TDP002 - Datortenta (DAT1)

2013-01-11

Regler

- Student får lämna salen tidigast en timme efter tentans start.
- Vid toalettbesök eller rökpaus ska pauslista utanför salen fyllas i.
- All form av kontakt mellan studenter under tentans gång är strängt förbjuden.
- Böcker och anteckningssidor kan komma att granskas av tentavakt i samband med tentans start samt under tentans gång.
- Frågor om specifika uppgifter eller om tentan i stort ska ställas via tentasystemet.
- Systemfrågor kan ställas till assistent i sal.
- Ingen uppgift rättas efter tentatidens slut.
- Ingen kompletteringsmöjlighet ges de sista tio minuterna.
- Så länge en praktisk uppgift inte har betyget U eller tilldelats maxpoäng för uppgiften kan den kompletteras till högre poäng.
- En teoretisk uppgift kan endast kompletteras efter förfrågan från rättare.
- Inga elektroniska hjälpmedel får medtas. Mobiltelefon ska vara avstängd och ligga i jacka eller väska.
- Inga ytterkläder eller väskor får förvaras vid skrivplatsen.

Antal uppgifter	10
Totalt antal poäng	20
Hjälpmedel	En pythonbok (tex Learning Python 4 ed.) En A4-sida med egna anteckningar

VÄND!

Betygssättning

Tentan är uppdelad i två delar, en praktisk och en teoretisk. Delarna är värda 10 poäng vardera och för godkänt betyg krävs minst fyra poäng från den praktiska delen. Poäng som krävs för de olika betygen kan ses i tabell 1.

Poängsumma	**Betyg**
1 – 9	U
10 – 12	3
13 – 15	4
16 – 20	5

Tabell 1: Poängfördelning för betygssättning

Bonus från labserien

Bonus från labserien ges endast vid första ordinarie tillfället i kursen och därmed ges ingen bonus under detta tillfälle.

Information

Inloggning

Logga in på tentakontot med följande användaruppgifter:

```
Användarnamn: examx
Lösenord:     kluring1
```

Följ menyvalen så långt det går tills du ska mata in ett engångslösenord. Tag fram ditt LiU-kort och visa det för tentavakten för att få detta lösenord.

När du är inloggad är det viktigt att du startar tentaklienten genom att högerklicka på bakgrunden och välja “tentaklient” i menyn.

Avslutning

Tryck på knappen märkt “exit” i menyn längst nere på skärmen och välj ok. Vänta ett tag och tryck sedan på knappen “Avsluta tentamen” när det är möjligt. När detta är gjort är det omöjligt att logga in igen.

Praktisk del

1. På filen *given_files/calculator.py* finns given kod för att bygga en enkel miniräknare. Modifiera koden så att programmet fungerar enligt nedanstående körexempel: [1 p]

Körexempel 1:

```
Välkommen till minräknaren!
Välj räknesätt (m/a/s/d): m
Mata in faktor ett: 6
Mata in faktor två: 2
Produkten av 6 och 2 är 12
```

Körexempel 2:

```
Välkommen till minräknaren!
Välj räknesätt (m/a/s/d): d
Mata in täljare: 5
Mata in nämnare: 2
Kvoten mellan 5 och 2 är 2
```

Svar:

```
#!/usr/bin/env python3
#-*- coding: latin-1 -*-

funs = {'m' : lambda a,b: a*b,
        'a' : lambda a,b: a+b,
        's' : lambda a,b: a-b,
        'd' : lambda a,b: a//b}

promt = {'m' : ('faktor ett', 'faktor två'),
         'a' : ('term ett', 'term två'),
         's' : ('minuend', 'subtrahend'),
         'd' : ('täljare', 'nämnare')}

ans_text = {'m' : 'Produkten av',
            'a' : 'Summan av',
            's' : 'Differensen mellan',
            'd' : 'Kvoten mellan'}

print('Välkommen till minräknaren!')
c=input('Välj räknesätt (m/a/s/d): ')

a=int(input('Mata in {}: '.format(promt[c][0])))
b=int(input('Mata in {}: '.format(promt[c][1])))

#endast följande rad är ny:
print('{} {:d} och {:d} är {:d}'.format(ans_text[c], a, b,
                                        funs[c](a,b)))
```

2. På filen *given_files/power_table.py* finns ett program som skriver ut en tabell över ett antal potenser. Din uppgift är att i filen *power.py* skapa den rekursiva funktionen *power* som ska beräkna x^N för heltalen x och N. Du behöver endast skicka in filen power.py för rättning. Tänk på att $x^{-N} = \frac{1}{x^N}$ [2 p]

Svar:

Ett poäng för $N \geq 0$, ett till om den även klarar negativa tal.

```
def power(x,N):
    if N == 0: return 1

    if N < 0:
        return 1/power(x,-N)

    return x*power(x,N-1)
```

3. Frank Benford var en fysiker som 1935 gjorde en intressant upptäckt. Eftersom datorer inte var så vanligt vid den tiden använde man långa tabeller med tal för att ta reda på 10-logaritmen av ett tal. Frank såg att de första sidorna (logaritmen för tal som börjar med siffran ett) var betydligt mer slitna än sidorna i slutet av boken. Eftersom boken användes av honom och alla hans kollegor drog han slutsatsen att tal oftare inleds med en etta än en nia. Ska man vara ärlig så gjorde Simon Newcomb (astronom född 1835) exakt samma iakttagelse redan år 1881 men den gick obemärkt förbi. Över flera år samlade Benford in data för att bevisa sin teori och idag kallar man den Benfords lag. Benfords lag säger att sannolikheten att ett mätvärde inleds med siffran i, $F(i)$, beskrivs av ekvation 1 sålänge inte värdet är påhittat eller slumpat med någon sorts begränsning (t.ex. ett telefonnummer eller lottoresultat). [2 p]

$$F(i) = log_{10}\left(1 + \frac{1}{i}\right) \quad (1)$$

Många statistiska data följer Benfords lag och man kan därför använda formeln för att granska olika data såsom taxeringsvärden och forskningsdata.

Din uppgift är nu att skapa en funktion *random* i filen *benford.py* som slumpar ett fyrasiffrigt tal enligt följande algoritm:

1. Skapa en lista *weights* som på index 0 tilldelas värdet 0 och på index i innehåller talet $F(i)$ för talen $[1, 9]$ enligt ovan.
2. Skapa en ny lista *accumulated* som på index i innehåller summan av alla tal till och med index i i weights
3. Slumpa ett tal x i intervallet $[1, 10000[$ och dividera det med 10000
4. Slumpa ett nytt värde *last* i intervallet $[100, 1000[$. Detta ska motsvara de sista tre siffrorna i det genererade talet.
5. Den första siffran för det genererade talet, $first$, är det första index i i accumulated som uppfyller `accumulated[i] >= x`
6. returnera svaret $1000 * first + last$

Punkt 1 och 2 kan göras utanför funktionen för ökad effektivitet.

I filen *given_files/test_benford.py* finns ett program som testar din funktion. Du kan öka variabeln *num* för att få noggrannare beräkningar.

För er som inte är vana med matematiska intervall betyder en hake vänd från ett tal att det talet inte ingår i intervallet. [1,4] betyder då talen 1,2,3,4 och]1,5[betyder talen 2,3,4.

Svar:

```
from math import log10 as log
from random import randrange

weight = [0,] + [log(1+1/i) for i in range(1,10)]
accumulated = [sum(weight[:i+1]) for i in range(10)]

def random():
    x = randrange(1,10000)/10000
    last = randrange(100,1000)
    for i,n in enumerate(accumulated):
        if n >= x:
            first = i
            break
    return first * 1000 + last
```

4. I UNIX-system finns ett verktyg som heter grep. grep tar emot ett ord och en eller flera filnamn som kommandoradsargument och skriver ut de rader i de givna filerna där det eftersökta ordet finns med. Din uppgift är nu att göra en egen variant av grep som även skriver ut radnummret för den matchande raden enligt körexempel 1 [2p]. För full pott ska programmet även klara en förenklad version av regulära uttryck enligt följande beskrivning (se även körexempel 2 och 3): [5 p]

   - * matchar noll eller flera tecken
   - ? matchar ett tecken
   - Övriga tecken matchar endast sig självt

   Det är givet att det aldrig förekommer flera asterisker efter varandra men ett frågetecken kan följa ett annat frågetecken och asterisker och frågetecken kan blandas.

   Körexempel 1:

```
zaza12 <1> ./grep.py JOHN surnames.txt boy_names.txt
surnames.txt:2: JOHNSON
surnames.txt:214: JOHNSTON
surnames.txt:614: JOHNS

boy_names.txt:100: JOHN
```

   Körexempel 2:

```
zaza12 <2> ./grep.py 'G*N?' surnames.txt
surnames.txt:38: GONZALEZ
surnames.txt:94: GONZALES
<vissa rader utelämnade>
surnames.txt:674: GENTRY
surnames.txt:960: ENGLAND
surnames.txt:968: SARGENT
```

   Körexempel 3:

```
zaza12 <3> ./grep.py 'J*N?' surnames.txt boy_names.txt girl_names.txt
surnames.txt:2: JOHNSON
surnames.txt:4: JONES
surnames.txt:83: JENKINS
<vissa rader utelämnade>
surnames.txt:614: JOHNS
surnames.txt:831: JOYNER

boy_names.txt:67: JENSON

girl_names.txt:42: JASMINE
```

   Tips: Tänk på att raderna i filerna avslutas med nyradstecken, dessa ska inte vara med vid matchningen.
   Tips 2: Körexempel 2 och 3 har apostrofer runt söktermen. Detta för att terminalen inte ska försöka tolka ordet utan skicka in det som det står till ditt program.

Svar:

Bifogat finns två lösningsmodeller för matchningen, en rekursiv och en interativ.

```
#!/usr/bin/env python3
#-*- coding: latin-1 -*-
import sys

def match(term, line, skip=True):
    if len(line) == 0: return False

    if len(term) == 1:
        if term == '*': return True
        if term == '?': return len(line) > 0
        if skip: return term in line
        return term == line[0]

    if term[0] == '*':
        return match(term[1:], line, True)
    elif term[0] == '?':
        return match(term[1:], line[1:], False)

    if not skip:
        return line[0] == term[0] and match(term[1:], line[1:], False)

    for i,c in enumerate(line):
        if c == term[0] and match(term[1:], line[i+1:], False):
            return True
    return False


def match_iter(term, line):
    skip = True
    index = 0
    for c in line:
        if not skip:
            if c == term[index] or term[index] in '*?':
                if term[index] == '*':
                    skip = True
                index += 1
            else:
                skip = True
                index = 0

        if skip and index < len(term):
            if c == term[index]:
                index += 1
                skip = False

        if index == len(term):
            return True

    return False


if len(sys.argv) < 3:
```

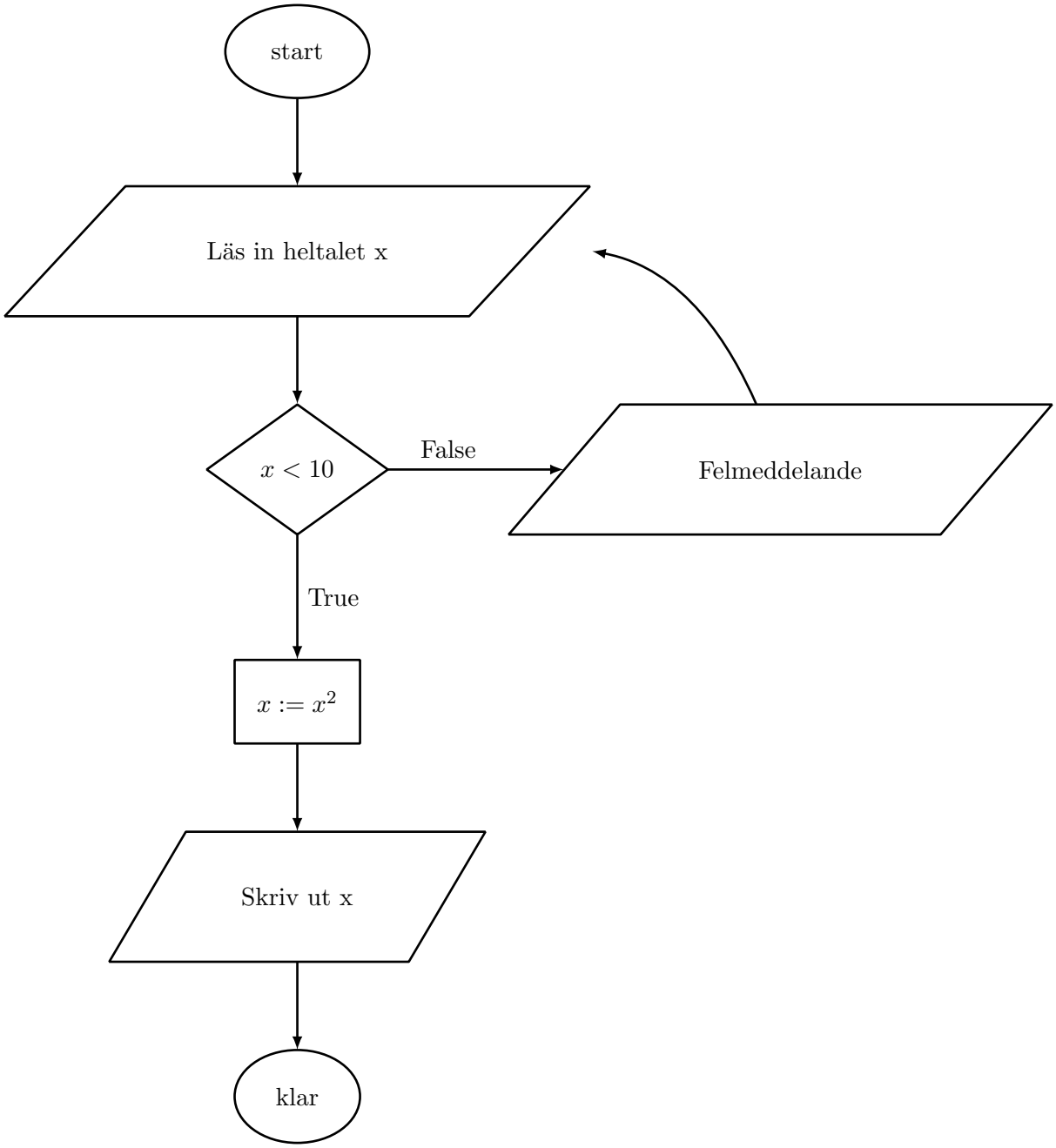
```
    print('grep kräver minst två argument!')
    sys.exit()

term, *files = sys.argv[1:]

for file_name in files:
    with open(file_name) as f:
        for row,line in enumerate(f,1):
            if match(term, line.strip()):  # "term in line" grundlösning
                print('{}:{:d}: {}'.format(file_name,row,line), end='')
    print()
```

Teoretisk del

5. Beskriv på ren svenska vad programmet beskrivet med flödesschemat i figur 1 gör. [2 p]



Figur 1: Flödesschema till uppgift 5

Svar:

Programmet ber om användaren om ett heltal så länge det är mindre än tio och skriver sedan ut kvadraten av det inmatade heltalet. Om användaren matar in ett felaktigt tal skriver programmet ut ett felmeddelande och ber om ny inmatning.

6. Vad menas med kortslutande beräkning av logiska uttryck (short-circuit evaluation)? [1 p]

Svar:

Att resultatet av ett uttryck kan beräknas utan att beräkna alla ingående delar, t.ex. kan uttrycket `A AND B` sägs vara falskt om A är falskt oberoende av värdet på B. (Se även Concepts of Programming Languages 7.6)

7. Vad betyder DRY-regeln och varför bör man ta hänsyn till den när man programmerar? [2 p]

Svar:

DRY, eller Don't Repeat Yourself, förespråkar att varje bit av information (t.ex. ledtexter, namn, inställningar osv) endast ska förekomma på en plats i programmet. Detta gör koden mer hanterbar då den blir lättare att både läsa och modifiera (det är t.ex. lättare att förstå "WINDOW_SIZE_X" än "800").

8. Vad är skillnaden på positionella- och nyckelordsargument? (positional / keyword arguments) [1 p]

Svar:

positionella argument är vanligaste sättet att göra ett anrop till ett underprogram, första argumentet binds till första parametern, andra argumentet till den andra parametern osv. Vissa språk (däribland Python) har även nyckelordsargument där man kan specificera vilken parameter man vill binda argumentet till med hjälp av det formella parameternamnet. (se Concepts of Programming Languages 9.2.3)

9. När beräknas värdet på ett defaultargument i Python? [1 p]

Svar:

I samband med definitionen (se slide 8 fö 3 för exempel).

10. Du ska för varje uttryck 1-7 para ihop uttrycket med den icketerminal a-g som enligt den utdelade grammatiken bäst beskriver uttrycket i sin helhet. 2-3 rätta svar ger ett poäng, 4-5 två poäng och 6-7 ger 3 poäng. [3 p]

1. `4+3`	a. `exponentfloat`
2. `0B0101`	b. `a_expr`
3. `[1, 4, 6]`	c. `fraction`
4. `2e0`	d. `bininteger`
5. `.24`	e. `u_expr`
6. `5`	f. `list_display`
7. `-5`	g. `decimalinteger`

Svar:

1. b
2. d
3. f
4. a
5. c
6. g
7. e