

Создай
текстовый
квест

Пошаговые
инструкции

```
it user_reply == "yes":  
    print("Beep boop!")
```

ПРОГРАММИРОВАНИЕ для детей на языке PYTHON



ПРОГРАММИСТ-
ЧЕМПИОН

Создай свою
компьютерную
игру!

БУМ!

Нарисуй
снежинку

Простое программирование
для начинающих

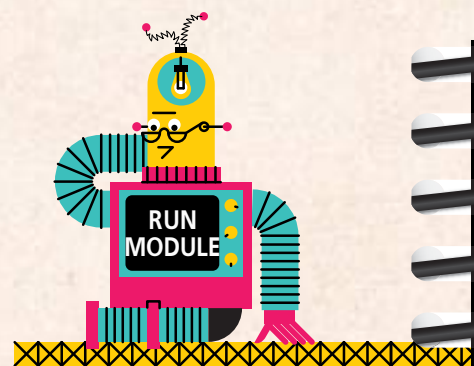
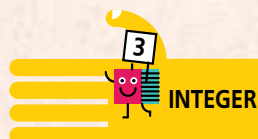
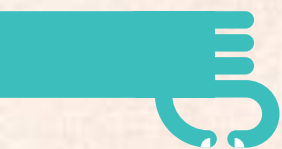
ПРОГРАММИРОВАНИЕ для детей на языке PYTHON



Создай свою
компьютерную
игру!

Аванта





Содержание

Что такое программирование?	4
Начинаем с языка Python	6
Играем с числами	9
Переменные	10
Принимаем решения	16
Строим блок-схемы	20
Они здесь!!!	28
Угадай число	30
Циклы	32
Таблица умножения	34
Использование списков	36
Словарь	40





ЧЕМПИОН ПО
ПРОГРАММИ-
РОВАНИЮ

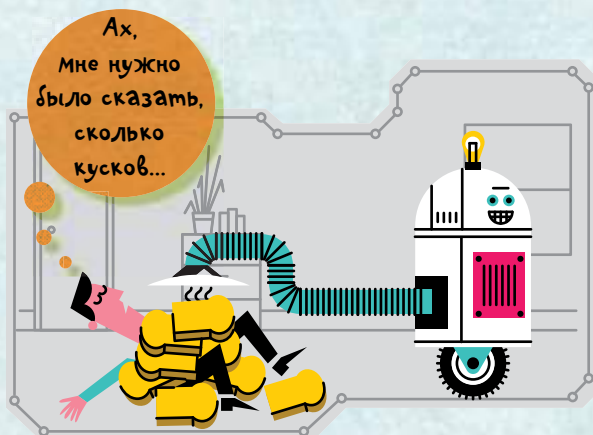
Шифрованное сообщение	42
Рисующая черепашка	46
Кнопка-сюрприз	54
Создаем шедевр	56
Сапер	60
Теннис	72
Скачивание среды Python	86
Работа с файлами	87
Отладка программ	88
Типы окон	90
Разбор программы	91
Глоссарий	92

ЧТО ТАКОЕ ПРОГРАММИРОВАНИЕ?

Программирование это, по сути, написание инструкций для компьютера, итоговая совокупность которых представляет собой программу. Компьютерные программы контролируют все: от работы смартфонов до полета космических ракет.

Знание нужного языка

Чтобы написать компьютерную программу, необходимо все понятные человеку действия разложить на простые и ясные для компьютера инструкции, выразив их на таком языке, который он понимает.



Что такое язык программирования

Язык программирования во многом схож с обычным языком человеческого общения. Только он имеет ограниченный набор слов и точные правила написания этими словами программ.

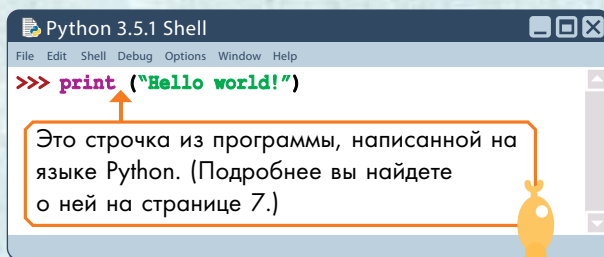
Язык программирования Python построен на текстовом формате программ. Это означает, что в текстах программ используются слова, числа и символы (такие как * или =).



Слово «Python» и приведенный выше логотип являются торговым знаком некоммерческой организации Python Software Foundation.

⚠ ВНИМАНИЕ! ⚠

Компьютер подчиняется инструкциям слепо и буквально. Он не может сам мыслить, так что все действия, выраженные не однозначно, не четко, остаются без исполнения.

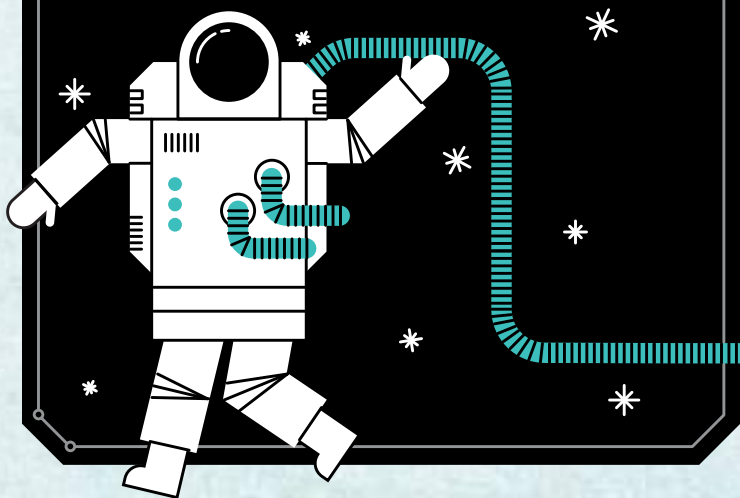


Хотя логотип языка Python изображает змею (питона), сам автор назвал свое творение в честь британской комик-группы, снимавшей в 1970-х годах популярное телешоу «Летающий цирк Монти Пайтона».

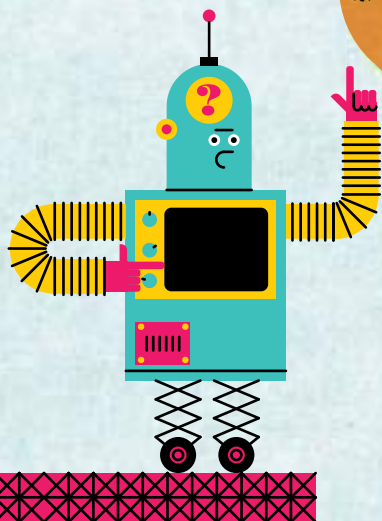
Почему мы выбрали Python?

Язык Python относится к числу наиболее популярных языков программирования и отличается чрезвычайно минималистичным синтаксисом. То есть при написании программ у вас не будет необходимости набирать слишком много символов.

Им пользуются для написания программ многие крупные организации, такие как Google, NASA и YouTube.



НАЧИНАЕМ!!!



Приступаем

Для запуска среды программирования Python вам потребуется только компьютер. Детальнее все описано на странице 86. Если Python не установлен на вашем компьютере, то его можно бесплатно скачать в интернете, например на их официальном сайте.

В Интернете имеются также ссылки на другие полезные ресурсы по программированию. Можно найти и файлы проектов, написанных на языке Python и содержащих законченный рабочий код для программ, приведенных в этой книге.

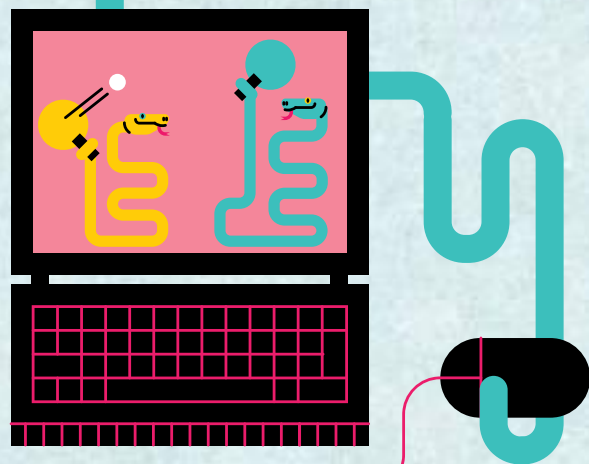
О чем эта книга

Из этой книги вы узнаете, как использовать арсенал языка Python в проектах самого разного рода: от простейшего приветствия до собственной компьютерной игры.

Все выполняемые компьютером операции разбиты на короткие и простые действия, а в конце имеется глоссарий полезных терминов.

В желтых вставках, как эта, приведены советы по использованию средств языка Python.

В синих вставках, как эта, объясняются ключевые идеи программирования в целом.



НАЧИНАЕМ С ЯЗЫКА PYTHON

Как только среда программирования Python будет установлена на ваш компьютер, вы сразу сможете начать создавать программы.

Программа IDLE

При скачивании среды программирования Python вы также получите программу **IDLE**, которая будет помогать вам набирать, редактировать и сохранять тексты программ. Не все работающие на Python программисты используют текстовый редактор **IDLE**, однако это хороший выбор для начала.

Версия для Windows

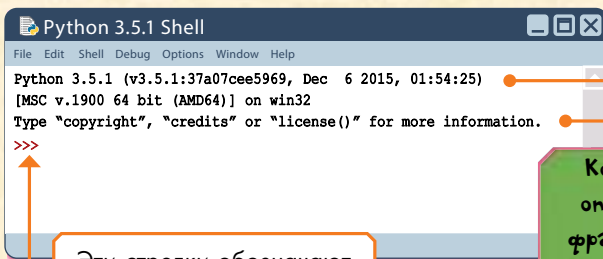
Чтобы открыть текстовый редактор **IDLE** в среде Windows, откройте стартовое меню (Start Menu) операционной системы, выберите опцию All Programs (Все программы), а затем в папке Python кликните по программе **IDLE**.

Версия для Mac OS

На компьютерах под Mac OS все очень похоже. Откройте Finder, затем Applications, директорию Python и кликните по **IDLE**.

Командное окно

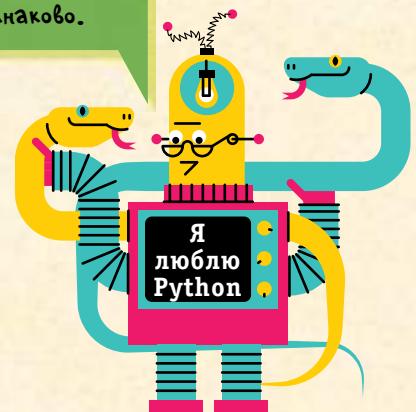
Когда вы открываете приложение **IDLE**, всплывает Командное окно (Shell window). Так оно выглядит на компьютере под Windows.



Эти стрелки обозначают приглашение на ввод команды, или Командную строку (**Command prompt**). Они показывают вам, где набирать строчки программы.

Командное окно полезно для оперативного тестирования фрагментов программы. В нем можно запустить набранный код, для чего достаточно выбрать в меню опцию **return** (возврат).

Среда разработки Python на различных компьютерах может выглядеть по-разному. Но функционирует она везде одинаково.



На стр. 86 вы можете прочитать о том, как скачать среду программирования Python. Только сначала убедитесь, что она еще не установлена у вас на компьютере.

Эти строки относятся к защите авторских прав с указанием версии Python. Они не являются частью кода вашей программы.



Программа-приветствие

По традиции, сложившейся в мире программистов, самой первой программой должна быть «Hello world!» (Привет, мир!). Именно эту строчку должен вывести на монитор компьютер, что в языке Python проще простого.

1. Откройте Командное окно.
2. Наберите эту строчку (за исключением уже появляющихся стрелок Командной строки).

```
>>> print("Hello world!")
```

Голубым полем выделяется ваш программный код (на с. 13 вы найдете, что обозначают те или иные выделения цветом).

Убедитесь, что вы написали без ошибок (важно соблюдать верную пунктуацию).

3. Нажмите return.
Вы должны увидеть что-то вроде:

```
Hello world!
```

Зеленый цвет прямоугольника означает, что это результат успешного запуска (run) программы.



Ошибка в программе

Если вы совершите ошибку в тексте программы, то на красном фоне появится сообщение об ошибке. Вы увидите что-то вроде:

```
SyntaxError: invalid syntax
```

Мы советуем вам, если такое случится, тщательно проверить и, возможно, снова набрать строчку программы, следующую за сообщением об ошибке. Вы должны в точности следовать правилам языка программирования.

Это сообщение означает, что в процессе набора была допущена ошибка. Проверьте написание, пунктуацию и заглавные буквы.

```
>>> print("Hello world!")
```

Строчная первая буква «р»

Открывающиеся и закрывающиеся кавычки

Для запуска программы нажмите return.



ФУНКЦИИ

В языке Python команда **print()**, или печать, это **функция**, а по сути она является миниатюрной, заранее написанной программой. Чтобы ее запустить, вам достаточно вписать в кавычки слово. Функция **print()** делает так, что компьютер показывает на мониторе все, что вы печатаете в кавычках.

ПРИВЕТ!



Наши поздравления!
Вы только что написали свою первую программу на языке Python. (На картинке Мир, который в ответ говорит вам «Привет!».)

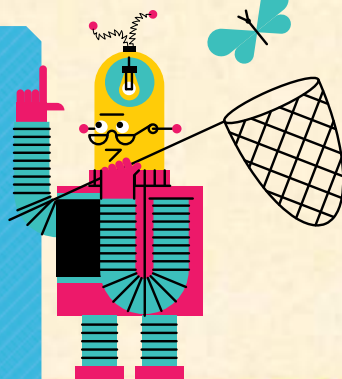


СИНТАКСИС

Итак, чтобы компьютер понимал вашу программу, вам необходимо научиться правильно писать служебные слова и нужным образом располагать их. Сочетание правописания, пунктуации и в некотором роде грамматики составляют синтаксис (syntax) языка программирования.

ОТЛАДКА

Процесс исправления ошибок в тексте программы называется отладкой (**debugging**). Пусть вас не беспокоят ошибки в программе — их делают все программисты. (Именно поэтому и потребовался специальный термин для названия того этапа написания программы, когда эти ошибки исправляются.)

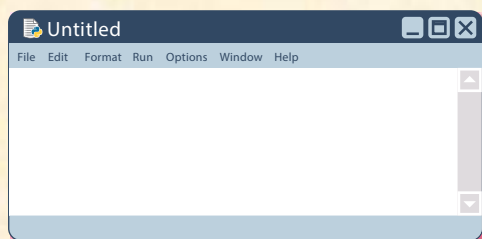


Сохранение текста программы

Командное окно дает вам прекрасную возможность для тестирования коротких фрагментов кода, однако сохранение текста программы в нем невозможно.

Чтобы позднее иметь возможность снова запускать или редактировать программу, ее текст необходимо сохранить, для чего вам потребуется вызвать Программное окно (Code window).

1. Кликните по пункту меню Файл (File) и, открыв этот пункт меню, выберите Новый файл (New file). На экране появится Программное окно с заголовком Untitled (Без названия), такое как это.



2. Кликните по пункту меню Файл (File) на верху Программного окна и выберите Сохранить как... (Save as). Введите удобное для вас имя файла, добавьте в конце .py и сохраните, кликнув Сохранить (Save).

hello world.py

Расширение «.py» даст знать компьютеру, что это программа, написанная на языке Python.

3. Наберите в окне текст, приведенный ниже, как вы уже делали в Командном окне.

```
print("Hello world!")
```

Программное окно не содержит стрелок командной строки.

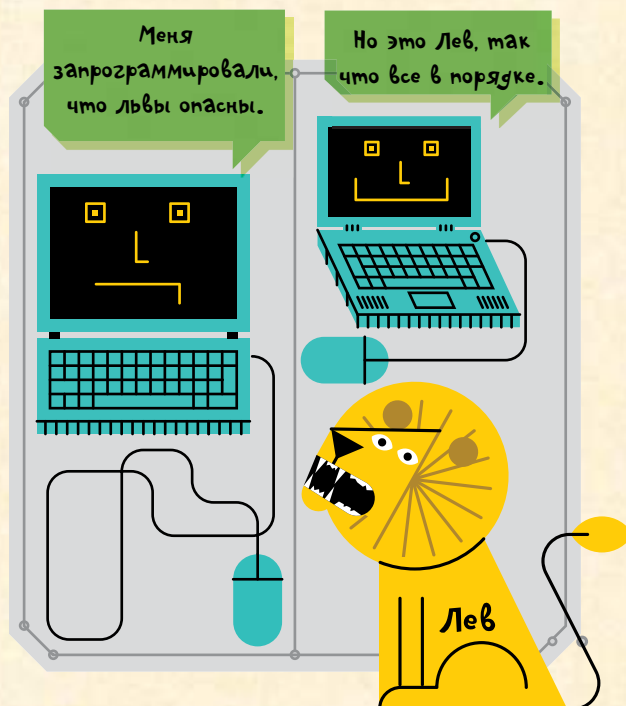
4. Снова кликните по пункту меню Файл (File) и выберите Сохранить (Save).

5. Теперь перейдите к пункту меню Запуск (Run), выберите из выпадающего меню Модуль запуска Run Module (он запускает только файлы среды Python).

6. На мониторе появится командное окно с текстом программы и слова «Hello world!» (Привет, Мир!).

КОМПЬЮТЕР НЕ МОЖЕТ МЫСЛИТЬ

Компьютеры не могут думать самостоятельно, так что любая, даже самая малая ошибка ставит его в тупик. Достаточно вам написать прописную букву вместо строчной, и компьютер остановит свою работу не в силах узнать слово.



Файлы языка Python автоматически сохраняются в той же директории (папке), что и программная среда Python.



Чтобы сохранить файлы в другом месте, кликните по пункту меню Файл (File), выберите Сохранить как... (Save as) и, кликнув по стрелке выпадающего меню, выберите место сохранения файла.



ИГРАЕМ С ЧИСЛАМИ

Среда программирования Python хорошо приспособлена для математики.

Достаточно вам набрать в командном окне арифметический пример и нажать в меню return, как вы получите ответ.

СЛОЖЕНИЕ

Откройте командное окно и наберите следующее:

```
>>> 2 + 2
```

Затем нажмите return, и вы увидите ответ:

```
4
```

Вычитание (и не только)

Если вы хотите произвести вычитание, то воспользуйтесь символом «-», как здесь:

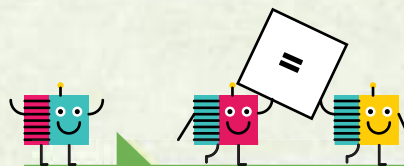
```
>>> 2 - 2
```

Для умножения используйте символ «*»:

```
>>> 2 * 2
```

Для деления используйте символ «/»:

```
>>> 2 / 2
```



В среде языка Python мы не используем символ «=» для получения ответа, поскольку у него там иное назначение. На следующей странице вы узнаете какое.



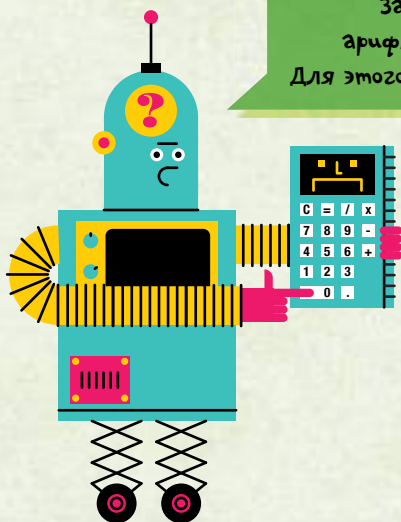
ОПЕРАТОРЫ

Символы, такие как «+» и «-», называются в среде Python математическими операторами. Вот самые распространенные математические операторы:

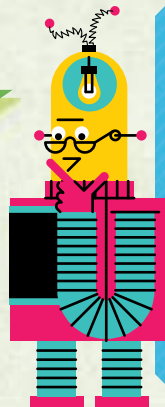
- + — сложение
- — вычитание
- * — умножение
- / — деление



Зачем мне Python для арифметических действий? Для этого у меня есть калькулятор.



Среда Python может производить намного более сложные операции, чем калькулятор. А полученные результаты нередко используются для дальнейших действий. Например, для отслеживания счета в игре.



В математике мы используем «x» для умножения и «÷» для деления. Но знак «÷» отсутствует на клавиатуре, а «x» похож на букву. Поэтому программисты используют знаки «*» и «/».

ПЕРЕМЕННЫЕ

Переменные (**variable**) это своего рода подписанные коробочки, в которых хранится информация. Вы можете изменять эту информацию, но только не название коробочки.



Как создавать переменные

Чтобы сообщить компьютеру, что мы хотим объявить переменную, мы и используем знак «=». Говоря строго, это называется «назначить переменную». Но можно объяснить все на пальцах, вернее, на бананах.

1. Откройте командное окно.
2. Наберите следующее:

```
>>> bananas = 5
```

Примечание: При исполнении этой строчки вы ничего не увидите. Компьютер всего лишь сохранит информацию для последующего использования.

Только что вы создали переменную под названием «bananas» (бананы) и присвоили ей значение 5.



ИМЕНА ПЕРЕМЕННЫХ

Вы можете давать переменным практически любые имена. Ограничения минимальны — имена не должны содержать пробелов. Также ими не могут быть специальные символы и «служебные слова», которые уже зарезервированы в среде программирования Python под другое назначение.

(Если вы случайно присвоите значение служебному слову, то быстро узнаете об этом, поскольку оно будет отмечено **оранжевым цветом**.)

Имена переменных не должны содержать цифр. На следующей странице вы узнаете, как составлять имена переменных из букв и символов.



Использование переменных

Чтобы использовать в программе переменную «bananas», вам придется написать еще несколько строчек кода.

1. Представьте себе, что обезьянка проголодалась и съела два банана. Для создания еще одной переменной, обозначающей количество съеденных бананов, нажмите return и наберите еще одну строчку.

```
>>> bananasEaten = 2
```

2. Нажмите return для получения еще одной строчки и напишите формулу для расчета количества оставшихся у обезьянки бананов.

```
>>> bananas - bananasEaten
```

3. Нажмите return и получите ответ. Вы должны увидеть:

```
3
```

4. Если обезьянка нашла больше бананов, вы можете назначить новое значение переменной «bananas». Повторите return и наберите новое значение, например, 10.

```
>>> bananas = 10
```

5. Нажмите return, вновь наберите формулу из пункта 2 и опять не забудьте return. Вы получите другой результат, поскольку переменная «bananas» изменила свое значение.

Без строчной переменной никуда!

В языке программирования Python слово и вообще любая последовательность букв и символов называется строкой и записывается в строчную переменную (тип string). Вот как вы можете поместить в переменную целую строку:

```
>>> exampleString = "I am a string."
```

Текстовая строка внутри кавычек «Я — строка» автоматически выделяется зеленым цветом.



На усмотрение пользователя

Если вы хотите, чтобы пользователь программы сам решал, чему будут равны переменные, то можете воспользоваться функцией `input()`.

В программировании любая информация, вводимая пользователем, называется «вводом» (`input`, см. синюю вставку). Но есть и специальная функция `input()`, которая заставляет компьютер остановить программу и дождаться, когда пользователь введет значение и нажмет `return`.

Пример ввода

С помощью функции `input()` попробуйте спросить имя пользователя.

1. Откройте командное окно и наберите:

```
>>> name = input("What is your name?")
```

Мы создаем переменную под названием `name` (имя).

Мы говорим компьютеру запустить функцию `input` и получить информацию, которую даст ему пользователь.

2. Нажмите `return`. Компьютер должен спросить ваше имя.

```
What is your name?
```

3. Введите ваше имя после этого вопроса и нажмите `return`. Ваше имя теперь хранится внутри переменной «`name`».

4. На следующей строчке выведите его на монитор.

```
>>> print(name)
```

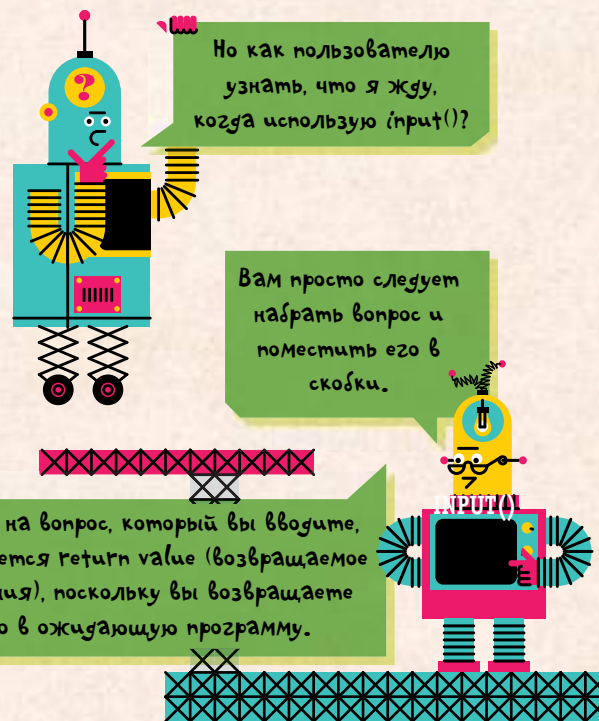
Наберите здесь именно имя переменной — `name`, а не свое имя.

5. Нажмите `return`. Компьютер покажет ваше имя синим цветом на следующей строке.



ВВОД-ВЫВОД

Вводом (`input`) называется все то, что вы «сообщаете» компьютеру, будь то набираемая строчка, перемещение указателя мыши или загрузка диска. (Только не стоит путать этот термин с функцией `input()` в языке Python, которая приглашает пользователя ввести значение.) Все, что компьютер «сообщает» нам — сообщение ли это на экране, распечатка или записанный диск — называется выводом (`output`).



ОТЛАДКА

Если вы получили сообщение об ошибке, то лучше проверьте набранный вами текст программы. Обратите внимание на следующее:

- Какие кавычки вы использовали? Допустимо и «», и "", но никак не «".
- Не пропустили ли вы скобки?
- Правильно ли вы написали служебные слова языка программирования?
- Не написали ли вы какую-нибудь команду с большой буквы? Это ошибка.



Создание персонажа

С помощью **переменной** и функций **print()** и **input()** вы сейчас будете создавать свой собственный персонаж.

1. Откройте командное окно. Кликните по пункту меню **File** (Файл) и выберите Новый файл (New File). Этим вы откроете программное окно. Теперь кликните по пункту меню **File** (Файл) в программном окне и выберите Сохранить как... (Save As). Придумайте имя для своей программы и наберите его в открывшейся строке.

2. Наберите заголовок, приглашающий в программу пользователя.

```
print("Create your character!")
```

Строчка «Создайте персонаж!» появится на экране, когда вы запустите программу.

3. Воспользуйтесь функцией **input()** и попросите пользователя дать своему персонажу имя.

```
name = input("What is your character called?")
```

4. Затем подобным образом попросите выбрать возраст персонажа, его сильные и слабые стороны (по одной строчке на каждый пункт).

```
age = input("How old is your character?")
strengths = input("What are your character's strengths?")
weaknesses = input("What are your character's weaknesses?")
```

Различные части кода будут иметь различные цвета. Переменные будут **черными**, функции — **фиолетовыми**, а строчки — **зелеными**.

5. Теперь попросите компьютер завершить процесс создания персонажа распечатыванием всех его данных при помощи функции **print()**. Как и прежде, новый пункт будет на следующей строчке.

```
print("Your character's name is", name)
print("Your character is", age, "years old")
print("Strengths:", strengths)
print("Weaknesses:", weaknesses)
print(name, "says, 'Thanks for creating me!'")
```

Переменная name (имя), созданная вами на шаге 3

СОХРАНИТЕ СВОЮ РАБОТУ

Чем длиннее будет ваша программа, тем чаще нужно сохранять ее. Для этого кликните по пункту меню **File** (Файл) и выберите опцию Сохранить (Save).

На обороте вы увидите, как работает эта программа...



ПРОВЕРКА ПРАВОПИСАНИЯ

В отличие от служебных слов языка программирования, то, что вы пишете внутри строчных переменных, может быть с ошибками или даже не иметь смысла. В кавычках вы можете набрать любую абракадабру, поскольку они (кавычки) указывают компьютеру, что это не команды и исполнять их не надо.

6. Итак, ваша программа завершена. Для ее сохранения кликните по пункту меню File (Файл) и выберите опцию Сохранить (Save). Затем кликните по пункту меню Запустить (Run) и выберите из выпадающего меню Модуль запуска (Run Module). При этом откроется командное окно или «всплывет» на фоне программного окна в зависимости от версии среды, и вы увидите в нем свой текст приглашения.

7. Создавайте свой персонаж, отвечая на вопросы, которые будут появляться на экране. Первое, что вы увидите:

Create your character!
What is your character called?

Выводимый на экран
текст будет синим.

8. Наберите имя персонажа, затем нажмите return. На следующей строке вы увидите предложение ввести возраст персонажа.

How old is your character?

9. Напишите возраст вашего персонажа и снова нажмите return. На следующие вопросы ответьте аналогично. (Сильная сторона — умею летать, слабая — скверный юмор.)

What are your character's strengths? Can fly
What are your character's weaknesses? Tells terrible jokes

10. Когда вы нажмете return после последнего вопроса, программа из пяти строчек выведет на экран информацию о вашем персонаже сразу после строчки приветствия.

Это будет примерно так («примерно» потому, что вы, наверняка, проявите фантазию):

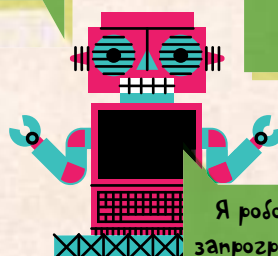
Your character's name is Brian
Your character is 245 years old
Strengths: Can fly
Weaknesses: Tells terrible jokes
Brian says, 'Thanks for creating me!'

Программирование заключается в написании «кода», как иногда называют программы, но не в их запуске. Так что в среде разработки Python при исполнении программы автоматически открывается командное окно.



Давай
назовем его
Брауном.

Ты всех
называешь
этим
именем.

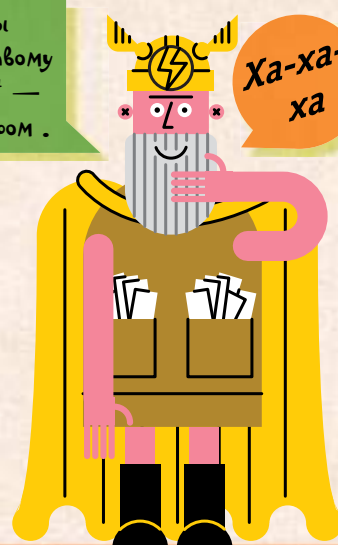


Я робот и я не
запрограммирован
на проявление
фантазии.



Что бы вы
сказали мертвому
роботу? RIP¹ —
ржаве́й с миром.

Ха-ха-
ха



¹ Rust in peace, в оригинале Rest in peace, покойся с миром. — Прим. пер.

Сложение чисел и строк

Когда пользователь вводит данные в программе на Python, они по умолчанию сохраняются в строчную переменную (**string**). Если же предполагается, что это будет число, то тогда вы должны прямо указать на это компьютеру, переключив тип переменной со строчной на числовую.

Напишите простую программу, которая будет показывать, сколько лет вам исполнится в следующем году.

1. Создайте новое программное окно и сохраняйте в нем текст программы, как вы это уже делали. Вот первая строчка текста программы, в которой вы спрашиваете, сколько лет пользователю сейчас:

```
age = input("How old are you?")
```

2. Создайте новую переменную, в которой вы будете сохранять, сколько лет будет пользователю в следующем году. Второй строчкой вы покажете это значение на экране.

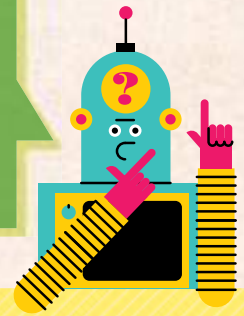
```
ageNextYear = age + 1  
print("You will be", ageNextYear, "next birthday.")
```

3. Запустите программу. Вы закономерно получите сообщение об ошибке, стоит вам нажать `return`. Это связано с тем, что компьютер не может произвести сложение без явного указания числового характера вводимой информации.

4. Чтобы исправить это, вам потребуется еще одна функция.

```
age = input("How old are you?")  
ageNextYear = int(age) + 1  
print("You will be", ageNextYear, "next birthday.")
```

Не забывайте, что слово или любой набор символов внутри кавычек — это строка, или строчная переменная. Вы сталкивались с ней, программируя вопросы и принимая ответы при создании своего персонажа.



КОМАНДА RESTART

Если вы хотите запускать программу более одного раза, не закрывая при этом командного окна, вам не обойтись без команды **RESTART** (рестарт). При использовании этого служебного слова компьютер «забывает» все, что было в командном окне раньше. Это равносильно открыванию нового командного окна для исполнения программы.



Функция **int()** сообщает компьютеру, что заключенную в скобки переменную следует рассматривать как **integer**, или целое число.



Сохраните программу и запустите ее снова. Теперь она будет работать без ошибок.

INTEGER



ЧИСЛА ЦЕЛЫЕ И С ПЛАВАЮЩИМ ЗНАКОМ

В программировании числа различаются, прежде всего, на целые (**integer**) и дробные (**float**). Пример целого числа — это 2. Дробные числа, или числа с плавающим знаком, — это все те, которые имеют в своей записи десятичную точку. Например, 2.5. Эта точка словно «плавает» по воле программиста между десятичными знаками.





ПРИНИМАЕМ РЕШЕНИЯ

Чтобы написать программу, которая сможет сама принимать решения, компьютеру надо научиться по-разному реагировать на различные ответы пользователя. Для этого вам потребуются операции сравнения для сопоставления величин и условные переходы для направления процесса расчетов.

Что такое условные операторы

Условные операторы — это миниатюрные программы, которые сопоставляют две части информации. На странице 9 приведены математические операторы, однако условные операторы также входят в их число. Например, оператор «==» проверяет, соответствуют ли два фрагмента данных друг другу.

Откройте командное окно и наберите:

```
>>> age = 10  
>>> age == 12
```

Вы создаете переменную и назначаете ее равной 10.

Оператор «==» проверяет, равны ли выражения друг другу.

Нажмите return. На следующей строчке вы увидите результат «Ложь».

False

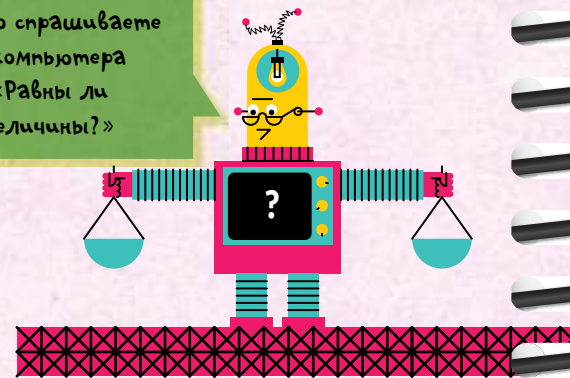
Программа выдала False (Ложь) потому, что переменная «age» (возраст) равна 10, а не 12. Для данного оператора возможны лишь два состояния True (Истина) и False (Ложь). Наберите в командном окне эти строчки и нажмите return.

```
>>> age = 10  
>>> age == 10
```

На этот раз на следующей строчке вы увидите результат «Истина».

В языке Python служебные слова True (Истина) и False (Ложь) пишутся с большой буквы, чтобы показать, что речь идет об условных операторах.

Оператор «==» превращает равенство в сравнение. Вы словно спрашиваете у компьютера «Равны ли величины?»



ОПЕРАТОРЫ СРАВНЕНИЯ

С помощью этих операторов компьютер сравнивает значения величин. Приведенная таблица будет полезна для некоторых проектов из этой книги.

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

БУЛЕВА ЛОГИКА

Так в математике называют работу с **логическими выражениями**. Истина и Ложь — это **булева величина** в отношении операций сравнения как булевых выражений. Если вы будете читать книги о программировании, то часто будете встречать эти термины, введенные в математику Джорджем Булем.

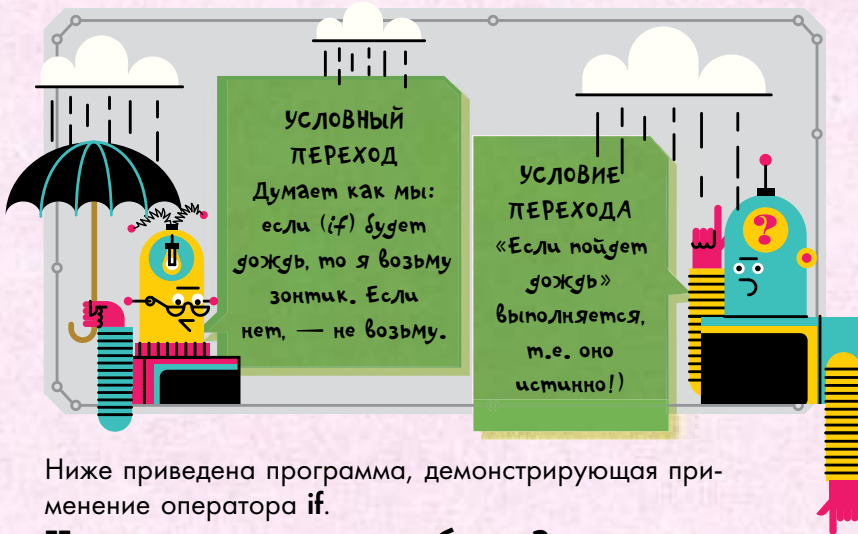
Что такое ветвление программы

Чтобы организовать в своей программе ветвление, нам необходимы команды условного перехода. Именно эти команды проверяют, истинно (True) или ложно (False) условие перехода. Затем они в зависимости от истинности направляют расчет по тому или иному пути.

Эти пути также называют **ветвлениями**, поскольку на блок-схеме таких программ получаются древовидные структуры.

Если (If)

Одним из важнейших операторов условных переходов является оператор **if** (если), который проверяет, истинно условие или ложно. Если условие истинно, то компьютер начинает исполнять инструкции, идущие непосредственно за условием. Если же ложно, то компьютер пропускает эти инструкции и переходит в следующую секцию (если она есть) — не делает ничего. В языке программирования Python **if** это служебное слово, так что не используйте его как имя переменной. В противном случае компьютер будет считать ее частью условного перехода и сообразит.



Ниже приведена программа, демонстрирующая применение оператора **if**.

Нравятся ли вам роботы?

1. Откройте новый файл и сохраните его. Наберите в нем такую строчку:

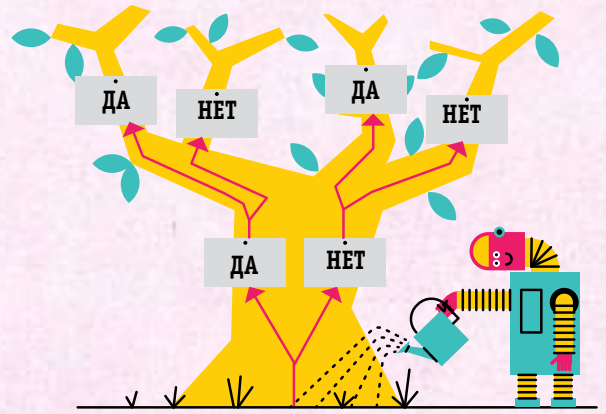
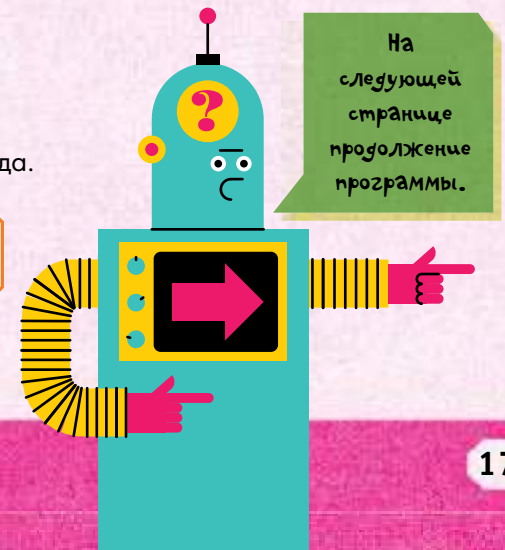
```
user_reply = input("Do you like robots? (Type yes or no)")
```

2. Нажмите **return** и наберите на новой строчке условие перехода.

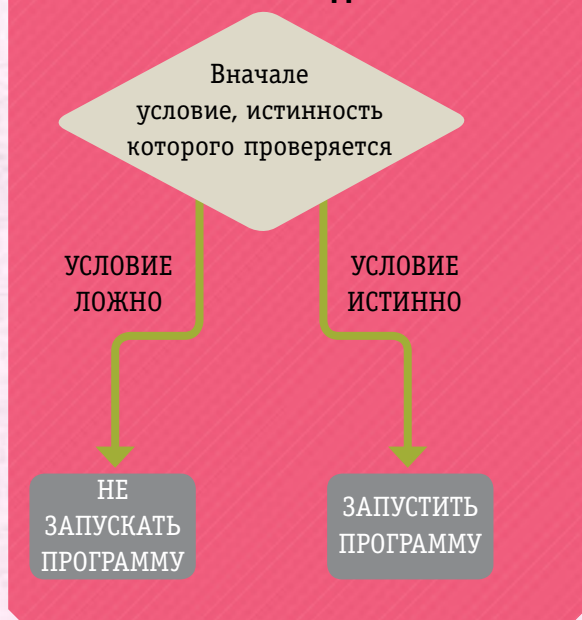
```
if user_reply == "yes":
```

Оператор «==» проверяет, ответил ли пользователь «да» (yes).

Оператор условного перехода, как и другие служебные слова, написан **оранжевым** цветом.



КАК РАБОТАЕТ IF-ПЕРЕХОД



3. Нажмите `return`, и в программном окне на следующей строчке автоматически появится стрелка длиной в четыре пробела. Наберите после нее следующее:

```
→ print("Beep boop!")
```

В этой книге смещение кода на новый уровень для удобочитаемости программы в дальнейшем будет изображаться стрелкой.

Если ответ пользователя совпадет с «yes», приведенной в программе, то фраза «Бип боп!» появится на экране.

4. Сохраните программу и запустите ее. Если вы ответите «yes», т.е. вам нравятся роботы, то компьютер доброжелательно напечатает вам в ответ «Бип боп!».

Если вы ответите «no» (или что-либо еще), то такая программа просто не ответит вам ничего.

Оператор `if` может включать в себя еще одну секцию, или ветвление, если вы поставите служебные слова `else` (иначе) или `elif` (иначе-если).

Else

Служебное слово «else» (иначе) сообщает компьютеру, по какому пути идти, если условие перехода `if` не выполняется.

1. Откройте программу с предыдущей страницы и добавьте под ней две новые строчки с текстом «Окей, тогда роботы тоже не любят тебя».

```
else:
→ print("Well, robots don't like you either.")
```

Окончательный вариант программы будет выглядеть так:

```
user_reply = input("Do you like robots? (Type yes or no)")
if user_reply == "yes":
→ print("Beep boop!")
else:
→ print("Well, robots don't like you either.")
```

Сохраните и запустите программу. Ответьте «нет» (no), чтобы активировать инструкции после слова `else`.

INDENTS

В языке программирования Python для группирования строк используется сдвигка кода на новый уровень. Это надо не только пользователю для лучшего понимания программы, но также и компьютеру, чтобы знать, какие инструкции исполнять дальше. Линии кода, объединенные одной сдвигкой, исполняются от начала до конца вместе. Если строчка кода сдвинута, это означает, что она принадлежит несдвинутой строчке над ней.

В этой книге сдвигка кода обозначается так:

4 пробела
8 пробелов
12 пробелов



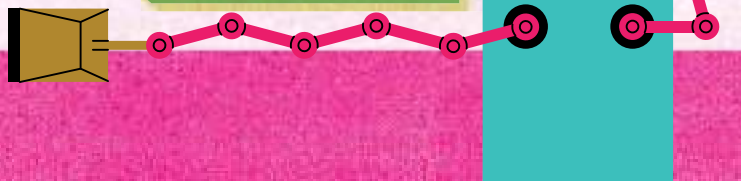
Каждая стрелка эквивалентна отступу в 4 пробела либо одной табуляции.

Табуляция вставляется нажатием кнопки «Tab».



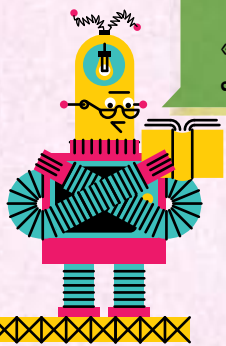
Для того чтобы убрать ненужную сдвигку строчки, воспользуйтесь клавишей `Backspace`. Слово `else` должно быть на одном уровне со строчкой с `if` выше.

Если условие после оператора `if` не выполняется, то компьютер переходит в секцию `else`, оставляя без внимания инструкции в предыдущей секции.



Оператор elif

Оператор `elif` — это сокращение от «else-if». Он всегда ставится после оператора `if` и вступает в действие, когда условие перехода не выполняется. Также он включает в себя свое собственное условие перехода и инструкции, которые исполняются только в случае его выполнения.



Оператор `else` не слишком сложен. Это как пункт «другое» в заданиях теста с множественным выбором.

Оператор `elif` позволяет вам встроить в свою программу еще один путь ветвления, каждый из которых реализуется при соответствующем выборе пользователя.

1. Сохраните программу под другим именем. Затем внесите в нее изменения, как показано ниже.

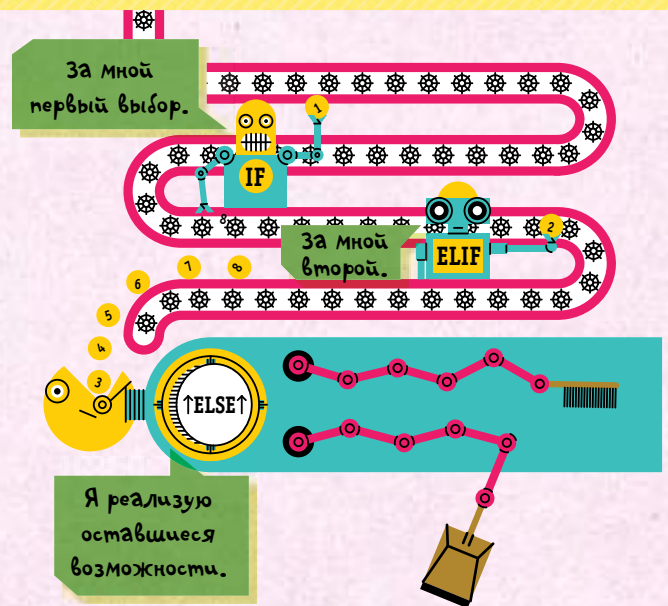
```
user_reply = input("Do you like robots? (Type yes, no, maybe or only ones with laser eyes)")
if user_reply == "yes":
    print("Beep boop!")
elif user_reply == "maybe":
    print("Make up your mind, human.")
elif user_reply == "only ones with laser eyes":
    print("Zzzap!")
elif user_reply == "no":
    print("Well, robots don't like you either.")
else:
    print("Your human nonsense offends us.")
```

2. Сохраните и запустите программу. Запустите ее несколько раз, выбирая каждый раз другой ответ. Если во время очередного запуска вы получите сообщения об ошибке, проверьте правильность написания служебных слов и пунктуации. Обратите внимание на программируемые вами реплики. (Дополнительные советы по отладке программ см. на стр. 88–89.)







IF, ELIF И ELSE

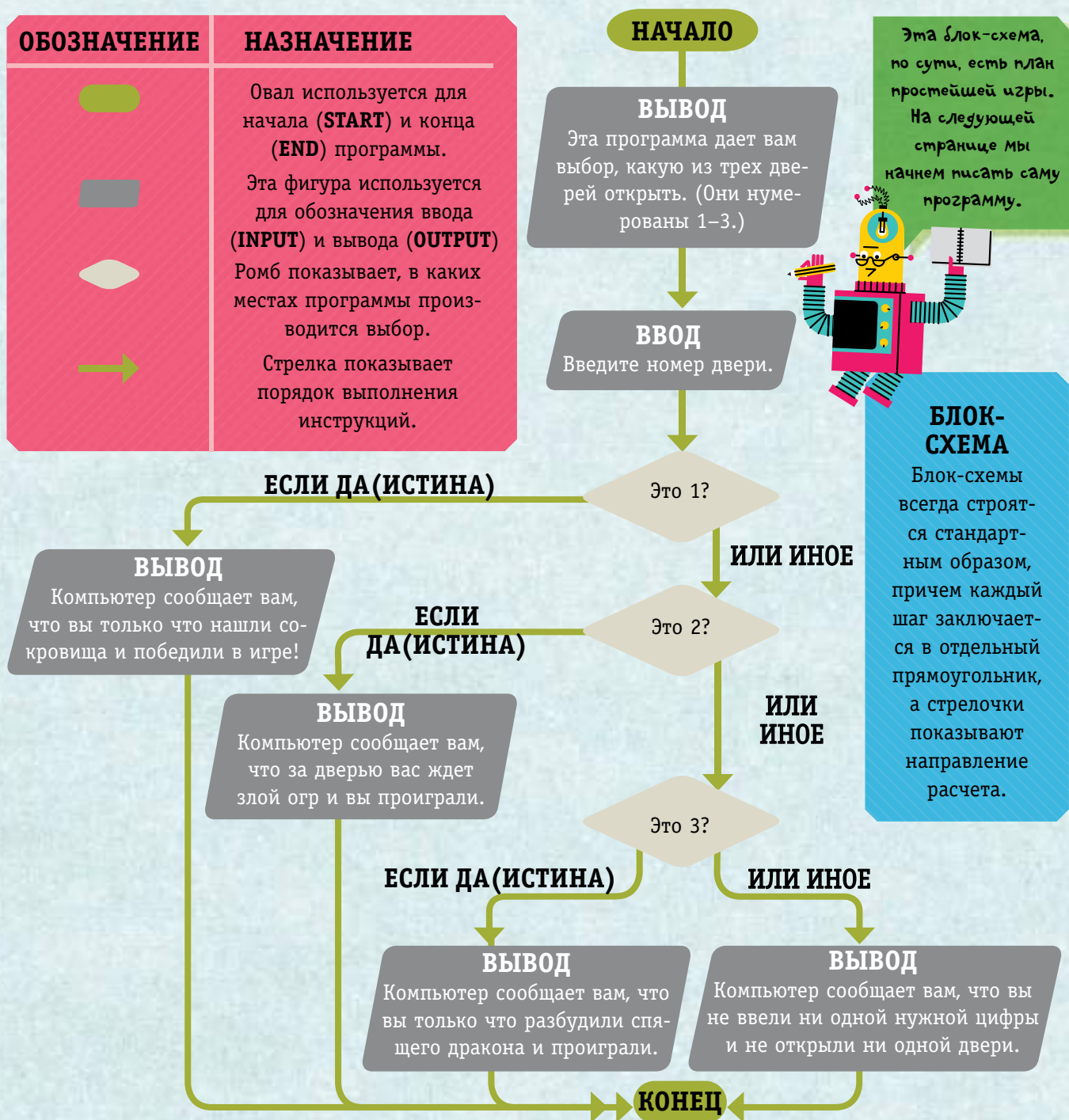
Работу этих трех операторов можно представить в виде конвейерной ленты. Оператор `if` — первый выбор, `elif` — второй выбор и, наконец, `else`, который активируется, если оба первых оператора не задействованы.



СТРОИМ БЛОК-СХЕМЫ

При написании программы с большим количеством ветвлений следует сразу в полной мере продумывать, что будет происходить в каждой ветви. Это помогает уменьшить количество ошибок при написании программы и эффективнее контролировать процесс ее исполнения.

ОБОЗНАЧЕНИЕ	НАЗНАЧЕНИЕ
	Овал используется для начала (START) и конца (END) программы.
	Эта фигура используется для обозначения ввода (INPUT) и вывода (OUTPUT)
	Ромб показывает, в каких местах программы производится выбор.
	Стрелка показывает порядок выполнения инструкций.



Готовы ли вы открыть замок с драконом?

А сейчас мы будем программировать вашу первую игру...

1. Откройте новый файл и сохраните его. Затем при помощи функции `print()` опишите мизансцену — «Вы в темной комнате таинственного замка».

```
print("You are in a dark room in a mysterious castle.")
```

2. Нажмите `return`. Теперь надо предложить пользователю выбор между тремя дверьми — «Перед вами три двери, и вы должны выбрать одну».

```
print("In front of you are three doors. You must choose one.")
playerChoice = input("Choose 1, 2 or 3...")
```

3. Нажмите `return` и создайте при помощи условия `if` и оператора «`==`» (см. стр. 16) первое ветвление в программе. Сейчас вы проверяете, нажал ли пользователь клавишу 1.

```
if playerChoice == "1":
```

Не забудьте двоеточие (:).

4. Нажмите `return` и при помощи функции `print()` наберите то, что компьютер сообщит пользователю при данном выборе — «Вы нашли комнату с сокровищами и победили в игре. Вы богаты!».

Сдвигка строк означает, что эти строчки кода будут исполняться, только если на них укажет находящаяся выше строчка без смещения.

```
→ print("You find a room full of treasure. You're rich!")
→ print("GAME OVER, YOU WIN!")
```

5. Затем сообщите компьютеру, что сообщить пользователю, если тот выберет вторую дверь — «За ней злой огр стукнул вас дубиной, и вы проиграли».

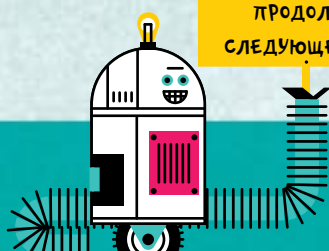
```
elif playerChoice == "2":
→ print("The door opens and an angry ogre hits you with his club.")
→ print("GAME OVER. YOU LOSE!")
```

6. Нажмите `return` и наберите, что сообщит компьютер пользователю, если тот выберет третью дверь — «Вы вошли в комнату и увидели спящего дракона. Он проснулся и с удовольствием съел вас. Вы проиграли.».

```
elif playerChoice == "3":
→ print("You go into the room and find a sleeping dragon.")
→ print("The dragon wakes up and eats you. You are delicious.")
→ print("GAME OVER. YOU LOSE!")
```

Нажмите `return`.

ПРОДОЛЖЕНИЕ НА
СЛЕДУЮЩЕЙ СТРАНИЦЕ.



7. Но пользователь может набрать и иное, кроме 1, 2 или 3. Чтобы предусмотреть и это, добавим оператор `else` и текст «Увы, вы не нажали ни 1, ни 2, ни 3!».

```
else:
```

```
    print("Sorry, you didn't enter 1, 2 or 3!")
```

8. Наконец, добавьте строчку, которая предложит пользователю сыграть еще после сообщения «Игра окончена».

```
print("Run the game again to have another go.")
```

9. Сохраните и запустите программу. Сделайте это несколько раз, выбирая каждый раз 1, 2 и 3. Затем введите что-то иное, чтобы проверить блок `else`.

Все сообщения,
записанные
в скобках, вы
можете менять
без вреда
для работы
программы.

Я думаю,
что поменяю
дракона на
дружелюбную
киску.



Возвращение в замок дракона

Вы можете добавлять еще операторы `if`, `elif` и `else`, чтобы придать игре больше вариантов и поворотов. Вы даже можете привнести в игру элемент случайности, используя для этого функцию `randint()`.

1. Откройте файл своей игры и выберите Сохранить как (Save As) в меню Файл (File). Затем присвойте программе другое имя. Так вы сохраните старую версию и сможете работать с новой без риска испортить оригинал.



При написании программы вы должны продумать все возможные действия пользователя. В отличие от компьютеров люди не всегда действуют по инструкциям.



На странице 87 вы узнаете немного больше о сохранении и запуске программы.

ОТЛАДКА

Если вы получили сообщение об ошибке, то проверьте:

- Правильность написания служебных слов
 - Не пропущены ли у вас кавычки или скобки?
 - Правильно ли выставлены сдвиги строчек?
- Не использовали ли вы случайно «=» вместо «==» в операторе `if`?
- Поставили ли вы двоеточие (:) после `if`, `elif` и `else`?



Вы можете запрограммировать столько вариантов выбора, сколько хотите, просто добавляя секции ветвления.



2. Первые несколько строчек остаются без изменения за исключением минимальных корректировок для описания новых правил игры с четырьмя дверями.

```
print("In front of you are four doors. You must choose one.")
playerChoice = input("Choose 1, 2, 3 or 4...")
```

3. Найдите функцию **print()** про спящего дракона и удалите две строчки, которые идут за ней.

```
elif playerChoice == "3":
    print("You go into the room and find a sleeping dragon.")
```

Изменяются только отмеченные места.

Удалите следующую пару строк после этой строки.

4. Добавьте под ней команду **print()**. В ней вы предлагаете пользователю попробовать украсть у дракона золото или проскользнуть мимо дракона к выходу.

```
print("You can either:")
print("1) Try to steal some of the dragon's gold.")
print("2) Try to sneak around the dragon to the exit.")
```

Эти строчки должны быть сдвинуты на четыре пробела. Если рабочая среда не сделает это автоматически, то не забудьте нажать клавишу табуляции либо четыре раза пробел.

5. Нажмите return. Теперь создайте новую переменную, которая позволит пользователю выбрать между вариантами взять золото дракона или просто уйти.

```
dragonChoice = input("Type 1 or 2...")
```

6. Нажмите return. Далее вам надо создать новый условный переход с **if**, который должен произойти, если пользователь нажмет клавишу 1 (т.е. выберет взять золото дракона).

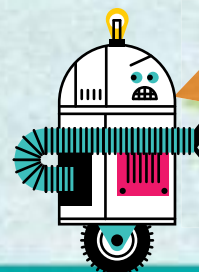
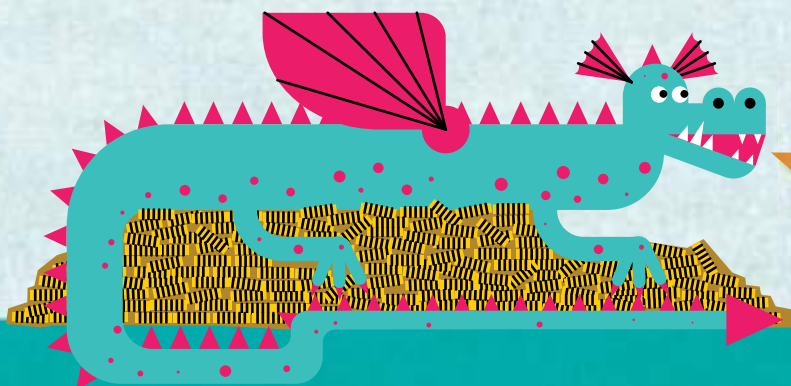
```
if dragonChoice == "1":
```

Эти строчки следует поставить с тем же отступом, что и строчку с **print()** из шага 4.

7. Для следующей строчки возьмите реплику из первого варианта игры — «Дракон проснулся и с удовольствием съел вас. Вы проиграли.»

```
print("The dragon wakes up and eats you. You are delicious.")
print("GAME OVER, YOU LOSE.")
```

Эти строчки автоматически будут иметь отступ в восемь пробелов.



На следующей странице вы найдете и другие варианты игры.

8. Нажмите return. Теперь добавьте оператор elif и две строчки с print(), приведенные ниже, чтобы создать ветвление. Оно будет задействовано, если пользователь выберет проскользнуть мимо дракона. В итоге он увидит на экране «Вы сбежали от дракона и выбрались из замка. Вы выиграли.»



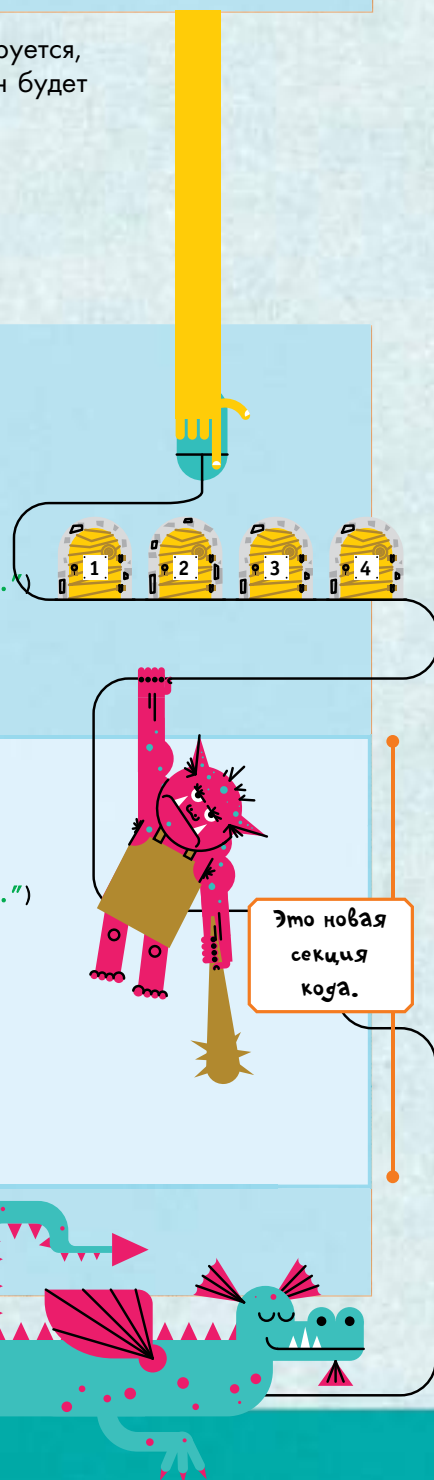
```
→ elif dragonChoice == "2":
→ print("You sneak around the dragon and escape the castle, blinking in the sunshine.")
→ print("GAME OVER! YOU WIN!")
```

9. На следующей строчке добавьте оператор else, который активируется, если пользователь не выберет ни 1, ни 2. Соответственно, на экран будет выведено «Увы, вы не нажали нужных клавиш.»

```
→ else:
→ print("Sorry, you didn't enter 1 or 2!")
```

Ниже приведено, как должна выглядеть ваша программа.

```
print("You are in a dark room in a mysterious castle.")
print("In front of you are four doors. You must choose one.")
playerChoice = input("Choose 1, 2, 3 or 4...")
if playerChoice == "1":
→ print("You find a room full of treasure. You're rich!")
→ print("GAME OVER, YOU WIN.")
elif playerChoice == "2":
→ print("The door opens and an angry ogre hits you with his club.")
→ print("GAME OVER, YOU LOSE.")
elif playerChoice == "3":
→ print("You go into the room and find a sleeping dragon.")
→ print("You can either:")
→ print("1) Try to steal some of the dragon's gold.")
→ print("2) Try to sneak around the dragon to the exit.")
→ dragonChoice = input("Type 1 or 2...")
→ if dragonChoice == "1":
→ print("The dragon wakes up and eats you. You are delicious.")
→ print("GAME OVER, YOU LOSE.")
→ elif dragonChoice == "2":
→ print("You sneak around the dragon and escape the castle,
blinking in the sunshine.")
→ print("GAME OVER! YOU WIN!")
→ else:
→ print("Sorry, you didn't enter 1 or 2!")
else:
→ print("Sorry, you didn't enter 1, 2 or 3!")
print("Run the game again to have another go.")
```



Повелитель случая Сфинкс

10. После того, как вы добавили строчку на шаге 9, создайте новый блок кода, начинающийся с `elif` и вводящий в игру сфинкса, который просит отгадать число от 1 до 10.

Пользователь должен ввести число либо я выхожу из программы!

```
elif playerChoice == "4":  
    print("You enter a room with a sphinx.")  
    print("It asks you to guess what number it is thinking of, between 1 and 10.")  
    number = int(input("What number do you choose?"))
```

Функция **int()** вам нужна, чтобы гарантировать, что компьютер будет обрабатывать ответ пользователя как целое число.

11. Для того чтобы Сфинкс сгенерировал случайное число, нам потребуется новая функция `randint()`. Все функции по генерации случайных чисел реализованы в отдельном модуле. Чтобы получить к ним доступ, необходимо служебное слово `import` (импорт). Перейдите в самый верх программы и наберите такую строчку:

```
import random
```

Эта команда загружает в вашу программу **random module** (модуль генерации случайных чисел).

12. Добавьте к программе оператор `if` с `else`, чтобы создать ветвление из двух возможностей.

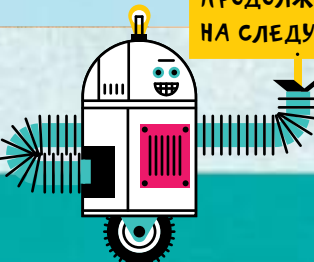
Функция **randint()** генерирует целое случайное число от 1 до 10.

```
if number == random.randint(1,10):  
    print("The sphinx hisses in disappointment. You guessed correctly.")  
    print("She must let you go free.")  
    print("GAME OVER. YOU WIN!")  
else:  
    print("The sphinx tells you that your guess is incorrect.")  
    print("You are now her prisoner forever.")  
    print("GAME OVER, YOU LOSE.")
```

Эта часть кода **if** активизируется, если пользователь угадывает число.

Часть кода с **else** вступает в действие, если игрок ошибается.

ПРОДОЛЖЕНИЕ ПРОГРАММЫ
НА СЛЕДУЮЩЕЙ СТРАНИЦЕ.



13. Для завершения игры остается отредактировать финальную часть. Эти инструкции используются в том случае, если пользователь так и не нажал ни одной из цифр от 1 до 4. Выводимой на экран строке будет небольшое изменение, отмеченное ниже:

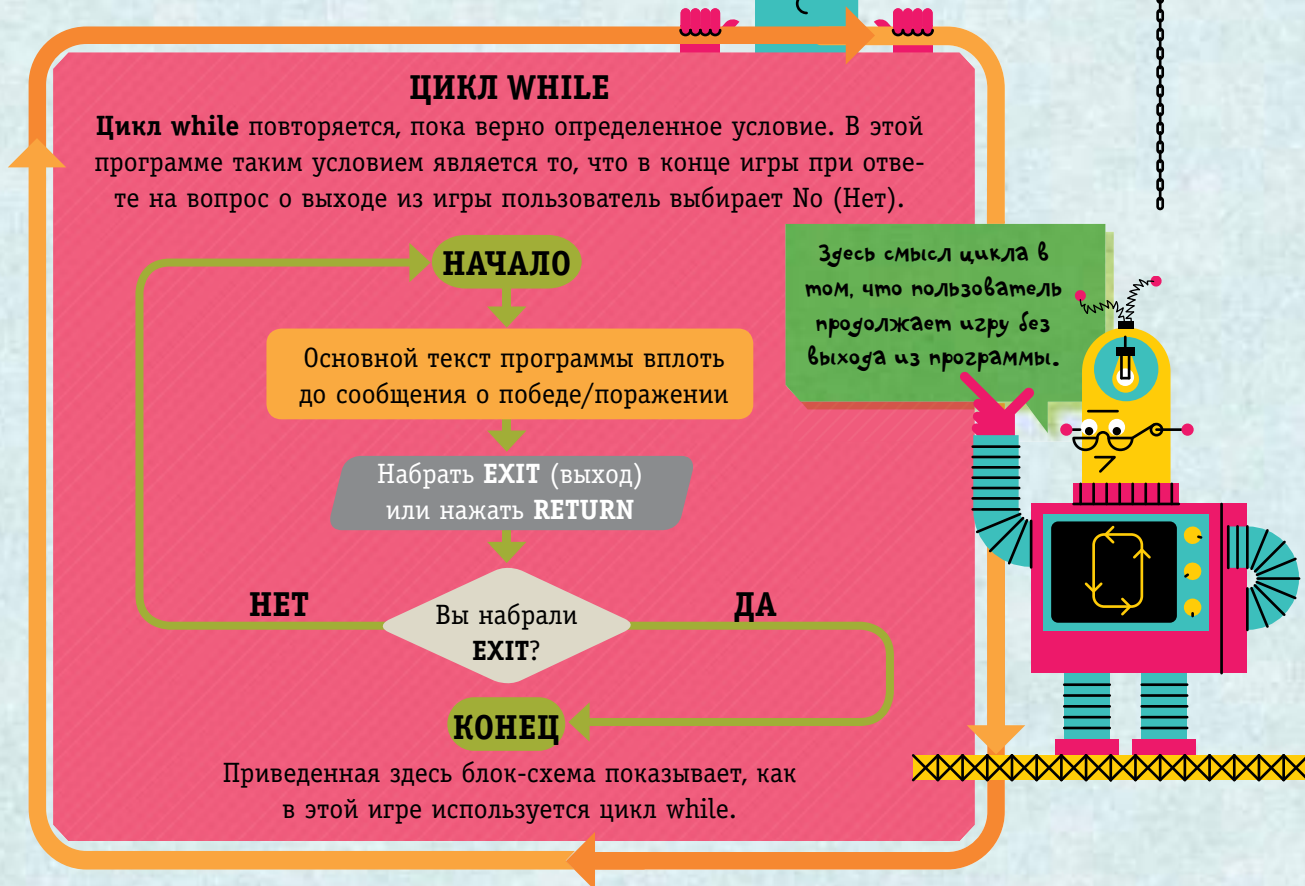
```
else:  
    print("Sorry, you didn't enter 1, 2, 3 or 4!")
```

Сохраните программу и запустите ее. Сообщение «Увы, вы не выбрали ни одного из чисел от 1 до 4» будет выведено на экран, а последняя строка с **print()** пригласит пользователя сыграть еще раз.

Добавление цикла

Цикл — это всего лишь повторяемая часть кода. Использование в программировании циклов экономит ваше время, поскольку позволяет не набирать несколько раз одно и то же.

При помощи **цикла while** (пока соблюдается условие) вы можете сделать, чтобы ваша программа перезапускалась автоматически без необходимости ее запуска пользователем.



Замок дракона: вариант с циклами

1. Откройте последнюю версию этой игры и сохраните ее под новым именем. Теперь переходите на конец самой первой строки.

```
import random
```

Нажмите **return** и вставьте новую вторую строчку.

2. Создайте переменную, которая будет использоваться для установки условия в цикле **while**.

```
exitChoice = "Nothing"
```

Нажмите **return**.

Значение этой переменной будет впоследствии изменено, так что сейчас совершенно без разницы, что вы наберете в кавычках.

3. На следующей строке добавьте команду на начало работы в цикле **while**.

```
while exitChoice != "EXIT":
```

Знак «!=» означает не равно.

То есть пока пользователь не наберет **exit** (выход) в конце игры, программа будет работать.

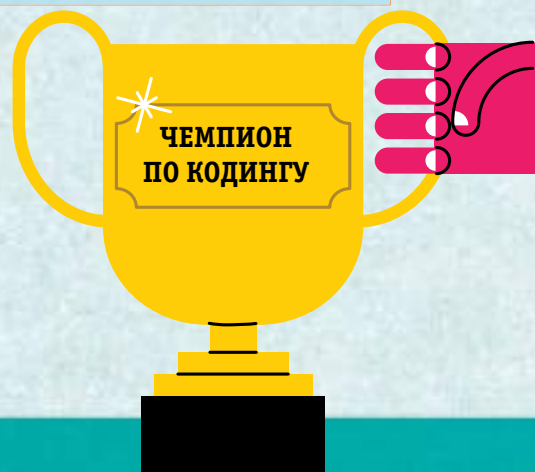
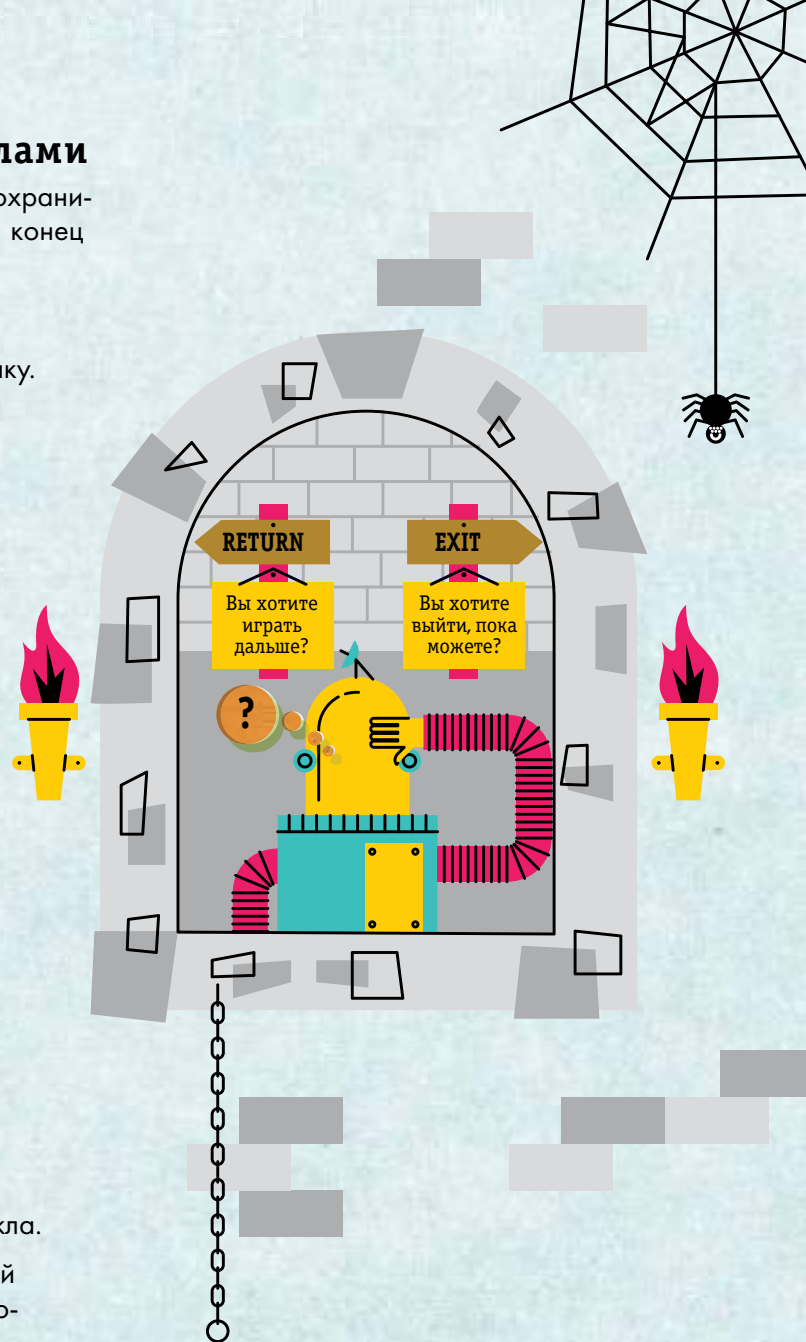
4. Продолжайте и добавьте еще одну сдвижку, чтобы программа знала, что все это внутри цикла.

5. Замените в конечной строчке, приглашавшей сыграть вновь, текст на «Нажмите return для продолжения игры или наберите exit для выхода.»

```
→ exitChoice = input("Press return to play again, or type EXIT to leave.")
```

То есть ваша программа запустится вновь после нажатия на **return** и остановится, если вы наберете «**exit**».

6. Сохраните программу и запустите ее несколько раз. Каждый раз делайте иной выбор, чтобы протестировать все ветвления.



ОНИ ЗДЕСЬ!!!

Пришельцы вторглись на Землю. Таков сюжет игры, и вам предстоит миссия остановить их. Но шансы не равны. Пришельцев больше и они быстрее.

Динамические игры

Теперь вам предстоит создать первую динамическую игру. Игроку предстоит угадать пароль от земного оружия, пока не настал конец.

1. Откройте новый файл, сохраните его и создайте переменную `aliens` (пришельцы). Определите значение переменной, равной количеству пришельцев в первой волне завоевателей.

```
aliens = 2
```

2. Нажмите `return` и создайте еще одну переменную (пароль).

```
password = "ALIENS"
```

3. Нажмите `return` и наберите все то, что компьютер должен сказать пользователю в начале игры.

```
print("Quickly! Aliens are invading the planet.")
print("You need to activate the global defence platforms.")
print("Hope you know the password, for Earth's sake...")
print()
print("-----")
print("      WELCOME TO THE GLOBAL DEFENCE NETWORK      ")
print("-----")
print()
```

4. Нажмите `return` и наберите строку:

```
guess = input("Please enter the password: ").upper()
```

Благодаря этой команде все, что вы напишите, будет напечатано строчными буквами.

СТЕПЕНИ

Чтобы в вашей программе пришельцы размножались быстрее, мы воспользуемся степенной функцией.

В математике степенное выражение это:

$$2^2 (2 \times 2)$$

$$2^3 (2 \times 2 \times 2)$$

$$2^4 (2 \times 2 \times 2 \times 2)$$

и так далее.

В языке Python для возведения в степень используется оператор «`**`».

Эта команда добавит пустую строку.

Воспользуйтесь тире, чтобы создать эти прерывистые линии.

Где ваш лидер?
Мы завоюем вас.

5. Нажмите **return** и создайте цикл **while**. Инструкции будут исполняться в этом цикле, пока вы не наберете правильный пароль, заданный в начале программы.

```
while guess != password:
```

«!=» означает не равно.

6. Нажмите **return** и добавьте еще одну команду **print**.

```
→ print()
→ print("INCORRECT PASSWORD.")
→ print()
```

Это сообщение будет показано, если пароль не угадан.

7. Нажмите **return**. Здесь вы будете программировать увеличение количества пришельцев при помощи оператора возведения в степень «**». Потом в строке с **print** вы выведете количество пришельцев.

```
→ aliens = aliens ** 2
→ print("There are", aliens, "aliens now on Earth. Try again!")
```

8. На следующей строчке при помощи операторов ветвления **if** и прерывания **break** (см. в желтой вставке) вы организуете выход из цикла, если количество пришельцев превышает количество людей. Игра при этом останавливается.

```
→ if aliens > 7400000000:
→     break
```

Предполагается, что на Земле живет 7,4 млрд человек.

9. На следующих строчках приводятся подсказки, но только до тех пор, пока количество пришельцев не превысит количество людей.

```
→ print()
→ print("Password hint: the things that are attacking us.")
→ print()
→ guess = input("Quick! Please enter the password: ").upper()
```

10. Закончите программу ветвлением **if/else**, как показано. Инструкции после условия **if** реализуются, если количество пришельцев превысит 7,4 млрд. Код в секции **else** активируется, если пароль угадан правильно.

```
if aliens > 7400000000:
→ print("Nooooo! The aliens have outnumbered us. All is lost.")
else:
→ print("Hooray! We won the fight and the world is saved!")
```

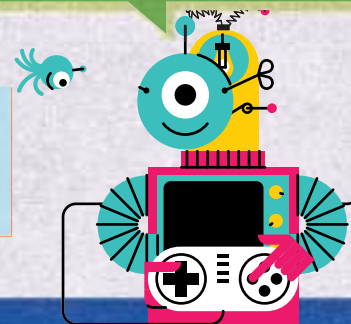
В этой блок-схеме проиллюстрировано, как работает эта программа.



BREAK

Для остановки работы программы вы можете воспользоваться служебным словом **break** (прерывание) в секции после условного перехода **if**.

Когда вы сохраните и протестируете программу, испытайте ее на своих друзьях. Возможно, вам придется добавить еще подсказок, если угадать пароль окажется слишком трудно.



УГАДАЙ ЧИСЛО

При помощи условных переходов вы можете создавать простые игры на угадывание числа, причем в них будет использоваться и генератор случайных чисел, и циклы. Со всем этим вы уже знакомы.

В этой игре компьютер будет генерировать случайное число от 1 до 20, которое вы попытаетесь угадать.

1. Начните с нового программного окна и загрузите модуль `random`. Затем вы с помощью функции `randint()` сгенерируете число.

```
import random
number = random.randint(1,20)
```

2. Попросите пользователя угадать число и сохраните его ответ в переменной `guess` (догадка).

```
guess = int(input("I'm thinking of a number from 1 to 20. What is it?"))
```

Нажмите `return`.

3. Теперь организуйте цикл `while`. Программа будет работать, пока пользователь не угадает число.

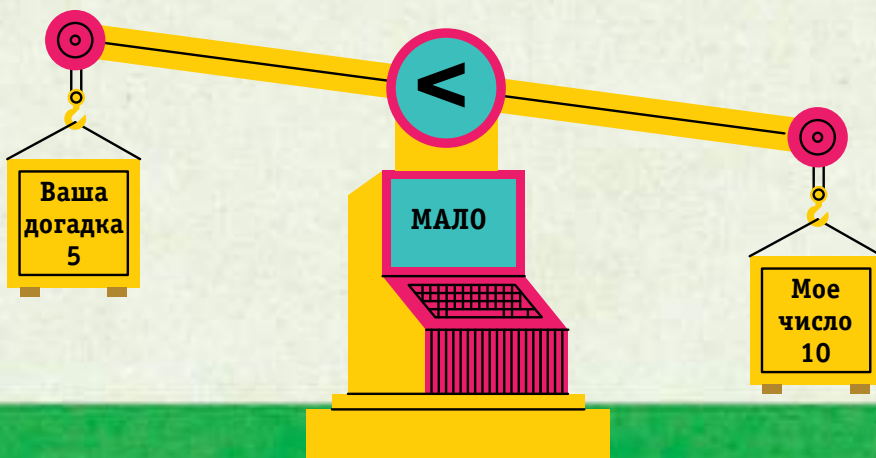
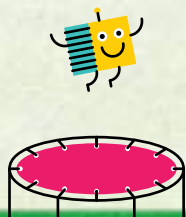
```
while guess != number:
```

Помните, оператор `!=` означает «не равно».

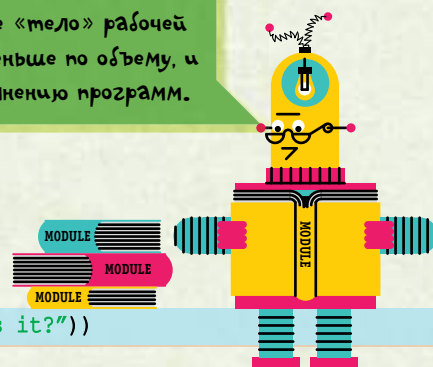
4. Нажмите `return`. На следующей строке добавьте условие `if`, которое будет сравнивать догадку пользователя с загаданным числом и проверять, больше или меньше эта догадка.

```
→ if guess < number:
```

Оператор `<` означает «меньше чем».



В языке Python модулями называются файлы, где хранятся функции, которые не нужны вам постоянно. Эти модули загружаются только тогда, когда вы об этом просите, что позволяет сделать основное «тело» рабочей среды Python и меньше по объему, и быстрее по исполнению программ.



КОНВЕРТИРОВАНИЕ ВВОДА

Среда языка Python по умолчанию воспринимает все, вводимое пользователем, как строчную переменную (`string`). Для перевода строки в целое число используется функция `int()`.

5. Нажмите **return**. На следующей строчке должно быть уже два отступа. В тексте укажите, что «Число пользователя мало по сравнению с загаданным».

```
→ print("Your number was too low...")
```

Это сообщение появится, если ваше число будет меньше загаданного компьютером.

6. Нажмите **return** и уберите один отступ. Добавьте оператор **else** и еще одну функцию **print()**, на случай, если число пользователя окажется больше загаданного.

```
→ else:  
→ print("Your number was too high...")
```

Это сообщение появится, если догадка окажется больше числа, задуманного компьютером.

Не переживайте, если тестовый запуск выявит ошибки в программе. Исправление собственных ошибок это неплохой путь познания того, как работают те или иные элементы кода.

7. Нажмите **return** и попросите пользователя сделать еще одну догадку. Поскольку в программе используется цикл **while**, она будет работать, пока пользователь не угадает число. Если же он не будет угадывать, на экран будет выводиться сообщение «Попробуйте еще раз...».

```
→ guess = int(input("Please try again..."))
```

8. Нажмите **return**, уберите отступ и добавьте последнюю строчку. Поскольку она не смещена, то компьютер считает, что она за пределами цикла **while**.

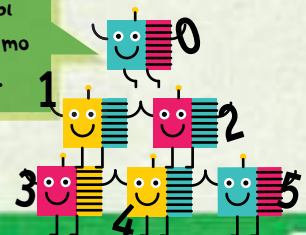
```
print("Congratulations! Correct answer!")
```

Сообщение «Наши поздравления, вы правильно угадали!» пользователь увидит при правильной своей догадке.

9. Сохраните программу и запустите ее для тестирования. Вот как выглядит текст программы в окончательном варианте:

```
import random  
number = random.randint(1,20)  
guess = int(input("I'm thinking of a number from 1 to 20. What is it?"))  
while guess != number:  
    if guess < number:  
        print("Your number was too low...")  
    else:  
        print("Your number was too high...")  
    guess = int(input("Please try again..."))  
print("Congratulations! Correct answer!")
```

Убедитесь, что в пунктах 2 и 7 вы поставили две пары открывающихся и, соответственно, закрывающихся скобок. Если вы пропустите хотя бы одну скобку, то получите сообщение об ошибке.



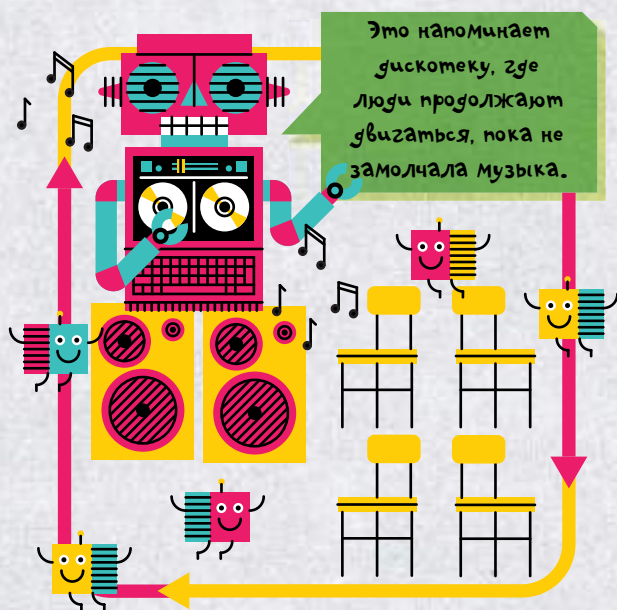
ЦИКЛЫ

Цикл **while** заставляет компьютер повторять определенные инструкции, пока выполняется некоторое условие. Если вы хотите сказать компьютеру, сколько именно раз надо повторить инструкции, то используйте цикл **for**.



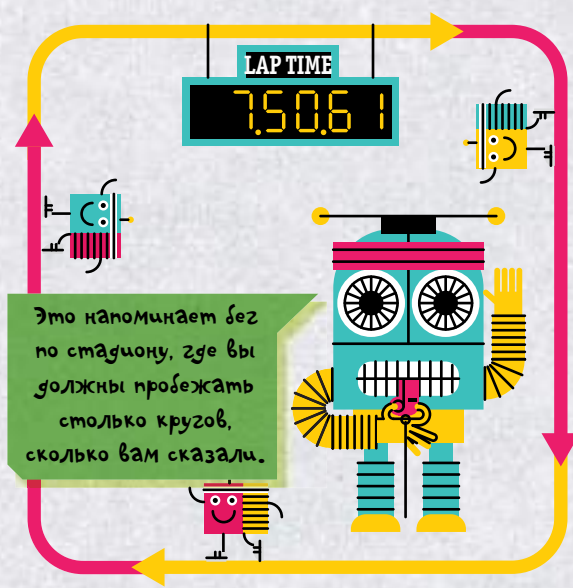
Цикл while

Цикл типа **while** прекращается только тогда, когда происходят изменения. Если ничего не происходит, то инструкции продолжают исполняться.



Цикл for

Цикл **for** (для) позволяет нам сказать машине, сколько именно раз повторять инструкции и когда остановиться.



Сколько раз?

Когда вы используете цикл **for**, вам необходимо указывать, сколько раз его повторять. Для этой задачи в языке программирования введена функция **range()**.

Чтобы создать с использованием **range()** самый простой цикл, откройте новый файл, сохраните его и наберите следующее:

```
for x in range(0,10):  
    print(x)
```

Эти две цифры разделяются при помощи запятой.



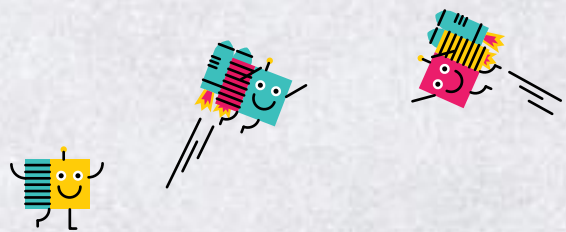
RANGE()

Если вы используете функцию **range()**, то создаете, по сути, список чисел. Числа, которые вы указываете в скобках, сообщают компьютеру, с какой цифры начинать и какой заканчивать.

Инструкция «**for x in range()**» говорит компьютеру выполнять указанные ниже действия для каждого из элементов сформированного списка.

Что делает цикл?

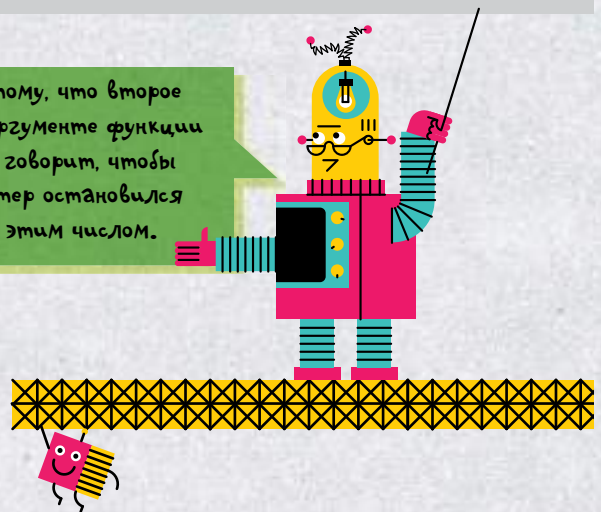
Если вы запустите коротенькую программу с предыдущей страницы, то увидите в командном окне цифры от 0 до 9.



0 1 2 3 4 5 6 7 8 9



Это потому, что второе число в аргументе функции `range()` говорит, чтобы компьютер остановился перед этим числом.



Детали и особенности

В простой строчке цикла `for` запрограммировано много операций. Давайте рассмотрим их подробнее.

«`for...in`» это те служебные слова, которые сообщают компьютеру переходить к списку, который далее и брать по порядку различные значения переменной из списка.

Буква «`x`» это **переменная**. Вы можете обращаться к ней всякий раз, когда это вам нужно.

```
for x in range(0,10):  
    print(x)
```

Функция `print()` говорит компьютеру выводить на экран значения переменной `x`. Каждый раз, как проходит цикл `for`, значение переменной `x` изменяется.

Функция `range()` создает список чисел. Этот список отвечает за то, сколько раз будут исполнены инструкции в цикле.

Сила языка Python в том, что в немногих служебных словах скрыто очень многое.



ТАБЛИЦА УМНОЖЕНИЯ

В этом примере вы с помощью **цикла for** средствами языка Python будете составлять таблицу умножения.



1. Откройте новый файл, дайте ему имя и наберите строку с предложением ввести множитель.

```
table = int(input("Please enter a times table: "))
```

Вы создали переменную **table** (таблица), которая по выбору пользователя будет участвовать в формировании таблицы умножения. Именно на нее будут умножаться все остальные числа.

2. Нажмите **return**, затем создайте **цикл for**, используя функцию **range()**.

```
for x in range(0,13):
```

Это диапазон (**range**) от 0 до 12.

3. Нажмите **return** и наберите эту команду. Компьютер будет выводить на экран столбик таблицы умножения с числом, которое выберет пользователь.

```
print(x, "x", table, "=", x*table)
```

Это оператор умножения.

Текст в кавычках будет выведен на экран между цифрами.

4. Сохраните и запустите программу. В командном окне вы увидите таблицу умножения с числом, которое вы выбрали. Например, такую:

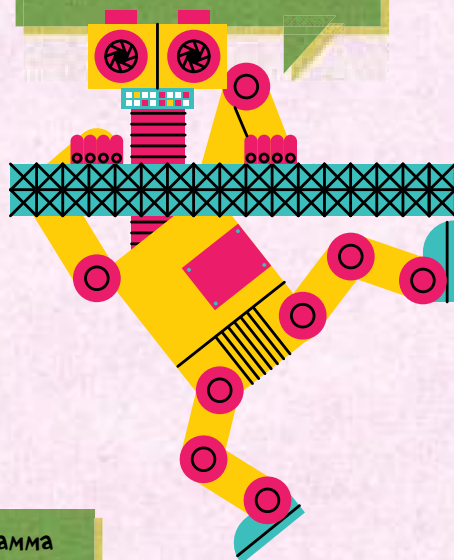
```
0 x 7 = 0
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
```

(И так далее до 12x7.)

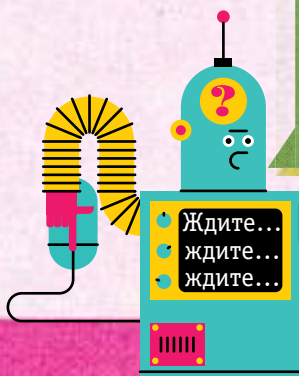


Напомним вам, что функция **int()** превращает строку в целое число. Все, что вводит пользователь, программа будет пытаться обрабатывать как числа.

Эта программа перемножает все числа от 0 до 12 на число, которое выберет пользователь.



Когда программа у вас заработает, постарайтесь изменить пределы в функции **range()**. Только не делайте второе число слишком большим, иначе вам придется слишком долго ждать.



Ждите...
ждите...
ждите...



Вложенные циклы

Если вы помещаете один цикл внутрь другого, то внутренний цикл называется **вложенным**.

Здесь в программе используется вложенный цикл **for**, в котором печатаются звуки, воспроизводимые роботом.

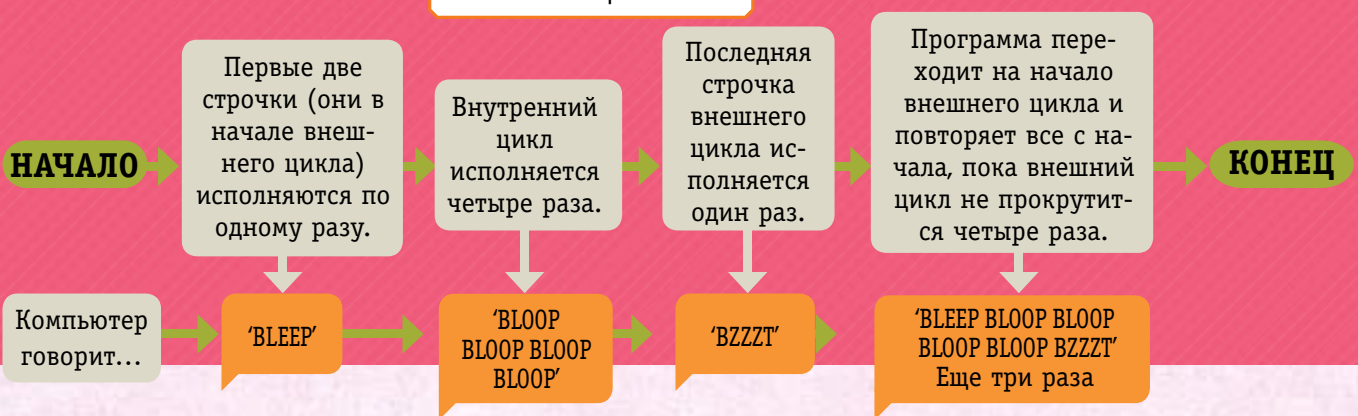


```
for x in range(1,5):  
    print("Bleep")  
    for y in range(1,5):  
        print("Bloop")  
    print("Bzzzt")
```

Желтым отмечен **внутренний** цикл.

Белым отмечен **внешний** цикл.

Вот как это работает:



1. Откройте новый файл, дайте ему имя и впечатайте эту пару строк:

```
for x in range(1,5):  
    print("Bleep")
```

Эта инструкция заставляет компьютер запускать внешний цикл четыре раза.

2. Затем создайте внутренний цикл:

```
for y in range(1,5):  
    print("Bloop")
```

Внутренний цикл также проходит четыре раза.

Строчки инструкций внутреннего цикла имеют двойной сдвиг.

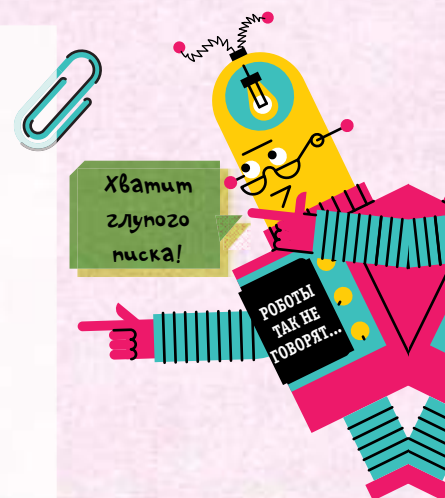
3. На следующей строчке наберите еще одну функцию `print()`, завершающую внутренний цикл. Ее сдвиг должен соответствовать сдвигу строки с `print()` из первого шага.

```
print("Bzzzt")
```

Одиночный сдвиг означает, что мы опять возвратились во внешний цикл.

Звуки, выводимые на печать (звукоимитацией), будут повторяться таким образом.

4. Сохраните и запустите программу. Вы увидите повторяющуюся звукоимитацию роботов.

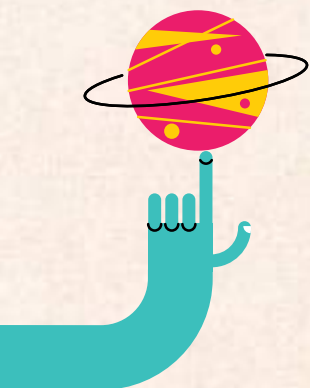


ВЛОЖЕННЫЙ VS НЕВЛОЖЕННЫЙ

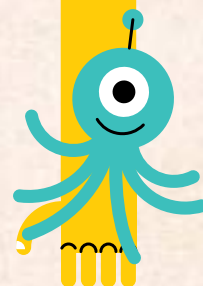
Обычный невложенный цикл позволяет вам повторять участок кода без необходимости его повторного написания.

Вложенный цикл позволяет повторять два или более участка кода — более короткий путь.

(Лучшие программисты — ленивые программисты. Чем короче ваш программный код — тем лучше.)



ИСПОЛЬЗОВАНИЕ СПИСКОВ



Язык программирования Python позволяет вам сгруппировать отдельные части информации в виде списков (**list**). Списки сохраняются в виде переменных отдельного вида.

Создание списка

Для создания списков используйте квадратные скобки, как здесь:

```
spacelist = ["rocket", "planet", "asteroid", "alien"]
```

Элементы списка разделяются запятыми.

Если вы хотите произвести операцию со словами из этого списка, то можете вызывать элементы по индексу, т.е. их положению в списке.

```
spacelist = ["rocket", "planet", "asteroid", "alien"]
```

0 1 2 3

Это те самые индексы. (В языке Python счет идет с 0.)

Индексация используется не только для организации списков из названий. Вы можете также использовать ее для сохранения букв алфавита в одной переменной (см. стр. 44).

Существует очень много вариантов организации списков и работы с его элементами.

Просмотр элементов списка

1. Откройте новый файл и создайте список.

```
spacelist = ["rocket", "planet", "asteroid", "alien"]
```

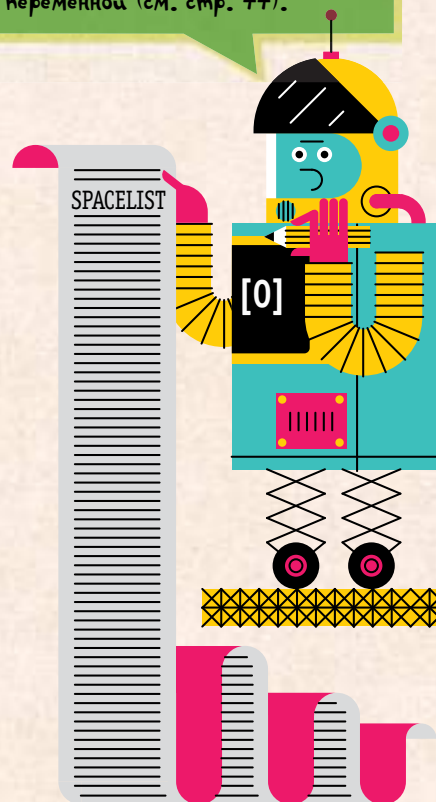
2. Выведите элемент списка на экран при помощи функции **print()**.

```
print(spacelist[0])
```

В квадратных скобках поместите номер элемента, который вы хотите увидеть.

3. Сохраните и запустите свою программу. Если вы правильно набрали строчки из этого примера, то должны увидеть следующее:

```
rocket
```



Просмотр всего списка

1. Вы можете использовать **цикл for** для просмотра всего списка. Замените вторую строчку на следующую:

```
for item in spacelist:  
    print(item)
```

item это переменная, которая «проходит» по всем индексам списка.

2. Сохраните программу и запустите ее. Вы увидите вот что:

```
rocket  
planet  
asteroid  
alien
```

Цикл **for** проходит по каждому индексу списка, выводя названия на экран каждый раз на новой строчке.

```
['rocket', 'planet', 'asteroid', 'alien']
```

Команда **print(spacelist)** покажет все названия в одной строке без пробелов.

Увидеть элементы списка можно и проще, всего лишь набрав `print(spacelist)`, однако выводиться они будут не так красиво.

Замена, удаление и добавление элементов

1. С помощью **цикла for** можно заменять элементы списка и вводить новые. Давайте напечатаем эту строчку перед **циклом for**.

```
spacelist[0] = "planet zarg"
```

Эта строчка заменяет первый элемент списка (**rocket**) на новый (**planet zarg**).

2. Сохраните и запустите программу. Первым элементом в списке теперь будет **«planet zarg»**.

3. Для удаления элемента наберите вместо строчки из пункта 1 следующую:

```
del spacelist[0]
```

4. Сохраните и запустите программу. Первым элементом (с индексом 0) теперь станет **«planet»**, поскольку элемент **«planet zarg»** был удален.

5. Попробуйте добавить элементы в конец списка при помощи команды **append**.

```
spacelist.append("moon")
```

Если вы запустите программу теперь, то последним элементом будет **«moon»**.

Если вы хотите сохранить различные версии своей программы, то при каждом сохранении давайте ей новое имя.



DEL И APPEND

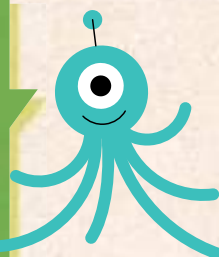
Команда **del** (удалить) удаляет все элементы в списке и «сбрасывает» индексы. Команда **append** добавляет указанные элементы в конец списка.

Объединение списков

1. Списки можно еще и складывать, т.е. объединять. Откройте новый файл и наберите в нем:

```
spacelist1 = ["rocket", "planet", "asteroid", "alien"]  
spacelist2 = ["space station", "star", "black hole"]  
spacelist = spacelist1 + spacelist2  
for item in spacelist:  
    print(item)
```

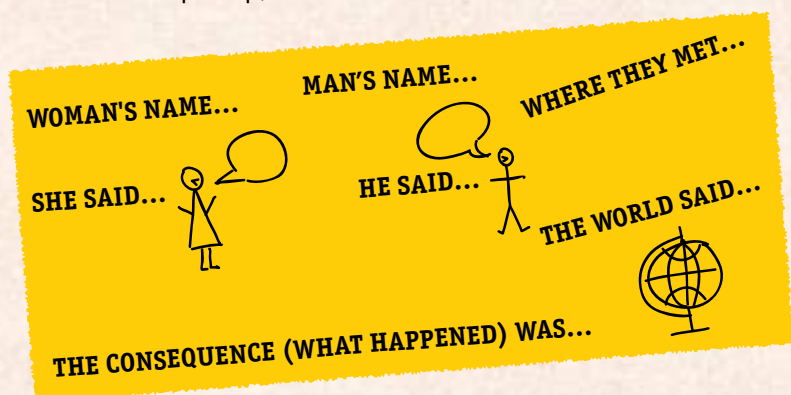
Окончательный вариант списка `spacelist` должен включать в себя все элементы из списков `spacelist1` и `spacelist2`.



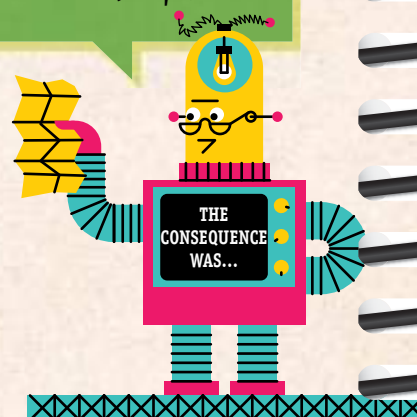
2. Сохраните файл и запустите программу. Вы увидите список из всех 7 элементов.

Игра в последовательности

Давайте назовем эту игру для вечеринок «Последовательности». В ней люди составляют неожиданные истории, «вкидывая» свои случайные персонажи, места действия и события. Например, такие:



Наверное, вы привыкли к спискам только на бумаге, однако можно составлять и электронные версии на компьютере.



Здесь мы покажем, как написать на Python версию этой игры, которая сама будет составлять случайные истории.

1. Откройте новый файл, присвойте ему имя и сохраните его. Создайте список по аналогии с этим:

```
woman = ["A scientist", "A queen", "A pirate", "A giant rabbit"]
```

Это лишь наш вариант. Вы можете включить в него и своих персонажей.

Эта программа выбирает одного из женских персонажей случайным образом.

2. Затем составьте список мужских персонажей.

```
man = ["a police officer", "an artist", "your grandfather", "a killer robot"]
```

3. Создайте списки и для остальных частей истории, такие как:

```
place = ["on Pluto.", "at the supermarket.", "in a spooky, bat-filled cave."]
sheWore = ["scuba diving gear.", "fairy wings.", "a paper bag."]
heWore = ["a purple suit.", "a shark costume.", "a beach towel."]
womanSays = ["'Who are you?'", "'How many beans make five?'", "'Why?'"']
manSays = ["'Beep boop!'", "'Don't eat frogs!'", "'What time do you call this?'"']
consequence = ["world peace.", "chaos.", "a giant foot squashed them.", "rainbows."]
worldSaid = ["'Errant nonsense!'", "'Cheese is trending now.'", "'I'm melting!'"]
```



4. Чтобы использовать списки в нашей истории, нам нужен **модуль random** (для выбора элемента из списка) и **цикл while** (для создания новой истории каждый раз, как мы нажимаем return).

Производим импорт **модуля random** и создаем **цикл while**.

```
import random  
while True:
```

Условие «while True» в начале **цикла while** заставляет компьютер проходить его бесконечное количество раз, пока вы сами не закроете программу.

5. Для начала истории вам надо подобрать персонажи и место действия, расположив их в нужной последовательности. Наберите такую строчку (она должна быть с отступом):

```
→ print(random.choice(woman), "met", random.choice(man), random.choice(place))
```

Здесь выбирается случайный элемент из списка «Женщины».

Здесь добавляется слово «встретила».

Здесь выбирается случайный элемент из списка «Мужчина».

Здесь выбирается случайный элемент из списка «Место».

6. Затем сделайте то же самое с другими списками.

```
→ print("She was wearing", random.choice(sheWore))  
→ print("He was wearing", random.choice(heWore))  
→ print("She said,", random.choice(womanSays))  
→ print("He said,", random.choice(manSays))  
→ print("The consequence was", random.choice(consequence))  
→ print("The world said,", random.choice(worldSaid))
```

7. Закончите программу вставкой пустой строки и просьбой нажать ввод для получения следующей истории. (Для остановки программы вам достаточно закрыть это командное окно.)

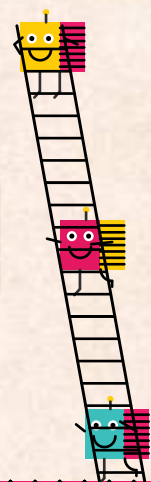
```
→ print()  
→ input("Press enter to play again.")  
→ print()
```

Цикл while делает паузу, и для продолжения истории пользователю надо нажать return.

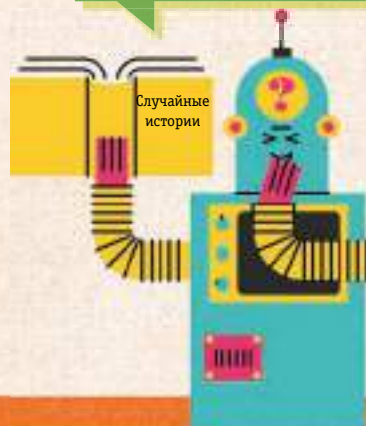
8. Сохраните и запустите программу, посмотрите, какие истории она генерирует. Если хотите, то можете поэкспериментировать со списками и создать свою версию этой игры.

RANDOM.CHOICE()

Если вы хотите случайным образом выбрать элемент из списка, вам потребуется функция **random.choice()**. Сам список мы помещаем в скобки.



«Королева встретила робота-убийцу на Плутоне. На ней был бумажный пакет...»



СЛОВАРИ

Словари, подобно спискам, также позволяют хранить информацию в среде языка Python. Каждая ячейка словаря идет в паре с меткой, называемой ключом (**key**). Ячейки вы просматриваете при помощи ключей.



Создание словаря

Словарь создается по аналогии со списком, только его элементы записываются в фигурных скобках. Например...

1. Для создания словаря, в котором будут храниться имена супергероев и их способности, откройте командное окно и наберите следующее:

```
>>> powers = {"The Pigeon": "flight", "Brainzar": "mind reader", "Cyborg": "controls machines"}
```

Это имя словаря.

Ключ всегда пишется перед ячейкой. (Здесь ключ это имя.)

Двоеточие соединяет каждый ключ с ячейкой, или, как говорят, ее значением. (У нас это перечень сверхспособностей.)

2. Теперь вы можете выводить ячейку (где в качестве значения суперспособности), в качестве ключа набирая имя. Если вы выберете героя голубя, то в качестве суперспособности получите «полет».

```
>>> print(powers["The Pigeon"])
```

Эта строка выводит на экран величину, ассоциированную с данным ключом.

Запятая разделяет сами элементы.

Компьютер на экран выведет следующее:

```
flight
```

3. Для того чтобы добавить в словарь новый элемент, наберите следующее:

```
>>> powers["Laser Girl"] = "shoots lasers"
```

В словарях отсутствует порядок перечисления элементов, так что каждый раз вы будете видеть их в различном порядке.

4. Также вы можете просмотреть весь словарь.

```
>>> print(powers)
```

Теперь он включает в себя элемент «Laser Girl».

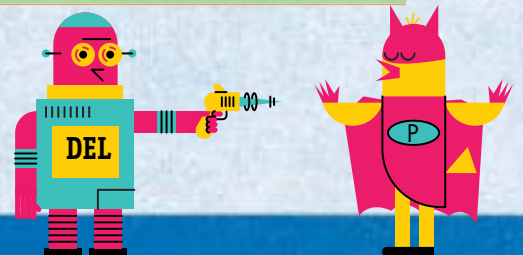
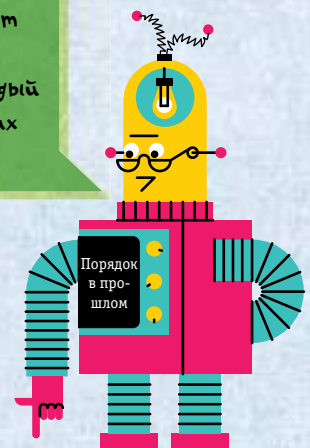
Если вы выполните эту команду, то увидите нечто вроде:

```
{'Cyborg': 'controls machines', 'Brainzar': 'mind reader', 'The Pigeon': 'flight', 'Laser Girl': 'shoots lasers'}
```

5. Также вы можете удалять элементы.

```
>>> del powers["The Pigeon"]
```

Если посмотрите словарь теперь, то голубя в нем не найдете.



6. Для изменения значения элемента словаря, сделайте следующее:

```
>>> powers["Brainzar"] = "seeing the future"
```

Если вы просмотрите словарь теперь, то у персонажа «Brainzar» появится новая способность.

Переводчик с инопланетного

С помощью структуры Словарь вы можете создать программу-переводчик с инопланетного языка.

1. Откройте и сохраните новую программу в программном окне. Затем начинайте создавать словарь.

```
alienDictionary = {"we": "vorag", "come": "thang", "in": "zon",  
"peace": "argh", "hello": "kodar", "can": "znak", "i": "az", "borrow": "liftit",  
"some": "zum", "rocket": "upgoman", "fuel": "kakboom", "please": "selpin",  
"don't": "baaaaaaaaaaarn", "shoot": "flabil", "welcome": "unkip",  
"our": "mandig", "new": "brang", "alien": "marangin", "overlords": "bap"}
```

2. Теперь попросите пользователя ввести для перевода какое-нибудь слово.

```
englishPhrase = input("Please enter an English word or phrase to translate: ")  
englishWords = englishPhrase.lower().split()
```

3. Далее создайте **список**, который будет использоваться для печати переведенных фраз, а в **цикле for** организуйте поиск внутри словаря, проверяя на совпадения английские слова.

```
alienWords = []  
for word in englishWords:  
    if word in alienDictionary:  
        alienWords.append(alienDictionary[word])  
    else:  
        alienWords.append(word)  
print("In alien, say: ", " ".join(alienWords))
```

Эта строка показывает собственно перевод на инопланетный язык.

Поставьте здесь один пробел.

Эта строка добавляет слова, которых не оказалось в словаре, к списку «alienWords» в непереверденном виде.

" " «.join() присоединяет все слова к списку «alienWords» в строку через пробел.

Сохраните и запустите программу, чтобы проверить ее.

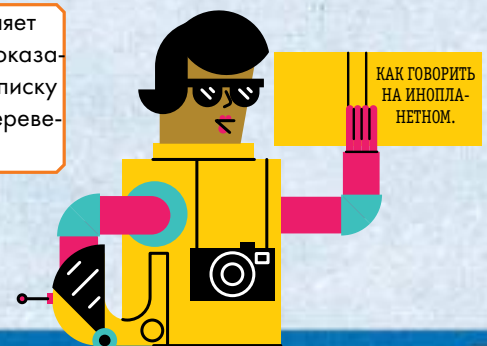
Набирайте все строчными буквами.

Оператор **lower()** превращает все, что вы введете, в написанное строчными буквами.

Оператор **split()** превращает строку в список слов.

Этот **цикл for** будет запускаться столько времени, сколько слов вы предложите программе для перевода.

Если напечатанные слова есть в словаре, то программа заглянет в значение ячейки (т.е. инопланетное слово) по ключу (английскому слову) и добавит слово к списку «alienWords».



ШИФРОВАННОЕ СООБЩЕНИЕ

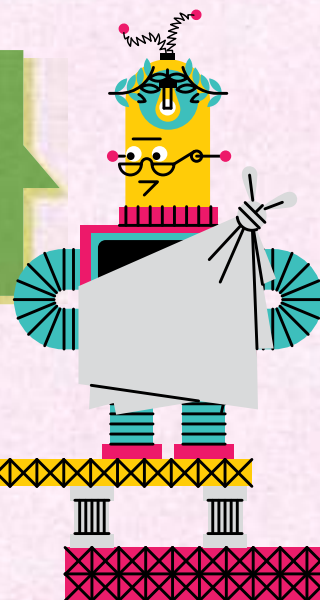
В разведке используется т.н. криптография, когда буквы в сообщении заменяются на другие. Соответственно, никто не может прочитать письмо без знания секретного правила замены букв. С помощью языка Python мы можем написать свою собственную шифровальную программу.

Шифр Цезаря

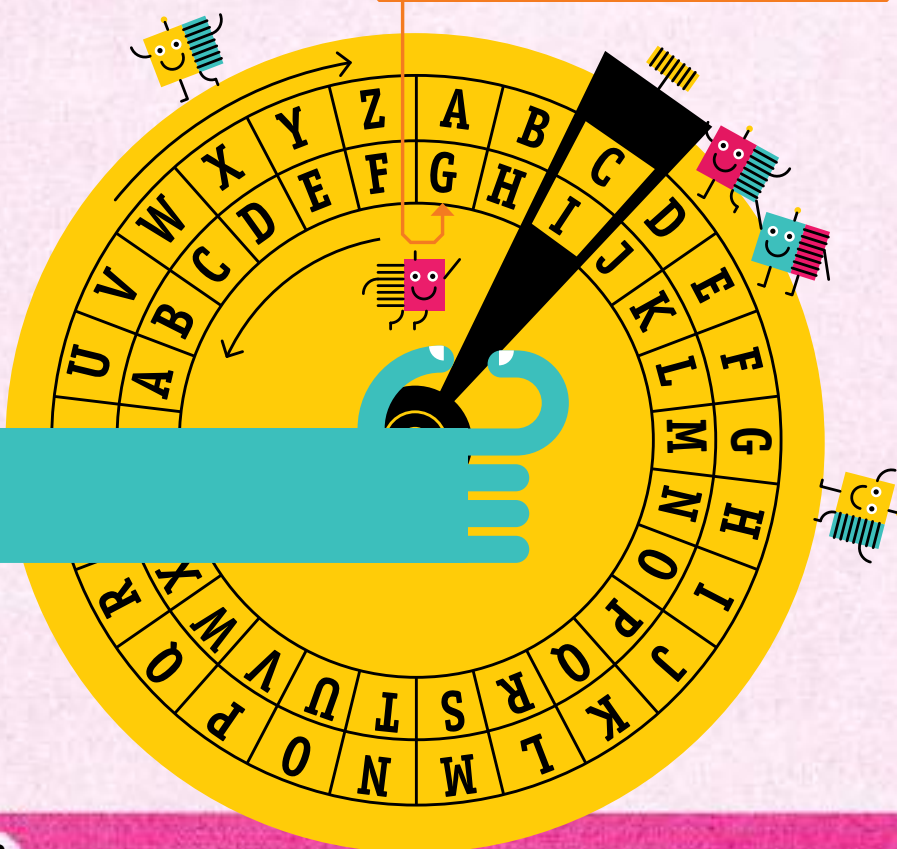
Шифр Цезаря — это метод зашифровывания сообщений, когда буквы заменяются другими, сдвинутыми по алфавиту на определенное количество мест. Ключом считается величина этого сдвига между буквами.

Например, если первая буква английского слова, которую вы хотите зашифровать, С и вы сдвигаете на 6 букв, то у вас получится буква I.

Есть легенда, что римский император Юлий Цезарь использовал этот шифр для отправки секретных приказов в войска.



Внутреннее кольцо показывает алфавит, сдвинутый на 6 букв. Попробуйте с его помощью раскодировать то, что зашифровал робот на рисунке ниже.

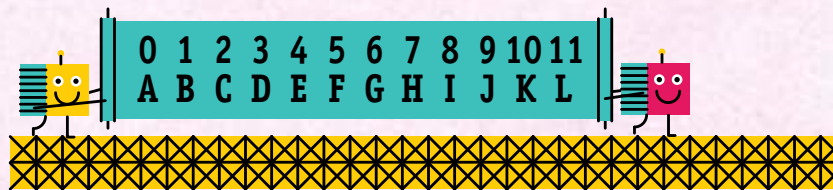


O NUVK TU KTKSE YVOKY
GXK XKGJOTM ZNOY
HUUQ, UX ZNKE'RR IXGIQ
UAX IUJK...



Создание своего шифра

1. Откройте новый файл и сохраните его. В первой строчке создайте строчную переменную, которая представляет собой целый алфавит, да еще напечатанный дважды.



```
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

Первый алфавит

Второй алфавит

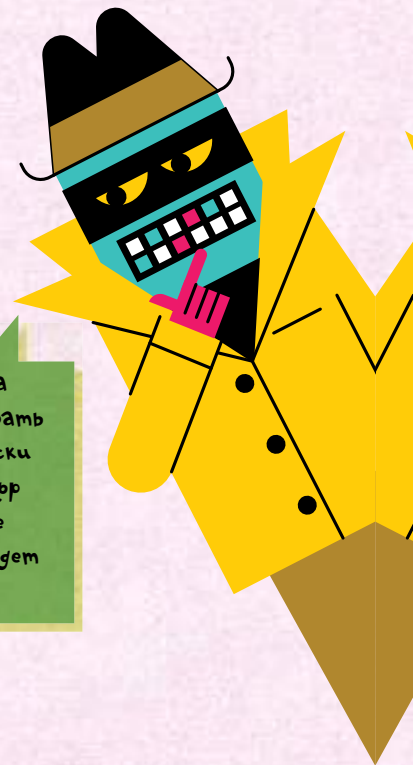
Каждая буква в строке имеет **индекс** — совсем как элементы в списке.

Второй алфавит необходим на тот случай, если после смещения вы доберетесь до Z.



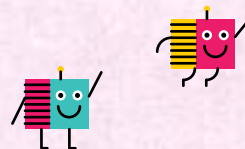
Например, если ваш ключ 10, а буква — Z, то вам придется считать от A до J во втором алфавите

Эти два алфавита позволяют вам выбирать ключ до 25. Технически до 26, но тогда шифр повторит исходное сообщение и оно не будет секретным



2. Далее попросите пользователя ввести сообщение.

```
stringToEncrypt = input("Please enter a message to encrypt: ")
```



3. На случай, если в сообщении будут строчные буквы, необходима функция **upper()**, которая превращает все буквы в прописные.

```
stringToEncrypt = stringToEncrypt.upper()
```

Эта команда превращает все строчные буквы в прописные, чтобы они соответствовали буквам в переменной «алфавит».

4. Затем попросите пользователя ввести цифру. Это и будет ключ для зашифрования вашего сообщения.

```
shiftAmount = int(input("Please enter a whole number from 1-25 to be your key."))
```

5. Теперь создаем пустую строку. Именно в ней мы будем хранить зашифрованное по итогу работы программы сообщение.

```
encryptedString = ""
```

Набрав пустые кавычки, вы создадите пустую строку.

6. Чтобы зашифровать сообщение, вам необходимо сдвигать по очереди каждую букву. Вы можете использовать для этого **цикл for**, обращаясь к каждой букве по индексу и вычисляя новый индекс.

```
for currentCharacter in stringToEncrypt:  
    position = alphabet.find(currentCharacter)  
    newPosition = position + shiftAmount  
    encryptedString = encryptedString + alphabet[newPosition]
```

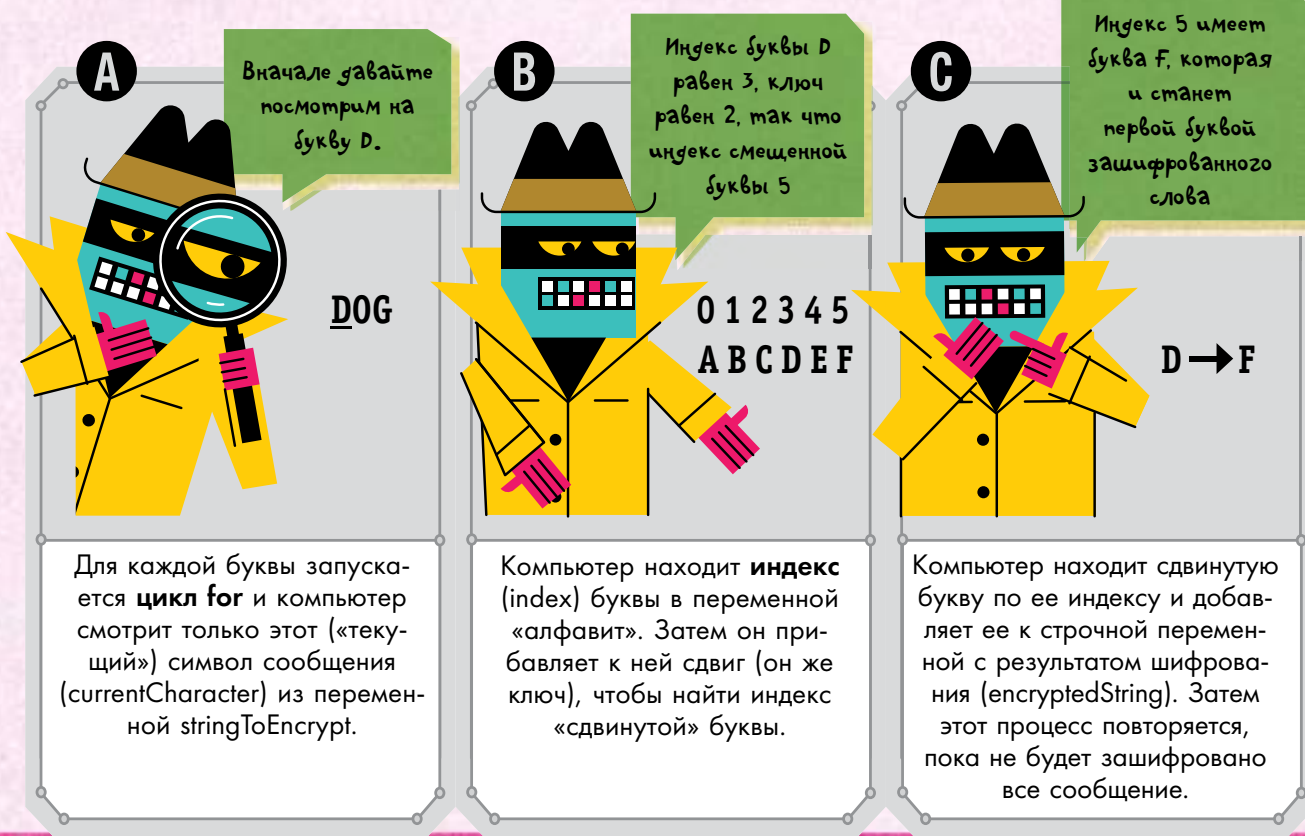
Цикл for проходит по разу для каждой буквы шифруемого сообщения.

Функция find() обращается к переменной «алфавит» для поиска первого вхождения текущего символа и возвращает индекс (index) этой буквы.

В этой строке находится новая буква. После смещения она тут же добавляется к зашифрованному сообщению.

К **индексу** вхождения каждой буквы мы тут же прибавляем смещение, или ключ.

Вот что происходит в приведенной выше программе. Представьте себе, что вы хотите зашифровать слово DOG (собака), а ключом у вас будет сдвиг 2.



7. Последняя строчка в программе не должна иметь смещения. Для компьютера это означает, что исполнить эту строку надо после окончания **цикла for**.

```
print("Your encrypted message is", encryptedString)
```

Ваше зашифрованное сообщение будет напечатано заглавными буквами.

8. Сохраните и запустите программу. Затем протестируйте ее, набрав слово и ключ. Например, если вы наберете «dog» и «2», то программа должна выдать следующее:

```
Your encrypted message is FQI
```

Добавляем пунктуацию

9. Если вы хотите, чтобы в шифрованные сообщения входили знаки пунктуации и символы, то добавьте их в строки, отмеченные ниже. Тогда цикл **for** должен выглядеть примерно так:

```
for currentCharacter in stringToEncrypt:
    position = alphabet.find(currentCharacter)
    newPosition = position + shiftAmount
    if currentCharacter in alphabet:
        encryptedString = encryptedString + alphabet[newPosition]
    else:
        encryptedString = encryptedString + currentCharacter
```

Эта строка должна иметь смещение относительно верхней строки с **if**.

Программная секция после **else** активируется, если символа нет в алфавите. Например, восклицательный знак.

Если символа нет в алфавите, то он добавляется к зашифрованному тексту как есть, без смещения.

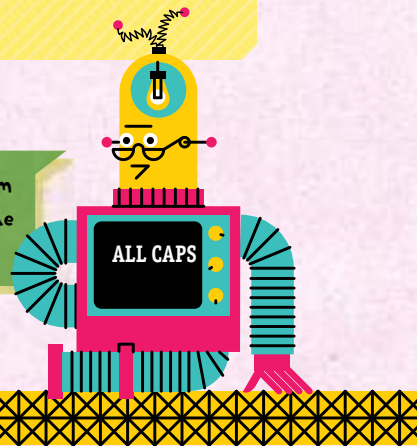
Расшифровка

Но зашифрованные сообщения также надо и расшифровывать. Чтобы расшифровать сообщение, вам потребуется та же программа и ключ расшифровки. Это тот же ключ шифрования, но только со знаком минус. Так, если мы зашифровывали сообщение с ключом-смещением 2, то для расшифровки вам необходимо ввести -2.

Попытайтесь подать на ввод программы зашифрованное сообщение с использованием ключа расшифровки. Например, если зашифрованное слово «FQI», то введя ключ расшифровки (-2), мы получим...

СОВЕТЫ ПО ОТЛАДКЕ

- Проверяйте сдвиги строк.
- Убедитесь, что нигде не перепутали «=» и «==».
- Убедитесь, что вы везде правильно написали имена переменных, включая соответствие по заглавным и строчным буквам. На стр. 88-89 вы найдете еще советы по отладке.



РИСУЮЩАЯ ЧЕРЕПАШКА

В среде языка Python вы можете создавать рисунки, т.н. графику, с помощью **модуля turtle**, в котором «зашит» большой набор функций для рисования фигур и линий.



Что такое turtle?

Черепашка как курсор перемещается по экрану и оставляет за собой линию. Она позволяет рисовать различного рода линии и даже рисунки. Для этого достаточно дать ей нужную команду.

Рисуем квадрат

Чтобы использовать черепашку, вам необходимо вначале импортировать модуль turtle. Короткая программа покажет вам, как импортировать этот модуль и использовать его для рисования квадрата.

1. Откройте новый файл и сохраните его. Затем загрузите модуль turtle. Примерно так:

```
from turtle import *
```

«import *» импортирует все команды модуля turtle, так что вам не придется набирать слово «turtle» перед каждой командой из этого модуля каждый раз, когда вы к ней обращаетесь.

2. Установите цвет, которым будет рисовать черепашка.

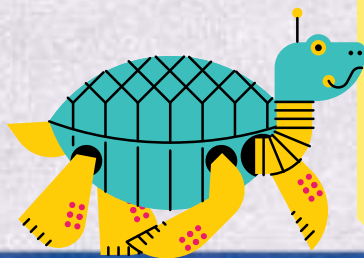
```
color("blue")
```

Обратите внимание, что слово «color» (цвет) надо использовать именно в американском написании.

3. На следующей строчке установите стиль отображения черепашки, используя для этого функцию shape (форма).

```
shape("turtle")
```

Вы можете менять стиль отображения рисующего курсора, выбирая его из списка стилей.



Вы можете выбрать для своей черепашки несколько различных стилей. Этот стиль так и назван — «turtle» (черепашка).



СТИЛИ ЧЕРЕПАШКИ

Здесь приведены другие доступные стили черепашки.

стрелка
круг
квадрат
треугольник
классический

ЦВЕТА ЧЕРЕПАШКИ

Черепашка не может быть абсолютно произвольного цвета, однако у вас есть небольшой выбор, куда входят некоторые весьма экзотичные оттенки. Вы можете попробовать любой из них.

аквамари́н	золотисто-берёзовый
циан	лимонный шифон
слива	розовый, барби-цвет
фиолетово-лавандовый	темное хаки
зеленый лайм	темно-оранжевый

4. Установите скорость движения черепашки, выбрав ее между 1 и 10 (10 — самая быстрая).

```
speed(10)
```

5. Вы можете устанавливать толщину линий, который оставляет за собой черепашка. Для этого есть функция **pensize** (размер пера).

```
pensize(4)
```

Этот параметр дает довольно толстую линию.

6. Теперь сообщим нашей черепашке, куда ей ползти.

```
forward(50)
```

```
right(90)
```

```
forward(50)
```

```
right(90)
```

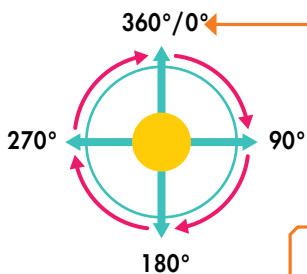
```
forward(50)
```

```
right(90)
```

```
forward(50)
```

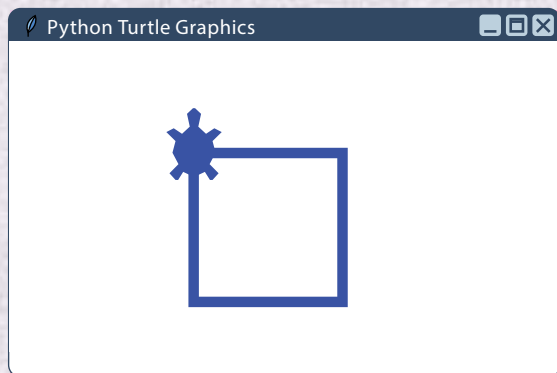
Черепашка перемещается вперед на 50 пикселей (см. справа).

Черепашка поворачивается по часовой стрелке на 90 градусов.



360 градусов соответствуют полному обороту.

7. Сохраните и запустите программу. Вы увидите следующее:



ПЕРВАЯ ЧЕРЕПАШКА

Название «turtle» пришло от имени робота, который перемещался по полу и рисовал фигуры ручкой, закрепленной в хвосте. Первый язык программирования, которым управлялась черепашка, назывался LOGO.



ПИКСЕЛИ

Расстояние, которое проходит черепашка, измеряется в пикселях — точках, которые образуют рисунок на экране монитора.



ДРУГИЕ ФИГУРЫ

Если вы измените программу на 6 шаге, то получите другие фигуры.

Треугольник

```
forward(50)
left(120)
forward(50)
left(120)
forward(50)
```

Восьмиугольник

Воспользуйтесь циклом **for**, чтобы нарисовать фигуру с 8 сторонами.

```
for x in range(8):
    → forward(50)
    → right(45)
```

Попытайтесь нарисовать фигуру самостоятельно, сочетая приведенные ниже функции (прямо, назад, левый и правый поворот) и добавляя свои значения в скобках:

```
forward()
```

```
left()
```

```
backward()
```

```
right()
```

Черепашка рисует снежинку

Можно сделать так, чтобы при помощи циклов и функций рисования черепашка рисовала снежинку, поскольку там применяются повторяющиеся алгоритмы.

Размещая инструкции рисования внутри функций, вы можете просто вызывать эти функции, что освободит вас от монотонного перепечатывания этих инструкций.

1. Откройте новый файл и настройте свою черепашку, как делали это ранее.

```
from turtle import *  
shape("turtle")  
speed(10)  
pencolor("white")  
pensize(6)
```

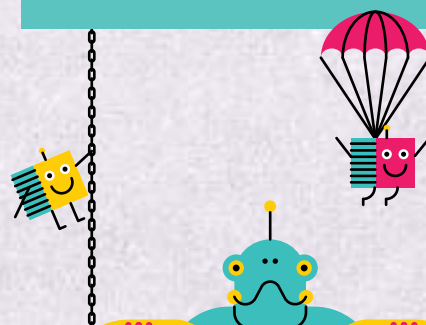
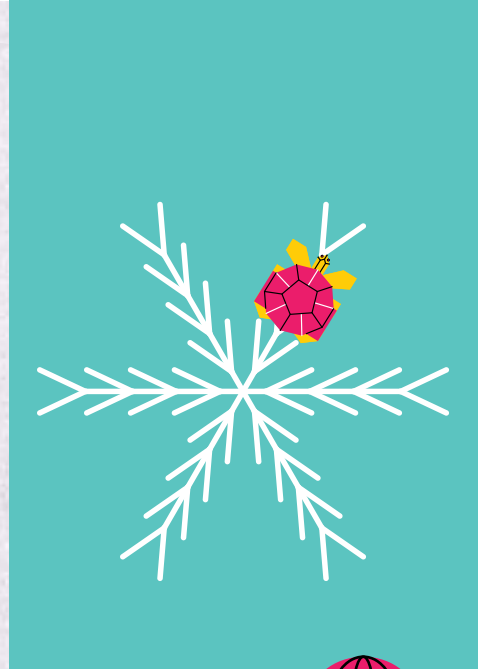
Мы выбрали белый цвет потому, что рисуем снежинку.

2. Установите цветной фон для графического экрана, по которому будет двигаться черепашка, определив для него «черепаховый» цвет.

```
Screen().bgcolor("turquoise")
```

Screen() это функция, которая вызывает графический экран, по которому движется черепашка.

Bgcolor() это функция, которая устанавливает цвет фона для экрана с вашей черепашкой.

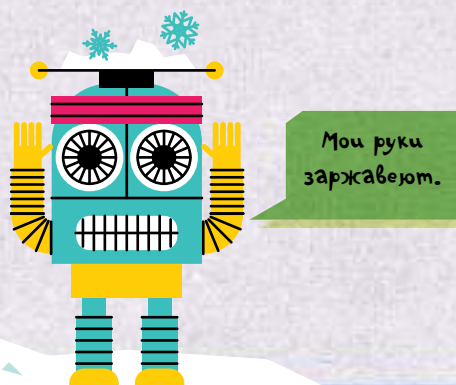
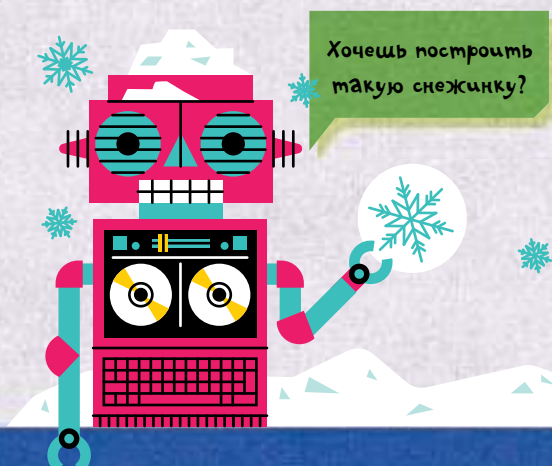


ЧЕРЕПАШКА ПРЯЧЕТСЯ

Если вы не хотите, чтобы черепашка была видна, то можете заставить ее стать невидимой функцией **hideturtle**.

Добавьте эту функцию в конце настройки на шаге 1.

```
hideturtle()
```



3. Теперь мы будем писать функцию `vshape`, которая нарисует нам часть снежинки в форме буквы «V».

```
def vshape():
```

Двоеточие сообщает компьютеру, что после него следует определение новой функции. Строки инструкций, которые идут далее, ВСЕГДА имеют смещение.

`def` сообщает компьютеру о вашем желании определить (англ. define) пользовательскую функцию.

Это **имя функции**. Оно будет отображаться **синим** цветом (а не **темно-фиолетовым** или **черным**, как предопределенные функции), поскольку его определяет сам пользователь.

4. Введите инструкции, выполнение которых приводит к получению «V». (Как они работают, вы можете посмотреть справа на диаграмме А.)

```
→ right(25)
→ forward(50)
→ backward(50)
→ left(50)
→ forward(50)
→ backward(50)
→ right(25)
```

Эта инструкция заставляет черепаху двигаться обратно по линии, которую она только что нарисовала.

Эта инструкция поворачивает черепаху снова мордой вперед.

5. Теперь давайте напишем функцию, превращающую созданную нами фигуру (V) в ответвления снежинки.

```
def snowflakeArm():
```

6. Введите эти инструкции. (На рисунке C и D показано, как работают эти функции.)

```
→ for x in range(0,4):
→     forward(30)
→     vshape()
→     backward(120)
```

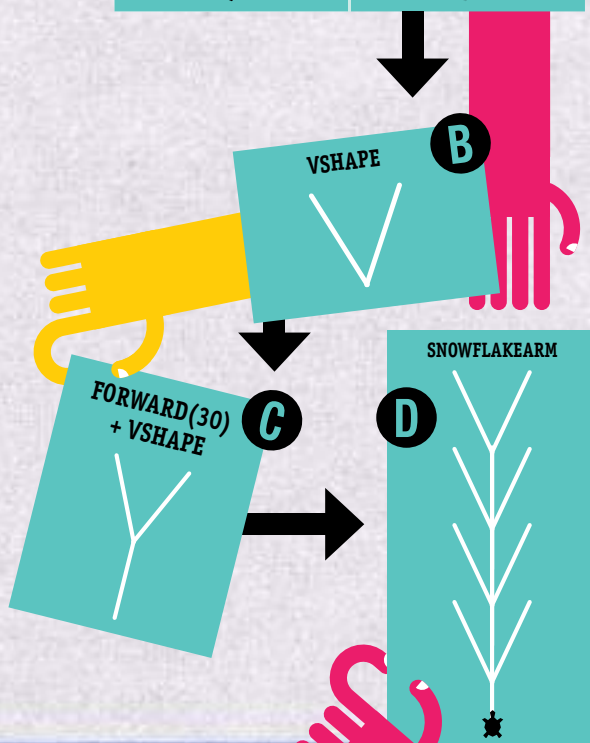
Этот **цикл for** пройдет четыре раза, создавая линию с четырьмя фигурами «V».

Эта инструкция возвращает черепаху обратно в точку старта.



ОПРЕДЕЛЯЕМ ПОЛЬЗОВАТЕЛЬСКУЮ ФУНКЦИЮ

В языке Python зашито много готовых к использованию функций, таких как `print()` и `input()`. Но вы также можете создавать и свои, пользовательские функции при помощи **ключевого слова** `def` (от английского define, или «определять»).



7. Создайте еще одну функцию, которая уже рисует снежинку целиком. Для этого можно использовать **цикл for**, в котором повторяется процедура рисования елочек.

```
def snowflake():
    for x in range(0,6):
        snowflakeArm()
        right(60)
```

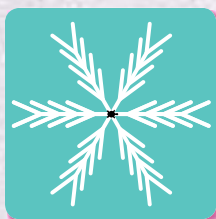
Эта инструкция заставит черепашку повернуть на 60 градусов перед рисованием следующей елочки.

Этот **цикл for** пройдет шесть раз и создаст шесть елочек.

8. Наконец, вызываем функцию **showflake()**

```
snowflake()
```

Сохраните и запустите свою программу. Вы увидите, как появится графическое окно, и черепашка начнет рисовать снежинку, такую как эта:

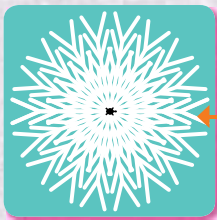


Разнообразим нашу снежинку

Вы можете создавать различные варианты снежинок, подставляя различные значения на шаге 7.

Вот как можно создать снежинку из 18 елочек:

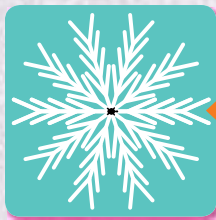
```
for x in range(0,18):
    snowflakeArm()
    right(20)
```



В снежинке с 18 елочками поворот между елочками составляет 20 градусов ($20 \times 18 = 360$ градусов).

Вот так мы создадим снежинку с 10 елочками:

```
for x in range(0,10):
    snowflakeArm()
    right(36)
```



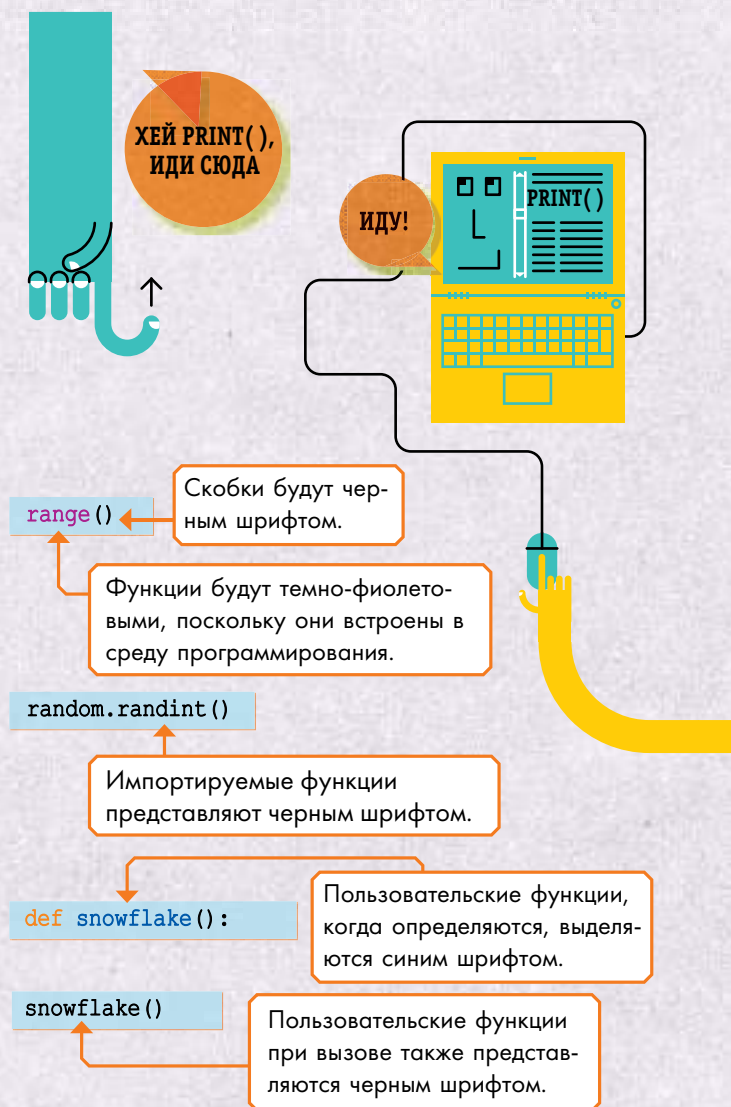
В снежинке с 10 елочками поворот между елочками составляет 36 градусов ($10 \times 36 = 360$ градусов).



Фантастически удобные функции

Функции широко используются в языке Python. Вот некоторые правила использования функций:

- Имена функций сопровождаются парой скобок. Мы используем, или, как говорят, вызываем, функции, просто набирая их имя и не забывая про скобки. Строго говоря, это есть **вызов функции**.
- Программная среда Python оборудована множеством встроенных функций, таких как **print()** или **range()**. Эти функции выделяются темно-фиолетовым цветом.
- Ко многим функциям мы получаем доступ, импортируя программные **модули** — по сути, библиотеки уже готовых функций. Примеры таких библиотек — **random** и **turtle**. Импортируемые библиотеки и функции из них отображаются **черным** шрифтом.
- Также вы можете создавать свои собственные функции при помощи **def** (см. стр. 49). Примером такой функции служит **snowflakeArm()**. Она будет выделена синим шрифтом в момент определения и станет черной, когда вы будете вызывать ее. Такие пользовательские функции легко получить из уже написанных программ. Эта уловка позволяет разбивать длинные программы на ряд коротких, благодаря чему структура проекта становится более ясной.



- Скобки после функции предназначены для **параметров** — переменных, которые вы хотите «дать» функции для использования. Когда вы вызываете функцию, то внутри скобок должны задать ей требуемый набор величин. Эти величины называют аргументами. Например, **range(0, 6)** имеет аргументы 0 и 6.
- Иногда скобки оставляют пустыми. Функция с пустыми скобками имеет постоянную задачу. Например, **snowflakeArm()** на предыдущей странице.
- Иногда функции возвращают некую величину. Обработав определенную информацию, они выдают результат, или, как говорят, возвращают его обратно в программу, чтобы та могла использовать его. Так, функция **range(0, 6)** возвращает список [0, 1, 2, 3, 4, 5]. Эта **функция** не только выполняет вычисления, но также и возвращает результат, так что ничего кроме вызова набирать не надо.

Цветные снежинки

Чтобы сделать снежинки разноцветными, можно воспользоваться **модулем random**.

1. Откройте программу построения снежинок и сохраните ее под новым именем. Затем добавьте эту строчку на самый верх.

```
import random
```

2. Уберите строчку, в которой вы устанавливаете цвет пера, поскольку мы будем использовать различные цвета.

```
pencolor("white")
```

Уберите эту инструкцию.

3. Теперь вам надо составить список цветов и поставить его сразу после строчки, где вы устанавливаете цвет фона.

```
colours = ["blue", "purple", "cyan", "white", "yellow", "green", "orange"]
```

Для большей красочности вы можете включить и другие цвета (см. список на стр. 46).

4. Затем подбирайте цвет из этого списка для каждой из елочек, из которых состоит снежинка.

```
def snowflake():  
    for x in range(0,6):  
        color(random.choice(colours))  
        snowflakeArm()  
        right(60)
```

Эта строчка новая. Убедитесь, что она имеет смещение.

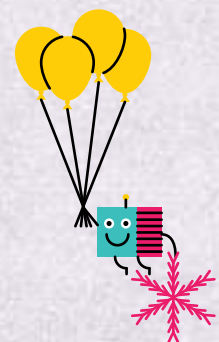
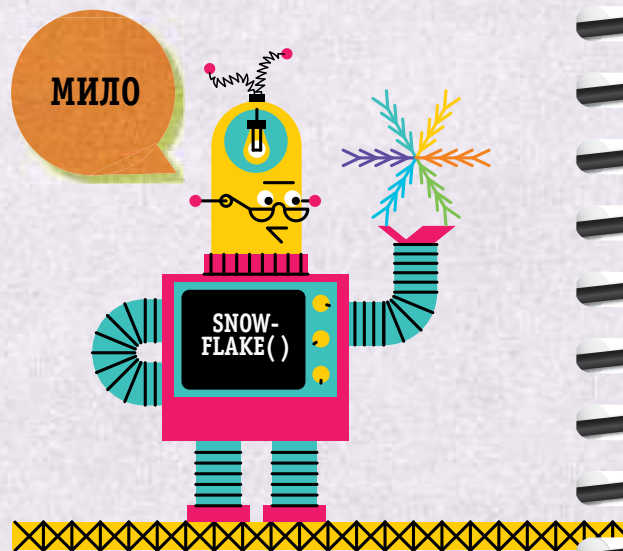
Эта функция случайным образом подбирает цвет из вашего списка.

5. Сохраните и запустите программу. Вы увидите, что снежинки стали разноцветными.

Создаем пургу

Также с помощью модуля random вы можете заполнить экран снежинками различных размеров.

1. Сохраните программу, создающую разноцветные снежинки, под новым именем.



2. Чтобы варьировать размер снежинок, нам необходимо добавить переменную **size** (размер) в вашу функцию **snowflake()**.

```
def snowflake(size):  
    for x in range(0,6):  
        color(random.choice(colours))  
        snowflakeArm(size)  
        right(60)
```

Переменная **size** каждый раз идет в скобках.

Если вы хотите, чтобы все части снежинки были одного цвета, переместите эту инструкцию вверх за **цикл for**.

3. Также вам надо добавить новую переменную **size** в функции **vshape()** и **snowflakeArm()**, заменив ими расстояния, на которые происходит смещение вперед и назад, таким образом:

```
def vshape(size):  
    right(25)  
    forward(size)  
    backward(size)  
    left(50)  
    forward(size)  
    backward(size)  
    right(25)
```

```
def snowflakeArm(size):  
    for x in range(0,4):  
        forward(size)  
        vshape(size)  
        backward(size*4)
```

Вам надо умножить последний размер на 4, чтобы черепашка возвратилась в начальную точку.

Не меняйте этот угол поворота.

4. Наконец, замените последнюю строку программы, поставив **цикл for**. Так программа будет рисовать несколько снежинок случайных размеров, каждая из которых будет появляться в произвольном месте.

```
for i in range(0,10):  
    size = random.randint(5,30)  
    x = random.randint(-400,400)  
    y = random.randint(-400,400)  
    penup()  
    goto(x,y)  
    pendown()  
    snowflake(size)
```

Установите количество снежинок на графическом экране.

Эта инструкция говорит компьютеру рисовать снежинки с использованием переменной **size**.

Эта инструкция выбирает произвольные **координаты** для появляющихся снежинок (см. вставку справа).

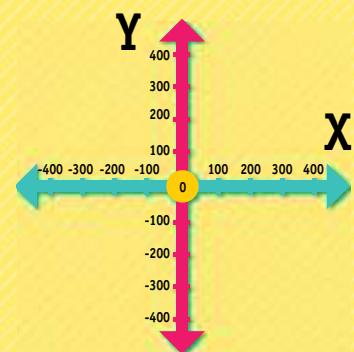
Эта инструкция выбирает произвольное значение размера снежинки.

5. Сохраните программу и запустите ее. Вы должны получить целое облако снежинок.



ИСПОЛЬЗОВАНИЕ КООРДИНАТ

Любую точку на экране вы можете описать при помощи **координат**. **Координата X** сообщает компьютеру, насколько левее или правее располагается точка относительно начальной. **Координата Y** — насколько выше или ниже она находится.



В модуле **turtle** есть функция **goto(x, y)** для прямого перехода по определенным координатам. С ее помощью вы можете перескакивать туда, не оставляя следа. Для этого надо лишь поднять перо, т.е. вбить команду **penup()**. Если вы опустите перо командой **pendown()**, то черепашка снова начнет оставлять след.

КНОПКА-СЮРПРИЗ

В этой программе вы будете создавать кнопку, которая будет выдавать разные сообщения при нажатии на нее.

Продумываем дизайн

Чтобы в среде Python запрограммировать на экране кнопку, нам потребуется модуль **tkinter**. Этот подгружаемый модуль является набором инструментов, которые позволят вам создавать изображения, или графику, на экране. А вся эта совокупность органов управления программой называется графическим пользовательским интерфейсом (см. справа).

1. Откройте новый файл и сохраните его. Импортируйте модуль **tkinter**.

```
import tkinter
```

2. Далее вам надо заставить компьютер создать окно для кнопки.

```
window = tkinter.Tk()
```

Мы окно назвали просто «окном» (window).

3. Если вы сохраните эту программу и запустите ее, то появится пустое окно.

4. Теперь в окне надо создать кнопку. Это можно сделать при помощи функции **Button()** из модуля **tkinter**.

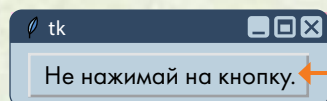
```
button = tkinter.Button(window, text="Do not press this button.", width=40)
```

5. Теперь нам надо сообщить компьютеру, чтобы он запаковал кнопку в окно модуля **tkinter**, для чего используется функция **pack** (запаковать).

```
button.pack(padx=10, pady=10)
```

pad добавляет пространство вокруг кнопки. В нашем случае мы добавляем по 10 пикселей с обеих сторон вдоль осей X и Y.

6. Сохраните и запустите программу.

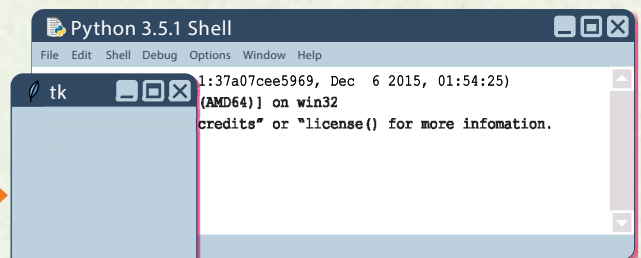


Теперь вы увидите кнопку.

GUI — ГРАФИЧЕСКИЙ ПОЛЬЗОВАТЕЛЬСКИЙ ИНТЕРФЕЙС

Графический пользовательский интерфейс (GUI) — это принятый в вычислительной технике термин, обозначающий кнопки, иконки и другие элементы, которые используются для управления программами.

Когда кто-то нажимает на кнопку на экране кликом мышки, — это называется событием (event).



...и появляется пустое окно.

На кнопке появится это сообщение.button.

Эта инструкция устанавливает ширину кнопки (40 пикселей).

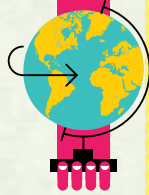
Кнопке всегда нужно окно, иначе она не будет видна.



7. Добавьте переменную, чтобы отслеживать, сколько раз вы нажмете на эту кнопку.

```
clickCount = 0
```

Счетчик следует установить на ноль.



ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ

Глобальные (**global**) переменные могут быть использованы в любой части программы. Переменные, создаваемые внешними функциями, всегда глобальны. Доступ извне к переменным внутри функции можно получить, используя служебное слово **global** (глобальный). Если вы просто создаете переменную внутри функции, то она будет доступна только внутри этой функции.

8. Теперь создайте функцию, которая скажет компьютеру, что делать после каждого события — нажатия на кнопку.

```
def onClick(event):
```

```
→ global clickCount
```

```
→ clickCount = clickCount + 1
```

```
→ if clickCount == 1:
```

```
→ button.configure(text="Seriously? Do. Not. Press. It.")
```

```
→ elif clickCount == 2:
```

```
→ button.configure(text="Gah! Next time, no more button.")
```

```
→ else:
```

```
→ button.pack_forget()
```

Слово **global** позволит этой функции использовать переменную, которую вы сейчас создаете (см. вставку).

Каждый раз, как вы кликаете по кнопке, компьютер увеличивает счетчик на единицу.

В зависимости от того, первое это нажатие или второе, вы будете получать разные сообщения.

Секция кода после **else** запускается, когда вы жмете на кнопку в третий раз.

pack_forget() делает так, что программа «забывает» об ассоциации кнопки с окном и кнопка исчезает.

9. Чтобы программа заработала, необходимо вначале связать (**bind**) вашу функцию с самим актом нажатия и отпускания кнопки.

Это цифра 1, а не буква «l».

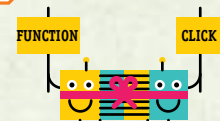
```
button.bind("<ButtonRelease-1>", onClick)
```

10. Добавьте строку, чтобы заставить работать все написанное.

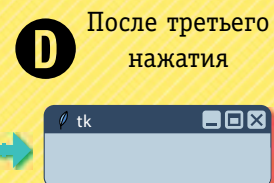
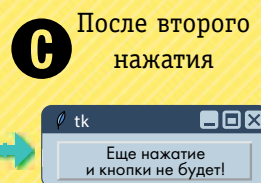
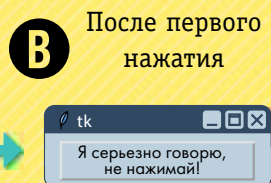
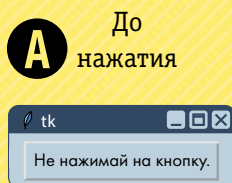
```
window.mainloop()
```

Эта инструкция запускает программу, привязанную к данному окну.

11. Сохраните и запустите программу. Вы получите кнопку с меняющимися сообщениями, как на картинках.



Люди! Вы никогда не выполняете инструкций, не так ли?



СОЗДАЕМ ШЕДЕВР

Эта программа позволит вам рисовать, перемещая указатель мыши по экрану.

Холст для картины

В модуле **tkinter** вы найдете функции, с помощью которых сможете создавать **холст** (canvas) — пустой экран с **координатами X** и **Y**, которые позволят отслеживать положение рисующего пера.

1. Откройте и сохраните новый файл, затем импортируйте **tkinter**.

```
import tkinter
```

2. Добавьте пару строк с операторами **print()** и текстом, объясняющим, как пользоваться программой.

```
print("To draw, hold down the left mouse button and move your mouse around.")  
print("To change your brush colour, click on one of the colours.")
```

Этот текст будет появляться в командном окне при запуске программы.

3. Теперь нам необходимо сказать компьютеру, чтобы он создал холст.

```
window = tkinter.Tk()  
canvas = tkinter.Canvas(window, width=750, height=500, bg="white")  
canvas.pack()
```

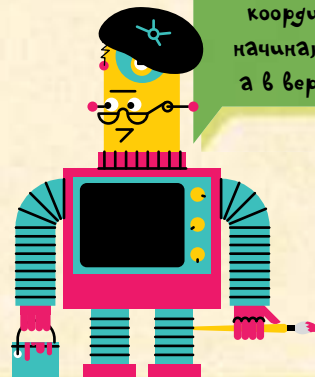
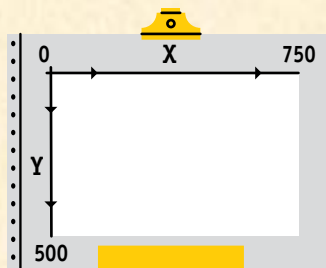
Эта инструкция создает окно средствами модуля **tkinter** и присваивает окну имя.

pack() помещает холст в окно модуля **tkinter**.

Эти **аргументы** задают размер холста в пикселях, а также его цвет.

Эта команда создает чистый холст в окне модуля **tkinter**.

В отличие от координат графического окна, обе координаты на холсте начинаются не в центре, а в верхнем левом углу.



4. Создайте две переменные, в которых будут храниться последние координаты (X и Y) указателя мыши. (Мы их создаем одновременно, разделяя запятой.) В начале работы эти переменные равны нулю. Также мы устанавливаем начальный цвет нашей кисти.

```
lastX, lastY = 0,0  
colour = "black"
```

Это верхний левый угол холста.

5. Теперь создадим функцию `store_position()`, которая будет отслеживать положение указателя мыши. Так мы сможем рисовать линии, перемещая указатель.

```
def store_position(event):  
    → global lastX, lastY  
    → lastX = event.x  
    → lastY = event.y
```

Эта инструкция позволяет получить доступ к переменным `lastX` и `lastY`, созданным на 4 шаге.

Эти инструкции отслеживают координаты X и Y движущегося указателя мыши.

6. Далее мы создаем функцию, которая сообщает компьютеру, что делать при щелчке (клике) мышью на поверхности холста. Сохранять положение указателя мыши в момент клика мы будем при помощи функции `store_position()`.

Параметром для этой функции будет «событие» (`event`), под которым понимается клик мышью.

```
def on_click(event):  
    → store_position(event)
```

Эту функцию мы создали на предыдущем шаге.

7. Определите еще одну функцию, которая будет рисовать линию при перемещении указателя мыши по холсту.

Эта инструкция вызывает функцию `create_line()` и активирует ее на холсте.

Опция `fill` позволит установить цвет линии.

Опция `width` позволит установить толщину линии в пикселях (3 пикселя это довольно тонкая карандашная линия).

```
def on_drag(event):  
    → canvas.create_line(lastX, lastY, event.x, event.y, fill = colour, width = 3)  
    → store_position(event)
```

Эта инструкция записывает положение указателя мыши, когда вы заканчиваете перемещение.

Эти переменные хранят координаты указателя в момент последнего клика или остановки (стартовая позиция).

`event.x` и `event.y` хранят текущие координаты X и Y, т.е. координаты точек, через которые проходит указатель мыши.

ПРОДОЛЖЕНИЕ
НА СЛЕДУЮЩЕЙ
СТРАНИЦЕ.

I ♥
BLACK

8. При помощи функции **bind()** свяжите функции **on_click()** и **on_drag()** с кликом и перемещением указателя мыши по холсту соответственно. (См. вставку.)

```
canvas.bind("<Button-1>", on_click)
canvas.bind("<B1-Motion>", on_drag)
```

Эта инструкция связывает клик по левой кнопке мыши с функцией **on_click()**.



Запись **<B1-Motion>** означает, что имеет место какое-либо движение мыши в то время, пока нажата и удерживается левая кнопка мыши. Эта инструкция связывается с вызовом функции **on_drag()**.

9. Добавьте строчку, которая запускает программу целиком. Затем сохраните и запустите программу несколько раз, чтобы протестировать ее. Вы сможете рисовать на холсте, но только черным цветом.

```
window.mainloop()
```

Эта инструкция должна стоять в самом конце программы, так что проследите, чтобы все остальное было НАД ней.

10. Чтобы рисовать другими цветами, вы можете добавить палетку с квадратиками, по которым можно кликать мышью. Чтобы создать и расположить эту палетку на холсте, вам уже нужно использовать координаты.

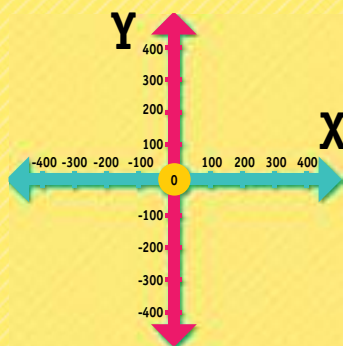
СВЯЗЫВАНИЕ

Модуль **tkinter** включает в себя ряд функций, которые позволяют связывать (**bind**) действия вашего программного кода с реакцией графического холста.

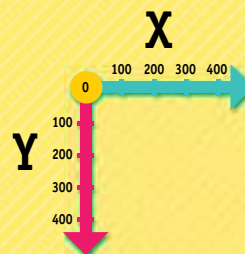
canvas.bind() связывает функции с событиями, такими как клик указателем мыши по холсту.

canvas.tag_bind() связывает события с объектами (например, фигурами) на холсте.

Вы уже использовали **координаты графического окна** в программе с **рисующей черепашкой** (см. стр. 53). Координаты холста во многом схожи с ними за исключением положения точки начала координат.



Координаты окна модуля **turtle**



Координаты холста модуля **tkinter**

Координаты холста растут при движении направо и вниз



Добавьте квадратики и установите их координаты таким образом:

Каждый квадратик получит свой идентификационный номер (**id**), по которому впоследствии квадратик можно будет идентифицировать (см. вставку).

Каждый квадрат описывается координатами **X** и **Y**.

Левый Верхний Правый Нижний
(x) (y) (x) (y)

Опция **fill** сообщает компьютеру, каким цветом окрасить каждый из квадратов.

```
red_id = canvas.create_rectangle(10, 10, 30, 30, fill="red")
blue_id = canvas.create_rectangle(10, 35, 30, 55, fill="blue")
black_id = canvas.create_rectangle(10, 60, 30, 80, fill="black")
white_id = canvas.create_rectangle(10, 85, 30, 105, fill="white")
```

ID

Когда вы создаете объект, такой как форма, на холсте, компьютер присваивает ей свой идентификатор **id**, чтобы отслеживать его. Например, **red_id** — это **id** вашего красного квадратика.

11. Для создания красок нам потребуются отдельные функции для каждого цвета на палетке, как показано ниже:

```
def set_colour_red(event):  
    → global colour  
    → colour="red"
```

Параметром этой функции является **событие**, а именно щелчок по красному квадратику.

```
def set_colour_blue(event):  
    → global colour  
    → colour="blue"
```

Служебное слово **global** позволяет вам здесь присваивать значение глобальной переменной **colour**, которую вы определили выше.

```
def set_colour_black(event):  
    → global colour  
    → colour="black"
```

```
def set_colour_white(event):  
    → global colour  
    → colour="white"
```

Белый цвет используется для стирания нарисованного.

12. Вы можете воспользоваться функцией **tag_bind()** из модуля **tkinter()**, чтобы связать щелчок мышью по любому из квадратов палетки с соответствующей функцией **set_colour**.

```
canvas.tag_bind(red_id, "<Button-1", set_colour_red)  
canvas.tag_bind(blue_id, "<Button-1", set_colour_blue)  
canvas.tag_bind(black_id, "<Button-1", set_colour_black)  
canvas.tag_bind(white_id, "<Button-1", set_colour_white)
```

Так компьютер распознает щелчок левой кнопки мыши.

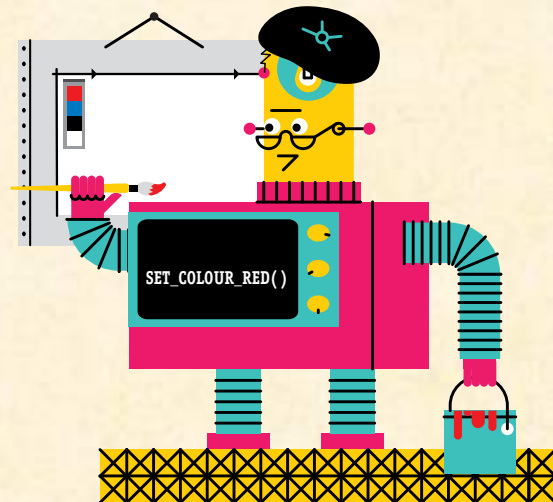
Эти инструкции вызывают одну из функций **set_colour**, которые вы создавали на предыдущем шаге.

tag_bind() используется для связи события (**event**) с объектом (**object**). Например, клика мышью по одному из квадратов с определенной функцией.

13. Сохраните и запустите программу для тестирования. Если при этом появятся сообщения об ошибках, то проверьте правильность написания служебных слов, пунктуацию и сдвиги строк. (На стр. 88–89 вы найдете еще советы по отладке.) Затем попытайтесь нарисовать что-нибудь интересное.

КОПИЯ ЭКРАНА

Если вы хотите сохранить свой рисунок, то нажмите клавишу **PrtScr (Print Screen)** на клавиатуре. Компьютер сохранит все, что изображено в настоящее время на экране монитора. Затем откройте программу редактирования изображения (графический редактор) и нажмите **Ctrl+v (Control+v)**, чтобы вставить скопированное изображение. (На MacOS для копирования экрана надо нажать **Command**, а затем **Shift+3**. Чтобы вставить изображение, надо открыть **Preview** и нажать **Control+v**.) Теперь вы можете сохранить изображение в виде рисунка.



САПЕР

Берегись! В этой игре под зеленым полем спрятаны мины.
Сможете ли вы найти все безопасные квадраты,
не взорвавшись на mine?

Игра

Здесь вы найдете краткое описание игры.
Игровое поле поделено на квадратики.

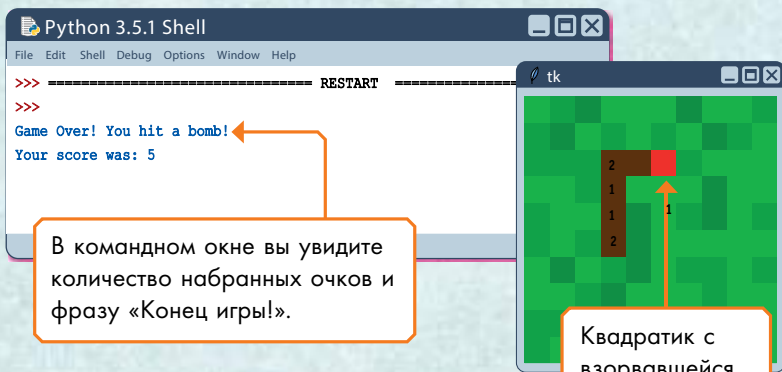
1. Когда вы запускаете игру, на фоне командного окна всплывает игровое поле.

Буквы «tk» показывают, что это окно было создано средствами модуля **tkinter**.



Полем в этой игре служит сетка 10x10 квадратов со спрятанными в начале игры минами.

3. Если вы выбираете клетку с миной, то квадратик краснеет и вы проигрываете. Игра заканчивается.



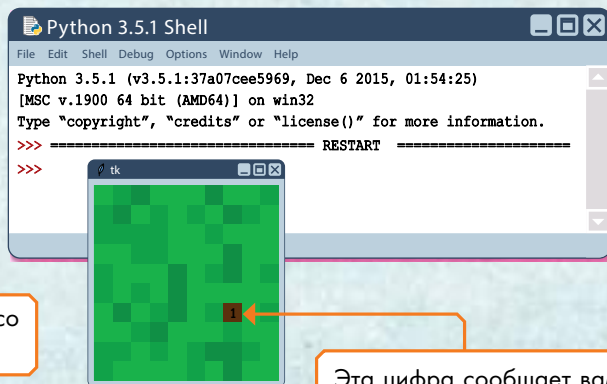
В командном окне вы увидите количество набранных очков и фразу «Конец игры!».

Квадратик с взорвавшейся миной

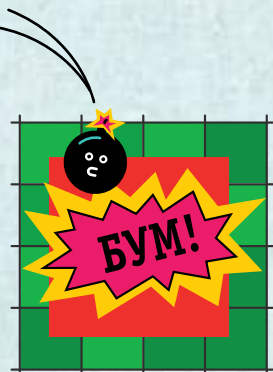
ВНИМАНИЕ!

Программы в этой и следующей играх (Теннис) довольно сложны, и процесс их создания основывается на всем том материале, который вы изучили ранее. Так что убедитесь, что у вас нет пробелов в знаниях, прежде чем приступать к программированию этих двух игр.

2. Чтобы проверить, заминирован ли квадратик, вы кликаете по нему указателем мыши. Если поле безопасно, оно становится коричневым и на нем появляется цифра.



Эта цифра сообщает вам, сколько мин заложено на соседних восьми клетках.

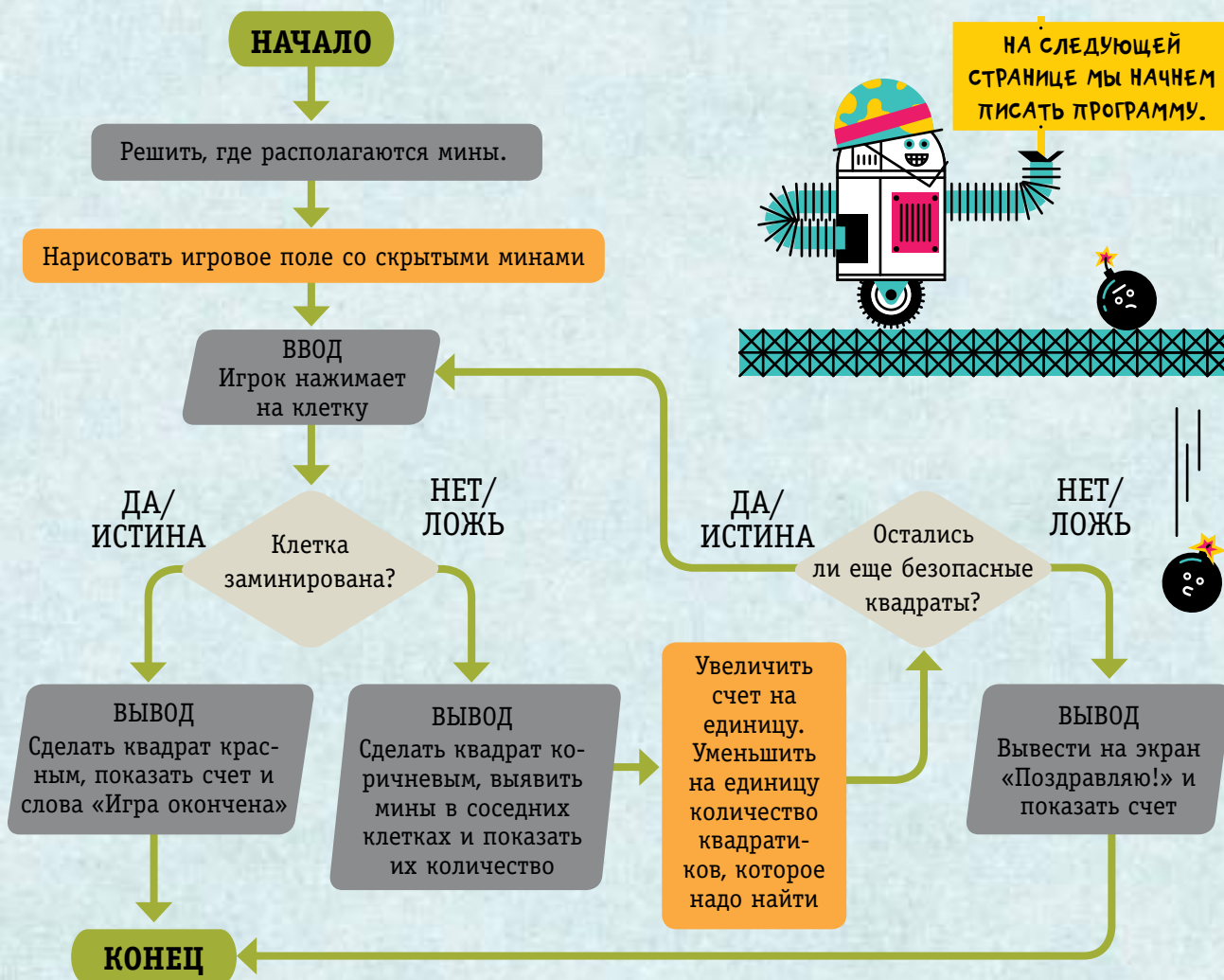
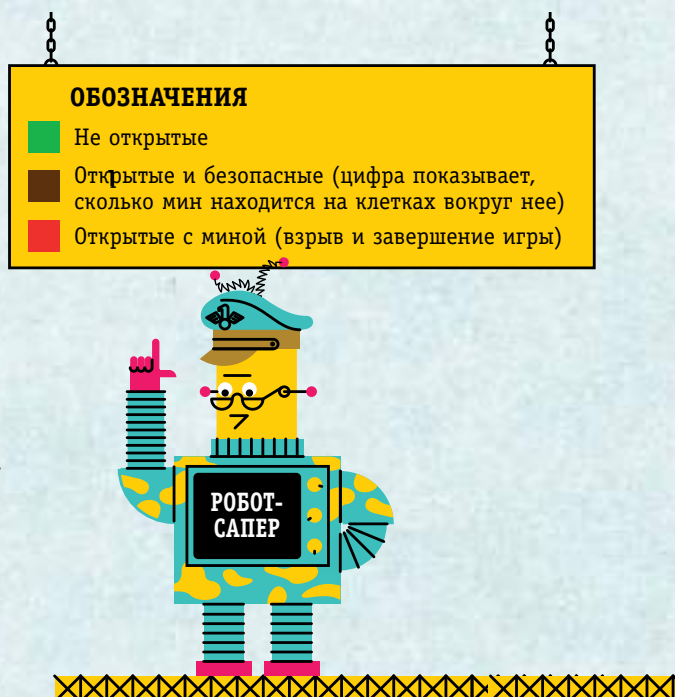


Как работает игра?

Чтобы игра работала как надо, программа должна:

- решить, где располагаются мины;
- нарисовать игровое поле на экране;
- реагировать, когда игрок делает щелчок по квадрату, изменяя цвет либо на красный, если мина, либо на коричневый с цифрой, если клетка безопасна (в квадрате тогда указывается, сколько мин расположено на соседних клетках);
- отслеживать, сколько очков набрал игрок (одна открытая безопасная клетка — одно очко);
- закончить игру и показать счет, если игрок взорвался;
- закончить игру и поздравить игрока, если он открыл все безопасные квадраты.

Ниже мы выразили это в виде блок-схемы.



Создаем игровое поле

Для того чтобы нарисовать поле, нам необходимо использовать модуль **tkinter**. Также нам потребуется модуль **random**, чтобы случайным образом разложить мины.

1. Откройте программное окно и сохраните в новый файл. Затем импортируйте модули **tkinter** и **random**.

```
import tkinter
import random
```

2. Далее нам необходимо создать несколько переменных, которые позволят отслеживать, окончена ли игра, каково количество очков (т.е. число открытых квадратов) и сколько квадратов осталось открыть.

```
gameOver = False
score = 0
squaresToClear = 0
```

Переменная «gameOver» сообщит компьютеру, когда игра закончена. Для начала и продолжения игры она должна равняться «False» (Ложь).

3. Теперь вы должны сказать компьютеру, что собственно нужно делать. (Если вы начинаете написание программы с плана, т.е. блок-схемы, то вам не придется беспокоиться о порядке, в котором вы определяете функции.)

```
def play_bombdodger():
    create_bombfield(bombfield)
    window = tkinter.Tk()
    layout_window(window)
    window.mainloop()
```

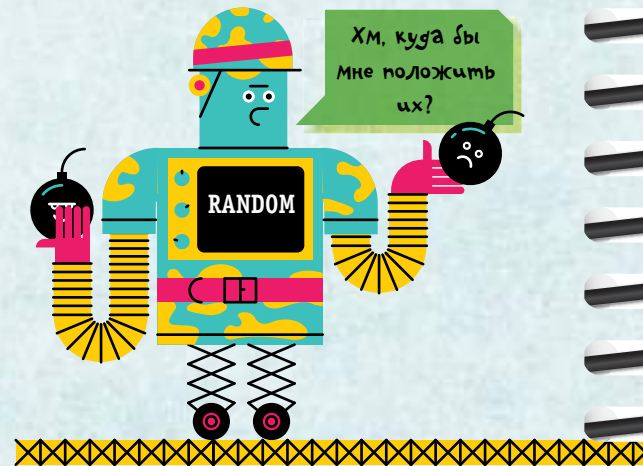
Когда вы играете в игру, компьютер...

...создает минное поле...

...вызывает графическое окно модуля **tkinter** и рисует собственно минное поле...

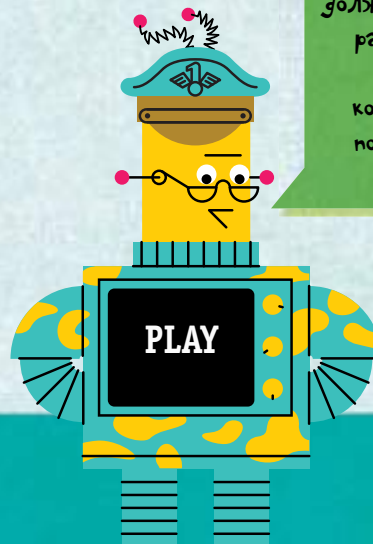
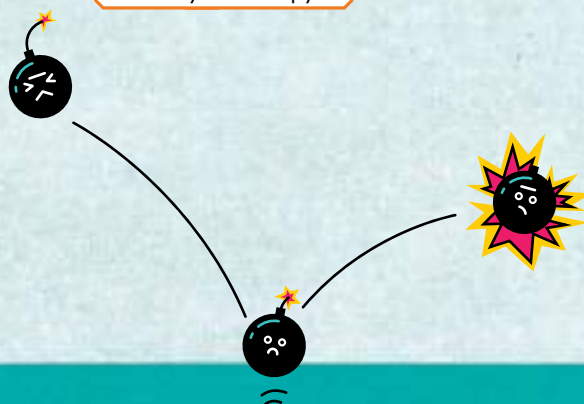
...и запускает игру.

Функции внутри `play_bombdodger()` должны быть определены ранее. Эта функция просто говорит компьютеру, в какой последовательности их вызывать



TRUE И FALSE

В программировании для принятия решений используются логические значения **true** (истина) и **false** (ложь). Не забывайте, что компьютер может отвечать лишь на простые вопросы, предусматривающие ответ да/нет. Это так называемая **булева логика** (см. стр. 16).



Планирование минного поля

Чтобы компьютер не забывал, где расположены мины, мы используем таблицу из чисел: **1** — там, где мина и **0** — где безопасный квадрат. Организовать эту таблицу можно при помощи нескольких **списков (list)**.

Мы уже использовали для хранения информации простые списки. Для более сложно организованных данных (как здесь) проще всего использовать списки внутри списка. Первый список (bombfield) включает в себя перечень строк. Каждая строка является, по сути, другим списком (rowList), включающим информацию по клеткам для каждого столбца в данном ряду.

4. Мы начинаем с пустого списка, а затем создаем функцию, которая заполняет его. Та же функция потребуется для того, чтобы обновлять информацию о том, сколько клеток вам осталось открыть.

```
bombfield = []  
def create_bombfield(bombfield):  
    → global squaresToClear
```

Эта инструкция создает список bombfield (минное поле).

Служебное слово **global** сообщает компьютеру, что нужно использовать не локальную (внутреннюю) переменную, а глобальную (внешнюю), которую вы определили в самом начале игры.

5. Теперь мы завершим создание функции. Создадим вложенный цикл. Внешний **цикл for** должен создавать десять пустых списков (строк). Затем добавим внутренний **цикл for**, который каждый раз будет запускаться однократно, заполняя клетки нулями и единицами. (Вы можете использовать генератор случайных чисел, чтобы решать, ноль будет или единица.)

```
→ for row in range(0,10):  
    → rowList = []  
    → for column in range(0,10):  
        → if random.randint(1,100) < 20:  
            → rowList.append(1)  
        → else:  
            → rowList.append(0)  
            → squaresToClear = squaresToClear + 1  
    → bombfield.append(rowList)
```

Эта инструкция присоединяет сформированный список rowList к изначальному «внешнему» списку.

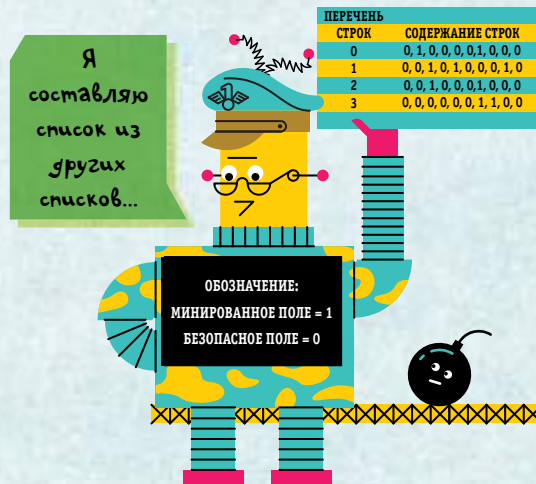
...в противном случае программа добавляет в список rowList ноль (безопасная клетка) и увеличивает на единицу счетчик squaresToClear (число клеток, которое предстоит открыть).

Этот **цикл for** запускается десять раз...

...и создает за раз пустой список для каждой строки.time.

Внутри каждого списка другой **цикл for** добавляет десять цифр (нулей и единиц) для клеток ряда.

Эта инструкция дает случайное число от 1 до 100. Если число меньше 20, то компьютер добавляет в список rowList единицу (мина)...



Кстати, вы можете оставлять в программе пустую строку перед каждой строчкой, в которой вы начинаете определять новую функцию. Это вам поможет, если вы будете разделять длинную программу на несколько коротких



6. Чтобы увидеть списки расставленных мин (это весьма удобно для отладки и проверки работоспособности кода), имеет смысл добавить функцию `print()`.

→ `printfield(bombfield)` ←

Когда эта инструкция выполняется, в командное окно выводится список расставленных мин.

(Если вы оставите эту отладочную строчку в игровой версии, то будете видеть карту минного поля в командном окне.)

7. Затем определите функцию, которая будет структурированно представлять содержание сложного списка.

```
def printfield(bombfield):  
    → for rowList in bombfield:  
    →     → print(rowList) ←
```

Эта инструкция отображает каждый список **rowList** в виде отдельных строк.

8. Сохраните и запустите программу для ее тестирования. Добавьте строчку вызова пользовательской функции `play_bombdodger`, затем сохраните и запустите программу. Эта строчка должна быть последней (самой нижней) в программе.

`play_bombdodger()` ←

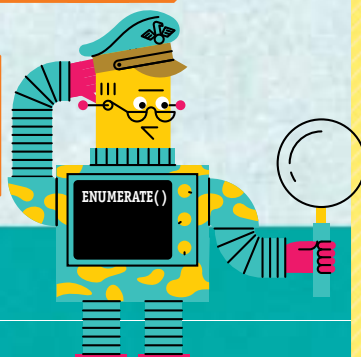
Вставьте код из пункта 9 **ВЫШЕ** этой строчки кода.

Вы увидите, как появится пустое окно модуля `tkinter`, а в командном окне образуется сетка с нулями и единицами. Также вы увидите сообщение об ошибке, которую и устраним уже на следующем шаге.

9. Далее нам необходимо расположить квадраты в окне на экране, определив для этого функцию `layout_window()`. Начните с того, что скажите компьютеру считывать элементы сложного списка минированных полей, используя для этого **цикл for** и функцию `enumerate()`. (См. ниже.)

```
def layout_window(window):  
    → for rowNumber, rowList in enumerate(bombfield):  
    →     → for columnNumber, columnEntry in enumerate(rowList): ←
```

Функция `enumerate()` индексирует элементы каждого списка `rowList`, т.е. создает пары «индекс — элемент». Это облегчает компьютеру работу со списками по частям.



ТЕСТИРОВАНИЕ ПРОГРАММЫ

Если программа длинная, то неплохо было бы тестировать ее по мере написания. Добавьте инструкцию `print()` в некоторых местах программы, чтобы видеть, что делает ваша программа, и отслеживать ошибки.

Когда закончите процесс тестирования, добавьте в начале программы инструкцию `printfield()`. Это остановит программу для выдачи **комментария** (comment) — примечания, которое программист оставляет для других разработчиков программы.

Пустое окно создано средствами `tkinter`.

Ноль это безопасное поле, а единица — мина.

>>> ===== RESTART =====

```
>>>  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]  
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0]  
[0, 1, 0, 0, 0, 1, 0, 0, 0, 0]  
[0, 0, 1, 1, 0, 0, 0, 0, 0, 0]  
[0, 0, 0, 1, 0, 0, 1, 0, 0, 0]  
[0, 0, 0, 0, 0, 1, 1, 0, 0, 1]  
[1, 0, 0, 0, 0, 0, 0, 0, 0, 1]  
[0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0]
```

Traceback (most recent call last):
File "C:\Users\Ziggy\My Documents\Dodge the bombs.py", line 45, in <module>
 play_bombdodger()
File "C:\Users\Ziggy\My Documents\Dodge the bombs.py", line 18, in play_bombdodger
 layout_window(window)
NameError: name 'layout_window' is not defined
>>>

Эта ошибка возникла потому, что мы пока еще не определили функцию `layout_window()`.

То самое всплывающее окно, которое программа создает средствами модуля `tkinter` при запуске.

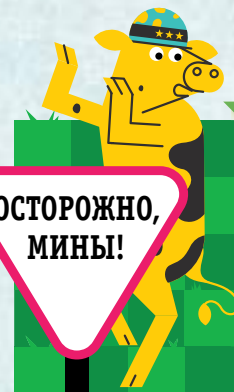
Внешний **цикл for** проходит по перечню строк сверху вниз.

Внутренний **цикл for** проходит по колонкам отдельной строки.

ENUMERATE()

`enumerate()` — функция, которая проходит по каждому элементу списка и создает связь индексов элементов с их содержанием (нумерует списки).

10. Теперь давайте создадим функцию, которая случайным образом генерирует цвет для каждого квадрата. Также нам следует добавить инструкцию, обеспечивающую постоянные размеры квадратиков.



Эти оттенки зеленого помогут вам различать отдельные квадратики минного поля

Секция if

Если случайное число меньше 25, то квадратик будет темно-зеленым.

Секция elif (else if)

Если случайное число больше 75, то квадратик будет цвета морской волны.

Секция else

Если не выполняются два верхних условия, то квадрат будет просто зеленым.

→ → → if random.randint(1,100) < 25:

→ → → square = tkinter.Label(window, text = "

→ → → elif random.randint(1,100) > 75:

→ → → square = tkinter.Label(window, text = "

→ → → else:

→ → → square = tkinter.Label(window, text = "", bg = "green")

", bg = "darkgreen")

", bg = "seagreen")

bg означает «фоновый цвет».

Параметры разделяются запятыми.

tkinter.Label()

создает ссылки на квадратики минного поля.

Связывает квадратики с окном **tk window**, которое вы создали ранее.

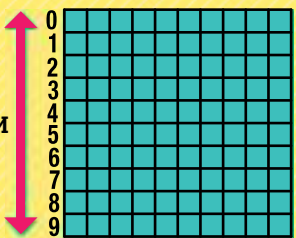
В кавычках напечатайте четыре пробела. Это будет ширина квадрата.

11. Теперь необходимо сообщить компьютеру, как расположить квадратики поля в сетке, которую вы видите в окне **tkinter**. Для этого потребуется новая функция **grid()**.

→ → → square.grid(row = rowNumber, column = columnNumber)

Точка «.» означает, что функция **grid()** связана с квадратом (**square**).

Функция **grid()** имеет много встроенных параметров. Для позиционирования квадрата нам нужны параметры **строка** (**row**) и **столбец** (**column**).



СТРОКИ

СТОЛБЦЫ



В нашей сетке 10 строк и 10 столбцов.

СОЗДАНИЕ СЕТКИ

Grid() разделяет окно, созданное средствами модуля **tkinter**, на сетку из столбцов и строк. Как только вы создадите сетку, вам станут доступны и другие функции работы с сеткой, такие как... **grid_info()** с помощью сетки систематизирует информацию, придавая ей структуру словаря (**dictionary**). Зная строку и столбец, намного проще быстро найти нужную информацию.

12. Сохраните и запустите программу. Проверьте ее работоспособность. Убедитесь, что в самом конце у вас имеется такая инструкция:

```
play_bombdodger()
```

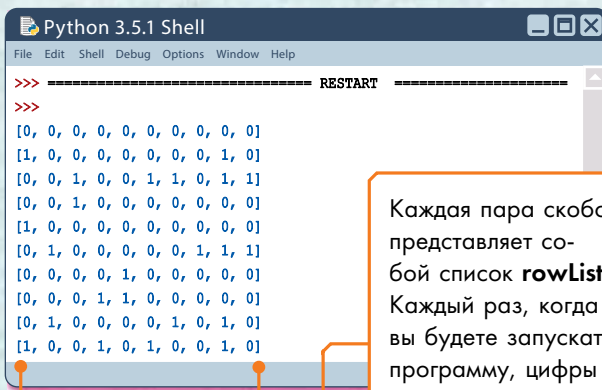
На экране должно появиться такое окно:



Это окно создано процедурами из модуля **tkinter**.

При каждом следующем запуске программы будет формироваться другой узор из зеленых квадратов.

Также в командном окне вы увидите карту минного поля.



Каждая пара скобок представляет собой список **rowList**. Каждый раз, когда вы будете запускать программу, цифры в строчках будут меняться.

13. Если программа работает, то строчку с **print()** на шаге 6 смело убирайте, поскольку там будут подсказки.

```
#printfield(bombfield)
```

Добавьте знак «#» перед строчкой (это называется комментированием). Такая строчка теперь считается **комментарием** и не исполняется.

Эта инструкция должна находиться в конце программы, поскольку компьютер считывает программу строго сверху вниз. Мы должны вначале определить все операции, а затем уже запускать программу

PLAY

Не переживайте, если получите сообщение об ошибках. В таких больших программах, как эта, они неизбежны

Возвратитесь назад и исправьте ошибки. Это уже само по себе хороший способ изучения языка программирования!

ОТЛАДКА

Если вы получите сообщение об ошибке, то там будут указаны и строки, которые следует проверить. Если вы не можете обнаружить ошибку, то просто проверьте строчку над строкой с ошибкой.

- Все ли скобки вы закрыли? Если скобка пропущена, то среда Python сообщит об этом в следующей строке по мере чтения кода.
- Правильно ли вы написали все служебные слова?
- Правильно ли вы расставили все смещения строк?

На странице 71 и на стр. 88–89 вы найдете еще советы по отладке программ.

Легко побеждать, когда знаешь карту!

КАРТА

Реагирование на клик мышью

Теперь, когда у нас есть минное поле, нам необходимо запрограммировать реакцию на клик мышью по квадратику. Это называется событием (**event**), и в модуле **tkinter** заложены специальные команды, с помощью которых компьютер распознает различного рода события.

14. Вы можете научить компьютер реагировать на щелчок левой кнопкой мыши при помощи команды **<Button-1>**. Также вам надо сказать компьютеру, что именно делать при наступлении события, и связать все это с определенным квадратиком. Это означает, что нам придется добавить еще одну строку из шага 11, которая будет отвечать за размещение квадратов.

Эту строчку вы уже использовали на шаге 10.

```
→ square.grid(row = rowNumber, column = columnNumber)
→ square.bind("<Button-1>", on_click)
```

bind() связывает событие (**event**), такое как клик мышью, с реакцией компьютера и координатой квадратика (для этого мы используем функцию **on_click()**).

Это «1» (один), а не английская буква «l».

Каждый раз, когда вы кликаете мышью, запускается функция **on_click()**. Мы создадим ее на следующем шаге.

15. Последняя созданная нами функция нужна для того, чтобы сказать компьютеру, что делать, когда кликают по квадрату. При этом используются переменные, которые вы создали ранее.

```
def on_click(event):
→ global score
→ global gameOver
→ global squaresToClear
```

Под событием (**event**) понимается щелчок указателем мыши по квадрату.

Команда **global** позволяет данной функции изменять эти переменные, несмотря на то, что они были созданы вне ее.

16. Вначале нам необходимо убедиться, что компьютер знает, по какому квадрату кликнули. Для этого нам нужно новое слово — **widget** (вошедшее в русский язык слово «виджет», исходно означавшее «штучка»). Его мы представляем для обозначения объекта, по которому кликаем мышью. Затем нам надо получить координаты квадрата, по которому кликнули.

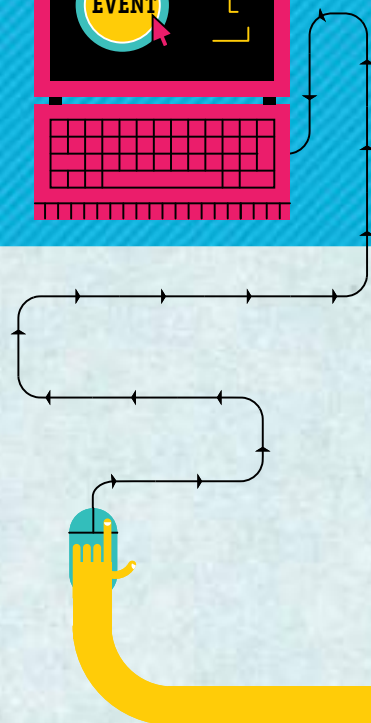
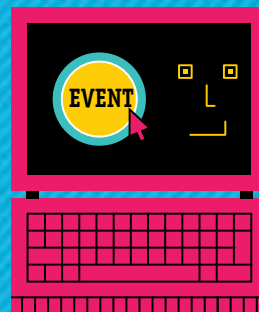
```
→ square = event.widget
→ row = int(square.grid_info()["row"])
→ column = int(square.grid_info()["column"])
```

Эта инструкция создает переменную «**square**» (квадрат) для описания объекта, на котором происходит событие. (В данном случае квадрата, по которому щелкнули мышью.)

square.grid.info() выводит информацию о квадрате сетки (а именно, строку и столбец). **int()** делает так, что компьютер работает с информацией, как с числовыми данными.

СОБЫТИЕ

В программировании событием (**event**) считается все, что должна распознавать программа. Обычно под этим подразумевается сигнал от пользователя, такой как клик мышью или нажатие на клавишу клавиатуры.



17. Затем надо сделать так, чтобы компьютер проверил текст, привязанный к квадрату (даже если это только пробелы). Для этого нам нужна новая функция — `cget()`.

```
→ currentText = square.cget("text")
```

Эта новая переменная отвечает за появляющийся в квадратике текст.

`cget("text")` просматривает имеющийся текст.

ПРОСМОТР КОНФИГУРАЦИИ

В модуле `tkinter` предусмотрена функция `cget()`, которая позволяет просматривать конфигурацию. Точнее, некоторые из установок данного модуля.

18. Теперь в вашей игре не хватает только реакции на клик указателем мыши по клетке. Вам надо сообщить компьютеру, что делать, если пользователь наткнется на мину.

```
→ if gameOver == False:
→   if bombfield[row][column] == 1:
→     gameOver = True
→     square.config(bg = "red")
→     print("Game Over! You hit a bomb!")
→     print("Your score was:", score)
```

Эта инструкция проверяет, есть ли в данной клетке мина (1 – мина, 0 – безопасная клетка).

Эта инструкция заканчивает игру.

Эта инструкция делает квадратик красным.

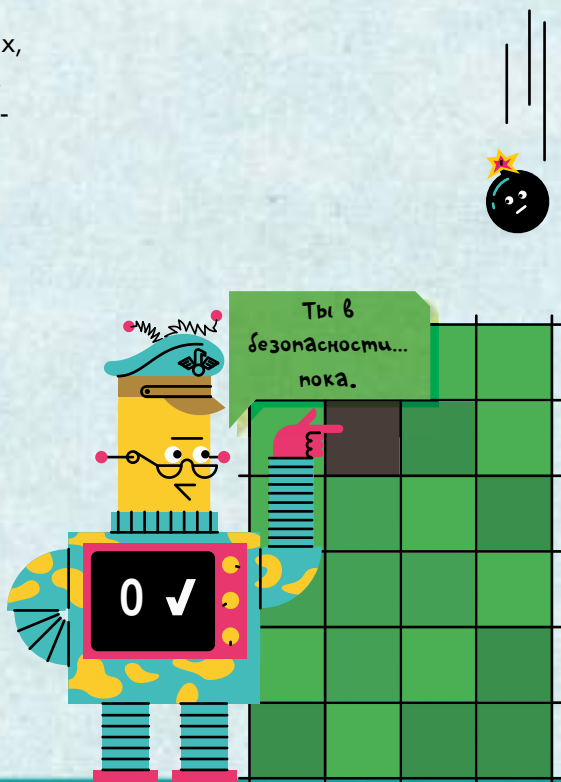
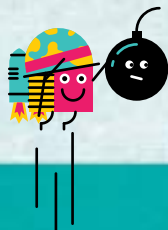
Эти строчки выводят в командное окно сообщение о проигрыше и количество очков, т.е. квадратиков, открытых до взрыва.

19. Далее сообщите компьютеру, что делать, если пользователь открыл безопасный квадрат. Во-первых, нам надо проверить, не щелкали ли по нему ранее. Если нет, то квадрат надо сделать коричневым и показать количество мин в окружающих клетках.

```
→ elif currentText == " ":
→   square.config(bg = "brown")
→   totalBombs = 0
```

Чтобы сосчитать количество мин вокруг этого квадрата, необходимо создать новую переменную и придать ей нулевое значение. (Считать количество мин мы будем на следующем шаге.)

Если в квадрате по-прежнему четыре пробела, значит, по нему пока еще не кликали.



20. Чтобы определить количество мин на соседних полях, вам нужно по очереди проверить все восемь квадратов.

Мы начнем с нижнего квадрата.

Если вы на нижнем ряду (строка 9), вам не нужно проверять строчку ниже.

```

→ → → if row < 9:
→ → → if bombfield[row+1][column] == 1:
→ → → totalBombs = totalBombs + 1
    
```

Если на нижнем поле мина, компьютер прибавит единицу к общему количеству мин.

Теперь проверим верхний квадрат.

```

→ → → if row > 0:
→ → → if bombfield[row-1][column] == 1:
→ → → totalBombs = totalBombs + 1
    
```

Если вы в верхнем ряду (строка 0), то эта проверка пропускается.

Далее проверим левый квадрат.

```

→ → → if column > 0:
→ → → if bombfield[row][column-1] == 1:
→ → → totalBombs = totalBombs + 1
    
```

Если вы уже в левом столбце (столбец 0), то эта проверка пропускается..

Далее проверим правый квадрат.

```

→ → → if column < 9:
→ → → if bombfield[row][column+1] == 1:
→ → → totalBombs = totalBombs + 1
    
```

Если вы уже в правом столбце (столбец 9), то эта проверка пропускается.

После этого проверим верхний левый угол.

```

→ → → if row > 0 and column > 0:
→ → → if bombfield[row-1][column-1] == 1:
→ → → totalBombs = totalBombs + 1
    
```

Если вы уже в верхней строке (строка 0) или в левой колонке (колонка 0), то эта проверка пропускается.

После этого проверим нижний левый угол.

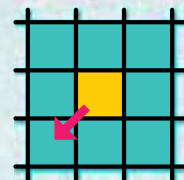
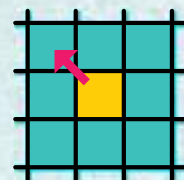
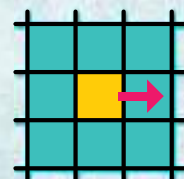
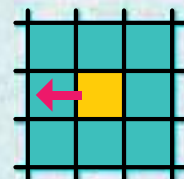
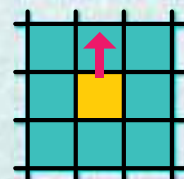
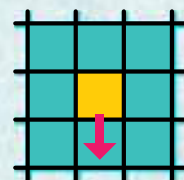
```

→ → → if row < 9 and column > 0:
→ → → if bombfield[row+1][column-1] == 1:
→ → → totalBombs = totalBombs + 1
    
```

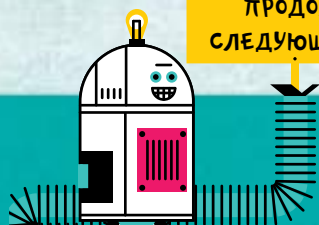
Если вы уже в нижней строке (строка 9) или в левой колонке (колонка 0), то эта проверка пропускается.

НЕ ОШИБИСЬ!

Внимательно используйте **операторы** при подсчете количества мин вокруг клеток. Если вы ошибетесь, игра не будет выдавать аварийных сообщений, однако цифры в коричневых клеточках будут неправильными.



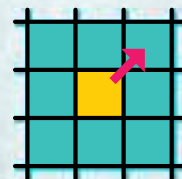
ПРОДОЛЖЕНИЕ НА
СЛЕДУЮЩЕЙ СТРАНИЦЕ.



После этого проверим верхний правый угол.

```
→ → → if row > 0 and column < 9:
→ → → if bombfield[row-1][column+1] == 1:
→ → → totalBombs = totalBombs + 1
```

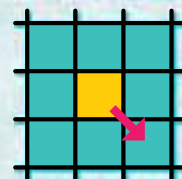
Если вы уже в верхней строке (строка 0) или в правой колонке (колонка 9), то эта проверка пропускается.



Наконец проверим нижний правый угол.

```
→ → → if row < 9 and column < 9:
→ → → if bombfield[row+1][column+1] == 1:
→ → → totalBombs = totalBombs + 1
```

Если вы уже в нижней строке (строка 9) или в правой колонке (колонка 9), то эта проверка пропускается.



21. Теперь, когда вы проверили все окружающие клетки, уже можно показать их количество внутри коричневого квадрата.

```
→ → → square.config(text = " " + str(totalBombs) + " ")
```

Эта строка превращает число (количество мин вокруг квадратика) в строчную переменную. Для этого используется функция `str()`, а результат помещается непосредственно в квадрат.

22. Чтобы отслеживать количество очков, нам необходимо прибавлять очко (единицу) всякий раз, когда открывается безопасный квадрат, и уменьшать на единицу переменную `squaresToClear` (число квадратиков, которое необходимо открыть).

```
→ → → squaresToClear = squaresToClear - 1
→ → → score = score + 1
```

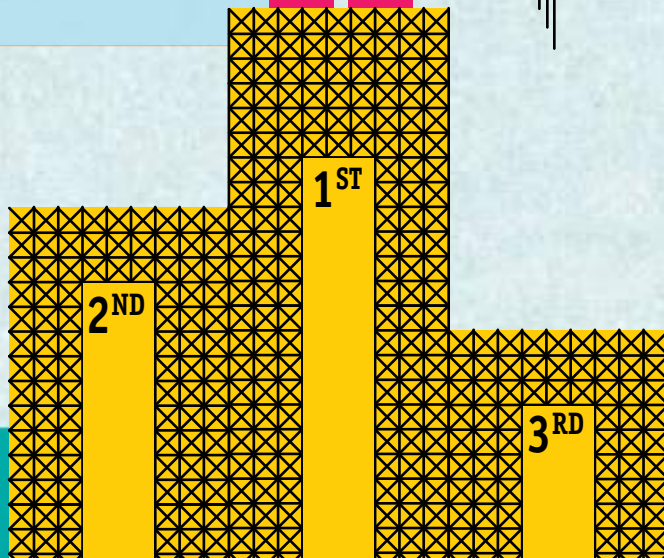
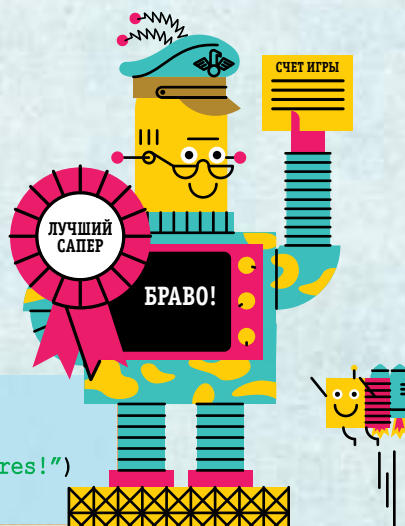
Оставьте по одному пробелу с каждой стороны цифры.

23. Когда вы откроете все безопасные квадраты, игра закончится. В этом случае компьютер должен напечатать поздравительное сообщение и количество очков.

```
→ → → if squaresToClear == 0:
→ → → gameOver = True
→ → → print("Well done! You found all the safe squares!")
→ → → print("Your score was:", score)
```

24. В последней строке программы надо привести инструкцию из шага 12. (Она нужна для того, чтобы запустить программу.)

```
play_bombdodger()
```



25. Вы только что закончили написание программы! Полностью текст программы приведен ниже. Сохраните текст программы и запустите ее для тестирования. (Если вы получите сообщения об ошибках, то воспользуйтесь контрольным списком вопросов из шага 12 для отладки.)

```
import tkinter
import random
gameOver = False
score = 0
squaresToClear = 0
def play_bombdodger():
    create_bombfield(bombfield)
    window = tkinter.Tk()
    layout_window(window)
    window.mainloop()
bombfield = []
def create_bombfield(bombfield):
    global squaresToClear
    for row in range(0,10):
        rowList = []
        for column in range(0,10):
            if random.randint(1,100) < 20:
                rowList.append(1)
            else:
                rowList.append(0)
                squaresToClear = squaresToClear + 1
        bombfield.append(rowList)
    #printfield(bombfield)
def printfield(bombfield):
    for rowList in bombfield:
        print(rowList)
play_bombdodger()
def layout_window(window):
    for rowNumber, rowList in enumerate(bombfield):
        for columnNumber, columnEntry in enumerate(rowList):
            if random.randint(1,100) < 25:
                square = tkinter.Label(window, text = " ", bg = "darkgreen")
            elif random.randint(1,100) > 75:
                square = tkinter.Label(window, text = " ", bg = "seagreen")
            else:
                square = tkinter.Label(window, text = " ", bg = "green")
            square.grid(row = rowNumber, column = columnNumber)
            square.bind("<Button-1>", on_click)
play_bombdodger()
def on_click(event):
    global score
    global gameOver
    global squaresToClear
    square = event.widget
    row = int(square.grid_info()["row"])
```

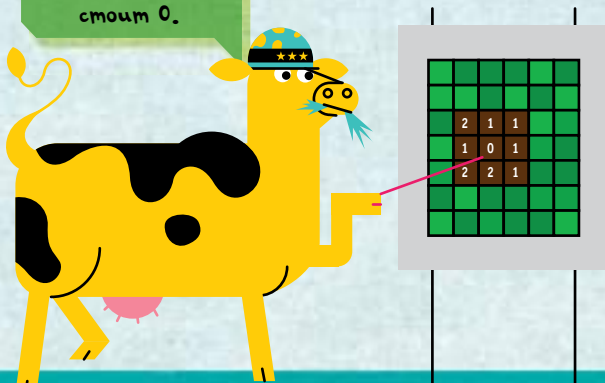
Если эту строчку не закомментировать, то в командном окне будет появляться карта минного поля.

Эта строчка была добавлена на шаге 8 для предварительной проверки.

Эта строчка была добавлена на шаге 11 для предварительной проверки.

Все клетки вокруг этой безопасны, поскольку в ней стоит 0.

ПРОГРАММА НАЧИНАЕТСЯ ЗДЕСЬ...



ФУНКЦИЯ PRINTFIELD()

Всю математику вашей программы вы можете проверить при помощи функции **printfield()** из шага 6. С ее помощью вы выводите карту минного поля в командное окно, затем кликаете по любым безопасным квадратам и проверяете, соответствует ли число в клетке реальному количеству мин в окружении. Если не соответствует, то вам надо проверять текст программы, в особенности на шаге 20.

... И ПРОДОЛЖАЕТСЯ СЮДА.

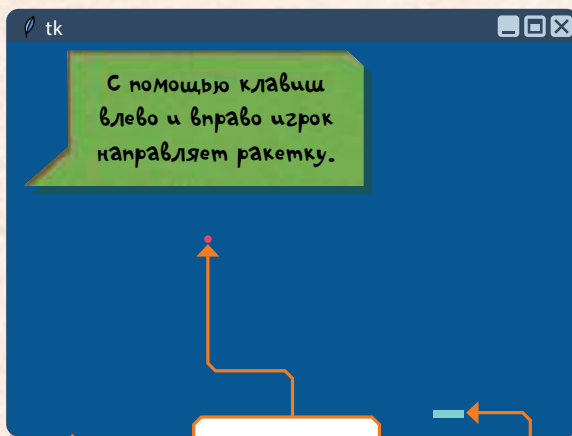
```
column = int(square.grid_info()["column"])
currentText = square.cget("text")
if gameOver == False:
    if bombfield[row][column] == 1:
        gameOver = True
        square.config(bg = "red")
        print("Game Over! You hit a bomb!")
        print("Your score was: ", score)
    elif currentText == " ":
        square.config(bg = "brown")
        totalBombs = 0
        if row < 9:
            if bombfield[row+1][column] == 1:
                totalBombs = totalBombs + 1
        if row > 0:
            if bombfield[row-1][column] == 1:
                totalBombs = totalBombs + 1
        if column > 0:
            if bombfield[row][column-1] == 1:
                totalBombs = totalBombs + 1
        if column < 9:
            if bombfield[row][column+1] == 1:
                totalBombs = totalBombs + 1
        if row > 0 and column > 0:
            if bombfield[row-1][column-1] == 1:
                totalBombs = totalBombs + 1
        if row < 9 and column > 0:
            if bombfield[row+1][column-1] == 1:
                totalBombs = totalBombs + 1
        if row > 0 and column < 9:
            if bombfield[row-1][column+1] == 1:
                totalBombs = totalBombs + 1
        if row < 9 and column < 9:
            if bombfield[row+1][column+1] == 1:
                totalBombs = totalBombs + 1
        square.config(text = " " + str(totalBombs) + " ")
        squaresToClear = squaresToClear - 1
        score = score + 1
    if squaresToClear == 0:
        gameOver = True
        print("Well done! You found all the safe squares!")
        print("Your score was:", score)
play_bombdodger()
```

ТЕННИС

Мы можем воспользоваться модулем **tkinter** для создания простой игры — тенниса. Посмотрим, как долго вы сможете продержаться мячик в движении.

Правила игры

Вот как выглядит игровое поле по окончании игры.



Игра запускается в окне модуля **tkinter** на чистом холсте.

Это мяч. Он отскакивает от левых и правых краев холста.

Это ракетка. Она движется из стороны в сторону и отбивает мяч, удерживая его в движении.

Как это работает

Чтобы заставить игру работать, ваша программа должна...

- Нарисовать собственно ракетку и мяч.
- Заставить ракетку и мяч двигаться.
- Определить момент отскока мяча.
- Остановить игру, если мяч коснулся нижнего края холста, и предложить пользователю сыграть еще.

Чтобы была возможность замедлять игру — иначе мяч слишком ускорится, мы можем добавить паузу. Для этого используется модуль **time**.

МОДУЛЬ TIME

Модуль **time** часто используется для получения информации о времени и дате. Также он позволяет программировать искусственное замедление в игре.



Создание холста, ракетки и мяча

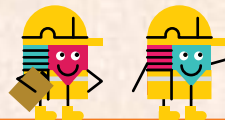
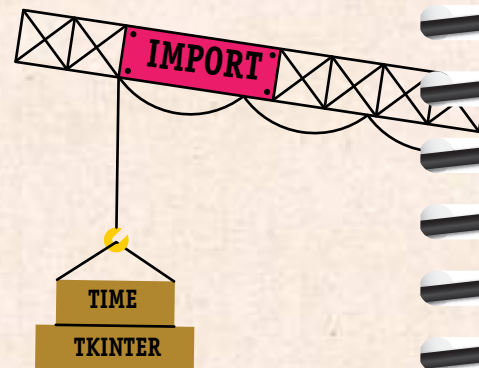
1. Откройте и сохраните новый файл. Импортируйте модули **tkinter** и **time**.

```
import tkinter
import time
```

2. Далее создайте холст.

```
canvasWidth = 750
canvasHeight = 500
window = tkinter.Tk()
canvas = tkinter.Canvas(window, width=canvasWidth, height=canvasHeight, bg="dodgerblue4")
canvas.pack()
```

Чтобы иметь возможность программировать отскоки мяча, нам необходимо задать точный размер холста (в пикселях).



3. С использованием функции **create_rectangle()** можно построить узкий прямоугольник ракетки. Чтобы построить мяч, мы воспользуемся функцией **create_oval()**. Для каждой из фигур мы можем установить размер и цвет, задав параметры таким образом:

create_rectangle() и **create_oval()** входят в состав модуля **tkinter**.

Верхний край (Y)

Левый край (X)

Правый край (X)(X)

Нижний край (Y)(Y)

Эта инструкция создает прямоугольник бирюзового цвета длиной 40 пикселей и высотой 10 пикселей.

```
bat = canvas.create_rectangle(0, 0, 40, 10, fill="dark turquoise")
ball = canvas.create_oval(0, 0, 10, 10, fill="deep pink")
```

Все это временные **координаты**, необходимые лишь для того, чтобы задать размеры фигур. Их начальные положения на холсте будут установлены позднее.

Эта инструкция создает розовый мяч диаметром 10 пикселей.

4. Задайте переменную, в которой будете отслеживать окно модуля **tkinter**.

```
windowOpen = True
```

Эта переменная будет использована для проверки того, открыто ли до сих пор окно модуля **tkinter**. Чтобы игра работала, эта переменная должна быть установлена на **True** (Да).

Теперь вы можете создавать основной цикл — «**main loop**», в котором будут отслеживаться все изменения на игровом поле. (Здесь мы определим все функции, которые будут использоваться в следующих шагах.)

```
def main_loop():
```

```
    → while windowOpen == True:
```

```
        → move_bat()
```

```
        → move_ball()
```

```
        → window.update()
```

```
        → time.sleep(0.02)
```

```
        → if windowOpen == True:
```

```
            → check_game_over()
```

Эта инструкция будет управлять ракеткой.

Эта инструкция будет управлять мячом.

Эта инструкция обновляет окно модуля **tkinter**, так что вы будете видеть движение объектов на экране. Также обновление необходимо, чтобы проверить наступление событий (**event**), таких как нажатия клавиш.

После каждого обновления необходимо проверять, открыто ли все еще игровое окно.

Эта инструкция добавляет небольшую паузу, чтобы запрограммировать искусственное замедление в игре.

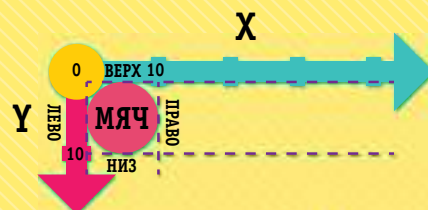
Суть инструкций **main_loop()** в вызове других функций.

КООРДИНАТЫ И ФИГУРЫ

Для построения прямоугольника достаточно задать координаты (X и Y) сторон — верха, низа, левого и правого краев.



Для построения круга задаются координаты (X и Y) левой, верхней, правой и нижней точек.



ФУНКЦИЯ UPDATE()

Даже если компьютер постоянно меняет положение объектов на экране, вы увидите это только в том случае, если будете обновлять картинку на экране. Для этого и нужна функция **update()**. Если обновлять экран достаточно часто, то будет складываться впечатление, что объекты плавно движутся. Но на самом деле это лишь последовательность статичных картинок.

Использование стрелок

5. На этом шаге мы введем две новые переменные. В этих переменных будет храниться информация о том, какая из клавиш со стрелкой нажата.

```
leftPressed = 0
rightPressed = 0
```

0 – не нажата
1 – нажата

Затем создадим функцию, которая изменяет эти переменные при нажатии клавиш со стрелками.

```
def on_key_press(event):
    → global leftPressed, rightPressed
    → if event.keysym == "Left":
    →     → leftPressed = 1
    → elif event.keysym == "Right":
    →     → rightPressed = 1
```

event.keysym связывает определенные клавиши с событиями. Слово **keysym** это сокращение от «**key symbol**» (символ клавиши). Эти символы имеют строчные значения: «**Left**» для левой стрелки и «**Right**» для правой стрелки.

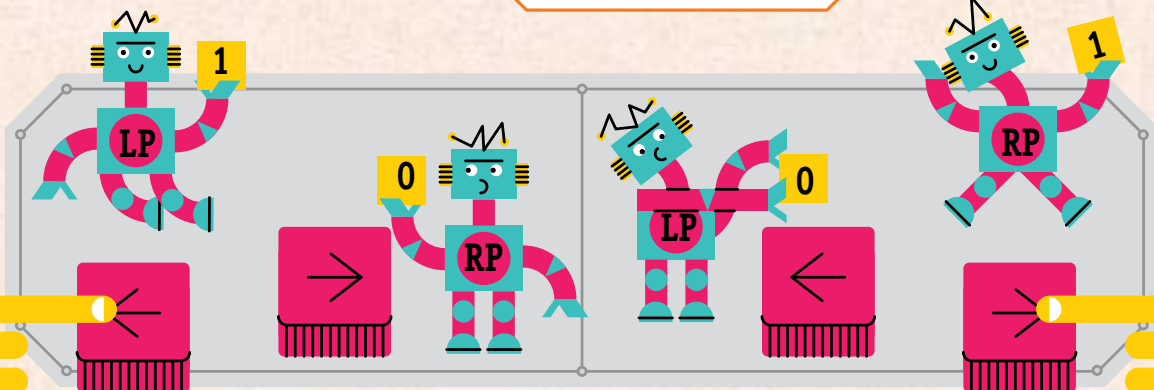
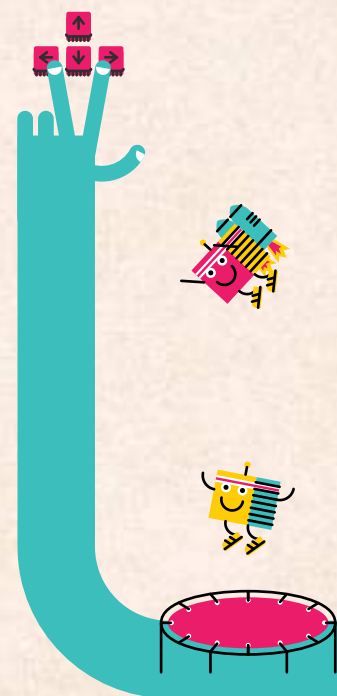
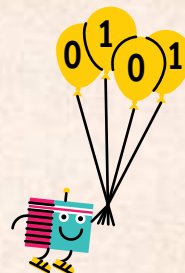
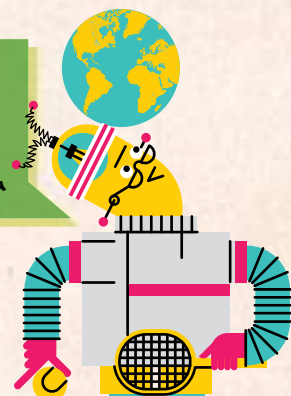
Если клавиша нажимается, то переменная становится равна единице. Ее значение будет использовано на шаге 8, чтобы заставить ракетку двигаться.

6. Компьютеру также надо знать, что делать, когда клавиша отпускается.

```
def on_key_release(event):
    → global leftPressed, rightPressed
    → if event.keysym == "Left":
    →     → leftPressed = 0
    → elif event.keysym == "Right":
    →     → rightPressed = 0
```

Когда вы отпускаете клавишу, эта переменная снова приобретает нулевое значение.

Эти переменные будут глобальными (*global*), поскольку обращение к ним идет за пределами функции



7. Ракетка может двигаться только влево и вправо, так что меняется лишь координата X. Чтобы запрограммировать это движение, мы используем координату «batMove». Еще нам понадобится формула, изменяющая эту переменную в зависимости от того, какая из клавиш была нажата.

Переменная **batSpeed** — это количество пикселей, на которое сдвигается ракетка при каждом обновлении экрана.

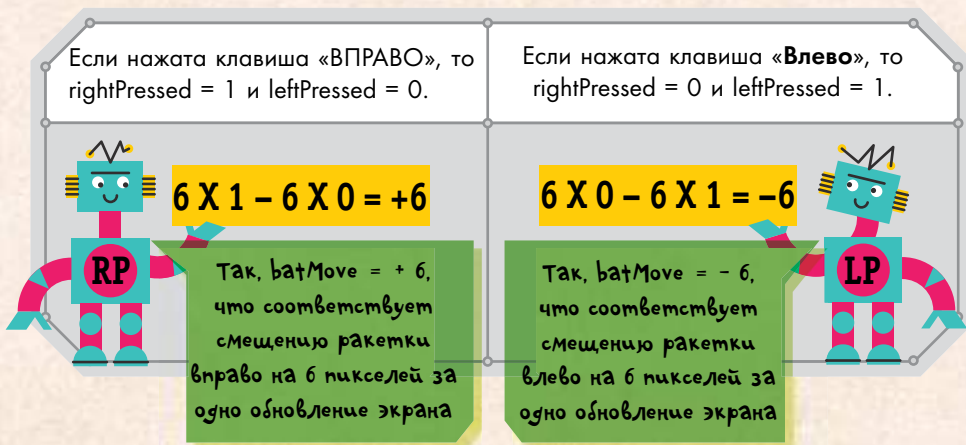
Если нажата клавиша «ВПРАВО», то перемещение ракетки положительное. (Если клавиша не нажата, то положительное движение нулевое.)

Если нажата клавиша «ВЛЕВО», то перемещение ракетки отрицательное. (Если клавиша не нажата, то отрицательное движение нулевое.)

```
batSpeed = 6
def move_bat():
    batMove = batSpeed*rightPressed - batSpeed*leftPressed
```

Переменная «batMove» показывает, насколько сдвинется ракетка в определенном направлении за одно обновление экрана.

Переменная «batSpeed» имеет постоянное значение (6), а вот переменная «batMove» может меняться в зависимости от того, в каком направлении должна двигаться ракетка. Вот как выглядят вычисления в обоих случаях:



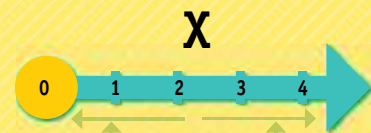
8. Создайте еще переменные, которые будут обозначать положения ребер ракетки, и задайте функцию **coords()**, см. справа, чтобы компьютер мог знать, где находится ракетка.

```
→ (batLeft, batTop, batRight, batBottom) = canvas.coords(bat)
```

Это края ракетки, вернее, координаты левого, верхнего, правого и нижнего краев.

Эта функция хранит текущие координаты ракетки на холсте (см. вставку).

БОКОВОЕ ДВИЖЕНИЕ НА ХОЛСТЕ

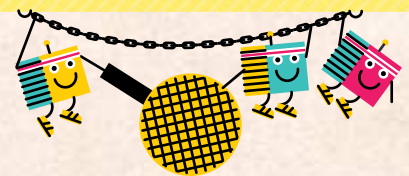


—

+

Если движение отрицательное, вам надо нажать клавишу ВЛЕВО.

Если движение положительное, вам надо нажать клавишу ВПРАВО.



ФУНКЦИЯ COORDS()

С помощью функции **coords()** можно определять положение объекта на холсте. А именно: **canvas.coords()**

Поместите в скобки название объекта, положение которого на холсте вы хотите найти. Эта функция возвращает координаты, так что на протяжении всей программы мы будем пользоваться этим ее свойством.

9. Ракетка может перемещаться лишь в границах холста. Если она достигает правого края, то может появиться только у левого. Мы можем использовать **условие if/and**, чтобы останавливать ракетку у левого и правого краев.

ФУНКЦИЯ MOVE()

Функция **move()** из модуля **tkinter** позволяет перемещать объекты по холсту. Она имеет три параметра: название объекта, величину перемещения по горизонтали (ось **X**) и величину перемещения по вертикали (ось **Y**).

Это условие исключает случай, когда вы на левом крае холста двигаетесь влево.

Это условие исключает случай, когда вы на правом крае холста двигаетесь вправо.

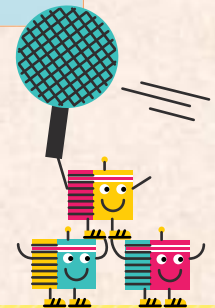
```
if (batLeft > 0 or batMove > 0) and (batRight < canvasWidth or batMove < 0):
    canvas.move(bat, batMove, 0)
```

Эта инструкция обращается к функции **move()**, смотри вставку выше.

Это название объекта, который должен двигаться (ракетка).

Это величина смещения (**batMove**) по оси **X**.

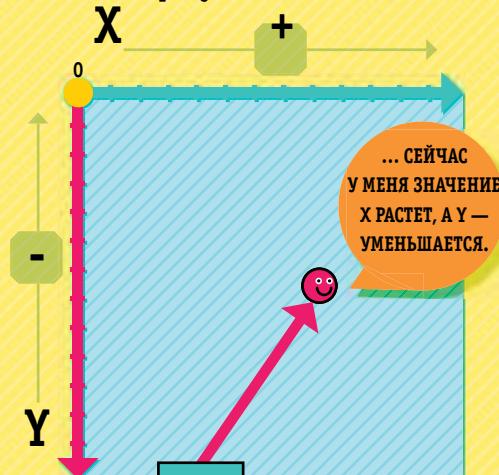
Высота (значение по оси **Y**) расположения ракетки не меняется.



Отскок мяча

Движение мяча более разнообразно. Он может перемещаться по всему холсту, отскакивая от верхнего и боковых краев. Соответственно, будет меняться и координата **X**, и координата **Y**. Движение по этим двум осям будет представлено переменными «**ballMoveX**» (по горизонтали) и «**ballMoveY**» (по вертикали).

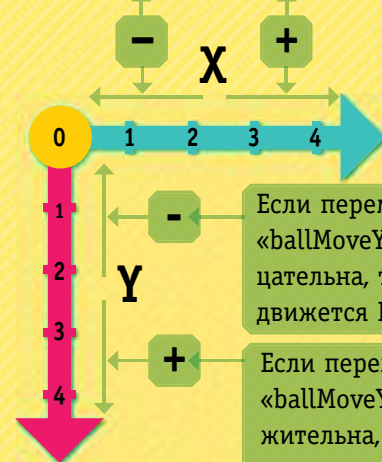
Например, в начале игры мяч будет отскакивать по диагонали вверх и вправо, что на рисунке выглядит так.



ДВИЖЕНИЕ ПО ХОЛСТУ

Если переменная «**ballMoveX**» отрицательна, то движение ВЛЕВО.

Если переменная «**ballMoveX**» положительна, то движение ВПРАВО.



Если переменная «**ballMoveY**» отрицательна, то мяч движется ВВЕРХ.

Если переменная «**ballMoveY**» положительна, то мяч движется ВНИЗ.

Мяч продолжает двигаться в одном направлении, пока не встретит край холста или ракетку



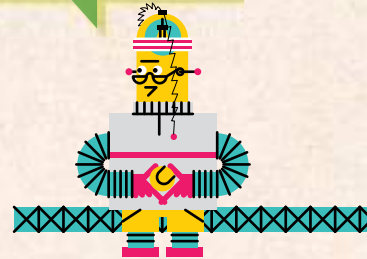
10. Давайте зададим глобальные переменные, которые отвечают за скорость мяча. Также нам потребуются глобальные переменные, в которых мы будем хранить «высоты» (координаты Y) верхнего и нижнего краев ракетки.

```
ballMoveX = 4
ballMoveY = -4
setBatTop = canvasHeight-40
setBatBottom = canvasHeight-30
```

ballMoveX отвечает за движение по горизонтали.
ballMoveY отвечает за движение по вертикали.

Эти уровни используются для вычисления того, достигнуто ли касание мяча и ракетки или мяч прошел мимо (это нужно для шага 13).

Для движения мяча необходимы уже две переменные, поскольку надо отслеживать изменения положения мяча по двум осям: горизонтальной (ось X) и вертикальной (ось Y)



11. Теперь мы зададим функцию движения мяча. Эта функция будет менять переменную «ballMove» при отскоке мяча. Чтобы определить момент, когда мяч отскочит, мы воспользуемся функцией **coord()**. Она потребуется нам для определения границ мяча.

```
def move_ball():
    global ballMoveX, ballMoveY
    (ballLeft, ballTop, ballRight, ballBottom) = canvas.coords(ball)
```

Объявление переменной глобальной (**global**) позволит программе изменять их за пределами этой функции.

Это границы мяча, вернее, координаты его левой, верхней, правой и нижней границ.

Эта функция возвращает текущие координаты мяча.

Вы не забыли о том, что переменная «ballMoveX» хранит данные не только о скорости мяча (4), но также и о направлении его движения (+4 или -4)?

12. Когда мяч сталкивается с правой границей холста, он отскакивает налево (переменная ballMoveX становится отрицательной).

Если переменная ballMoveX больше нуля, то мяч летит НАПРАВО.

Эта инструкция проверяет, достиг ли мяч правой границы холста.

```
if ballMoveX > 0 and ballRight > canvasWidth:
    ballMoveX = -ballMoveX
```

Придавая переменной ballMoveX отрицательное значение, мы направляем мяч в левом направлении. Значение ballMoveY остается неизменным.

Когда мяч сталкивается с левой границей холста, он отскакивает направо (переменная ballMoveX становится положительной).

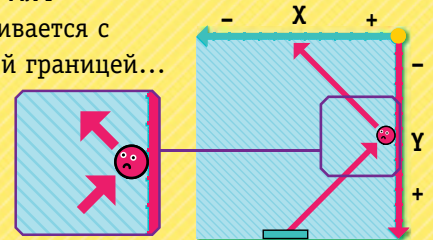
Если переменная ballMoveX меньше нуля, то мяч летит НАЛЕВО.

Эта инструкция проверяет, достиг ли мяч левой границы холста.

```
if ballMoveX < 0 and ballLeft < 0:
    ballMoveX = -ballMoveX
```

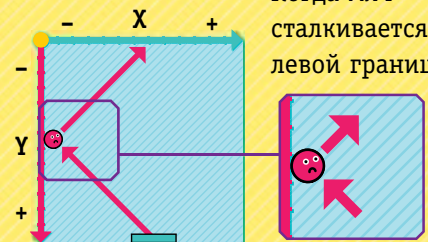
Эта инструкция снова придает переменной ballMoveX положительное значение.

Когда мяч сталкивается с правой границей...



... переменная ballMoveX меняет знак на отрицательный.

Когда мяч сталкивается с левой границей...



... переменная ballMoveX меняет знак на положительный.

Когда мяч сталкивается с верхним краем холста, он отскакивает назад (переменная `ballSpeedY` становится положительной).

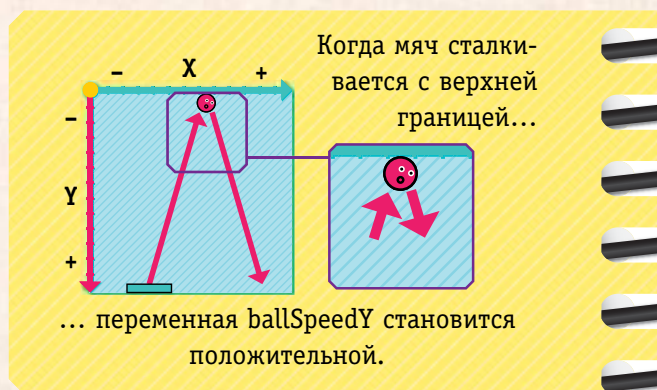
Если `ballSpeedY` меньше нуля, то мяч движется ВВЕРХ.

Эта инструкция проверяет, достиг ли мяч верхней границы холста.

```
→ if ballMoveY < 0 and ballTop < 0:
→   → ballMoveY = -ballMoveY
```



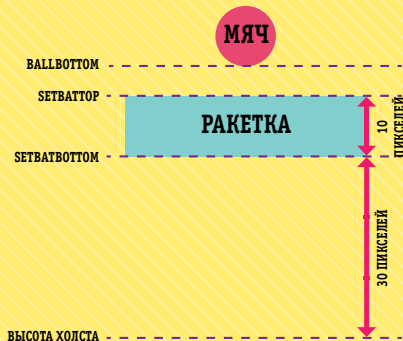
Эта инструкция превращает `ballSpeedY` из отрицательного в положительное. Соответственно, мяч отскакивает и начинает движение вниз.



13. Когда мяч достигает нижней части холста, вам надо проверить, коснулся ли он ракетки. Для этого потребуются переменные `setBatTop` и `setBatBottom`, которые мы создавали на шаге 10.

Мяч отскочит, только если попадет на ракетку.

Нам необходимо знать, когда уровень `ballBottom` сравняется с уровнем ракетки, т.е. окажется между уровнями `setBatTop` и `setBatBottom`.



Если `ballMoveY` больше нуля, мяч движется ВНИЗ.

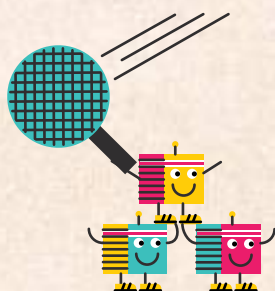
Эта инструкция проверяет, достиг ли мяч того уровня, чтобы отскочить, но не пролететь мимо.

```
→ if ballMoveY > 0 and ballBottom > setBatTop and ballBottom < setBatBottom:
→   → (batLeft, batTop, batRight, batBottom) = canvas.coords(bat)
→   → if ballRight > batLeft and ballLeft < batRight:
→     → ballMoveY = -ballMoveY
```

Если оба эти условия верны, мяч коснется ракетки и должен изменить направление.

Эта инструкция проверяет, находится ли мяч в момент касания над ракеткой. В противном случае он пройдет левее или правее.

Эта инструкция обновляет переменные с координатами ракетки.



Мяч пройдет мимо ракетки, если `ballRight < batLeft` или `ballLeft > batRight`.





14. Давайте закончим функцию движения мяча.

```
→ canvas.move(ball, ballMoveX, ballMoveY)
```

15. Если мяч проходит мимо ракетки и попадает в нижнюю часть холста, игра заканчивается. Это условие будет проверять новая функция **check_game_over()**. Также вы можете спросить у пользователя, не хочет ли он сыграть еще раз, для чего используется всплывающее сообщение. Если пользователь выберет «yes», игра возобновится, если «no» — окно закроется.

ОКНО С СООБЩЕНИЕМ

Модуль **tkinter** включает в себя инструмент **messagebox**, который позволяет создавать окно с сообщением, в частности диалоговое окно. Вы можете сочетать **messagebox** с другими функциями, создавая особые диалоговые окна. Например, **askyesno()**. Эта функция добавляет кнопки «Да» (Yes) и «Нет» (No).

```
def check_game_over():
```

```
→ (ballLeft, ballTop, ballRight, ballBottom) = canvas.coords(ball)
```

```
→ if ballTop > canvasHeight:
```

```
→ → playAgain = tkinter.messagebox.askyesno(message="Do you want to play again?")
```

```
→ → if playAgain == True:
```

```
→ → → reset()
```

```
→ → → else:
```

```
→ → → close()
```

Эта инструкция обновляет координаты мяча.

Эта инструкция проверяет, прошел ли мяч нижний край холста.

Эти функции будут определены ниже (на этой странице и следующей).

Эта инструкция создает окно с сообщением и кнопками yes/no.

Завершение игры

16. Если вы выбрали не продолжать игру, то должны закрыть окно модуля **tkinter**.

```
def close():
```

```
→ global windowOpen
```

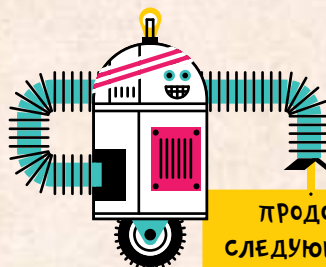
```
→ windowOpen = False
```

```
→ window.destroy()
```

Эта инструкция придает переменной **windowOpen** значение **False** и прерывает работу функции **main_loop()**.

destroy() — это функция модуля **tkinter**, которая «уничтожает», или закрывает окна.

Я НЕ ХОЧУ
ПРОДОЛЖАТЬ ИГРУ,
УНИЧТОЖИТЬ ОКНО,
УНИЧТОЖИТЬ!



ПРОДОЛЖЕНИЕ НА
СЛЕДУЮЩЕЙ СТРАНИЦЕ.



17. Чтобы снова начать игру, нам необходимо привести переменные в начальное состояние.

```
def reset():
```

```
→ global leftPressed, rightPressed
```

```
→ global ballMoveX, ballMoveY
```

```
→ leftPressed = 0
```

```
→ rightPressed = 0
```

```
→ ballMoveX = 4
```

```
→ ballMoveY = -4
```

```
→ canvas.coords(bat, 10, setBatTop, 50, setBatBottom)
```

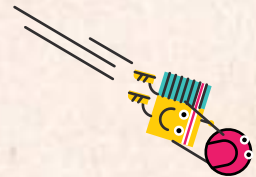
```
→ canvas.coords(ball, 20, setBatTop-10, 30, setBatTop)
```

Надо обновить глобальные переменные, которые вы создавали на шаге 10.

Надо очистить все предыдущие нажатия клавиш.

Эти строчки устанавливают начальную скорость мяча, задавая движение по осям X и Y.

Поместите ракетку внизу экрана, а мячик непосредственно над ним.



18. Также необходимо научить игру реагировать на щелчки кнопкой мыши и клавишами клавиатуры. Это потребует двух различных команд связи: **protocol** и **bind** (см. вставку).

window — это окно модуля **tkinter**.

Когда окно закрывается, автоматически посылается следующее сообщение.

Эта инструкция связывает нажатие кнопки **close [x]** с функцией **close()**, которая прерывает работу основного цикла программы.

```
window.protocol("WM_DELETE_WINDOW", close)
window.bind("<KeyPress>", on_key_press)
window.bind("<KeyRelease>", on_key_release)
```

Эта строчка связывает нажатие и отпускание клавиш с функциями, которые мы определили на этапах 5 и 6.

Имена функций внутри скобок не должны иметь собственных скобок. Это связано с тем, что вы осуществляете связь с функцией, а не вызываете ее

КОМАНДЫ СВЯЗИ

Мы часто используем связь функций с различными событиями, чтобы при наступлении этих событий вызывались соответствующие функции.

protocol связывает функцию с сообщением из окна модуля **tkinter**.

bind связывает функцию с другими событиями, такими как нажатие и отпускание клавиш.

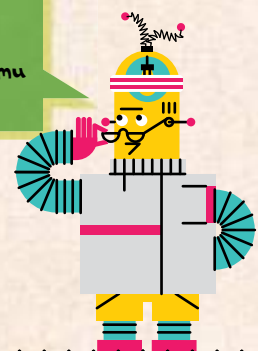
Если в работе программы возникают проблемы, некоторые советы по отладке вы можете найти на стр. 88–89

19. Наконец начинаем игру, вызывая функции **reset()** и **main_loop()**.

```
reset()
main_loop()
```

Эта инструкция гарантирует, что программа начнет работу с правильными начальными установками.

Вы закончили! Теперь сохраните и запустите программу для тестирования. Сколько времени вам удастся удерживать мяч в движении?



Начисление очков

Если вы хотите отслеживать свои успехи, то можете считать очки, т.е. количество раз, когда вы отбили мяч.

1. Для отслеживания количества очков нам необходима новая переменная `score`. Добавьте ее после переменной `windowOpen` на шаге 4.

```
score = 0
```

Установите в начале игры переменную на ноль.

2. Увеличивайте переменную на единицу каждый раз, когда мяч отскакивает от ракетки. Вы можете делать это внутри функции `move_ball()`.

```
def move_ball():
```

```
    → global ballMoveX, ballMoveY, score
```

Добавьте переменную `score` в конце этой строки.

```
    → ballMoveY = -ballMoveY
```

```
    → score = score + 1
```

```
    → canvas.move(ball, ballMoveX, ballMoveY)
```

Вставьте эту инструкцию почти в самом конце функции `move()` из этапа 14. Именно там мы программировали отскок мячика от ракетки.

3. Добавьте `print()` в функцию `check_game_over()`.

```
→ if ballTop > canvasHeight:
```

```
→ print("Your score was " + str(score))
```

Добавьте эту строчку, чтобы показывать количество очков в командном окне.

4. Наконец сделайте так, чтобы в начале игры количество очков обнулялось, добавив функцию `reset()`.

```
def reset():
```

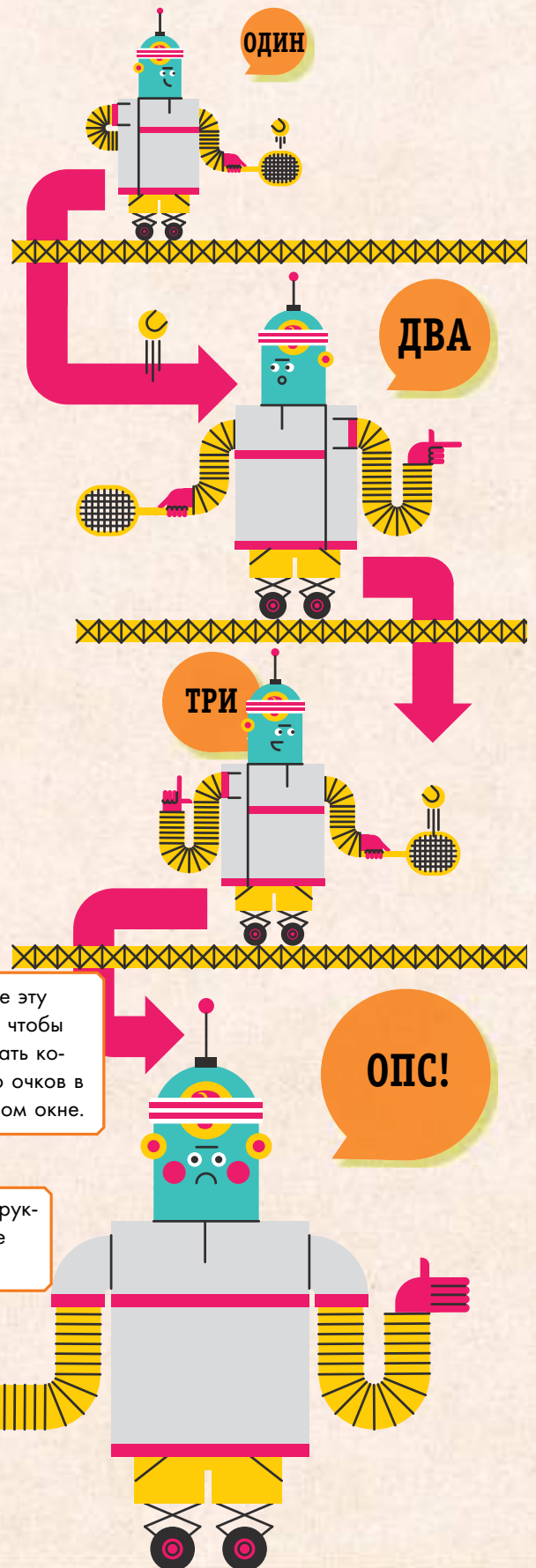
```
→ global score
```

Вставьте в начале программы эту инструкцию, чтобы объявить переменную `score` глобальной и получить к ней доступ.

```
→ score = 0
```

Добавьте эту инструкцию в конце определения функции.

Попробуйте поиграть в игру теперь. На заднем плане, за холстом, вы увидите в командном окне свое количество очков.



Ускоряемся

Если хотите, можете сделать эту игру веселее (и труднее), добавив постепенное увеличение скорости.

1. Для этого вам потребуется еще одна переменная, в которой будет отсчитываться количество отскоков. Добавьте ее после переменной `score`.

```
bounceCount = 0
```

2. Вставьте в функцию `move_ball()` новые строки кода. В частности, переменную `bounceCount`, которая должна увеличиваться после каждого отскока. Пусть скорость будет расти после каждого четвертого соударения.

```
def move_ball():  
    → global ballMoveX, ballMoveY, score, bounceCount, batSpeed
```

Добавьте эти две переменные сразу после числа очков (`score`).

```
    → ballMoveY = -ballMoveY  
    → score = score + 1  
    → bounceCount = bounceCount + 1  
    → if bounceCount == 4:  
    →     bounceCount = 0  
    →     batSpeed = batSpeed + 1  
    → if ballMoveX > 0:  
    →     ballMoveX = ballMoveX + 1  
    → else:  
    →     ballMoveX = ballMoveX - 1  
    → ballMoveY = ballMoveY - 1
```

Вставьте ниже этой строки новые программные строки.

Мы делаем так, чтобы когда число отскоков (`bounceCount`) достигает четырех, оно устанавливалось на ноль...

... и увеличиваем скорость ракетки на единицу...

... и увеличиваем на единицу скорость движения мяча туда и обратно.

3. В начале игры установите переменные `bounceCount` и `batSpeed` на начальное значение, добавив эти строки в функцию `reset()`.

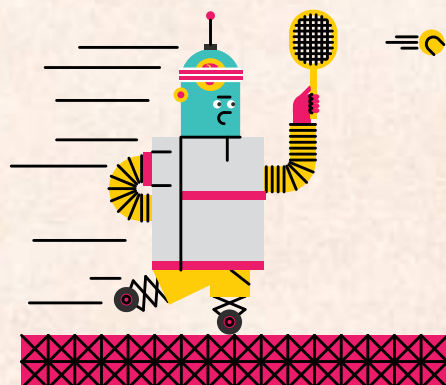
```
def reset():  
    → global score, bounceCount, batSpeed
```

```
    → bounceCount = 0  
    → batSpeed = 6
```

Добавьте эти строки в конце определения функции.

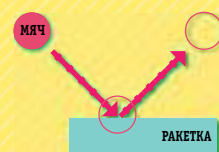
4. Поиграйте в игру еще. Теперь она постепенно ускоряется. Когда скорость мяча будет довольно высокой, вы можете обнаружить, что мяч просто проскакивает сквозь ракетку. Чтобы исправить этот «баг», вам надо модифицировать некоторые условия.

Мяч может пройти сквозь ракетку, если его скорость слишком велика.



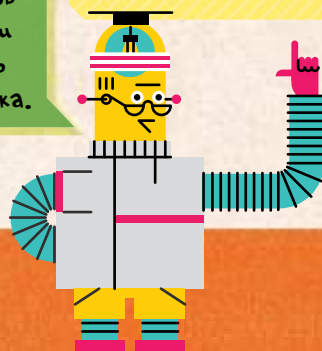
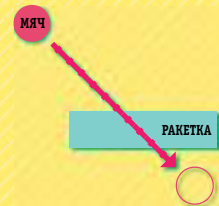
МЕДЛЕННО

Когда мяч перемещается за обновление лишь на несколько пикселей, он легко попадает в ракетку.



БЫСТРО

Но когда мяч проходит за обновление на много пикселей вперед, он может проскочить между обновлениями сквозь ракетку.



В функции `move_ball()` удалите эту строку:

→ `if ballRight > batLeft and ballLeft < batRight:`

И замените ее этим:

→ `if ((ballMoveX > 0 and (ballRight+ballMoveX > batLeft and ballLeft < batRight) or ballSpeedX < 0 and (ballRight > batLeft and ballLeft+ballMoveX < batRight))):`

Ниже приведен полный текст программы, причем новые вставки выделены в рамки.

```
import tkinter
import time
canvasWidth = 750
canvasHeight = 500
window = tkinter.Tk()
canvas = tkinter.Canvas(window, width=canvasWidth, height=canvasHeight, bg="dodgerblue4")
canvas.pack()
bat = canvas.create_rectangle(0, 0, 40, 10, fill="dark turquoise")
ball = canvas.create_oval(20, 0, 30, 10, fill="deep pink")
windowOpen = True
score = 0
bounceCount = 0
def main_loop():
    while windowOpen == True:
        move_bat()
        move_ball()
        window.update()
        time.sleep(0.02)
        if windowOpen == True:
            check_game_over()
leftPressed = 0
rightPressed = 0
def on_key_press(event):
    global leftPressed, rightPressed
    if event.keysym == "Left":
        leftPressed = 1
    elif event.keysym == "Right":
        rightPressed = 1
def on_key_release(event):
    global leftPressed, rightPressed
    if event.keysym == "Left":
        leftPressed = 0
    elif event.keysym == "Right":
        rightPressed = 0
batSpeed = 6
def move_bat():
    batMove = batSpeed*rightPressed - batSpeed*leftPressed
    (batLeft, batTop, batRight, batBottom) = canvas.coords(bat)
    if (batLeft > 0 or batMove > 0) and (batRight < canvasWidth or batMove < 0):
        canvas.move(bat, batMove, 0)
ballMoveX = 4
ballMoveY = -4
setBatTop = canvasHeight-40
setBatBottom = canvasHeight-30
def move_ball():
    global ballMoveX, ballMoveY, score, bounceCount, batSpeed
    (ballLeft, ballTop, ballRight, ballBottom) = canvas.coords(ball)
    if ballMoveX > 0 and ballRight > canvasWidth:
        ballMoveX = -ballMoveX
    if ballMoveX < 0 and ballLeft < 0:
        ballMoveX = -ballMoveX
    if ballMoveY < 0 and ballTop < 0:
        ballMoveY = -ballMoveY
    if ballMoveY > 0 and ballBottom > setBatTop and ballBottom < setBatBottom:
        (batLeft, batTop, batRight, batBottom) = canvas.coords(bat)
```

Не забудьте УДАЛИТЬ эту строчку, которая проверяет, находится ли ракетка под мячом в момент касания.

Поставьте более сложное условие, которое мы привели в этих скобках.

... И ПЕРЕХОДИТ СЮДА.

```
if ballRight > batLeft and ballLeft < batRight:
    if (ballMoveX > 0 and (ballRight+ballMoveX > batLeft and ballLeft < batRight) or ballMoveX < 0 and (ballRight > batLeft and ballLeft+ballMoveX < batRight)):
```

```
        ballMoveY = -ballMoveY
        score = score + 1
        bounceCount = bounceCount + 1
        if bounceCount == 4:
            bounceCount = 0
            batSpeed = batSpeed + 1
            if ballMoveX > 0:
                ballMoveX = ballMoveX + 1
            else:
                ballMoveX = ballMoveX - 1
                ballMoveY = ballMoveY - 1
```

```
        canvas.move(ball, ballMoveX, ballMoveY)
def check_game_over():
    (ballLeft, ballTop, ballRight, ballBottom) = canvas.coords(ball)
```

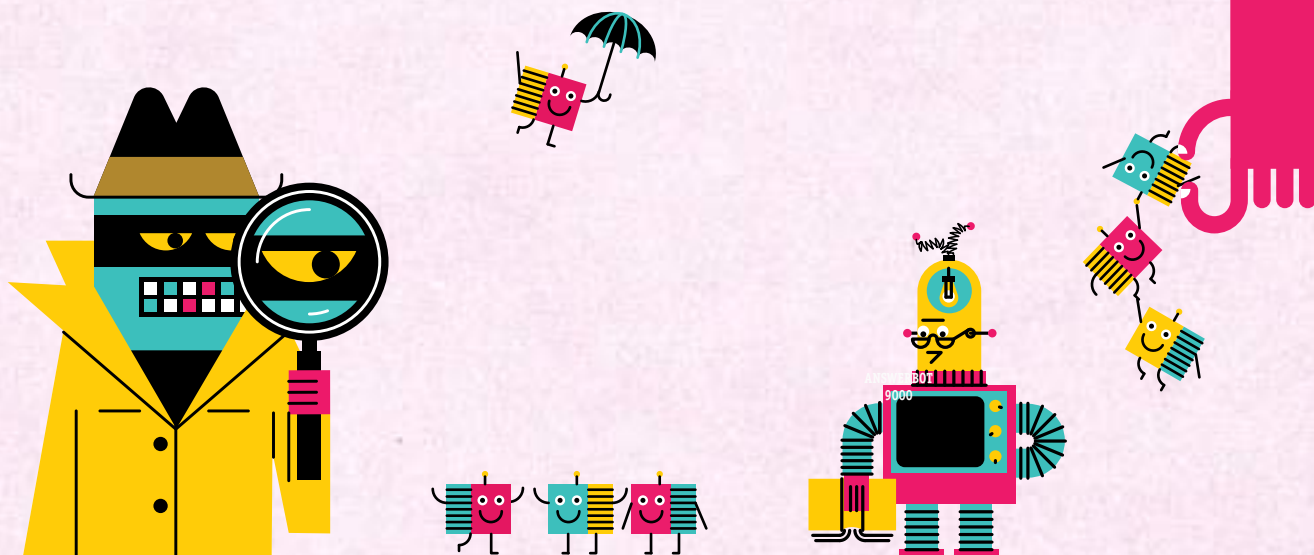
```
    if ballTop > canvasHeight:
        print("Your score was " + str(score))
        playAgain = tkinter.messagebox.askyesno(message="Do you want to play again?")
        if playAgain == True:
            reset()
        else:
            close()
def close():
    global windowOpen
    windowOpen = False
    window.destroy()
def reset():
    global score, bounceCount, batSpeed
    global leftPressed, rightPressed
    global ballMoveX, ballMoveY
    leftPressed = 0
    rightPressed = 0
    ballMoveX = 4
    ballMoveY = -4
    canvas.coords(bat, 10, setBatTop, 50, setBatBottom)
    canvas.coords(ball, 20, setBatTop-10, 30, setBatTop)
```

```
    score = 0
    bounceCount = 0
    batSpeed = 6
window.protocol("WM_DELETE_WINDOW", close)
window.bind("<KeyPress>", on_key_press)
window.bind("<KeyRelease>", on_key_release)
reset()
main_loop()
```

Это начальный вариант условия соприкосновения, который мы заменили в этой части.

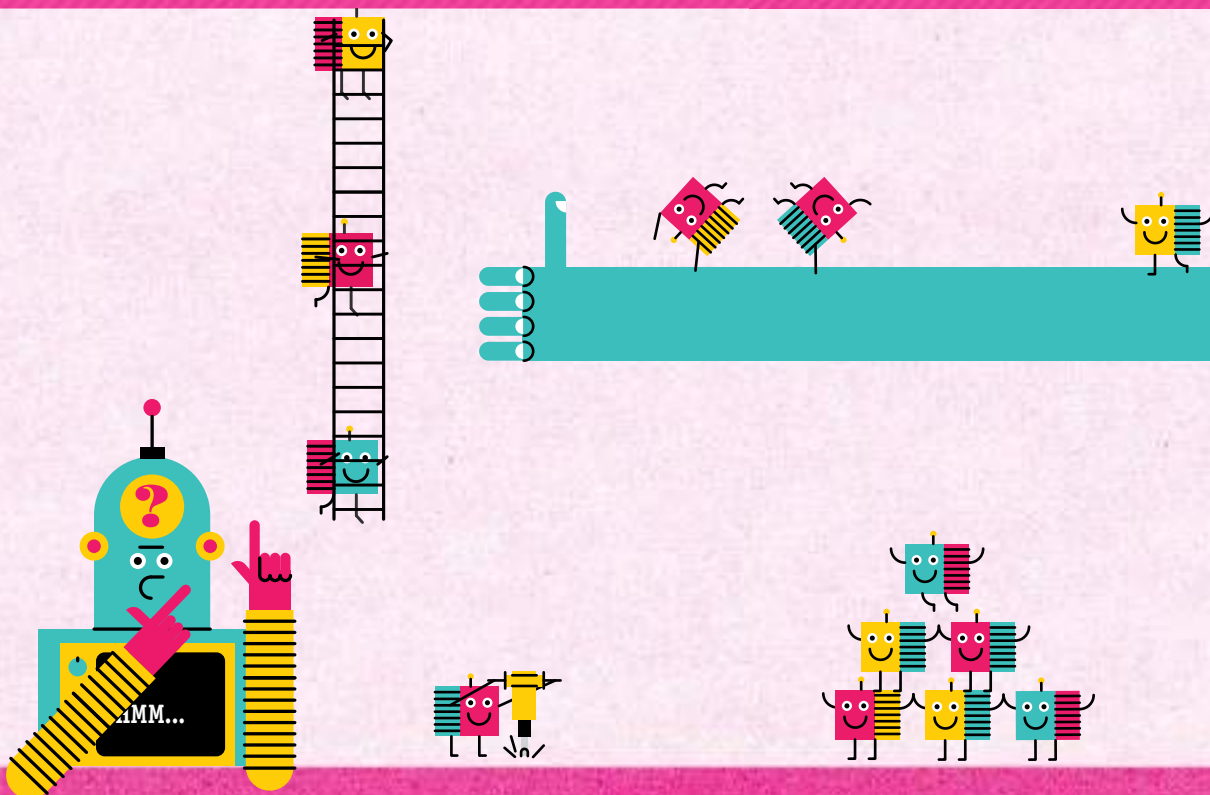
Это новое условие соприкосновения мяча с ракеткой.

ПРОГРАММА НАЧИНАЕТСЯ ЗДЕСЬ...



ПОЛЕЗНЫЕ СВЕДЕНИЯ


В этой части книги вы найдете множество общих советов по языку Python, включая советы по отладке, глоссарий компьютерных терминов и информацию о том, как «читать» компьютерные программы.



СКАЧИВАНИЕ СРЕДЫ PYTHON



Вы можете совершенно бесплатно скачать программную среду Python, если она еще не установлена на вашем компьютере. Здесь мы расскажем вам, как получить среду программирования на компьютер под управлением Windows Vista или более новой операционной системы.

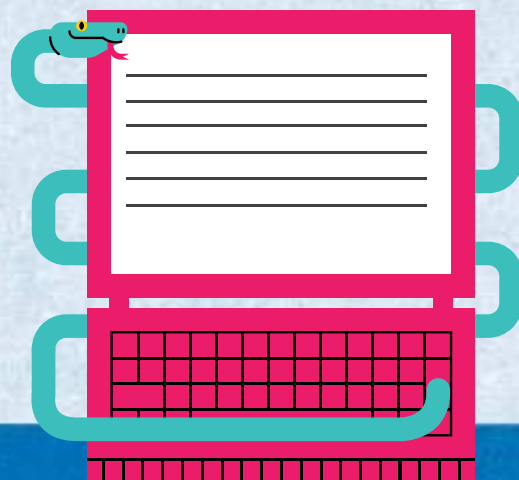
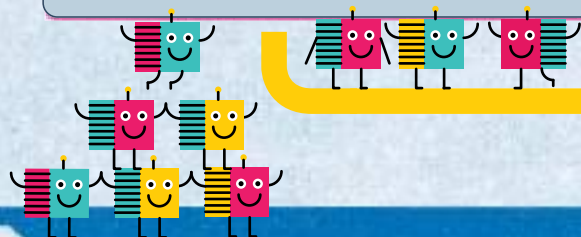
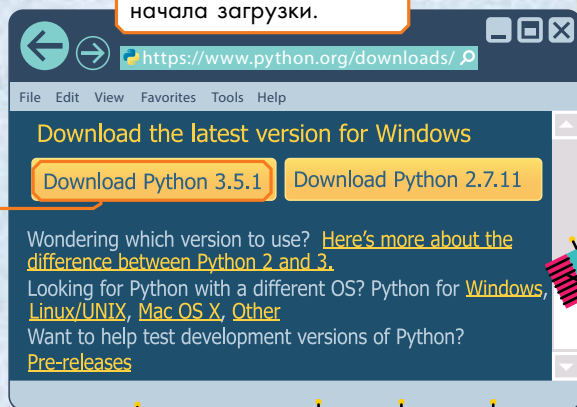
1. Во-первых, проверьте, не установлена ли уже среда программирования Python на вашем компьютере. Перейдите в меню Start и кликните по строчке All Programs (Все программы). Если вы увидите слово «Python» или IDLE, либо все что угодно с иконкой Python (), значит, программная среда Python у вас уже установлена. Посмотрите версию этой программной среды. Номер версии должен начинаться с тройки. В более ранних версиях этой программной среды код примеров из этой книги работать не будет.

2. Если на вашем компьютере программа не установлена или установлена устаревшая версия, нужную версию можно совершенно бесплатно скачать с ресурса Python Software Foundation. Для этого перейдите по ссылке www.usborn.co.uk/quicklinks (Usborne Quicklinks) и наберите слово «Python».

3. Программная среда Python доступна в различных версиях (номер версии вы увидите сразу после имени файла). Для данной книги вам потребуется третья версия, так что подойдет любая, начинающаяся с тройки. Мы рекомендуем вам скачать самую последнюю версию.

4. Установка программы начнется автоматически, а когда она закончится, просто следуйте инструкциям, которые будут появляться на экране. Если вы хотите выбрать иное место инсталляции на вашем компьютере, вам поможет опция «Customize installation», которая позволяет выбирать место для копирования программных файлов и хранения файлов проекта. Например, вы можете сохранять файлы в папке «My Documents» (Мои документы).

Кликните по кнопке для начала загрузки.



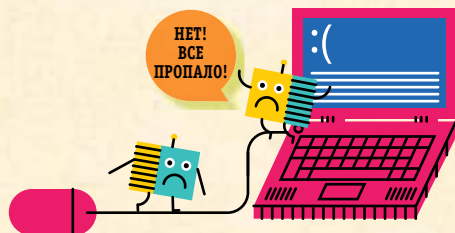
РАБОТА С ФАЙЛАМИ

Когда вы будете использовать программы из этой книги или писать собственные, советуем вам регулярно сохранять файлы.



1. Сохранение файлов

Чтобы сохранить текст программы, войдите в меню File (Файл) программного окна и выберите Save As (Сохранить как). Набейте имя в строчку «File name», затем кликните по кнопке Save (Сохранить). Дописав немного кода, сохраните эти изменения. Иначе вы можете потерять часть программы, особенно если она длинная.



Файлы проекта Python по умолчанию сохраняются в ту же директорию, куда вы устанавливали языковую среду. Чтобы сохранить файл в другую директорию, кликните снова Save As (Сохранить как) и выберите, куда вы хотите сохранить.



2. Наименования файлов

Файлы Python имеют расширение «.py». Давайте файлам осмысленные названия, чтобы проще было находить их. Если вы вносите в файл изменения и хотите сохранить старую версию файла, то в новом имени добавляйте номер версии, например, **mygame_2.py**.



3. Открытие файлов

Найдите нужный файл, кликните по нему правой кнопкой мыши и выберите «Edit with IDLE». Также можно открыть файл, кликнув в программном окне по закладке File и выбрав Open.

! ВНИМАНИЕ! !

Если вы откроете этот файл двойным кликом, компьютер попытается запустить его. Это может привести к ошибке, поскольку файл располагается не в той папке.



4. Удаление файла

Советуем вам по окончании работы над программой удалять старые версии, поскольку они вам больше не нужны и только занимают место на диске. Только перед окончательным удалением проверяйте двойным щелчком работоспособность проекта, чтобы не удалить нужный файл.



5. Оставляйте комментарии

Чтобы оставлять комментарии в программе, надо ставить в начале строки знак #. Эти комментарии могут оказаться полезными не только для других, но и для вас самих спустя время. Например, «Это финальная версия программы. Не стирать!».

```
#This is the final version.  
#Do not delete!
```



ОТЛАДКА ПРОГРАММ

В программировании на этом языке характерны ошибки в написании служебных слов и пропущенные сдвиги строк. Это приводит к остановке программы. Вообще говоря, небольшое количество ошибок при программировании это норма. Ниже мы дадим вам несколько советов по поиску и исправлению некоторых наиболее характерных ошибок.



Узнай свои ошибки

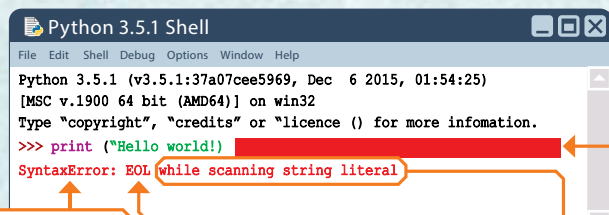
Существует три основных типа ошибок, которые вы можете совершить в программе: синтаксические ошибки (**syntax error**), ошибки периода исполнения (**runtime error**) и логические ошибки (**logic error**). Одни из них исправлять проще, другие — труднее.

1. Синтаксические ошибки наиболее просты для исправления. Это могут быть просто ошибки написания служебных слов, пропущенные двоеточия в конце условия `if` и другие ошибки набора текста. Когда такая ошибка имеет место, программа не может быть исполнена. Например:

```
print("Hello world!")
```

В этой строке пропущены кавычки.

Если вы запустите эту программу в командном окне, то получите сообщение об ошибке в том же командном окне, а ее место будет отмечено красным.



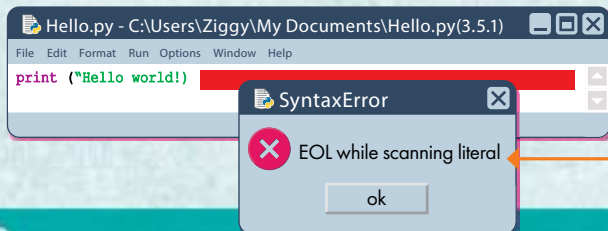
Здесь указано, что ошибка в написании команды.

EOL означает «End Of Line» (конец строки) — место, где возникла ошибка.

Здесь сказано, что проблема не в числе, а в строковом плане (**string**).

Красным цветом подсвечено место возникновения ошибки.

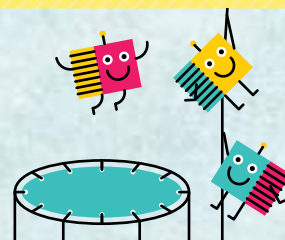
Если вы будете запускать тот же код в виде программы, сохраненной в программном окне, то получите всплывающее сообщение, а в своем программном окне, где ошибка, увидите красную метку.



Здесь та же ошибка, что и выше, но сообщение об ошибке появляется во всплывающем окне.

ОПРОСНЫЙ ЛИСТ ПРИ БЫСТРОЙ ПРОВЕРКЕ ПРОГРАММЫ

- **Правописание:** компьютер не может распознавать написанные с ошибкой служебные слова. Проверьте их написание вплоть до строчной/прописной первой буквы.
- **Отступы строк:** все строки одного блока должны иметь одно и то же смещение (отступ). После двоеточия добавляйте на следующей строке еще один отступ. В конце блока убирайте отступ.
- **Скобки и кавычки:** убедитесь, что все скобки и все кавычки не только открываются, но и закрываются.
- **Двоеточие:** убедитесь, что каждое условие ветвления заканчивается двоеточием (:). Оно должно стоять непосредственно после условия и перед секцией, имеющей дополнительное смещение.



ЕСЛИ ВОЗНИКАЕТ ОШИБКА, ПРОГРАММА НЕ БУДЕТ РАБОТАТЬ:

Если вы не можете найти ошибку в строке, на которую указывает сообщение, то смотрите в ближайших строках выше. Она в принципе не способна распознавать проблему, пока исполнение программы не сталкивается с ошибкой, или «багом». Тогда сообщение об ошибке появляется сразу после «бага» или на следующей строке.

2. Ошибки периода исполнения находить чуть труднее, поскольку они выявляются только при исполнении программы. Примеры таких ошибок — вы используете переменную до ее определения либо совсем забыли определить ее.

Ошибки периода исполнения коварны, как подводные камни. Натолкнувшись на такую ошибку, программа останавливается.

```
print(variable1)
```

Эта инструкция не будет работать, если переменная «variable1» не была определена выше.

Ошибки периода исполнения вызывают сообщения в командном окне. Например, если вы сохраните приведенный выше код в программном окне и запустите его, то в командном окне увидите такое сообщение об ошибке.

Это название программы и путь к ней.

```
Traceback (most recent call last):  
File "C:\Users\Ziggy\My Documents\changes.py", line 1, in <module>  
    print(variable1)  
NameError: name 'variable1' is not defined
```

Здесь указан тип ошибки.

Это описание проблемы (variable1 не определена).

Это строка, в которой произошла ошибка.

Это номер строки, в которой прервалось исполнение программы.

Эти ошибки вызываются некоторыми некорректными действиями с числами, такими как попытка деления на ноль или сложения числовой и строковой переменных.

3. Труднее всего находить логические ошибки, поскольку программа работает и вы не видите никаких сообщений об ошибках. Правда, программа не дает вам то, чего вы от нее ожидаете. Причинами логических ошибок являются ошибки в алгоритме — ввод неправильного значения или использование не того оператора (например, «<» вместо «>»).

В приведенной ниже программе предполагалось получить таблицу умножения вплоть до 12 x 5, но этого не получилось из-за логической ошибки.

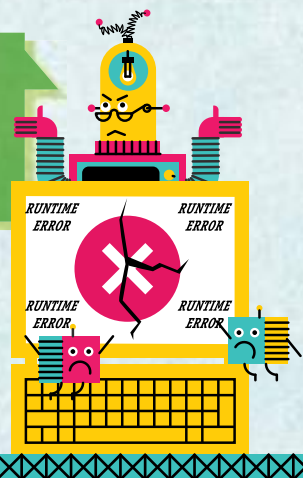
```
for x in range(1,12):  
    print(x, "x 5 =", x*5)
```

В диапазоне 1–12 только 11 чисел.

Если вы сохраните и запустите эту программу, то не получите сообщения об ошибке. Однако не получите вы и полную таблицу умножения, как планировали. Это связано с тем, что вы неправильно указали числовой диапазон.

```
1 x 5 = 5  
2 x 5 = 10  
3 x 5 = 15  
4 x 5 = 20  
5 x 5 = 25  
6 x 5 = 30  
7 x 5 = 35  
8 x 5 = 40  
9 x 5 = 45  
10 x 5 = 50  
11 x 5 = 55
```

Иногда для поиска ошибок в программе по ее тексту расставляют функции print(), как мы делали на стр. 64



Лучше всего в поисках ошибки двигаться из конца в начало, начиная с последней строки, где произошла ошибка



Программное окно используется для написания программы, или кода. Все длинные программы следует писать в этом окне, поскольку оно позволяет сохранять и редактировать текст. Зайдите в программном окне в меню File (Файл), выберите New file (Новый файл) и вы откроете окно для новой программы.

В самом начале **программное** окно совершенно чистое.

В **программном** окне вы пишете как на листе бумаги, не дожидаясь появления никаких командных строк.

Чтобы запустить программу из программного модуля, вам надо кликнуть по опции меню **Run** (Запустить) и выбрать **Run module** (Модуль запуска). На экране появится командное окно.

Если вы запустите программу дважды, то во второй раз увидите в командном окне строчку «RESTART» (Перезапуск).

РАЗБОР ПРОГРАММЫ

Многое можно сказать о программе, просто взглянув на нее беглым взглядом. На этой странице мы покажем, как это сделать. Если вы встретите незнакомое слово, то можете заглянуть в глоссарий, который начинается на следующей странице.

Скобки означают, что это **функция**.

```
print()
```

В скобках при функциях можно привести параметры, которые принимает эта функция.

Кавычки означают, что это строка (**string**).

```
print("Beep boop!")
```

Знак равенства сообщает компьютеру, что переменной (**variable**) нужно присвоить значение. Также этим мы создаем новую переменную.

```
bananas = 5
```

Программные строки с одинаковым смещением (**indent**) являются единой секцией кода.

```
def vshape():  
    right(25)  
    forward(50)
```

Двоеточия устанавливают связи между объектами. Если в конце строки стоит двоеточие, то на следующей строке будет смещение. Это смещение показывает, что новая строка принадлежит верхней.

Квадратные скобки означают список (**list**).

```
spacelist1 = ["rocket", "planet", "asteroid", "alien"]
```

Здесь двоеточие связывает ключ (**key**) с содержанием ячейки словаря.

Фигурные скобки выделяют словарь (**dictionary**).

```
powers = {"The Pigeon": "flight", "Cyborg": "controls machines"}
```

Цикл for часто используется для повторения инструкций заданное количество раз.

```
for x in range(0,10):  
    print(x)
```

Проверяйте **аргументы** — то, что внутри скобок при функциях и указывает им параметры.

Цикл while может использоваться для повтора инструкций неопределенное количество раз, пока выполняется определенное условие.

```
password = "password"  
while password != "Open Sesame":  
    password = input("Enter password.")  
    if password == "Open Sesame":  
        print("Correct!")  
    else:  
        print("Wrong! Try again.")
```

Операторы **if/else** сообщают компьютеру, как реагировать на различные варианты выполнения условий (да/нет).

В редакторе IDLE используются цвета, которые помогают программисту при отладке.

- **Фиолетовый** обозначает встроенные функции.
- **Синий** означает определяемые функции, также им отмечается вывод программ в командном окне.
 - **Зеленый** обозначает строки.
 - **Оранжевый** — ключевые слова.
 - **Красный** — сообщения об ошибках.
 - Черный — все прочее!



Умение быстро читать код поможет вам понимать программы и легче находить в них ошибки





ГЛОССАРИЙ



elif — элемент инструкции ветвления `else-if`. Специфическая для языка Python команда, которая активирует второе ветвление, если условие перехода не выполняется (`False`).

IDLE — редактор, встроенный в среду Python и облегчающий процесс написания программ, их редактирования, сохранения и запуска.

in — специфическое служебное слово языка Python, используемое, чтобы указывать на переменную типа `range`.

randint() — в языке Python эта функция генерирует случайное число между двумя целыми числами — параметрами этой функции, приводимыми в скобках.

range() — в языке Python эта функция возвращает список чисел.

statement — в программировании так называется часть кода, которая описывает действие, производимое оператором.

Анимация — ряд графических образов, демонстрируемых в определенной последовательности, чтобы создать иллюзию движения объекта.

Аргумент — в языке Python значение, которое передается функции в качестве параметра (см. отдельную статью).

Баг — ошибка в коде, которая не позволяет программе работать правильно.

Байт — единица количества компьютерных данных. Также см. **Мегабайт**.

Библиотека — в языках программирования это заранее написанная часть кода, которую можно импортировать в память компьютера для использования в вашей программе. См. также **Модуль**.

Блок — в языке Python так называют секцию кода, которая имеет один и тот же отступ и исполняется как единое целое.

Блок-схема — тип диаграммы, которая используется для планирования каждого этапа выполнения программы.

Булева логика — реализуемый во всех компьютерах подход к вычислениям, при котором принятие любого решения разбивается на ряд простых вопросов, на которые можно ответить да или нет.

Булево выражение — способ сопоставления двух частей информации с использованием булевой логики.

Булево значение — статус булева выражения, который представляют в виде «True» (Правда)/«False» (Ложь) или Да/Нет.

Ввод — информация или инструкция, которые вы вводите в компьютер. В языке Python функция `input()` используется для запроса информации у пользователя.

Веб-сайт — страница, а чаще несколько страниц, которые вы можете просматривать в сети Интернет.

Ветвление — пути, по которым может быть направлено вычисление в зависимости от того, верно условие перехода (`True`) или нет (`False`). Если условие выполняется, то активируется один сектор кода, если нет — то другой.

Вложенный цикл — цикл внутри другого цикла.

Возвращаемая величина — в языке Python так называется результат, который дает функция при вызове.

Вывод — результат, который вы получаете от компьютера.

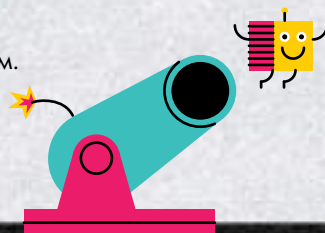
Выгружать — так называется посылка файлов или данных с вашего компьютера в другое место. Обычно выгрузка происходит на сетевой ресурс, после чего этими данными могут пользоваться (или просто их просматривать) любые пользователи данного Интернет-ресурса.

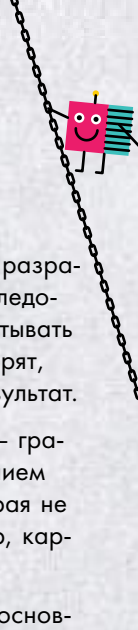
Вызов функции — в языке Python так называется использование или запуск функции.

Выпадающее меню — список пунктов меню, который появляется, когда вы кликаете по нему мышью.

Глобальная переменная — в языке Python так называют переменную, которая создается снаружи всех функций и может быть использована с тем же значением внутри любой из функций.

Глобальный — в языке Python есть ключевое слово `global`, которое позволяет изменять значения глобальных переменных внутри функций.





Данные — информация, используемая компьютером. Если данные могут меняться, они становятся переменными. (Переменные бывают разного типа, включая отдельные значения или целый список.) Часть данных, которая не меняется, может представлять собой определенные константы. См. также **Строковый тип данных**.

Двойной клик — быстрое двойное нажатие левой кнопки мыши.

Если — служебное слово `if` (если) используется для проверки условий перехода на истинность. Если условие удовлетворяется (`True`), то компьютер переходит в следующей секции кода. Если нет (`False`), то пропускает ее.

Загрузка, или скачивание — сохранение чего-либо из Интернета на жесткий диск компьютера.

Запустить — сделать так, чтобы часть кода или вся программа начали исполняться.

Иконка — небольшие изображения, которые представляют в операционной системе файл, папку или элемент контроля.

Импорт — в языке Python служебное слово `import` используется для загрузки в память компьютера вместе с вашей программой модуля со встроенными функциями.

Имя файла — то, как мы называем файл, когда сохраняем его на компьютере.

Иначе — элемент инструкции ветвления `else`, который активирует свой сегмент кода, когда условие первого перехода (`if`) и условие второго перехода (`elif`) не выполняются (`False`).

Индекс — в математике и вычислительной технике положение

элемента в списке (`list`). В языке Python отсчет начинается с нуля.

Интернет — огромная сеть, которая позволяет компьютерам по всему миру взаимодействовать друг с другом.

Истина — значение булевого выражения (`True`), когда условие выполняется. В языке Python всегда пишется с большой буквы.

Клик — выбор чего-либо на экране щелчком указателя мыши. Обычно щелчок производится левой кнопкой мыши, если только об этом будет сказано особо.

Ключ — в языке Python служебное слово `key` (ключ) позволяет в словаре (`dictionary`) получить доступ к содержимому ячейки по ее названию (собственно, ключу).

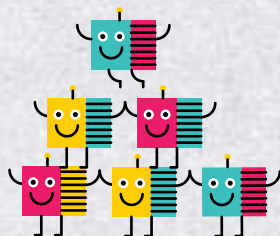
Код — инструкции для компьютера, записанные на языке программирования.

Команда — часть программы, которая сообщает компьютеру, какую операцию выполнять.

Командная строка — в языке Python стрелки в командном окне, которые указывают, где набирать строчки кода.

Командное окно — в языке Python так называется окно для запуска и тестирования кода.

Комментарий — дополнительная информация для тех, кто читает программу. В языке Python неисполняемые строчки комментария начинаются с символа `#`.



Компьютер — устройство, разработанное для того, чтобы следовать инструкциям и обрабатывать данные. На входе, как говорят, данные, а на выходе — результат.

Компьютерная графика — графика, достигаемая изменением элементов на экране, которая не является текстом. Например, картинки, иконки и символы.

Компьютерная логика — основные правила, по которым функционирует компьютер.

Константа — в программировании так называется часть данных, которые не меняются. Противоположность ей *переменные*, которые меняются.

Конфигурация — так в мире вычислительной техники называют настройки.

Координата x — число, которое показывает, насколько левее или правее находится нечто на экране.

Координата y — число, которое показывает, насколько ниже или выше находится нечто на экране.

Координаты — способ разделения пространства сеткой для измерения расстояний, чтобы вы могли определять положение объекта по координатам `x` (лево/право) и `y` (вверх/вниз).

Курсор — моргающая линия, которая показывает, где на экране появится то, что вы напечатаете. Также этот термин используется для указателя мыши.

Ложь — значение булевого выражения (`False`), когда условие не выполняется. В языке Python всегда пишется с большой буквы. См. также **Истина** (`True`).

Мегабайт — чуть больше одного миллиона байт (1 048 576 байт).

Меню — список опций — того, что позволяет программа. Часто меню располагается в верхней части окна.

Модуль — в языке Python программный файл можно запускать на исполнение или преобразовывать из него библиотеку (library) функций, которую вы можете импортировать (import) в память компьютера при исполнении своих программ.

Модуль random — в языке Python так называется модуль, который генерирует случайные числа.

Модуль tkinter — в языке Python так называется библиотека (library), используемая для создания графики, особенно для пользовательского графического интерфейса.

Модуль turtle — в языке Python так называется библиотека (library), используемая для создания графики. Курсор в окне этого модуля называют «черепашкой» (turtle), поскольку в одном из стилей он похож на черепашку.

Обновление экрана — процедура, когда компьютер обновляет картинку на экране.

Окно — в операционной системе так называется ограниченная рамкой область экрана, показывающая информацию и связанная с определенной программой.

Оператор — математический символ, такой как «+», «-», «/» (деление) и «*» (умножение). См. стр. 9. Также см. **Операторы сравнения**.

Операторы сравнения — в языке Python для сравнения двух частей информации используются специальные операторы, такие как «==». См. стр. 16.

Определение — в языке Python ключевое слово **def** (define, или «определять») используется для создания пользовательской функции.

Отладка — исправление текста программы с целью устранения ошибок (см. **Bar**).

Палитра — в информатике так называется показ доступных опций (обычно это цвета).

Папка — способ группировки различных файлов на компьютере, который выбирается при сохранении.

Параметры — в языке Python так называются переменные, которые стоят внутри скобок при функциях.

Переменная — отдельное имя, присваиваемое в компьютерной программе кусочку информации. Таким образом, этот кусочек информации можно использовать в вычислениях, для чего отслеживают его изменение.

Перетаскивание — в визуальных средах программирования так называется перенос какого-либо элемента мышью при нажатой кнопке.

Печать — в языке Python используется функция **print()**, которая выводит информацию на экран.

Пиксели — цветные точки, которые составляют рисунок на экране.

Питон — так по-русски принято называть язык программирования Python, которому посвящена эта книга.

Плавающая запятая — в вычислительной технике любое десятичное число, которое включает в себя целую и дробную часть, называется числом с плавающей запятой.



Показатель степени — часть степенного выражения, которое показывает, сколько раз число должно быть помножено само на себя. Например, 3^4 означает $3 \times 3 \times 3 \times 3$, где 4 — это показатель степени.

Пользовательский графический интерфейс — кнопки и иконки, которые позволяют вам взаимодействовать с компьютером.

Правый щелчок мыши — клик по правой кнопке мыши.

Прерывание (break) — команда, которая выводит компьютер из режима исполнения программного цикла.

Программа — набор инструкций, написанных на языке программирования, которые указывают компьютеру, что делать.

Программирование — написание инструкций для компьютера.

Программное окно — в языке Python это окно используется для написания, редактирования и сохранения больших фрагментов программ.

Процедура — так называется часть кода, которая имеет отдельное имя и используется многократно по вызову.

Расширение файла — буквенный код после точки и имени файла, который сообщает компьютеру, какого рода информация хранится в файле. Например, .py указывает на то, что это файл проекта Python.





Связывание — в языке Python установление связи между кодом и отдельным объектом на экране (или событием).

Сдвиг строк — пустое пространство в начале строк. В языке Python сдвиги строк используются для группирования программных строк в блоки.

Синтаксис — способ написания кода, чтобы компьютер мог понимать его.

Скобки () — в математике и вычислительной технике скобки используются для группирования в выражениях. В языке Python скобки обязательны после функций и между ними ставят параметры выполнения этих функций.

Словарь — в языке Python ключевое слово **dictionary** (англ. словарь), которое используется для структурированного хранения информации. Каждая ячейка имеет название — ключ (key), по которому может «извлекаться» содержимое ячейки.

Служебное слово — фиксированное слово с четко определенным для компьютера значением. Например, оператор **if** в языке Python.

Случайный — так говорят о величине, рассчитанной не по какой-либо системе, а потому непредсказуемой.

Событие — клик мышью или нажатие клавиши, которое дает сигнал программе.

Сохранить — записать компьютерный файл таким образом, чтобы потом использовать его снова.



Список — способ организации информации на компьютере. Каждый элемент списка индексируется, начиная не с единицы, а с нуля.

Строковый тип данных — в программировании так называется последовательность букв. Также это может быть последовательность цифр, которые компьютер обрабатывает как символы (а не как числа).

Удаление — стирание чего-либо из памяти компьютера.

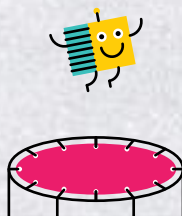
Указатель мыши — стрелка, которую вы видите на экране и можете перемещать при помощи манипулятора-мыши.

Условие — в программировании так называется величина, которую компьютер должен оценить, прежде чем выберет направление расчетов.

Условное выражение — инструкция, которая говорит компьютеру, как реагировать на различные значения выражения, стоящего в таких операторах, как ветвление (if/else).

Файл — набор данных, хранящихся под общим именем на компьютере. Файлы различных типов имеют после имени и точки различные буквы расширения. См. также **Модуль**.

Функция — в языке Python так называются секции кода, отвечающие за конкретные задачи. Например, **print()** или **input()**.



Холст — в языке Python чистый экран с координатами X и Y.

Целое число — в языке Python функция **int()**, от слова «integer» (целый), используется для преобразования числовой или строковой переменной в целочисленную переменную.

Целочисленный — не имеющий дробной части.

Цикл — часть кода, исполняемая в программе несколько раз. См. также **вложенные циклы**, **циклы типа for** и **while**.

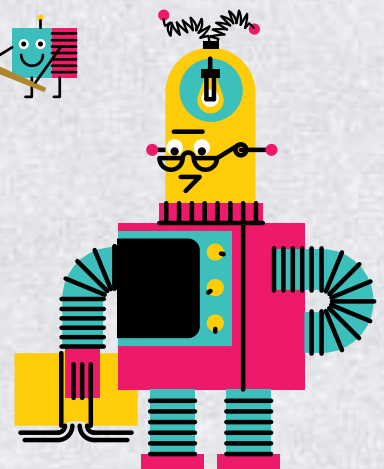
Цикл for — вид организации цикла, при котором компьютер повторяет инструкции строго определенного количества раз.

Цикл while — цикл, инструкции в котором повторяются до тех пор, пока выполняется определенное условие (True).

Шифрование — преобразование информации по секретному алгоритму, чтобы ее нельзя было прочитать.

Эллипс — овал, получающийся при вытягивании круга.

Язык программирования — специально разработанный для компьютера язык с четко определенными словами и синтаксисом. Язык Python не исключение.



УДК 087.5:551.5
ББК 26.23
П78

Научно-популярное издание
Для младшего школьного возраста

ПРОГРАММИРОВАНИЕ ДЛЯ ДЕТЕЙ НА ЯЗЫКЕ PYTHON

Coding for Beginners using Python

Перевод *Александра Банкрашкова*

Оформление обложки *Нatalьи Ворламовой*
Редактор *К.Г. Петров*
Художественный редактор *Е.А. Гордеева*
Технический редактор *Е.П. Кудиярова*
Компьютерная верстка *Е.Б. Гвоздева*

Общероссийский классификатор продукции ОК-005-93, том 2; 953000 — книги, брошюры

Подписано в печать 06.09.2017

Формат 84х108/16. Бумага офсетная. Печать офсетная

Усл. печ. л. 3,51 Тираж экз. Заказ №



ООО «Издательство АСТ»

129085 г. Москва, Звездный бульвар, д. 21, строение 1, комната 39

Наш электронный адрес: malysh@ast.ru

Мы в социальных сетях. Присоединяйтесь!

https://vk.com/AST_planetadetstva

https://www.instagram.com/AST_planetadetstva

<https://www.facebook.com/ASTplanetadetstva>

«Баспа Аста» деген ООО. 129085 г. Мәскеу, жұлдызды гүлзар, д. 21, 1 құрылым, 39 бөлме

Біздің электрондық мекенжайымыз: www.ast.ru. E-mail: malysh@ast.ru

Қазақстан Республикасында дистрибьютор және өнім бойынша арыз-талаптарды қабылдаушының өкілі

«РДЦ-Алматы» ЖШС, Алматы қ., Домбровский көш., 3«а», литер Б, офис 1.

Тел.: 8(727) 251 59 89,90,91,92, факс: 8 (727) 251 58 12 вн. 107; E-mail: RDC-Almaty@eksmo.kz

Өнімнің жарамдылық мерзімі шектелмеген. Өндірген мемлекет: Ресей. Сертификация қарастырылған

П78 Программирование для детей на языке Python — Москва: Издательство АСТ — 2017. — 95, [1] с.: ил.

ISBN 978-5-17-982809-9.

Язык Python — активно развивающийся язык программирования, который, в силу своей простоты и прозрачности, легко освоит даже ребенок. Им пользуются для написания программ многие крупные организации, такие как Google, NASA и YouTube.

Интересные проекты, яркие иллюстрации, понятные инструкции — благодаря всему этому можно запросто разобраться в основах программирования, понять логику работы компьютера, что в дальнейшем позволит легко перейти к программированию и на других языках.

Книга содержит множество задач, которые были написаны исходя из интересов юного читателя: здесь вы найдете подробные инструкции по созданию забавных игр, на примере которых демонстрируется вся красота программирования.

Для младшего школьного возраста.

УДК 087.5:551.5

ББК 26.23



© Usborne Publishing Ltd., 2018

© Банкрашков А., пер., 2018

© ООО «Издательство АСТ», 2018

© Copyright GEMSER PUBLICATIONS S.L., 2018



ПРОГРАММИРОВАНИЕ ДЛЯ ДЕТЕЙ НА ЯЗЫКЕ PYTHON

На языке Python написано множество популярных программ и игр: BitTorrent, Civilization IV, Battlefield 2, World of Tanks.

Этим языком пользуются различные крупные компании: Яндекс, Google, Facebook, IBM, NASA и многие другие.

С этой книгой любой ребёнок сможет научиться основам программирования благодаря пошаговым инструкциям.

Это не скучно! Ведь программируя можно создавать собственные игры.

Аванта

6+

EAC

www.asl.ru

ISBN 978-5-17-962809-9



9 785179 828099