Tentamen för TDP002 Imperativ programmering - teoridel
2009-10-22 08-12

Instruktioner:

- Skriv varje svar på ett separat papper, ej på uppgiftspappret.
- Uppge namn och personnummer på varje inlämnat papper.
- Uppgifterna löses enskilt.
- Inga hjälpmedel är tillåtna.
- Kommunikation med andra under tentamenstillfället är strikt förbjudet.

Fråga:	1	2	3	4	5	6	7	Totalt
Poäng:	2	2	2	2	1	1	2	12

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 problemdel.

1. (2 poäng) Vad är flaskhalsen i en von Neumann-dator och varför är det viktigt att känna till den?

Lösningsförslag: Se sidan 47 i PL-boken

2. (2 poäng) Förklara vad en interpretator, kompilator och hybridmodell innebär för körning av program. Vilken modell använder Python vanligen för att köra program?

Lösningsförslag: Se sidorna 25–26 i LP-boken och föreläsning 7. En interpretator läser källkod och genererar direkt resultat jämfört med en kompilator som läser källkod och genererar maskinkod. En hybridmodell innebär att man genererar bytekod som är en binär, effektiv representation av programmet som kan köras av en virtuell maskin (bytekodsinterpretator).

Python använder vanligen en hybridmodell för exekvering (interpretator är också ok som svar).

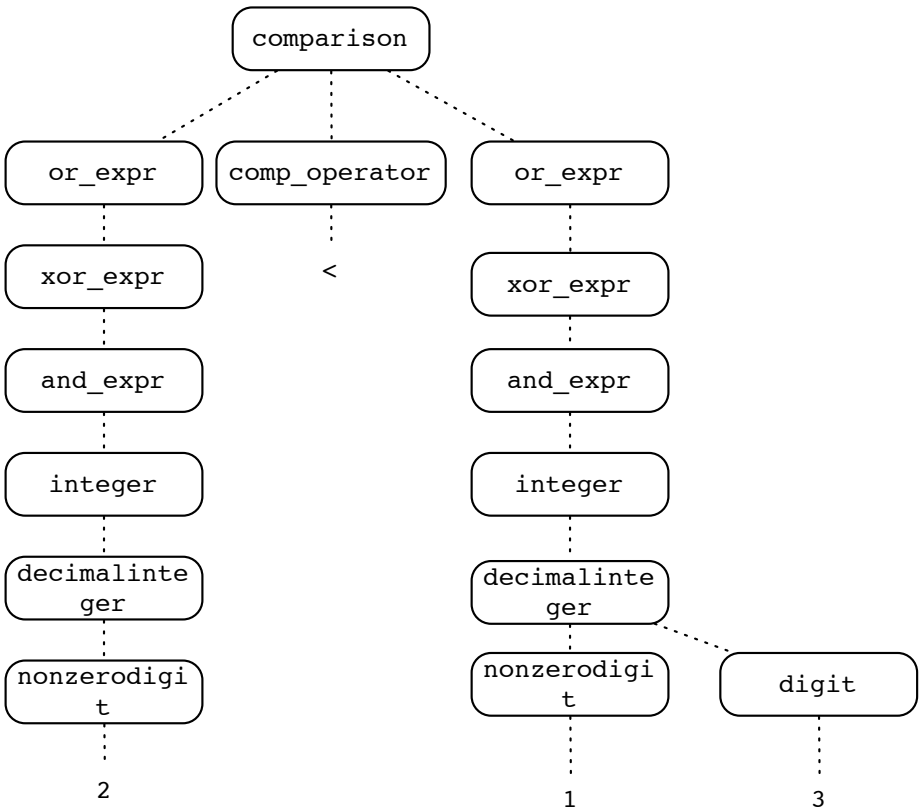
3. (2 poäng) Ange skillnaden mellan en funktion och en procedur och ge exempel på båda i Python.

Lösningsförslag: Se bilderna till föreläsning 7

4. (2 poäng) Visa vilket parse-träd grammatiken nedan ger för uttrycket

$$2 < 13$$

Lösningsförslag:



Observera att regeln för `digit` saknades, men det går utmärkt att hänvisa till den (med den uppenbara expansionen ”0”…”9”) eller `nonzerodigit` istället.

```
integer ::=
            decimalinteger | octinteger | hexinteger
decimalinteger ::=
            nonzerodigit digit* | "0"
octinteger ::=
            "0" octdigit+
hexinteger ::=
            "0" ("x" | "X") hexdigit+
nonzerodigit ::=
            "1"..."9"
and_expr ::=
            integer | 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"
```

5. (1 poäng) Förklara *operator precedence* med hjälp av grammatiken ovan.

Lösningsförslag: *Operator precedence* anger ordningen i vilken operatorer (aritmetiska, booleska) appliceras. I grammatiken ovan är regler som expanderas sist och därmed utgör de nedersta delarna i trädet de som kommer evalueras först och således är associerade med operatorer med högst prioritet. I grammatiken framgår ordningen `and_expr > xor_expr > or_expr`, det vill säga `&` har företräde framför `^` som i sin tur har företräde framför `|`. Exempelvis skulle `1 | 2 & 5 ^ 1` tolkas som `(1 | (2 & 5) ^ 1))`.

6. (1 poäng) Ange talet 122 på hexadecimal form.

Lösningsförslag: 7A

7. (2 poäng) Vad är likheterna och skillnaderna mellan `search1` och `search2` i figur 1? Svara på frågan genom att ange vilken sorts algoritmer de utgör exempel på, vilken indata de accepterar respektive vad de returnerar.

Lösningsförslag: Båda är sökalgoritmer som tar sekvenser `seq` (listor, tupler eller strängar) och element `x` att söka efter som argument. `search1` kräver endast att man ska kunna jämföra dessa element med avseende på == medan `search2` kräver att man också ska kunna applicera < och > på `x` och att `seq` är sorterad , det vill säga ordnad så att `seq[i] <= seq[i+1]` gäller för alla

```
def search1(x, seq):
    for index in range(len(seq)):
        if x == seq[index]:
            return index
     return -1

def search2(x, seq):
    low = 0
     high = len(seq) - 1
     while low <= high :
         mid = (low + high) / 2
        if seq[mid] < x:
             low = mid + 1
         elif seq[mid] > x:
             high = mid - 1
         else:
            return mid;
     return -1;
```

Figur 1: Sökalgoritmer

i. `search2` är en binärsökning som successivt delar upp sekvensen i halvor och kräver att `x` ska återfinnas på den plats i sekvensen där tal som är lika stora som `x` förekommer. Båda funktionerna returnerar index för positionen i `seq` där `x` återfinns eller -1 om `x` inte hittas.