TDP007 - Tenta

2020-08-17

Regler

- All kod efterrättas på denna tenta
- Vid toalettbesök ska vakt informeras.
- All form av kontakt mellan studenter under tentamens gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av assistent, vakt eller examinator under tentamens gång.
- Frågor om specifika uppgifter eller om tentamen i stort ska ställas via tentasystemets kommunikationsklient.
- Systemfrågor kan ställas till assistent via Zoom
- Endast uppgifter inskickade före tentamenstidens slut rättas.

Hjälpmedel	En Rubybok (exempelvis the pickaxe Ett A4-ark med egna anteckningar Tillgång till webresurser: ruby-docs, rubular och tidig version av kursboken

Information

Betygssättning vid tentamen

Tentamen består av ett antal uppgifter på varierande nivå. Uppgifter som uppfyller specifikationen samt följer god sed och konventioner krävs för maxpoäng på en uppgift. Avvikelser från ovanstående ger avdrag.

Tentan består av uppgifter (några indelade i deluppgifter) som totalt kan ge 50 poäng.

Ruby-docs

På tentan har du tillgång till referenssidorna på `https://ruby-doc.org/` (både core och std-lib finns tillgänglig) via webbläsaren.

Rubular

På tentan har du tillgång till sidan `https://rubular.com/`.

Givna filer

Eventuella givna filer finns länkade från kurssidan.

Avslutning

När du skickat in alla uppgifter och känner dig färdig kan du stänga tetaklienten, lämna Zoom och stänga thinlinc.

Uppgift 1 - Regexp (7+5p)

Praktisk (7p)

ALL-TIME ETC MEDAL TABLE

Country	🏆	🏆	🏆	Σ	№
Poland	9	8	2	19	22
Germany	6	5	2	13	22
Italy	2	2	4	8	21
Sweden	2	1	1	4	20
Denmark	1	2	2	5	20
England	1	0	3	4	19
United States	1	0	2	3	16
France	0	1	2	3	22
Greece	0	1	1	2	18
Spain	0	1	1	2	18
Austria	0	1	0	1	20
Australia	0	0	1	1	11
Russia	0	0	1	1	22

Overall

I den givna filen `etc.html` finns en tabell över länder som tagit medaljer över åren i tävlingen European Team Championship där man tävlar i spelet Warhammer. Ditt jobb är att läsa in denna tabell med hjälp av regexp och spara den i en rubyhash där landet är nyckeln och värdet är en array av de siffror som hör till landet. Tabellen innehåller från vänster till höger: landets namn, guldmedaljer, silvermedaljer, bronsmedaljer, totalt antal medaljer och antal gånger landet deltagit.

Följande är ett exempel på utskriften av en korrekt databehållare:

```
{"Poland"=>[9, 8, 2, 19, 22],
 "Germany"=>[6, 5, 2, 13, 22],
 ...
 "Russia"=>[0, 0, 1, 1, 22]}
```

Formateringen är inte viktig vid utskriften men jag rekommenderar att du skriver ut `Hashen` på följande sätt för läsbarhetens skull:

```
require 'pp'
pp my_hash
```

Ickefunktionella krav för full poäng:

- Regepx används för att hitta länderna och relevanta siffror.
- Varje data särskiljs med matchgroup
- Fungerar för tabell med godtyckligt antal länder

Teoretisk (5p)

Besvara denna fråga i en separat textfil och lämna in på samma uppgift. Förklara skillnaden mellan klassfunktioner(statiska funktioner) och medlemsfunktioner(metoder). Ge ett exempel på ett problem som lämpligen löses med en klassfunktion respektive en medlemsfunktion.

Uppgift 2 - XML (10+3p)

Praktisk (9p)

I filen `adventure.xml` hittar du ett antal platser som ingår i ett litet textbaserat äventyrsspel. Varje plats har ett `id` som är unikt och en beskrivning. Utöver det har också platserna anslutningar till andra platser. Varje anslutning har ett väderstreck och det `id` vars plats man kommer till om man färdas i det väderstrecket.

Ditt jobb är att skriva ett program som låter användaren färdas mellan olika platser genom att mata in väderstreck. När användaren kommer till en plats ska platsens beskrivning skrivas ut enligt formatet i körexemplet. Sedan ska alla anslutningarna skrivas ut enligt körexemplet. Efter det får användaren frågan om vart de vill gå nu och matar sedan in ett väderstreck.

Användaren börjar alltid sin resa vid vattendraget. Om användaren matar in ett inkorrekt väderstreck ska användaren få en ny chans att mata in.

Körexempel:

```
You see a: A babbling brooke

To the west you see a path to the clearing
Which way do you go?: north

To the west you see a path to the clearing
Which way do you go?: west

You see a: A lush forrest clearing
To the east you see a path to the riverbank
To the north you see a path to the cave
Which way do you go?: north

You see a: A dark cave, with a light at the end
To the south you see a path to the clearing
To the west you see a path to the town
Which way do you go?: west

You see a: A nice little town
To the east you see a path to the cave
Which way do you go?: east

You see a: A dark cave, with a light at the end
To the south you see a path to the clearing
To the west you see a path to the town
Which way do you go?:
```

Teoretisk (3p)

Om man läser in ett XML-dokument med hjälp av strömparsning (SAX), vad gör då lyssnarklassen? Beskriv kortfattat vad lyssnarklassen gör och vilken roll dess standardmetoder har.

Uppgift 3 - parser (13p)

Praktisk (13p)

I filen `uppgift3.rb` hittar du det bekanta diceroller-språket. Ditt jobb är att utöka språket så att användaren kan definiera sina egna tärningar och sedan rulla dessa. Användaren definierar en tärning genom att skriva nyckelordet `def` följt av namnet på den nya tärningen följt av ett godtyckligt antal tal som separeras med mellanslag. Den nya tärningen kan sedan slås genom att ange dess namn, när tärningen slås så slumpas ett av de angivna talen.

Körexempel:

```
[diceroller] def even 2 4 6 8
=> [2, 4, 6, 8]
[diceroller] even
=> 8
[diceroller] even
=> 2
[diceroller] 2+d6+even
=> 5
[diceroller] 2+d6+even
=> 13
[diceroller] def uneven 1 3 5 7 9
=> [1, 3, 5, 7, 9]
[diceroller] even+uneven+1
=> 14
```

Om användaren skulle ange namnet `d` eller `def` på en tärning är detta odefinierat beteende och du behöver inte hantera det fallet. Ett tips är att spara undan de angivna talen i en array kopplade till namnet på tärningen, förslagsvis i en hashtabell. När tärningen sedan slås kan man slumpa ett tal ur aktuell array med medlemsfunktionen `sample` som finns för alla arrayer:

```
irb(main):001:0> [1, 2, 3].sample
=> 2
```

Uppgift 4 - DSL (12p)

Praktisk (12p)

I files `games.rb` finns ett antal relationer. Den första relationen beskriver att spelet `Descent` tillhör kategorin `Boardgame`. Descent tillhör också 2 genrer och finns i 2 språk.

Ditt jobb är att modifiera den givna koden i `uppgift4.rb` så att filer av denna typ kan läsas in och att man sedan kan söka i informationen med hjälp av relationerna. När du läst in datan bör du spara denna i en lämplig databehållare för att underlätta sökningen.

Skapa sedan medlemsfunktionen `search` som tar 3 parametrar. Den första parametern är typen av relation, den andra är spelet och den tredje är informationen som spelet knyts till genom denna relation. Den andra eller tredje parametern kan anges som `nil` vilket fungerar som ett wildcard (se exemplet). Alla relationer som matchar sökningen ska returneras av funktionen i ett lämpligt format, om inga matchingar finns kan en tom array returneras.

I exemplet nedan anropas `find` tre gånger. Den första gången slår man upp om fotboll är en sport. Den andra gången slår man upp alla språk som spelet Descent är kopplat till. Den tredje gången slår man upp alla spel som är kopplade till genren Rpg. Körexempel:

```
storage = Storage.load("games.rb")
print storage.find('category', "Football", "Sport")
puts " "
print storage.find('language', "Descent", nil)
puts " "
print storage.find('genre', nil, "Rpg")
puts " "
```

```
[["Football", "Sport"]]
[["Descent", "English"], ["Descent", "German"]]
[["Descent", "Rpg"], ["CoC", "Rpg"], ["Warcraft", "Rpg"]]
```

Programmet ska ta hänsyn till att nya relationer och nya typer av relationer kan läggas till fritt. Exempelvis kanske man vill lägga till relationer som beskriver antalet spelare eller åldersgränser i framtiden.