Ruby — programowanie

1 Cel laboratoriów

Instalacja kompilatora Ruby. Zapoznanie się z podstawami języka programowania.

2 Dlaczego warto nauczyć się Ruby

Języki programowania są podobne do języków naturalnych. Każdy język programowania należy do jednej lub więcej kategorii. Możliwe kategorie to obiektowego, deklaratywne czy proceduralne. Jeżeli nauczysz się jednego języka dużo łatwiej jest nauczyć się drugiego języka w tej samej kategorii.

Ruby jest językiem programowania ogólnego przeznaczenia opracowanym w 1990 przez Yukihiro Matsumoto „Matz". Jest to także jeden z najlepszych języków by zacząć się uczyć kodować. Jest obiektowy.

Porównanie języka C++ i Ruby. Przykład "Hello World"

Ruby jest uważane za język programowania o wyższym poziomie niż C ++. Język wysokiego poziomu jest bardziej abstrakcuyjny. W językach niskiego poziomu należy dbać o szczegóły maszyny wykonującej kod, takie jak: adresy pamięci lub rejestrów CPU. Języki wysokiego poziomu są zbliżone do języka naturalnego.

Ułatwienia:

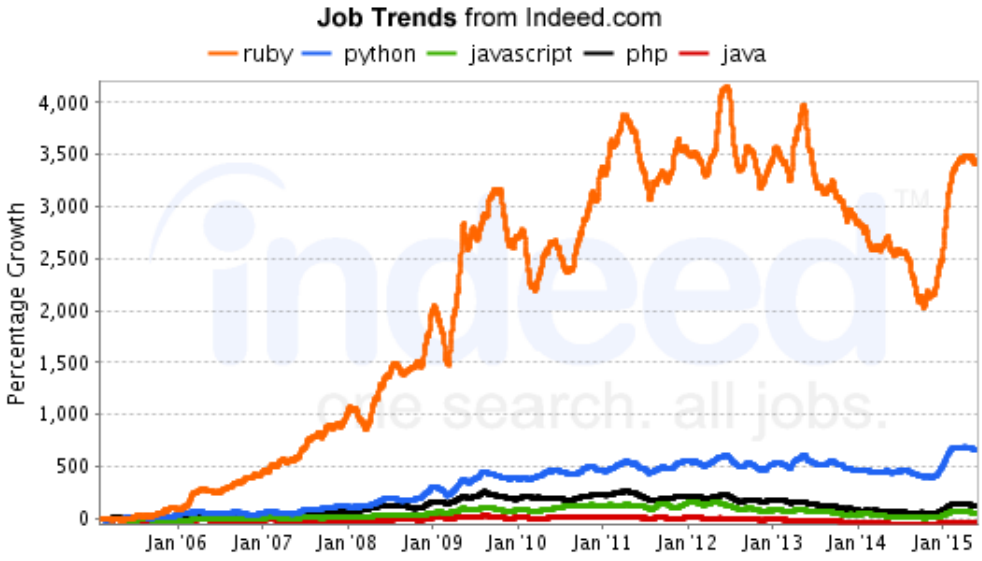
1. Istniejący kod, który można wykorzystać - Gems 60000 bibliotek
2. Dokumentacja
3. Zasoby pozwalające na naukę
4. Społeczność

```
1 #include <iostream>
2 using namespace std;
3
4 int main() {
5     cout << "Hello, world"  << endl;
6     return 0;
7 }
```

Rysunek 1: Przykład programu Hello World C++

```ruby
puts "Hello, world!"
```

Rysunek 2: Przykład programu Hello World w Ruby



Rysunek 3: Popularność Ruby

3 Co to za rubin?

Przejdź do strony: `http://tryruby.org` i zacznij zabawę.

4 Instalacja Ruby pod Windows

Wykorzystaj adresy:

1. Instalator ruby - `http://rubyinstaller.org`
2. Edytor tekstu: `www.sublimetext.co`

5 Wprowadzenie i podstawy

5.1 REPL - reed eveluate print loop

IRB jest to interaktywne środowisko do wykonywania w locie poleceń w języku Ruby. Jest dodany wraz z instalacją.

1. Wykonaj w cmd polecenie:

   ```
   irb
   ```

2. Następnie wpisz:

```
"hello world"
puts "hello world"

3.times { puts "Hello World" }

# this is a comment
puts 5 # so is this
3 # and this

p "Got it" # => Got it
```

Co wynika z zadań. Wszystko jest ewaluowane i nie musi być przypisane do zmiennej. Po znaku „=>" obserwuje się co dana metoda zwraca (return). W drugim przykładzie „nil" oznacza, że nic nie jest zwrócone, ale w trakcie jej wykonania wypisany zostanie komunikat na ekranie.

5.2 Instrukcje sterujące

Pracuj z plikami z podkatalogu Lecture02-Control-Flow.

1. Instrukcja warunkowa if/elseif/else oraz unless: otwórz plik if_unless.rb w sublime. Wykonaj plik poleceniem w cmd:

   ```
   ruby if_unless.rb
   ```

2. Instrukcja while/until: otwórz plik while_until.rb w sublime. Wykonaj plik.

3. Zapisy jednolinijkowe: otwórz plik modifier_form.rb w sublime. Wykonaj plik.

4. Wartości logiczne. W Ruby wszystko ma wartość TRUE opócz $false$ i nil. Otwórz plik true_false.rb w sublime. Wykonaj plik.

5. Potrójna równość „===". Stosowane do „regular expression", które są wzorcami opisującymi łańcuch symboli. Otwórz plik triple_equals.rb w sublime. Wykonaj plik.

6. Wyrażenie „case". Otwórz plik case_expressions.rb w sublime. Wykonaj plik. Operator „===" jest nazywany operatorem case, bo w domyśle jest tam właśnie używany.

7. Pętla „for". Sporadycznie używana. Otwórz plik for_loop.rb w sublime. Wykonaj plik.

5.3 Funkcje i metody

Pracuj z wskazanymi plikami z podkatalogu Lecture03-Functions-Methods.

Ważne by wiedzieć jak i jakie wartości są zwracane. w Ruby, każda funkcja lub metoda ma co najmniej jedną klasę, do której będzie należeć. Chociaż nie zawsze można zobaczyć ją zapisaną wewnątrz klasy, to każda funkcja lub metoda należy do klasy. Zatem w Ruby mówimy o metodach.

1. Metody mogą mieć nawiasy, ale są one całkowicie opcjonalne zarówno przy definiowaniu jaki wywoływaniu metody. Jeżeli uważasz, że za pomocą nawiasów kod jest bardziej przejrzysty to można ich używać. Ale nie ma potrzeby, by to zrobić. Otwórz plik parens.rb w sublime. Wykonaj plik.

2. Nie ma potrzeby deklarowania typu parametru. Polecenie „return" też jest opcjonalne. Ostatnia wykonana linia kodu w metodzie jest uznawana za element zwracany. Otwórz plik return_optional.rb w sublime. Wykonaj plik. Wprowadź polecenie

   ```
   puts add("ola","ala")
   ```

   i wykonaj ponownie.

3. Nazwa metody może zawierać ma końcu symbol „?" tzw. predykaty, które zazwyczaj zwracją wartość logiczną. Otwórz plik expressive.rb w sublime. Wykonaj plik.

4. Domyślne argumenty. Metody mogą mieć domyślne argumenty. Jeżeli chcesz zapewnić pewną wartość, gdy argument nie zostanie przekazany, to używasz domyślnej wartości. Otwórz plik default_args.rb w sublime. Wykonaj plik. W przykładzie użyty ternary operator „condition? true: false".

5. Splat jest ciekawym parametrem i działa podobnie jak var args. Używa sie w sytuacji, gdy chcemy wykorzystywać parametry metody, ale nie wiemy ile ich dokładnie będzie. Przekazane parametry stają się tablicą. Otwórz plik splat.rb w sublime. Wykonaj plik.

Przez dynamiczne ocenianie i wartościowanie parametrów Ruby daje wiele możliwości i swobody. Jednakże może to prowadzić też do nieoczekiwanego zachowania.

5.4 Bloki

Pracuj z wskazanymi plikami z podkatalogu Lecture04-Blocks.

Bloki są czymś wyjątkowym. Blok składający się z pojedynczej liniii obejmuje się nawiasami klamrowymi, a bloki wielonijkowe zaczyna się słowem kluczowym „do" i kończy słowem „end". Bloki często wykorzystuje się jako iteratory. Przekazywane do metody jako ostatni parametr.

1. Otwórz plik times.rb w sublime. Wykonaj plik. W przykładzie pierwszym jest liczba, metoda „times" i w nawiasach klamrowych to co się ma wykonać. W drugim przykładzie metoda times wykona blok z parametrem „index" który przekazuje się w pipline. index jest zmienna, której wartość zmienia się od zera do liczba iteracji -1. W trzecim przykładzie to samo jest zapisane w formie 1 linii.

2. Wykorzystanie bloku we własnej metodzie w sposób implicit (niejawny). Otwórz plik implicit_blocks.rb w sublime. Wykonaj plik. Należy użyć polecenia „block_given?", by sprawdzić, czy blok przekazany i „yield" by wywołać blok.

3. Wykorzystanie bloku we własnej metodzie w sposób explicite (jawny). Otwórz plik explicit_blocks.rb w sublime. Wykonaj plik. Wykorzystaj „&" przed ostatnim parametrem i użyj metody „call" by blok wykonać.

5.5 Odczytywanie i zapisywanie plików

Pracuj z wskazanymi plikami z podkatalogu Lecture05-Files-Environment-Vars.

Przedstawione zostaną sposoby odczytywania i zapisywania plików. Odczytywanie wartości ze zmiennych środowiskowych.

1. Otwórz plik test.txt w sublime.

2. Otwórz plik ireading_from_file.rb w sublime. Wykonaj plik. „File" jest klasą, która wykona metodę „foreach" i podajemy plik do wykorzystania jako parametr. Potem zaczyna się blok. Wprowadź do pliku zapis:

   ```
   File.foreach("file_that_do_not_exost.txt) do |line|
    puts line.chomp
   end
   ```

   Wykonaj. Co jest efektem?

3. Przykład rozwiązuje problem nie istniejącego pliku. Otwórz plik read_from_file_-handle_exceptions.rb w sublime. Wykonaj plik.

4. Można sprawdzić, czy istnieje plik zanim wykona się z niego odczyt. Otwórz plik read_from_file.rb w sublime. Wykonaj plik.

5. Zapis do pliku też wykorzystuje blok. Otwórz plik write_to_file.rb w sublime. Wykonaj plik. Sprawdź co jest zapisane w pliku test1.txt

6. Zmienne środowiskowe są dostępne przez ENV. Otwórz plik environment_vars.rb w sublime. Wykonaj plik.

Pliki są automatycznie zamykane po wykonaniu kodu.

5.6 Zadanie samodzielne

Celem tego zadania jest wdrożenie mechanizmów sterowania programem przy użyciu Ruby: testowanie równości obiektów, testowanie wartości nil, testowanie wartości logicznych true / false.

Należy przepisać zadany kod zmieniając instrukcję if/else na case.

1. Wyedytuj w sublime plik module2_lesson1_formative.rb z podkatalogu Assignments/Lesson01-Assignment01-Case-Statement/student-start. Przepisz kod używając instrukcji „case".

2. Przeanalizuj wynik skryptu oryginalnego i zmienionego. Dlaczego 3 piersze testy zawodzą i co trzeba zrobić, by dawały wartość TRUE?

3. W tym samym podkatalogu snajduje się katalog „spec" z testami do zadania. Plik „.rspec" zawiera konfigurację niezbędną do wykonania testów jednostkowych.

4. Aby przeprowadzić testy na Twoim kodzie zainstaluj używając poleceń cmd:

```
gem install rspec
gem install rspec-its
```

5. Uruchom skryp z instrukcjami if/then. Co uzyskuje się w wyniku wykonania?

6. Uruchom polecenie „rspec", które wykona testy jednostkowe z katalogu spec. Uruchamianie z katalogu głównego projektu.

7. Zaimplementuj swoje zmiany. Uruchom ponownie testy.

5.7 Symbole, ciągi znaków

Pracuj z wskazanymi plikami z podkatalogu Lecture06-Strings.

Ciągi znaków w Ruby mogą być wskazane w dwojaki sposób: (

1. pojedyncze apostrofy - przedstawia wszystko tak jak jest (dosłownie)

2. cudzysłów - pozwala na stosowanie znaków specjalnych jak „
   n" czy „
   t". Pozwala na interpolację ciągów.

Zadanie

1. Otwórz plik strings.rb w sublime. Wykonaj plik. Metoda multipy pokazuje interpolację. Można korzystać z wartości zmiennych, ale tylko w ciągach otoczonych cudzysłwiem.

2. Przykłady metod dla ciągów znaków. Otwórz plik more_strings.rb w sublime. Wykonaj plik. Ciąg %Q wykorzystuje się jak cudzysłów, ale dla wielolinijkowych ciągów.

3. Odwiedź stronę z dokuementację dla klasy String `http://ruby-doc.org/core-2.2.0/String.html`

Sybmole w Ruby to np. „:foo-" to klasa bardziej zoptymalizowanych ciągów. Mają mniej metod niż klasa String. Mają predefiniowane ciągi. Symobole są unikalne i niezmienne. Można je zmienić na siąg metodą „to_s" lub ciąg w symbol metodą „to_sym". Np. nazwa metody jest sybolem. Nie chcemy jej zmieniać.

5.8 Tablice

Pracuj z wskazanymi plikami z podkatalogu Lecture07-Arrays.

Trzeba poznać sposoby na tworzenie, modyfikowanie i odczytywanie elementów z tablic. W ruby tablica to zbiór odniesień do obiektów. Tablice są automatycznie rozszerzalne.

Aby otrzymać wartość z tablicy stosuje się znaki „[]"

Indeksy w tablicach mogą być ujemne lub mogą być zakresem. Elementy tablicy mogą być różnych typów.

Polecenie

```
%w{string1 string 2}
```

służy do tworzenia tablicy zawierającej ciągi.

Modyfikowanie tablicy może odbywać się poleceniami:

1. Dopisywanie „push” lub „«”
2. Usuwanie „pop” lub „shift”
3. ustawienie „[]=(metoda)”

Możliwe jest losowanie elementu z tablicy poleceniem „sample”, sortowanie „sort!” lub odracanie „reverse!”. Metoda bez wykrzynika wymaga przypisania wyniku sortowania do nowej zminnej. W przypadku zastosowania wykrzyknika przy metodzie posrtowana wersja zostanie zapamiętana w starej zmiennej.

Ciekawe metody dla tablic:

1. each — pętla po elementach tablicy
2. select — filtrowanie tablicy
3. reject — filtrowanie tablicy
4. map — modyfikowanie każdego elemntu tablicy.

1. Otwórz plik arrays.rb w sublime. Wykonaj plik. Obserwuj tworzenie, indeksowanie, dostęp do elementów tablicy.
2. Otwórz plik arrays2.rb w sublime. Wykonaj plik. Obserwuj modyfikowanie tablic. Tablica jest jak kolejka LIFO lib FIFO w zależności czy metodą odjęcia jest pop czy shift.
3. Otwórz plik array_processing.rb w sublime. Wykonaj plik. Przegląd metod możliwych do wykorzystania na tablicach.
4. Odwiedź stronę z dokumentacją dla klasy Array `http://ruby-doc.org/core-2.2.0/Array.html`

5.9 Zakresy

Pracuj z wskazanymi plikami z podkatalogu Lecture08-Ranges.

Zakresy są używane do wyrażania naturalnej sekwencji np liczby od 1 do 20 czy litery od A do Z np. 1..20 liczby od 1 do 20; 'a'..'z', 1...20 to liczby od 1 do 19.

Można przekształcić zakres w tablicę wywołując metodę „to_a”. Często używane w instukcjach warunkowych.

Otwórz plik ranges.rb w sublime. Wykonaj plik.

5.10 Hash

Pracuj ze wskazanymi plikami z podkatalogu Lecture09-Hashes.

Hash to zbiór indeksów. Tworzy się je porzez „{}" lub Hash.new. Nazywa się je też tablicami asocjacyjnymi. W normalnej tablicy masz zbiór elementów i każdy z nich ma swój indeks. W hahses indeks nie jest liczba naturalną może być czymkolwiek, czyli każdy elementy tablicy asocjacyjnej zawiera element i skojarzony z nim indeks.

Dostęp przez operator „[]". Wartości przypisuje się przez operatory „=>" (tworzenie) i „[]" (kolejne przypisania).

1. Otwórz plik hash.rb w sublime. Wykonaj plik. Jak tworzy się tablicę asocjacyjną i modyfikuje dane.

2. Otwórz plik word_frequency.rb w sublime. Wykonaj plik. Metoda downcase sprawadza wszystkie wyrazy (gdy różnią się tylko wielkością liter) do tego samego stringu.

3. Otwórz plik more_hashesy.rb w sublime. Wykonaj plik. Kolejność elementów w tablicy asocjacyjnej jest zachowana (inaczej niż w innych językach). Jeżeli używa się symboli jako kluczy pisze się „symbol:". Nomalnie używa się zapisu „:symbol", jeżeli jednak myśliy o kluczu z tablicy, to piszemy „symbol:"

4. Otwórz plik block_and_hash_confiusion.rb w sublime. Wykonaj plik.

5. Odwiedź stronę z dokumentacją dla klasy Array `http://ruby-doc.org/core-2.2.2/Hash.html`

5.11 Klasy

Pracuj ze wskazanymi plikami z podkatalogu Lecture10-Classes.

W programowaniu obiektowym wyróżniamy:

- Klasy to rzeczy np. książki, samochody i są kontenerami na metody (zachowania, np. samochód jedzie)

- Obiekty to instancje/podstawienia klasy

- Obiekty zawierają zmienne instancji (określają stan/cechę obiektu, np. kolor=czerwony). W Ruby zmienna instancji zaczyna się od symboli @ np.:

  `@name`

  . Nie muszą być deklarowane. Zaistnieją przez ich pierwsze użycie. Są dostępne wszystkim metodom instancji danej klasy.

Tworzenie obiektu wykonuje się klauzulą:

`new`

która powoduje inicjalizację. Konstruktor — initialize.

Dostęp do zmiennych instancji nie może nastąpić spoza klasy, gdyż zmienne są prywatne. Można zatem stworzyć metodę, która z definicji jest publiczna. Tworzy się metody getter/setter. Przypisywanie można wykonać:

- attr_accessor — getter and setter w tym samym czasie
- attr_reader — getter only
- attr_writer — setter only

Obiekt nie jest zainicjalizowany podczas tworzenia.

„Self" jest metodą, która odwołuje się do obiektu

1. Otwórz plik classes.rb w sublime. Wykonaj plik. Definiowanie klasy i tworzenie obiektu.
2. Otwórz plik instance_vars.rb w sublime. Wykonaj plik. Pokazuje metodę getter i setter (dla age nie jest ustalony getter i dlatego nie wypisze wartości).
3. Otwórz plik attr_accessor.rb w sublime. Wykonaj plik. Zastosowanie getter i setter w pojedynczym wywołaniu. Problemy wynikające z zapisanego kodu: 1. Przy tworzeniu nowego obiektu nie nadaje się wartości zmiennym (rozwiązanie — konstruktor) i np. nie kontrolujemy wartości podawanego wieku.
4. Otwórz plik self.rb w sublime. Wykonaj plik. Kontrola wartości w zmiennej wiek. Użycie konstruktora. Metoda „self" użyta do przypisania lokalnej wartości zmiennej. Self — odwołanie do wartości obiektu.

5.12 Klasy — dziedziczenie

Pracuj ze wskazanymi plikami z podkatalogu Lecture11-MoreClasses.

Operator || działa w taki sposób, że ewaluuje lewą stronę, jeżeli jest „true" zwraca ją, jeżeli nie, to zwraca prawą stronę. np.:

```
@x = @x || 5
@x ||= 5
```

Zmienne klasy rozpoczynają się od „@@".

Klasa domyślnie dziedziczy z obiektu (symbol „<"). Obiekt dziedziczy domyślnie BasicObject.

1. Otwórz plik double_pipe.rb w sublime. Wykonaj plik. Użycie || do określenia domyślnej wartości dla zmiennej instancji.
2. Otwórz plik class_methods_and_variables.rb w sublime. Wykonaj plik. Pokazuje 3 sposoby definiowania metod w klasie (1) self. (2) « self (3) Poza klasą. **Ważne**: nie tworzy się instancji klasy!!! Wykorzystuje się metody z klasy bezpośrednio.
3. Otwórz plik inheritance.rb w sublime. Wykonaj plik. Klasa „Dog" dziedziczy po „Object" a klasa „SmallDog" dziedziczy po klasie „Dog".

5.13 Moduły — modules

Pracuj ze wskazanymi plikami z podkatalogu Lecture12-Modules.

Moduły to kontenery dla klas, metod i stałych lub innych modułów. Są jak klasy, ale nie można z nich stworzyć obiektu / instancji. „Class" dziedziczy po „Module" i dodaje metodę „new". Stosowane dla: (1) namespaces, (2) Mix-in.

Mixin pozwalają na dzielenie kodu pomiędzy klasami. Można stosować wbudowane moduły np.: Enumerable. Zapewnia metody: map, select, reject, detect, itp., używane przez klasę „Array", można wykorzystywać w swoich klasach, trzeba zaimplementować „each".

1. Otwórz plik module_namespace.rb w sublime. Wykonaj plik. Pokazuje, że stosowanie modułów nie powoduje konfliktu z nazwach klas.

2. Otwórz plik module_mixin.rb w sublime. Wykonaj plik. Pokazuje zastosowanie modułu do dzielenia kodu pomiędzy klasami.

3. Otwórz plik player.rb w sublime. Co jest w nim zapisane?

4. Otwórz plik team.rb w sublime. Co jest w nim zapisane?

5. Otwórz plik picks.rb w sublime. Wykonaj plik. Pokazuje zastosowanie „require_relative" oznacza dołączenie kodu z innego pliku .rb.

5.14 Scope — zasięg

Pracuj ze wskazanymi plikami z podkatalogu Lecture13-Scope.

1. Otwórz plik variables_scope.rb w sublime. Wykonaj plik. Pokazuje zakres zmiennej (zmienna poza klasą nie jest w niej dostępna).

2. Otwórz plik constant_scope.rb w sublime. Wykonaj plik. Pokazuje zasięg stałej, której nazwa zaczyna się z dużej litery (uwaga na rozróżnienie zmiennej i stałej). Można widzieć zewnętrzną stałą wewnątrz klasy. Manipulacja wewnątrze klasy nie zmienia wartości na zewnątrz.

3. Otwórz plik block_scope.rb w sublime. Wykonaj plik. Bloki dziedziczą zewnętrzny zasięg. Bloki są zamknięte. Zmienna wewnątrz bloku ma inny zasięg niż zmienna z klasy.

4. Otwórz plik block_local_scope.rb w sublime. Wykonaj plik. Zmienne wewnątrz bloku i jego parametry są lokalne i choć mają taką samą nazwę jak zmienne zewnętrzne nie naruszają ich wartości.

5.15 Kontrola dostępu do metod

Pracuj ze wskazanymi plikami z podkatalogu Lecture14-AccessControl.

Tworząc klasę należy uwzględnić jak wiele z niej ma być pokazane na zewnątrz. Chcemy zachować wewnętrzną reprezentację klasy jako prywatną. Nazywa się to enkapsulacją. Stosuje

się po to, że ni jest istotne dla świata wewnętrznego jak elementy/metody klasy są w szczegółach realizowane. Zmiana wewnątrz klasy nie wymaga zmian na zewnątrz. Ruby zapewnia trzy poziomy dostępu: public, private i protected.

Wszystkie metody w Ruby domyślnie są public. Można to zmienić podając inny dostęp poprzez słowo public, protect lub private.

Public — oznacza, że nie ma kontroli dostępu do metody i każdy z zewnątrz lub wewnątrz może z tej metody korzystać.

Protected — oznacz, że może być wywołany przez obiekty danej klasy lub jej podklas.

Private — nie mogą być wywołane z zewnątrz. Nawet przez self. w obrębie klasy. Jedynym wyjątekim jest setter.

1. Otwórz plik encapsulation.rb w sublime. Wykonaj plik. Przedstawia koncept enkapsulacji. Szczegóły obliczania „rating" są częścią wewnętrzną klasy. Metoda „rating" zwraca na razie wartość, ale można potem ją zmienić.

2. Otwórz plik specify_access.rb w sublime. Wykonaj plik. Dwa sposoby wskazywania dostępu do metod.

3. Otwórz plik private_access.rb w sublime. Wykonaj plik. Sposoby użycia self w metodzie private.

5.16 Zadania powtórkowe

5.16.1 Zadanie z tablic

Napisz polecenie, które wykorzysta metody działające w klasie array

1. reject

2. select

3. sort

4. reverse

Szczegóły zadania:

1. Wyedytuj w sublime plik module2_lesson2_formative.rb z podkatalogu Assignments/Lesson02-Assignment01-Collections/student-start.

2. Utwórz zgodnie z instrukcją polecenie, które na wylosowanych 23 elementach tabliy sypisze tylko te podzielne przez 3 i większe do 5000, a następnie je pososrtuj w koleności malejącej.

Szczegóły podane w kodzie.

Dokonaj samoewaluacji kodu uruchomiając testy jednostkowe polecenie „rspec". Uruchamianie z katalogu głównego projektu.

5.16.2 Zadanie z klas

Napisz klasę Person z atrybutami:

1. imię
2. nazwisko

Można mieść odstęp do każdej osoby. Zapewnij szukanie osoby przez nazwisko.

Szczegóły zadania:

1. Wyedytuj w sublime plik module2_lesson3_formative.rb z podkatalogu Assignments/Lesson03-Assignment01-Class/student-start.
2. Uzupełnij podając dwa atrybuty klasy imię i nazwisko i publiczne metody dla getter i setter.
3. klasa ma mieć atrybut people będący tablicą obiektów.
4. metodę inicjalizującą każdą instancję (konstruktor)
5. metodę „to_s" zwracająca sformatowany ciąg identyfikator osoby
6. metodę search działającą na nazwiskach.

Szczegóły podane w kodzie.

Dokonaj samoewaluacji kodu uruchomiając testy jednostkowe polecenie „rspec". Uruchamianie z katalogu głównego projektu.

6 Literatura

1. `http://learnrubythehardway.org/book`
2. `http://mislav.uniqpath.com/poignant-guide/book/` Something interesting
3. `http://pine.fm/LearnToProgram/` A very basic, ground-level tutorial for the beginner to Ruby. By Chris Pine.
4. `https://www.ruby-lang.org/en/` The official home page for the Ruby language.
5. `http://www.rubyist.net/~slagell/ruby/` A guide to learning Ruby (with code examples). By Matz, the creator of Ruby. Translated into English.