций). Это выражение можно прочитать так: «прибавь к 5 30, потом это всё умножь на 20 и результат подели на 10». Без скобок всё было бы иначе:

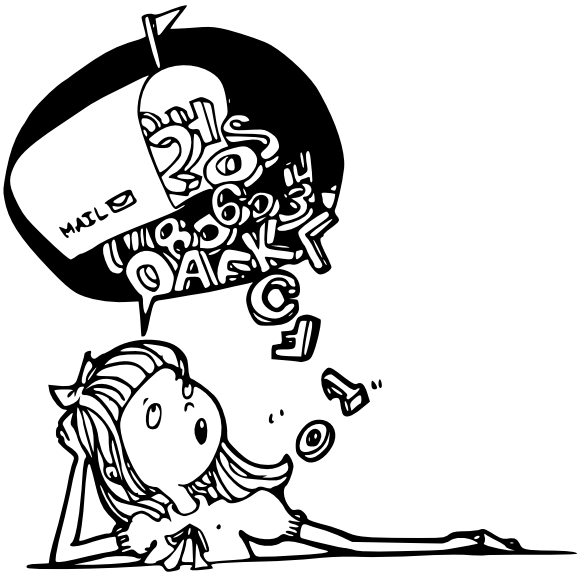
```
>>> 5 + 30 * 20 / 10
65
```

Тут сначала 30 умножается на 20, потом делится на 10 и к этому всему прибавляется 5.

Помни, что умножение и деление всегда выполняются перед сложением и вычитанием, если только скобки не указывают делать иначе.

2.2. Нет ничего столь же непостоянного, как переменная

«Переменная» в программировании — название места для хранения разных штук. «Разные штуки» могут быть числами, текстом, списками чисел или строк и множеством других вещей, о которых обо всех сейчас пока рассказать сложно. Давай пока считать, что переменная — это что-то вроде почтового ящика.



В почтовый ящик можно положить что-нибудь — письмо, газету — и в переменную можно точно так же положить что-нибудь — число, строку, список чисел. Именно таким образом работают переменные в большинстве языков программирования: в них что-нибудь кладут. Правда, не во всех языках так.

В Питоне переменные используются немножко иначе. Они не хранят что-то внутри себя, а, скорее работают наклейкой на почтовом ящике. Эту наклейку можно снять и переклеить на другой ящик, например. А можно прилепить сразу на несколько каких-нибудь коробок (обмотав их предварительно скотчем). В питоне эта операция прилепливания выглядит как знак равенства. Чтобы создать переменную, нужно написать её имя слева от этого знака, а справа — то, на что эта переменная указывает. Например, вот так:

```
>>> Фёдор = 100
```

Мы сейчас создали переменную, называющуюся `Фёдор`, и сказали, что она указывает на число 100. Тут мы ещё говорим Питону запомнить это число, потому что собираемся его потом использовать. Чтобы узнать, на что переменная указывает, можно вывести её значение функцией `print`, как-то так:

```
>>> Фёдор = 100
>>> print(Фёдор)
100
```

Можно сказать Питону, что `Фёдор` теперь должен указывать на другое число:

```
>>> Фёдор = 200
>>> print(Фёдор)
200
```

Теперь `Фёдор` указывает на число 200 (а число 100, которое больше не нужно, Питон потом забудет, когда оно будет мешать). И `print` подтвердил нам, что Фёдор теперь указывает на 200. Можно наклеить на это же самое 200 и ещё одну наклейку, то есть создать ещё одну переменную, указывающую на него:

```
>>> Фёдор = 200
>>> Василий = Фёдор
>>> print(Василий)
200
```

В этом кусочке кода мы создаём переменную `Василий`, которая указывает туда же, куда и `Фёдор`. Если Фёдор поменяет своё значение, Василия это не коснётся.

Вообще, конечно, «Фёдор» — не самое подходящее имя для переменной. Обычно имена переменных записывают английскими буквами — так программистам со всего мира гораздо проще читать и дополнять программы друг друга. Ну и кроме того, имя переменной должно бы говорить, зачем она нужна. Вот с почтовым ящиком всё понятно, он нужен, чтобы разную почту хранить. А переменная может хранить совершенно разные вещи, совсем по-разному устроенные, и переменных в программе обычно очень много. Поэтому называть переменную нужно так, чтобы сразу было понятно, для чего она используется[1].

Представь себе, что ты открыл консоль Питона, напечатал `Фёдор = 200`, а потом отошёл от компьютера. На 10 лет. Чтобы залезть на Эверест, потом пересечь пустыню Сахару, попрыгать с моста на резинке в Новой Зеландии и спуститься на лодках по реке Амазонке. И потом обратно вернулся к консоли. Самое сложное будет понять: почему Фёдор и почему 200?!

Обычно, чтобы всё забыть, хватает гораздо меньшего времени, чем 10 лет. А часто программу пишут сразу несколько людей, каждый из которых не знает мысли остальных. Так что лучше с самого начала называть все переменные понятно и не рассчитывать на свою память.

[1] Некоторые даже считают, что придумывать хорошие имена переменным — самое сложное в работе программиста.

Так-то! Смотри, например, мы можем назвать переменную так: `number_of_students`.

```
>>> number_of_students = 200
```

Так сделать получится, потому что имена переменных можно составлять из букв, цифр и знаков подчёркивания (главное, чтобы имя не начиналось с цифры). Если ты вернёшься к своему коду через 10 лет, то «number_of_students» всё ещё будет иметь смысл. Можно будет напечатать вот так:

```
>>> print(number_of_students)
200
```

И сразу будет понятно, что количество студентов (где-то) — 200. Не всегда есть смысл придумывать длинные осмысленные имена переменных: иногда переменная используется всего в паре строк, иногда можно использовать вообще одну букву в качестве имени, и есть такие случаи, когда всем это будет привычно и понятно. В основном, это зависит от того, захочется ли потом понять по имени переменной, зачем она вообще нужна, или это будет почему-нибудь понятно и без этого.

```
это_тоже_допустимое_имя_переменной_но_наверное_не_слишком_полезное
```

2.3. Используем переменные

Ладно, как создавать переменные мы разобрались. Но что с ними делать?

А вот что. Помнишь тот пример, который мы считали выше? Сколько денег заработается за год? Вот этот:

```
>>> print((5 + 30 - 10) * 52)
1300
```

Что если нам сюда добавить три переменных? Вот так:

```
>>> chores = 5
>>> paper_round = 30
>>> spending = 10
```

В переменных записано, сколько денег получается за неделю: 5 руб. за домашние дела и 30 — за курьерство — и сколько денег тратится в неделю: 10 рублей. Можно теперь перепечатать тот же пример вот так:

```
>>> print((chores + paper_round - spending) * 52)
1300
```

Ответ такой же. А что если теперь за домашние дела ты будешь получать семь рублей вместо пяти? Надо записать в переменную `chores` значение 7, а потом нажать пару раз кнопку вверх на клавиатуре, чтобы появился опять пример на экране, и Enter, вот так:

```
>>> chores = 7
>>> print((chores + paper_round - spending) * 52)
1404
```

Так пришлось гораздо меньше печатать, чтобы получить новый ответ, чем без использования переменных. Можно попробовать поменять другие переменные и опять нажимать клавишу вверх, чтобы получать новые ответы:

```
Что будет если тратить вдвое больше:
>>> spending = 20
>>> print((chores + paper_round - spending) * 52)
884
```

Если тратить 20 рублей в неделю, то к концу года получится накопить только 884 рубля. Пока переменные не сильно много пользы нам принесли, но уже понятно, что с ними проще и что они могут что-нибудь хранить.

Вспоминай почтовый ящик с наклейкой на нём!

2.4. Кусочек строки

Если ты уделяешь внимание моим словам, а не просто бегло проглядываешь книжку в поисках интересных картинок, то можешь вспомнить, что я говорила, что в переменных можно хранить разного сорта вещи, а не только числа. Например, можно хранить текст, или, как его обычно называют в программировании, строки (по-английски: *string*). Такое название может поначалу показаться странным, но, если подумать, текст — это просто буквы, составленные в слова, составленные в строки, и ничего больше. Может быть, так понятнее.

А может, и не понятнее...

В таком случае, всё, что тебе необходимо знать, — это что строка — не более чем составленные вместе буквы, цифры и любые другие символы. Ну и обычно они что-то значат все вместе. Всё содержимое этой книги можно записать в строку. Твоё имя можно записать строкой. Твой домашний адрес

тоже. Мы, кстати, уже использовали строку в первой программе, вот эту: "Всем привет!".

В Питоне строка создаётся просто: нужно поставить кавычки (такие: " или такие: ') вокруг текста. Можно вспомнить про нашу бесполезную переменную Фёдора и записать в неё строку, вот так:

```
>>> Фёдор = "я маленькая строка"
```

И как всегда, можно узнать, что хранит Фёдор:

```
>>> print(Фёдор)
я маленькая строка
```

Или вот пример с одинарными кавычками:

```
>>> Фёдор = 'и я строка'
>>> print(Фёдор)
и я строка
```

Но если ты попытаешься сохранить в переменную сразу несколько строк текста[1], используя просто кавычки, то ничего не выйдет. Например, попробуй напечатать команду, написанную ниже, и получишь пугающе непонятное сообщение об ошибке, что-то вроде такого:

```
>>> Фёдор = "первая строка
  File "<stdin>", line 1
    Фёдор = "первая строка
                           ^
SyntaxError: EOL while scanning string literal
```

Про эту ошибку мы поговорим позже, а пока можешь просто запомнить, что чтобы сохранить в переменную сразу несколько строк текста, нужно использовать три одинарных кавычки[2]:

```
>>> Фёдор = '''вот строка
... с переносом строки внутри'''
```

Напечатай эту переменную, чтобы проверить, что всё получилось:

[1] По-английски это называется *multilne string*, то есть «многострочная строка». Строка с переносами строк внутри, иначе говоря.

[2] Обычно одинарную кавычку можно ввести только в английской раскладке и она находится на той же клавише, где русская «Э», справа→ на клавиатуре.

```
>>> print (Фёдор)
вот строка
с переносом строки внутри
```

Пока ты печатал эту строчку с переносом, то мог обратить внимание, что Питон вместо приглашения на второй строке напечатал три точки (...). Он это делает в тех случаях, когда что-то, начатое на первой строке, продолжается на следующей — вот как строка. Пока кавычки не закроются, Питон так и будет печатать «...», показывая, что ждёт завершения строки. Потом мы и в других случаях увидим эти три точки.

2.5. Развлечения со строками

Вот есть интересный вопрос: чему равно 10 * 5 (10 умножить на 5)? Ответ, конечно: 50.

Ладно, это был неинтересный вопрос.

А чему равно 10 * 'a' (10 умножить на букву а)? Вопрос может показаться бессмысленным, но вот ответ от Питона:

```
>>> print(10 * 'a')
aaaaaaaaaa
```

Это работает не только с отдельными символами, но и со строками подлиннее:

```
>>> print(20 * 'abcd')
abcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcdabcd
```

Есть ещё один трюк со строками. В них можно вставлять значения. Для этого в строку нужно добавить «%s» как маркер, куда вставлять значение. Проще всего это показать на примере:

```
>>> mytext = 'Мне %s лет'
>>> print(mytext % 12)
Мне 12 лет
```

В первой строке создаётся переменная `mytext`, содержащая некоторые слова и шаблон подстановки (в английских инструкциях он называется *placeholder*) `%s`. Этот шаблон — это такая инструкция Питону: «замени меня чем-нибудь». На следующей строке, когда мы печатаем значение переменной `mytext`, мы используем символ `%`, чтобы указать Питону, на что именно заменять шаблон подстановки: на число 12. Можно использовать одну и ту же строку и подставлять внутрь неё разные значения:

```
>>> mytext = 'Привет, %s, как твои дела?'
>>> name1 = 'Иннокентий'
>>> name2 = 'Саша'
>>> print(mytext % name1)
Привет, Иннокентий, как твои дела?
>>> print(mytext % name2)
Привет, Саша, как твои дела?
```

В примере выше создаются три переменных (`mytext`, `name1` и `name2`), первая из которых содержит маркер «`%s`». Потом мы печатаем значение этой строки, подставляя внуть неё значения строк `name1` и `name2`, используя операцию «`%`».

Можно вставить в строку больше, чем один шаблон для подстановки, вот так:

```
>>> mytext = 'Привет, %s и %s, как обстоят ваши дела?'
>>> print(mytext % (name1, name2))
Привет, Иннокентий и Саша, как обстоят ваши дела?
```

Когда используется несколько маркеров, нужно подставляемые переменные окружать скобками, как в примере выше. Несколько переменных, окружённых скобками, называются *кортеж* (по английски — *tuple*). Это слово потом ещё будет использоваться.

2.6. Не совсем список для покупок

Яйца, молоко, мыр, сельдерей, масло и сода. Не вполне полный список для покупок, но сойдёт для наших целей. Если бы тебе захотелось сохранить это в переменной, можно было бы это сделать так, в строку:

```
>>> shopping_list = 'яйца, молоко, сыр, сельдерей, масло, сода'
>>> print(shopping_list)
яйца, молоко, сыр, сельдерей, масло, сода
```

Другой способ — создать «список», особый сорт объектов в Питоне:

```
>>> shopping_list = [ 'яйца', 'молоко', 'сыр', 'сельдерей', 'масло', 'сода' ]
>>> print(shopping_list)
[ 'яйца', 'молоко', 'сыр', 'сельдерей', 'масло', 'сода' ]
```

Чтобы сделать список, приходится больше печатать (все эти кавычки), но результат сильно полезнее строки. Например, мы можем просто напечатать третий элемент списка, указав его *индекс в списке* (позицию) в квадратных скобках:

```
>>> print(shopping_list[2])
сыр
```

Да, здесь нет опечатки. Мы напечатали третий элемент, хотя в квадратных скобках стоит цифра 2. Всё потому, что номера элементов в списке считаются с нуля. Это достаточно непривычно и кажется бессмысленным большинству людей — но не программистам[1]. Скоро ты тоже привыкнешь считать с нуля вместо одного, чем тоже будешь удивлять многих людей.

Можно ещё напечатать все элементы с третьего по пятый, используя двоеточие внутри квадратных скобок:

```
>>> print(shopping_list[2:5])
['сыр', 'сельдерей', 'масло']
```

`[2:5]` — значит: нужны все элементы с индексами от 2 до 5 (не включая 5). Поскольку индексы считаются с нуля, то получатся как раз третий, четвёртый и пятый элементы, а шестой (который с индексом 5) не будет включён в результат.

В списках можно хранить (как и в переменных) что угодно. Можно хранить числа:

```
>>> mylist = [ 1, 2, 5, 10, 20 ]
```

...и строки:

```
>>> mylist = [ 'a', 'bbb', 'ccccccc', 'ddddddddd' ]
```

...и всё вместе:

```
>>> mylist = [1, 2, 'a', 'bbb']
>>> print(mylist)
[1, 2, 'a', 'bbb']
```

...и другие списки:

```
>>> list1 = [ 'a', 'b', 'c' ]
>>> list2 = [ 1, 2, 3 ]
>>> mylist = [ list1, list2 ]
>>> print(mylist)
[['a', 'b', 'c'], [1, 2, 3]]
```

В примере выше создаётся перменная `list1`, содержащая список из трёх букв, и `list2`, содержащая список из трёх цифр; а потом создаётся переменная `mylist`, которая содержит оба эти списка. Дальше можно сходить с ума,

[1] Индекс элемента в списке просто показывает, сколько элементов надо пропустить от начала списка, чтобы добраться до нужного. Индекс 0 значит, что пропускать не надо ни одного, индекс 2 — что надо пропустить 2 элемента, и получить как раз третий.

создавая списки списков списков списков... но обычно никому не нужно делать такие сложные вещи в Питоне. С другой стороны, полезно знать, что так вообще можно и что список может хранить всё что угодно.

А не только какие продукты надо купить.

Изменяем элементы

Можно изменить элемент списка, просто присвоив ему новое значение, как будто это обычная переменная. Можно, например, заменить сельдерей алатом:

```
>>> shopping_list[3] = 'салат'
>>> print(shopping_list)
['яйца', 'молоко', 'сыр', 'салат', 'масло', 'сода']
```

Добавляем ещё элементы...

Можно добавить к списку новых элементов, вызвав функцию `append`. Подробно про функции (и как делать свои) мы поговорим позже, но пока можно воспользоваться готовой вот так:

```
>>> shopping_list.append('шоколадка')
>>> print(shopping_list)
['яйца', 'молоко', 'сыр', 'салат', 'масло', 'сода', 'шоколадка']
```

И этот список, несомненно, гораздо лучше оригинальноrо.

...и удаляем элементы

Можно удалить из списка элементы, используя функцию `remove`. Например, чтобы удалить соду из списка, надо написать вот так:

```
>>> shopping_list.remove ('сода')
>>> print shopping_list
['яйца', 'молоко', 'сыр', 'салат', 'масло', 'шоколадка']
```

Ещё можно удалить элемент списка по ему индексу (как, кстати, и любую другую переменную). Для этого есть команда `del` (сокращение от слова `delete`, удалить), которая используется вот как:

```
>>> del shopping_list[0]
>>> print(shopping_list)
['молоко', 'сыр', 'салат', 'масло', 'шоколадка']
```

Теперь из списка исчез первый элемент — тот, который с индексом 0.

Два списка лучше одного!

Можно объединить два списка, сложив их, как будто мы складываем числа:

```
>>> list1 = [ 1, 2, 3 ]
>>> list2 = [ 4, 5, 6 ]
>>> print(list1 + list2)
[1, 2, 3, 4, 5, 6]
```

Можно записать результат сложения списков в новую переменную:

```
>>> list1 = [ 1, 2, 3 ]
>>> list2 = [ 4, 5, 6 ]
>>> list3 = list1 + list2
>>> print(list3)
[1, 2, 3, 4, 5, 6]
```

Кстати, этот метод можно использовать, чтобы добавлять в список новые значения вместо использования функции `append`. Вот, например, так:

```
>>> list1 = [ 1, 2, 3 ]
>>> list1 = list1 + [ 4 ]
>>> print (list1)
[1, 2, 3, 4]
>>> list1 = list1 + [5, 6, 7]
>>> print (list1)
[1, 2, 3, 4, 5, 6, 7]
```

Ещё список можно умножать примерно так же, как мы до этого умножали строку:

```
>>> list1 = [ 1, 2 ]
>>> print(list1 * 5)
[1, 2, 1, 2, 1, 2, 1, 2, 1, 2]
```

В примере выше умножение списка на 5 — это инструкция Питону «повтори список 5 раз.

Деление и вычитание для списка не имеют смысла, поэтому если попытаться так сделать, получится только ошибка:

```
>>> list1 / 20
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for /: 'list' and 'int'
```

или вот:

```
>>> list1 - 20
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for -: 'type' and 'int'
```

И попробуй угадай, что Питон хочет сказать этими сообщениями об ошибке...

2.7. Кортежи и списки

Кортеж (про который мы говорили раньше) вроде как список, но чтобы сделать его, нужно использовать круглые скобки вместо квадратных. Пользоваться кортежами можно примерно так же, как списками:

```
>>> t = (1, 2, 3)
>>> print(t[1])
2
```

Основное отличие от кортежей списков — это что кортежи нельзя изменять. Если попытаться так сделать, то просто получится ещё одно сообщение об ошибке:

```
>>> t[0] = 4
Traceback (most recent call last):
  File "<stdin>", line 1, in ?
TypeError: 'tuple' object does not support item assignment
```

Это, конечно, не значит, что если переменная хранила кортеж, то в неё нельзя записать ничего больше. Нельзя именно поменять тот кортеж, на который она указывает, но можно вместо него записать новый.

Например, вот этот код будет работать без проблем:

```
>>> myvar = (1, 2, 3)
>>> myvar = [ 'список', 'строк' ]
```

Тут мы вначале создаём переменную `myvar`, указывающую на кортеж из трёх чисел. Потом говорим Питону, что теперь эта переменная указывает на список строк. Это может показаться немного странным — зачем вообще нужны кортежи и как именно они отличаются от списков. Тут можно представить шкафчики для вещей в бассейне, только с прозрачной дверцей. Можно положить вещи в шкафчик и выбросить ключ. Тогда нельзя будет поменять, что именно лежит в шкафчике (хотя посмотреть на них всё-таки получится), и

шкафчик будет кортежем. А можно не выкидывать ключ, тогда шкафчик будет как список: всегда можно будет что-то туда положить или вынуть. На все шкафчики можно приклеивать и отлеплять метки — это наши переменные.

Кортежи иногда бывают необходимы, когда списки не подходят именно из-за того, что списки можно менять. Мы потом столкнёмся с такими ситуациями.

2.8. Ещё раз вкратце

В этой главе мы увидели, как вычислять простые математические примеры в консоли Питона. Потом мы ещё обсудили, как использовать скобки для изменения результатов вычисления, чтобы менять приоритеты операций. Ещё мы научились говорить Питону запоминать значения для дальнейшего использования — при помощи переменных — и узнали, что Питон использует «строки» для хранения текста, а ещё списки и кортежи, в которых можно хранить несколько разных значений сразу.

Упражнение 1

Создай список своих любимых игрушек `toys` и список своей любимой еды `foods`. Объедини списки и сохрани результат в переменную `favourites`, а затем напечатай на экран её значение.

Упражнение 2

Если у тебя есть 3 коробки по 25 шоколадок и ещё 10 коробок по 32 конфеты, сколько у тебя всего сладостей в штуках? Как можно добавить сюда переменных и быстро узнать, сколько будет сладостей, если в каждую из коробок с конфетами положить по 35 конфет вместо 32?

Упражнение 3

Запиши в переменные свои имя и фамилию. Потом создай строку с шаблонами подстановки, подставь в неё имя и фамилию (операцией %) и выведи результат на экран.

Глава 3

Черепахи и другие медленные создания

Есть нечто общее между черепахами из реального мира и черепахой в Питоне. В реальном мире черепаха — это (иногда) зелёная рептилия, которая движется очень медленно и носит на себе свой дом. В мире Питона всё почти так же: черепаха — это маленькая чёрная стрелочка, которая очень медленно ползает по экрану. Правда, с домиком у этой стрелочки не сложилось.

Вообще, черепашка в Питоне оставляет за собой след, так что она похожа больше на улитку, чем на черепаху. Но «черепаха» звучит более гордо, так что модуль в Питоне называется именно так. Можно представлять себе черепаху, зажавшую в зубах маркер и рисующую, пока ползёт.

Давным-давно, в тёмные старые времена люди придумали язык программирования Лого (Logo). Это был язык для управления роботом-черепашкой Ирвингом. Со временем черепашка превратилась из робота, ползающего по полу, в маленькую стрелочку, перемещающуюся по экрану.

Что, кстати, показывает нам, что с развитием технологии не всегда всё улучшается. Маленькая робот-черепашка на полу была бы куда как забавнее.

В Питоне есть модуль[1] `turtle` (черепаха), и он в целом похож на язык Лого. Только Лого ничего, кроме черепашки, и не умеет, а Питон умеет ещё много всего другого. Модуль `turtle` полезен, чтобы понять, как компьютер рисует изображение на экране.

Ладно, давай теперь просто посмотрим, как же этот модуль работает. Во-первых, его надо «импортировать», то есть сказать Питону, что он нам нужен:

```
>>> import turtle
```

Потом нам надо создать «холст» для рисования — смысл тот же, что и в

[1] Про модули подробно поговорим мы чуть позже, а пока просто стоит иметь в виду, что модуль — это что-то, что можно использовать в своей программе, набор разных функций.

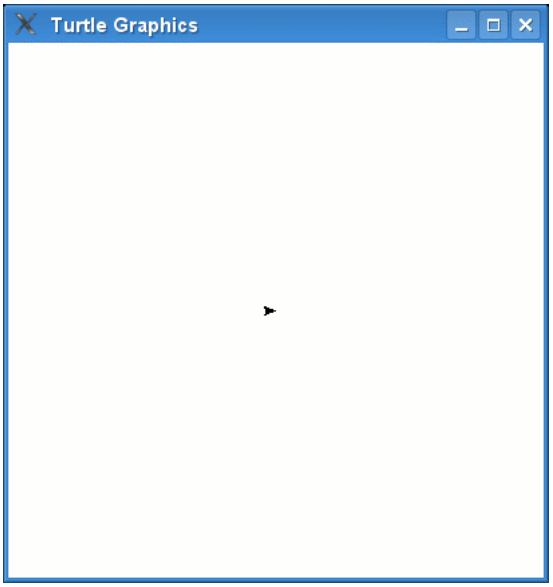


Рис. 3.1: Стрелочка, изображающая черепаху

реальном холсте, которым пользуются художники. Холст будет нужен, чтобы на нём рисовать. Мы создадим пустой:

```
>>> t = turtle.Pen()
```

Тут мы вызываем функцию `Pen` модуля `turtle`, и она автоматически создаёт холст (*canvas* по-английски). Вообще, функция — это что-то вроде маленькой программы, то есть кусочек кода, который можно использовать много раз (подробно функции мы обсудим потом). В данном случае функция `Pen` *возвращает* нам черепашку, то есть результат этой функции — черепашка, к которой мы приклеиваем переменную `t`. Результатом этого кода должна быть картинка наподобие 3.1.

Да, эта маленькая стрелочке посреди экрана — действительно черепаха. И да, на черепаху она внешне не похожа примерно ничем.

Этой черепахе можно отправлять инструкции, используя функции созданного объекта `t`. Вот, например, можно попросить черепаху продвинуться вперёд, туда, куда показывает стрелочка. Для этого есть функция `forward`, в скобках которой надо указать, на сколько точек на экране продвинуться. Например, чтобы подвинуть черепаху вперёд на 50 точек (и нарисовать линию длиной 50 точек), нужно выполнить такую команду:

```
>>> t.forward(50)
```

И в результате должно получиться как-то так: 3.2.

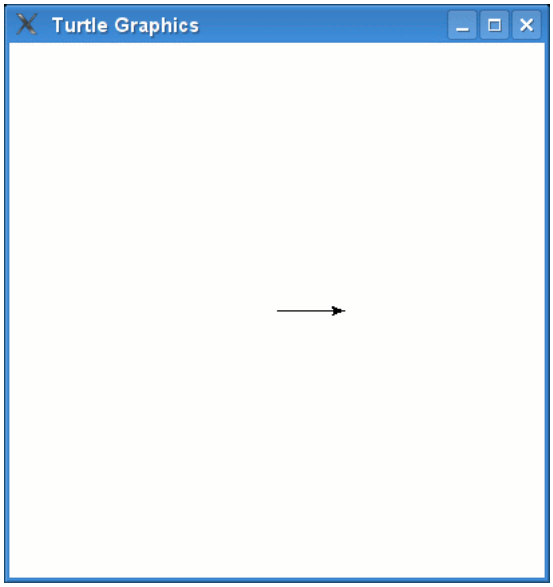


Рис. 3.2: Черепашка нарисовала линию.

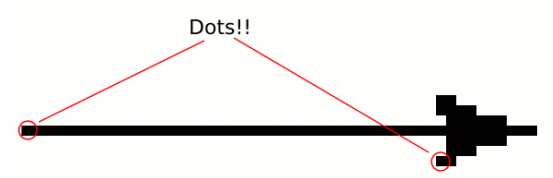


Рис. 3.3: Сильно увеличенные линия и стрелочка.

С точки зрения черепахи, она прошла вперёд 50 шагов. А мы бы сказали, что она прошла 50 точек по экрану.

И что это за точки такие?

Всё на экране компьютера состоит из отдельных маленьких точек, каждая из которых окрашена в свой цвет. Обычно их называют пикселями, чтобы не путать ни с какими другими точками, и так я и буду дальше делать. Все программы на компьютере, все игры заставляют точки на экране окрашиваться в разные нужные цвета. Эти отдельные точки можно увидеть, вооружившись лупой. Если сильно увеличить линию, нарисованную черепахой, то можно увидеть, что это много квадратных точек, оказавшихся рядом, как на картинке 3.3.

В следующих главах мы ещё вспомним про пиксели, они нам пригодятся.

Вернёмся к черепашке. Её ещё можно попросить повернуть направо и налево.

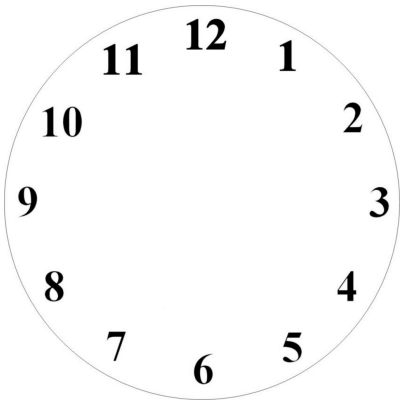


Рис. 3.4: Циферблат с отметками часов

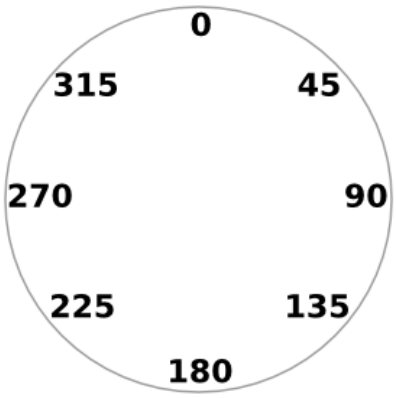


Рис. 3.5: Градусы.

```
>>> t.left(90)
```

Эта команда говорит черепашке повернуть налево на 90 градусов (то есть против часовой стрелки). Если ты ещё не знаешь про градусы и как ими меряют углы, то это можно представить себе вот как. На рисунке 3.4 есть циферблат от часов.

На циферблате по кругу написаны числа от 1 до 12 (или до 60, если там написаны минуты). Так вот градусы пишутся так же по кругу, только всё умножается на 30. Вместо цифры 3 — 90° (90 градусов), вместо шести — 180°, как на рисунке 3.5. А черепашка как будто стоит в центре циферблата и смотрит в сторону нуля, наверх.

И что происходит, когда мы отдаём команду `left(90)`?

Если встать, поднять руку ровно в сторону и показать туда, то чтобы повернуться лицом в ту сторону, в которую ты показываешь, нужно повернуться как раз на 90 градусов. Если показывать правой рукой, то на 90° вправо, левой рукой — на 90° влево. Так же и черепашка в питоне поворачивается туда, где

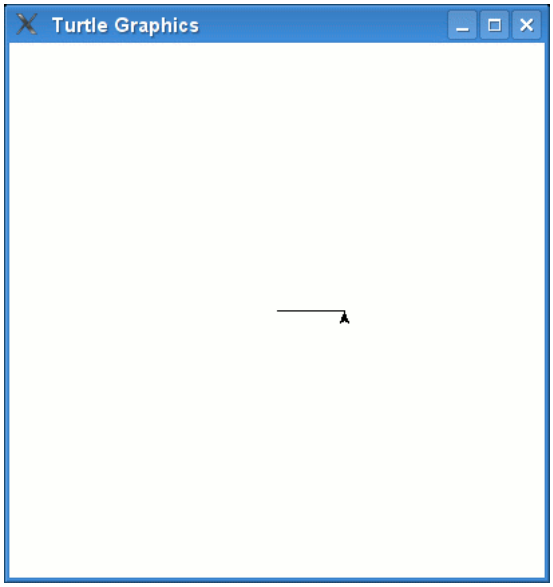


Рис. 3.6: Черепашка, повернувшая налево.

её правый бок или левый. При этом голова черепашки остаётся на месте (рисует она как раз маркером, зажатым в зубах), так что функция `t.left(90)` приводит к тому, что есть на рисунке 3.6. Черепашка ползла вправо и повернула вверх.

Давай теперь посмотрим, как все эти команды работают вместе:

```
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
>>> t.forward(50)
>>> t.left(90)
```

На экране черепашка нарисовала квадрат и остановилась, глядя в ту же сторону, что и в начале пути, как на рисунке 3.7.

Можно взять и очистить весь холст, воспользовавшись функцией `clear` (что как раз и переводится как «очистить»):

```
>>> t.clear()
```

Есть и другие полезные функции, применимые к черепашке. Например, `reset`: тоже очищает экран и ещё перемещает черепашку в начальное положение. Ещё есть функция `backward`, которая говорит черепашке двигаться назад (при этом направление её взгляда не меняется, она просто пятится). Функция `right` говорит черепашке повернуть направо; функция `up` говорит ей оторвать маркер от холста, то есть перестать рисовать при движении: не

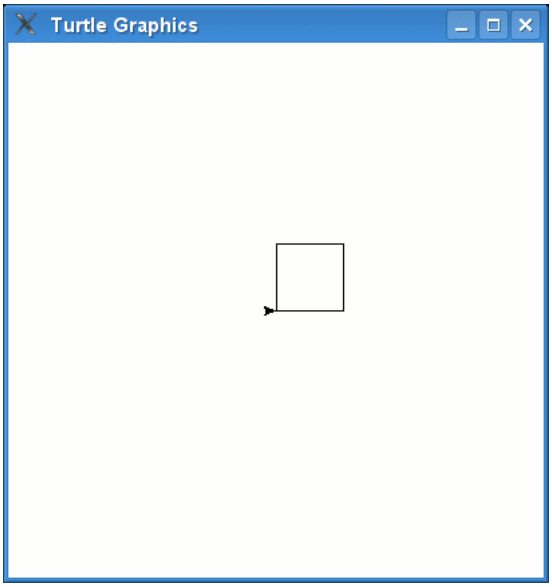


Рис. 3.7: Квадрат нарисовался.

всё можно нарисовать, если рисовать при каждом движении, иногда нужно просто переместиться. Есть и функция `down`, которая говорит ей обратно опустить маркер на холст и снова рисовать, пока она перемещается. Все эти функции вызываются таким же образом, как в примерах выше:

```
>>> t.reset()
>>> t.backward(100)
>>> t.right(90)
>>> t.up()
>>> t.down()
```

В следующих главах мы ещё воспользуемся услугами черепашки.

3.1. Глава закончилась

В этой главе мы познакомились с маленькой черепашкой, которая нарисовала нам немного линий, поворачиваясь направо и налево. Ещё мы обсудили градусы, которые здорово похожи на числа на циферблате часов.

Упражнение 1

Создай холст, используя функцию `Pen`, и нарисуй там прямоугольник.

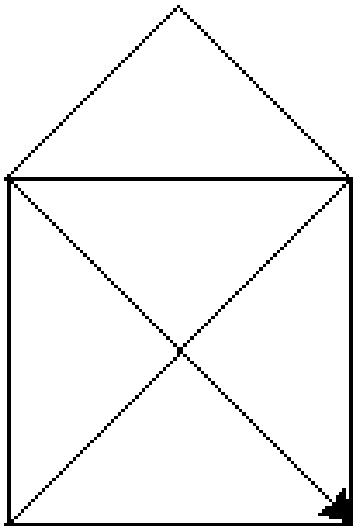


Рис. 3.8: Домик, который нарисовали, не отрывая карандаш от бумаги.

Упражнение 2

Создай холст, используя функцию `Pen`, и нарисуй там треугольник.

Упражнение 3*

Создай холст, используя функцию `Pen`, и нарисуй там домик, как на рисунке 3.8, не отрывая маркер от холста (не пользуясь функцией `up`). Это упражнение сложнее предыдущих и может не получиться без посторонней помощи, ничего страшного[1].

[1] Чтобы всё получилось, нужно иметь в виду, что может понадобиться повернуть на 45 или 135 градусов и что если стороны домика длиной 100 точек, то косые линии будут длиной примерно 71 и 141 точка.

Глава 4

Как задать вопрос

С точки зрения программистов, вопрос задаётся тогда, когда в зависимости от ответа нужно выполнить одни команды или другие. Во многих языках программирования такой вопрос записывается с использованием слова `if`, что на русский переводится как «если»[1]. Такое выражение ещё называется **условным оператором**.

Сколько тебе лет? Если тебе больше двадцати, ты супер стар!

Утверждение выше на Питоне может быть записано вот так:

```
if возраст > 20:
    print('ты супер стар!')
```

Условный оператор состоит из слова `if`, после которого записывается условие, завершаемое двоеточием (`:`). Все следующие строки, которые выполняются в зависимости от этого условия, должны начинаться с одинакового количества пробелов. Большинство людей использует тут 4 пробела, потому что так уже легко видеть *блок кода*, и при этом он не слишком далеко уезжает направо. Для вставки такого отступа в начало строки обычно используют клавишу `Tab`, она на клавиатуре слева (←) под цифрами, слева от буквы Й.

Если ответ на вопрос, который написан после `if`, — да, или `True`, как это записывается в Питоне, то блок кода, начинающийся с отступов, выполняется.

Условие, которое надо писать после `if`, — это выражение, на которое можно ответить «да» (`True`, «истина») или «нет» (`False`, «ложь»). Чтобы записывать условия, есть специальные значки, вот они:

[1] В некоторых языках прямо по-русски и пишут «если», но всё же обычно так не принято делать. В Питоне надо писать `if`.

==	равно
!=	не равно
>	больше чем
<	меньше чем
>=	больше чем или равно
<=	меньше чем или равно

Например, если тебе 10 лет, то условие `твой_возраст == 10` истинно (равно `True`). Если же тебе не 10 лет, то оно ложно (равно `False`). Тут есть хитрость: чтобы сравнить два числа, надо написать два знака равенства подряд. Если написать только один, будет ошибка. А один знак нужно писать, чтобы в переменную занести какое-нибудь значение, то есть никак не после `if`.

Теперь допустим, что тебе больше 10 лет и в переменной `age` хранится твой возраст. Тогда вот такое условие...

```
age > 10
```

...будет равно `True`. А если тебе меньше 10 лет, то это условие будет равно `False`. И если тебе 10 лет, условие тоже будет ложно, зато будет истинно условие `age>=10`.

Давай попробуем теперь ввести примеры в консоль:

```
>>> age = 10
>>> if age > 10:
...     print('я тут!')
```

Если ввести это в консоль, что произойдёт?..

Да ничего.

Переменная `age` не больше 10, так что `print` не выполнится. А как насчёт такого:

```
>>> age = 10
>>> if age >= 10:
...     print('тут я!')
```

Вот если этот пример запустить, то Питон выведет сообщение в консоль. И следующий пример тоже сработает:

```
>>> age = 10
>>> if age == 10:
...     print('вот я где!')
вот я где!
```

4.1. Сделай вот это... ИЛИ ВОТ ЭТО!

Можно расширить условный оператор и сказать Питону, что делать, когда условие ложно. Можно, например, напечатать в консоль «Привет», если тебе 12 лет или «Пока» в ином случае. Для этого пригодится слово `else` (в переводе — «иначе»).

```
>>> age = 12
>>> if age == 12:
...     print('Привет!')
... else:
...     print('Пока.')
Привет!
```

Если ты напечатаешь в консоль этот пример, то увидишь в ответ «Привет!». Стоит изменить значение переменной `age` на что-нибудь другое, как сообщение от Питона поменяется:

```
>>> age = 8
>>> if age == 12:
...     print('Привет!')
... else:
...     print('Пока.')
Пока.
```

4.2. Сделай вот это... или ещё вот это... ИЛИ ВОТ ЭТО!

Можно ещё дальше расширить условный оператор, используя слово `elif` (сокращение от «else if»). Например, можно вот так печатать, сколько тебе лет (да, не слишком полезно, но позволяет ухватить суть этого условного оператора):

```
 1. >>> age = 12
 2. >>> if age == 10:
 3. ...     print('похоже, тебе 10 лет')
 4. ... elif age == 11:
 5. ...     print('я знаю, тебе 11 лет')
 6. ... elif age == 12:
 7. ...     print('ух ты, а тебе 12 лет')
 8. ... elif age == 13:
 9. ...     print('тебе целых 13 лет!')
10. ... else:
11. ...     print('Столько люди не живут.')
12. ...
13. ух ты, а тебе 12 лет
```

В примере кода выше строка 2 проверяет, равно ли значение возраста 10. Если нет, то сразу после этого выполняется строчка 4, которая проверяет, равно ли значение возраста 11. Если нет — то проверяется условие в строке 6. Оно оказывается истинным, поэтому выполняется строка 7 и больше никаких проверок не производится.

4.3. Комбинируем условия

Можно проверять внутри одного условия сразу несколько выражений. Для этого используются английские слова «и»: `and` и «или»: `or`. Так например, пример выше можно было бы записать следующим образом, объединив проверки в одно большое условие:

```
1. >>> if age == 10 or age == 11 or age == 12 or age == 13:
2. ...     print('Я знаю, тебе %s лет' % age)
3. ... else:
4. ...     print('Столько люди не живут.')
```

Если любое из условий в строке 1 истинно, то все следующие и не проверяются и выполняется блок кода, следующий за `if`, то есть строка 2 в этом примере. Если же все условия ложны, то выполнится блок кода под `else`, то есть строчка 4. Этот пример можно ещё сократить, воспользовавшись операциями сравнения $<=$ и $>=$:

```
1. >>> if age >= 10 and age <= 13:
2. ...     print('Тебе %s лет' % age)
3. ... else:
4. ...     print('А сколько же?')
```

Тут если твой возраст не меньше 10 лет и не больше 13, то Питон напечатает, сколько тебе лет, а иначе — удивится.

4.4. Пустота

Есть ещё специальное значение, которое можно присвоить любой переменной, и о котором мы раньше не говорили: **ничего**.

Точно так же, как переменной можно присвоить числа, строки и списки, переменной можно присвоить и «ничего». В Питоне это записывается словом `None` и значит, что в переменной ничего нет (в других языках используют слова типа `nil`, `null`, `nullptr`). При этом значение этой переменной можно напечатать, как и значение любой другой, и это не вызовет ошибки, как было бы, если бы переменная вообще не была объявлена.

```
>>> myval = None
>>> print(myval)
None
```

Присвоить переменной `None` может быть нужно, чтобы указать, что переменная чему-то будет равна потом, но сейчас её значение неизвестно[1].

Вот пример: допустим, мы хотим сходить в кино втроём, и для этого нам надо скинуться деньгами, кому сколько не жалко. Когда все решат, сколько им не жалко, и положат сумму, например, в конверт, можно на эти деньги выбрать фильм и купить билеты (а чтобы показать все фильмы из ближайших кинотеатров, на которые хватит этих денег, вполне можно написать программу на Питоне). Так вот, для этого мы заведём три переменных для каждого из зрителей и запишем в них `None`, что будет значить, что человек ещё вообще не сдавал деньги (если бы мы записали туда 0, это бы значило, что человек не хочет и не будет сдавать денег):

```
>>> зритель1 = None
>>> зритель2 = None
>>> зритель3 = None
```

Теперь можно проверить, все ли сдали деньги, пользуясь `if`'ом:

```
>>> if зритель1 is None or зритель2 is None or зритель3 is None:
...     print('Надо подождать ещё, не все сдали деньги')
... else:
...     print('Мы собрали %s руб.' % (зритель1 + зритель2 + зритель3))
```

`if` проверяет, записано ли в какую-то переменную значение `None` и, если это так, сообщает об этом. Если же в каждую переменную записано, кто сколько сдал денег, то Питон напечатает нам общую собранную сумму.

Вот что будет, если только два человека решились:

[1] Если же хочется вообще удалить переменную, то надо не присвоить ей `None`, а написать вот так (для переменной `myval`): `del myval`.

```
>>> зритель1 = 100
>>> зритель2 = None
>>> зритель3 = 300
>>> if зритель1 is None or зритель2 is None or зритель3 is None:
...     print('Надо подождать ещё, не все сдали деньги')
... else:
...     print('Мы собрали %s руб.' % (зритель1 + зритель2 + зритель3))
Надо подождать ещё, не все сдали деньги
```

А вот, если все трое:

```
>>> зритель1 = 100
>>> зритель2 = 500
>>> зритель3 = 300
>>> if зритель1 is None or зритель2 is None or зритель3 is None:
...     print('Надо подождать ещё, не все сдали деньги')
... else:
...     print('Мы собрали %s руб.' % (зритель1 + зритель2 + зритель3))
Мы собрали 900 руб.
```

4.5. В чём разница...?

Какая разница между `10` и `'10'`?

Так вообще, кажется, что, кроме пары кавычек, её и нет. Хотя вот в предыдущих главах ты узнал, что `10` — это число, а `'10'` — строка. И это различие гораздо существеннее, чем можно подумать.

Недавно мы проверяли, чему равен возраст, вот так:

```
>>> if age == 10:
...     print('помни: тебе 10 лет')
```

И если в переменную `age` записать значение 10, то всё, что надо, на экране напечатается:

```
>>> age = 10
>>> if age == 10:
...     print('помни: тебе 10 лет')
...
помни: тебе 10 лет
```

Но если в ту же переменную записать `'10'` (с кавычками), то ничего печататься не будет:

```
>>> age = '10'
>>> if age == 10:
...     print('помни: тебе 10 лет')
...
```

Как же так? Почему теперь ничего не работает? Ну потому что строка — это не число, хотя и выглядят они одинаково:

```
>>> age1 = 10
>>> age2 = '10'
>>> print(age1)
10
>>> print(age2)
10
```

Вот, видишь! Выглядят совсем одинаково, если напечатать. Но число никогда не будет равно строке.

Это вроде как странно, но смысл какой-то такой, что если сравнивать 10 книг и 10 кирпичей, они никогда не будут равны — они просто разные. То есть можно сравнить 10 штук книг и 10 штук кирпичей — количество (число) одинаковое, но сказать, что 10 кирпичей — это одно и то же («равно»), что и 10 книг, вряд ли получится. Вот так и тут.

Но это не страшно, Питон умеет прочитать строку и понять, какое число там записано, и наоборот, записать цифрами число в строку. Вот так можно превратить строку `'10'` в число `10`:

```
>>> age = '10'
>>> converted_age = int(age)
```

Теперь переменная `converted_age` хранит число 10 (не строку). Функция `int`, которая используется для такого преобразования, названа как сокращение от английского слова «integer», что значит «целое число», число без дробной части, без запятой.

Чтобы обратно перевести число в строку, есть функция `str` (сокращение от «string», «строка»):

```
>>> age = 10
>>> converted_age = str(age)
```

Теперь в переменной `converted_age` лежит строка `'10'`. Самое время вернуться к тому сравнению, которое у нас не работало:

```
>>> age = '10'
>>> if age == 10:
...     print('Тебе %s лет' % age)
...
```

Если мы преобразуем переменную перед проверкой, тогда мы получим другой результат:

```
>>> age = '10'
>>> converted_age = int(age)
>>> if converted_age == 10:
...     print('Тебе %s лет' % age)
...
Тебе 10 лет
```

Или даже прямо так, короче и без дополнительной переменной:

```
>>> age = '10'
>>> if int(age) == 10:
...     print('Тебе %s лет' % age)
...
Тебе 10 лет
```

Предметный указатель