TDP007 - Tenta

2023-06-09

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 jourhavande genom att räcka upp handen
- 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 och logga ut.

Uppgift 1 - Teori (9p)

Du har i kursen TDP007 arbetat med en generell parser som du hittar i rdparse.rb. Målet med denna uppgift är att förklara hur denna parser fungerar. Du kan utgå från det existerande diceroller språket där det behövs. För full poäng ska följande aspekter föklaras på ett sätt som visar att du förstår hur det fungerar:

- I diceroller språket i klassen `DiceRoller` defineras funktionen `initialize`. I denna funktion skapas en datamedlem `@diceParser`. Vad representerar denna datamedlem?
- När `@diceParser` skapas så finns 4 saker med i blocket som skickas med till `Parser`s konstruktor: `token`, `start`, `rule`, och `match`. Förklara vad dessa är och varför de specificeras på sättet de gör.
- Förklara hur rdparse använder `token`, `start`, `rule`, och `match` för att sedan parsa texten som skickas in och faktiskt gör det som språket ska göra när texten parsas..

I din lösning av denna uppgift har du maximalt 450 ord att använda, var koncis i ditt svar. (Du kan räkna ord i din texteditor, i Emacs kan man räkna orden i en buffer med commandot `M-x count-words`).

Uppgift 2 - DOM-parsning av XML (10p)

I filen `royalfamily.xml` finns ett släktträd för några av personerna i släkten Tudor, som under främst 1500-talet satt på den engelska tronen. Den information som finns lagrad utgörs av personernas namn, när de levde, under vilka år de regerade vilket land, samt vilka barn de hade. Alla personer i släkten finns inte med, utan enbart de viktigaste. Formatet framgår av en inkluderad DTD.

Din uppgift är att skriva olika funktioner för att kunna läsa in och hantera informationen i filen. Du ska göra detta med DOM-parsning.

a) (4p) Skriv en funktion find_regents som läser in släktträdet och i kronologisk ordning skriver regenterna för ett viss land. Så här ska det se ut:

```
>> find_regents("royalfamily.xml", "England")
Henry VII of England (1485-1509)
Henry VII of England (1509-1547)
Edward IV of England (1547-1553)
Mary I of England (1553-1558)
Elizabeth I of England (1558-1603)
James I of England (1603-1625)
```

b) (3p) Skriv en funktion regency som läser in släktträdet och skriver ut alla kungar och drottningar, sorterade efter hur länge de satt på tronen. Personen med längst regeringstid ska skrivas ut först. För personer som har haft flera uppdrag på tronen räknas dessa som olika, så man behöver inte summera ihop dem. Födelseåret för respektive person ska också skrivas ut. Så här ska det se ut:

```
>> regency("royalfamily.xml")
Elizabeth I of England (b. 1533) 45 years
Henry VIII of England (b. 1491) 38 years
James VI of Scotland (b. 1566) 36 years
James V of Scotland (b. 1512) 29 years
Mary, Queen of Scots (b. 1542) 25 years
...
```

c) (3p) Skriv en funktion children som läser in släktträdet och hittar den person som har flest barn. Körexempel:

```
>> children("royalfamily.xml")
Henry (1457) had 4 children.
```

Eftersom så många personer har samma namn vill vi även ha med personens födelseår i utskriften.

Lösningen på alla deluppgifter lämnas in i en fil på tentan

Uppgift 3 - SAX-parsning av XML(10p)

Lös uppgift 2, fast med SAX-parsning istället. Exempel på hur man kan arbeta med en SAX-parser finns i `sax_example.rb`.

Uppgift 4 - Parser(11p)

I den här uppgiften ska vi konstruera ett litet språk för att rita mycket enkel ASCII-grafik. Varje program i språket måste börja med att man anger hur stor yta man ska rita på, t.ex.: `init 10 10`

Detta gör att vi skapar ett rutnät med 10 x 10 tecken (första talet är bredden och andra talet är höjden). Vi tänker oss att vi har en penna som kan rita ett tecken i varje ruta. När vi precis har skapat rutnätet svävar pennan ovanför position (0, 0) vilket motsvarar övre vänstra hörnet av rutnätet. Vi kan sätta ner pennan på pappret med kommandot: `down W`

Detta ritar ett `W` på den position som är under pennan. Om vi därefter gör en förflyttning kommer pennan att fortsätta rita i de rutor vi kommer till. Vi kan förflytta oss i fyra olika riktningar, motsvarande väderstrecken. Exempel:

```
e 5
s 2
w 2
n 1
```

Ovanstående kommandon flyttar pennan österut (höger) fem steg, därefter söderut (nedåt) två steg, o.s.v. Om pennan är nedsänkt ritas det tecken vi har angivit, annars händer inget annat än att vi förflyttar oss. Vi kan lyfta pennan med kommandot: `up`

Uppgiften går ut på att implementera en parser för detta lilla språk med utgångspunkt i parsern i filen rdparse.rb. Tanken är att parsern bygger upp en datastruktur som motsvarar ritningen, och när parsningen är klar ska denna ritas ut. Exempel:

```
>> dp = DrawParser.new
>> dp.draw("init 10 10 down # e3s3w3n3 up s4 down * e3s3w3n3")
------------
|####      |
|#  #      |
|#  #      |
|####      |
|****      |
|*  *      |
|*  *      |
|****      |
|          |
|          |
------------
```

Ett fullständigt program börjar alltid med init-kommandot och hela programmet är på endast en rad. Som framgår av exemplet behöver man inte använda mellanslag mellan kommandon och argument om det inte är nödvändigt för tolkningen.

De yttre strecken i exemplet hör inte till själva ritningen utan är bara en ram för att vi lättare ska se vad vi ritat. Definiera klassen DrawParser som, med hjälp av rdparse.rb, kan tolka vårt enkla rit-språk och producera ASCII-grafik enligt ovanstående exempel.

Uppgift 5 - DSL (10p)

Macros är i princip text som kan expanderas till annan text. I den givna filen `dsl.rb` finns en fil som definerar 4 macros. Detta är ett domänspecifikt språk där man definerar macron genom att skriva nyckelordet *define* följt av ett ord, följt av godtyckligt många ord fram till slutet av raden. Det första ordet efter nyckelordet är namnet på macrot som ska defineras och det ordet skall sedan kunna ersättas med alla orden som kommer efter.

Du ska implementera detta domänspecifika språk med hjälp av `method_missing` i ruby. Du får också skapa en medlemsfunktion för nyckelordet *define*. I den givna filen `dsl_reader.rb` finns lite given kod som du kan utgå ifrån.

Resultatet av att köra denna parser ska vara en `Hash` som associerar varje macro med sin text. Se detta exempel:

```
macros = MacroParser.parse("dsl.rb")
print macros
>> {
    "tl"=>"too long",
    "dr"=>"did read",
    "tlbdr"=>"tl but dr",
    "ltlbdr"=>"like tlbdr"
   }
```

Du ska sedan skapa en funktion `expand` som tar emot denna hash av macros och en sträng. `expand` ska skriva ut varje ord i strängen om den inte finns i hashen av macros. Om ordet var ett macro ska det ersättas med alla orden det macrot är associerat med. **OBS:** denna ersättning av macrot måste ske rekursivt så att eventuella ord som macrot ersätts med också ersätts om de i sig är macros. I körexemplet kan vi exempelvis se att macrot `tldr` associeras med orden `tl` och `dr` som i sin tur associeras med ord:

Körexempel:

```
macros = MacroParser.parse("dsl.rb")
text = "this text was really long so I'm ltlbdr"
expand(macros, text)
>> "this text was really long so I'm like too long but did read"
```

- Formateringen av utskrift spelar ingen roll, men datatyperna och hur de associeras med varandra spelar roll