Tentamen för TDP002 Imperativ programmering
2010-08-25 08-12

Ola Leifler

Instruktioner:

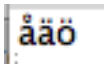
- Skriv svar på varje fråga i en separat fil i hemkatalogen (~) namngiven enligt mönstret `login-uppgift.(txt)|(py)` exempelvis `olale507-1.txt` eller `fregy123-2.py`.
- Uppge namn och personnummer i varje inlämnad fil.
- Uppgifterna löses enskilt.
- Kursboken *Learning Python* är tillåten.
- Kommunikation med andra under tentamenstillfället är förbjudet.

Fråga:	1	2	3	4	5	6	7	8	9	10	Totalt
Poäng:	3	1	3	2	1	3	3	4	3	3	26

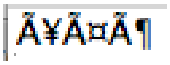
För betyg 3 gäller 50% rätt, för betyg 4 65% och för betyg 5 80%, avrundat till närmaste hela poäng.
Både rätt innehåll och välformulerad text krävs för full poäng på uppgifterna.
Det totala betyget på tentamen är detsamma som det lägsta betyget av teoridel och Pythondel. Det är lika många poäng på bägge delarna.
Ni får ut lösenord till tentasystemet för att göra Pythondelen av tentan efter att ni lämnat in era svar på teoridelen.

Teori

1. (3 poäng) Figurerna 1, 2 och 3 visar hur tre tecken kan se ut beroende på den binära representationen av dem i en fil och hur de visas i en texteditor. I bild 1 visas tecknen ”ääö” enligt representationen UTF-8 som också användes när filen skapades. I figur 2 har vi låtit texteditorn tolka samma fil enligt Latin-1 och i figur 3 har tecknen på representerats enligt teckenrepresentationen Latin-1 men visas enligt UTF-8. Förklara det utseende tecknen får i varje bild i termer av hur texteditorn hanterar att läsa in, spara ner och visa tecknen.



Figur 1: Tre tecken inlästa från en fil i UTF-8-format och representerade som UTF-8-tecken



Figur 2: Tre tecken inlästa från en fil i UTF-8-format och representerade som Latin-1-tecken

Figur 3: Tre tecken inlästa från en fil i Latin-1-format och representerade som UTF-8-tecken

2. (1 poäng) Förklara vad en stack är med ett exempel som använder en rekursiv funktion.
3. (3 poäng) Vad är en trädstruktur? Förklara genom att ge exempel på hur trädstrukturer kan användas för att representera både data och programkod.
4. (2 poäng) Förklara hur ett datorprogram exekveras av en dator som följer John von Neumanns arkitektur genom att använda begreppet *the fetch-execute cycle*.
5. (1 poäng) Vad är en sidoeffekt när man evaluerar en procedur i Python? Visa med ett kodexempel.
6. (3 poäng) I grammatiken nedan kan varje typ av uttryck (`u_expr`, `m_expr` och så vidare) reduceras till en annan typ av uttryck (`m_expr` kan reduceras till ett `u_expr`).
   1. Förklara exakt vad i grammatikens regler som gör detta möjligt.
   2. Ange hur `23 + 4 < 8 | false` kommer att tolkas enligt grammatiken. Ni får förkorta grenar i trädet om ni behöver så länge ni visar hur en av dem expanderas fullt ut.

```
digit ::= "0"..."9"

primary ::= ["-"] "1"..."9" digit* | "0" |true | false

power ::=
             primary ["**" u_expr]

u_expr ::=
             power | "-" u_expr
              | "+" u_expr | "\~" u_expr

m_expr ::=
```

```
              u_expr | m_expr "*" u_expr
               | m_expr "//" u_expr
               | m_expr "/" u_expr
                 | m_expr "\%" u_expr

a_expr ::=
              m_expr | a_expr "+" m_expr
               | a_expr "-" m_expr

shift_expr ::=
              a_expr
               | shift_expr ( "<<" | ">>" ) a_expr

and_expr ::=
              shift_expr | and_expr "&" shift_expr

xor_expr ::=
              and_expr | xor_expr "^" and_expr

or_expr ::=
              xor_expr | or_expr "|" xor_expr

comparison ::=
              or_expr ( comp_operator or_expr )*

comp_operator ::=
              "<" | ">" | "==" | ">=" | "<=" | "<>" | "!="
                 | "is" ["not"] | ["not"] "in"
```

Python

7. (3 poäng) I filen ~/given_files/primes.py finns funktionen `lcm` (*Least Common Multiplier*) som tar ett godtyckligt antal argument `args` och ska returnera den minsta gemensamma multipeln för talen (det minsta talet som är jämt delbart med alla tal `args`). Det finns dock fel dolda som ni måste rätta till för att få funktionen och dess hjälpfunktioner `prime_factorize` och `merge_lists` att fungera som de ska. Rätta felen och visa att funktionen `lcm` fungerar som avsett med testfall.

8. (4 poäng) Skriv funktioner för att representera datat i filen ~/given_files/cpu.with.vendor.arff som en abstrakt datatyp *CPUs*. Alla attribut som anges i början av filen och som sedan finns med som fält på varje rad med data ska finnas representerade i er datatyp. För full poäng får ni inte hårdkoda fälten som anges i början av filen utan ska läsa in dem som en datastruktur. Ni ska också skapa en *sökfunktion* för att presentera alla CPU-objekt där något eller några av attributen har de värden som ges till sökfunktionen. Sökfunktionen ska ta som ett av sina argument en dictionary med nycklar som motsvarar attribut och värden som motsvarar det man söker efter. För full poäng ska man inte bara kunna ange konstanta värden som sökkriterier utan intervall av värden för numeriska attribut.

9. (3 poäng) Skapa en function `accumulate` med tre parametrar: en funktion `fn`, en sekvens `seq` och ett startvärde `value`. Funktionen `fn` ska ha två parametrar: ett tidigare resultat `result` och ett värde `x` som tillsammans ska generera ett nytt resultat som funktionen `fn` returnerar. Funktionen `accumulate` ska returnera resultatet av att successivt applicera `fn` på element i `seq` och det ackumulerade resultatet från att applicera `fn`. Värdet `value` anges som det första värdet på parametern `result` till `fn`.

   Skapa sedan funktionerna `sum` och `product` med hjälp av *lambda-uttryck* för att summera respektive beräkna produkten av elementen i en sekvens med hjälp av `accumulate`.

10. (3 poäng) Gör en funktion som fungerar som översättare mellan tal och räkneord. Funktionen ska översätta tal mellan 0 och 99 till motsvarande svenska räkneord. Med argumentet `17` ska funktionen returnera `"sjutton"`. För full poäng krävs att du tydligt anger hur funktionen skulle kunna generaliseras till godtyckligt stora tal samt att den inte enbart använder sig av en tabell för översättning utan utnyttjar regelbundenheter i svenska språket.