Tentamen på programmeringsteknik II 2021-12-17

Skrivtid: 08:00 – 13:00

Observera: Denna skrivning är avsedd för de studenter som läst kursen ht 2020 eller senare.

Praktiska detaljer och hjälp under tentamen:

- Under stor del av tentan kommer lärare att finnas tillgängliga i Zoom: https://uu-se.zoom.us/j/61112733698. Du får gärna vara i ett breakout rum under tentan som under en vanlig lektion, men det är inte ett krav.

  Ställ inga frågor i ”huvudrummet”, utan gå till ett breakout room och skriv upp dig i dokumentet som beskrivs i punkten nedan.

- Ha Google-dokumentet
  https://docs.google.com/spreadsheets/d/1zggSOID0L4Tr7hKbAS4Sb_3D70nedfp9D_tm7fmfZy8
  öppet under tentans gång. Där kommer vi skriva evenutella rättelser och förtydliganden. Större förändringar görs även med annonsering i Studium. Google-dokumentet kan även användas för att kontakta lärare. Fyll i namn och nummer på ett breakout rum så kommer vi dit.

- Det går också att nå lärarna per email (sven-erik.ekstrom@it.uu.se och tom.smedsaas@it.uu.se)

Inlämning:

- Inlämning av tentan sker genom uppladdning av Python-filerna på samma ställe där skrivningen hämtades (på samma sätt som en vanlig inlämningsuppgift).

- Tentan består av A- och B-uppgifter. A-uppgifter måste fungera (inlämnade program ska kunna köras och lösa uppgiften) för att godkännas. B-uppgifter kan ge ”poäng” även om de inte löser problemet fullständigt.

- All inlämnad kod måste kunna köras direkt även på andra datorer än din.

- Gör en zip-fil som innehåller alla dina filer och ladda upp den. Om du inte vet hur du gör zip-filer kan du ladda upp filerna direkt (helst i en uppladdning). Det går att ladda upp filer flera gånger och då är det den senaste versionen som gäller.
  Glöm inte bort att klicka ‘’Skicka uppgift”!

- Om det blir problem med STUDIUM kan ni e-maila filerna till sven-erik.ekstrom@it.uu.se. Du måste ange din anmälningskod i brevet.

- **Inga inlämningar efter 13:00 kommer att accepteras!**

Regler

- Du får inte samarbeta med någon annan under tentan. Du får inte kopiera kod eller text utan måste skriva dina svar själv.
- Du får använda nätet.
- Du ska skriva dina lösningar *på anvisade platser* i filerna `m1.py`, `m2.py`, `m3.py` och `m4.py`. Du måste behålla namn på filer, klasser, metoder och funktioner. Du måste också ha exakt de parametrar som är angivna i uppgiften.
- Du får inte använda andra paket än de som redan är importerade i filerna såvida inte annat sägs i uppgiften. (Att ett paket finns importerat i betyder *inte* att det behöver användas!)
- Innan din tentamen blir godkänd kan du behöva förklara och motivera dina svar muntligt för lärare.

OBSERVERA ATT VI ÄR SKYLDIGA ATT ANMÄLA VARJE
MISSTANKE OM OTILLÅTET SAMARBETE ELLER
KOPIERING SOM MÖJLIGT FÖRSÖK TILL FUSK!

Tentamens beståndsdelar:

Tentamen är indelad i fyra sektioner var och en med uppgifter svarande mot de fyra modulerna. Det finns A-uppgifter och B-uppgifter i alla sektionerna.

Betygskrav:

3: Minst åtta A-uppgifter godkända.

4: Minst åtta A-uppgifter godkända samt antingen två B-uppgifter stor sett korrekt eller den frivilliga inlämningsuppgiften.

5: Minst åtta A-uppgifter godkända samt antingen alla B-uppgifter eller tre B-uppgifter och den frivilliga inlämningsuppgiften.

Uppgifter i anslutning till modul 1

Lösningen till dessa uppgifter ska skrivas på anvisade platser i filen `m1.py`. Filen innehåller också en `main`-funktion demonstrerar kodens funktion.

A1: Skriv funktionen `number_of_negative(lst)` som returnerar antalet negativa värden på samtliga nivåer i `lst`. Funktionen ska hantera sublistor med *rekursion*. Du kan förutsätta att alla listor innehåller tal.

Exempel:
`number_of_negative(1)` = 0
`number_of_negative(-1)` = 1
`number_of_negative([2, 0]` = 0
`number_of_negative([1,-2,[[-1],[1,-2]]])` = 3
`number_of_negative([1,2,[[1],[-1,-2,-3,-4]]])` = 4

A2: Nedanstående funktion ordnar om listan som den får som parameter.

```
1 def s_sort(aList):
2     for i in range(len(aList)-1):
3         m = i
4         for j in range(i+1,len(aList)):
5             if aList[j] < aList[m]:
6                 m = j
7         aList[i], aList[m] = aList[m], aList[i]
```

Hur beror tiden på längden n av listan? Svara med ett Θ-uttryck.

Du kan antingen motivera svaret med ett teoretiskt resonemang utifrån vad koden gör eller genom systematisk tidmätning.

Tips: Den inkluderade funktionen `random_list` kan vara bra att använda om du vill göra praktiska försök.

Skriv svaret på anvisad direkt efter funktionen. Om du motiverar med tidmätningar så måste du skriva koden du har använt i `main`-funktionen samt de mätresultat du använder för att motivera svaret.

B1: Hur beror tiden för nedanstående funktion av parametern `n`?

```
def foo(n):
    result = 1
    for i in range(n, n*n*n):
        result += 10*n + (n+2)*n
    return result
```

Svara med ett Θ-uttryck. Du måste styrka ditt svar med tidmätning på körningar.

Uppskatta också hur lång tid det skulle ta att beräkna `foo(5000)` och `foo(10000)` på din dator.

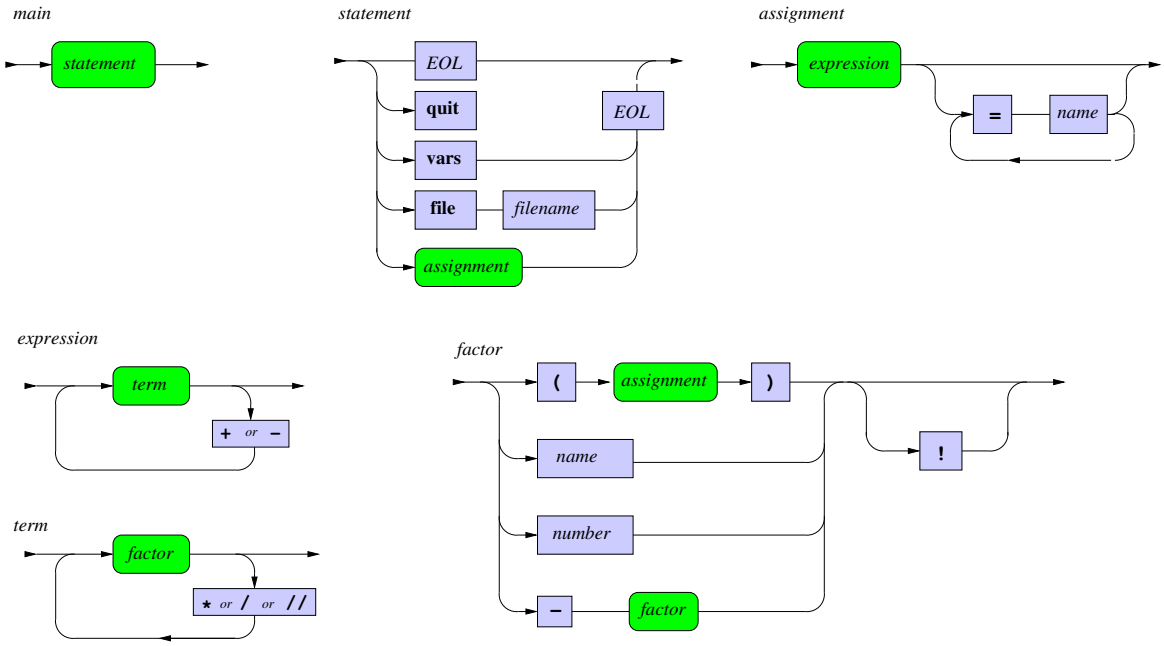
Skriv koden du använder för detta i `main`-funktionen och lägg utskrifterna på anvisad plats i slutet av koden.

Observera: Det ska gå att köra denna kod på annan dator!

Uppgifter i anslutning till modul 2

Den nedladdade koden `m2.py` implementerar en kalkylator liknande den i den andra obligatoriska uppgiften. Några saker är borttagna och några ska läggas till. Observera att filen också innehåller klassen `TokenizeWrapper`.

Syntaxen för uttrycken beskrivs av följande diagram:



I den givna koden finns alla funktioner (gröna rutor) men alla fungerar inte helt som de ska. Uppgifterna består alltså av att komplettera eller modifiera funktionerna och, eventuellt, lägga till några hjälpfunktioner.

Kommandot `file` utan parameter läser från filen `m2_test.txt` (som du laddade ner). Filen börjar så här:

```
1 - (2 - 4) = x      # 3.0
x*4/2 + x            # 9.0
(1=x) + (2=y)*(3=z)  # 7.0
```

Det förväntade resultatet står efter kommentartecknet `#`.

I resten av `m2_test.txt` ligger testfall för uppgifterna som alltså inte fungerar på rätt sätt innan du löst uppgifterna.

A3: Den givna koden fungerar inte för unärt minus. Här är exempel (från filen `m2_test.txt`) på hur det *ska* fungera:

```
Input : file
File  : #### A3: Unary minus
File  :
File  : -3                    # -3.0
Result: -3.0
File  : ----3                 # 3.0
Result: 3.0
File  : -(3 - 2*3)            # 3.0
Result: 3.0
File  : -3 - 4 + --10 = x     # 3.0
Result: 3.0
File  : -(x+2*-x)             # 3.0
Result: 3.0
```

Syntaxen framgår av syntaxdiagrammen.

A4: För in operatorn `//` för *heltalsdivision*.

Exempel (från filen `m2_test.txt`):

```
File  : #### A4: The //-operator
File  :
File  : 5//2                  # 2.0
Result: 2.0
File  : (5+3)//3              # 2.0
Result: 2.0
File  : (5+3)//3*5//3         # 3.0
Result: 3.0
File  : 5//2.5                # 2.0
Result: 2.0
File  : 5//2.6                # 1.0
Result: 1.0
File  : 5.2//2.6              # 2.0
Result: 2.0
File  : 2///3                 # Syntax error
*** Syntax error:  Expected number, word, '-' or '('
Error occurred at '/' just after '//'
File  : 4//(2*3-6)*0.5        # Evaluation error
*** Evaluation error:  Division by zero
```

Division med 0 ska ge ett evalueringsfel.

Tips 1: Tokenizern returnerar sekvensen `13//3` som tre tokens: `'13'`, `'//'` och `'3'`.

Tips 2: Operatorn ska fungera precis som Pythons `//`-operator.

B2: Implementera fakultetsoperatorn `!`. Denna operator står alltså *efter* sin operand. Se syntaxdiagrammen!

Exempel från filen `m2_test.txt`:

```
File  : #### B2 Factorial operator
File  :
File  : 0!                    # 1.0
Result: 1
File  : 1!                    # 1.0
Result: 1
File  : 2!                    # 2.0
Result: 2
File  : 3!                    # 6.0
Result: 6
File  : 5!                    # 120.0
Result: 120
File  : 5+ 5! - 10*3!*2       # 5.0
Result: 5.0
File  : 20!                   # 2432902008176640000
Result: 2432902008176640000
File  : 30!/20!               # 109027350432000.0
Result: 109027350432000.0
File  : 51!/50!               # 51.0
Result: 51.0
File  : 10!!                  # Syntax error
*** Syntax error:  Expected end of line or an operator
Error occurred at '!' just after '!'
File  : -3!                   # -6.0
Result: -6
File  : (-3)!                 # Evaluation error
*** Evaluation error:  Argument to fac is -3.0. Must integer >= 0
File  : 1.4!                  # Evaluation error
*** Evaluation error:  Argument to fac is 1.4. Must integer >= 0
```

Uppgifter i anslutning till modul 3

I detta avsnitt ska du arbeta med filen `m3.py` som innehåller klasserna `LinkedList` och `BST`.

Det finns också en `main`-funktion med några demonstrationskörningar. Detta är inte heltäckande tester utan du bör själv skriva fler i `main`!

Klassen `LinkedList.py` innehåller kod för att hantera *länkade listor* av objekt. Listorna är *inte* sorterade och samma värde kan förekomma flera gånger. I exemplen används heltal som data men koden ska fungera för alla typer av objekt.

Klassen `BST` innehåller kod för vanliga binära sökträd.

A5: I den givna koden för klassen `LinkedList` finns en metod `append` som lägger in ett nytt element sist i listan:

```
    def append(self, x):
        if self.first == None:
            self.first = self.Node(x)
        else:
            node = self.first
            while node.succ:
                node = node.succ
            node.succ = self.Node(x)
```

Denna metod måste börja från början och gå igenom hela listan för att hitta sista element. Bättre vore om klassen `LinkedList` förutom `first` också får en instansvariabel `last` som håller reda på sista elementet i listan.

Lägg till den instansvariabeln i `__init__` och skriv metoden `append_better` så att den utnyttjar och underhåller `last`-referensen och därmed får en väsentligt bättre komplexitet ($\Theta(1)$ i stället för $\Theta(n)$).

A6: Vad har nedanstående funktion för genomsnittlig tidskomplexitet?

```
def random_tree(n):
    t = BST()
    for x in range(n):
        t.insert(random.random())
    return t
```

Svara med av typen $\Theta(f(n))$! Motivera!

A7: Skriv metoden `epl` i klassen BST som beräknar och returnerar den *externa väglängden* (se kursmaterialet för definition!)

Exempel:

```
    t = BST()
    print(f'{t.epl()} should be 1')
    t = BST([1])
    print(f'{t.epl()} should be 4')
    t = BST([1, 2])
    print(f'{t.epl()} should be 10')
    t = BST([1, 2, 3])
    print(f'{t.epl()} should be 13')
    t = BST([2, 1, 3])
    print(f'{t.epl()} should be 12')
    t = BST([2, 1, 3, 4])
    print(f'{t.epl()} should be 17')
```

Utskrift:

```

1 should be 1

4 should be 4

8 should be 8

13 should be 13

12 should be 12

17 should be 17
```

A8: Implementera operatorn `==` för klassen `BST` som ska returnera `True` om de två operanderna innehåller precis samma nycklar (oavsett trädens struktur), annars `False`.

Exempel:

```
t1 = BST([1,2,3])
t2 = BST([2,1,3])
t3 = BST([3,1,2,4])
t4 = BST([1,2,3,4])
print(f'{t1} == {t2} : {t1==t2}')
print(f'{t1} == {t3} : {t1==t3}')
print(f'{t3} == {t4} : {t3==t4}')
```

Utskrift:

```
<1, 2, 3> == <1, 2, 3> : True
<1, 2, 3> == <1, 2, 3, 4> : False
<1, 2, 3, 4> == <1, 2, 3, 4> : True
```

Tips: Vid lösandet av denna uppgift kan du förutsätta att två träd är lika (`==`) om och endast om deras respektive utskrift är identiska. (Se dock uppgift B3 nedan.)

B3: Påståendet i uppgiften ovan är inte sant om nycklarna är strängar:

Exempel:

```
t5 = BST([1,2])
t6 = BST(['1, 2'])
print(f't5 : {t5}, t5.size(): {t5.size()}')
print(f't6 : {t6}, t6.size(): {t6.size()}')
```

Utskrift:

```
t5 : <1, 2>, t5.size(): 2
t6 : <1, 2>, t6.size(): 1
```

Det ena trädet innehåller två noder och det andra endast en men utskrifterna blir trots det identiska.

Skriv metoden `equal(self, other)` som fungerar även om nycklarna är strängar.

Exempel:

```
t5 = BST([1,2])
t6 = BST(['1, 2'])
print(f't5 : {t5}, t5.size(): {t5.size()}')
print(f't6 : {t6}, t6.size(): {t6.size()}')
print(f't5 == t6 : {t5 == t6}')
print(f't5.equal(t6): {t5.equal(t6)}')
```

Utskrift:

```
t5 : <1, 2>, t5.size(): 2
t6 : <1, 2>, t6.size(): 1
t5 == t6 : True
t5.equal(t6): False
```

Metoden ska ha komplexiteten $\mathcal{O}(n)$ och bara göra *ett* pass over noderna i respektive träd. Du får alltså, till exempel, *inte* göra listor av nycklarna och sedan jämföra dessa.

Anmärkning: Om din lösning för A8 redan uppfyller detta låter du `equal` använda `==`-operatorn.

Uppgifter i anslutning till modul 4

A9: Funktionen `dice(n,sides)` i `m4.py` simulerar `n` tärningskast med en tärning med `sides` sidor. Denna funktion ska inte ändras.

Modifiera funktionen `throw_dice(ns,n_sides)` i `m4.py` så att följande uppfylls:

- Första argumentet, `ns`, är en lista av antal kast som ska ske parallellt. T.ex. om argumentet är `[5, 8, 2]` så ska funktionen `throw_dice(ns,n_sides)` i tre samtidiga processer/trådar göra 5, 8 och 2 kast (d.v.s. köra funktionen `dice(n,sides)` tre gånger parallellt med respektive `n=5`, `n=8` och `n=2`).
- Andra argumentet, `n_sides`, ska vara ett heltal, d.v.s. alla parallella körningar sker med likadan tärning (t.ex. 6 eller 20 sidor).
- Funktionen ska returnera kvoten mellan antalet gånger man fick högsta möjliga värdet (t.ex. 6 på en 6-sidig tärning) och det totala antalet kast. Så om man totalt kastar 100 kast med en 6-sidig tärning, och får 14 6:or, så ska man returnera $14/100 = 0.14$.
- Ni kan använda vilken parallelliserings-modul ni vill. Ni ska inte importera några andra moduler utöver detta.

A10: Vid programmering kan man ofta få en lista av listor, som man vill kunna konvertera till en lista. Ett exempel nedan:

Konvertera `[[1,7,8],[9],[3,3],[1]]` till `[1, 7, 8, 9, 3, 3, 1]`.

Modifiera `make_list(lst)` i `m4.py` så att funktionen gör detta och

- Du kan anta att `lst` är en lista av listor (det är inte mer än två nivåer, d.v.s., `[[[1,2], 8], [9]]` är inte en giltig lista).
- Funktionen ska endast bestå av en en-rads list comprehension.
- Du ska inte importera några moduler.

B4: Bland filerna som du laddade ner för tentan finns filen `84-0.txt`, som är en rensad version av boken "Frankenstenin" (ursprungligen från https://www.gutenberg.org/files/84/84-0.txt).

Du ska modifiera `count_letters()` i `m4.py` så att du räknar hur många av varje bokstav (a-z) som finns i texten. Stora och små bokstäver ses som samma bokstav, och ska räknas. Man ska ignorera siffror och andra tecken.

Funktionen ska producera en bild, som sparas på hårddisken, med ett stapeldiagram (bar chart) med en stapel för varje bokstav, och höjden för varje stapel definieras av antalet som finns i filen `84-0.txt` för respektive bokstav. Funktionen ska även printa ut den bokstav som var vanligast, samt antalet.

Ni kan importera modulerna `string` och `matplotlib` för denna uppgift men inga andra moduler.

Ni behöver inte ladda upp bilden som svar för uppgiften (den ska genereras vid rättning).