Задание 2

Выполните упражнения и представьте написанный исходный код куратору вашего направления.

Упражнения 1.

Разработайте метод palindrome?(string), который будет определять, подана ли ему на вход строка string палиндром, т.е. строка, читающаяся одинаково с начала и с конца, при условии игнорирования пробелов, знаков препинания и регистра. Тесты для примеров и проверки:

```
palindrome?("A man, a plan, a canal -- Panama")  # => true
palindrome?("Madam, I'm Adam!")                  # => true
palindrome?("Abracadabra")                       # => false (nil is also ok)
```

Упражнение 2

Разработайте функцию count_words(string), которая будет возвращать хэш со статистикой частоты употребления входящих в неё слов. Тесты для примеров и проверки:

```
count_words("A man, a plan, a canal -- Panama")
    # => {'a' => 3, 'man' => 1, 'canal' => 1, 'panama' => 1, 'plan' => 1}
count_words "Doo bee doo bee doo"
    # => {'doo' => 3, 'bee' => 2}
```

Упражнение 3

Мы будем разрабатывать методы для программы Камень-Ножницы-Бумага.

Метод rps_game_winner должен принимать на вход массив примерно следующей структуры [["Armando", "P"], ["Dave", "S"]], где P - бумага, S - ножницы, R - камень, и функционировать следующим образом:

- если количество игроков больше 2 необходимо вызывать исключение WrongNumberOfPlayersError;
- если ход игроков отличается от ‘R’, ‘P’ или ‘S’ необходимо вызывать исключение NoSuchStrategyError;
- в иных случаях необходимо вернуть имя и ход победителя, если оба игрока походили одинаково - выигрывает первый игрок.

Упражнение 4

Анаграмма — литературный приём, состоящий в перестановке букв или звуков определённого слова (или словосочетания), что в результате даёт другое слово или словосочетание.

Реализуйте метод combine_anagrams(words), который принимает на вход массив слов и разбивает их в группы по анаграммам, регистр букв не имеет значение при определении анаграмм. Тест для примера и проверки:

```
# input: ['cars', 'for', 'potatoes', 'racs', 'four', 'scar', 'creams', 'scream']
# output: [ ["cars", "racs", "scar"],
#           ["four"],
#           ["for"],
#           ["potatoes"],
#           ["creams", "scream"] ]
```

Упражнение 5

Реализуйте класс Dessert с геттерами и сеттерами для полей класса name и calories, конструктором, принимающим на вход name и calories, а также двумя методами healthy? (возвращает true при условии калорийности десерта менее 200) и delicious? (возвращает true для всех десертов).

Упражнение 6

Создайте класс JellyBean расшряющий класс Dessert новыми геттерами и сеттерами для атрибута flavor. Измените метод delicious?, он должен возвращать false только в тех случаях, когда flavor равняется "black licorice".

Упражнения 7

Сделайте возможным выполнять следующий код:

```
5.dollars.in(:euros)
10.euros.in(:rubles)
```

Вы должны поддерживать следующие валюты: dollars, euros и rubles, курс обмена установите следующий: $1 = 32,26 руб., 1€ = 43,61 руб.

Должны подерживаться как единственное, так и множественное именование метода, т.е. 1.dollar.in(:rubles) и 1.ruble.in(:euro) должны работать.

Упражнение 8*

Реализуйте метод attr_accessor_with_history который предоставляет ту же фунциональность, что и attr_accessor, но также записывает всю историю изменения атрибута. Пример фунционирования:

```
class Foo
    attr_accessor_with_history :bar
end

f = Foo.new     # => #<Foo:0x127e678>
f.bar = 3       # => 3
f.bar = :wowzo  # => :wowzo
f.bar = 'boo!'  # => 'boo!'
f.bar_history   # => [nil, 3, :wowzo, 'boo!']
```

Упражнение 9

Адаптируйте ваше решение из Упражнения 1, чтобы вместо palindrome?("foo") вы могли его использовать как "foo".palindrome?

Адаптируйте ваше решение из Упражнения 1 так, чтобы оно работало для Enumerable.

```
[1,2,3,2,1].palindrome? # => true
```

Упражнение 10*

Необходимо реализовать возможность Декартового Произведение (http://ru.wikipedia.org/wiki/%D0%9F%D1%80%D1%8F%D0%BC%D0%BE%D0%B5_%D0%BF%D1%80%D0%BE%D0%B8%D0%B7%D0%B2%D0%B5%D0%B4%D0%B5%D0%BD%D0%B8%D0%B5). Пример:

```
a × b = [ [:a,4], [:a,5], [:b,4], [:b,5], [:c,4], [:c,5] ]
```

Создайте класс CartesianProduct, на вход конструктора которого можно подать две последовательности. Определите метод each, ожидающий на входе блок и применяющий этот блок к каждому элементу полученного произведения. Пример функционирования:

```
c = CartesianProduct.new([:a,:b], [4,5])
c.each { |elt| puts elt.inspect }
# [:a, 4]
# [:a, 5]
# [:b, 4]
# [:b, 5]

c = CartesianProduct.new([:a,:b], [])
c.each { |elt| puts elt.inspect }
# Ничего не выводится
```