DA2005: Laborationer

Lars Arvestad, Evan Cavallo, Christian Helanow, Anders Mörtberg, Kristoffer Sahlin

16 juni 2025

Innehåll

Labbregler

- Alla deadlines är **strikta**. Vid missad deadline måste du göra om labben eller projektet vid nästa kurstillfälle. Kontakta huvudläraren om detta händer.
- Om du vet att du inte kommer hinna klart med en labb eller projektet (p.g.a. giltigt skäl, t.ex. sjukdom) informera kursledare **innan** deadline. Om du kontaktar kursledaren först efter deadline så räknas den som missad och du måste göra om vid nästa kurstillfälle.
- Du måste jobba individuellt med labbarna och projektet. Det betyder att man ska skriva sin egen kod och hitta sina egna lösningar. Du får inte ge lösningar till varandra eller kopiera kod från Internet. Alla inlämningar jämförs automatiskt och misstänkt fusk rapporteras till universitetets disciplinnämnd vilket kan leda till avstängning.
- Du får Googla efter Python-kommandon, syntax, felmeddelanden, etc, men inte efter färdiga lösningar. Googla på engelska för att få flest svar. På sajten https://stackoverflow.com/ finns förmodligen svaret på många av dina frågor.
- Du får använda AI-tjänster som ChatGPT, men du måste ange hur du har använt tjänsten genom att skriva kommentarer i koden. Du måste även kunna förklara hur koden du lämnat in fungerar under redovisning (se nedan). Observera att vi har restriktioner i uppgifterna som AI-tjänsterna gärna ignorerar.
- Använd ej funktioner från något bibliotek i labbarna om det inte står explicit att du får det (d.v.s., använd inte "import" någonstans om det inte står att du ska det).
- Har du problem med saker som inte har med själva programmeringen att göra, t. ex. problem med att använda terminalen, bibliotek som är i konflikt, etc, kontakta någon av lärarna för hjälp.
- Lösningen ska ha en rimlig struktur, d.v.s. det är inte ok med extremt långa program med samma kopierade kod istället för t. ex. en loop.

Inlämning

- Inlämningarna får inte göras i något annat språk än Python 3. Det är alltså ej tillåtet att göra inlämningarna i Python 2.
- Inlämning görs via inlämningsmoduler på kurshemsidan.
- Lösningar ska lämnas in i form av .py-filer. Inget annat filformat än .py är godkänt om det inte specificeras explicit i instruktionerna.
- Din lösning ska ge rätt utdata för en given indata för alla tester i uppgiften.

Redovisning

- För att bli godkänd på labb 3 till 5 ska *kanske* inlämnad kod redovisas för en handledare på ett labbpass, beroende på behov och slump.
- För att få redovisa måste koden vara inlämnad i tid och det är den inlämnade koden som ska redovisas.
- Under redovisning måste du visa att du förstår hur koden fungerar och förklara hur du tänkt när du skrivit den. Du kan bli ombedd att modifera din kod för små ändringar i uppgiften.

- Handledaren kan be er att köra koden på ny indata utöver testerna som finns i uppgiften. Koden bör då fungera som den ska och gör den inte det ska du kunna förklara varför den inte gör det och ge förslag på hur det kan lösas.
- Vid brister i lösningen kan du bli ombedd att komplettera genom att rätta till det som saknas eller är fel. Eventuella kompletteringar görs muntligen för handledare.
- Vid osäkerhet i bedömningen kan du behöva redovisa din kod för huvudlärare för att bli godkänd.
- Då redovisning är ett examinerande moment i kursen måste man kunna legitimera sig, så ta med giltig legitimation till redovisningen.

Laboration 1

Temperaturkonvertering

Huvudsyftet med denna laboration är att ni ska bekanta er med utvecklingsmiljön antingen i salarna eller på er egen dator.

1.1 Lärandemål

Du ska kunna skriva, köra, och ändra i ett mindre Python-program.

1.2 Förkunskaper

För att lösa uppgiften behöver du förbereda dig så pass att du åtminstone vet hur följande Python-instruktioner fungerar:

- `def`
- `if`, `else`, och `elif`
- `print`
- `input`
- `float`

1.3 Ej accepterade tekniker

En lösning som använder tekniker listade nedan kommer inte att accepteras.

- Andra moduler än `sys` och `math`.
- `match` (använd `if` istället).
- Listomfattning (*list comprehension*)
- Särfallshantering (*exception handling*)
- Objektorientering

AI-tjänster använder rutinmässigt mer avancerade tekniker än vad som behövs, så undvik dessa.

1.4 Uppgift

Programmet `konvertera.py` i mappen "Kod för laborationer" på kurshemsidan är tänkt att fråga efter en temperatur i Fahrenheit, läsa in från användaren, konvertera till Celsius-skalan och sedan skriva ut resultatet.

Här är dina uppgifter:

1. Ladda ned, provkör och studera `konvertera.py` programmet från kurshemsidan (se mappen "Kod för laborationer"). Fungerar det som det ska?
2. Skriv om funktionen `fahrenheit_to_celsius` så att den räknar rätt.
3. Utöka programmet med en funktion som konverterar från Celsius till Fahrenheit.
4. Utöka programmet med funktioner så att det också kan konvertera till och från Kelvin från både Celsius och Fahrenheit.
5. Utöka programmet så att det frågar efter vilken konvertering du vill göra. Exemplet här nedanför visar hur det bör se ut (ungefär) när man kör sitt program.
6. Lämna in din lösning i inlämningsmodulen på kurshemsidan.

1.4.1 Tips

- Det är alltid bra att kommentera koden där det behövs (för att förtydliga syfte med t.ex. en rad kod eller ett kodblock) och att dokumentera funktioner. Alla funktioner man skriver bör även testköras grundligt så att man vet att de fungerar som man tänkte sig!
- Använd gärna `elif` när du har flera `if`-satser på rad.

1.4.2 Exempelkörning

När man kör ditt program bör det se ut ungefär så här:

```
Temperaturkonverteraren
1. Celsius till Fahrenheit
2. Fahrenheit till Celsius
3. Celsius till Kelvin
4. Kelvin till Celsius
5. Fahrenheit till Kelvin
6. Kelvin till Fahrenheit

Vilken konvertering vill du göra? 3

Ange temperatur i Celsius: 30
Det motsvarar 303.15 Kelvin
```

1.5 Inlämning

Innan du lämnar in koden bör du läsa igenom, renskriva och dokumentera koden. Lämna sedan in din `konvertera.py` i inlämningsmodulen på kurshemsidan.

Laboration 2

Polynom

I den här uppgiften ska vi representera polynom med hjälp av listor av koefficienter. Ordet "representation" betyder här ungefär "lagringssätt i dator", och det betyder rent konkret att ett polynom som `1+3x+7x^2` lagras i Python som listan `[1,3,7]`. Generellt lagras alltså koefficienten för termen av grad `n` på listans plats `n`.

Du bör ha läst till och med kapitel 4 i kompendiet för uppgifterna i den här laborationen.

2.1 Lärandemål

- Du ska se hur man kan ge ett abstrakt begrepp (polynom) en konkret representation i datorn och hur man kan göra beräkningar på det.
- Du ska kunna iterera med `for`.
- Du ska kunna skriva små enkla funktioner
- Du ska kunna arbeta med datastrukturen `list`.

2.2 Förkunskaper

För att lösa uppgiften bör du kunna använda listor, iteration och funktioner.

2.3 Ej accepterade tekniker

En lösning som använder tekniker listade nedan kommer inte att accepteras.

- Andra moduler än `sys` och `math`.
- `match` (använd `if` istället).
- Listomfattning (*list comprehension*)
- Särfallshantering (*exception handling*)
- Objektorientering

2.4 Uppgifter

I labben ska polynom representeras som listor. Nedan följer några fler exempel på hur polynom kan implementeras som listor:

Polynom	Python-representation
x^4	`[0,0,0,0,1]`
$4x^2 + 5x^3$	`[0,0,4,5]`
$5 + 4x + 3x^2 + 2x^3 + x^4$	`[5,4,3,2,1]`

Obs: du kan utgå ifrån att listorna du jobbar med bara innehåller tal.

Funktionen i Python-filen `polynom.py` (finns på kurshemsidan i "Kod för laborationer" mappen) innehåller en funktion, `poly_to_string`, som konverterar polynom representerade som listor till strängar.

Börja med att skapa en fil `labb2.py`, för att komma igång med labben, och kopiera över funktionen `polynomial_to_string` dit. Ni ska nu lösa alla uppgifter nedan genom att lägga till nödvändig kod för att lösa problem som specificeras i uppgifterna. Som du kommer att se måste du också i avsnitt 2.4.5 modifiera funktionen `polynomial_to_string`.

Obs: man får inte använda funktioner från något bibliotek i labben, d.v.s. man får inte använda `import` någonstans i lösningen.

2.4.1 Uppgift 1

Antag att polynomen `p` och `q` är definierade matematisk som nedan.

$$\begin{aligned} p &:= 2 + x^2 \\ q &:= -2 + x + x^4 \end{aligned}$$

Skriv kod för att lagra listrepresentationen för dessa två polynom i variablerna `p` och `q` i Python. Det vill säga,

```
p = [...]
q = [...]
```

där innehållet i listorna ska fyllas i. Testa att du skrivit rätt på följande sätt:

```
>>> polynomial_to_string(p)
'2 + 0x + 1x^2'
>>> polynomial_to_string(q)
'-2 + 1x + 0x^2 + 0x^3 + 1x^4'
```

Här är `>>>` "prompten" i Python-tolken, d.v.s. `polynomial_to_string(p)` är ett kommando som ska köras av Python och på raden efter kommer resultatet. Detta kan se annorlunda ut på er dator och uppnås på olika sätt, t.ex. i Spyder kan man istället skriva:

```
print(polynomial_to_string(p))
```

och sen observera resultatet i konsolen till höger när man kört programmet. I så fall kommer du inte att se `'2 + 0x + 1x^2'` utan bara `2 + 0x + 1x^2`, d.v.s. `'` kommer inte att skrivas ut. Denna typ av tester är väldigt hjälpsamma när man utvecklar koden, men bör inte vara med i slutversionen som ni lämnar in. Det kan dock vara bra att ha med testerna i form av kommentarer för att underlätta för den som ska läsa igenom och testa koden.

2.4.2 Uppgift 2

a) Skriv en funktion `drop_zeroes` som tar bort alla nollor i slutet av ett polynom och **returnerar** resultatet. Funktionen får inte modifiera sitt argument, d.v.s., en ny lista ska skapas.

Exempel:

```
>>> ex_poly = [1, 2, 0, 0]
>>> drop_zeroes(ex_poly)
[1, 2]
>>> ex_poly    # Verifiera att ex_poly inte har modifierats
[1, 2, 0, 0]
```

Börja så här:

```
def drop_zeroes(p_list):
    # here be code
```

Tips: använd en `while`-loop och list-operationen `pop()`.

Definiera sen några polynom med nollor i slutet,

```
p0 = [2,0,1,0]      # 2 + x^2 + 0x^3
q0 = [0,0,0]        # 0 + 0x + 0x2
```

och testa funktionen:

```
>>> drop_zeroes(p0)
[2, 0, 1]
>>> drop_zeroes(q0)
[]
>>> drop_zeroes([])
[]
```

b) Skriv en funktion som testar när två polynom är lika genom att ignorera alla nollor i slutet och sedan testa likhet:

```
def eq_poly(p_list,q_list):
   # here be code
```

Testa så att koden fungerar med `p` och `q` från Uppgift 1 samt `p0` och `q0` från deluppgift **a** ovan:

```
>>> eq_poly(p,p0)
True
>>> eq_poly(q,p0)
False
>>> eq_poly(q0,[])
True
```

Obs: funktionerna `drop_zeroes` och `eq_poly` ska **returnera** sina resultat med hjälp av `return` och inte bara skriva ut dem med `print`. Skillnaden kan vara svår att greppa till en början då resultatet ser liknande ut när man kör koden, men det är väldigt stor skillnad på en funktion som returnerar något och en som bara skriver något. Se slutet av 2.5.1 i kompendiet för mer information om detta.

2.4.3 Uppgift 3

Skriv en funktion med namnet `eval_poly` som tar ett polynom och ett värde på variabeln `x` och **returnerar** polynomets värde i punkten `x`.

Förslag på algoritm:[1]

- Iterera över polynomets termer genom att iterera över koefficienterna.
- Håll reda på graden på den aktuella termen och summan av de termer du adderat hittills. I varje iteration beräknar du värdet av termen som `coeff * x ** grad` (kom ihåg att upphöjt är `**` i Python). Addera sedan termens värde till summan.
- När du itererat klart kan summan returneras.

Tester:

```
>>> eval_poly(p,0)
2
>>> eval_poly(p,1)
3
>>> eval_poly(p,2)
6
>>> eval_poly(q,2)
16
>>> eval_poly(q,-2)
12
```

2.4.4 Uppgift 4

Nu ska vi definiera några funktioner för att skapa nya polynom från gamla – med negation, addition och subtraktion. Var och en av dessa funktioner ska ta en eller flera listor som argument och returnera en ny lista. Dina funktioner ska inte innehålla några strängar.

Obs: Funktionerna `neg_poly`, `add_poly` och `sub_poly` ska returnera nya listor och inte ändra de listor som ges som indata. Du kan använda funktionen `copy()` för att vid behov skapa en kopia av en lista.

a) Definiera funktionen `neg_poly(p)` som för ett givet polynom $p(x)$ returnerar negationen $-p(x)$. Det innebär alltså att alla koefficienterna i $p(x)$ ska negeras. Funktionen ska returnera ett nytt polynom, så du får inte modiera den lista som ges som indata till funktionen.

Exempel:

```
>>> p1 = [1, 2, 3]
>>> neg_poly(p1)
[-1, -2, -3]
>>> p1 # Verifiera att polynomet inte har ändrats
[1, 2, 3]
```

Du kan börja så här:

```
def neg_poly(p_list):
    # here be code
```

[1] Den föreslagna algoritmen är inte den mest effektiva. Vill man optimera så kan man om man vill istället implementera Horners algoritm: https://sv.wikipedia.org/wiki/Horners_algoritm

b) Definiera funktionen `add_poly(p, q)` som adderar två polynom. Det innebär att alla koefficienter adderas, läggs i en ny lista, och returneras. Din funktion får inte modifera de polynom som ges som indata.

Exempel:

```
>>> p1 = [1, 2, 3]
>>> p2 = [4, 3, 2]
>>> add_poly(p1, p2)
[5, 5, 5]
>>> p1  # Verifiera att polynomet inte har ändrats
[1, 2, 3]
>>> p2  # Verifiera att polynomet inte har ändrats
[4, 3, 2]
```

Börja så här:

```
def add_poly(p_list,q_list):
    # here be code
```

c) Definiera funktionen `sub_poly(p, q)` som beräknar subtraktion av polynom. Funktionen får inte modifiera de givna polynomen.

Exempel:

```
>>> p1 = [1, 2, 3]
>>> p2 = [4, 3, 2]
>>> sub_poly(p1, p2)
[-3, -1, 1]
>>> p1  # Verifiera att polynomet inte har ändrats
[1, 2, 3]
>>> p2  # Verifiera att polynomet inte har ändrats
[4, 3, 2]
```

Börja så här:

```
def sub_poly(p_list,q_list):
    # here be code
```

Tips: tänk på att `p - q` kan definieras som `p + (- q)`, d.v.s. för att subtrahera polynomet `q` från `p` kan man först ta negationen av `q` och sen addera med `p`.

Testa så att funktionerna fungerar:

```
# p + q = q + p
>>> eq_poly(add_poly(p,q),add_poly(q,p))
True
# p - p = 0
>>> eq_poly(sub_poly(p,p),[])
True
# p - (- q) = p + q
>>> eq_poly(sub_poly(p,neg_poly(q)),add_poly(p,q))
True
# p + p != 0
```

```
>>> eq_poly(add_poly(p,p),[])
False
# p - q = 4-x+x^2-x^4
>>> eq_poly(sub_poly(p,q),[4, -1, 1, 0, -1])
True
# (p + q)(12) = p(12) + q(12)
>>> eval_poly(add_poly(p,q),12) == eval_poly(p,12) + eval_poly(q,12)
True
```

Obs: kommentarerna är bara där för att förklara vad testerna testar. Kan du tänka ut fler bra tester för att hitta eventuella buggar i koden?

2.4.5 Uppgift 5

Ändra i `polynomial_to_string` så att:

- Tomma listan konverteras till `'0'`.
- Termer med koefficient `1` skrivs utan koefficient. D.v.s., `'1x^2'` ska istället skrivas som `'x^2'`.
- Termer med koefficient `-1` lägger ett minus framför termen, men ettan skrivs inte ut. T.ex. `'2 + -1x^2'` skrivs istället som `'2 + -x^2'`.
- Termer med koefficient `0` skrivs inte ut. D.v.s., `'0 + 0x + 2x^2'` ska förenklas till `'2x^2'`.
- En lista innehållande endast `0` som element, t.ex. `[0, 0, 0]` skrivs som `'0'`.

Testa funktionen! Så här ska utdata se ut:

```
>>> polynomial_to_string(p)
'2 + x^2'
>>> polynomial_to_string(q)
'-2 + x + x^4'
>>> polynomial_to_string([])
'0'
>>> polynomial_to_string([0,0,0])
'0'
>>> polynomial_to_string([1,2,3])
'1 + 2x + 3x^2'
>>> polynomial_to_string([-1, 2, -3])
'-1 + 2x + -3x^2'
>>> polynomial_to_string([1,1,-1])
'1 + x + -x^2'
```

2.5 Inlämning

Innan du lämnar in koden bör du läsa igenom, renskriva och dokumentera koden. Lämna sedan in din `labb2.py` i inlämningsmodulen på kurshemsidan.

Tips: läs redan nu igenom kapitlet “Tumregler för programstruktur och bra kod” i kompendiet för tips på hur man skriver bättre kod.

Laboration 3

Iteration, filhantering, felhantering och uppslagstabeller

Den här labben innehåller ett antal oberoende uppgifter som involverar grundläggande algoritmer, uppslagstabeller, filhantering och felhantering.

3.1 Lärandemål

- Du ska kunna kunna översätta en algoritm till kod.
- Du ska kunna arbeta med uppslagstabeller.
- Du ska kunna läsa och skriva data från fil.
- Du ska kunna använda felhantering.

3.2 Förkunskaper

För att lösa uppgiften bör du ha bekantat dig med uppslagstabeller, satt dig in i filhantering (öppna, läsa, skriva), och läst om `try-except`.

3.3 Ej accepterade tekniker

En lösning som använder tekniker listade nedan kommer inte att accepteras.

- Datastrukturen `set`.
- De inbyggda funktionerna `sort` och `sorted`.
- `match` (använd `if` istället).
- Andra moduler än `sys` och `math`.
- Objektorientering

3.4 Uppgifter

3.4.1 Uppgift 1: insättningssortering

Insättningssortering (eng.: *insertion sort*) är en vanlig sorteringsalgoritm, d.v.s. en metod för att sortera en lista med element. Idén bakom den här algoritmen liknar ett sätt som man kan sortera en kortlek på: för varje kort i kortleken, sätt in det på rätt plats i en hög med sorterade kort.

Vi kan dela upp detta i två delproblem:

a) Skriv en funktion som sätter in ett element `x` i en redan *sorterad* lista `sorted_list`:

```
def insert_in_sorted(x,sorted_list):
    # here be code
```

Algoritmförslag:

1. Antag att `sorted_list` är sorterad.
2. Iterera över alla index `i < len(sorted_list)` tills ni hittar något element `sorted_list[i]` som uppfyller `x < sorted_list[i]` och sätt då in `x`.
3. Om det inte finns något `sorted_list[i]` som är större än `x`, sätt då in `x` i slutet.

Tester:

```
>>> insert_in_sorted(2,[])
[2]
>>> insert_in_sorted(5,[0,1,3,4])
[0, 1, 3, 4, 5]
>>> insert_in_sorted(2,[0,1,2,3,4])
[0, 1, 2, 2, 3, 4]
>>> insert_in_sorted(2,[2,2])
[2, 2, 2]
```

b) Skriv insättningssortering med hjälp av `insert_in_sorted`:

```
def insertion_sort(my_list):
    # here be code
```

Algoritmförslag:

1. Initiera en variabel `out` med tomma listan.
2. För varje element `x` i `my_list` sätt in det i `out` med hjälp av er funktion `insert_in_sorted`.
3. Returnera `out`.

Tester:

```
>>> insertion_sort([12,4,3,-1])
[-1, 3, 4, 12]
>>> insertion_sort([])
[]
```

Obs: för godkänt måste `insertion_sort` använda sig av `insert_in_sorted`. Lösningen får inte heller använda någon inbyggd Python-funktion för sortering, som `sort` eller `sorted`.

3.4.2 Uppgift 2: filhantering

Skriv en funktion `number_lines(f)` som tar ett filnamn `f` som parameter och skriver ut till en ny fil med samma innehåll, men med ett radnummer först på varje rad. Den nya filen ska få namn efter `f`, men med prefixet `numbered_` tillagt.

Exempel: Antag att filen `poem.txt` innehåller denna text av Kipling:

```
A Dead Statesman

I could not dig; I dared not rob:
Therefore I lied to please the mob.
Now all my lies are proved untrue
And I must face the men I slew.
What tale shall serve me here among
Mine angry and defrauded young?
```

En körning av `number_lines('poem.txt')` ska då producera en fil `numbered_poem.txt` som innehåller:

```
0 A Dead Statesman
1
2 I could not dig; I dared not rob:
3 Therefore I lied to please the mob.
4 Now all my lies are proved untrue
5 And I must face the men I slew.
6 What tale shall serve me here among
7 Mine angry and defrauded young?
```

3.4.3 Uppgift 3: enkel textindexering

a) Skriv en funktion `index_text(filename)` som tar ett filnamn och **returnerar** en uppslagstabell som parar ihop orden med de radnummer där ordet finns, för den text som förväntas finnas i filen.

Exempel: Med en enkel fil `Sommar.txt` med innehållet

```
Sommar och sol
sol och vatten
```

så ska `index_text('Sommar.txt')` returnera en uppslagstabell som den här:

```
{
  'sommar': [0],
  'och': [0, 1],
  'sol': [0, 1],
  'vatten': [1]
}
```

Du kan utgå ifrån att det bara finns bokstäver, mellanslag och nyradstecken (`\n`) i indata.

Om ett ord upprepas på samma rad så ska raden inte upprepas i utdata. Det innebär att om indata är

```
sommar sommar sommar
```

så ska den returnerade uppslagstabellen vara

```
{
  'sommar': [0]
}
```

Krav på din lösning

- Funktionen ska inte göra skillnad på versaler och gemener (d.v.s. stora och små bokstäver) för indexerade ord.
- Din lösning ska klara indata av godtycklig storlek.

b) Skriv en funktion `important_words(an_index, stop_words)` som tar resultatet från `index_text`-funktionen och en lista med "stoppord", d.v.s. ord som ska ignoreras, för att **returnera** de fem mest frekventa orden i indexet (d.v.s. de fem ord som förekommer på flest rader). Om det finns fler än fem ord med samma "toppfrekvens", till exempel om det finns sex ord som återfinns 10 gånger, så ska du välja fem godtyckliga ord av dem. Om det finns färre än fem ord i indata så returnerar du de som finns.

Exempel: Om vi kör programmet

```
sommar_index = index_text('sommar.txt')

for w in important_words(sommar_index, ['och']):
   print(w)
```

så ska orden `sommar`, `vatten`, `sol` skrivas ut.

c) Skriv ett program som frågar användaren efter en textfil och skriver ut de viktigaste orden (i den tolkning vi lagt i förra funktionen). Programmet ska ha stopporden `'och'`, `'jag'`, `'som'`, `'det'`, och `'för'` definierade.

Testa programmet på Idas sommarvisa, given i textfilen `idas.txt` (hämtbar på kurswebben). Resultatet kan se ut så här, berorende på hur man råkar välja ut likvärdiga ord, men ordet "barna" ska vara ett av de mest viktiga orden.

```
En textfil: idas.txt
De viktigaste orden är:
så
där
små
barna
gör
```

Obs: om man har problem med åäö så finns det på kurswebben även en version `idas_aaaeoe.txt` där "å" är utbytt mot "aa", "ä" är utbytt mot "ae", och "ö" är utbytt mot "oe". Det räcker att lösningen fungerar på den filen för att man ska bli godkänd.

Krav på din lösning:

- Din lösning *ska* använda sig av de funktionerna `index_text` och `important_words` som du skrivit ovan.
- Det måste finnas med lämplig felhantering med hjälp av `try-except` så att om man försöker öppna en fil som inte finns så ska inte programmet bara krascha utan ett lämpligt meddelande ska skrivas ut och användaren ska få försöka öppna en fil igen.

3.5 Inlämning

Lämna som vanligt in koden i rätt inlämningsmodul på kurshemsidan. All kod ska ligga i en fil som heter `labb3.py`. Funktioner först, sedan huvudprogrammet som kallar på funktionerna. Inga moduler ("bibliotek") får användas, d.v.s. inga `import`. På Uppgift 1 får man inte använda inbyggda funktioner för sortering, som `sort` eller `sorted`.

Muntlig redovisning: Den kod du har lämnat in *kan* behöva redovisas inför lärare vid nästa labbtillfälle.

Kom ihåg: det är alltid bra att tänka på att kommentera koden där det behövs (för att förtydliga syfte med t.ex. en rad kod eller ett kodblock) och läsa igenom din kod innan du lämnar in!

Tips: använd dokumentationssträngar i alla funktioner ni har skrivit så att man enkelt kan få reda på vad indata är och vad funktionen gör.

Laboration 4

Programstruktur

I den här labben ska du strukturera ett givet program, lägga till god felhantering, och utöka funktionaliten med plottning. Du får det relativt korta programmet `batch_means.py` (se mappen `Kod för laborationer/Lab_4/` på kurshemsidan) som går att skriva snyggare. Du hittar testdata i filerna `sampleX.csv`, för $X \in \{1, 2, 3, 4\}$ i samma mapp.

4.1 Lärandemål

- Du ska kunna följa rekommendationer för lättläst kod.
- Du ska kunna skriva om befintlig kod till att följa kodrekommendationer.
- Du ska kunna använda särfall för felhantering.
- Du ska kunna använda ett bibliotek för att plotta data.

4.2 Förkunskaper

För att lösa uppgiften bör du ha läst om tumregler för programmering och hur man skriver bra kod.

4.3 Ej accepterade tekniker

Andra moduler än matplotlib och de som följer med i varje Python-installation ska inte användas.

4.4 Det givna programmet

Programmets uppgift är att läsa in en datafil med (påhittade) data från mätningar, tagna i olika omgångar från olika punkter i planet, och för varje omgång beräkna medelvärdet för mätningar gjorda inom enhetscirkeln. En punkt (x, y) i planet är inom enhetscirkeln om $x^2 + y^2 \leq 1$.

Mätningar tagna utanför enhetscirkeln ska ignoreras. Datafilen har fyra kolumner *separerade med kommatecken*: det är en s.k. `csv`-fil (där "csv" står för *comma-separated values*). Det första talet anger vilken omgång (batch) som en mätning tillhör, andra och tredje talet ger dig x- och y-koordinater för punkten där mätningarna är gjorda, och det fjärde talet är mätvärdet.

Datafilen kan till exempel se ut så här:

```
1, 0.1, 0.2, 73
1, 0.11, 0.1, 101
2, 0.23, -0.01, 17
2, 0.9, 0.82, 23
```

Här finns alltså mätningar från två omgångar, 1 och 2, och det är därför två medelvärden som ska beräknas. Notera att sista mätningen är utanför enhetscirkeln.

4.5 Uppgifter

4.5.1 Uppgift 1: En bättre struktur

Din uppgift är att använda lärdomar från kapitlet "Tumregler för programstruktur och bra kod" i kurskompendiet så att koden blir mer lättläst, lättare att återanvända för andra projekt, samt lättare att hitta fel i. Du bör kunna identifiera flera mindre funktioner utifrån den enda funktion som är given.

Testa din kod på filen `sample1.csv` och egna testfiler. Notera att `sample2–sample4.csv` troligtvis fortfarande inte fungerar; för att få dem att fungera måste man göra Uppgift 2 nedan.

Krav

- Ditt program ska ha *förbättrad* struktur på koden och vara lätt att läsa och förstå.
- Ditt program ska följa kursens tumregler för god programmering.
- Din version av programmet ska inte plocka bort funktionalitet från det givna programmet (oroa dig inte över buggarna/felen ännu).
- Funktionerna ska vara dokumenterade med "docstrings".
- Identifierare ska vara beskrivande.

Tips

- Det gör inget om kodfilen blir längre.
- Gör ändringarna steg för steg och verifiera att programmet fortfarande fungerar i varje steg. Det är svårare att göra en stor ändring med bibehållen funktionalitet.
- Skriv i en kommentar vad du gjort för förbättringar.

4.5.2 Uppgift 2: Felhantering

Om du provar programmet på testfilerna `sample2.csv`, `sample3.csv`, och `sample4.csv` kommer du upptäcka att det finns brister i `batch_means.py`. Du ska därför lägga till felhantering (till ditt förbättrade program från Uppgift 1) med hjälp av `try-except` så att krascher undviks och medelvärdena beräknas så bra man kan önska givet problemen. Du ska inte rätta testfilerna, utan bara anpassa ditt program så att det klarar av dåliga indata.

Krav

- Alla givna testfiler ska kunna analyseras utan problem.
- Om det inte går att öppna den fil som användaren skrivit in ska det påpekas i ett artigt felmeddelande.
- Om det inte går att tolka en rad ska en varning skrivas ut och raden ignoreras (se exempelkörning på Uppgift 4 för exempel på hur en varning kan se ut).
- Ange i kommentarer vilka fel du rättat och hur du gjort det.

4.5.3 Uppgift 3: Sorterade batch-data

Som `batch_means.py` är implementerad så skrivs omgångarnas medelvärden ut i en ordning som beror på i vilken ordning som de givits i indatafilen. Ändra programmet så att utskriften blir sorterad så att medel för omgång 1 skrivs ut före omgång 2, osv. Det vill säga, istället för att en körning av programmet ser ut som

```
Which csv file should be analyzed? sample3.csv

Batch    Average
3        2.0
1        87.0
2        17.0
```

ska resultat bli:

```
Which csv file should be analyzed? sample3.csv

Batch    Average
1        87.0
2        17.0
3        2.0
```

Tips: det är nog bra att använda funktionen `sorted`. Läs mer i Pythons dokumentation.

4.5.4 Uppgift 4: Plotta värdena

Läs på om modulen matplotlib: https://matplotlib.org/stable/tutorials/introductory/pyplot.html

Du kan importera biblioteket genom att lägga till följande kod:

```
import matplotlib.pyplot as plt
```

Obs: för att detta ska fungera kan du behöva installera matplotlib. Detta kan göras i Anaconda eller med hjälp av `pip`. För installationsinstruktioner se https://matplotlib.org/stable/users/installing/

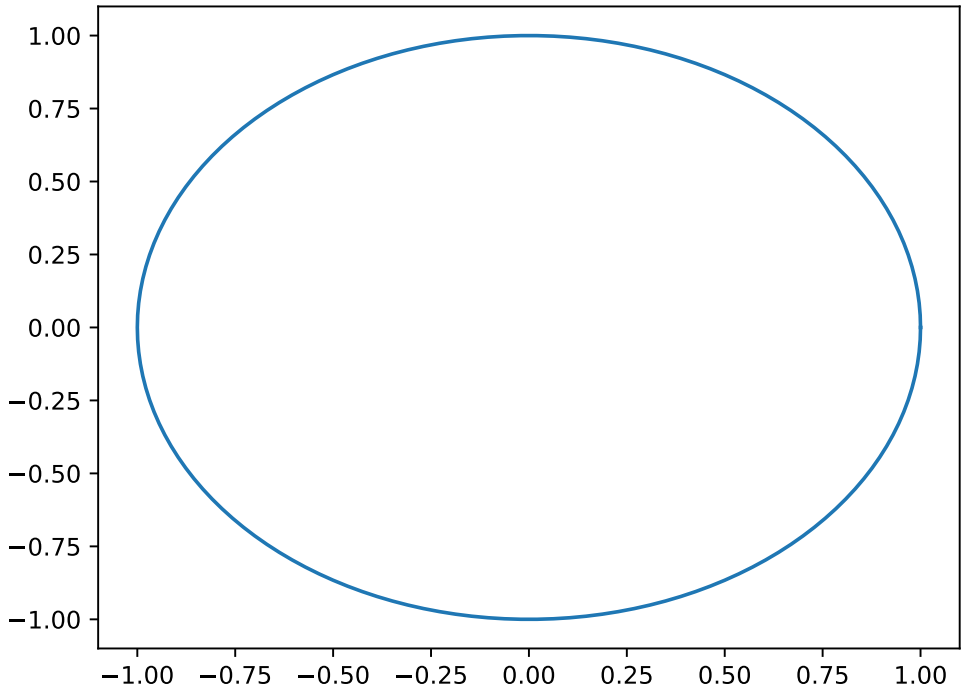
Efter import-instruktionen ovan måste alla funktioner från `matplotlib.pyplot` ha `plt.` framför sitt namn. Så för att komma åt `plot(...)` funktionen måste du alltså skriva `plt.plot(...)`.

Lägg nu även till följande kodsnutt:

```
import math

def plot_data(data,f):

    # Calculate 150 coordinates to draw the circle
    angles = [ n/150 * 2 * math.pi for n in range(151) ]
    x_coords = [ math.cos(a) for a in angles ]
    y_coords = [ math.sin(a) for a in angles ]

    # Draw the circle
    plt.plot(x_coords,y_coords)

    # Here be your code to plot "data"
```

```
    plt.savefig(f + ".pdf")
```

Kodsnutten ovan ska fungera och om ni kör den kommer ett diagram med enhetscirkeln sparas i `f.pdf`. För att se så att matplotlib funkar för er bör ni testa funktionen genom att köra `plot_data(None,'test')` (t.ex. genom att lägga in det temporärt i `main` eller på toppnivå i filen). Koden ovan körs då utan någon data och resultatet sparas i `test.pdf`. Öppnar ni filen i en PDF-läsare ska det se ut så här:



Be lärare om hjälp på ett labbpass om du får problem!

Obs: notera att enhetscirkeln ser lite oval ut. Det beror på att avståndet i y-skalan är mindre än i x-skalan. Detta går att korrigera, men det är inte ett krav.

När ni fått koden ovan att fungera är uppgiften att utöka funktionen så att den plottar den inlästa datan i `data`. Man behöver inte filtrera bort punkter utanför enhetscirkeln och det räcker att plotta data utan att plotta några medelvärden.

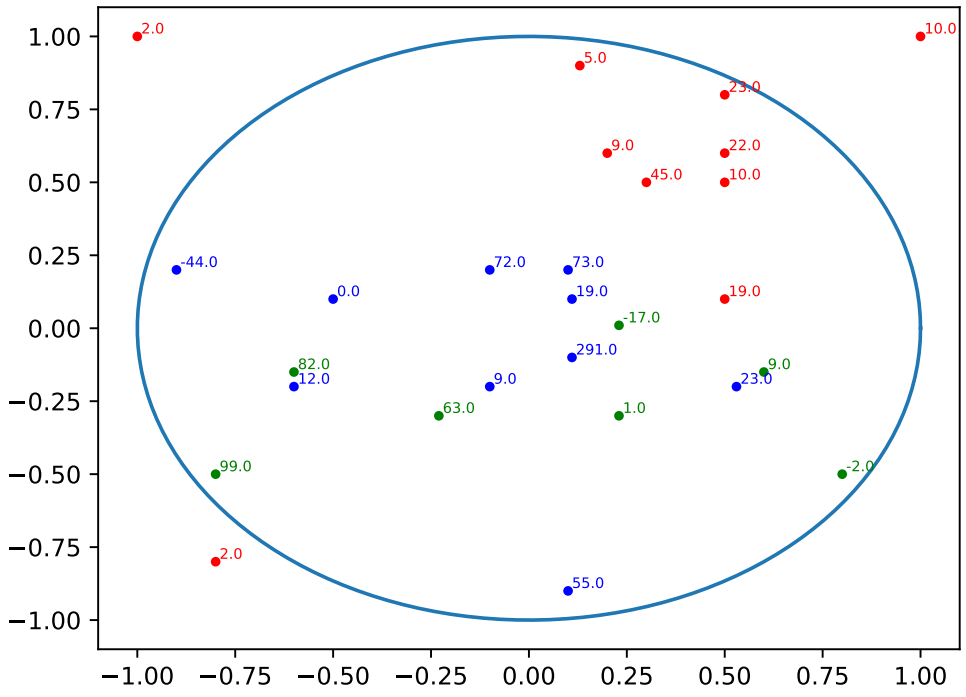
Funktionen `plot_data` ska sedan anropas i huvudprogrammet efter att medelvärdena har skrivits ut, så en körning kan se ut på följande sätt:

```
Which csv file should be analyzed? sample4.csv

Warning: wrong input format for entry: 2  -0.93  -0.01  77

Warning: wrong input format for entry: 2  0.93  -0.01  53
```

```
Batch    Average
1        51.0
2        33.57142857142857
3        19.0
```

```
A plot of the data can be found in sample4.pdf.
```

Filen `sample4.pdf` innehåller då följande diagram:



Krav

- Alla givna testfiler ska kunna plottas utan problem.
- Alla punkter ska ritas med siffervärde bredvid som i exemplet ovan.
- Även punkter utanför enhetscirkeln ska plottas.
- Alla mätningar i samma batch ska ritas med samma färg och alla batcher ska ha olika färg. Dock räcker det om man stödjer ett ändligt antal batcher och det viktiga är att det fungerar för alla testfiler.
- Enhetscirkeln ska fortfarande ritas ut.
- Diagrammets filnamn bestäms från infilen genom att byta ut suffixet. Om indata ges i `file.csv` så ska diagrammet sparas i `file.pdf`.

Man behöver inte plotta några medelvärden, utan det räcker med mätvärdena i filerna.

Tips

- Vad parametern `data` har för typ beror på hur du löst de andra uppgifterna och bör dokumenteras i funktionens *docstring*.
- För att lösa den här uppgiften underlättar det mycket att läsa dokumentation och titta på exempel som kan hittas online.
- På den här uppgiften är det helt OK att använda kod man hittar online, men man bör ha med referenser till var man hittat koden.

4.6 Inlämning

Lämna som vanligt in koden i rätt inlämningsmodul på kurshemsidan. Lämna in lösningen i en fil med det slutgiltiga programmet. Skriv gärna i kommentarer vad du gjort för att lösa varje uppgift.

Muntlig redovisning: Den kod du har lämnat in *kan* behöva redovisas inför lärare vid nästa labbtillfälle.

Obs: på den här labben får man importera `matplotlib` och `math` så att `plot_data` fungerar.

Laboration 5

Objektorienterad programmering

För att introducera objektorienterad programmering så tittar vi på ett beräkningsproblem som dyker upp i modern molekylärbiologi. När man analyserar DNA-sekvenser så vill man ibland veta hur pass två sekvenser överlappar (förenklat: hur pass mycket DNA de har gemensamt). Du ska skriva enkel objektorienterad kod för att stödja sådan analys.

5.1 Lärandemål

- Du ska kunna skriva en egen klass, och metoder som hör till klassen, för att lösa programmeringsuppgifter.
- Du ska kunna använda använda funktioner som argument till funktioner.
- Du ska kunna använda särfall (eng: *exceptions*) för att hantera fel.

5.2 Förkunskaper

För att lösa uppgiften bör du ha läst på om objektorientering och högre ordningens funktioner.

5.3 Ej accepterade tekniker

Du kan använda alla tekniker vi har diskuterat i kursen.

5.4 Given kod

Det finns en kodstubbe `dna.py` på kurswebben som du ska använda som startpunkt. Där hittar du en gnutta kod som ska hjälpa dig att komma igång, samt lite kod för att hjälpa dig testa om du löst uppgiften eller inte. Testkoden har två syften.

(1) Det ska bli lättare och snabbare för dig att avgöra om du gjort rätt. Det är ofta praktiskt att ha testkod av detta slag. I kursen kommer vi att introducera "riktig", systematisk, testning.

(2) Testerna är en kodad specifikation av hur din kod ska fungera. Om anropet `test_all()` skriver ut `Yay, all good` så är du antagligen klar, men kom ihåg att det alltid är bra att testa koden själv!

5.5 Uppgifter

5.5.1 Uppgift 1: Definiera en klass för DNA-sekvenser

Definiera klassen `DnaSeq` som har två attribut: en unik identifierare (även känd som ett *accession*, på engelska) och en DNA-sekvens. Kalla attributen `accession` och `seq`. Det ska vara möjligt att skapa ett DnaSeq-objekt så här:

```
my_seq = DnaSeq('abc123', 'ACGTACGT')
```

Krav

- Din klass ska ha en konstruktor (`__init__`, som givet i startkoden) och två metoder: `__len__` och `__str__`. Metoder med dessa namn har särskild mening i Python: det går att tillämpa funktionerna `len` och `str` på ett objekt från en klass med dessa definierade.
- I denna övning ska `__len__` returnera längden på DNA-sekvensen (dvs, strunta helt i `accession`), och `__str__` ska returnera en "etikett" för praktiska utskrifter på formen `<DnaSeq accession='abc123'>` om tillämpat på exempelobjektet ovan.
- Instansiering med en tom identifierare (`accession`) eller tom DNA-sekvens ska orsaka särfallet `ValueError`. Dvs, anrop som `DnaSeq('', None)` ska resultera i ett särfall.
- Metoderna ska inte skriva ut något, bara returnera värden.
- När du är klar ska anropet `test_class_DnaSeq()` generera utskriften `DnaSeq passed` och inga särfall eller fel ska märkas.

5.5.2 Uppgift 2: Läs DNA-sekvenser från fil

Implementera funktionen `read_dna()`, som givet ett filnamn som parameter returnerar en lista med `DnaSeq`-objekt som representerar sekvenserna i filen. Filformatet är som följer:

- En fil är en rad med "poster" (eng: *records*).
- Varje post innehåller en id-rad (eng: *accession line*) och en sekvensrad.
- En identifierare är unik i filen, representeras av en enkel sträng, och kommer efter ett större-än-tecken (>) som inte är del av identifieraren.
- En sekvens är en sträng av bokstäverna A, C, G, och T.
- Det kan finnas tomrader mellan posterna. Dessa tomrader ska ignoreras. Tips: tomrader har inte längd 0, eftersom det finns alltid ett nyradstecken (skrivs som `\n`) när man läser in en tomrad.

Här är ett enkelt exempel:

```
>s123
ACGGACGT
>abc
GATTACA

>X20
AAAAAAGAATTACCCACACACAC
```

Kodmappen på kurswebben innehåller filer som du kan testa din kod med. Filerna har suffixet ".fa" och är vanliga textfiler som du kan öppna med en programmeringseditor som Spyder och enkla textediteringsprogram som Notepad och TextEdit. Filerna `ex1.fa`, `ex2.fa`, och `sars_cov_2.fa` används av testkoden, så du måste ladda ner dem.

5.5.3 Uppgift 3: En funktion för att detektera överlapp

Vi vill kunna identifiera *överlapp* mellan två DNA-sekvenser med en funktion kallad `check_exact_overlap`. Vi säger att sekvensen a överlappar med sekvensen b om det finns ett suffix till a av någon längd L som är identisk med ett prefix i b av längd L. Definitionen är inte kommutativ: a kan överlappa med b utan att b överlappar med a.

Exempel 1: Här nedan visar vi alla sätt som DNA-sekvensen `AAACCC` kan överlappa med `CCCCGATT`.

```
AAACCC          AAACCC          AAACCC
     CCCCGATT        CCCCGATT        CCCCGATT
```

överlappslängd 1 | överlappslängd 2 | överlappslängd 3

Sekvenserna har inget överlapp längre än 3. Om vi testar överlappslängd 4 och 5, så är det inte ett identiskt överlapp:

```
AAACCC          AAACCC
  CCCCGATT        CCCCGATT
```

Inget överlapp | Inget överlapp

Exempel 2: Jämför sekvenserna från exempel 1 i motsatt ordning. Sekvensen "CCCCGATT" har inget överlapp med "AAACCC", eftersom det inte finns något suffix för "CCCCGATT" som är identiskt med ett prefix för "AAACCC".

```
CCCCGATT        CCCCGATT        CCCCGATT
     AAACCC          AAACCC          AAACCC
```

Inget överlapp | Inget överlapp | Inget överlapp | ...

Exempel 3: `GAGATCAT` och `ATCATTT` har två överlapp, ett med längd 2 och ett med längd 5.

```
GAGATCAT        GAGATCAT
     ATCATTT         ATCATTT
```

överlappslängd 2 | överlappslängd 5

Exempel 4: `AAACCC` överlappar med `ACCC`. Hela `ACCC` överlappar faktiskt med längd 4, och det är det enda överlappet.

```
AAACCC          AAACCC          AAACCC          AAACCC
   ACCC            ACCC            ACCC            ACCC
```

Inget överlapp | Inget överlapp | Inget överlapp | length 4 overlap

Implementera `check_exact_overlap` med tre parametrar: två DnaSeq-objekt (motsvarande *a* och *b* ovan) och en minimilängd. Överlapp kortare än minimilängden ska ignoreras.

Krav

- Parametern för minsta överlapp ska ha skönsvärdet (eng: *default value*) 10.
- Längden på längsta överlapp ska returneras, men längden får inte vara kortare än minimilängd.
- Returnera 0 om inget överlapp hittas.
- Funktionen ska ha en dokumentationssträng.

5.5.4 Uppgift 4: Högre ordningens funktion för att detektera överlapp

Implementera funktionen `overlaps` med två parametrar:

- En lista med DnaSeq-objekt.
- En funktion för att detektera överlapp.

Ett exempel på överlappsfunktion är förstås `check_exact_overlap`, men vilken funktion som helst som tar samma parametrar ska kunna användas. I testkoden så konstrueras alternativa överlappsfunktioner. Om `lst` är en lista med DnaSeq-objekt så ska `overlaps(lst, check_exact_overlap)` vara ett giltigt funktionsanrop.

Alla detekterbara överlapp mellan par av sekvenser i indatalistan ska returneras.

Krav

- Returnera en uppslagstabell (`dict`) med uppslagstabeller (så en "dict of dicts") som innehåller längder på överlapp. Om ett anrop till `overlaps` ger `d` som resultat, och sekvenserna med id `s1` och `s2` överlappar med längd 10, då ska `d['s1']['s2'] == 10` vara sant. Om dessa två sekvenser var enda indata så skulle den nästlade uppslagstabellen kunna se ut så här: `{'s1': {'s2': 10}}`.
- Det ska inte finnas några tomma element i den returnerade uppslagstabellen. Om `s1` inte överlappar med någon annan sekvens så ska man inte hitta `s1` i returdata.
- Inget ska skrivas ut av funktionen.
- Anropet `test_overlap()` ska ge utskriften `overlap code passed` och inga fel ska uppstå när din kod är klar.

5.6 För den nyfikne

- Filformatet vi använder är en förenklad version av formatet "FASTA" som används i stor utsträckning i bioinformatiken. Därav suffixet ".fa" på testdata.
- Man kan säga att DNA har riktning, men i praktiken är denna riktning oftast inte känd. Man måste då överväga både fram- och bakriktning på en DNA-sekvens, och dessutom med så kallat "omvänt komplement" av DNA (eng: *reverse complement DNA*).
- Beräkning av överlapp är kärnan i sammansättning av arvsmassa (eng: *genome assembly*), men man kan då inte titta på perfekta överlapp (som vi gjort här) utan måste acceptera att det finns fel och små skillnader i DNA-läsningarna. Eftersom datamängden i sammansättning av arvsmassa tenderar att vara mycket stor så krävs snabba och smarta algoritmer särskilt anpassade för ändamålet.

5.7 Inlämning

Lämna som vanligt in koden i rätt inlämningsmodul på kurshemsidan. Det räcker att lämna in en fil med den slutgiltiga versionen av koden. Kommentera gärna koden där du har gjort ändringar.

Muntlig redovisning: Den kod du har lämnat in *kan* behöva redovisas inför lärare vid nästa labbtillfälle.