ЛЕКЦИЯ 5

ПИШЕМ ВОЛШЕБНЫЙ ШАР.

ОБЪЕКТЫ, ПЕРЕМЕННЫЕ, УСЛОВНЫЙ ОПЕРАТОР IF

ВАЖНЕЙШЕЕ ПРАВИЛО ПРОГРАММИРОВАНИЯ

«Любая программа начинается с постановки задачи».

Вы не можете написать программу, пока вы не поняли, какой результат от неё ждёте. Продумайте вашу программу до того, как её писать. Это во-первых, облегчит вам работу, во-вторых, ваша программа будет содержать меньше ошибок.

Задача для программы «Волшебный шар»:

Написать программу, которая реализует поведение игрушки волшебный шар.

Чтобы понять как работает шар, посетим страницу Википедии, посвященную магическому шару. Она называется «Magic 8 Ball». Нас здесь интересуют все возможные варианты ответов, один из которых случайным образом должна выбрать программа.

НАША ПЕРВАЯ ПРОГРАММА

Создадим файл **8ball.rb**, скопировав варианты ответов из Википедии и оформив это в виде массива ruby.

```
answers = [
  # Положительные
  "Бесспорно",
  "Предрешено",
  "Никаких сомнений",
  "Определённо да",
  "Можешь быть уверен в этом",

  # Нерешительно положительные
  "Мне кажется — «да»",
  "Вероятнее всего",
  "Хорошие перспективы",
  "Знаки говорят — «да»",
  "Да",

  # Нейтральные
  "Пока не ясно, попробуй снова",
  "Спроси позже",
  "Лучше не рассказывать",
  "Сейчас нельзя предсказать",
  "Сконцентрируйся и спроси опять",

  # Отрицательные
  "Даже не думай",
  "Мой ответ — «нет»",
  "По моим данным — «нет»",
  "Перспективы не очень хорошие",
  "Весьма сомнительно"
]
```

НАША ПЕРВАЯ ПРОГРАММА

После этого мы выведем на экран произвольный элемент этого массива с помощью специального метода **sample**.

Добавьте в вашу программу в конце следующую строчку. И сохраните программу в папке урока.

```
puts answers.sample
```

Запускаем программу из консоли:

```
ruby 8ball.rb
```

КАК И ЗАЧЕМ ПИСАТЬ КОММЕНТАРИИ (В RUBY И НЕ ТОЛЬКО)

В Ruby комментарии начинаются с # — всё, что идёт после этого символа и до конца строки будет проигнорировано руби. Как будто вы ничего и не писали.

```
# Тут можно писать всякую ерунду ;)
```

Комментарии — очень нужная вещь.

Пишите их, чтобы пояснять действия ваших инструкций в программах для вас в будущем или для других людей, которые будут читать ваши программы.

Очень помогает при обучении прежде, чем писать программу, сначала написать всю программу в виде комментариев на русском языке прямо в файле программы. В этих комментариях просто описать все шаги — что и как делает программа.

ОТСТУПЫ В ПРОГРАММИРОВАНИИ

Для удобства восприятия текста программы их авторы используют отступы. Например, если вы пишете блок между скобками или на нескольких строчках, всегда используйте отступы.

Согласитесь, что такой текст читать гораздо сложнее и непонятнее:

```
answers = ['один’,
'два’,
        'три', 'четыре’,
'пять’
]
```

ОТСТУПЫ В ПРОГРАММИРОВАНИИ

Чем такой:

```
answers = [ 'один', 'два', 'три', 'четыре', 'пять’ ]
```

В хорошо отформатированном тексте гораздо проще найти и избежать ошибки. Хотя для ruby оба варианта равнозначны.

VS Code обычно сам подсказывает вам правильные отступы. Но их всегда можно сделать вручную клавишей **Tab**.

ОШИБКИ В ПРОГРАММИРОВАНИИ

Пока вы новичок — ваши программы часто не будут работать с первого раза из-за досадных ошибок и опечаток. Это нормально, все через это проходят.

К счастью большинство таких ошибок довольно просто обнаружить и исправить. Можно запустить Ruby в режиме проверки файла, добавив к запуску ключ **-c** (параметры с минусами, передаваемые в программу часто называют ключами).

Для этого в консоли (находясь конечно в папке с программой) выполните:

```
ruby -c 8ball.rb
```

ОШИБКИ В ПРОГРАММИРОВАНИИ

Забытая скобка, забытая запятая, опечатка в названии переменной или метода — это те причины, по которым ваша программа может даже не запуститься. Ни в коем случае не расстраивайтесь, просто читайте текст ошибок и исправляйте ошибки по одной — сверху вниз.

В тексте ошибки вам не нужно понимать всё, что написано. Достаточно в начале найти название файла и после двоеточия цифру. Эта цифра — номер строки, где обнаружена ошибка.

Но обратите внимание, что ошибка может оказаться не ровно на этой строке, а плюс-минус одна строка (пустые строки и комментарии не учитываются!).

ОШИБКИ В ПРОГРАММИРОВАНИИ

Если же плюс–минус одна строка в порядке, то возможно ошибку следует искать в симметричной конструкции. Например, если ошибка в районе строки, где есть символ } или], или), то возможно дело в том, что для этой закрывающей скобки нету открывающей. На какой-то совсем другой строке, где она должна быть по смыслу вашей программы.

Точно также ошибки могут проявиться и без всяких ключей, при обычном запуске программы. Искать и исправлять их нужно точно также – по одной, всегда сначала самую верхнюю из списка ошибок. Бывает, что исправление одной самой верхней ошибки автоматически исправляет и все остальные.

ОШИБКИ В ПРОГРАММИРОВАНИИ

После того, как мы написали нашу первую более-менее сложную программу и научились делать разметку и комментарии в тексте программ, а также бороться с опечатками в Ruby — можно переходить к изучению основ программирования.

ВОЛШЕБНЫЙ ШАР С ПРИВЕТОМ

Сделайте так, чтобы ваш «Волшебный шар» прежде, чем выводить свой ответ, здоровался с пользователем:

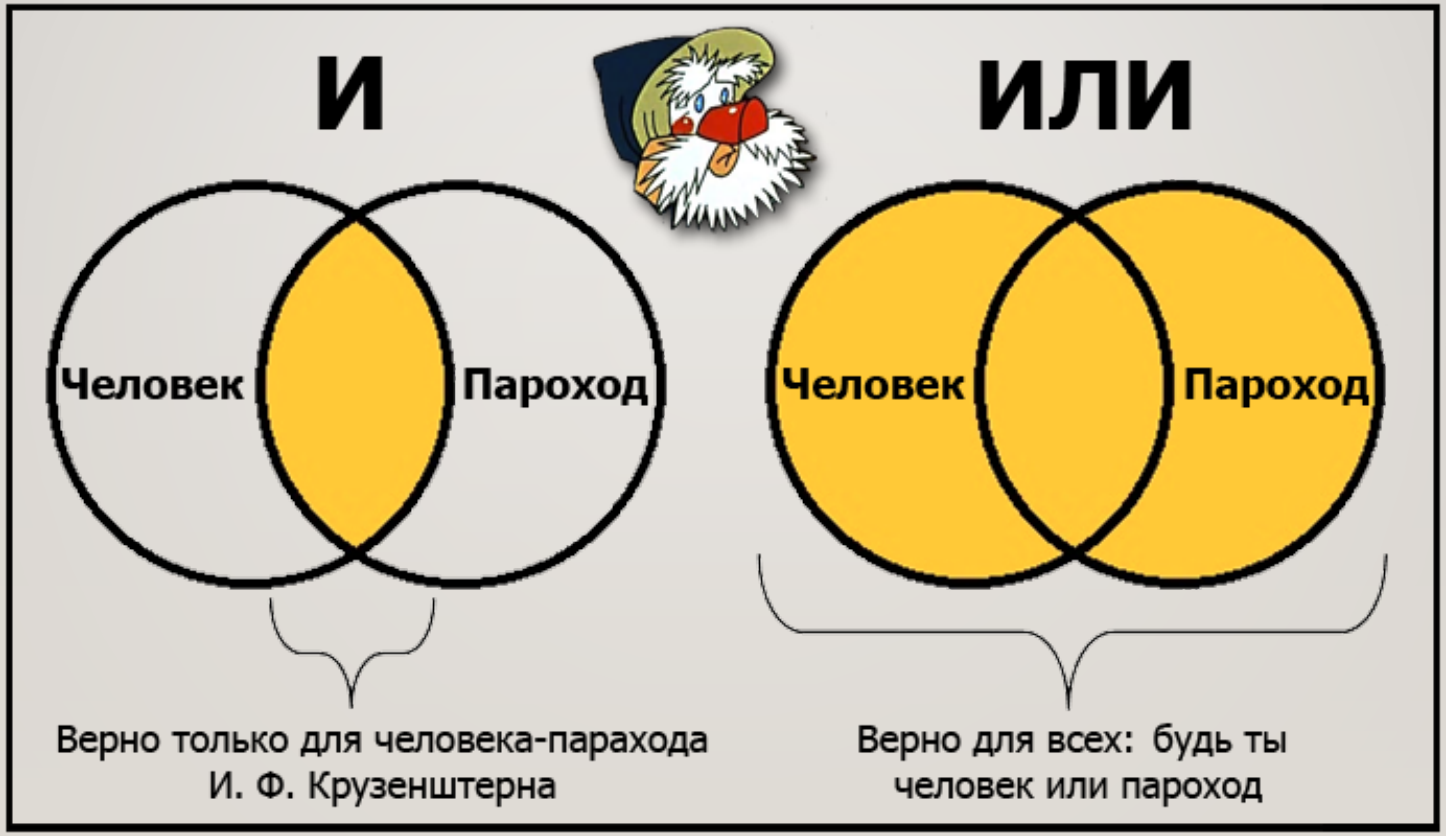
Например:

```
Привет, дорогой друг. Отвечаю на твой вопрос...


Знаки говорят – да.
```

Да, пустая строка между приветствием и ответом тоже нужна!

ОБЪЕКТЫ, ПЕРЕМЕННЫЕ, УСЛОВНЫЙ ОПЕРАТОР IF



ОБЪЕКТЫ В ПРОГРАММАХ

В реальном мире все состоит из объектов: люди, животные, дома, горы, реки, планеты, дожди и т. д.

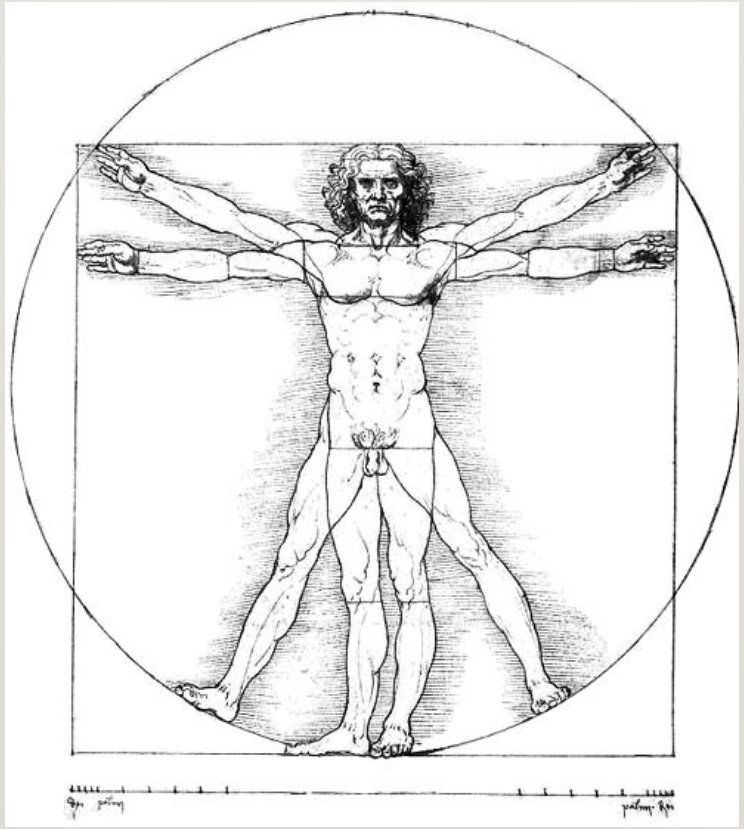
ОБЪЕКТЫ В ПРОГРАММАХ

Современные программы принято тоже создавать из объектов. Но объектов «абстрактных», то есть виртуальных, существующих только в тексте программы и в памяти компьютера.

Например в программе на Руби любое число или любая строка — это объект.

В жизни у любого конкретного объекта есть его тип. Например, дом, в котором вы живете имеет тип «дом», как и множество других домов, даже непохожих на ваш. Или вы сами — объект типа «человек».

ОБЪЕКТЫ В ПРОГРАММАХ



ОБЪЕКТЫ В ПРОГРАММАХ

В программировании тип конкретного объекта называется класс. Чтобы узнать класс любого объекта в Ruby можно написать после этого объекта .class.name.

Вот несколько примеров классов в Ruby:

Строка:

```
puts "Я строка".class.name
```

Выведет в консоли:

```
String
```

ОБЪЕКТЫ В ПРОГРАММАХ

Целое число (что такое целое число):

```
puts 5.class.name
```

Выведет:

```
Fixnum
```

Число с плавающей точкой (что такое число с плавающей точкой):

```
puts 15.2.class.name
```

Выведет:

```
Float
```

ОБЪЕКТЫ В ПРОГРАММАХ

Наш обычный мир состоит не только из объектов, но из различных процессов, действий, взаимодействий между объектами («люди» ходят в «магазины», «рыбы» плавают в «водоёмах» и т. д.).

Программы в современных языках программирования тоже по большому счету состоят из объектов и различных взаимодействий между ними. Этот вопрос еще подробнее раскроем далее.

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

Для того, чтобы обращаться с объектами, нам нужно их как-то назвать, повесив на объект ярлык. Например при рождении нового объекта типа «человек» ему обычно дают имя. И вместе с фамилией и отчеством это позволяет отличить этого ребенка от десятков других в этом же роддоме.

Вся эта уникальная информация записывается в свидетельство о рождении и позже и в паспорт. Эти документы для этого и придуманы, чтобы точно определять конкретного человека из миллионов других.

В программе чтобы оперировать объектами их тоже нужно как-то отличать, как-то на них ссылаться.

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

Для этого в программировании есть такое понятие как переменные. Итак, грубо говоря, переменная — это ярлычок, указывающий на конкретный объект.

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

Сама по себе переменная в Ruby — просто уникальный набор символов английского алфавита и цифр, который можно использовать в программе для действий с объектом. Этот набор символов называемся «имя переменной» и вы сами придумываете имена своих переменных.

Связь объекта с переменной происходит с помощью знака =.

```
hello = 'Привет, ребята!'
```

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

Таким образом мы переменной hello присвоили объект 'Привет, ребята!' класса «строка» (String).

Обратите внимание, что таким образом мы создали объект, он теперь сохранён в памяти компьютера. В этом случае принято говорить, что мы присвоили переменной hello значение 'Привет, ребята!'.

Если мы после этого напишем

```
puts hello
```

То программа выведет эту строку из переменной hello на экран, потому что программа уже знает, что такое hello и что в этой переменной лежит.

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

Переменная может быть пустой, то есть не указывать ни на какой конкретный объект. Как пустой бланк паспорта. А может указывать на один объект, а потом на другой, если мы так захотим.

Такие номера проделывают с паспортами разные жулики, к счастью в программировании в отличие от реальной жизни, это не является правонарушением. :)

ОПЕРАЦИЯ IF-ELSE (ЕСЛИ-ИНАЧЕ)

Это самая важная, самая простая и самая часто применяющаяся конструкция, которая есть в любом языке программирования.

ОПЕРАЦИЯ IF-ELSE (ЕСЛИ-ИНАЧЕ)

Суть этой операции в проверке какого-то условия и выполнении определённых действий в зависимости от того, выполнено это условие или нет.

```
if (условие)
        сделать, если условие выполнено
else
        сделать, если условие не выполнено
end
```

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

Для того, чтобы записать условие, важно понять, что такое «Логический тип».

По-другому он называется булевский тип (по-английски Boolean) — это такой тип данных, который может содержать только одно из двух значений: true или false, истина или ложь. Думайте об этом как об ответе на закрытый вопрос, на который можно ответить только «Да» или «Нет».

Для формулировки условия нам нужно записать тот самый закрытый вопрос и для этого в Ruby есть специальный синтаксис.

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

В самом простом случае нам нужно проверить, совпадает ли значение переменной с каким-то нашим ожиданием. В этом нам поможет оператор «равно-равно» ==, который сравнивает то, что справа от него, с тем, что слева от него, и говорит true, если эти два объекта совпадают и false, если нет.

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

Запись if в этом случае будет выглядеть вот так:

```
if (hello == 'Привет, ребята')
        puts 'Приветствую!'
else
        puts 'Кто вы?'
end
```

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

Условие в Ruby можно записывать и без скобок, но, пока вы новичок, рекомендуется писать именно так. В приведённом примере если в переменной hello содержится строка 'Привет, ребята', то программа выведет на экран текст «Приветствую!», а если это условие не выполнено, то напишет текст «Кто вы?».

ПЕРЕМЕННЫЕ, СОЗДАНИЕ И ПРИСВОЕНИЕ

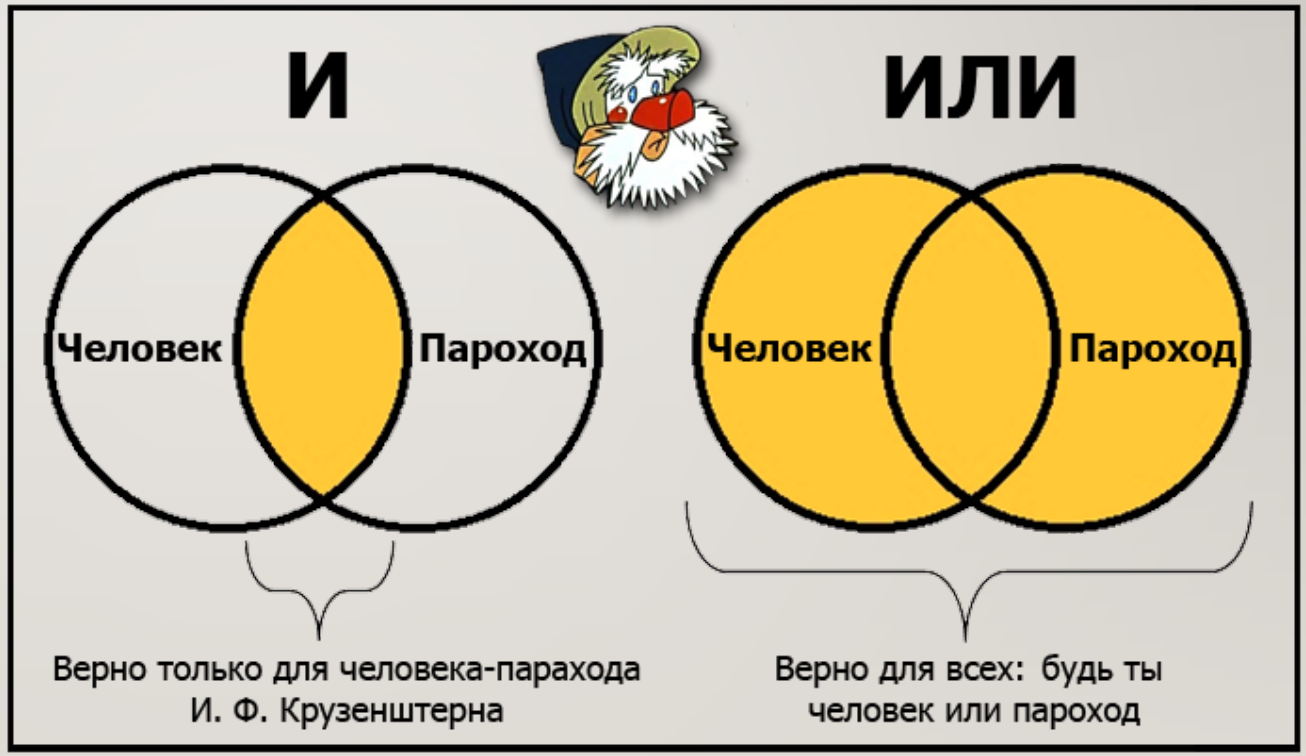
Если нам необходимо сравнить два объекта и сделать что-то, если они не совпадают, нам необходимо использовать оператор «не-равно» !=:

```
if (hello != 'Здравствуйте, Господа’)
        puts 'Не изволите ли поздороваться, сударь?’
else
        puts 'И вам не хворать, любезнейший’
end
```

В этом случае если в переменной hello содержится что-то, отличное от строки "Здравствуйте, Господа", мы пойдём по верхнему пути, а если (вдруг) содержится, то по нижнему.

ОПЕРАТОРЫ И, ИЛИ

Если нам нужно проверить сложное условие, которое состоит из двух сравнений, то нам понадобятся операторы || и &&— это операторы ИЛИ и И. Они позволяют объединять вместе сразу несколько сравнений.



ОПЕРАТОРЫ И, ИЛИ

Они используются для сложных проверок:

```
if (hello == 'Здравствуйте, Господа' || hello == 'Привет, ребята')
        puts 'Привет!'
else
        puts 'Хам!'
end
```

ОПЕРАТОРЫ И, ИЛИ

Если в переменной hello строка 'Здравствуйте, Господа' **или** строка 'Привет, ребята', то этот код напишет нам «Привет». Если же нет, всё пойдёт по другому сценарию и программа будет ругаться.

```
if (hello != 'Здравствуйте, Господа' && hello != 'Привет, ребята')
        puts 'Хам!'
else
        puts 'Привет!'
end
```

ОПЕРАТОРЫ И, ИЛИ

Всё то же самое. Обратите внимание на логику: если в переменной hello не содержится 'Здравствуйте, Господа' **и** не содержится 'Привет, ребята', то значит, с нами не поздоровались. Имеем право возмутиться. :)

Важно четко запомнить — оператор И объединяет другие условия в одно большое условие, и это большое условие выполняется только когда **все условия** выполняются.

Оператор же ИЛИ объединяет другие условия в одно большое условие, и это большое условие выполняется когда **хотя бы одно из условий** выполняются.

ВЫВЕСТИ БОЛЬШЕЕ ИЗ ЧИСЕЛ

В программе объявить две численные переменные, разные по значению. Вывести их на экран.

Затем с помощью оператора if выбрать наибольшее из них и вывести его на экран. Если числа равны — программа должна сообщить об этом.

Например:

```
172.169
31.514
Наибольшее число: 172.169
```

ВЫВЕСТИ БОЛЬШЕЕ ИЗ ЧИСЕЛ ПОДСКАЗКА

Чтобы вывести на экран число используйте метод puts. Если нужно преобразовать число к строке (чтобы соединить с другой строкой), используйте метод to_s.

```
number = 777

puts "Наибольшее число:" + number.to_s
```

Попробуйте написать без .to_s.

СРЕДНЕЕ АРИФМЕТИЧЕСКОЕ ДВУХ ЧИСЕЛ

Напишите программу, которая находит среднее арифметическое двух чисел: заведите две переменные и запишите в каждую из них по числу. Потом заведите третью и запишите в неё среднее арифметическое первых двух. А потом выведите всё это на экран:

Например:

Первое число: 2308

Второе число: 13

Среднее: 1160

СРЕДНЕЕ АРИФМЕТИЧЕСКОЕ ДВУХ ЧИСЕЛ ПОДСКАЗКА

Чтобы вывести на экран число, используйте метод puts, нужно преобразовать его к строке с помощью метода to_s.

```
number = 42

puts 'Первое число:' + number.to_s
```

Чтобы вычислить среднее арифметическое двух чисел, нужно сложить их с помощью оператора + и поделить на два с помощью оператора деления /:

```
average = (number1 + number2) / 2
```

ЧЕТНОЕ ЛИ ЧИСЛО?

Создать переменную — большое число. Проверить является ли оно четным и вывести результат на экран.

Например:

```
987298

Число четное
```

ЧЕТНОЕ ЛИ ЧИСЛО?

Создать переменную — большое число. Проверить является ли оно четным и вывести результат на экран.

Например:

```
987298

Число четное
```

СЕГОДНЯ ВЫХОДНОЙ?

Напишите программу, которая подсказывает, выходной сегодня или нет (для простоты мы не учитываем государственных праздников, итак много дней для безделья):

```
Сегодня выходной!
```

или

```
Сегодня будний день, за работу!
```

ПРОСТО МОНЕТКА

Напишите программу «Монетка»: она генерирует случайное число 0 или 1. И выводит на экран «Выпал орел» для нуля или «Выпала решка» для единицы.

НЕПРОСТАЯ МОНЕТКА

Модифицируйте программу из предыдущей задачи так, чтобы иногда выпадало еще и "ребро" («Монета встала на ребро»). Причем ребро должно выпадать намного реже, чем орел или решка (вероятность 1/10 или меньше).

КОМАНДА GETS

Мы уже знакомы с командой puts, которая выводит строку на экран (put string):

```
puts "Привет, я строка, меня вывели на экран!"
```

У этой команды есть друг, команда gets, которая «берёт строку» (get string) и кладёт её в переменную, которой мы хотим присвоить значение строки, введённой пользователем во время выполнения программы:

```
my_string = gets
```

КОМАНДА GETS

Если выполнить программу, состоящую из одной только этой строчки, то программа сразу после запуска будет ждать, пока пользователь наберёт какие-то символы и после набора нажмёт клавишу Ввод (Enter).

После нажатия Enter все эти символы (вместе со специальным символом перевода строки **\n**, который обозначает нажатую клавишу Enter) попадут в переменную my_string.

ВВОД ДАННЫХ В ПРОГРАММУ ИЗ КОМАНДНОЙ СТРОКИ

Продемонстрируем на примере: как обычно мы создадим в рабочей директории c:\project%username папку lesson5 и напишем в ней программу, которая спрашивает у пользователя, как его зовут:

```
puts "Привет! Как тебя зовут?"

name = gets

puts "Привет, " + name + ", как дела?"
```

ВВОД ДАННЫХ В ПРОГРАММУ ИЗ КОМАНДНОЙ СТРОКИ

Программа gets.rb выводит на экран приветствие и ждёт, пока пользователь введёт своё имя и потом нажмёт Enter, после этого программа здоровается с пользователем, назвав его по имени (тому самому, что он только что ввёл) и спрашивает, как у него дела. :)

ВВОД ДАННЫХ В ПРОГРАММУ ИЗ КОМАНДНОЙ СТРОКИ

Для того, чтобы склеить строку из нескольких строк мы используем оператор + - он работает очень просто:

```
puts "а" + "б"
```

Выведет на экран аб.

```
puts "а" + " и " + "б"
```

Выведет на экран а и б.

ВВОД ДАННЫХ В ПРОГРАММУ ИЗ КОМАНДНОЙ СТРОКИ

В Windows строка будет передана в программу из консоли в кодировке Windows и в программе может возникнуть ошибка, вы увидите абракадабру или что похуже.

Потому что мы работаем в кодировке UTF-8, к результату работы (строке, полученной от пользователя) мы с помощью точки применим метод encode("UTF-8").

Что такое кодировка?

Подробнее о методах работы с кодировками в Ruby (на английском).

ВВОД ДАННЫХ В ПРОГРАММУ ИЗ КОМАНДНОЙ СТРОКИ

Также мы уберём из результата, полученного gets, последний символ — перенос строки (\n = Enter). Для этого мы используем метод chomp. Подробнее о методе chomp.

Получится вот так:

```
puts "Привет! Как тебя зовут?"

name = gets.encode("UTF-8").chomp

puts "Привет, " + name + ", как дела?"
```

ПИШЕМ ТЕКСТОВЫЙ КВЕСТ С НЕЛИНЕЙНЫМ СЮЖЕТОМ

```
puts "Вы решили прогуляться по темному переулку и наткнулись на спортивных ребят

1.Попытаться убежать

2.Продолжать идти"

choice = gets.chomp

if choice == "1"

        abort "Ребята без труда догнали вас и побили. Не знаю, за что"

else
```

ПИШЕМ ТЕКСТОВЫЙ КВЕСТ С НЕЛИНЕЙНЫМ СЮЖЕТОМ

```
puts "Один из ребят вышел вперёд и спросил \“Сигареты есть?\"

1. Дать прикурить

2. -- Не курю"

choice = gets.chomp
if choice == "1"

        abort "Прикурив, ребята отправились дальше"

else

        abort "Ребята расстроились и побили вас. Теперь уже ясно, за что"

end

end
```

КОМАНДА ABORT

Если в какой-то момент повествование дошло до своей кульминации и нам нужно его прекратить, сказав пользователю последнюю строку, то мы делаем это с помощью команды abort:

```
abort "Ну вот и всё!"

puts "Меня нет"
```

Эта команда, как и puts, выведет на экран строку, которую ей передали, а потом немедленно закончит выполнение программы. Следующую строку 'Меня нет' мы на экране уже не увидим.

СВОЙ ВАРИАНТ КВЕСТА

Напишите свой вариант квеста, используя операторы: if-else, encode, chomp, abort.

СПРАВОЧНЫЙ МАТЕРИАЛ

- Логический тип
- Логическая операция
- Операторы в Ruby
- Логический тип
- Операторы И,ИЛИ
- Проблема с русскими буквами
- Подробнее об if-else
- Строка в Ruby и ее свойства
- Чем отличается оператор and и &&
- Работа со строками

СПАСИБО ЗА ВНИМАНИЕ!

Лекция 5. Пишем волшебный шар. Объекты, переменные, условный оператор IF