

Tentamen Programmeringsteknik II 2022-05-30

Skrivtid: 08:00 – 13:00

Skrivningen sker i datorsalarna Å4102, Å4103, Å4104 samt Å10K1201. Var du ska skriva beror på din registreringskod, 10362-0001 till 10362-0054 i Å4102, Å4103, Å4104 och de med högre kod i Å10K1201.

Om det blir fullt kan en assistent hänvisa dig till de andra salarna.

Före tentamen:

- I god tid (**flera dagar!**) före skrivningen bör du gå till datorsalarna och se till att du kan logga in där och installera den Python-miljö du vill använda. För att installera program, använd ZENWorks (ikon på skrivbordet).

Du måste göra detta både i hus 4 och hus 10 eftersom det är olika system.

Du kan inte räkna med att få hjälp med inloggnings- eller installationsproblem under själva skrivningen.

- Om du har möjlighet är det bra att ha med en egen laptop som i nödfall kan användas vid ett totalt haveri på universitetets datorsystem (inte om det bara är du som har datorproblem)

Under tentamen:

- Ha Google-dokumentet
https://docs.google.com/spreadsheets/d/1tDcb768X0i81S1C4KX6sZ0Zzsv8L_JnQ8ph-NMh7GQU/edit
(välj fliken "Tentamen 30/5 8" i botten)
öppet under tentans gång. Där kommer vi skriva eventuella rättelser och förtydliganden. Större förändringar görs även med annonsering i STUDIUM.

Där finns även plats för frågor från studenter och svar på dessa från lärare.

Inlämning:

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

Observera att du måste ange din anmälningskod som kommentar först i alla filer!

- 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 vara körbar.

- 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 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 skicka zip-filen till sven-erik.ekstrom@it.uu.se. Det måste göras senast kl 13:00:00.

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

Regler

- Du får inte samarbeta med någon annan under tentan varken fysiskt eller elektroniskt. Att t.ex. en mail- eller en chat-klient öppen under skrivningen kommer att rapporteras som fuskförsök. Inga hörlurar får användas (och inga hemsidor med ljud får användas, typ Youtube).
- Du får inte kopiera kod eller text utan måste skriva dina svar själv.
- Du får använda internet. Om din lösning inspireras av någon resurs på nätet, ange då källan med fullständig länk (Bättre att ge för mycket länkar än för lite. Ni behöver inte ange källor om ni använder kursmaterial eller ren dokumentation för Python eller en modul).
- 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. Funktioner måste gå att anropa exakt på det sätt som står i uppgiften. I uppgifter för modul 1 och 3 är det tillåtet att lägga till parametrar med default-värden med de måste fortfarande fungera med de anrop som står 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 `is_sorted(lst)` som returnerar `True` om elementen i listan `lst` är sorterade i stigande ordning. Funktionen ska implementeras med *rekursion* och får alltså inte innehålla någon iteration. Du kan förutsätta att listans element är jämförbara (t.ex. alla tal eller alla strängar).

Exempel:

```
is_sorted([1, 3, 3, 8]) ska returnera True
is_sorted([138]) ska returnera True
is_sorted([]) ska returnera True
is_sorted([1, 5, 7, 8, 7]) ska returnera False
```

- A2** Skriv en funktion `depth(arg)` som returnerar djupet på `arg`. Djupet definieras som 0 om `arg` inte är en lista annars är djupet 1 plus djupet av det djupaste av elementen i listan. Sublistor ska hanteras med rekursion.

Exempel:

```
1 print(f'Argument                                Depth')
2 lists = (1, [], [[]], [1, 2, 3],
3          [[1], 2, [[3]]],
4          [[1, (1, [2])], [[[2]]], 3],
5          ['[', [']']])
6 for lst in lists:
7     print(f'{str(lst):30} {depth(lst)}')
```

Utskrift:

Argument	Depth
1	0
[]	1
[[]]	2
[1, 2, 3]	1
[[1], 2, [[3]]]	3
[[1, (1, [2])], [[[2]]], 3]	5
['[', [']']]	2

- B1** Nedanstående kod är ett vanligt sätt att rekursivt beräkna det största elementet i en lista.

```
1 def largest(lst):
2     if len(lst)==1:
3         return lst[0]
4     elif lst[0] > lst[-1]:
5         return largest(lst[:-1])
6     else:
7         return largest(lst[1:])
```

Dock är det så att list-skrivning skapar en kopia av listan så ett anrop med en lista som har n element kommer att skapa en lista med $n - 1$ element. Vad blir komplexiteten för denna funktion?

Skriv en rekursiv funktion `largest(lst)` som inte gör kopior av listor och därmed får en bättre komplexitet. Vad blir komplexiteten för denna funktion?

Exempel:

```
1 lists = ([1,2,3,4], [1,2,5,3], [1,6,2,3], [7,1,2,3])
2 for lst in lists:
3     print(f'In {lst} is {largest(lst)} largest')
```

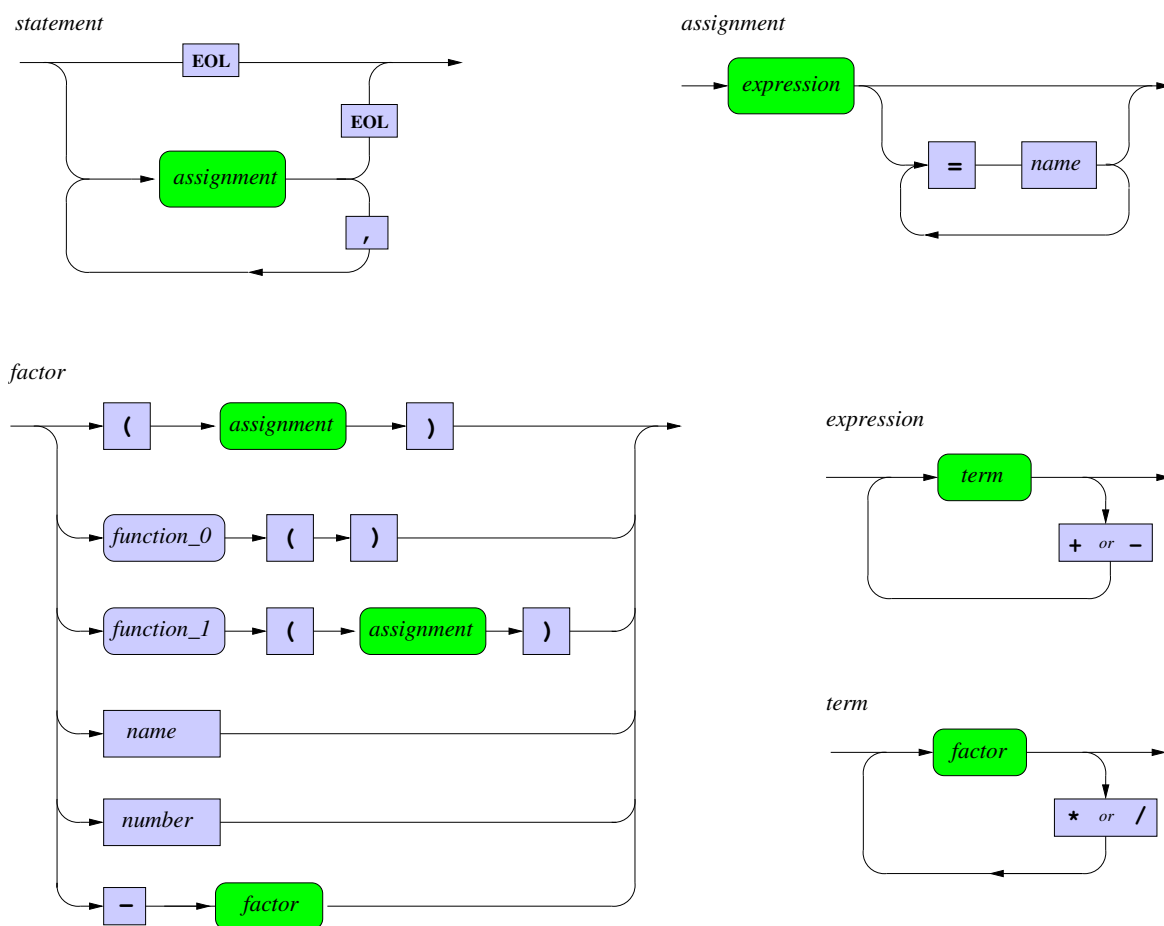
Utskrift:

```
1 In [1, 2, 3, 4] is 4 largest
2 In [1, 2, 5, 3] is 5 largest
3 In [1, 6, 2, 3] is 6 largest
4 In [7, 1, 2, 3] is 7 largest
```

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.

Programmet börjar med att läsa från filen `m2_init.txt` (som du laddade ner). Filen börjar så här:

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

Det förväntade resultatet står efter kommentartecknet `#`.

I resten av `m2_init.txt` ligger testfall för uppgifterna som alltså inte fungerar på rätt sätt innan du löst uppgifterna.

A3: Enligt syntaxdiagrammen ska det gå att skriva flera *assignments* separerade med kommatecken i ett *statement*. Exempel:

```
1 From init file: 1, 2, 3, 4, 5          # Last evaluated value is the value
2 5
3
4 From init file: ans
5 5
6
7 From init file: 1 = x, 2 = y, x + y    # Assignment performed successively
8 3
9
10 From init file: 10=x 11=y 3           # Syntax error - no comma
11 *** Syntax error: Expected end of line
12 *** Error occurred at '11' just after 'x'
13
14 From init file: x                      # x got a value
15 10
16
17 From init file: (1, 2, 3)              # Not allowed
18 *** Syntax error: Expected ')'
19 *** Error occurred at ',' just after '1'
```

Syntaxen framgår av syntaxdiagrammen.

Skriv om koden så att detta fungerar!

A4: I den givna koden accepteras bara funktioner med *ett* argument. Enligt syntaxdiagrammen ska man kunna ha funktioner utan argument. Se till att den möjligheten finns med i koden och lägg in funktionerna `random` som ska ge ett slumptal (float) mellan 0 och 1 samt `time` som ska returnera en streäng med aktuellt datum och tid.

Tips: `random.random()` och `time.ctime()` ger de svar som behövs. Både `random` och `time` är importerade i den nedladdade koden. Exempel:

```
1 From init file: random()
2 0.7400992923211458
3
4 From init file: random(1)
5 *** Syntax error: Expected ')'. This function has no arguments.
6 *** Error occurred at '1' just after '('
7
8 From init file: time()
9 Thu May 5 18:34:30 2022
10
11 From init file: time(1)
12 *** Syntax error: Expected ')'. This function has no arguments.
13 *** Error occurred at '1' just after '('
14
15 From init file: time
16 *** Syntax error: Expected '(' after function name.
17 *** Error occurred at '' just after 'time'
```

B2: Den givna implementationen av kalkylatorn hanterar bara flyttal. Utvidga kalkylatorn så att den även kan hantera heltal och strängar. Exempel:

```

1
2 From init file: 1          # Without decimal point
3 1
4
5 From init file: 1.         # With decimal point
6 1.0
7
8 From init file: 2 + 1      # Should give an integer result
9 3
10
11 From init file: 2. + 1    # Should give a float result
12 3.0
13
14 From init file: 1/2       # Should, as in Python, give a float result
15 0.5
16
17 From init file: 'Hello!' # No apostrophes in the result
18 Hello!
19
20 From init file: 'Kalle' = name1, 'Lisa' = name2
21 Lisa
22
23 From init file: 'Hello ' + name1 + ' and ' + name2 + '!'
24 Hello Kalle and Lisa!
25
26 From init file: 'Grattis' + '!'*5
27 Grattis!!!!
28
29 From init file: 'Todays date and time: ' + time()
30 Todays date and time: Sat May 7 17:06:05 2022
31
32 From init file: 'sin(1) = ' + sin(1)
33 *** Invalid type or operation:
34 *** can only concatenate str (not "float") to str
35
36 From init file: 'sin(1) = ' + str(sin(1))
37 sin(1) = 0.8414709848078965
38
39 From init file: 'abc' - 'c'
40 *** Invalid type or operation:
41 *** unsupported operand type(s) for -: 'str' and 'str'
42
43 From init file: 4*'a'*4
44 aaaaaaaaaaaaaaaaaa
45
46 From init file: 'a'* exp(3)*2
47 *** Invalid type or operation:
48 *** can't multiply sequence by non-int of type 'float'
49
50 From init file: 'a'* int(exp(3))*2
51 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

```


Det ska alltså gå att

- Mata in strängar omgivna av apostrofer (`'`).
- Konkatenera strängar med `+`.
- Multiplicera med strängar med `*` (samma funktion som i Python).
- Konvertera tal till strängar med funktionen `str`.
- Konvertera strängar till tal med funktionerna `int` och `float`.

Strängoperationerna `+` och `*` ska fungera som i Python. Programmet ska också diagnosticera felaktiga operationer.

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!

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

Klassen `BST` innehåller kod för vanliga binära sökträd.

A5: Vad har nedanstående funktion för tidskomplexitet?

```
1 def build_list(n):
2     llist = LinkedList()
3     for x in range(n):
4         llist.insert(x)
5     return llist
```

Svara med av typen $\Theta(f(n))$! Motivera! Styrk ditt svar med att göra en testkörning med tidmätning. Skriv koden för detta på anvisad plats och ange vilka tider du fått.

Uppskatta hur lång tid anropet `build_list(1000000)` skulle ta.

Plats för att svara på detta och skriva koden för verifiering finns i `main`-funktionen.

A6: Den givna koden för listor lagrar, som sagt, samma nyckel flera gånger. Modifiera koden så varje nyckel lagras en gång men att noderna räknar hur många gången varje nyckel lagrats.

Ledning: Lägg till en instansvariabel i noderna som `insert`-metoden räknar upp och låt iteratorn returnera en tuppel med nyckel och antal gånger nyckeln har lagts in.

Exempel:

```
1 lst = LinkedList()
2 for x in (3,1,2,5,4,3,3,4):
3     lst.insert(x)
4 print(lst)
```

Utskrift:

```
1 <(1, 1), (2, 3), (3, 2), (4, 1), (5, 1)>
```

A7: Skriv metoden `count_nodes_on_level(level)` som räknar och returnerar antalet noder på nivå `level`.

Exempel:

```
1 bst = BST()
2 for x in (5, 3, 2, 4, 8, 10, 6, 1, 7, 9):
3     bst.insert(x)
4 for i in range(5):
5     print(f'level {i}: {bst.count_nodes_on_level(i)}')
6 print()
```

Utskrift:

```
1 level 0: 1
2 level 1: 2
3 level 2: 4
4 level 3: 3
5 level 4: 0
```

A8: Implementera indexoperatoren `[]` för klassen `BST`. Uttrycket `bst[0]` ska returnera det minsta värdet i trädet, `bst[1]` det näst minsta etc. Ett felaktigt index ska skapa ett `IndexError`. Operatoren ska bara kunna användas för att hämta värden, inte ändra på värden.

Exempel:

```
1 bst = BST()
2 for x in (5, 3, 2, 4, 8, 10, 6, 1, 7, 9):
3     bst.insert(x)
4 for i in range(0, 6, 2):
5     print(f'bst[{i}] = {bst[i]}')
6 try:
7     bst[10]
8 except IndexError as ie:
9     print(ie)
```

Utskrift:

```
1 bst[0] = 1
2 bst[2] = 3
3 bst[4] = 5
4 Tree index out of range: 10
```

B3: Skriv klassen `LevelOrderIterator` som returnerar nycklarna i nivåordning det vill säga först värdet i roten, sedan värdena i rotens barn och så vidare.

Exempel:






```
1 bst = BST()
2 print('Insertion order: ', end=' ')
3 for x in [5, 3, 2, 4, 8, 10, 6, 1, 7, 9 ]:
4     print(x, end = ' ')
5     bst.insert(x)
6 print('\nSymmetric order: ', end=' ')
7 for x in bst:
8     print(x, end=' ')
9 print('\nLevel order      : ', end = ' ')
10 loi = LevelOrderIterator(bst)
11 for x in loi:
12     print(x, end= ' ')
13 print()
```

Utskrift:

```
1 Insertion order:  5 3 2 4 8 10 6 1 7 9
2 Symmetric order:  1 2 3 4 5 6 7 8 9 10
3 Level order      :  5 3 8 2 4 6 10 1 7 9
```

Uppgifter i anslutning till modul 4

Alla tre uppgifter i denna modul (A9, A10 och B4) bygger på följande premisser.

En vanlig kortlek består av 52 kort (inga jokrar), uppdelat i 4 färger/sviter (hjärter ♥, ruter ♦, klöver ♣ och spader ♠). Det finns alltså 13 kort av olika valör för varje färg (1-10 , , knekt , dam , kung ). I uppgifterna är det enklast att ni identifierar varje unikt kort med värdena 1-52 (t.ex., 1-13 hjärter, 14-26 ruter, 27-39 klöver och 40-52 spader). Ess identifieras då som {1, 14, 27, 40}. Ni får använda vilket sätt ni vill för att identifiera unika kort.

Inga andra Python-moduler/paket än de som explicit tillåts får importeras.

A9: I poker får man en “hand”, d.v.s. 5 slumpvisa kort ur kortleken. Modifiera funktionen `hands_of_cards(n)` i `m4.py` så att den returnerar en lista med `n` “händer” (lista med 5 kort). Varje hand ska dras ur en ny kortlek.

Om vi representerar alla kort med talen 1-52 enligt introduktionstexten, och väljer `n=2` kanske vi får:

```
[[1, 13, 4, 26, 51], [7, 22, 1, 44, 45]]
```






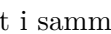

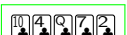



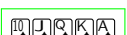
vilket då motsvarar     ,     . Som synes, så råkade man i exemplet ovan få samma kort, 1 , i båda händerna.

- Skriv gärna hjälpfunktioner.
- Ni får importera vilka funktioner ni vill från modulen `random`

Tips: `random.shuffle()` kan vara användbar.

- Någon högre ordningens funktion, som nämnades under kursen, ska användas någonstans i koden. Dock, om ni gör detta i A10 istället så behöver ni inte göra det i denna uppgift.

A10: I poker så kan man få så kallade “pokerhänder” som är speciella kombinationer av kort, och är värda olika mycket i spelet. Nedan visas de olika varianterna, där exemplen är tagna från https://en.wikipedia.org/wiki/Poker_probability#Frequency_of_5-card_poker_hands



Namn	Beskrivning	Exempel	Teoretisk frekvens
Ingenting	Inget av nedanstående		50.1177%
Ett par	Två kort har samma valör		42.2569%
Två par	Två par i samma hand		4.7539%
Triss	Tre kort har samma valör		2.1128%
Straight/Stege	Fem kort i följd (ej samma färg) Ess kan räknas både som kortet före Tvåa eller efter Kung Giltigt:  och 		0.3925%
Flush/Färg	Fem kort i samma färg/svit (ej i följd)		0.1965%
Kåk	Triss + par		0.1441%
Fyrtal	Fyra kort har samma valör		0.02401%
Straight flush	Straight/Stege samtidigt som Flush/Färg		0.00139%
Royal straight flush	Som Straight flush och minsta valören är 10 (Ess kommer då efter Kung)		0.000154%

En av dessa händer, är att få **Ett par**, d.v.s. 2 av korten i en hand (5 kort) har samma valör (men man har ej två par, triss, kåk eller fyrtal). Om vi representerar alla kort med talen 1-52, så har man t.ex. två femmor om två av talen {5, 18, 31, 44} finns med i en hand. Ni ska nu skriva en parallell kod som skapar `n` slumpvisa händer, hittar antalet händer, `n_pairs`, som innehåller ett par (om en hand har två par, triss, kåk eller fyrtal, så ska den ej räknas), och returnerar kvoten `n_pairs/n`.

Ni ska modifiera funktionen `compute_one_pair_ratio(n, n_processes)` i `m4.py`:

- Inparametrar är
 - `n`: Totala antalet händer (5 kort) som slumpas
 - `n_processes`: Antal processorer/trådar som används. Koden ska vara parallell, ni får använda vilken parallell modul ni vill. Ni kan anta att `n` är jämnt delbart med `n_processes`.
- I koden ska ni räkna ut antalet händer `n_pairs` med ett par (två kort med samma valör i handen, dock inga mer värdefulla händer som, två par, triss, kåk, ellerfyrtal), och sedan returnera kvoten `n_pairs/n`.
- Om ni inte gjort A9, så accepteras en kod i denna uppgift om den fungerar ifall A9 hade varit löst korrekt.

Ifall n är tillräckligt stort (t.ex., 10000) så bör kvoten n_pairs/n vara ungefär 0.42, se tabellen ovan under “Teoretisk frekvens”.

- B4:** Modifiera `compute_all_ratios(n,n_processes)` i `m4.py`, som funkar som uppgiften A10, men att du räknar ut kvoterna för alla typer av “pokerhänder”. Alla olika “pokerhänder” listas i tabellen i uppgiften A10. Glöm inte kvoten för “Ingenting”, och att Ess kan komma både före Tvåa eller efter Kung i en Straight/stege (Giltigt: “pokerhänder”  och .

Presentera resultaten på ett bra sätt, t.ex. i en figur. För själva presentationen av resultaten får ni använda vilka moduler ni vill, t.ex. `matplotlib` eller gör en bra formaterad utprinting i terminalen.