TDP007 - Tenta

2020-03-24

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)

Kapten Janeway har tröttnat på hur svårt det är att hitta tider som passar alla i hennes studiegrupp under pandemin. Hon har därför skapat ett delat textdokument där alla deltagare kan fylla i vilka tider som passar dem. Du hittar denna information i den givna filen `schedule.txt`. Varje person skriver en kort kommentar om sin situation, sedan sitt namn och sedan om de kan närvara under varje tid. Det är dock jobbigt att läsa detta och manuellt hitta tider. Janeway har därför börjat skriva ett datorprogram som automatiskt ska välja den optimala tiden. Men Hon har aldrig jobbat med Regexp och har därför svårt att få in filen i en lämplig datorstruktur i sitt program.

Ditt jobb är att med hjälp av Regexp läsa in den relevanta informationen ur filen och sedan spara detta i en Array av Arrayer. Varje inre Array ska bestå av 5 element där det första elementet är personens namn och de kvarvarande 4 elementen är informationen om deras möjlighet att närvara. Kommentaren i början av raden kastar vi bort. Skriv sedan ut denna Array av Arrayer:

```
[
  ["Alex", "---", "---", "Yes", "Yes"],
  ["Petra", "Yes", "Yes", "---", "---"],
  ["Sam", "Yes", "Yes", "Yes", "Yes"],
  ["Erik", "Maybe", "Maybe", "Yes", "Yes"]
]
```

Formateringen av utskriften är irrelevant. Den presenteras radbruten och indragen för att vara läsbar i uppgiften.

För full poäng:

- Det ska gå för deltagarna att skriva andra saker än bara Yes, `---` och Maybe som svar på varje tidsblock i kolumnerna. Du kan dock anta att det inte kommer tillkomma några whitespaces i dessa kolumner.
- Kommentaren i början av raden kan vara godtycklig i längd och antal ord
- Problemet ska i huvudsak lösas med Regexp.

Teoretisk (5p)

Besvara dessa frågor i en separat textfil och lämna in på samma uppgift.

- I detta dokument finns 5 kolumner intressant data (namn och 4 tider). Beskriv de ändringar du skulle behöva göra för att hantera ett dokument där det finns 2 eller fler(godtyckligt många fler) kolumner intressant data för varje person.
- I denna lösning representerar varje rad i dokumentet en person och de tider de kan närvara. Om dokumentet istället representerade en person med kolumner(första raden innehåller alltså allas namn, andra raden alla personers svar på första tiden, etc.) beskriv hur du behöver ändra din lösning för att hantera detta.

Uppgift 2 - XML (9+4p)

Praktisk (9p)

Sisco har ett antal rollspelskampanjer igång på sin rymdstation. Han är en trevlig prick så han försöker låta alla som vill spela utifrån sina egna förutsättningar med hänsyn till tid. Därför har han ett system där spelarna kan skriva upp sig inför kommande veckor. Den givna filen `schedule.xml` representerar datan inhämtad från detta system. Han har dock lite svårt att läsa in och behandla denna data. Ditt jobb är att implementera de funktioner som anropas i den givna filen `uppgift2.rb`.

`active_players` är en funktion som tar ett namn på en fil och en månad som parametrar. Båda är av typen string. Funktionen skriver sedan ut namnet på alla spelare som under någon dag den givna månaden skrivit upp sig och angett att de är tillgängliga (`avail='yes'`). Formatet på utskriften spelar ingen roll.

`list_day` är en funktion som tar ett namn på en fil och en dag som parametrar. Båda är av typen string. Funktionen skriver sedan ut alla dagar som parametern specificerade på formatet du kan se i körexemplet, formateringen av utskriften spelar roll.

Körexempel:

```
Active players in August:
Alex
Roxane
Joan
Sam
Roy

List all Mondays:
Monday week 37:
name: Alex, available: yes
name: Roxane, available: yes
name: Roy, available: no

Monday week 38:
name: Alex, available: no
name: Roxane, available: yes
name: Roy, available: yes

Monday week 39:
name: Alex, available: yes
name: Roxane, available: yes
name: Tina, available: yes
```

Teoretisk (4p)

Du löste den ovanstående uppgiften antingen med en SAX- eller DOM-parser. Beskriv hur du skulle löst uppgiften med den andra tekniken. Ditt jobb här är att övertyga oss om att du hade kunnat lösa problemet med den andra tekniken.

Uppgift 3 - Parser (9+4p)

Praktisk (9p)

Picard gillar att spela rollspel men hatar tärningar(inte helt realistiskt, jag vet). Han har jobbat lite med programspråk tidigare och har en lexer och parser redo för ett språk som heter `DiceRoller`. Detta fungerar fint och låter honom skriva aritmetiska uttryck som innefattar tärningar. Han vill nu bygga ut det språket så man också kan skriva jämförelser, på detta sätt kommer hans hobby bli ännu njutbarare.

Ditt jobb är att utöka språket i den givna filen `uppgift3.rb` med det som krävs för att språket ska kunna hantera större än(`>`), mindre än(`<`), och(`&`), eller(`|`). Dessa ska fungera så långt som möjligt med korrekt associativitet och prioritet. Aritmetiska uttryck bör ha samma prioritet som de har nu i språket, efter det bör och komma. Efter det bör och komma.

Hur programmet ska fungera vid uttryck som `1&2`(där det inte finns ett sanningsvärde på båda sidorna) är inte definierat och behöver inte hanteras.

Körexempel:

```
[diceroller] 15 > 2d6 + 2
=> true
[diceroller] 15 > 3d6 + 10
=> false
[diceroller] 1 < 2 & 3 < 4
=> true
[diceroller] 1 < 1d6 | 3 < 1d6
=> true
[diceroller] 15 > (2+4)*3
=> false
[diceroller]
```

Teoretisk (4p)

Efter din utökning i uppgiften ovan kan nu språket hantera jämförelser av tal, tärningsslag och aritmetiska uttryck. Du kan också skriva uttryck som sätter ihop dessa jämförelser med `&` samt `|`. Det finns inget stöd för att hantera råa sanningsvärden, man kan alltså inte skriva `TRUE` eller `FALSE`. Beskriv vilka ändringar du behöver göra för att tillåta användning av detta.

Uppgift 4 - DSL (8+4p)

Praktisk (9p)

LaForge håller på att skriva en server för en av sina hemsidor. Han vill dock enkelt kunna köra igång projektet i både test och produktionsmiljö och har därför skapat ett enkelt sätt att skriva en config-fil på. Du kan se hur denna ser ut i den givna filen `dsl.rb`. Tyvärr finns det ingen färdig modul som hantera inläsning av filer som skrivs i just detta format.

Ditt jobb är därför att implementera ett litet DSL för att hantera denna sorts filer. Filen är strukturerad så att man inleder en konfiguration med `config "namn på konfigurationen"`. Allting som kommer under den raden fram till `config` eventuellt förekommer igen tillhör den konfigurationen. I `dsl.rb` finns 2 konfiguarioner. Du ska med hjälp av method_missing skapa en klass. Denna klass ska ha en funktion `read` som tar in ett filnamn och ett namn på en konfiguration. funktionen ska sedan läsa in och spara all data som hör till den konfigurationen i en hash(se körexemplet). I den givna filen `uppgift4.rb` finns en påbörjad lösning på problemet.

Det är ok att lägga till andra medlemsfunktioner för att lösa problemet men man ska kunna stoppa in godtyckliga nyckel-värde par i konfigurationsfilen.

Körexempel:

```
Test environment config:
{
  "port" : 9999,
  "is_test" : true,
  "require_password" : false
}

Production environment config:
{
  "port" : 64,
  "is_test" : false,
  "require_password" : true,
  "app_name" : "my_application"
}
```

Teoretisk (4p)

1. I denna uppgift har du använt dig av en klassfunktion som heter `read` för att lösa problemet. Denna typ av funktion är också kända som statiska funktioner i exempelvis c++. Redogör för hur en klassfunktion skiljer sig från en medlemsfunktion
2. Vad är ett domänspecifikt språk? Hur skiljer sig det från andra språk du stött på? Var går gränsen mellan dsl och gpl?