



UPPSALA
UNIVERSITET

OU2, objekt, datahantering

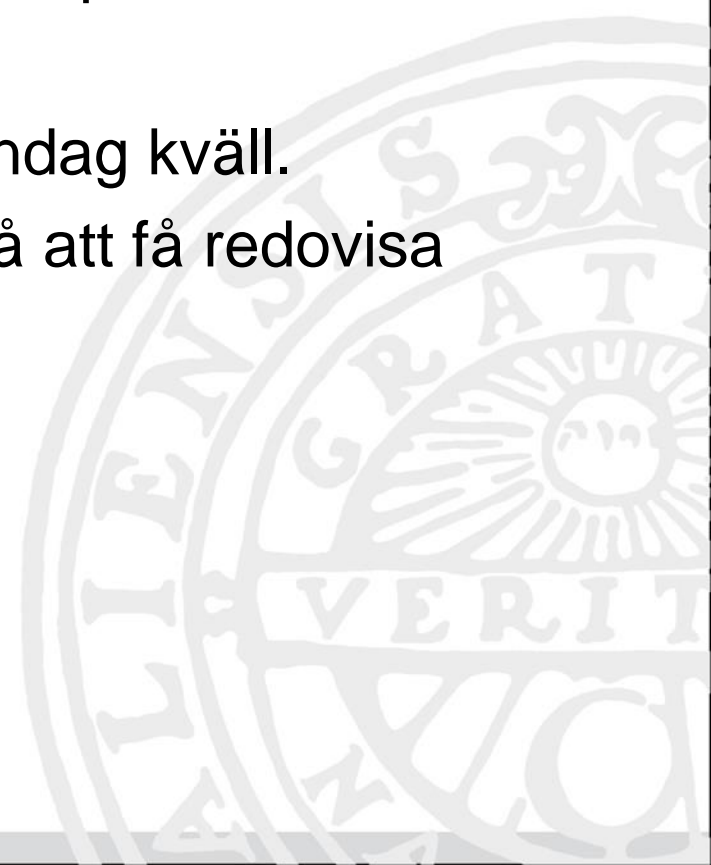
Carl Nettelblad

2020-02-14





- Föreläsningen kan sluta lite tidigare för att ge er tid att fylla i mittkursvärderingen på Studentportalen.
- Annars: Fyll i någon gång före måndag kväll.
 - Exempelvis medan du väntar på att få redovisa OU3.



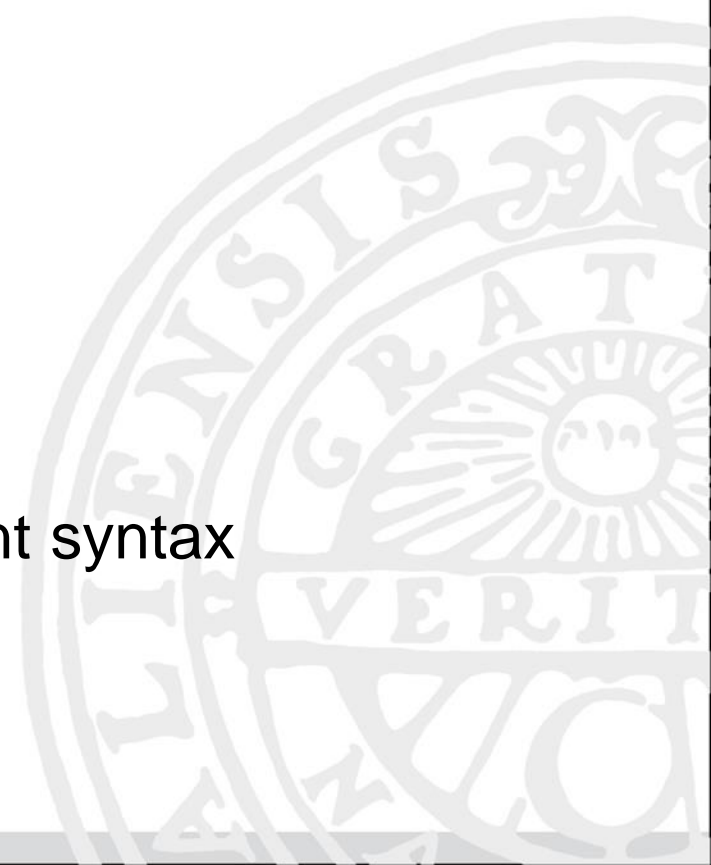


- Skriva egna funktioner för listhantering.
- Använda de funktioner som finns i Python.
- Python är effektivt och elegant när du låter språket hjälpa dig.
 - Det finns språk där man måste skriva egna forslingar för allting.
 - Att låta bli att använda Pythons listhantering är som att ha en motorsåg och försöka såga med den utan att slå på motorn.
 - Man blir bara hindrad av att det inte är tänkt att användas så (långsamt o.s.v.)



Hur kan man tänka?

- Grundstrategier
 - Tänk ut index snyggt
 - "Padda" (från *padding*) listan
 - Dela upp på olika fall
- Genomförande
 - Långt med slingor/if-satser
 - Kort med hjälpfunktioner/elegant syntax





smooth_a

- **Insikt:**
 - Vi vill beräkna medelvärden som om listan fortsatte åt båda sidorna med första/sista elementet upprepat n gånger.
 - Lätt om vi skapar en lista som har n element av $a[0]$, följd av ursprungliga a , följd av n element av $a[-1]$

$a[0]...$

a

$a[-1]...$



Så inte...

```
def smooth_a(a, n):  
    res = []  
    for i in range(len(a)):  
        sum = 0  
        for j in range(i-n, i+n+1):  
            if j < 0:  
                sum += a[0]  
            elif j >= len(a):  
                sum += a[-1]  
            else:  
                sum += a[j]  
        res.append(sum/(2*n+1))  
    return res
```





Utgå från kodexemplet...

```
def smooth_a(a, n):  
    for i in range(n):  
        a.insert(0, a[0])  
        a.append(a[-1])  
    res = []  
    for i in range(n, len(a)-n):  
        res.append(sum(a[i-n:i+n+1])/(2*n+1))
```



```
def smooth_a(a, n):  
    a = a.copy()  
    for i in range(n):  
        a.insert(0, a[0])  
        a.append(a[-1])  
    res = []  
    for i in range(n, len(a)-n):  
        res.append(sum(a[i-n:i+n+1])/(2*n+1))  
    return res
```


Skriv det du tänker

a[0]...

a

a[-1]...

- Syns det att nya a byggs upp så här?
- Vi lägger till ett element i taget.
 - Måste läsa slingan för att se var de hamnar.
- Dessutom är det ineffektivt att lägga till element i början av listan.
 - Alla andra måste flyttas ett steg varje gång.

Skriv det du tänker

a[0]...

a

a[-1]...

```
def smooth_a(a, n):  
    a = [a[0]] * n + a + [a[-1]] * n  
    res = []  
    for i in range(n, len(a)-n):  
        res.append(sum(a[i-n:i+n+1])/(2*n+1))  
    return res
```



Listbeskrivningar

- Syntax för att skapa listor
[*uttryck* for element in *itererbar*]
 - Skapa en ny lista, där vi för varje element i *itererbar* utvärderar *uttryck* och stoppar in det i den nya listan



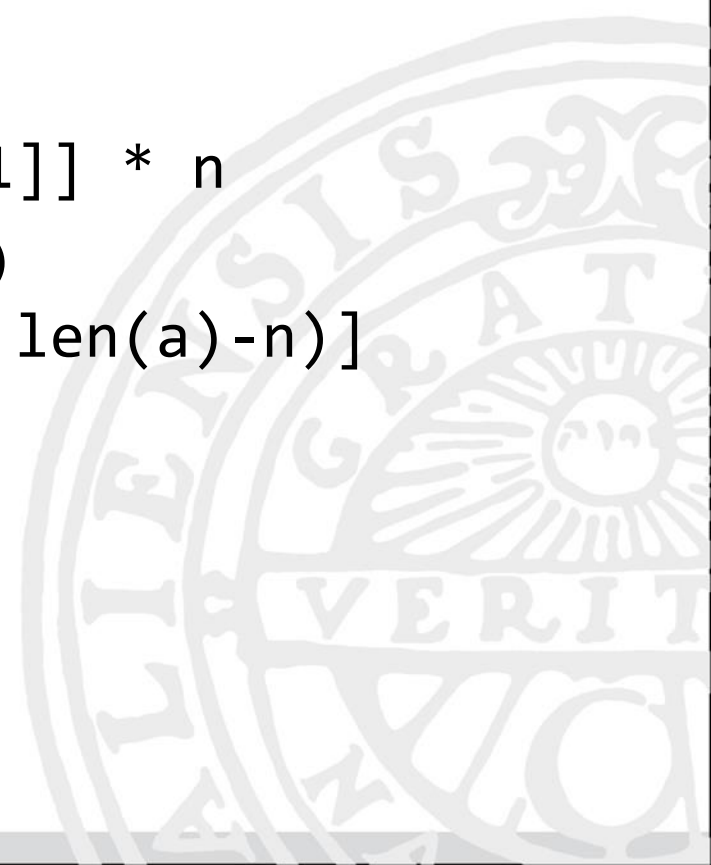


Med mean

```
from statistics import mean

def smooth_a(a, n):
    a = [a[0]] * n + a + [a[-1]] * n
    return [mean(a[i-n:i+n+1])
            for i in range(n, len(a)-n)]
```

- Rätt hjälpfunktion hjälper oss...





Skapa extra lista?

- Padding kan verka ineffektivt
- Men vi skapar ändå en ny lista med medelvärdena vi returnerar
 - "Bara" ungefär dubbelt så mycket
- Tydlig och kort kod
- Alternativet vore att räkna ut hur många extra $a[0]$ och $a[-1]$ som ska läggas till summan för varje element
 - Kan bli snyggt genom att hitta hur mycket som ska läggas till med \min och \max



smooth_b

- Olika nämnare
- Ett sätt
 - Skapa en lista för nämnarna också!

```
def smooth_b(a, n):
```

```
    num = [0] * n + a + [0] * n
```

```
    denom = [0] * n + [1] * len(a) + [0] * n
```

```
    return [sum(num[i-n:i+n+1])
```

```
            / sum(denom[i-n:i+n+1])
```

```
            for i in range(n, len(a)-n)]
```



UPPSALA
UNIVERSITET

Extra listor

- Nu två extra listor, extra summering
- Hur kan vi göra i stället?

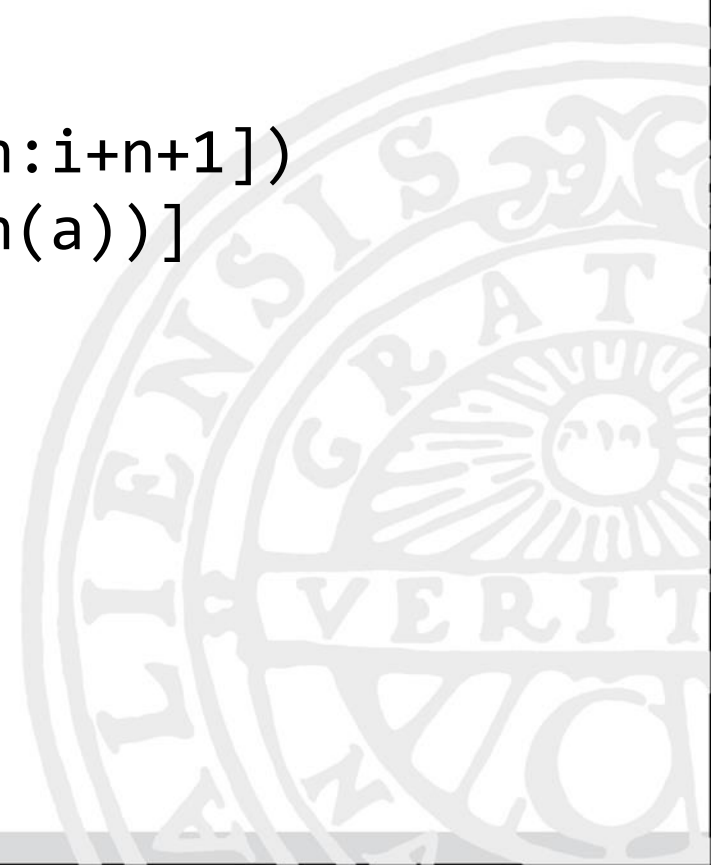




smooth_b

```
from statistics import mean

def smooth_b(a, n):
    return [mean(a[-len(a)+i-n:i+n+1])
            for i in range(len(a))]
```





Kommentar

- $-len(a)$ "bottnar" som startpunkt i intervall på samma sätt som $len(a)$ slår i taket som slutpunkt
- På detta sätt får vi på riktigt ett intervall som börjar som tidigast på element \emptyset och slutar som senast på element $len(a)$
- Vanligt att försöka dela upp i tre fall:
 - Vänster kant
 - "Mitten"
 - Höger kant
- Problem med det! Vad?

Om filer

- Att läsa från en fil innebär att texten *inte* finns inuti ert Pythonprogram
- Ert program har ett filnamn
 - Skapar ett filobjekt
 - Ber det filobjektet att läsa från filen
 - En rad i taget och flytta fram markören
 - Eller hela filen
- Om ert program körs i en annan katalog än filen finns i måste filnamnet ange sökvägen till katalogen
 - Enklast: spara filen där programmet körs
 - Inte säkert att det är där programmet är lagrat

Stänga filen

- Det är lätt hänt att man glömmer att stänga.
 - Särskilt om det kan inträffa något fel, eller man returnerar eller liknande.
 - Med ett `with`-block i Python kan man skapa ett objekt som ska "slängas bort" korrekt när blocket är slut.

```
with open("filen.txt", "r") as fil:  
    helafilen = fil.read()  
print(helafilen)
```



Problem med åäö

- De flesta datorer är numera överens om hur det engelska alfabetet och andra grundläggande tecken skrivs (ASCII-standarden).
 - Inte lika självklart för åäö
 - Ännu mindre självklart för ryska
 - Ännu mindre självklart för kinesiska
 - Vad Python tar som standard beror på operativsystem och språkinställningar
 - Problem om din fil inte sammanfaller med det
- Ibland kan man behöva ange encoding (teckenkodning) manuellt till open
- De två vanligaste för svenska:

```
fil = open("filen.txt", "r", encoding="latin-1")  
fil = open("filen.txt", "r", encoding="utf-8")
```



Läsa lite i taget

- Inte praktiskt att hantera en fil som en enda sträng.
 - Tänk om filen är gigantisk.
- För textfiler ofta rimligt att arbeta rad för rad.
`fil.readlines()` skapar en lista med alla rader
 - Behöver ändå vänta på att hela filen har lästs in innan man kan fortsätta
 - Läsa filer är ofta *mycket* långsammare än allt annat datorn gör
- Ett filobjekt är ett itererbart objekt, så du kan skriva:
`for line in fil:`
- Läser en rad i taget, kör innehållet i slingan



Reguljära uttryck

```
import re
instr = 'här har vi några ord'
ordlista = re.findall(r'[a-zA-ZåäöÄÖÖ]+', instr)
```

- Ett reguljärt uttryck är en "mall" för hur en sträng ska se ut. Här använder vi hakparentes för att ange flera möjliga tecken. Inuti hakarna anger - ett intervall av tecken.
- + har den speciella betydelsen att innebära en eller förekomster av föregående tecken. (ba+b matchar bab, baab, baaab o.s.v.)
- * innebär 0 eller flera förekomster (så ba*b matchar bb, bab, baab o.s.v.)
- Parentes kan innebära grupperingar ((ba)+ innebär ba, baba, bababa, o.s.v.
- ? innebär 0 eller 1 förekomst (så b(aa)?b innebär bb eller baab)



Olika funktioner

- `findall` returnerar en lista med alla icke-överlappande träffar
 - Får du något annat än alla ord på OU3? Kolla ditt reguljära uttryck *noga*. Ett missat bindestreck, ett missat plustecken, ett extra mellanslag kan ge en lista, men något helt annat än alla ord. (Testa!)
- `search` söker efter en match (None om ingen finns)
- `finditer` returnerar ett itererbart objekt med alla icke-överlappande träffar
 - Om du ändå ska köra i en `for`-slinga kan `finditer` vara bättre än `findall`
- `sub` ersätter icke-överlappande träffar på ett reguljärt uttryck med en sträng



Filtrera rader

- ”Hitta alla rader som börjar med !, slutar med ö och innehåller ett mellanslag följt av mer än 6 siffror i följd”

```
with open('filen.txt', 'r') as fil:
```

```
    for ln in fil:
```

```
        if re.search(r'^!.* [0-9]{7,}.*ö$', ln):
```

```
            print(ln)
```

- ^ början på strängen/raden
- . valfritt tecken
- \$ slutet på strängen/raden
- Jämför med att skriva ett eget villkor för att kolla detta...

Reguljära uttryck inte bara i Python

- Många olika språk och bibliotek har stöd för reguljära uttryck
 - Lite olika exakt vilken syntax
- Reguljära uttryck innehåller ofta specialtecken
 - I Python sätter vi `r` före strängen för att markera "rå" sträng, så många specialtecken tappar sin betydelse
- I terminalen kan vi använda kommandot `grep` för att söka i filer med reguljära uttryck

```
grep "^.*[0-9]\{7,\}.*ö$" test.txt
```

- Notera citationstecken och `\` före `{` för att inte specialtecken ska feltolkas



Funktioner som parametrar

- Tänk på `rectangle` och `pentagram`

```
def rectangle(x, y, width, height, color):
```

```
    t = make_turtle(x, y)
```

```
    t.speed(0)
```

```
    t.hideturtle()
```

```
    t.fillcolor(color)
```

```
    t.begin_fill()
```

```
    for dist in [width, height, width, height]:
```

```
        t.forward(dist)
```

```
        t.left(90)
```

Det enda unika för rektangeln!

```
    t.end_fill()
```



Mönster

- Precis som i en slinga har vi saker som ska göras "runt" vår kod
- Annat exempel i OU3:
 - Vi vill sortera, men vi använder en funktion för att styra vad vi sorterar efter





filled_figure

- En funktion med bara skelettet till att rita en fylld figur.
- Vet inte hur själva figuren ser ut!

```
def filled_figure(x, y, painter, color):  
    t = make_turtle(x, y)  
    t.speed(0)  
    t.hideturtle()  
    t.fillcolor(color)  
    t.begin_fill()  
    painter(t)  
    t.end_fill()
```

- Vad är painter för något? Vad har den för typ?



Ny rectangle

- Äntligen har vi en funktion som helt fokuserar på att rita en rektangel.
 - Inte hur man sätter färger eller skapar turtlar.

```
def rectangle(x, y, width, height, color):  
    def drawedges(t):  
        for dist in [width, height, width, height]:  
            t.forward(dist)  
            t.left(90)  
    filled_figure(x, y, drawedges, color)
```

- När vi definierar en inre funktion "får den med sig" parametervärdena från det nuvarande anropet
 - Varifrån kommer t?



Egna objekt

- Vi har skrivit egna *funktioner*
- Sett hur olika moduler kan ha egna *typer*
 - *Klasser*
 - Skapa instanser av klasser; objekt
 - Anropa objekts metoder



En minimal klass

```
class KlassForSig:  
    pass
```

```
kfs1 = KlassForSig()  
kfs2 = KlassForSig()  
kfs1.x = 1  
kfs2.x = 2
```

- Två olika objekt av samma klass
- Vi kan lägga till nya fält i listor, turtlar också

Initierare

```
class KlassForSig:  
    def __init__(self):  
        self.x = random.randint(0,100)
```

```
kfs1 = KlassForSig()  
kfs2 = KlassForSig()  
print(kfs1.x)  
print(kfs2.x)
```

- `__init__` är en "magisk metod" som anropas när ett nytt objekt av klassen skapas



Metoder

- Ser alltså ut som funktioner
- Definieras *inuti* klassen
- Första parametern kallas (nästan alltid) `self`
 - Den refererar till den aktuella instansen

```
def incx(self, step = 1):  
    """Öka instansens x-värde med step,  
    som standard 1."""  
    self.x += step
```



Skriva ut objekt

- Finns fler "magiska metoder", för speciell funktionalitet i Python
- En av de viktigaste är `__str__`
 - Används när du vill ha en strängbeskrivning av objektet
 - Som anrop med funktionen `str`
- En annan är `__repr__`
 - Som när man använder `print`
- Finns också `__format__` för mer avancerad hantering i formatsträngar

Exempel `__str__`

```
def __str__(self):  
    return f'(KFS: {self.x:3d})'
```

```
def __repr__(self):  
    return self.__str__()
```

...

```
print(kfs1)  
print(str(kfs1))
```

- Hur ser det ut när vi inte har dessa metoder?

Definiera och definiera om klasser

- Om vi ändrar `KlassForSig` och lägger till `__str__`
 - Kommer alla nya objekt att ha den nya metoden
 - Gamla objekt kommer att ha den gamla
- När jag definierar klassen igen skapas en "ny" klass med samma namn
- Mest något att vara uppmärksam på medan man testar



Fler magiska metoder

- Går att skriva metoder för vad `+`, `-`, `*`, `/`, `%` o.s.v innebär, vad indexering med `[]` innebär, vad det innebär att iterera över objektet...
- Ett till exempel är `__eq__`, för att jämföra likhet
 - Tar in ett annat objekt
 - Returnerar `True` om objekten är ekvivalenta
- Det verkar rimligt att `kfs1 == kfs2` om och endast om `kfs1.x == kfs2.x` o.s.v.



Många jämförelser

- Totalt 6 jämförelser `__ne__` (\neq), `__lt__` ($<$), `__le__` (\leq), `__gt__` ($>$), `__ge__` (\geq)
- Ofta, men inte alltid, bra att låta dessa vara symmetriska
- Varför så krångligt? Inte självklart att `not x < y` and `not x > y` implicerar `x == y`



Rektangel igen

```
class Rectangle:
    def __init__(self, x, y, width, height):
        self.x = x
        self.y = y
        self.width = width
        self.height = height

    def area(self):
        return self.width * self.height

    def paint(self):
        pass # Inte implementerad än

    def __str__(self):
        return str((self.x, self.y, self, width, self.height))

    def __repr__(self):
        return self.__str__()
```



Pickle

```
rectlist = [Rectangle(0, 0, 100, 50),  
            Rectangle(100, 50, 50, 100)]
```

- Hur gör vi om vi vill spara detta i en fil?
- Vi kan förstås plocka ut värdena

```
with open('rectlist', 'w') as fil:
```

```
    for r in rectlst:
```

```
        print(r.x, r.y, r.width, r.height,  
              file=fil)
```




Skriv till fil själv

```
with open('rectlist', 'w') as fil:  
    print(rectlist, file=fil)
```

- Det var väl inte så svårt?
- Vad är kruxet?
- Läsa in listan igen...





pickle

```
import pickle
with open('rectlist.p', 'wb') as fil:
    pickle.dump(rectlist, fil)
```

Senare, i en annan Pythontolk (på en annan dator?)

```
import pickle
with open('rectlist.p', 'rb') as fil:
    rectlist = pickle.load(fil)
```

- Fungerar på mycket mer komplicerade objekt.
- b betyder att filen inte ska behandlas som text utan **binärt**

Varför dela upp i objekt?

- Strukturera kod ("inkapsling")
 - Variabler och koden som arbetar med dem placeras tillsammans
 - Rita samma rektangel flera gånger
 - Antingen skapa rektangel `r`, anropa `r.paint`
 - Eller spara `rx`, `ry`, `rwidth`, `rheight`, `rcolor` och anropa `rectangle(rx, ry, rwidth, rheight, rcolor)`
 - `Rectangle` använder i sin tur `Turtle`, som också är en klass



Varför dela upp i objekt?

- Likheter mellan klasser

```
class Circle:
    def __init__(self, x, y, r):
        self.x = x
        self.y = y
        self.r = r

    def area(self):
        return self.r ** 2 * math.pi

    def paint(self):
        pass # Inte implementerad än

    def __str__(self):
        return str((self.x, self.y, self.r))

    def __repr__(self):
        return self.__str__()
```





Räkna areor

```
figlist = [Rectangle(0, 0, 50, 150),  
           Circle(200, 200, 20)]  
areas = [f.area() for f in figlist]
```

- Vad som görs när `f.area()` anropas beror på vad `f` är
- En allmän `paint`-funktion kan vara en slinga över alla objekt som ska ritas
 - De kan göra helt olika saker
 - Kan lägga till nya utan att ändra i `paint`



Metod eller funktion

```
with open('rectlist.p', 'rb') as fil:  
    rectlist = pickle.load(fil)  
print(rectlist[0].area())
```

- Vilka funktionsanrop ser vi?
- Vilka metदानrop?
- Det avgörande är om det som anropas är en metod i en klass. Inte om det står en punkt. pickle är en modul, rectlist[0] är ett objekt.
- Metoder kan t.o.m. anropas `Rectangle.area(rectlist[0])`

Comma-separated values (CSV)

- När man ska läsa in data från någon yttre källa är det ibland i form av tabeller
- Finns moduler till Python för att läsa mängder av format, inklusive Excel (som `openpyxl` för Excel specifikt och `pandas` för mängder av format)
- CSV är ett informellt textformat
 - Varje kolumn skiljs av komma
 - Ofta citationstecken runt strängar som i sig innehåller komma
- Modulen `csv` kan hjälpa oss att tolka sådana filer



UPPSALA
UNIVERSITET

people.csv

No, Name, Country
1, Elsa, Germany
2, Erik, Sweden
3, Gregor, Finland

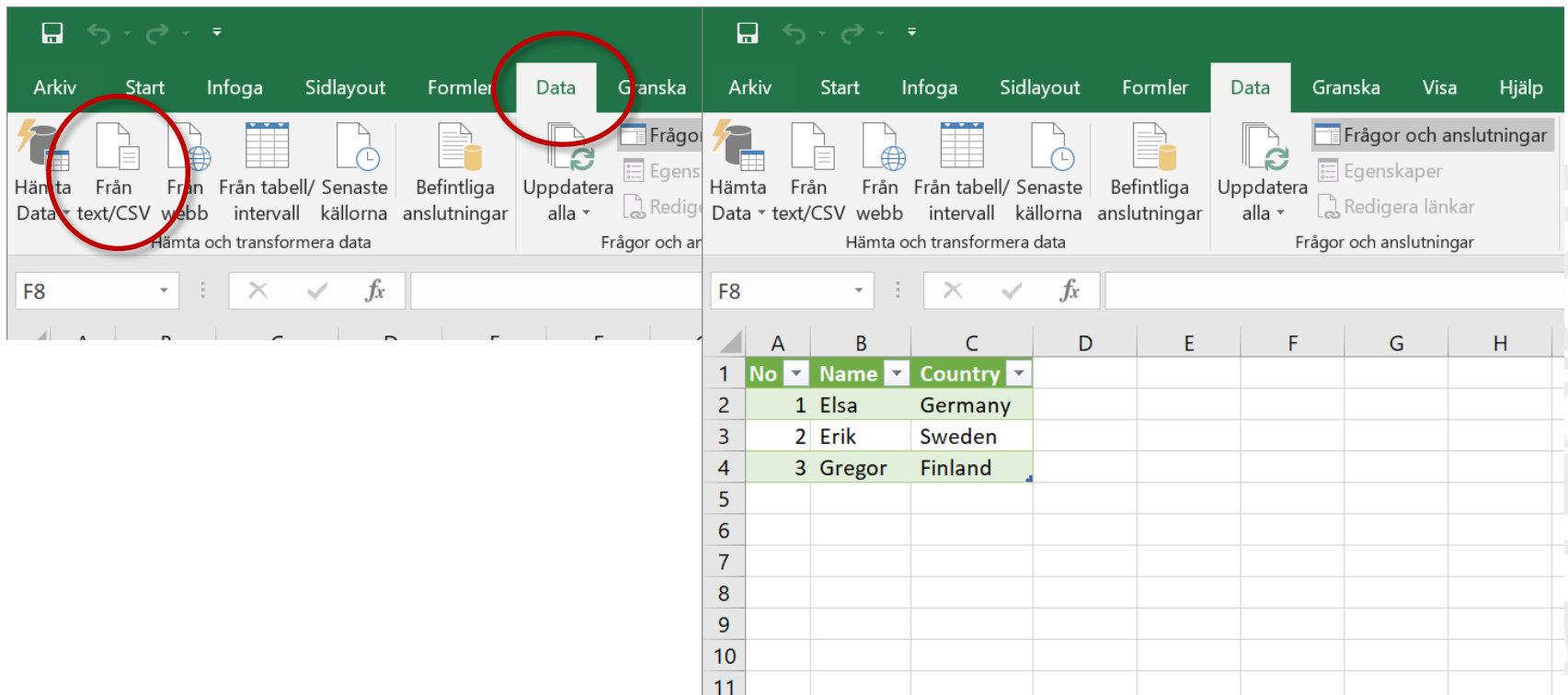




CSV i svenska program

- CSV använder komma
- Flera program för att läsa textfiler använder ditt användarkontos språkinställningar
 - Med amerikanskt språk används . för decimaler och , som avskiljare
 - Med svenskt språk används , för decimaler och ; som avskiljare
 - Excel tolkar en CSV-fil som en semikolonseparerad fil om du bara öppnar den

CSV i Excel



No	Name	Country
1	Elsa	Germany
2	Erik	Sweden
3	Gregor	Finland

- Liknande verktyg för dataimport finns i andra program.



CSV i Python

```
import csv
```

```
with open('people.csv', 'r') as csvFile:  
    reader = csv.reader(csvFile)  
    for row in reader:  
        print(row)
```

- Varje element från reader är en lista med strängar.
 - Är det strängar du vill arbeta med?
- Lika många som antalet kolumner på den raden
 - Snälla filer har lika många kolumner på alla rader
 - Alla filer är inte snälla...

Skippa rubrikraden

```
import csv
```

```
with open('people.csv', 'r') as csvFile:  
    reader = csv.reader(csvFile)  
    for _, _ in zip(range(1), reader):  
        pass  
    for row in reader:  
        print(row)
```

- zip fortsätter så länge *båda* källorna har nya element.
 - Läs en rad och gå vidare.
 - Att ange variabelnamnet `_` är ett vanligt sätt att säga "det här är skräp som jag inte använder"



Fundering

- Kan man skriva följande?

```
with open('people.csv', 'r') as csvFile:  
    reader = csv.reader(csvFile)  
    for row in reader[1:]:  
        print(row)
```

- reader är itererbar, inte indexerbar
- Den kan bara flytta markören i filen framåt, inte flytta till viss rad



matplotlib

- Mycket populärt paket för att generera figurer i Python
- Ingår *inte* i standard-Python
 - Finns med i Anaconda
 - Finns i datorsalarna
 - Kan ofta installeras med pip

Möjligt exempel:

```
python3 -m pip install matplotlib
```

- Svårt att täcka "allt", bäst att titta på befintliga exempel och fylla på med dokumentation



Exempel

- Växelspänning som varierar med tiden

```
import matplotlib.pyplot as plt
from math import pi,sin
L = 100
time = list(range(L))
Voltage = [sin(2*pi*t/L) for t in time]
fig, ax = plt.subplots()
ax.plot(time, Voltage)
ax.set(xlabel='Tid (s)', ylabel='Spänning (mV)',
       title='Exempel 1')
ax.grid()
fig.savefig("test.png")
plt.show()
```



Steg för steg

- Importera moduler med alias
- Skapa de data vi vill plotta (vanlig Python)
- Skapa en figur och ta fram objekten `fig` för hela figuren och `ax` för en yta med axlar (packar upp en tupel)
- Rita i ytan med `ax.plot`, enklaste formen tar x- och y-värden och ritar dem med sammanbindande linjer
 - *Många* möjliga ytterligare parametrar för att styra utseende
- Lägg på etiketter med namngivna parametrar
- Slå på ett rutnät för att lättare läsa av
- Spara till fil
- Visa i fönster (hoppa över i Jupyter)

Olika kurvor med olika utseende

```
fig, ax = plt.subplots()
negV = [-v for v in Voltage]
ax.plot(time, Voltage, color='C0', linestyle='--',
        label='V')
ax.plot(time, negV, color='C1', linestyle=':',
        linewidth=4, label='minus V')
ax.set(xlabel='time (s)', ylabel='Spänning (mV)',
        title='Exempel 2')
ax.set(xlim=(0,100), ylim=(-1,1))
ax.legend()
plt.show()
```



Steg för steg

- Den valfria namngivna parameteren `color` tar färgen. Den kan beskrivas med namn, med värden, eller med ett färdigt färgschema 'C0'..'C9'. [Mer info](#)
- Man kan också ange linjeformat med `linestyle` eller `ls`. Vi ser två exempel. [Mer info](#)
- Linjebredd med `linewidth` eller `lw` som tal
- `label` kan styra etiketter som visas i teckenförklaringen för varje dataserie
- Med `xlim` och `ylim` kan du styra figurens intervall (ange tupel min, max)



Punktdiagram

```
import random

random.seed(20191126)
fig, ax = plt.subplots(figsize=(5, 5))
N = 50
for color in ['tab:blue', 'tab:orange', 'tab:green']:
    x = [200**random.random() for i in range(N)]
    y = [200**random.random() for i in range(N)]
    scale = [500*random.random() for i in range(N)]
    ax.scatter(x,y,s=scale,c=color,label=color,alpha=0.3,
              edgecolors='none')
ax.legend()
plt.show()
```



Steg för steg

- Skapa figur med viss storlek
- Skapa för varje färg 50 x-koordinater, 50 y-koordinater och 500 storlekar
- Rita upp (parametern s anger storlek och c färg)
- Man kan också ändra punkternas form med parametern marker



Byta skala

```
import random
random.seed(20191126)
fig, ax = plt.subplots(figsize=(4, 4))
N = 50
for color in ['tab:blue', 'tab:orange', 'tab:green']:
    x = [200**random.random() for i in range(N)]
    y = [200**random.random() for i in range(N)]
    scale = [500*random.random() for i in range(N)]
    ax.scatter(x,y,s=scale,c=color,label=color,alpha=0.3,
               edgecolors='none')
ax.set_xscale('log')
ax.set_yscale('log')
ax.legend()
```



UPPSALA
UNIVERSITET

Kommentar

- `set_xscale`, `set_yscale` kan hantera [några andra](#) viktiga varianter också



Till sist...

- Fyll i mittkursvärderingen!
- Stänger på måndag (vid midnatt)
- Två labbpass i dag
 - Fler assistenter på plats 15-17 än 10-12
 - Vi använder inte 2507 nu passet 10-12

