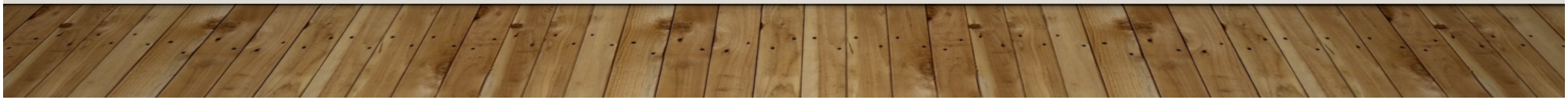
# ЛЕКЦИЯ 13

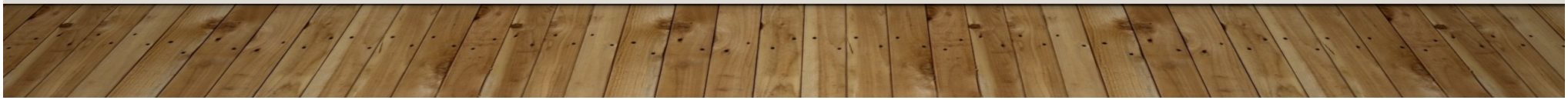
## ЧТЕНИЕ ФАЙЛОВ В RUBY



# ПЛАН ЗАНЯТИЯ

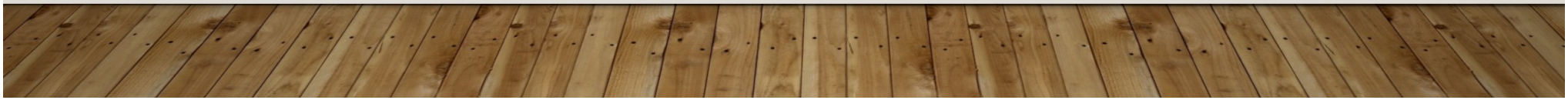
1. Файлы, что это такое и как программы с ними работают
2. Чтение данных из файла в Ruby и не только
3. Зачем нужно закрывать файлы и как это делать

- Мы узнаем о классе File и о том, как работают его методы new, dirname, exist? и методы его экземпляров read, readlines и close.
- Мы научимся читать из файлов данные целиком и построчно, как выводить на экран произвольную строчку файла и как, а главное, для чего необходимо закрывать файлы.



# ЧТО ТАКОЕ ФАЙЛЫ?

- Однажды программисты поняли, что хранить данные в памяти компьютера не надёжно, затратно, не выгодно и вообще, программа должна быть постоянно запущена. Было бы здорово, если бы можно было выключить программу, а потом начать с того же места.
- Как-то так были придуманы файлы.
- Абсолютно вся информация на вашем или любом другом компьютере когда он выключен, хранится в файлах.



# ЧТО ТАКОЕ ФАЙЛЫ?

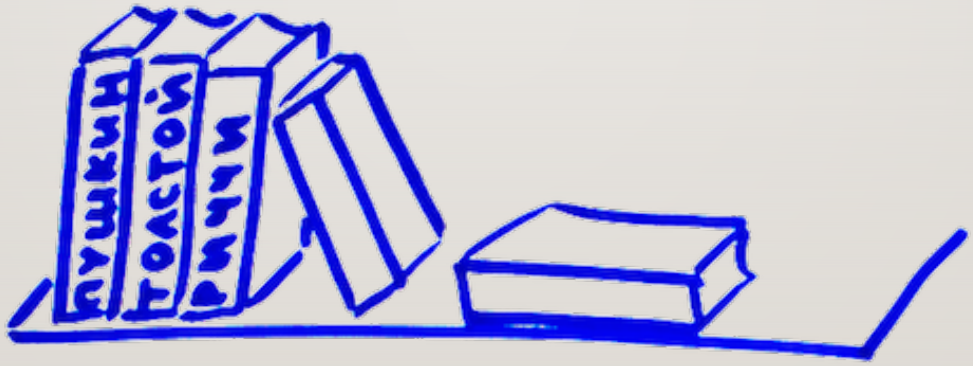
- Файлы есть как у вас на домашнем компьютере — ваши документы и фотографии или системные файлы Windows или Mac OS X, так и на любом сервере, к которому вы на самом деле обращаетесь, когда вводите в браузере адрес любого сайта. То, что вы видите как сайт — это тоже файл (чаще всего несколько файлов), подготовленных определённым образом.
- Именно поэтому так важно и удобно уметь работать с файлами в ваших программах.

# КАК ПРОГРАММЫ РАБОТАЮТ С ФАЙЛАМИ

- Программы — они как люди. Только люди читают книги, когда им нужна какая-то информация, а программы могут читать не только книги, но и картинки, аудиозаписи, видеоролики и много-много другого.

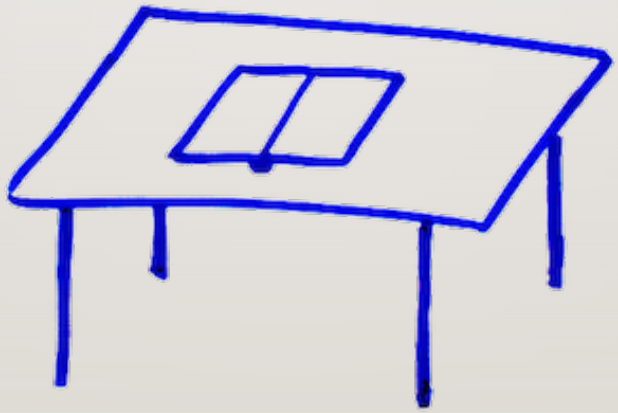
# КАК ПРОГРАММЫ РАБОТАЮТ С ФАЙЛАМИ

- Когда программе сказали, что ей нужно прочесть какой-то файл, она ищет его в файловой системе вашего компьютера и достаёт его, подобно тому, как вы можете заказать книгу в ближайшей библиотеке.



# КАК ПРОГРАММЫ РАБОТАЮТ С ФАЙЛАМИ

- После того, как файловая система предоставила ей этот файл (как заботливый библиотекарь), программа его открывает и может начать читать.

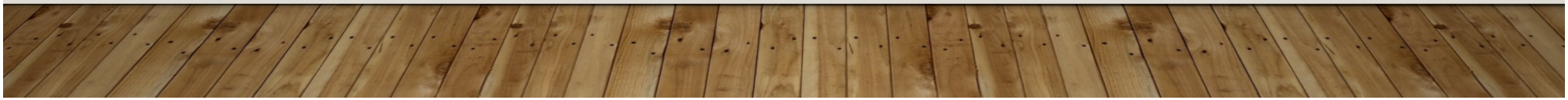


# КАК ПРОГРАММЫ РАБОТАЮТ С ФАЙЛАМИ

- А потом, конечно же, файл нужно закрыть и вернуть на место, чтобы его могли читать другие программы (хотя, в отличие от библиотеки, один и тот же файл могут читать сразу несколько программ, копируя его себе в память или в специальное место на диске).

# ЧТЕНИЕ ФАЙЛОВ В RUBY

- Давайте напишем простенькую программку, решающую вот такую задачу:
- **Вывести на экран произвольный афоризм из файла со списком афоризмов.**
- Список будет храниться в файле quotes.txt, по одному афоризму на одну строчку файла.
- Как обычно, файлы мы храним в нашей папке урока: c:\%username\lesson13.



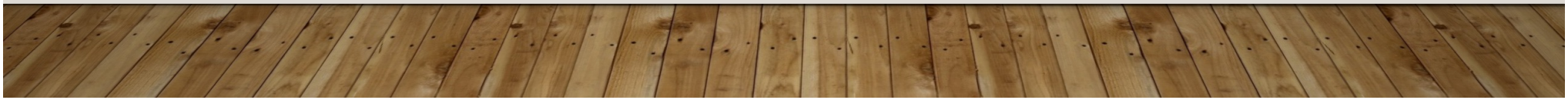
# ЧТЕНИЕ ФАЙЛОВ В RUBY

- Однако, для файлов с данными, не являющимися текстом программ, удобно всегда создавать вложенные подпапки. Наш файл с цитатами quotes.txt мы положим в подпапку data в нашей рабочей директории. Сам файл будет содержать вот это:

```
Учиться, учиться и еще раз учиться... программированию! // В. И. Ленин
Слышь, пацан, либы есть? // Ванек со второго подъезда
Я программирую, значит я существую // Фрэнсис Бэкон
Кто не умеет программировать, тот лох :) // Билл Гейтс
Тяжело в учении – легко в программировании // Народная мудрость
Программировали, перепрограммировали, да не запрограммировали // Скороговорка
```

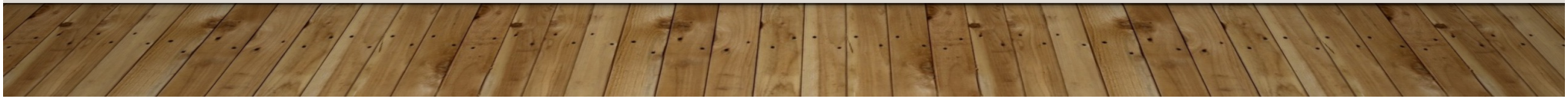
# ЧТЕНИЕ ФАЙЛОВ В RUBY

- Вы можете придумать свои цитаты ;-)
- Теперь создадим нашу программу open_file.rb в рабочей папке урока c:\%username\lesson13.
- В этой программе мы будем открывать файл data/quotes.txt, брать из него произвольную строчку и выводить её на экран.
- Для пользователей Windows: убедитесь, что вы сохранили файл quotes.txt в кодировке UTF-8 (для этого тоже можно использовать редактор Sublime).



# ОТКРЫТИЕ ФАЙЛА

- Для работы с файлами в Ruby есть специальный встроенный класс File, который позволяет в программе создавать объекты для работы с файлами.
- Любой экземпляр класса начинается с конструктора. Также и здесь, чтобы создать новый файловый объект, нам надо у класса File вызвать метод new, который вернёт нам экземпляр класса File.
- Методу new нужно передать несколько параметров мы уже умеем это делать: первый параметр — путь (относительно текущей директории, из которой вы запускаете программу) к файлу, который нужно открыть, второй параметр — специальный ключ.



# ОТКРЫТИЕ ФАЙЛА

- Этот ключ говорит классу File, как именно мы хотим открыть файл, и в какой кодировке мы этот файл хотим прочитать.
- В нашем случае мы хотим открыть наш файл с афоризмами ./data/quotes.txt для чтения в кодировке UTF-8, поэтому пишем так:

```
file = File.new("./data/quotes.txt","r:UTF-8")
```

# ОТКРЫТИЕ ФАЙЛА

- Обратите внимание на ключ "r:UTF-8", первая буква обозначает тип открытия файла:
  - r — только для чтения: мы будем только читать файл, писать в него мы так не сможем
  - w — только для записи: мы не хотим знать, что в файле, мы просто перепишем его содержимое
  - a — только для записи, но дописывать будем в конец файла, сам файл не трогаем
- Есть и другие ключи, но в этом блоке не будем ими морочить вам голову, в этом уроке нам понадобится только первый, так что все остальные — для любознательных. После указания способа открытия файла через двоеточие идёт кодировка, в нашем случае UTF-8.

# ОТКРЫТИЕ ФАЙЛА

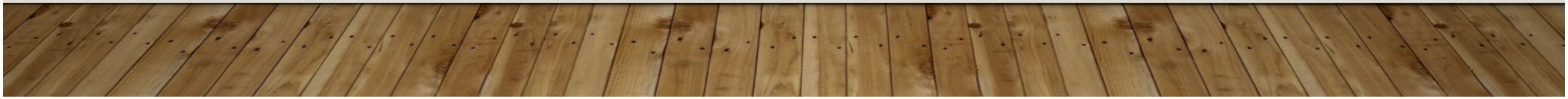
- Обратите внимание на ключ "r:UTF-8", первая буква обозначает тип открытия файла:
  - r — только для чтения: мы будем только читать файл, писать в него мы так не сможем
  - w — только для записи: мы не хотим знать, что в файле, мы просто перепишем его содержимое
  - a — только для записи, но дописывать будем в конец файла, сам файл не трогаем
- Есть и другие ключи, но в этом блоке не будем ими морочить вам голову, в этом уроке нам понадобится только первый, так что все остальные — для любознательных. После указания способа открытия файла через двоеточие идёт кодировка, в нашем случае UTF-8.

# ЧТЕНИЕ ИНФОРМАЦИИ ИЗ ФАЙЛА

- Мы открыли файл, но пока ещё ничего с ним не сделали, мы просто получили некую переменную file с экземпляром класса File, которая знает, к какому файлу она относится и как ему с этим файлом нужно обращаться.
- Мы можем получить всё содержимое файла в одной большой строке (со всеми словами, пробелами и переносами) с помощью метода read экземпляра класса File, то есть нашего объекта file.

```
content = file.read
```

- Теперь по идее в переменной content у нас будут все цитаты одна за другой.



# ЧТЕНИЕ ФАЙЛА ПОСТРОЧНО

- По условию задачи нам нужно прочитать только одну цитату, а не весь файл целиком. Поэтому нам нужно немного переписать нашу программу. Мы напишем новый файл read_lines.rb

```
file_path = "/data/aphorizmus.txt"

f = File.new(file_path, "r:UTF-8")
lines = f.readlines
puts lines.sample
```

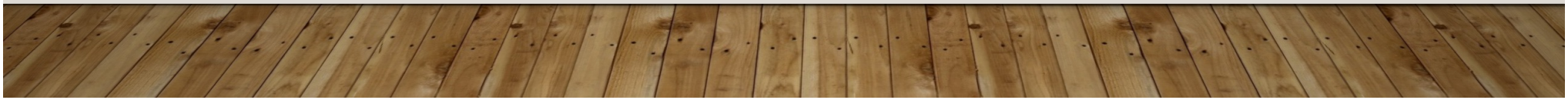
- Обратите внимание, что мы сохранили путь к файлу в переменную file_path, чтобы каждый раз не писать его вручную.

# ЧТЕНИЕ ФАЙЛА ПОСТРОЧНО

- Вместо file.read вызываем метод readlines, который в отличие от предыдущего возвращает не одну большую строку, а массив строк, разбитых по символам переноса.
- Этот массив мы сохранили в переменную lines, поэтому чтобы вывести одну случайную строчку файла мы можем просто написать:

```
puts lines.sample
```

- Метод sample у массива возвращает один случайный элемент из этого массива — каждый раз разный.



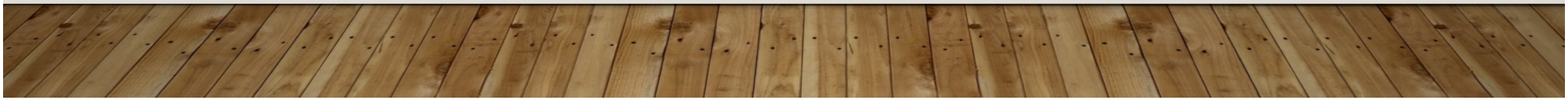
# ЗАПУСК ПРОГРАММЫ

- Программа написана и осталось её запустить, для этого как обычно переходим в нашу рабочую папку в консоли и запускаем программу с помощью команды ruby

```
ruby read_lines.rb
```

# ЧТО ДЕЛАТЬ, КОГДА ФАЙЛ НЕ НАЙДЕН?

- Первое, с чем сталкивается любой программист, когда пишет программу, общающуюся с файлами — ситуация, когда файл не открылся. Не важно по какой-то причине.
- Может, его стёрли, может, он опечатался и неправильно указал имя файла, может быть, диск, на котором хранился файл, недоступен. Мало ли чего в наше неспокойное время может произойти.
- Запомните основное правило при работе с файлами: перед открытием файла всегда убедитесь, что он есть.



# ЧТО ДЕЛАТЬ, КОГДА ФАЙЛ НЕ НАЙДЕН?

- Проверить наличие файла можно с помощью метода exist? у класса File:

```
if File.exist?(file_path)
  # тут можно работать с файлом, не боясь, что он не откроется
else
  puts "Файл не найден"
end
```

- Как вы видите, метод очень похож на new, только никакой ключ ему не нужен, потому что методу exist? не важно, как с этим файлом собрались работать, ему только нужно проверить, есть ли файл и если файл есть, то вернуть true, а если нет — вернуть false.

# ЧТО ДЕЛАТЬ, КОГДА ФАЙЛ НЕ НАЙДЕН?

- Именно поэтому очень удобно использовать этот метод в качестве условия в конструкции if-else, если файл есть, мы его откроем и прочитаем, если его нет, мы напишем об этом пользователю и либо пойдём дальше, либо прекратим выполнение программы.
- Теперь можно переименовать файл quotes.txt в, например, quotes_.txt и убедиться, что при запуске программы, мы увидим строчку:

```
Файл не найден
```

# КАК ПРАВИЛЬНО УКАЗЫВАТЬ ПУТИ К ФАЙЛАМ

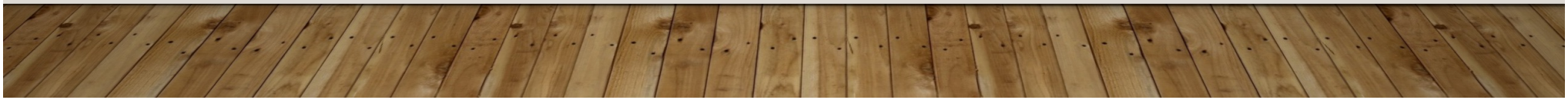
- При открытии файла мы считали, что он лежит в папке data, которая находится в той же папке, в которой мы запускаем программу. Если мы в консоли поднимемся на уровень выше:

```
cd ../
```

- и снова запустим нашу программу, дописав к её пути спереди название папки, в которой она лежит (мы не говорили, но так можно делать):

```
ruby lesson13/qoute.rb
```

- то встретимся как раз с ситуацией, когда файл не найден.



# КАК ПРАВИЛЬНО УКАЗЫВАТЬ ПУТИ К ФАЙЛАМ

- Наша проверка обломалась, ведь файла уже нет рядом с нами, он лежит в другой папке, о чём программа не знает, она ищет его в c:\%username\data, то есть рассматривает путь data относительно текущей папки консоли.
- Знак точки (.) в пути к файлу означает: «текущая папка, в которой мы находились в тот момент, когда запустили программу», то есть текущая папка консоли в нашем случае.
- Но нам нужно, чтобы файл в подпапке data искался всегда относительно папки, в которой лежит программа, а не из которой она запущена.

# КАК ПРАВИЛЬНО УКАЗЫВАТЬ ПУТИ К ФАЙЛАМ

- Это легко исправить, ведь для этого в Ruby есть классная штука: специальный объект __FILE__ — он содержит путь к файлу программы относительно той папки, из которой программа запущена. То есть откуда бы мы не запустили, с помощью объекта __FILE__ мы всегда можем восстановить правильный путь к файлам нашей программы.
- Для того, чтобы из этой переменной получить путь к папке текущей программы, нам нужно снова обратиться к нашему классу File:

```
current_path = File.dirname(__FILE__)
```

# КАК ПРАВИЛЬНО УКАЗЫВАТЬ ПУТИ К ФАЙЛАМ

- Теперь в переменной current_path всегда (откуда бы мы ни запустили нашу программу) будет лежать путь к папке с этой программой.
- А от него и до data/quotes.txt рукой подать, нужно просто склеить две строчки плюсиком, как мы это умеем делать:

```
file_path = current_path + "/data/quotes.txt"
```

# КАК ПРАВИЛЬНО УКАЗЫВАТЬ ПУТИ К ФАЙЛАМ

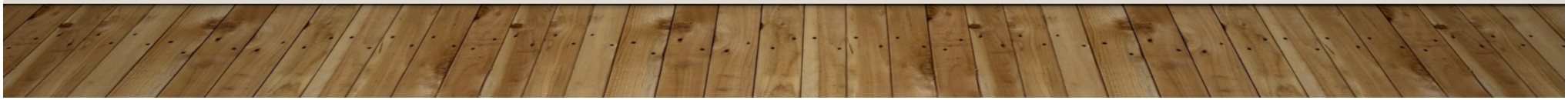
- Теперь откуда бы мы ни запустили нашу программу, она будет искать файл data/quotes.txt рядом с собой, а не в том месте, откуда её вызвали.
- Это очень удобно. Можно в консоли перейти на папку выше и запустить программу вот так:

```
ruby lesson13/read_lines.rb
```

- Тяжело в учении - легко в программировании!

# ЗАКРЫТИЕ ФАЙЛОВ

- Если вкратце, то файлы нужно закрывать.
- Во-первых, незакрытые файлы могут приводить к ошибкам в ваших и чужих программах, которые работают с этим же файлом.
- Во-вторых, каждый открытый файл занимает у памяти какой-то её объём и чем больше таких открытых файлов, тем медленнее работает ваша программа. Хорошие программисты помнят об этом и стараются делать свои программы быстрее.
- Для нашей маленькой программы это не так важно — Ruby сам закроет все файлы после выполнения программы.



# ЗАКРЫТИЕ ФАЙЛОВ

- **Но усвоить эту привычку надо уже сейчас, когда ваши программы станут большими и навороченными, вы еще скажете спасибо за эту привычку.**
- Это как правило хорошего тона: подобно тому, как прилежный читатель всегда возвращает книгу туда, откуда он её взял — полка в собственном доме или библиотека, также и прилежный программист всегда должен помнить о том, что чем меньше на Земле незакрытых вовремя файлов, тем больше на Земле добра!

# ЗАКРЫТИЕ ФАЙЛОВ

- Делайте добро, Дамы и Господа, всегда закрывайте ваши файлы.
- Закрывать файлы можно сразу после того, как вы сделали с файлом всё, для чего он был нужен. Прям вот сразу же.

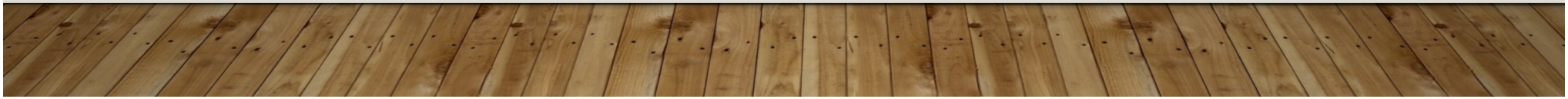
# ЗАКРЫТИЕ ФАЙЛОВ

- В нашей программе мы закроем файл ещё до того, как выведем строчку на экран.

```
current_path = File.dirname(__FILE__)
file_path = current_path + "/data/aphorizmus.txt"

if File.exist?(file_path)
  f = File.new(file_path, "r:UTF-8")
  lines = f.readlines
  f.close
  puts lines.sample
else
  puts "Файл не найден"
end
```

# ПОДВЕДЕНИЕ ИТОГОВ

- Итак, мы научились работать с файлами, узнали о классе File, как работают его методы new, dirname, exist? и методы его экземпляров read, readlines и close.
- Узнали, как читать из файлов данные целиком и построчно, как выводить на экран произвольную строчку файла и как, а главное, для чего необходимо закрывать файлы.
- А в следующем уроке нас ждёт третья версия нашей замечательной игры «Виселица», мы будем использовать полученные данные и будем открывать файлы с псевдографикой.



# СЧИТАЕМ СТРОКИ В ФАЙЛАХ

- Написать программу, которая считает сколько в указанном файле строк всего, сколько пустых строк, а также выводит на экран последние 5 строк этого файла.
- Пользователь вводит название файла в качестве параметра консоли при запуске. Если же файл не найден, то сообщает об этом пользователю.
- Проверьте работу программы на различных вами же придуманных файлах.

# СЧИТАЕМ СТРОКИ В ФАЙЛАХ

- **Например:**

```
Открыли файл: data/file.txt
Всего строк: 10
Пустых строк: 4
Последние 5 строк файла:

Это пятая строка с конца.
Это четвёртая строка с конца.
Это третья строка с конца.
Это вторая строка с конца.
Это последняя строка.
```

# СЧИТАЕМ СТРОКИ В ФАЙЛАХ. ПОДСКАЗКА

- Чтобы открыть файл, используя аргумент из командной строки, нужно записать этот аргумент в переменную и затем проверить, что эта переменная не пустая и что такой файл существует.
- Помните при этом что файл, который вы передаете как параметр, должен находиться на диске в той же папке, из которой вы запускаете программу!
- Когда открыли файл — можно сохранить все строчки в массив, а после этого работать с этим массивом строк: количество строк — метод size, посчитать пустые строки можно в цикле и в цикле же вывести последние 5 элементов этого массива с помощью команды puts.
- Если захотите в одном цикле и собрать 5 последних строк и посчитать число пустых, изучите метод массива: each_with_index.

# ГЕНЕРАТОР РОЖИЦ

- Напишите программу–генератор рожиц с использованием псевдографики: программа должна генерить лицо из произвольно выбранных фрагментов: лба, глаз, носа и рта.
- Эти фрагменты придумайте сами с помощью символов в виде строчек.
- Например, из такого лба / --- \, таких глаз | O o |, носа \ v / и рта \ - /, получится рожица:

```
 / --- \
| 0   o |
\   v   /
 \  -  /
```

- Варианты для каждой части лица программа должна брать из соответствующего файла, например, глаза из файла data/eyes.txt.

# ГЕНЕРАТОР РОЖИЦ. ПОДСКАЗКА

- Нарисуйте в текстовом файле рожицу, состоящую из 4-х строк: лоб с волосами, глаза, нос и рот.
- Потом сохраните каждую строчку в отдельный файл (соответственно foreheads.txt, eyes.txt, noses.txt, mouths.txt) и, скопировав, сделайте ещё несколько вариантов для каждого фрагмента лица.
- Потом в основной программе откройте все эти файлы, выберите из каждого произвольную строку и выведите это всё в нужном порядке на экран.

# ВИКТОРИНА

- Создайте два файла: один с вопросами (по одному вопросу на одну строчку файла), другой с ответами на эти вопросы (также на каждой строчке один ответ).
- Например:
- **questions.txt**

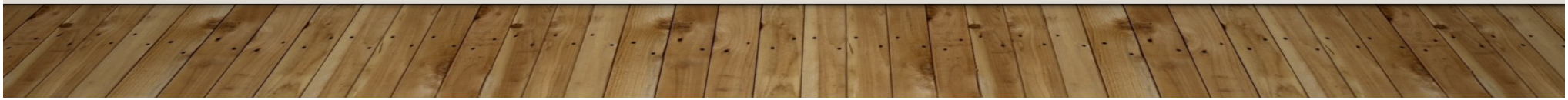
```
Сколько байт в килобайте?
В каком году Гагарин полетел в космос?
Сколько дней в высокосном году?
```

- **answers.txt**

```
1024
1961
366
```

# ВИКТОРИНА

- Напишите программу, которая считывает оба файла и задает пользователю в консоли последовательно все эти вопросы.
- Задав вопрос, программа ждет ответа пользователя в консоли. А затем сравнивает с правильным ответом и сообщает, правильно ответил пользователь, или нет.
- Если ответ неверный, программа сообщает какой ответ правильный. В конце сообщает результат — сколько было дано правильных ответов.



# ВИКТОРИНА

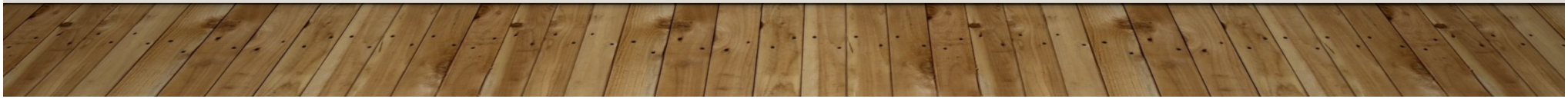
- **Пример:**

```
Мини-викторина. Ответьте на вопросы.
Сколько байт в килобайте?
> 1024
Верный ответ!

В каком году Гагарин полетел в космос?
> 1962
Неправильно. Правильный ответ: 1961

Сколько дней в высокосном году?
> 366
Верный ответ!

У вас 2 правильных ответов из 3
```

- Только придумайте свои вопросы и ответы :)

# ВИКТОРИНА. ПОДСКАЗКА

- Вам нужно открыть два файла с помощью File.read и записать их содержимое в два разных массива строк с помощью метода readlines.
- Затем в цикле while пройтись по массиву вопросов и задавать эти вопросы пользователю. Проверять правильность ответа, считать количество правильных и в конце выводить результат.
- Не забывайте при проверке равенства строк преобразовывать строку пользователя в кодировку UTF-8, и отрезать символ конца строки как от того, что ввел пользователь, так и от строки прочитанной из файла (метод chomp).



# СПРАВОЧНАЯ ИНФОРМАЦИЯ

- Когда ты легко плаваешь в потоке современных технологий :)
- Работа с файлами в Ruby (1)
- Работа с файлами в Ruby (2)
- Метод each_with_index

# СПАСИБО ЗА ВНИМАНИЕ!

Чтение файлов в Ruby