TDP007 - Tenta

2026-03-25

Regler

- All kod liverättas under denna examination
- 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 vakt 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 2 delar. Del 1 är uppgifter på grund-nivå som relaterar till tekniker från kursens seminarier. Del 3 består av mer avancerade uppgifter, där varje uppgift relaterar till någon teknik från kursens seminarier. Alla uppgifter bedöms som G/U. Betygsättning vid tentamen görs enligt följande tabell (för alla betyg 3, 4, 5 krävs att alla uppgifter på grund-nivå är avklarade):

Avklarade uppgifter del 2	**Betyg**
0	3
1	4
2	5

Duggor

Varje dugga motsvarar halva tentan. Dugga 1 tar upp koncept från kursens första 2 seminarier. Dugga 2 tar upp koncept från de sista 2 seminarierna. För varje avklarad del under en dugga tillgodoräknas motsvarande uppgifter på första ordinarie tentamen efter kursen. Exempel: (1) Som student har jag klarat endast klarat av del 1 av dugga 2 i kursen. Om jag skriver ordinarie tentamen behöver jag då inte göra motsvarande hälften av del 1 på tentamen. (2) Som student har jag klarat av del 1 och del 2 på både dugga 1 och dugga 2. Jag har nu betyg 5 på tentamen och det finns ingen anledning för mig att skriva tentan.

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.

Webresurser

På tentan har du tillgång till referenssidorna, rubular och en version av kursboken. Klicka på web access på skrivbordet.

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 är nöjd med din insats (inte tänker arbeta vidare) kan du avsluta.

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

Del 1: Uppgift 1 - Regexp

I den givna filen **company.xml** finns information om anställda. Pontus skulle vilja ringa alla personer som har en arbetstelefon (typen av telefonnummer framgår i form av ett attribut i filen) och behöver därför skriva ut en telefonslista (se funktionella kraven). Lös en icke-trivial del av uppgiften med regexp.

Du behöver anta att ytterligare personer med andra kombinationer av telefonnummer kan läggas till i filen, men själva matchningen av ett arbetstelefonnummer kan vara ganska naivt. Alltså kan du anta att elementen som innehåller namn/telefonnummer är formatterade på sätten som de är i den givna filen.

Funktionella krav:

```
Anna: +46-13-4401101
Elin: +46-13-4401201
Oskar: +46-13-4401301
Tim: +46-13-4401501
```

Del 1: Uppgift 2 - XML

Uppgiften använder **company.xml**. Med en xml-parser (sax eller dom, men vi rekommenderar dom) läs ut efternamnen av all personal som arbetar på avdelningen (Department) som heter Research. Varje avdelning sparar id'n på anställda som jobbar på avdelningen, du kommer behöva använda dessa för att komma fram till vilka anställdas namn du ska skriva ut.

Funktionella krav:

```
Bergström
Holmqvist
Nyholm
Hedman
```

Del 1: Uppgift 3 - rdparse

Du ska modifiera parsern i den givna filen **uppgift3.rb**. Språket du skapar ska stödja heltal, parenteser och tre operatorer. Operatorerna beskrivs i prioritetsordning nedan (högst prioritet överst i listan, uttryck inom parentes har högre prioritet):

- ***** vanlig multiplikation
- **@@** skapar ett nytt tal av sina operander genom att sätta siffrorna i följd. **12 @@ 3** blir alltså **123**.
- **+** vanlig addition

Funktionella krav:

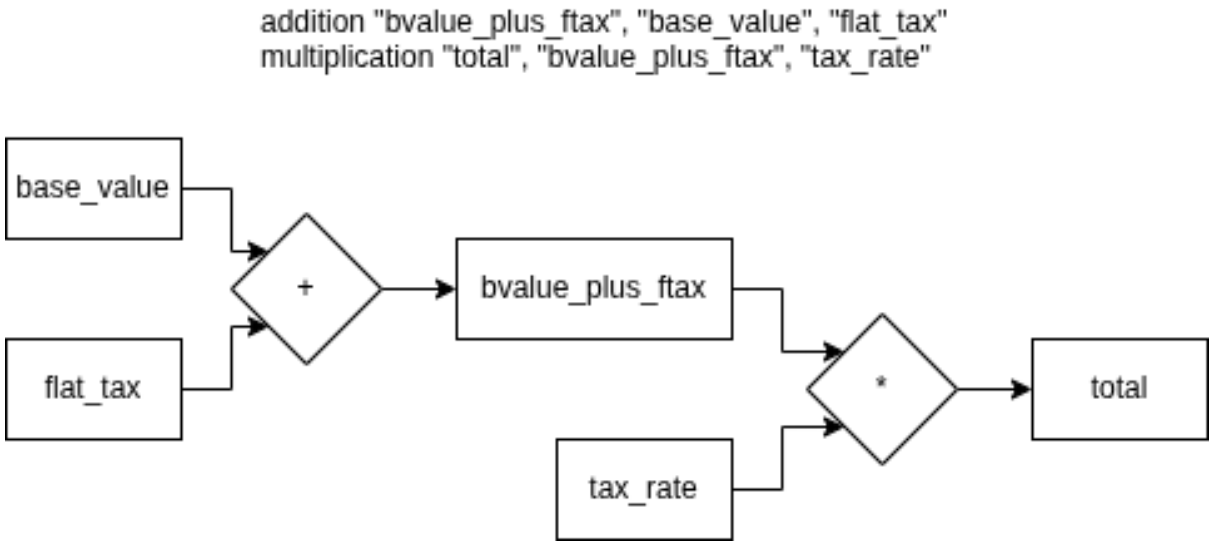
```
[funkyaritlang]2+2 => 4
[funkyaritlang]2@@2 => 22
[funkyaritlang]2+2 @@ 2 => 24
[funkyaritlang]2 * 2@@2 => 42
[funkyaritlang]2@@2+2 => 24
[funkyaritlang]2@@2*2 => 24
[funkyaritlang]2@2+1@3 => 233
```

Del 1: Uppgift 4 - DSL och Constraints

Denna uppgift utgår från de givna filerna **uppgift4.rb, dsl.rb och constraints.rb**. Filen med constraints är samma som från labbarna (utan buggen) och exemplet med att sätta upp en constraint för addition hittar du i botten av den filen. Du ska med hjälp av instance eval och method missing läsa det domänspecifika språket i dsl.rb.

I språket kan användaren skapa en ny connector genom att skriva ett valfritt namm (se första 5 raderna i dsl.rb). Man kan också koppla ihop connectors (skapa en constraint) genom att skriva **addition** eller **multiplication** följt av tre komma-sepparerade strängar (se rad 7-8 i dsl.rb), där strängarna är namnen på connector c, a och b (se bilden). Till sist kan man tilldela existerande connectors ett värde (se rad 10-12 i dsl.rb).

Särskilda funktioner kan skapas för att sätta upp additions- och multiplikations-constraints, men att connectors kan ha valfria namn när de skapas och när värden sätts ska lösas med method missing.



Funktionella krav:

```
[...] total got new value: 137.5
```

Du kan undersöka typen på ett objekt i Ruby med medlemsfunktionen **.class**. Du kan konvertera en sträng till en symbol med medlemsfunktionen **.to_sym** och en symbol till en sträng med medlemsfunktionen **.to_s**.

Del 2: Uppgift 5 - XML

Uppgiften använder den givna filen **company.xml**. Skapa en funktion i den givna filen **uppgift5.rb** som tar emot ett REXML::Document och tre strängar som parametrar. Funktionen ska returnera en **Hash** med alla anställda som jobbar på avdelningen för mjukvaruutveckling och som matchar vissa kriterier. Kriterierna anges genom de tre strängarna som skickas in i funktionen. Där den första strängen motsvarar **namnet** på ett element i en **<Person>.** Den andra motsvarar ett **attribut** i angivet element. Den tredje motsvarar **värdet** det attributet ska ha. En anställd ska endast inkluderas i hashen om alla dessa kriterier uppfylls.

Hashen består av personers för- och efternamn som nyckel och deras lön som värden. Uppgiften ska i icke-trivial utsträckning lösas med XML och xpath.

Funktionella krav (formattering av utskriften får variera):

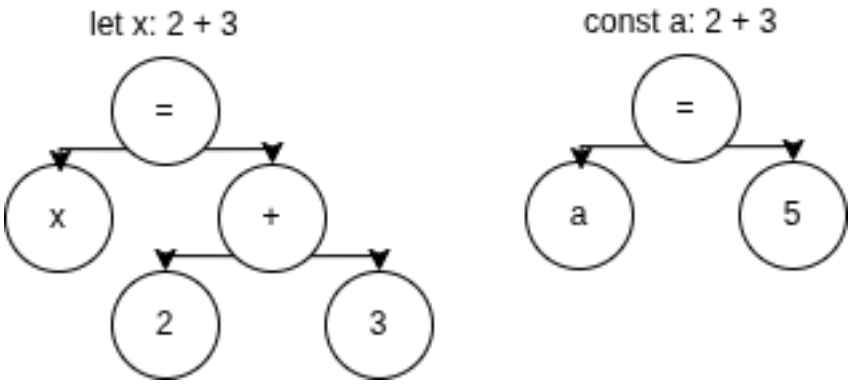
```
{"Elin Söderberg"=>"47000"}

{"Elin Söderberg"=>"47000",
 "Jonas Ekdal"=>"54000",
 "Tim Forsblom"=>"46000",
 "Sara Lundqvist"=>"41000"}
```

Del 2: Uppgift 6 - Parser och AST

I den givna filen **uppgift6.rb** finns parsern från labbserien (den är något förenklad och innehåller nu bara aritmetik). Du ska lägga till två typer av variabeltilldelning i språket, uppslagning av variabler och göra så att parsern returnerar ett Abstrakt Syntax Träd (AST) istället för att utvärdera uttrycken direkt. Parsern ska fungera med det givna huvudprogrammet som innehåller ett program som är skrivet på flera rader, se den givna filen.

Variabler skapas med nyckelordet **let** eller **const**. I första fallet ska AST sättas upp som vanligt (se bilden) för att sedan utvärderas under runtime. Men om nyckelordet **const** används så ska beräkningar som den tilldelningen bero på lösas under parse-time. I bilden ser du ett exempel på hur AST skulle skilja sig åt i parsern mellan **let** och **const**.



Det finns olika sätt att lösa detta på men en rekomendation är att identifiera de delträd som ger värdet på konstanter och exekvera dessa i förtid för att sedan ersätta delträdet med en nod som representerar det beräknade värdet.

Du kan anta att variabler som slås upp alltid har definierats i förväg och att tilldelningar med **const** endast beror på variabeluppslagning där variablerna är const.

Efter parsern satt upp ditt AST ska du evaluera det och skriva ut resultatet av utvärderingen (förslagsvis i funktionen run).

Funktionella krav:

```
=> 55
```