Utvecklarblogg Se1

Den här dokumentationen är för utvecklarblogg till seminarium 1 i TDP007, meningen med att skriva detta är för att redovisa hur vi har tänkt och löst uppgifterna.

I början märkte vi att Ruby är lite annorlunda programmeringsspråk, men snart efter första två uppgifterna så bilden var tydligare för oss eftersom man hade de flesta kunskaperna innan. Precis som andra kända datorspråk så finns det massor av källor för att lära oss nya saker om Ruby såsom Ruby-dokumentationen, Test::unit och självklart *stackoverflow.*

Det finns inte väldigt stor skillnad mellan Ruby och de andra kurser vi har lärt oss under första terminen utan med Ruby handlar det mer om “frihet” att skriva kod. Vi lärt oss att det kan t.ex. finnas olika sätt skriva en *loop* bara. Detta var ganska ny för oss, vilket gör saken både lätt och svårt för oss då vi var vana med väldigt små fel såsom *indentering* eller att man inte behöver avsluta kodraden med ";", samt att man inte behöver skriva kod inom parenteser hela tiden utan man kan göra det om man tycker att koden kommer bli lättare att läsa. Det som är mest irriterande med Ruby är att det inte finns ett generellt sätt att t.ex. definiera variabler och sådant även om man har den friheten att skriva koden nästan hur man vill. Detta gör det lite svårt att sätta sig in i vilket gör oss att lägga mer tid att komma igång med språket.

I första labbpasset jobbade vi mest med de grundläggande saker om Ruby och första uppgiften då vi var inte så bekanta med språket. Där hade vi också svårt att komma igång samt hur man skriver enhetstesten. I andra labbpasset arbetade vi om hur vi kan skriva utvecklarbloggen och vad den ska exakt innehålla, samt en specifik fråga om en av uppgifterna.

Hur vi har löst och tolkat uppgifterna:

- Uppgift 2
  I denna uppgift använde vi *Range#inject* funktion eftersom vi var ombedd att använda den. Vi behövde skriva en funktion som kan beräkna factorial av ett positivt heltal.
  Denna uppgift var både lätt och svårt eftersom vi måste ha tagit reda på hur *inject* funkar och det var lätt för att det räckte med att veta hur factorial fungerar.

- Uppgift 3
  I denna uppgift använde vi oss av *sort_by* för att skriva en funktion som tar en array med olika längd av strängar och returnerar den längsta strängen genom att sortera arrayen efter längden.
  Denna uppgift var inte så svårt att tolka och lösa, utan det var lite svårt att hitta på vilket sätt ska vi returnera den längsta strängen.

- Uppgift 5
  I denna uppgift behövde vi skapa en klass som innehåller namn, efternamn och personens hela namn samt funktioner för att komma åt namnet och efternamnet. Uppgiften var ganska svårt eftersom vi hade svårt att definiera variabler och inte specificera datatypen till variablerna.

- Uppgift 6
  I denna uppgift behövde vi skapa en klass som innehåller personens ålder, födelseår och namn(namnet kom vi åt från uppgift 5) .
  Eftersom uppgiften var mycket lik den föregående uppgiften så kunde vi lösa uppgiften snabbare. Det som fick hjälp av var ‘date’ modulen för att hämta DateTime och Date som stod på labbupgiften med.

- Uppgift 7
  I denna uppgift använde vi oss av *if-satser* för att kunna skapa en funktion som kan beräkna Fibonacci-tal genom att addera de två föregående siffror i Fibonacci-listan.
  Uppgiften var ganska lätt att lösa eftersom man behövde inte använda mer än *if-satser* vilket är vi redan bekanta med. Men det var lite svårt i början eftersom vi visste inte vad är Fibonacci-tal och hur ser listan ut.

- Uppgift 8
  I denna uppgift behövde vi skapa en klass som skulle returnera en ord(acronym) som består av begynnelsebokstäverna. Uppgiften gjorde oss att gå tillbaka och gå igenom våra kunskaper inom *Regex* från TDP001 och TDP002. Vi använde både *split* och *map(för att skapa en ny array från värdena i blocket)* och *join(för att kombinera de splittade strängar)* för att skriva funktionen *acronym.*
  Uppgiften var ganska skönt kan vi säga då är det alltid bra att jobba med *Regex*, tycker vi i alla fall.

- Uppgift 10
  I denna uppgift behövde vi använda *Regex* för att kunna skapa en funktion som kan matcha användarnamnet ur en sträng som kan fungera också oavsett vad som står först på raden, så länge det är en följd av små eller stora bokstäver innan kolon.
  Vi löste uppgiften genom att matcha allt förutom det namnet och sedan använde vi oss av *post_match* för att returnera det resten som vi inte matchade.
  Uppgiften var både svårt och lätt. Lätt eftersom vi är bekanta med reguljära uttryck från föregående terminen. Svårt eftersom vi har tagit mycket tid att tolka uppgiften och hitta det bästa sättet för att lösa det.

- Uppgift 12
  Precis som föregående uppgifterna, behövde vi att använda *Regex* för att skriva en funktion som returnerar registreringsnumret ur angiven formen på labbuppgiften.
  Vi använde nästan samma metod som vi gjorde i uppgift 10 fast vi skrev funktionen så att den kan fångar första matchningen ur angiven formen.
  Denna uppgift tog lite mer tid då vi var tvungen att söka hjälp från labbassistenten om den. Till slut lyckades vi skriva en fungerande *Regex* som kunde fånga den giltiga registreringsnumret ur en sträng.