Szkolenie Rails

Zadanie1. Stwórz klasę Osoba posiadające następujące dane takie jak:
-name
-surname
-age
-gender

Stwórz odpowiedni initalize , który przyjmuje wszystkie argumenty.
Dodaj metody dostępowe do zmiennych obiektu.
Stwórz przykładowe obiekty.

Zadanie2. Stwórz klasę Samochód posiadający następujące atrybuty:
-brand
-model
-price

Stwórz metody dostępowe do atrybutów.

Stwórz metodę show -wyświetlającą marke oraz cenę w jednej linii.

Zadanie3. Zrefaktoryzuj stworzone klasy z Zadanie1 i Zadanie2 używając attr_accessor.

Zadanie4. Zdefiniuj klasę odpowiedzialną za przechowywanie informacji o Zawodniku oraz Grze.
Game powinna zawierać:
-name
-players
-metodę who_win?(jeśli jeden z zawodników ma więcej niż 100 punktów , to wygrywa , jeśli 2 zawodników ma więcej niż 100 punktów , to wygrywa ten , który ma większą liczbę punktów.
Metoda powinna wypisać na ekran imię i nazwisko zawodnika.

Oraz Player , która powinna zawierać:
-name
-surname
-number_of_points

Metody dostępowe oraz dodatkową metodę full_name

Zadanie5. Stwórz Klasę Employee dziedziczącą po Person.

Dodatkowo Employee powinien posiadać salary oraz commision.

Oraz metodę salary_year zwracającą pensje roczną.

Zadanie6. Zdefiniuj klasę Bicycle posiadający gears, wheels, brand,seats.

Zdefiniuj Initalize z brand i gears standardowo =1.

Zdefniuj Klasę Tandem dziedziczącą po Bicycle , użyj metody super.

Zadanie7. Stwórz Klasę BMW dziedziczącą po Car , dodatkowo zmień initalize tak ,żeby zawsze jako brand było name.
Przedefiniuj metodę show jako puts "My the best car BMW”

Zadanie8. Zdefinuj klasę Product posiadający name,description , specification. Dodaj odpowiedni konstruktor. Zdefiniuj Klasę Guaranty , która odnosi się do danego produktu oraz valid_until.

Zdefiniuj metodę publiczną valid?. Jeśli produkt jest ważny zwraca true w przeciwnym wypadku false. Jeśli Gwarancja jest nieważna to czyści Gwarancje ustawiając na nil valid_until oraz product.

Użyj metody protected do niszczenia obiektu.

Zadanie9 . Zdefiniuj Klasę User oraz Klasę Friend. Obie klasy powinny Dziedziczyć po Person. Zdefinuj metodę bump_karma , która zwraca puts "bump karma #{name".

Zdefinuj metodę vote(friend) , która wywołuje bump_karma na osobie oraz na przyjacielu.

Zadanie10. Zdefiniuj klasę Post o polu title. Utwórz metody klasowę author , która zawsze zwraca takiego samego autora oraz metodę full_title zwracajacą authora wraz z tytułem.

Zadanie11. Zdefiniuj klasę Fraction , która odpowiada za reprezentacje ułamków.
Zdefiniuj metody odpowiedzialne za dodawanie , odejmowanie ułamków , oraz skracanie danego ułamka. Skorzystaj z przeciążenie operatorów.

Zadanie12. Z wcześniejszych ćwiczeń skorzystaj z definiowanej Osoby (Person) , stwórz metodę statyczną get_all_people zwracającą full_name wszystkich osób. Zdefiniuj metodę to_s.

Zadanie13. Do klasy Employee zdefiniuj metodę statyczną zwracającą sumę wszystkich pensji pracowników.

Zadanie14. Dopisz do klasy Fixnum metodę , która sprawdza czy liczba ma największą liczbę dzielników , to znaczy nie ma liczby z przedziału od 1 do liczba -1 , która ma większą liczbę dzielników niż dana liczba.

Zadanie15. Przerób następującą metodę tak żeby było możliwe wywołanie
new_game("Street Figher II")

```
def new_game(name, year, system)
 {
 name: name,
 year: year,
 system: system
 }
end
game = new_game("Street Figher II", nil, nil)
```

Zadanie16.
Przerób następujący kod , tak żeby było możliwe wywołanie
new_game("Street Figher II",name: "SNES", system: 1992)

```
def new_game(name, year=nil, system=nil)
 {
 name: name,
 year: year,
 system: system
 }
end
game = new_game("Street Figher II", "SNES", 1992)
```

Zadanie17.
Zaimplementuj metodę initialize tak żeby trzymała name,system and year jako zmienne instancji:

```
class Game
 def initialize(name, options={})
 end
end
```

Zadanie18. Przerób wspomniany kod , tak żeby używał attr_accessor:

```
class Game
 def initialize(name, options={})
 @name = name
 @year = options[:year]
 @system = options[:system]
 end
 def name
 @name
 end
 def year#@year
 end
 def system
 @system
 end
end
```

Zadanie19. Dodaj do naszej klasy atrybut created_at , tak żeby w initialize pobierał aktualny czas:

```
class Game
 attr_accessor :name, :year, :system
 def initialize(name, options={})
 @name = name
 @year = options[:year]
 @system = options[:system]
 end
end
```

Zadanie20. Dodaj 2 klasy ArcadeGame i ConsoleGame dziedziczące po Game.
Zadanie21. Dodaj metodę to_s do klasy:

```
class ConsoleGame < Game
end
```

Wywołaj puts na danym obiekcie co się stanie??