Instruktioner - Datortentamen TDDD73 Funktionell och imperativ programmering i Python

Hjälpmedel

Följande hjälpmedel är tillåtna:

- Exakt en valfri bok, t.ex. den rekommenderade kursboken. Boken får ha anteckningar, men inga lösa lappar.
- Exakt ett A4-papper med egna anteckningar om precis vad som helst. Det kan vara hand- eller maskinskrivet, på ena eller båda sidorna.
- Pennor, radergummin och liknande för att kunna skissa lösningar på papper.

Icke tillåtna hjälpmedel inkluderar bl.a. alla former av elektronisk utrustning samt böcker och anteckningar utöver det som listats ovan.

Lösa tomma papper för att göra anteckningar tillhandahålls av tentamensvakterna.

Genomförande

Under datortentan kommer du att arbeta i en begränsad och övervakad miljö. Med hjälp av en särskild tentaklient skickar du in dina svar. Du kan även använda tentaklienten för att ställa frågor till examinator. Besök i skrivsalen kommer endast ske vid allvarligare tekniska problem.

Varje uppgift kan skickas in en enda gång och kommer därefter poängsättas. Det finns alltså inga möjligheter till komplettering.

Betygsättning

Datortentan består av totalt sex uppgifter som vardera kan ge maximalt fem poäng. Uppgifterna är utvalda för att ha olika svårighetsgrad och är ordnade så att den lättaste oftast kommer först. Observera att vad som är lätt eller svårt naturligtvis skiljer sig från person till person, så det är alltid en bra idé att snabbt läsa igenom alla uppgifterna för att bättre kunna prioritera arbetet. Betygssättningen sker enligt följande tabell:

- För betyg 3 krävs minst 12 poäng.
- För betyg 4 krävs minst 19 poäng.
- För betyg 5 krävs minst 25 poäng.

Rättningskriterier

Följande allmänna kriterier används vid poängsättning av uppgifter. Lösningar som bryter mot dessa kommer att få poängavdrag.

- Funktioner ska ha exakt samma namn som i uppgiften. Detta underlättar rättningen.
- Namn på parametrar, variabler och hjälpfunktioner som inte specificerats explicit i uppgiften ska vara beskrivande och följa namnstandarden.
- Lösningen ska vara välstrukturerad och väldokumenterad på samma sätt som kursens laborationer.
- Lösningen ska följa de regler som uppgiften har satt upp. Det kan t.ex. röra sig om att vissa funktioner eller metoder ej får användas, eller att lösningen ska göras enligt en särskild modell.
- Lösningen ska fungera exakt som i körexemplen i uppgiften, om inte texten indikerar något annat.
- Lösningen ska vara generell, d.v.s. den ska inte enbart fungera för de körexempel som finns i uppgiften, utan även för alla andra möjliga indata på samma form. Felkontroller av indata behöver dock normalt inte göras, om inte uppgiften särskilt uttrycker det.

Uppgifter - Datortentamen TDDD73 Funktionell och imperativ programmering i Python

Tisdag 10 januari 2017 kl 8-13

Uppgift 1

En diofantisk ekvation är en ekvation med heltalslösningar. De är döpta efter den grekiska matematikern Diofantos. Ett känt exempel är Pythagoras sats. Ett sätt att lösa dessa är att prova alla möjliga heltal, vilket är möjligt eftersom de är uppräkningsbara. Din uppgift är att räkna antalet lösningar på diofantiska ekvationer på formen ax + by + cz = d givet konstanterna a, b, c, och d samt intervallen för variablerna. Skriv en funktion `diophantine(a, b, c, d, min_val, max_val)` som returnerar antalet lösningar på ax + by + cz = d där variablerna ligger i intervallet [`min_val`, `max_val`]. Din funktion behöver inte vara effektivt utan du kan till exempel prova alla möjligheter, dvs göra en uttömmande sökning.

```
>>> diophantine(2, 3, 4, 10, 0, 2)
3
>>> diophantine(2, 3, 4, 10, 0, 10)
5
```

Uppgift 2

Skriv en funktion som tar en lista och ett tal n större än 0 och summerar vart n:te tal i listan. Funktionen ska finnas i två varianter: en som arbetar enligt en rekursiv modell (kallad `sum_nth_r`) och en som arbetar enligt en iterativ modell (kallad `sum_nth_i`). Du får inte använda inbyggda funktioner eller metoder som behandlar hela listor. Du får inte heller använda listbyggare (eng. *list comprehensions*). Lösningarna ska vara funktionella i bemärkelsen att de inte får modifiera indata. Funktionerna ska fungera exakt likadant, bortsett från att de ska ha olika lösningsmodeller. Här är några exempel:

```
>>> sum_nth_r([1, 2, 3, 4, 5], 1)
15
>>> sum_nth_r([1, 2, 3, 4, 5], 2)
6
>>> sum_nth_i([1, 2, 3, 4, 5], 3)
3
```

De båda funktionerna `sum_nth_r` och `sum_nth_i` ger vardera 2.5 poäng, och det är möjligt att få poäng på dem oberoende av varandra.

Uppgift 3

Skriv en funktion `split(seq, threshold)` som tar en lista som kan innehålla godtyckligt nästlade listor och returnerar tre listor, en med alla tal mindre än `threshold`, en med alla tal lika med `threshold` och en med alla tal större än `threshold`. Funktionen ska vara icke-destruktiv. Det går bra att lösa uppgiften rekursivt eller iterativt. Tänk som vanligt på att dokumentera funktionen på lämpligt sätt.

```
>>> split([5, [2, 1, 7], 2, [5]], 5)
([2, 1, 2], [5, 5], [7])
```

Uppgift 4

Deluppgift 4a (3p)

Att integrera funktioner är en viktig del av den matematiska analysen. En funktion kan antingen integreras symboliskt eller numeriskt. Du har huvudsakligen lärt dig att integrera funktioner symboliskt, dvs att ta fram den primitiva funktionen. Ett sätt att numeriskt uppskatta integralen från a till b av en funktion f(x) är enligt (b-a)((f(a)+f(b))/2). Skriv en högre ordningens funktion `integrate(f)` som tar en funktion `f(x)` och returnerar en funktion som tar två argument, a och b, och beräknar integralen av `f` över intervallet [a, b] numeriskt enligt ovan.

Deluppgift 4b (2p)

Skriv ett uttryck som använder `integrate` för att räkna ut integralen från 2 till 4 av $3x^2$.

Uppgift 5

Så länge det har funnits pengar har människan behövt växla mellan olika valörer. Visste du att det finns 2156 olika sätt att växla 100kr till mynt med dagens svenska mynt (10kr, 5kr, 2kr, 1kr)? Skriv en funktion `count_change(coins, amount)` som tar en lista med valörer och ett belopp och räknar ut hur många olika sätt man kan växla beloppet till mynt. Du kan anta att det finns tillräckligt många mynt av varje valör. Funktionen behöver inte vara minnes- eller tidseffektiv.

```
>>> count_change([10, 5, 2, 1], 5)
4
>>> count_change([10, 5, 2, 1], 10)
11
>>> count_change([10, 5, 2, 1], 100)
2156
>>> count_change([50, 25, 10, 5, 1], 20)
9
>>> count_change([50, 25, 10, 5, 1], 100)
292
```

Uppgift 6

Matriser spelar en väldigt stor och viktig roll i både matematiken och datavetenskapen. En matris består av ett antal rader där varje rad har lika många element (kolumner).

Matrisen

$$A = \begin{pmatrix} 1 & 2 & 3 \\ -1 & 0 & 7 \\ 5 & -9 & 2 \\ 6 & 1 & 5 \end{pmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ -1 & 0 & 7 \\ 5 & -9 & 2 \\ 6 & 1 & 5 \end{bmatrix}$$

är en 4×3-matris. Elementet $A[2,3]$ eller $a_{2,3}$ är 7.

Din uppgift är att implementera några vanliga matrisoperationer enligt följande beskrivning. En matris är representerad som en lista med listor, där varje lista är en rad. Alla funktioner ska vara väl dokumenterade. De två första funktionerna ger 0.5 poäng, övriga ger 1 poäng vardera.

- `rows(matrix)` – Returnera antalet rader i `matrix`.
- `columns(matrix)` – Returnera antalet kolumner i `matrix`.
- `transpose(matrix)` – Returnera en ny matris där rader och kolumner i `matrix` har bytt plats, dvs en transponering av `matrix`.

  Transponering är en operation som bildar en matris genom att rader och kolonner för en given matris byter plats. En m×n-matris A har således en n×m-matris som sitt transponat. Transponatet till en matris betecknas

  $$A^T$$

  där

  $$A^T[i,j] = A[j,i]$$

  Exempel: $\begin{pmatrix} 1 & 0 & 2 \\ -1 & 3 & 1 \end{pmatrix}^T = \begin{pmatrix} 1 & -1 \\ 0 & 3 \\ 2 & 1 \end{pmatrix}$

- `add(matrix1, matrix2)` – Returnera en ny matris som är summan av `matrix1` och `matrix2`.

  Addition av två matriser förutsätter att matriserna har samma dimensioner.

  Om A och B är två m×n-matriser, så definieras C=A+B genom

  $$c_{i,j} = a_{i,j} + b_{i,j}$$

  Exempel:

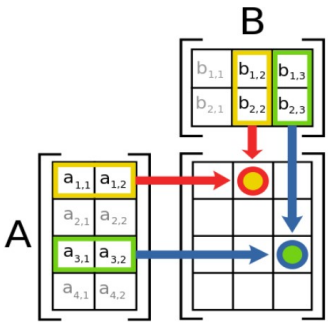
  $$\begin{pmatrix} 1 & 3 & 2 \\ 1 & 0 & 0 \\ 1 & 2 & 2 \end{pmatrix} + \begin{pmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \\ -2 & 1 & 1 \end{pmatrix} = \begin{pmatrix} 1+0 & 3+0 & 2+5 \\ 1+7 & 0+5 & 0+0 \\ 1+(-2) & 2+1 & 2+1 \end{pmatrix} = \begin{pmatrix} 1 & 3 & 7 \\ 8 & 5 & 0 \\ -1 & 3 & 3 \end{pmatrix}$$

- `multiply(matrix1, matrix2)` – Returnera en ny matris av rätt dimension som är produkten av `matrix1` och `matrix2`.

  Produkten *AB* av två matriser *A* och *B* är endast definierad om antalet kolumner i *A* är lika med antalet rader i *B*. Om *A* är en *m*×*n*-matris (*m* rader, *n* kolumner) och *B* en *p*×*q*-matris, är produkten *AB* endast definierad om *n* = *p* och produkten *BA* är endast definierad om *q* = *m*.

  Om *C* = *AB* gäller

  $$c_{i,j} = a_{i,1}b_{1,j} + a_{i,2}b_{2,j} + \cdots + a_{i,n}b_{n,j} = \sum_{r=1}^{n} a_{i,r}b_{r,j}$$

  Noterbart är att *AB* är en *m*×*q*-matris.

  Exempel:

  $$\begin{pmatrix} 1 & 0 & 2 \\ -1 & 3 & 1 \end{pmatrix} \times \begin{pmatrix} 3 & 1 \\ 2 & 1 \\ 1 & 0 \end{pmatrix} = \begin{pmatrix} (1\cdot 3 + 0\cdot 2 + 2\cdot 1) & (1\cdot 1 + 0\cdot 1 + 2\cdot 0) \\ (-1\cdot 3 + 3\cdot 2 + 1\cdot 1) & (-1\cdot 1 + 3\cdot 1 + 1\cdot 0) \end{pmatrix} = \begin{pmatrix} 5 & 1 \\ 4 & 2 \end{pmatrix}$$

- `map(matrix, fun)` – Returnera en ny matris där `fun` har applicerats på varje element i `matrix`.

Här följer några exempel på hur primitiverna ska fungera.

```
>>> m1 = [[1, 0, 2], [-1, 3, 1]]
>>> rows(m1)
2
>>> columns(m1)
3
>>> m2 = transpose(m1)
>>> m2
[[1, -1], [0, 3], [2, 1]]
>>> m3 = [[3, 1], [2, 1], [1, 0]]
>>> add(m2, m3)
[[4, 0], [2, 4], [3, 1]]
>>> multiply (m1, m3)
[[5, 1], [4, 2]]
>>> map(m1, lambda x: -x)
[[-1, 0, -2], [1, -3, -1]]
```