Instruktioner - Datortentamen TDDD73 Funktionell och imperativ programmering i Python TDDE24 Funktionell och imperativ programmering del 2

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.
- Den implementerade lösningen ska kunna köras inom rimliga tidsramar. Till exempel är en lösning som tar 1 minut för att konkatenera två strängar som är 4 bokstäver långa inte acceptabel.

Uppgifter - Datortentamen TDDD73 Funktionell och imperativ programmering i Python TDDE24 Funktionell och imperativ programmering del 2

Fredag 12 januari 2018 kl 8-13

Uppgift 1

Hantering av datum är en viktig funktion i många program. Ett sätt att representera ett datum är som en tupel bestående av månadsnamn (jan, feb, mar, apr ,maj, jun, jul, aug, sep, okt, nov, dec) och dag i månaden, t.ex. `('jan', 12)` eller `('dec', 24)`. Ett annat är som antal dagar sedan början av året. Skriv en funktion `day_of_year(dates, is_leap_year)` som returnerar antalet dagar sedan 1/1 där `date` är en tupel enligt ovan och `is_leap_year` är sant om och endast om det är skottår.

```
>>> day_of_year(('jan', 2), False)
2
>>> day_of_year(('mar', 10), False)
69
>>> day_of_year(('mar', 10), True)
70
```

Uppgift 2

Skriv en funktion som tar en lista med tal och returnerar en ny lista som bara innehåller de udda talen i listan. Funktionen ska finnas i två varianter: en som arbetar enligt en rekursiv modell (kallad `odd_r`) och en som arbetar enligt en iterativ modell (kallad `odd_i`). Du får inte använda inbyggda funktioner eller metoder som behandlar hela listor (`len` är ok). 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:

```
>>> odd_r([1, 2, 3, 4, 5])
[1, 3, 5]
>>> odd_i([2, 4, 6])
[]
```

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

Uppgift 3

Skriv en funktion `partition(seq)` som tar en lista som kan innehålla godtyckligt nästlade listor med strängar och returnerar en dictionary där nyckeln är ett heltal och värdet är en lista med alla strängar av den längden. Alla strängar av längd `n` ska alltså hamna i samma lista i dictionaryn. Ordningen i listorna i dictionaryn är godtycklig (du kan alltså få en annan ordning än i exemplen nedan). Funktionen ska vara icke-destruktiv. Det går bra att lösa uppgiften rekursivt eller iterativt. Tänk som vanligt på att dokumentera funktionen på lämpligt sätt.

```
>>> partition(['a', 'aa', 'b', 'ccc', 'dd'])
{1: ['a', 'b'], 2: ['aa', 'dd'], 3: ['ccc']}
>>> partition(['a', ['aa', ['b'], ['ccc', 'dd']]])
{1: ['a', 'b'], 2: ['aa', 'dd'], 3: ['ccc']}
```

Uppgift 4

Deluppgift 4a (3p)

Skriv en högre ordningens funktion `bind_1st(f, v)` som binder första argumentet i en binär funktion `f` till värdet `v` genom att returnera en ny funktion med ett argument som ger samma resultat som att räkna ut `f(v, x)`.

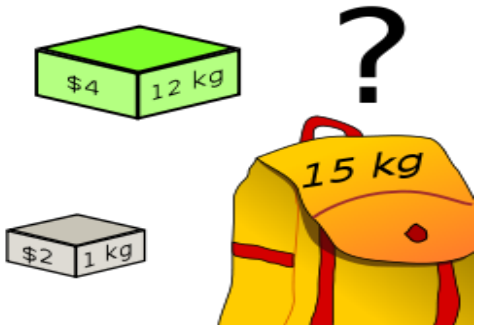
```
>>> def multiply(x, y):
...     return x*y
>>> f = bind_1st(multiply, 3)
>>> f(2)
6
```

Deluppgift 4b (2p)

Skriv ett uttryck som adderar 3 till varje tal i en lista `numbers` genom att använda `bind_1st`. Om `numbers` har värdet `[2, 1, 3]` så ska uttrycket ge `[5, 4, 6]`. Du kan anta att `numbers` är definierat och har ett värde innan uttrycket evalueras.

Uppgift 5

Kappsäcksproblemet är ett problem som dyker upp i väldigt många sammanhang. Ursprungligen handlade problemet om att saker med olika volym ska packas ner i en kappsäck med begränsat utrymme så att samtliga saker inte kan packas ner. Sakerna anses också ha olika värden för personen som ska ta med sig kappsäcken. Frågan är vilka saker som ska packas ned i kappsäcken så att värdet av de nedpackade sakerna blir som störst.



Anta att vi har en ryggsäck som rymmer upp till 15kg och att vi har 5 olika paket med olika värde och olika vikt enligt bilden. Om vi packar ner det gröna, blåa och gråa paketen så blir det en totalvikt på 15kg och ett totalvärde på $8. Om vi istället tar det gula, gråa, blåa och orangea paketen blir totalvikten 8kg och totalvärdet $15 vilket är betydligt bättre. I det här fallet finns det inget bättre alternativ.

Skriv en funktion `knapsack(things, max_weight)` som tar en lista med par (vikt, värde) och den maximala vikten och räknar ut hur stort värde som mest kan packas ned. Funktionen behöver inte vara minnes- eller tidseffektiv. Det är till exempel acceptabelt att lösningen tar mer än en minut att köra för nedanstående data.

```
>>> knapsack([(12,4), (2,2), (1,2), (1,1), (4,10)], 15)
15
>>> knapsack([(12,12), (2,2), (1,2), (1,1), (4,10)], 15)
16
>>> knapsack([(12,9), (2,2), (1,3), (1,2), (1,2), (4,10)], 15)
19
```

Uppgift 6

Databaser finns idag överallt. En databas består av ett antal tabeller. En tabell lagrar data som rader där varje rad har samma fält. Givet en databas vill man ställa frågor för att få ut relevant information. Ett av de viktigaste frågespråken för databaser är SQL. Ett enkelt exempel på en databas med två tabeller `student` och `kurs` visas i bilden nedan.

Namn	Klass	Kurs
Ada	U1	TDDE24
Bo	D1	TDDE24
My	D1	TDDE25

Kurs	Examinator
TDDE24	Peter
TDDE25	Fredrik

Din uppgift är dels att designa och implementera en abstrakt datatyp för tabeller och dels att använda datatypen för att implementera några vanliga SQL-operationer enligt följande beskrivning. Den abstrakta datatypen ger 2 poäng och implementationen av SQL-operationerna (funktionerna nedan) ger 1 poäng vardera. Tänk på att operationerna ska använda datatypen och att allt ska vara väl dokumenterat och väl testat. Enkla tester ska bifogas till lösningen.

- `select(table, pred)` – Returnera en ny tabell där varje rad finns i `table` och uppfyller predikatsfunktionen `pred` (funktionen tar en rad som indata och returnerar `True` eller `False`) . Exempel, `select` på tabellen `student` med en predikatsfunktion som ger `True` när värdet på kolumnen `Kurs` är TDDE24 borde ge

Namn	Klass	Kurs
Ada	U1	TDDE24
Bo	D1	TDDE24

- `project(table, columns)` – Returnera en ny tabell som bara innehåller de kolumner som finns i listan `columns`. Exempel, `project` på tabellen `student` med kolumnerna `Namn` och `Klass` borde ge

Namn	Klass
Ada	U1
Bo	D1
My	D1

- `join(table1, column1, table2, column2)` – Returnera en ny tabell där varje möjlig kombination av rader från `table1` och `table2` som uppfyller kravet att `column1` och `column2` har samma värde finns med. Exempel, `join` på kolumnen `Kurs` i tabellen `student` och kolumnen `Kurs` i tabellen `kurs` borde ge

Namn	Klass	Kurs	Kurs	Examinator
Ada	U1	TDDE24	TDDE24	Peter
Bo	D1	TDDE24	TDDE24	Peter
My	D1	TDDE25	TDDE25	Fredrik

Ett annat exempel, `join` på kolumnen `Kurs` i tabellen `kurs` och kolumnen `Kurs` i tabellen `student` borde ge

Kurs	Examinator	Namn	Klass	Kurs
TDDE24	Peter	Ada	U1	TDDE24
TDDE24	Peter	Bo	D1	TDDE24
TDDE25	Fredrik	My	D1	TDDE25