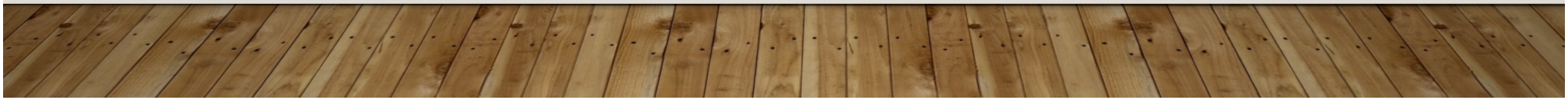
ЛЕКЦИЯ 19

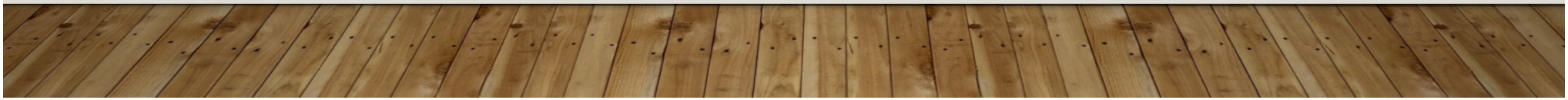
ОШИБКИ И ИСКЛЮЧЕНИЯ



ПЛАН ЗАНЯТИЯ

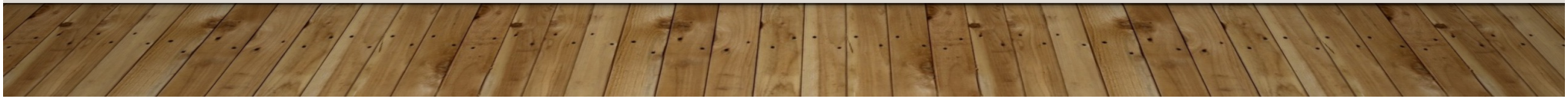
1. Какие бывают ошибки и как с ними бороться
2. Что такое исключения и почему они важны
3. Как работать с исключениями в Ruby

- Мы узнаем, какие бывают в программах ошибки, как правильно реагировать на разные виды ошибок, что такое и как работают исключения в Ruby. И как различать разные виды ошибок, как их предупреждать.
- Мы научимся не бояться ошибок, разберём глобальные причины ошибок: непонимание постановки задачи, баги и исключения. А также узнаем как работает конструкция begin-rescue, зачем там нужен ensure и как ловить только конкретные типы ошибок в Ruby.



ОШИБКИ В ПРОГРАММАХ

- Программные ошибки — наверное, самая обсуждаемая тема среди программистов, так как процесс написания программ связан с решением довольно творческих задач по относительно строгим правилам.
- Грубо говоря, программу вы придумываете сами, но пишете её по строгим законам языка и технологии, поэтому если вы нарушаете какие-то правила, ваша программа в определённых ситуациях будет ломаться и не давать нужного результата.
- Но как говорится, волков бояться — в лес не ходить. Вот и ошибок в программировании бояться не стоит, нужно лишь для себя понимать как реагировать и предупреждать разные виды ошибок.



КАКИЕ БЫВАЮТ ОШИБКИ?

- Практически все ошибки можно отнести к одной из трёх групп

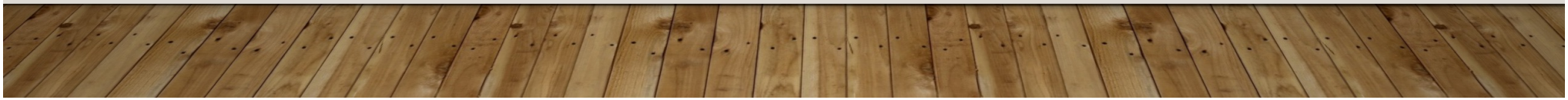
I. **Ошибки программиста, связанные с неправильно написанным кодом**

- Это опечатки, забытые запятые или скобки, из-за которых программа может даже и не компилироваться. Эти ошибки как правило становятся видны на первых же этапах написания и запуска программ.
- Про эти ошибки не нужно особо никак париться, они есть у всех (у новичков больше, у опытных – меньше), они исправляются «по ходу пьесы».

КАКИЕ БЫВАЮТ ОШИБКИ?

2. **Ошибки программиста, связанные с неверным пониманием задачи («баги»)**

- Баги — это ошибки, которые не заметно во время написания и первого запуска программы. Это ошибки в логике программы.
- Например, по четным дням недели светофор должен после 22 часов переключаться в режим желтого света. А оказалось, что он это делает во все дни, кроме последнего дня каждого месяца.
- Во время постановки задачи забыли явно сказать «в любой день года», с другой стороны программист должен был догадаться или уточнить у заказчика. Вроде бы никто не виноват, но программа работает не так, как надо.



КАКИЕ БЫВАЮТ ОШИБКИ?

2. **Ошибки программиста, связанные с неверным пониманием задачи («баги»)**

- И таких ошибок полно в любой программе, даже самой простой. Если вы были внимательные — заметили, что в программах нашего курса мы частенько исправление таких ошибок оставляем как домашнее задание для пытливого студента.
- Исправление багов — неотъемлемая часть работы программиста, не важно занимаетесь ли вы этим на профессиональном или любительском уровне.
- Не надо пытаться писать программу без багов, это невозможно. Нужно сфокусировать свое внимание на том, чтобы как можно подробнее и нагляднее описать для себя задачу и тщательно ее реализовать. Все остальное придет только с опытом.

КАКИЕ БЫВАЮТ ОШИБКИ?

3. **Внешние ошибки, исключения**

- Наконец, если вы всё написали идеально, опечаток, ошибок и багов в вашей программе нет — то обстоятельства при запуске программы могут сложиться так, что ей не суждено выполнится.
- Например, вы попытались открыть какой-то файл, но в этот момент полетел жёсткий диск, или вы хотели создать новый файл, а на диске кончилось место. Светофор должен загореться желтым, но вдруг выключили электричество.
- Такой нештатный режим работы программы называется **исключение** (англ.*exception*).

КАК ОБРАБАТЫВАТЬ ИСКЛЮЧЕНИЯ?

- В прошлом уроке мы уже попробовали, что будет в нашей программе для отправки почты, если, например, указать неправильный пароль к почте: мы получим ошибку авторизации.
- Это и есть исключение. Давайте посмотрим, что мы можем с ним сделать.
- Работать будем на основе нашей программы для отправки почты из урока про библиотеки RubyGems, скопируйте файл send_mail.rb в новую папку c:\%username\lesson19.

КОНСТРУКЦИЯ BEGIN-RESCUE

- Для того, чтобы программа не вылетала при появлении какого-то исключения, нам необходимо участок программы, где может возникнуть ошибка написать внутри специальной конструкции begin-rescue:

```
begin
  # код, который может вызвать ошибку
rescue
  # этот код выполнится, если ошибка произойдет
end
```

КОНСТРУКЦИЯ BEGIN-RESCUE

- Работает эта штука так. Программа доходит до begin (по англ. *начать*) и продолжает выполняться дальше как ни в чём не бывало. Если между строчками begin и rescue ничего страшного не произошло, то дойдя до слова rescue программа перескакивает на end и движется дальше, как будто никакого begin-rescue она и не встретила вовсе.
- Если же между строками begin и rescue программа поймала исключение: какой-то метод сломался и выдал ошибку наподобие тех, что мы видели в конце 15-го урока, то программа тут же переходит на слово rescue (по-англ. *спасти*) и начинает выполнять инструкции, которые написаны между rescue и end.

КОНСТРУКЦИЯ BEGIN-RESCUE

- Там разработчики пишут код, который должен сообщить пользователю о том, что произошла ошибка и, если надо, заканчивают программу, а если можно продолжать работу программы — продолжают.

```
...
rescue
  puts "Не удалось отправить письмо"
end
```

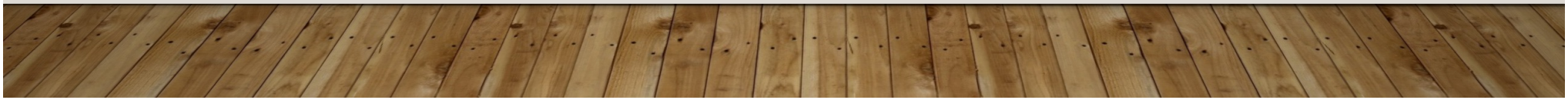
КЛЮЧЕВОЕ СЛОВО ENSURE

- Между rescue и end можно добавить ещё одно ключевое слово ensure (по-англ. *убедиться*):

```
begin
  # код, который будет выполняться, пока не возникнет ошибка
rescue
  # код, который будет выполнен, если возникнет ошибка
ensure
  # код, который будет выполнен всегда в конце
end
```

КЛЮЧЕВОЕ СЛОВО ENSURE

- Тогда если программа не встретила на своём пути от begin до rescue никаких ошибок, то она переходит не на end, а на ensure и перед тем, как закончить возиться с нашей конструкцией, выполняет все инструкции из блока ensure-end.
- Если же ошибка возникла, то программа после выполнения всех инструкций после rescue выполняет также инструкции после ensure.



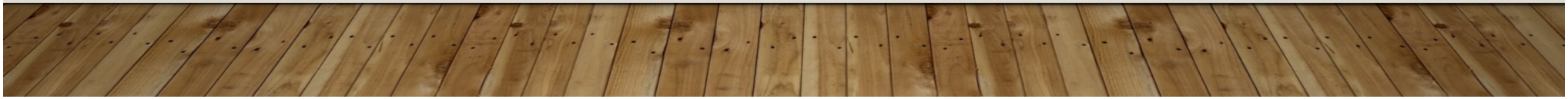
КЛЮЧЕВОЕ СЛОВО ENSURE

- То есть, инструкции в блоке ensure-end будут выполнены в любом случае.

```
...
ensure
  puts "Попытка отправки письма закончена"
end
```

ПОЛУЧЕНИЕ ДАННЫХ ОБ ИСКЛЮЧЕНИИ

- Описанный способ ловли ошибок с помощью begin-rescue — самый примитивный и простой. С его помощью мы не можем сообщить пользователю что же конкретно пошло не так.
- Ошибка в пароле, нет сети или неправильный email адресата — один чёрт. Всё равно, всё что напишет программа пользователю: «Не удалось отправить письмо».
- Это не хорошо, т.к. всех нас бесит, когда кто-то не делает то, что мы просим и даже ничего не объясняет. Хорошо бы сообщить пользователю, что именно не сработало и как ему можно исправить эту ситуацию, чтобы всё-таки выполнить задуманное.



ПОЛУЧЕНИЕ ДАННЫХ ОБ ИСКЛЮЧЕНИИ

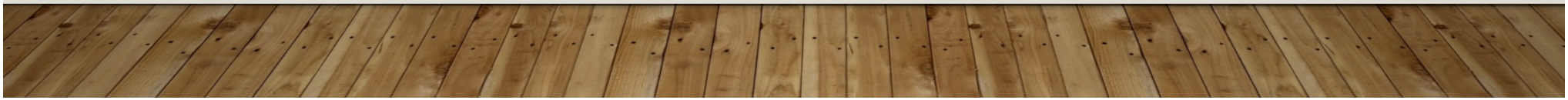
- Давайте уберём (или лучше закомментируем) все строчки, связанные с обработкой ошибок. Чтобы ошибки снова начали вылезать наружу: нам нужно их изучить.
- Комментировать строчку в VS Code можно комбинацией Ctrl+/:

```
# begin
... # то, что между begin и rescue, не надо комментировать
# rescue SocketError
#   puts "Не удалось отправить письмо"
# ensure
#   puts "Попытка отправки письма закончена"
# end
```

ПОЛУЧЕНИЕ ДАННЫХ ОБ ИСКЛЮЧЕНИИ

- После этого снова накосячим, введя неправильный пароль от почты, с которой мы хотим отправить письмо.
- После этого программа выдаст ошибку, в тексте которой нас интересует то, что идёт в скобочках после первого сообщения о том, что что-то пошло не так (найдите глазами это место):

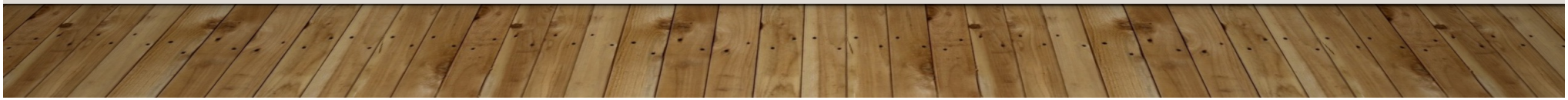
```
... (Net::SMTPAuthenticationError)
```



ПОЛУЧЕНИЕ ДАННЫХ ОБ ИСКЛЮЧЕНИИ

```
... (Net::SMTPAuthenticationError)
```

- Это так называемый тип (класс) ошибки, которая произошла.
- Зная этот класс мы сможем поймать именно эту конкретную ошибку и уже в этом случае будем уверены, что пользователь (то есть вы, ведь в программе используется ваш почтовый адрес для отправки) ошибся при вводе пароля.



ПОЛУЧЕНИЕ ДАННЫХ ОБ ИСКЛЮЧЕНИИ

- Уберите значки комментариев #, которые мы поставили чуть раньше и допишите к rescue условие

```
begin
  ...
rescue Net::SMTPAuthenticationError => error
  puts "Вы неправильно указали пароль: " + error.message
end
```

- Обратите внимание на то, что название класса полностью совпадает с тем, что мы увидели при запуске программы, когда неправильно указали пароль.

ПОЛУЧЕНИЕ ДАННЫХ ОБ ИСКЛЮЧЕНИИ

```
begin
  ...
rescue Net::SMTPAuthenticationError => error
  puts "Вы неправильно указали пароль: " + error.message
end
```

- А также на то, что после этого имени класса мы написали знак равно и знак больше (которые вместе образовали стрелочку =>), и еще слово error.
- error — это временная переменная, куда записывается ошибка, которая произошла. Да, ошибка это тоже объект. Его можно использовать после rescue, чтобы вывести сообщение об ошибке.

ПОЛУЧЕНИЕ ДАННЫХ ОБ ИСКЛЮЧЕНИИ

- Теперь, если мы снова ошибёмся с паролем, программа напишет нам:

```
Вы неправильно указали пароль: 535 Incorrect
authentication data: authentication failed for <my_mail@mail.ru>
```

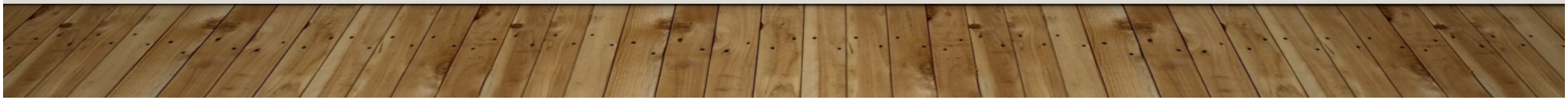
ЛОВИМ ИСКЛЮЧЕНИЯ: SocketError И Net::SMTPSyntaxError

- Чтобы поймать ещё два исключения (когда нет сети и когда пользователь сделал ошибку в почтовом адресе), мы добавим ещё два блока rescue сразу под первым (или над, главное, чтобы они шли друг за другом):

```
begin
  ...
rescue SocketError
  puts "Не могу соединиться с сервером. "
rescue Net::SMTPSyntaxError => error
  puts "Вы некорректно задали параметры письма: " + error.message
rescue Net::SMTPAuthenticationError => error
  puts "Неправильный пароль, попробуйте еще: " + error.message
ensure
  puts "Мы постарались отправить письмо."
end
```

ЛОВИМ ИСКЛЮЧЕНИЯ: SocketError И Net::SMTPSyntaxError

- Это напоминает конструкцию case-when: если в блоке begin-rescue (от begin до первого rescue) возникает ошибка, Ruby смотрит, что за класс у этой ошибки и в зависимости от класса заходит в один из нескольких вариантов. Если класс ошибки SocketError — выполнятся инструкции между rescue SocketError и end (или другим rescue, смотря чем заканчивается блок).
- Если доступа в интернет нет, то во время отправки почты будет отправлена ошибка SocketError, а если почта, например, не содержит символа @ (собака), то Pony создаст исключение Net::SMTPSyntaxError. В каждом из этих случаев мы напишем пользователю о том, что случилось.



ЛОВИМ ИСКЛЮЧЕНИЯ: SocketError И Net::SMTPSyntaxError

- Осталось только перенести наш вывод строчки с фразой, что письмо успешно отправлено в конец блока begin, так как нам надо сделать так, что если письмо не отправилось, то эта строчка не выводится. Ошибка возникнет в длинном методе Pony.mail, так что до команды puts дело просто не дойдёт. Что нам и нужно.

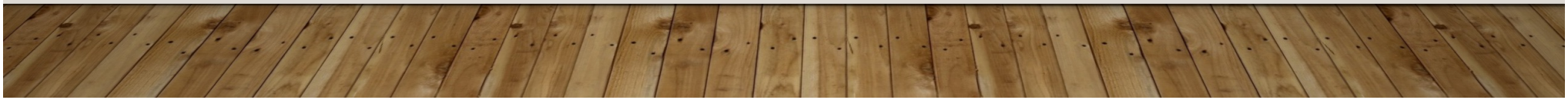
```
begin
  ...
  puts "Письмо отправлено!"
rescue
  ...
rescue
  ...
end
```

ПРАВИЛА ЛОВЛИ ИСКЛЮЧЕНИЙ

- Напоследок сформулируем небольшие правила написания программы, в которой могут быть исключения:

I. **Не доверяйте пользователю, будьте всегда готовы, что самый прилежный пользователь введёт данные неправильно.**

- И ни в коем случае не стоит его за это винить: новые пользователи ваших будущих программ часто просто не знают, что нужно вводить, другие пользователи могут просто опечататься.
- Надо прощать такие ошибки людям, ведь если вы посмотрите на мир вокруг вас, огромное количество программ и устройств именно так и поступает. А те, которые не дают вам право на ошибку, как правило, крайне бесят.

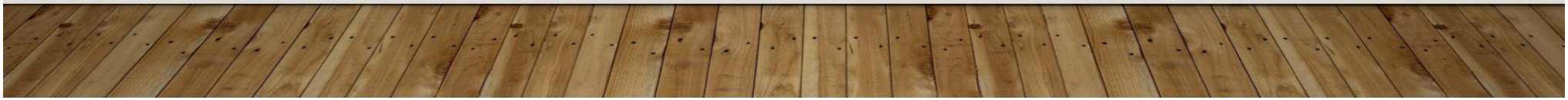


ПРАВИЛА ЛОВЛИ ИСКЛЮЧЕНИЙ

- Ловите только конкретные ошибки: не стоит ловить вообще все исключения.

2. **Самые критические должны происходить и действительно ломать ход вашей программы, не стоит страховаться от всего сразу (да это и невозможно, количество всевозможных ошибок исчисляется сотнями).**

- Представьте самые частые конкретные исключения, которые могут произойти в определенных местах вашей программы и защитите пользователя только от них.



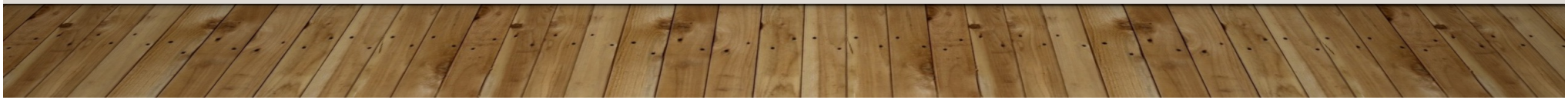
ПРАВИЛА ЛОВЛИ ИСКЛЮЧЕНИЙ

3. **Некоторые ошибки — критичны.**

- При написании программы вы должны чётко для себя решить, какие ошибки заканчивают её работу, а какие игнорируются или просто как-то влияют на ход выполнения.
- Если пользователь опечатался, можно попросить его ввести данные ещё раз, а вот если вы хотите в вашей программе отправить письмо, а компьютер не подключён к сети — не стоит просить пользователя повторно ввести пароль или запускать бесконечный цикл ожидания связи. Можно просто спокойно сообщить об этом и выйти. Если конечно вы не почтовый сервер пишете :)

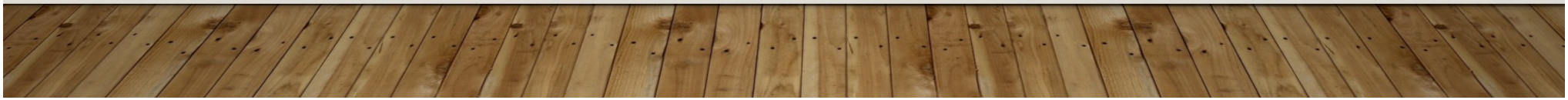
ПРАВИЛА ЛОВЛИ ИСКЛЮЧЕНИЙ

- Ещё пара примеров.
- Вспомните программу, которая выводила один из афоризмов. Если нам не удалось открыть файл, выводить нечего, поэтому выполнение программы не имеет смысл продолжать.
- В программе Виселица мы открывали файл, чтобы прочитать картинку с изображением виселицы: это нужно для красоты. А если вдруг картинка не открылась, мы просто пишем об этом пользователю, но на основной ход программы это не влияет.



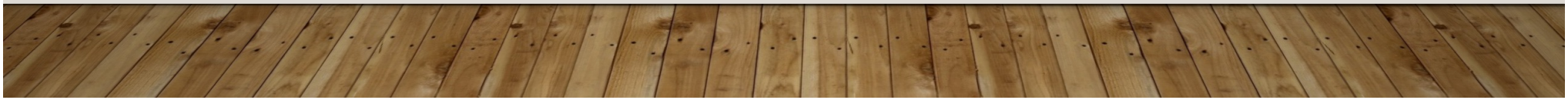
ПРАВИЛА ЛОВЛИ ИСКЛЮЧЕНИЙ

- Ситуации бывают разные и понимание, как обрабатывать исключения в том или ином случае, к вам придёт с опытом. Главное, не пугайтесь ошибок: это всего лишь иной ход развития вашей программы, который обычно даже опытные программисты не учитывают.
- Поэтому достаточно помнить об этом и внимательно относиться к тем, методам, которые связаны с внешним миром (файлами, сетью и т. п.) — тогда вы уже обойдете в этом вопросе многих более опытных программистов!
- В этом уроке мы научились не бояться ошибок, разобрали глобальные причины ошибок: непонимание постановки задачи, баги и исключения. А также узнали как работает конструкция begin-rescue, зачем там нужен ensure и как ловить только конкретные типы ошибок.



ПРАВИЛА ЛОВЛИ ИСКЛЮЧЕНИЙ

- Ситуации бывают разные и понимание, как обрабатывать исключения в том или ином случае, к вам придёт с опытом. Главное, не пугайтесь ошибок: это всего лишь иной ход развития вашей программы, который обычно даже опытные программисты не учитывают.
- Поэтому достаточно помнить об этом и внимательно относиться к тем, методам, которые связаны с внешним миром (файлами, сетью и т. п.) — тогда вы уже обойдете в этом вопросе многих более опытных программистов!
- В этом уроке мы научились не бояться ошибок, разобрали глобальные причины ошибок: непонимание постановки задачи, баги и исключения. А также узнали как работает конструкция begin-rescue, зачем там нужен ensure и как ловить только конкретные типы ошибок.



ЦЕЛОЧИСЛЕННЫЙ КАЛЬКУЛЯТОР

- Напишите простенький калькулятор, который умеет делать операции с двумя целыми (и только целыми) числами: сложение, вычитание, умножение, деление.
- Числа и операцию он по очереди спрашивает у пользователя.

```
Первое число:
50
Второе число:
10
Выберите операцию (+ - * /):
*
Результат:
500
```

ЦЕЛОЧИСЛЕННЫЙ КАЛЬКУЛЯТОР

- Добавьте в этот калькулятор обработку ошибок при попытке деления на ноль:

```
Первое число:
50
Второе число:
0
Выберите операцию (+ - * /):
/
Результат:
На ноль делить нельзя
```

ЦЕЛОЧИСЛЕННЫЙ КАЛЬКУЛЯТОР. ПОДСКАЗКА

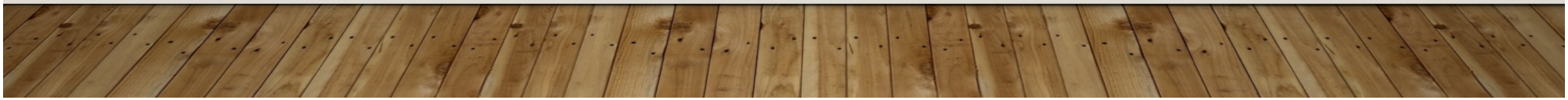
- Спросите у пользователя два числа и сохраните их в разные переменные. Для перевода введённых пользователем символов в целые числа используйте метод строки to_i.

```
number = gets.chomp.to_i
```

- Спросите у пользователя операцию и сохраните результат в переменную, потом с помощью конструкции case выберите в зависимости от операции, какой результат вывести пользователю. Не забудьте про вариант, когда пользователь в качестве операции указал что-то неподходящее, в вашем case должен быть else с адекватной реакцией.

ЦЕЛОЧИСЛЕННЫЙ КАЛЬКУЛЯТОР. ПОДСКАЗКА

- Наконец, обработайте ошибку ZeroDivisionError при делении с помощью конструкции begin-rescue, как мы это делали на уроке.
- Обратите внимание, что если вы хотите в методе puts выводить сразу результат перемножения (или любой другой операции) нескольких переменных, то их нужно сгруппировать в круглые скобки и метод to_s применить у всего этого выражения целиком.



ЦЕЛОЧИСЛЕННЫЙ КАЛЬКУЛЯТОР. ПОДСКАЗКА

- Неправильно!

```
puts a1 * a2.to_s
puts a1.to_s + a2.to_s
```

- Правильно:

```
puts (a1 + a2).to_s # тогда в строку преобразуется
результат сложения a1 и a2
```

КАЛЬКУЛЯТОР С FLOAT

- Напишите калькулятор, который работает с числами с плавающей точкой.
- Обратите внимание, что в этом случае на ноль делить можно (получится бесконечность Infinity) и не надо ловить исключения.
- **Например:**

```
Первое число:
> 92.7
Второе число:
> 0
Выберите операцию (+ - * /):
> /
Результат:
Infinity
```

КАЛЬКУЛЯТОР С FLOAT. ПОДСКАЗКА

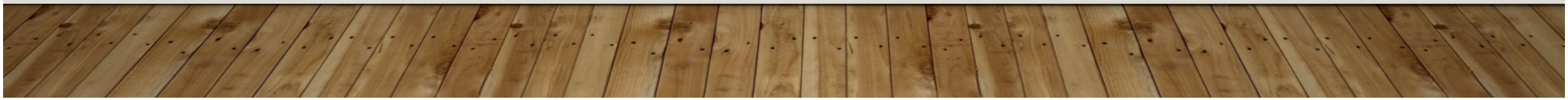
- Для того, чтобы перевести введённые пользователем данные в числа с плавающей точкой, используйте метод строки to_f:

```
number = gets.to_f
```

- В этом случае конструкция begin-rescue уже не нужна.

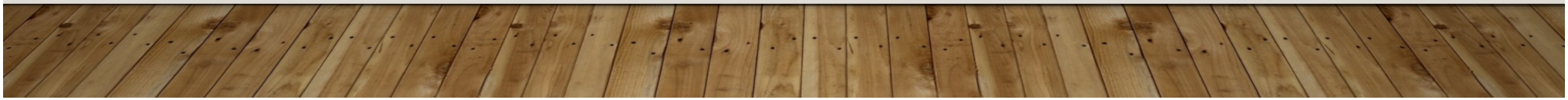
ВИСЕЛИЦА С ОБРАБОТКОЙ ИСКЛЮЧЕНИЙ

- Добавьте в программу Виселица обработку исключений при открытии файла со списком слов и при загрузке картинок-виселиц.
- При ошибке открытия списка слов завершите программу. Если не хватает файлов–картинок, используйте вместо незагрузившихся картинок какую-нибудь строку.
- При открытии файла для чтения единственная легко воспроизводимая ошибка это отсутствие файла. Погуглите какой тип исключения нужно ловить в этом случае. Переименуйте нужные файлы и проверьте, что ваша программа правильно работает.



ВИСЕЛИЦА С ОБРАБОТКОЙ ИСКЛЮЧЕНИЙ. ПОДСКАЗКА

- Ошибки при открытии файла, если вы предварительно проверили его существование с помощью File.exist? — маловероятны. Поэтому, чтобы убедиться, что ваша обработка ошибок работает, сперва удалите такую проверку. Она нам будет не нужна.
- Затем поместите открытие файла в блок begin ... rescue ... end и поймайте ошибку SystemCallError (подробнее см. доки).



ВИСЕЛИЦА С ОБРАБОТКОЙ ИСКЛЮЧЕНИЙ. ПОДСКАЗКА

- Обратите внимание, что мы ловим SystemCallError чтобы обработать все возможные ошибки связанные с открытием файла. Не только отсутствие файла, но и всевозможные системные ошибки доступа к нему.
- Этим мы с одной стороны расширяем группу отлавливаемых ошибок (что не очень хорошо), с другой — остаемся в рамках определенной группы (вызовы ОС), что уместно поскольку единственный вызов, который делаем — чтение файла.
- http://stackoverflow.com/questions/11457795/how-to-rescue-all-exceptions-under-a-certain-namespace

СПРАВОЧНАЯ ИНФОРМАЦИЯ

- Работа с исключениями в Ruby (1)
- Работа с исключениями в Ruby (2)
- Когда обработал все исключения :)
- Ищем ошибку в программе :)
- Про исключения в программировании

СПАСИБО ЗА ВНИМАНИЕ!

Ошибки и исключения