Utvecklarblogg - Seminarium 2

Grupp B6 - guser766 | hadan326 | vinah331

Att lära sig Ruby (nya saker)

För att lösa regexuppgiften refererade vi till https://rubular.com/ för själva uttrycket och till https://www.rubyguides.com/2015/06/ruby-regex/ för att hitta de Ruby-sepcifika funktioner som används i vår lösning, t ex *scan*. Man förstår bäst regex genom att testa sig fram, vilket Rubular är bra för.

DOM-parser

Vi använde en träd-parser för att lösa uppgift 2 i den här labben eftersom vi tyckte att det skulle gå mycket smidigare. Ett DOM objekt har många goda inbyggda funktioner som man kan använda som gör det hela sökningen i en xml dokument mycket enklare. Vi antog att det skulle kräva mycket mer kodning ifall vi skulle använda ström-parsning eftersom man själv behöver implementera en klass och funktioner för den.

XPath

För att lära oss XPath refererade vi till https://www.w3schools.com/xml/xpath_syntax.asp, dock var beskrivningen ibland svår att översätta till kod och det krävdes en del experimenterande för att få till korrekt syntax.
Vi har egentligen ingen vidare motivering till varför vi valde REXML som parser. Framförallt fanns det några exempel från föreläsningen som vi kunde referera till.

Tolkning av uppgifter och förklaring av lösningar

Uppgift 1

Uppgiften var att utifrån ett givet text-dokument skapa en rapport som rangordnar fotbollslag utifrån skillnaden i gjorda och insläppta mål. Man ska också kunna hämta ut specifikt vilket lag som hade minst skillnad.
Vår tolkning var att "rapporten" skulle vara en array som innehöll hashar, en för varje lag.
Man skulle dels kunna skapa rapporten direkt från dokumentets ordning, och sedan skapa en sorterad version utifrån jämförelsen av två data.
Programmet skulle också kunna hantera en väderrapport, som sorterade på temperaturskillnader.

Vår lösning använder regex-uttryck för att hitta matchningar, och grupperingar i dessa uttryck används för att plocka ut relevant data.

- **Football**: *([a-zA-z]+).+\s(\d+)\s+-\s+(\d+).+*

- **Weather**: *^\s+(\d{1,2})\s+(\d{1,2})*?\s+(\d{1,2}).+\n*

Lösningen förutsätter att regex-uttrycket har tre grupper, eftersom data explicit hämtas ut från matchnings-index.
Grupper kan namnges så att data kan hämtas ut med *match.namn*. Dock fungerar det inte med funktionen *scan* eftersom den returnerade arrayen inte bibehåller namn-datan. Därför fick vi nöja oss med att hämta ut index-platser och spara ner dem som namngivna data i hashen.

Lösningen med regex-yttryck gjorde att vi inför del två av uppgiften inte behövde ändra någon kod utöver själva regex-uttrycket. Vissa funktioner och variabler döptes dock om för att vara mer generella och inte specifika för fotboll. T ex Team och Goals döptes om till Identifier och Data1.

Uppgift 2

Vår tolkning av uppgiften var att skapa ett "Event-objekt" som kan spara en kalenderhändelse. Ett Event-objek i vår implementation är en ***Hash*** som kan spara olika detaljer för en händelse. Med hjälp av en medlemsfunktion i Event-klassen som heter ***to_s()*** kan dessa information konverteras till en sträng och senare skrivas ut på terminalen om man önskar. Event-klassen har en till medlemsfunktion som heter ***add_field()***. Den får in en key och ett value som inparameter och skapar motsvarande paret i hashen.

Exempel:
event = Event.new
event.add_field("Location", "Nobeltorget 1")
Resultatet är nu att event-objektet innehåller en nyckel som heter "Location" vars värde är "Nobeltorget 1".

Vi har en funktion som heter ***xhtml_read(website)*** som tar in en html-fil och skapar en DOM objekt. Dessutom sker alla sökningar inuti denna funktion och som resultat returnerar *xhtml-read()* en array som innehåller alla kalenderhändelser.
Arrayen senare skickas till funktionen ***print_events()*** för att skriva de ut på ett strukturerad sätt.

Tanken med vår lösning är att vi först letar efter matchningar för följande mönster:

doc.elements.each("//div[@class='vevent']") { block }

Som kan tolkas som: "Alla div-taggar som har en attribut som heter "class" och värdet är "vevent"".

Inuti block-delen som kommer efteråt skriver vi ytterligare sökningar för att hitta relevanta data (Name, Location, Date, Posted by och Website) för en event och sparar vi de i ett event objekt.
Varje iteration i blocken genererar ett event-objekt som sparas senare i en Array och till slut skrivas arrayen ut på terminalen.

Nedan kommer ett exempel på hur vi hittar information om datum för ett event.

1. **doc.elements.each("//div[@class='vevent']") do |element|**
2. **event = Event.new**

3. **event.add_field(“Date”, element.elements[“.//span[@class=’dstart’]”].text)**
4. **.**
5. **.**
6. **.**
7. **event_list << event**
8. **end**

Rad 1 kommer att hitta alla div-taggar och för varje div-tag skapas en ny event på rad 2 och sedan och på rad 3 läggs till relevanta information för datum i event-objektet och till slut på rad 7 läggs till event i en array. Koden (till exempel Xpath-uttryck) för att hitta andra information som kommer efter rad 3 och innan rad 7 för event ser lite olika ut men principen är samma.