[**Чтение 0**](https://cs50.harvard.edu/x/2021/notes/0/#lecture-0)

* [Добро пожаловать](https://cs50.harvard.edu/x/2021/notes/0/#welcome)
* [Что такое информатика?](https://cs50.harvard.edu/x/2021/notes/0/#what-is-computer-science)
* [Представление чисел](https://cs50.harvard.edu/x/2021/notes/0/#representing-numbers)
* [Текст](https://cs50.harvard.edu/x/2021/notes/0/#text)
* [Изображения, видео, звуки](https://cs50.harvard.edu/x/2021/notes/0/#images-video-sounds)
* [Алгоритмы](https://cs50.harvard.edu/x/2021/notes/0/#algorithms)
* [Псевдокод](https://cs50.harvard.edu/x/2021/notes/0/#pseudocode)
* [Царапать](https://cs50.harvard.edu/x/2021/notes/0/#scratch)

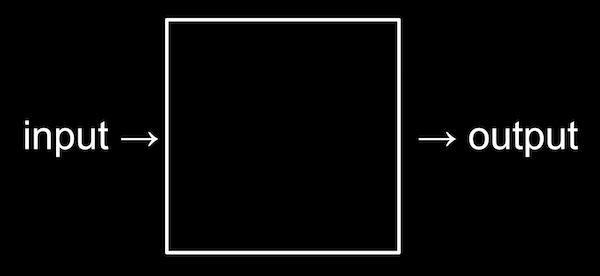
[**Добро пожаловать**](https://cs50.harvard.edu/x/2021/notes/0/#welcome)

* В этом году мы будем в [Драматическом центре Леба](https://americanrepertorytheater.org/venue/loeb-drama-center-3/) при Гарвардском университете, где, благодаря нашему тесному сотрудничеству с [Американским репертуарным театром](https://americanrepertorytheater.org/) , у нас есть потрясающая сцена и даже реквизит для демонстраций.
* Мы превратили [акварельную картину](https://images.hollis.harvard.edu/permalink/f/100kie6/HVD_VIAolvwork671391) 18-го века студента Джонатана Фишера в [Гарвардском кампусе](https://images.hollis.harvard.edu/permalink/f/100kie6/HVD_VIAolvwork671391) в фон для сцены.
* Двадцать лет назад, будучи студентом, Дэвид преодолел собственное беспокойство, вышел за пределы своей зоны комфорта и сам прошел CS50, обнаружив, что курс был не столько о программировании, сколько о решении проблем.
* Фактически, две трети студентов CS50 никогда раньше не посещали курсы информатики.
* И что немаловажно:

что в конечном итоге имеет значение в этом курсе, так это не столько то, где вы закончите по отношению к своим одноклассникам, а где вы закончите по отношению к себе, когда вы начали

* Мы начнем курс с воссоздания компонента игры [Super Mario](https://en.wikipedia.org/wiki/Super_Mario_Bros.) , позже создадим веб-приложение под названием CS50 Finance, которое позволит пользователям виртуально покупать и продавать акции, и закончим курс созданием вашего собственного финального проекта.

[**Что такое информатика?**](https://cs50.harvard.edu/x/2021/notes/0/#what-is-computer-science)

* Информатика - это фундаментальное решение проблем.
* Мы можем думать о **решении проблемы** как о процессе принятия некоторых входных данных (подробностей о нашей проблеме) и генерирования некоторых выходных данных (решение нашей проблемы). «Черный ящик» посередине - это информатика или код, который мы научимся писать.  
  
* Чтобы начать это делать, нам понадобится способ представления входных и выходных данных, чтобы мы могли хранить информацию и работать с ней стандартизованным способом.

[**Представление чисел**](https://cs50.harvard.edu/x/2021/notes/0/#representing-numbers)

* Мы могли бы начать с определения посещаемости, подсчитав количество людей в комнате. Мы можем поднимать руку по одному пальцу, чтобы представить каждого человека, но мы не сможем сосчитать очень высоко. Эта система называется **унарной** , где каждая цифра представляет собой единичное значение.
* Вероятно, мы узнали более эффективную систему представления чисел, где у нас есть десять цифр от 0 до 9:
* 0 1 2 3 4 5 6 7 8 9
  + Эта система называется десятичной, или **основание 10** , так как существует десять различных значений , что цифра может представлять.
* Компьютеры используют более простую систему, называемую **двоичной** , или основание два, только с двумя возможными цифрами, 0 и 1.
  + Каждую *двоичную цифру* также называют **битом** .
* Поскольку компьютеры работают от электричества, которое можно включить или выключить, мы можем удобно представить бит, включив или выключив какой-либо переключатель, чтобы представить 0 или 1.
  + Например, с помощью одной лампочки мы можем включить ее, чтобы считать до 1.
* С тремя лампочками мы можем включать их по-разному и считать от 0 (все три выключены) до 7 (все три включены):  
    
    
    
    
    
    
    
  
* Внутри современных компьютеров есть не лампочки, а миллионы крошечных переключателей, называемых **транзисторами,** которые можно включать и выключать для отображения различных значений.
* Например, мы знаем, что следующее десятичное число представляет сто двадцать три.
* 1 2 3
  + Знак 3находится в столбце единиц, знак 2- в столбце десятков, а 1- в столбце сотен.
  + Так и 123есть 100×1 + 10×2 + 1×3 = 100 + 20 + 3 = 123.
  + Каждое место в цифре представляет собой степень десяти, поскольку для каждого места существует десять возможных цифр. Крайнее правое место - 10 0 , среднее - 10 1 и крайнее левое место - 10 2 :
  + 102 101 100
  + 1 2 3
* В двоичном формате, состоящем всего из двух цифр, у нас есть степени двойки для каждого разряда:
* 22 21 20
* # # #
  + Это эквивалентно:
  + 4 2 1
  + # # #
* Когда все лампочки или выключатели выключены, у нас все равно будет значение 0:
* 4 2 1
* **0 0 0**
* Теперь, если мы изменим двоичное значение, скажем, 0 1 1на десятичное значение будет 3, поскольку мы добавляем 2 и 1:
* 4 2 1
* **0 1 1**
* Если бы у нас было еще несколько лампочек, у нас могло бы быть двоичное значение 110010, которое имело бы эквивалентное десятичное значение 50:
* 32 16 8 4 2 1
* **1 1 0 0 1 0**
  + Обратите внимание на это 32 + 16 + 2 = 50.
* Чем больше битов, тем выше число.

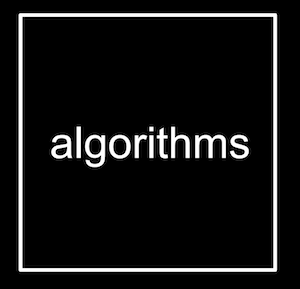
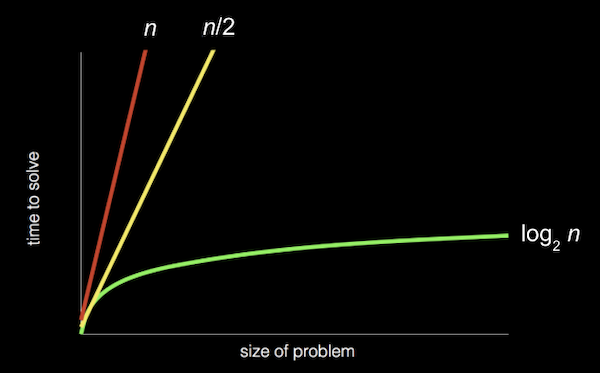
[**Текст**](https://cs50.harvard.edu/x/2021/notes/0/#text)

* Чтобы представить буквы, все, что нам нужно сделать, это решить, как числа соответствуют буквам. Некоторые люди много лет назад коллективно приняли решение о стандартном преобразовании чисел в буквы. Например, буква «А» - это число 65, «В» - это 66 и так далее. Используя контекст, например, смотрим ли мы на электронную таблицу или электронное письмо, разные программы могут интерпретировать и отображать одни и те же биты как числа или текст.
* Стандартное отображение [**ASCII**](https://en.wikipedia.org/wiki/ASCII) также включает строчные буквы и знаки препинания.
* Если мы получили текстовое сообщение с рисунком бит , которые имели десятичные значения 72, 73и 33эти биты будут отображаться на буквы HI!. Каждая буква , как правило , представлены с узором из восьми бит, или **байт** , так что последовательности битов , мы получили бы это 01001000, 01001001и 00100001.
  + Возможно, мы уже знакомы с использованием байтов в качестве единицы измерения данных, например, в мегабайтах или гигабайтах для миллионов или миллиардов байтов.
* С восемью битами или одним байтом мы можем иметь 2 8 или 256 различных значений (включая ноль). (Максимальное *значение, которое* мы можем подсчитать, будет 255.)
* Другие символы, такие как буквы с диакритическими знаками и символы на других языках, являются частью стандарта [**Unicode**](https://en.wikipedia.org/wiki/Unicode) , который использует больше битов, чем ASCII, для размещения всех этих символов.
  + Когда мы получаем смайлик, наш компьютер фактически просто получает число в двоичном формате, которое затем сопоставляется с изображением смайлика на основе стандарта Unicode.
    - Например, смайлик «лицо со слезами радости» - это всего лишь кусочки 000000011111011000000010:  
      

[**Изображения, видео, звуки**](https://cs50.harvard.edu/x/2021/notes/0/#images-video-sounds)

* Изображение, как и изображение смайлика, состоит из цветов.
* Используя только биты, мы также можем отображать числа в цвета. Существует множество различных систем для представления цветов, но наиболее распространенной является **RGB** , которая представляет разные цвета, указывая количество красного, зеленого и синего в каждом цвете.
* Например, наша модель битов раньше, 72, 73и 33может указывать количество красного, зеленого и синего в цвете. (И наши программы будут знать, что эти биты отображаются в цвет, если мы откроем файл изображения, а не получим их в текстовом сообщении.)
  + Каждое число может быть байтом с 256 возможными значениями, поэтому с тремя байтами мы можем представить миллионы цветов. Наши три байта сверху будут представлять темный оттенок желтого:  
    
* Точки или квадраты на наших экранах называются **пикселями** , и изображения также состоят из многих тысяч или миллионов этих пикселей. Итак, используя три байта для представления цвета каждого пикселя, мы можем создавать изображения. Мы можем увидеть пиксели в эмодзи, если увеличим масштаб, например:  
  
* **Разрешение** образа это число пикселей есть, по горизонтали и по вертикали, так что с высокой разрешающей способностью изображение будет иметь большее количество пикселей и требуют более байтов , которые будут сохранены.
* Видео состоят из множества изображений, которые меняются несколько раз в секунду, чтобы создать впечатление движения, как в старомодном [флипбуке](https://youtu.be/p3q9MM__h-M) .
* Музыка также может быть представлена ​​битами, с отображением чисел на ноты и длительность, или более сложными отображениями битов на звуковые частоты в каждый момент времени.
* Форматы файлов, такие как JPEG и PNG, или документы Word или Excel, также основаны на каком-то стандарте, с которым согласились некоторые люди, для представления информации в битах.

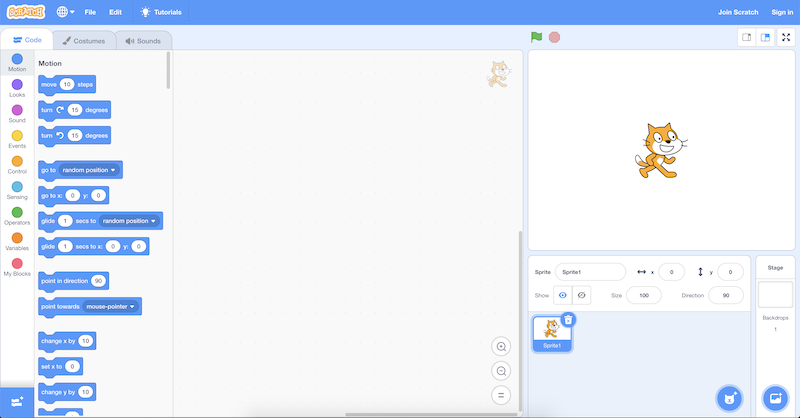
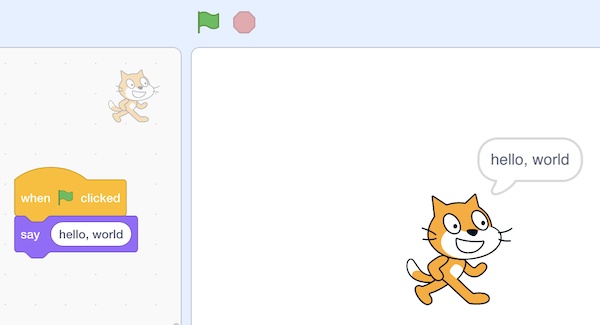
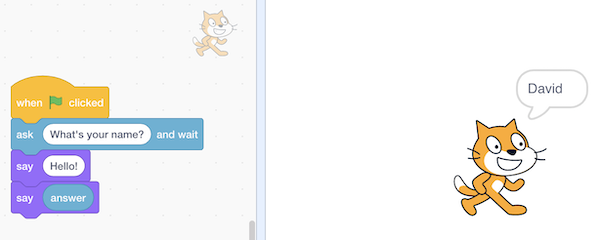
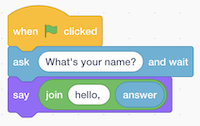
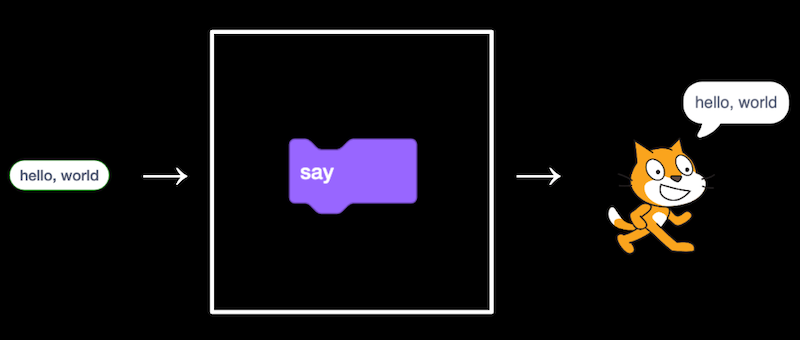
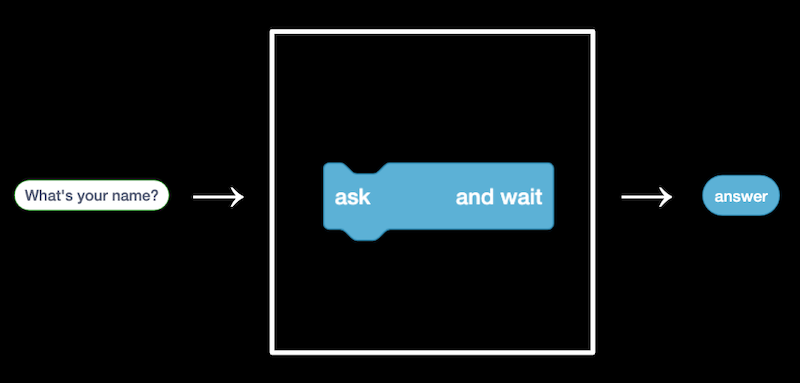
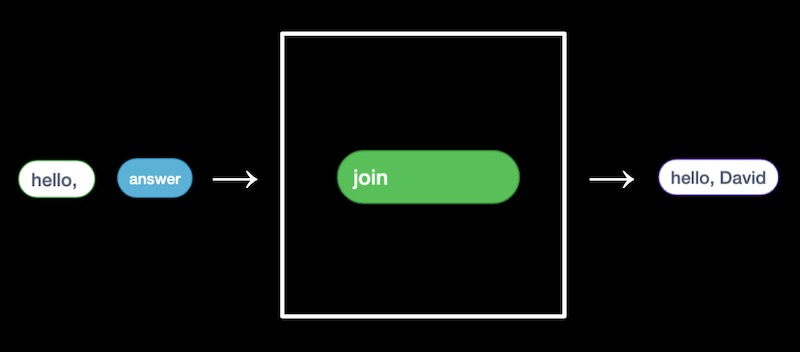
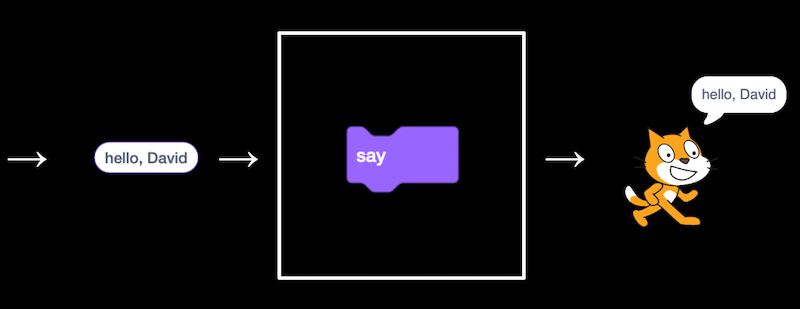
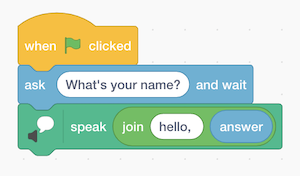
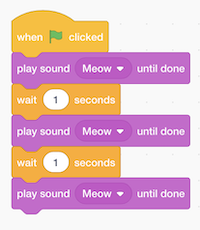
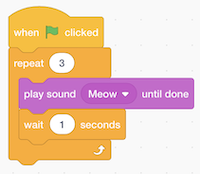
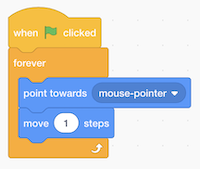
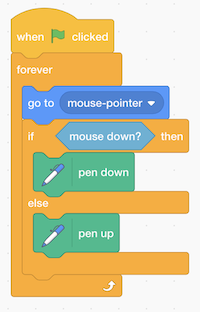
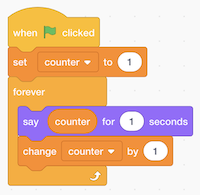
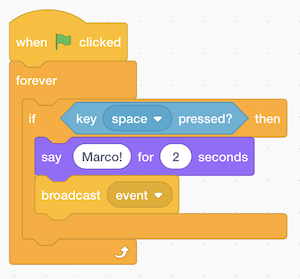
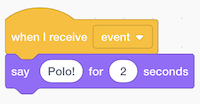
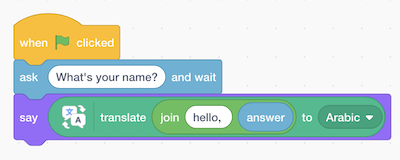
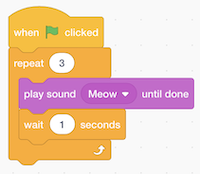
[**Алгоритмы**](https://cs50.harvard.edu/x/2021/notes/0/#algorithms)

* Теперь, когда мы можем представлять входы и выходы, мы можем работать над решением проблем. Черный ящик ранее будет содержать **алгоритмы** , пошаговые инструкции по решению проблем:  
  
* Люди тоже могут следовать алгоритмам, например рецептам приготовления. При программировании компьютера нам нужно быть более точными с нашими алгоритмами, чтобы наши инструкции не были двусмысленными или неверно истолкованными.
* У нас может быть приложение на наших телефонах, которое хранит наши контакты с их именами и номерами телефонов, отсортированными в алфавитном порядке. Эквивалент старой школы может быть телефонной книгой, печатной копией имен и телефонных номеров.
* Нашим вкладом в задачу поиска чьего-либо номера будет телефонная книга и имя, которое нужно искать. Мы можем открыть книгу и начать с первой страницы, ища имя по одной странице за раз. Этот алгоритм был бы **правильным** , поскольку мы, в конце концов, найдем имя, если оно есть в книге.
* Мы можем пролистывать книгу по две страницы за раз, но этот алгоритм будет неверным, так как мы можем пропустить страницу с нашим именем на ней. Мы можем исправить эту **ошибку** или ошибку, вернувшись на одну страницу назад, если пролистаем слишком далеко, поскольку мы знаем, что телефонная книга отсортирована по алфавиту.
* Другой алгоритм - открыть телефонную книгу до середины, решить, будет ли наше имя в левой или правой половине книги (потому что книга составлена ​​по алфавиту), и уменьшить размер нашей задачи вдвое. Мы можем повторять это, пока не найдем свое имя, каждый раз разделяя проблему пополам. С 1024 страницами для начала нам потребуется всего 10 шагов деления пополам, прежде чем у нас останется только одна страница для проверки. Мы можем увидеть это наглядно в [анимации многократного деления телефонной книги пополам](https://youtu.be/F5LZhsekEBc) по сравнению с [анимацией поиска по одной странице за раз](https://youtu.be/-yTRajiUi5s) .
* Фактически, мы можем представить эффективность каждого из этих алгоритмов в виде диаграммы:  
  
  + Наше первое решение, поиск по одной странице за раз, может быть представлено красной линией: наше время на решение линейно увеличивается с увеличением размера проблемы. *n* - некоторое число, представляющее размер проблемы, поэтому с *n* страницами в наших телефонных книгах мы должны предпринять до *n* шагов, чтобы найти имя.
  + Второе решение, выполняющее поиск на двух страницах одновременно, может быть представлено желтой линией: наш наклон менее крутой, но все же линейный. Теперь нам нужно всего (примерно) *n* / 2 шагов, так как мы переворачиваем две страницы за раз.
  + Наше окончательное решение, каждый раз разделяя телефонную книгу пополам, может быть представлено зеленой линией с принципиально другим соотношением между размером проблемы и временем ее решения: [**логарифмическим**](https://en.wikipedia.org/wiki/Logarithm) , поскольку время, необходимое для решения, увеличивается все больше и больше. медленно по мере увеличения размера проблемы. Другими словами, если бы в телефонной книге было от 1000 до 2000 страниц, нам потребовался бы только один шаг, чтобы найти свое имя. Если бы размер снова удвоился с 2000 до 4000 страниц, нам все равно понадобился бы еще один шаг. Зеленая линия помечена как log 2*n* или log base 2 of *n* , так как мы делим задачу на два с каждым шагом.
* Когда мы пишем программы с использованием алгоритмов, нас обычно заботит не только то, насколько они правильны, но и насколько **хорошо** они **спроектированы** с учетом таких факторов, как эффективность.

[**Псевдокод**](https://cs50.harvard.edu/x/2021/notes/0/#pseudocode)

* Мы можем написать **псевдокод** , который представляет наш алгоритм на точном английском (или другом человеческом языке):
* 1 Возьмите телефонную книгу
* 2 Открыто до середины телефонной книги
* 3 Посмотрите на страницу
* 4 Если человек находится на странице
* 5 Позвонить человеку
* 6 Иначе, если человек в книге раньше
* 7 Открыть до середины левой половины книги
* 8 Вернуться к строке 3
* 9 В противном случае, если человек находится позже в книге
* 10 Открыта до середины правой половины книги
* 11 Вернуться к строке 3
* 12 Еще
* 13 Выйти
  + С помощью этих шагов мы проверяем среднюю страницу, решаем, что делать, и повторяем. Если человека нет на странице, и в книге больше не осталось страниц, мы останавливаемся. И этот последний случай особенно важно помнить. Когда другие программы на наших компьютерах забывают об этом последнем случае, они могут зависать или перестать отвечать, поскольку они столкнулись с неучтенным случаем, или продолжают повторять одну и ту же работу снова и снова за кулисами, не выполняя никаких действий. прогресс.
* Некоторые из этих строк начинаются с глаголов или действий. Мы начнем вызывать эти *функции* :
* 1 **Возьмите** телефонную книгу
* 2 **Открыто до** середины телефонной книги
* 3 **Посмотрите на** страницу
* 4 Если человек находится на странице
* 5 **Позвонить** человеку
* 6 Иначе, если человек в книге раньше
* 7 **Открыть до** середины левой половины книги
* 8 Вернуться к строке 3
* 9 В противном случае, если человек находится позже в книге
* 10 **Открыта до** середины правой половины книги
* 11 Вернуться к строке 3
* 12 Еще
* 13 **Выйти**
* У нас также есть ответвления, ведущие к разным путям, например развилки дороги, которые мы назовем *условиями* :
* 1 Возьмите телефонную книгу
* 2 Открыто до середины телефонной книги
* 3 Посмотрите на страницу
* 4 **Если** человек находится на странице
* 5 Позвонить человеку
* 6 **Иначе, если** человек в книге раньше
* 7 Открыть до середины левой половины книги
* 8 Вернуться к строке 3
* 9 В **противном случае, если** человек находится позже в книге
* 10 Открыта до середины правой половины книги
* 11 Вернуться к строке 3
* 12 **Еще**
* 13 Выйти
* И вопросы, которые решают, куда мы идем, называются *логическими выражениями* , которые в конечном итоге приводят к значению да или нет, или истина или ложь:
* 1 Возьмите телефонную книгу
* 2 Открыто до середины телефонной книги
* 3 Посмотрите на страницу
* 4 Если **человек находится на странице**
* 5 Позвонить человеку
* 6 Иначе, если **человек в книге раньше**
* 7 Открыть до середины левой половины книги
* 8 Вернуться к строке 3
* 9 В противном случае, если **человек находится позже в книге**
* 10 Открыта до середины правой половины книги
* 11 Вернуться к строке 3
* 12 Еще
* 13 Выйти
* Наконец, у нас есть слова, которые создают циклы, в которых мы можем повторять части нашей программы, называемые *циклами* :
* 1 Возьмите телефонную книгу
* 2 Открыто до середины телефонной книги
* 3 Посмотрите на страницу
* 4 Если человек находится на странице
* 5 Позвонить человеку
* 6 Иначе, если человек в книге раньше
* 7 Открыть до середины левой половины книги
* 8 **Вернуться к строке 3**
* 9 В противном случае, если человек находится позже в книге
* 10 Открыта до середины правой половины книги
* 11 **Вернуться к строке 3**
* 12 Еще
* 13 Выйти

[**Царапать**](https://cs50.harvard.edu/x/2021/notes/0/#scratch)

* Мы можем писать программы, используя только что обнаруженные нами строительные блоки:
  + функции
  + условия
  + Логические выражения
  + петли
* И мы откроем для себя дополнительные функции, в том числе:
  + переменные
  + потоки
  + События
  + …
* Прежде чем мы научимся использовать текстовый язык программирования под названием C, мы воспользуемся графическим языком программирования под названием [Scratch](https://scratch.mit.edu/) , куда мы будем перетаскивать блоки, содержащие инструкции.
* Простая программа на C, которая выводит «привет, мир», будет выглядеть так:
* #include <stdio.h>
* int main(void)
* {
* printf("hello, world\n");
* }
  + Нам нужно выяснить множество символов и синтаксиса или их расположения.
* Среда программирования для Scratch немного более дружелюбна:  
  
  + В правом верхнем углу у нас есть этап, который будет отображаться нашей программой, где мы можем добавлять или изменять фон, персонажей (называемых спрайтами в Scratch) и многое другое.
  + Слева у нас есть кусочки головоломки, которые представляют функции, переменные или другие концепции, которые мы можем перетащить в нашу область инструкций в центре.
  + В правом нижнем углу мы можем добавить больше символов для использования нашей программой.
* Мы можем перетащить несколько блоков, чтобы Скретч сказал «привет, мир»:  
  
  + Блок «при щелчке зеленого флажка» относится к началу нашей программы (поскольку над этапом есть зеленый флажок, который мы можем использовать для его запуска), а под ним мы зафиксировали блок «сказать» и набрали "Привет мир". И мы можем выяснить, что делают эти блоки, исследуя интерфейс и экспериментируя.
* Мы также можем перетащить блок «спроси и подождать» с таким вопросом, как «Как тебя зовут?», И объединить его с блоком «сказать» для ответа:  
  
  + Блок «ответа» - это переменная или значение, в котором хранится то, что вводит пользователь программы, и мы также можем поместить его в блок «сказать», перетащив и отпустив.
* Но мы не стали ждать после того, как сказали «Привет» в первом блоке, поэтому мы можем использовать блок «join», чтобы объединить две фразы, чтобы наша кошка могла сказать «привет, Дэвид»:  
  
  + Когда мы пытаемся вкладывать блоки или размещать их один внутри другого, Scratch помогает нам, расширяя места, где их можно использовать.
* Фактически, сам блок «say» подобен алгоритму, в котором мы предоставили ввод «hello, world», а на выходе Scratch (кот) «произнес» эту фразу:  
  
* Блок «спросить» также принимает входные данные (вопрос, который мы хотим задать) и выдает результат блока «ответа»:  
  
* Затем мы можем использовать блок «answer» вместе с нашим собственным текстом «hello» в качестве двух входных данных для алгоритма соединения ...  
  
* … Выходные данные, которые мы передаем, могут использоваться в качестве входных данных для блока «say»:  
  
* В левом нижнем углу экрана мы видим значок расширений, и одно из них называется Text to Speech. После того, как мы добавим его, мы можем использовать блок «Speak», чтобы услышать, как говорит наша кошка:  
  
* Расширение Text to Speech, благодаря облаку или компьютерным серверам в Интернете, преобразует наш текст в аудио.
* Мы можем попытаться заставить кошку мяукать:  
  
  + Мы можем заставить его произнести «мяу» три раза, но теперь мы повторяем блоки снова и снова.
* Давайте использовать цикл или блок «повторения»:  
  
  + Теперь наша программа достигает тех же результатов, но с меньшим количеством блоков. Мы можем считать, что у него лучший дизайн: если есть что-то, что мы хотели бы изменить, нам нужно было бы изменить это только в одном месте, а не в трех.
* Мы можем заставить кошку указывать на мышь и двигаться к ней:  
  
* Мы пробуем расширение Pen, используя блок «перо вниз» с условием:  
  
  + Здесь мы перемещаем кошку к указателю мыши, и если щелкнуть мышью или нажать вниз, мы опускаем ручку вниз, и она рисует. В противном случае кладем ручку вверх. Мы повторяем это очень быстро, снова и снова, так что мы получаем эффект рисования всякий раз, когда мы удерживаем мышь.
* У Scratch также есть разные костюмы или изображения, которые мы можем использовать для наших персонажей.
* Сделаем программу, которая умеет считать:  
  
  + Здесь counterпеременная, значение которой мы можем устанавливать, использовать и изменять.
* Мы рассмотрим еще несколько программ, например, [bounce](https://scratch.mit.edu/projects/277536611/editor/) , где кошка бесконечно движется вперед и назад по экрану, поворачиваясь всякий раз, когда мы находимся на краю экрана.
  + Мы можем улучшить анимацию, [заставив](https://scratch.mit.edu/projects/277536630/editor/) кошку переодеваться в другой костюм после каждых 10 шагов в [bounce1](https://scratch.mit.edu/projects/277536630/editor/) . Теперь, когда мы щелкаем зеленый флаг, чтобы запустить нашу программу, мы видим, как кошка поочередно двигает ногами.
* Мы даже можем записывать собственные звуки с помощью микрофона нашего компьютера и воспроизводить их в нашей программе.
* Чтобы создавать все более и более сложные программы, мы начинаем с каждой из этих более простых функций и накладываем их друг на друга.
* У нас также может появиться мяуканье Scratch, если мы коснемся его указателем мыши в [pet0](https://scratch.mit.edu/projects/277537223/editor/) .
* В [bark](https://scratch.mit.edu/projects/326130490/editor/) у нас есть не одна, а две программы в одном проекте Scratch. Обе эти программы будут запущены одновременно после щелчка по зеленому флажку. Один из них будет воспроизводить звук морского льва, если для mutedпеременной установлено значение false, а другой установит для mutedпеременной значение либо trueна false, либо falseна true, если нажата клавиша пробела.
* Другое расширение просматривает видео в том виде, в каком оно было снято веб-камерой нашего компьютера, и воспроизводит звук мяуканья, если в видео есть движение выше некоторого порога.
* С несколькими спрайтами или персонажами мы можем иметь разные наборы блоков для каждого из них:  
  
  + Для одной марионетки у нас есть блоки с надписью «Марко!», А затем блок «широковещательное событие». Это «событие» используется для двух наших спрайтов, чтобы общаться друг с другом, как если бы они отправляли сообщение за кулисами. Так что другая наша марионетка может просто дождаться этого «события», чтобы сказать «Поло!»:  
    
* Мы можем использовать расширение Translate, чтобы сказать что-нибудь на других языках:  
  
  + Здесь выходные данные блока «join» используются в качестве входных данных для блока «translate», выходные данные которого передаются в качестве входных данных для блока «say».
* Теперь, когда мы знаем некоторые основы, мы можем подумать о дизайне или качестве наших программ. Например, мы можем захотеть, чтобы кошка мяукнула три раза с помощью блока «повтор»:  
  
* Мы можем использовать **абстракцию** , которая упрощает более сложную концепцию. В этом случае мы можем определить наш собственный блок «мяу» в Scratch и повторно использовать его где-нибудь в нашей программе, как показано в [meow3](https://scratch.mit.edu/projects/421542702/editor/) . Преимущество в том, что нам не нужно знать, как мяуканье реализовано или написано в коде, а просто использовать его в нашей программе, чтобы сделать его более читабельным.
* Мы даже можем определить блок с вводом в [meow4](https://scratch.mit.edu/projects/421543064/editor/) , где у нас есть блок, который заставляет кошку мяукать определенное количество раз. Теперь мы можем повторно использовать этот блок в нашей программе, чтобы мяукать любое количество раз, примерно так же, как мы можем использовать блоки «переводить» или «говорить», не зная **деталей реализации** или того, как блок на самом деле работает.
* Мы рассмотрим еще несколько демонстраций, в том числе [ремикс Gingerbread tales](https://scratch.mit.edu/projects/277536784/) и [Oscartime](https://scratch.mit.edu/projects/277537196/) , оба из которых объединяют петли, условия и движение для создания интерактивной игры.
* На самом деле Oscartime был создан Дэвидом много лет назад, и он начал с добавления одного спрайта, затем по одной функции за раз и так далее, пока они не добавились в более сложную программу.
* Бывший студент Эндрю создал [Raining Men](https://scratch.mit.edu/projects/37412/) . Несмотря на то, что Эндрю в конечном итоге не стал заниматься информатикой как профессией, навыки решения проблем, алгоритмы и идеи, которые мы узнаем в ходе курса, применимы везде.
* До скорого!