7 października 2016

Kurs języka Ruby

Lista 1.

Zadanie 1. Zaprogramuj

1. funkcję *slownie*, której argumentem jest liczba naturalna, a wynikiem string ze słownym zapisem liczby, np. wynikiem *slownie*(123) jest *"sto dwadzieścia trzy"*. Możesz przyjąć, że argument jest nie większy niż dziewięć tysięcy dziewięćset dziewięćdziesiąt dziewięć.

2. jednoargumentową funkcję `pascal` wypisującą na ekran kolejne wiersze trójkąta Pascala. Argumentem jest liczba wierszy trójkąta do wypisania. Na przykład po wywołaniu `pascal(3)` powinno się pojawić

   ```
   1
   1 1
   1 2 1
   ```

3. jednoargumentową funkcję `wielkaliczba`, która dla zadanego argumentu (liczby naturalnej) wypisuje na ekran tę liczbę ale powiększoną, np. po wywołaniu `wielkaliczba(123)` powinno się pojawić

   ```
     x     x      xxxx
    xx   x   x   x   xx
   x x      x       xx
     x    x      x   xx
     x   xxxxx    xxxx
   ```

   Oczywiście, kształt cyfr może się różnić od tych w przykładzie;

4. jednoargumentową funkcję `podzielniki(n)`, gdzie n jest liczba naturalną, zwracającą tablicę dzielników pierwszych liczby n bez powtórzeń, np. `podzielniki(1025)` powinno zwrócić `[5, 41]`. Co prawda 5 dwukrotnie dzieli 1025, jednak w wyniku nie powinno być powtórzeń;

5. dwuargumentową funkcję `godzina`, której argumentami są dwie liczby naturalne oznaczające godzinę i minutę. Wynikiem działania funkcji ma być string zwracający tę godzinę w potocznej formie. Przykładowo `godzina(12,45)` powinna zwrócić *"za kwadrans pierwsza"*.

Uwaga: można przyjąć, że argumenty są zawsze poprawne i nie jest konieczna kontrola argumentów.
Najwygodniej jest umieścić wszystkie rozwiązania w jednym pliku, a na końcu umieścić kilka testów prezentujących możliwości zaimplementowanych funkcji. Za każde zadań można otrzymać 2 punkty, jednak za całą listę nie można otrzymać więcej niż 4 punkty (a więc do oceny można przedstawić co najwyżej 2 funkcje). Termin: zajęcia w przyszłym tygodniu.

Marcin Młotkowski

14 października 2016

Kurs języka Ruby

Lista 2.

Zadanie 1. Zaprogramuj interaktywny (tj. obsługa jest z poziomu konsoli) kalendarz do którego można wprowadzać spotkania/zadania: dzień, godzina i treść. Kalendarz powinien mieć możliwość interaktywnego:

- wprowadzania danych wraz z sygnalizacją konfliktów;
- wypisywania uporządkowanych (w czasie) spotkań/zadań; można przyjąć że wypisywane są wszystkie zadania.

Aktualizacja: można przyjąć, że każde spotkanie trwa jedną godzinę.

Zadanie 2. Zaprogramuj interaktywny (tj. dane są wprowadzone bezpośrednio z konsoli) notatnik z osobami i numerami telefonów. Przyjmij, że dla każdej osoby są pamiętane przynajmniej następujące dane:

- imię/nick;
- numer telefonu (dowolna postać: Fixnum bądź String);
- lista grup do jakich zaliczona jest dana osoba, np. `[rodzina, ulubiona]`.

Program powinien implementować przynajmniej następujące operacje:

- dodawanie osoby wraz z numerem telefonu i listą grup;
- wyszukiwanie po imieniu;
- znalezienie wszystkich grup;
- wyszukanie wszystkich osób należących do danej grupy.

Zadanie 3. Zaproponuj jakąś reprezentację grafu nieskierowanego za pomocą słowników. Przyjmij, że wierzchołki są etykietowane elementami typu `String`, oraz że różne wierzchołki mają różne etykiety[1]. Napisz procedurę `sciezka(graf, a,b)`, której argumentami są: graf i etykiety, a wynikiem tablica etykiet z `a` do `b` lub lista pusta jeśli droga nie istnieje.

Zadanie 4. Wymyśl reprezentację grafu takiego jak w poprzednim zadaniu. Zaimplementuj sumowanie dwóch grafów zgodnie z regułami:

- w sumie są wierzchołki z obydwu grafów, przy czym wierzchołki o identycznych etykietach traktujemy jak te same wierzchołki;
- w sumie grafów są krawędzie z obydwu grafów.

Sumowanie grafów powinno być implementowane przez dwie funkcje: `suma(g1, g2)` i `suma!(g1, g2)` które gdzie pierwsza zwraca nowy graf a druga modyfikuje pierwszy argument.

Każde zadanie jest warte 4 punkty, na zajęciach oddaje się dwa zadania. W zadaniach można dla wygody w kodzie źródłowym utworzyć sobie dane początkowe.

Marcin Młotkowski

[1] Taki graf może służyć do reprezentowania np. schematu komunikacji miejskiej

21 października 2016

Kurs języka Ruby

Lista 3.

Zadanie 1. (3 pkt) Bloki z jednym parametrem można traktować jak definicje jednoargumentowych funkcji. Korzystając z tej obserwacji zaprogramuj dwie procedury. Pierwsza z nich `calka(a,b,&b)` powinna obliczać numerycznie całkę oznaczoną na przedziale $[a, b]$ funkcji zadanej jako blok. Dokładność obliczeń może być ustalona. Druga funkcja to `wykres(a, b, &blok)`, która za pomocą znaków ASCII naszkicuje wykres funkcji danej jako blok. Można przyjąć arbitralny rozmiar terminala.

Implementacje poniższych funkcji powinne być w postaci jednego wyrażenia. Jest to możliwe używając tylko zakresów, operacji na tablicach i bloków. W przypadku bardzo długich wyrażeń akceptowane będzie podzielenie rozwiązania na podwyrażenia.

Zadanie 2. (2 pkt) Napisz jednoargumentową funkcję `pierwsza(n)`, która zwraca tablicę liczb pierwszych nie większych niż n.

Zadanie 3. (2 pkt) Napisz jednoargumentową funkcję `doskonale(n)`, która zwraca tablicę liczb doskonałych nie większych niż n, na przykład

```
doskonale(1000)
==> [6, 28, 496, 8128]
```

Zadanie 4. (2 pkt) Napisz jednoargumentową funkcję `rozklad(n)` która oblicza rozkład liczby n na czynniki pierwsze i zwraca jako wynik tablicę tablic $[[p_1, w_1], [p_2, w_2], \ldots, [p_k, w_k]]$ taką, że
$n = p_1^{w_1} * p_2^{w_2} * \ldots * p_k^{w_k}$ oraz $p_1, \ldots, p_k$ są różnymi liczbami pierwszymi. Na przykład

```
rozklad(756)
==> [[2, 2], [3, 3], [7, 1]]
```

Zadanie 5. (1 pkt) Napisz jednoargumentową funkcję `zaprzyjaznione(n)`, która zwraca tablicę par liczb zaprzyjaźnionych nie większych niż n, na przykład

```
zaprzyjaznione(1300)
==> [[220, 284], [1184, 1210]]
```

Dodatkowe wyjaśnienia można znaleźć w polskiej Wikipedii.

Za rozwiązanie powyższych zadań można uzyskać co najwyżej 4 pkt.

Marcin Młotkowski

4 listopada 2016

Kurs języka Ruby

Lista 4.

Zadanie 1. Typowym zadaniem na zajęciach z programowania jest implementacja wypożyczalni (płyt, samochodów etc) czy biblioteki. Zazwyczaj funkcjonalności czy schematy takich programów są bardzo podobne. Zdefiniuj więc moduł wypożyczeń ***Wypozyczenia*** (lub kilka odpowiednich modułów) implementujący odpowiednie schematy wypożyczenia i zwracania przedmiotów do wypożyczalni. Moduł ten powinien być tak napisany, aby łatwo było go *wmiksować* do odpowiednich klas by utworzyć prawdziwą wypożyczalnię.

Korzystając z tego modułu zaimplementuj klasę ***Biblioteka*** korzystającą z ***Wypozyczenia*** implementującą wypożyczalnię książek.

Zadbaj o personalizację komunikatów, tj. aby komunikaty były postaci „wypożyczono książkę ...” a nie „wypożyczono obiekt #<Ksiazka:0x7fd04e280f20>”, o ile oczywiście przykładową klasą będzie biblioteka.

Zadanie 2. Ważnym choć czasem niedocenianym elementem rozwijania oprogramowania jest testowanie. Obiekty można na przykład testować dodając odpowiednie metody testujące. Dla wygody można przyjąć, że nazwy metod testujących zaczynają się od `test_`.

Rozszerz podany na wykładzie mix-in ***Debug*** o procedurę `check`, która wyszukuje w klasie wszystkie metody zaczynające się na `test_` i wykonująca je. Przyjmij, że metody `test_*` zwracają wynik testu jako obiekt klasy ***String***.

Zadanie 3. Zaprogramuj klasę ***DrzewoBinarne*** implementujące drzewo poszukiwań binarnych wraz z operacjami `wstaw`, `istnieje?` i `usun`. Przyjmij, że elementy drzewa są obiektami klasy ***Element***. Podaj przykład wykorzystania tych klas. Zwykle do implementacji takiej klasy konieczne są operatory $>$, $<$ czy $==$, jednak zamiast tego wygodniej jest zaimplementować tylko metodę $<=>$ i dołączyć *mix-in* ***Comparable***.

Zaprogramuj ***StringBT*** jako podklasę ***DrzewoBinarne*** przechowującą obiekty klasy ***String***.

Każde zadanie jest warte 3 punkty. Na pracowni oddaje się jedno zadanie.

Marcin Młotkowski

18 listopada 2016

Kurs języka Ruby

Lista 5.

W poniższych zadaniach przyjmij, że parametry takie jak adresy stron czy głębokość wyszukiwania mogą być podawane jako parametry wywołania programu.

Zadanie 1. Zaprogramuj pakiet funkcji przeglądających istniejące serwisy WWW. Pakiet powinien zawierać ogólną funkcję przeglądania stron `przeglad(start_page, depth, block)`, gdzie `start_page` to adres strony startowej, `depth` to głębokość z jaką należy przeglądać serwis, a `block` to jednoargumentowy blok o argumencie typu ***String***. Funkcja ta powinna przeglądać strony serwisu od podanej strony, i dla każdej z nich wykonać instrukcje zawarte w bloku. Zaprogramuj dwie funkcje:

- `page_weight(page)`, która oblicza liczbę elementów wpływających na czas ściągania i renderowania strony, takich jak obrazki czy aplety;
- `page_summary(page)`, która dla każdej strony wypisuje informację o stronie uzyskaną z nagłówka `<head>`, tj. tytuł, opis, autor, słowa kluczowe etc.

Zadanie 2. Napisz własną wyszukiwarkę, która przegląda wybrany serwis internetowy i zapamiętuje wystąpienia poszczególnych słów, oraz umożliwia wyszukiwanie słow zadanych jako wyrażenie. Kod wyszukiwarki powinien mieć postać modułu zawierającego funkcje:

- `index(start_page, depth)` która przegląda od podanej strony oraz indeksuje słowa ze strony;
- `search(reg_exp)`, która podaje listę stron na których występują słowa pasujące do `reg_exp`, oczywiście korzystając wyłącznie z zebranej wcześniej informacji.

Zadanie 3. Zaprogramuj funkcję `distance(page_a, page_b)`, która oblicza odległość od strony o url'u `page_a` do strony o url'u `page_b`, przy czym odległość jest rozumiana jako liczba kliknięć w kolejne odnośniki. Oczywiście można przyjąć jakieś arbitralne ograniczenie na czas przeszukiwania.

Zadanie 4. Zaprogramuj pakiet służący do monitorowania wybranych stron i informujący o zmianie treści strony. Na początku podaje się listę stron jakie mają być monitorowane, następnie program oblicza sumę kontrolną np. md5. Następnie co jakiś czas wskazane strony są ponownie odczytywane, i jeżeli nastąpiła zmiana wysyłany jest komunikat na konsolę. Uzupełnij program o możliwość zapisywania stanu sesji (tj. stron oraz sum kontrolnych) i jej odtwarzania. Do tego przydatny będzie moduł YAML.

Choć treść powyższe zadań sugeruje rozwiązanie w postaci modułu, proszę zaprogramowane funkcje "opakować" w klasy i obiekty. Każde z tych zadań jest warte 4 punkty, na zajęciach proszę oddać jedno zadanie.

Marcin Młotkowski

25 listopada 2016

Kurs języka Ruby

Lista 6.

Do poniższych zadań proszę dołączyć interaktywny miniinterfejs do obsługi poniższych programów; wystarczy jeśli się skorzysta z `gets` i `puts`. Poniższe zadania powinne implementować przeglądanie, wyszukiwanie, dodawanie i usuwanie wpisów.

Zadanie 1. Zaprogramuj własny organizator swojego czasu zawierający planowane spotkania (od-do), sprawy do załatwienia (do czasu), wraz z opcją przypominania. Dane niech będą przechowywane na dysku, np. korzystając z dbm'a, SQL'a czy YAML'a.

Zadanie 2. Napisz program, który przechowuje w swojej lokalnej bazie danych informacje o posiadanych płytach z muzyką (identyfikator płyty, lista utworów i autorzy) wraz z informacjami o wypożyczeniu płyty znajomym.

Zadanie 3. Zaprogramuj własny notatnik z kontaktami do znajomych zawierający ich numery telefonów, adresy email czy nicka we wspólnym portalu społecznościowym. Dane niech będą przechowywane w bazie danych (typu dbm, pickle czy SQLite).

Na zajęcia należy wykonać jedno z tych zadań. Każde zadanie jest warte 3 punkty. Proszę zadbać też o komentarze w swoich programach.

Marcin Młotkowski

5 grudnia 2016

Kurs języka Ruby

Lista 7.

Zadanie 1. Zaprogramuj serwer logów (drb), który będzie przechowywał nadesłane komunikaty w bazie danych wraz z czasem ich otrzymania. Serwer powinien implementować funkcję `save(prg_id, msg)`, gdzie `prg_id` jest identyfikatorem programu (serwer może zbierać logi z różnych aplikacji), a `msg` oznacza obiekt klasy ***String***. Dodatkowo zaimplementuj na serwerze metodę `raport(od, do, prg_id, re)`, gdzie `od` i `do` są obiektami klasy ***Time*** definiującymi zakres wyszukiwania, `prg_id` jest identyfikatorem programu, a `re` wyrażeniem regularnym. Funkcja powinna zwracać obiekt ***String***, będący html'em. Sposób przechowywania (dbm, sqlite3 etc.) jest dowolny.

Zadanie 2. Zaimplementuj repozytorium do przechowywania obiektów, tj. serwer implementujący metody `store(obj, id)`, `restore(id)` i `delete(id)`, gdzie `id` jest dowolnym identyfikatorem obiektu.

Zaprogramuj również metody: `stan`, która zwraca html (jako string) z informacją o zapisanych obiektach, tj. klasę obiektu wraz ze stanem jego pól, oraz wyszukiwarkę obiektów implementujących daną jako argument listę metod.

Zadanie 3. Serwer `drb` może być wykorzystywany jako narzędzie do zdalnego monitorowania i zarządzania komputerami, tj. na każdym komputerze klienckim jest uruchomiony serwer z metodami, które wywołują polecenia systemowe sprawdzające podstawowe wartości, takie jak sprawdzenie obciążenia procesora czy ilość wolnego miejsca na dysku. Zaprogramuj taki serwer wraz z klientem, który mając daną listę komputerów będzie je odpytywał co jakiś czas i raportował ich stan. Zbadaj, jaki musi być ustawiony poziom zmiennej `$SAFE`.

Na zajęcia należy wykonać jedno z tych zadań. Każde zadanie jest warte 3 punkty. Proszę zadbać też o komentarze w swoich programach.

Marcin Młotkowski

19 grudnia 2016

Kurs języka Ruby

Lista 8.

Zadanie polega na uzupełnieniu wcześniej zaprogramowanych zadań o wątki i testy jednostkowe. Dobrym kandydatem są np. zadania z listy 5. W przypadku testów można wybrać inne zadanie niż wybrane dla wątków. Wymagane jest stworzenie przynajmniej 3 metod z testami.
Za to zadanie można otrzymać do 3 punktów.

Marcin Młotkowski

5 stycznia 2017

Kurs języka Ruby

Lista 9.

Zadanie 1. Zaprogramuj program rysujący wykresy kilku ustalonych funkcji. Przyjmij, że wskazanie funkcji następuje w menu, natomiast wartości *od do* przedziału rysowania są określane w kontrolkach ***TkEntry***.

Zadanie 2. Zaprogramuj następującą prostą grę: na rysunku jest armata, która ma regulowany kąt wystrzału i prędkość początkową pocisku, oraz cel (odległość między armatą i celem może być losowa). Kąt wystrzału oraz prędkość pocisku powinna być zadawana przez użytkownika, np. za pomocą kontrolek ***TkEntry***. Zadanie polega na takim wybraniu kąta i prędkości, aby pocisk trafił w cel. Korzystając z prostych praw fizyki narysuj tor pocisku oraz oblicz, czy pocisk trafił w cel.

Zadanie 3. Bardzo ładnymi figurami geometrycznymi są fraktale. Sporo materiałów o nich można znaleźć w internecie (są również książki o fraktalach w naszej bibliotece). Zadanie polega na zaprogramowaniu kilku fraktali. Fraktale zwykle mają parametry, które powinny być podawane np. poprzez kontrolki ***TkEntry***.

Zadanie 4. Zaprogramuj grę *kółko i krzyżyk* (można przyjąć, że dla planszy 3 x 3), gdzie jednym z graczy jest komputer, a drugim człowiek. Plansza powinna być zaimplementowana w Tk. Początkowy gracz jest wybierany z menu.

Zadanie 5. Zaprogramuj interfejs graficzny (np. **tk**) do jednego z wcześniej implementowanych zadań. Na przykład **lista 2 zad. 1 i 2, lista 4 zad. 1:** interfejs użytkownika; **lista 2 zad.3 i 4:** rysowanie grafów, z wyróżnioną kolorem znalezioną ścieżką (zad. 3).

To zadanie jest warte 3 pkt.

To już jest ostatnia lista zadań.

Marcin Młotkowski