TDP007 - Dugga 2

2025

Regler

- All kod efterrättas på denna dugga
- Inga elektroniska hjälpmedel får medtas. Mobiltelefon ska vara avstängd och ligga i jacka eller väska.
- Inga ytterkläder eller väskor vid skrivplatsen.
- Student får lämna salen tidigast en timme efter start.
- 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 i sal 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 3 delar. Del 1 är uppgifter på grund-nivå som relaterar till tekniker från kursens första två seminarier, dessa bedöms som G/U. Del 2 är uppgifter på grund-nivå som relaterar till tekniker från kursens andra två seminarier, dessa bedöm som G/U. Del 3 består av mer avancerade uppgifter, där varje uppgift relaterar till någon teknik från kursens fyra seminarier. Uppgifter på del 3 ger upp till 10 poäng. Betyg på tentamen ges enligt följande tabell. För alla betyg så måste förutom de insamlade poängen alla uppgifter på Del 1 och Del 2 lösas med godkänt betyg.

Insamlade poäng Del 3	**Betyg**
10	3
17	4
24	5

Duggor

Den första duggan består av två delar. Del 1 är uppgifter på grundnivå som relaterar till tekniker från kursens första två seminarier och bedöms som G/U. Del 2 består av en mer avancerad XML-uppgift som ger upp till 10 poäng. Om alla uppgifter på Del 1 har godkänt betyg från denna dugga så har du Del 1 på ordinarie tentamen tillgodo.

Den andra duggan består av två delar. Del 1 är uppgifter på grundnivå som relaterar till tekniker från kursens sista två seminarier och bedöms som G/U. Del 2 består av en mer avancerad parser-uppgift som ger upp till 10 poäng. Om alla uppgifter på Del 1 har godkänt betyg från denna dugga så har du Del 2 på ordinarie tentamen tillgodo.

Om del 1 är avklarad på båda duggorna och du har samlat in totalt 14 poäng från duggorna så har du förutom Del 1 och Del 2 på ordinarie tentamen tillgodo också betyg 3 på tentamensmomentet.

Inloggning

Logga in på datorn i labbsalen med ditt liu-id och lösenord. Detta är samma inloggningsuppgifter som du använder i Lisam.

Skrivbordsmiljön

När du kommit in i tentasystemet har du en normal skrivbordsmiljö (Mate-session i Ubuntu). Efter en stund kommer även din kommunikationsklient att dyka upp automatiskt. Startmenyn är nedskalad till enbart det som examinator bedömt relevant. Andra program kan fortfarande finnas tillgängliga genom att starta dem från en terminal. Observera att en del program som använder nätverkstjänster inte fungerar normalt, eftersom nätverket inte är åtkomligt.

När du är inloggad är det viktigt att du har tentaklienten igång hela tiden. Om den inte dykt upp fem minuter efter inloggning och inte heller när du startar den manuellt från menyn (fisken) tar du kontakt med assistent eller vakt i sal.

Terminalkommandon

`ruby` används för att köra en ruby-fil.
`irb` används för att starta ruby i interaktivt läge.
`ri` används för att komma åt den inbyggda dokumentationen.

Ruby-docs

På tentan har du tillgång till referenssidorna på `https://ruby-doc.org/` (både core och std-lib finns tillgänglig) via webbläsaren Chrome. Starta `chromium-browser` i terminalen eller välj lämpligt alternativ från startmenyn. Observera att allt utom referenssidorna är avstängt. Om du inte kan komma åt en sida du tycker hör till referenssidorna (som kanske blockerats av misstag) kan du skicka ett meddelande via tentaklienten. Tag hjälp av assistent i sal om det inte fungerar.

Rubular

På tentan har du tillgång till sidan `https://rubular.com/`.

Givna filer

Eventuella givna filer finns i katalogen `given_files`. Denna underkatalog är skrivskyddad, så det är ingen risk du råkar ändra på dessa filer. Skrivskyddet gör dock att du måste kopiera in de givna filer du vill använda till tentakontots hemkatalog. Hur du listar och kopierar filer ska du kunna. Hemkatalogen står du i från början, och du kommer alltid tillbaka till den genom att bara exekvera kommandot `cd` i terminalen.

Avslutning

När du skickat in alla uppgifter och känner dig färdig kan du logga ut och lämna salen. Antalet poäng meddelas i efterhand via webreg.

Avsluta alla öppna program och logga ut ur datorn. Lämna inte datorn innan du ser att du är utloggad.

Del 1: Uppgift 1 - Parser variabler

I filen `uppgift1.rb` finns parsern som vi är bekanta med från labbserien, tillsammans med Diceroller-språket som vi känner igen. Din uppgift är att utöka detta språk så att det är möjligt att spara resultatet av beräkningar/tärningsslag variabler. Genom att skriva `a = d20` så ska resultatet av att slå en 20-sidig tärning lagras i variabelnamnet `a`. Exempelvis så slog denna 20-sidiga tärning 12, vilket blir värdet som lagras i variabeln. Skriver vi sedan `b = a + 2` skall värdet från `a` (12) hämtas, och adderas med 2. Resultatet sparas då i variabeln `b`. Du behöver inte ta höjd för vad som händer när en variabel skrivs över eller inte finns, utan kan utgå från att allting används korrekt. Du kan använda den redan existerande aritmetiska logiken i filen, och behöver inte ta hänsyn till parenteserna för denna uppgift.

Det är också ok att köra parsern direkt i huvudprogrammet, du behöver alltså inte nödvändigtvis få det att fungera i interaktivt läge som i de funktionella kraven.

Funktionella krav:

```
irb(main):001> require './uppgift1.rb'
=> true
irb(main):002> DiceRoller.new.roll
[diceroller] a = d20
=> 12
[diceroller] b = a+2
=> 14
```

Del 1: Uppgift 2 - Constraints

I den givna filen `uppgift2.rb` hittar du constraint-nätverken vi arbetat med i labbserien. Du kan anta att de aritmetiska constraintsen är fullt fungerande och att du bara behöver använda constraints av typen `Subtracter` och `Multiplier`.

En enkel formel för att räkna ut vinsten vid försäljning av en vara är $R = (p - c) \times q$, där R representerar den totala vinsten, p är priset per vara, c är kostnaden för tillverkningen och q är antalet varor sålda.

Din uppgift är att implementera funktionen `revenue_constraint` i den givna filen `uppgift2.rb` så att huvudprogrammet fungerar som förväntat. Funktionen ska sätta upp ett constraint-nätverk som motsvarar vinstberäkningsformeln. Funktionen tar emot parametrarna `price` och `cost`, som motsvarar p och c respektive. Constraint-nätverket ska därefter sättas upp, och de connectors som motsvarar R, p och q ska returneras från funktionen. Bägge bör vara 0 initialt. Justeras sedan q (antal varor sålda) eller p ska R uppdateras för att visa vinsten. Justeras R ska q uppdateras för att visa antalet varor som behöver säljas för att nå denna vinst.

Funktionella krav:

```
r after assigning q=100: 3000.0
q after assigning r=13680: 456
r after assigning p=30 and q=456: 9120
```

Del 1: Uppgift 3 - DSL

I filen `uppgift3.rb` finns den påbörjade DSL-läsaren vi känner igen från labbserien. I den givna filen `dsl.rb` finns ett litet domänspecifikt språk som används för att skicka beställningar till ett varuhus. På en rad skriver man `order`, och varje order ska få ett ordernummer. Detta ska börja på 1, och öka med 1 för varje order som läggs till. För varje order kan ett godtyckligt antal varor läggas till, samt en kvantitet för dessa varor (en integer). I slutet dyker det upp en address och en mottagare i form av strängar. Din uppgift är att lösa så att denna typ av filer kan läsas in och lagras i en associativ behållare (se funktionella krav). Problemet ska lösas med `method_missing` för alla rader. Du får däremot lägga till en separat funktion för `order` om du vill.

Du ska sedan returnera behållaren och skriva ut den. Formatteringen av behållaren spelar ingen roll, inte heller om nycklarna är av typen symbol eller sträng. Strukturen av behållaren och datan ska däremot stämma med kraven. Du behöver inte heller ta höjd för ifall address eller mottagare inte finns med i den DSL som läses in.

Funktionella krav:

```
{
  1 => {
    cucumber: 12,
    coffee: 2,
    butter: 5,
    address: "Willow Street 3",
    recipient: "Bright Coffee"
  },
  2 => {
    noodles: 6,
    kimchi: 10,
    flour: 3,
    address: "Canal Street 19",
    recipient: "Korean BBQ"
  }
}
```

Del 2: Uppgift 4 - Parser AST

I filen `uppgift4.rb` finns parsern som vi är bekanta med från labbserien. Diceroller-språket har ersatts med ett språk som heter `Arithmetic Parser` som egentligen är en förenklad version av Diceroller. Din uppgift är att justera språket så att det kan hantera aritmetiska uttryck i naturligt tal. Användaren ska, till exempel, kunna skriva (2 *times* 2) *plus* 18 *divide* (4 *minus* 2), vilket motsvarar $(2*2)+18/(4-2)$ för att få resultatet 13. Din implementation skall även klara av negativa tal, t.ex. (−2 *times* −3) *divide* (20 *divide* 10) ska motsvara $(-2*-6/(20/10)$. De aritmetiska operatorer som ska inkluderas och hur de ska tolkas i språket listas nedan. Dessutom noteras det även en siffra för vilken prioritet de skall ha, där 1 indikerar högsta prioritet. För de operatorer med samma siffor spelar det ingen roll vilken som utvärderas först.

- 1: Parenteser: För uttrycket 4 *times* (4 *minus* 2) ska uttrycket inom parenteser utvärderas först
- 2: Exponenter: 4^2 skrivs som 4 *power* 2
- 3: Multiplikation: $4*2$ skrivs som 4 *times* 2
- 3: Division: $4/2$ skrivs som 4 *divide* 2
- 4: Addition: $4+2$ skrivs som 4 *plus* 2
- 4: Subtraktion: $4-2$ skrivs som 4 *minus* 2

Exakt hur du väljer att implementera detta är till stor del fritt, men parsern måste returnera ett abstrakt syntaxträd (AST). Var utvärderingen av din AST sker väljer du själv, men hela trädet skall byggas innan du utvärderar det. Du får själv välja hur du representerar det abstrakta syntaxträdet. Utskriften behöver innehålla korrekta resultat av beräkningarna på ett enkelt läsbart sätt, och utvärderingen ska ske i ordningen som listas ovan.

Funktionella krav:

```
AritParser, evaluating: 2 plus 2 => 4
AritParser, evaluating: 6 times 4 => 24
AritParser, evaluating: 2 power 10 => 1024
AritParser, evaluating: (4 minus 1) divide 2 => 1
AritParser, evaluating: (2 times 2) plus 18 divide (4 minus 2) => 13
AritParser, evaluating: (5 times 2) plus ((5 times 3) minus 5) => 20
AritParser, evaluating: (-2 times -3) divide (20 divide 10) => 3
AritParser, evaluating: 6 plus 6 plus 3 times 2 => 18
```