Utvecklarblogg - Seminarie 2

Uppgift 1.

Tolkning av uppgiften:

- Vi antar att det alltid kommer finnas en <pre> och </pre> tagg
- Vi antar att kommer vara samma antal kolumner med samma typ av innehåll i kolumnerna, bokstäver/siffror
- Vi antar att header raden kommer vara samma i varje tabell
- Vi antar att skillnaden i sorteringen avser absoluta värden
- Vi antar att endast de kolumner som har tomma rader i ursprungs exemplet kan ha tomma platser. dvs kolumnerna "HDDay", "1HrP" och "WxType".
- Vi antar att de kolumner som har decimaler alltid kommer ha decimaler tex "0.00"
- Vi antar att man vill läsa in filen på ett sätt så man kan komma åt varje datafält, dvs en 2d array där varje subarray motsvarar en rad och varje element i den ett ord/tal ifrån tabellen: [["Arsenal", 38],["Liverpool",38]]
- Vi antar att det ska skrivas ut på samma sätt som i textfilen, antal kolumner/rader/innehåll etc, bara i en annan ordning.

Förklaring funktioner:

load_url(url_string)
tar in en sträng som motsvarar en url man vill läsa in. Använder sig av Nokogiri och URI för att läsa in texten ifrån urlen. xpath funktionen extraherar det som är mellan <pre> taggarna och sparar i @text. vi kallar sedan på create_array för att omvandla strängen @text till en 2d array i stil med: [["Arsenal", 38],["Liverpool",38]].

load_file(path)
Inget spännande, läser in utifrån en filväl i format './filename.txt' och sparar som en sträng i @text

print_table
den publika print metoden för hela tabellen, då tabellen/arrayen redan är sorterad så kallar vi bara på den privata metoden "print" som löser själva utskriften.

print_top
Då tabellen/arrayen är sorterad så behöver vi bara returnera det första elementet(en array) i 2d arrayen för att få den som har minst målskillnad/temperaturskillnad. För att skriva ut i korrekt format gör vi en check mot längden på den första arrayen då den kommer variera beroende på om det är fotboll(8)/väder(17) och då använder vi korrekt proc objekt för att skriva ut i ett rimligt format.

create_array
Eftersom vi antar att innehållet i tabellerna kommer vara likadant men värdena skiljer sig åt så kan vi söka i texten för att se om det är fotboll/väder. Beroende på detta skapar vi 2d arrayen med .scan och det korrekta regex objektet, vi kallar även här sort_array funktionen med korrekt proc objekt för att göra jämförelsen i sort.

sort_array(compare_proc)
Sorterar 2d arrayen "in place" och använder sig av inparameterns block för att göra själv jämförelsen. Inparametern kommer skilja sig när det är fotboll/väder och korrekt skickas med ifrån create_array funktionen.

print(array)
Vi kollar här längden på först subarrayen för att se om det är fotboll(8)/väder(17) och kallar då på korrekt header att skriva ut och korrekt proc object att skicka med när vi använder .each på hela 2d arrayen för att skriva ut varje subarray på korrekt sätt.

Reflektion:

I det stora hela så var det mycket kod som gick att återanvända inför den andra delen av uppgiften. Båda load funktionerna är likadana, print_table är likadan, sort_array är likadan. De andra är modifierade för att själva hantera om det är en fotboll/väder tabell så man hade säkert kunnat låta de vara och i print_top tex bara ta in en sträng och ett proc objekt som i ett annat steg valts för att vara lämpligt beroende på om det är fotboll/väder.
Tex:

```
def print_top(header, print_proc)
        puts @header
        print_proc.call(@table_array.first))
end
```

Men det kändes trevligare att lägga in funktionaliteten så det hanteras automatiskt av koden beroende på vad man laddar in.

Så som implementationen ser ut nu om man vill lägga till något mer behöver man lägga till en header sträng, ett proc objekt som hanterade formateringen på utskriften av tabellen, ett regex objekt man använde i matchningen för att få korrekta matchgrupper samt ett proc objekt man använder för jämförelsen när man ska sortera tabellen. Man behöver även uppdatera "print_top", "create_array" och "print" för de har logik som väljer vad som ska göras beroende på vad som laddats in.

Testning:

Då mycket av funktionaliteten bara är att skriva ut det på korrekt sätt som man kan kolla lätt i terminalen bara genom att köra programmet så har vi valt att fokusera testningen på att inläsningen av texten till en sträng, omvandlingen ifrån en sträng till en 2d array samt sorteringen av denna array. Genom att testa om strängen matchar originaltexten och sedan i sorteringen om en subarray i @table_array matchar en given array så testar vi implicit att rätt sak hamnar på rätt ställe och att programmet hanterar både väder och fotbollsdata.

Uppgift 2

Tolkning av uppgiften:

- Vi ska läsa in informationen direkt ifrån hemsidan
- Vi har valt att spara informationen om varje event i en hash och sedan varje hash i en array.
- Det ska finnas en funktion som skriver ut information om alla events
- Vi använder REXML och har valt att använda oss av DOM parsning med hjälp av xpath.
- Vi kommer inte implementera alla hCalender/hCard properties utan bara de vi kan se att exempelsidan innehåller

Vi tänkte använda Nokogiri ifrån början för att hämta in xml datan då vi använde det för uppgift 1 men det visade sig vara svårt att hitta information om hur man använde Nokogiri när man väl parsat texten första början och det tog för lång tid att lista ut varje steg så vi bytte till rexml som är mer intuitivt.

Grundtanken är att vi ska DOM parsea hela XML dokumentet och i ett första steg hämta ut alla events som indikeras av 'vevent' class-attributet som då sparas i en array av REXML::element och sedan därifrån hämta ut informationen vi behöver för varje event när vi itererar igenom arrayen.

I loopen för varje event skapar vi en hash med nycklar som representerar hCalender/hCard properties och anpassar så att koden kollar om det finns en " " tagg då den påverkar vilka nycklar som skapas. Varje hash sparas sedan i en array som är en instansvariabel i klassen och kan därifrån skrivas ut etc.

Förklaring funktioner:

def load_url(url_string)
Först läser vi in själva xml sidan med hjälp av URI.open och använder då urlen direkt för att hämta datan och skapar ett REXML::Document. Ifrån det hämter vi ut alla div taggar med attributet class='vevent' med hjälp av get_elements och xpath kommandot "//div[@class='vevent']" som hämtar den taggen oavsett var, eller hur många gånger den existerar och sparar i en variabel vi sedan kan jobba med.

Vi itererar sedan igenom arrayen 'events' där alla ovan nämnda taggar sparades och ifrån varje element i den arrayen hämtar vi ut informationen med .text som hör till eventet i form av hCalender/hCard properties med hjälp av xpath. Något vi fastnade på ett bra tag var att vi hade missförstått hur xpath fungerade och vi inte sa åt xpath att leta relativt till den nuvarande noden.
Tex:
event_dictionary["Name"] = event.elements["//span[@class='summary']"].text <-- felaktig

event_dictionary["Name"] = event.elements[".//span[@class='summary']"].text <-- korrekt

För varje iteration i events så skapar vi en hash där varje key motsvarar en av de hCalender/hCard properties som finns i dokumentet och värdet i paret blir texten som finns i taggen. Vi pushar sedan hela hashen till en instansvariabel i form av en array av dessa hashes som vi då sedan kan skriva ut information ifrån när man laddat in xml infon till objektet.

def print_events
Vi skriver bara ut alla events som laddats in genom att iterera igenom alla events och sedan för varje event iterera igenom varje hash med dess key/value par som då skrivs ut och avsluta varje event med en \n för att det ska se någorlunda rimligt ut.

Testning:

Även här går uppgiften ut på att skriva ut information man hämtar så det mesta går att kolla med utskrifterna ifrån programmet. Således valde vi att inte göra någon utförlig testning för denna delen utan bara kontrollera att det ser rätt ut i stora drag. Då eventen bara existerar i två versioner kollar vi endast att de två typerna har lästs in i det formatet vi förväntar oss, här kollar vi även att innehållet i varje hash och att dess keys/values är vad vi förväntar oss. Vi kontrollerar sedan att vi har läst in rätt antal event och att varje version av eventen har korrekt antal key/value pairs vilket kommer variera beroende på om eventet hade en " " tagg eller inte.