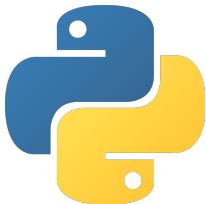


# Eksamensforelesning

TDT4109 Informasjonsteknologi  
Grunnkurs

Opprett anonym bruker på  
Sembly for spørsmål ol.:  
<https://s.ntnu.no/itgk-sporsmal>



# Eksamensforelesning

## TDT4109 Informasjonsteknologi Grunnkurs



24.11.2019  
Magnus Eide Schjølberg



# Les først!

- Dette er **IKKE** en offisiell eksamensforelesning.
  - Jeg vet ingenting om årets eksamen
  - Dette er ikke en fasit
  - Selv om jeg dropper å nevne noe, så betyr ikke det at dette ikke kan komme på eksamen
- Greit å vite:
  - Jeg prøver å forenkle begreper og konsepter slik at det forhåpentligvis blir enklere å forstå
  - Noen ting kan derfor potensielt være litt upresist i forhold til hva som er 100% teoretisk korrekt. I denne forelesningen vil det være fokus på forståelse og praktisk anvendelse.

# Litt om meg

- Magnus Schjølberg, data 3. klasse
  - Undass i ITGK for data/komtek/informatikk etc.
- Tidligere erfaring:
  - Undass i ITGK høst 2018
  - Studass i Java vår 2018 og ITGK høst 2017 og sommer 2018.
  - Holdt tilsvarende eksamensforelesning i ITGK i fjor, Java i vår og noen av øvingsforelesningene i ITGK dette semesteret



Den onkelen du egentlig ikke vil dra på besøk til



## Praktisk info

- Vi holder på i ca. 4-5 timer
  - Vi kjører akademisk kvarter -> pause fra hel til kvart over
- Still spørsmål på Sembly: <https://s.ntnu.no/itgk-sporsmal>
  - Gi tilbakemelding med “give advice”
- Hent slides fra Drive: <https://s.ntnu.no/itgk-slides>



# Praktisk info

- **Si ifra** hvis det går for raskt framover! (“Confused?” på Sembly)
- **Still spørsmål!** Jeg kan ikke garantere et smart svar, men jeg skal gjøre mitt beste :-) (“Ask” på Sembly”)
  - Husk: hvis det er noe du lurer på er det mest sannsynlig flere som lurer på det samme.
  - Dere kan også svare hverandre på Sembly.
  - Jeg svarer gjerne på spørsmål i pausen, men husk at jeg og trenger pause!





# Agenda for dagen

- Generelle tips til programmering og eksamen
- Litt essensiell teori
  - Omregning mellom tallsystemer, toerkomplement
- Variabler og datatyper
  - Variabler, Heltall, flyttall, strenger og operatorer
- Betingelser og logiske uttrykk
  - If, elif, else
- Løkker
  - While, For, Foreach, nøstede løkker
- Lister, mengder, Tupler og Dictionaries
  - + 2D-lister, iterering og tips & tricks
- Strenger, slicing og formatering
- Funksjoner
  - + Random og innebygde funksjoner
- Filbehandling
  - Litt om Pickle-biblioteket
- Unntakshåndtering
- Rekursjon
- Algoritmer og kjøretid



# #Denfølelsen når koden ikke kjører for ørtende gang

Traceback (most recent call last):

```
File "/home/anilil/PycharmProjects/untitled/asd sd.py", line 10, in <module>
```





# Hvordan fikse koden:

- Se etter røde linjer under koden og andre advarsler (hvis du bruker Pycharm eller tilsvarende).
  - Hvis du har røde streker kjører mest sannsynlig ikke koden uansett, så dette er et umiddelbart **rødt** flagg
- Får du opp en feilmelding? **Google** den!
  - Prøv med wildcards (\*) hvis du ikke får treff på første forsøk



# Bruk Stackoverflow!

- Stackoverflow er guds gave til alle som driver med programering
- Hemmeligheten er at ingen egentlig har peiling på hvordan man gjør det meste.
- Alle bare googler det man lurer på.
- Å bli god i programering handler i stor grad om å bli god på å google og finne svaret/koden du trenger



A screenshot of a Stack Overflow question page. The question is titled "Difference between append vs. extend list methods in Python". It has 2808 views, 24 answers, and was asked by Claudio on Oct 31 '08 at 5:55. The accepted answer, by codeforester, explains that `append()` appends objects at the end, while `extend()` extends lists by appending elements from an iterable. A green checkmark indicates the answer is correct.

Difference between append vs. extend list methods in Python

What's the difference between the list methods `append()` and `extend()`?

2808 python list data-structures append extend

share improve this question edited Oct 15 '17 at 4:43 by codeforester 16.6k 7 36 57 asked Oct 31 '08 at 5:55 by Claudio 123k 120 390 572

547 add a comment

24 Answers

`append` : Appends object at the end.

4255 `x = [1, 2, 3]  
x.append([4, 5])  
print (x)`

gives you: [1, 2, 3, [4, 5]]

`extend` : Extends list by appending elements from the iterable.



# Appendiks

- Husk at du får en appendiks med dokumentasjon av fleste funksjoner du trenger på eksamen. Ikke bruk mye tid på å pugge spesifikke funksjonsnavn, fokuser på forståelsen!
- Hvis du står fast på en oppgave, sjekk om noen av funksjonene i appendiks kan være til hjelp!



# Pseudokode

- Hvis det skulle skje at du ikke husker navnet på en funksjon kan du skrive det som pseudokode i stedet
  - Fullt lov å blande python-kode og pseudokode
- Generelt, står du fast, eller har dårlig tid, **skriv pseudokode**. Det er mye bedre å ha masse pseudokode enn tomme svar.
- Husk: **Forståelsen er det viktigste**, og du kan få god uttelling selv om det ikke er 100% korrekt syntaks.
  - **Semantikk > Syntaks**



# Antagelser

- Skriv **antagelser** hvis nødvendig
  - Oppgaver kan være tvetydige/vage, det er helt innafor å gjøre en antagelse, skrive den i starten av oppgaven og løse ut i fra denne
- Pass på at du gjør en *rimelig* antagelse.

- 
- Svar uansett hva!
    - Alt er bedre enn ingenting ;-)
  - Hvis du må gjette helt vilt på teoridelen:
    - Det lengste svaret ofte riktig \*
    - Statistisk sett er B det vanligste korrekten alternativet på tester med fire alternativer (28% av tiden) \*
    - “Ingen av de ovennevnte” og “alle de ovennevnte” er ofte riktig! \*

\* [http://www.bbc.com/future/story/20140905-the-secret-to-acing-exams?ocid=global\\_future\\_rss](http://www.bbc.com/future/story/20140905-the-secret-to-acing-exams?ocid=global_future_rss)



# Hvordan hacke teoridelen 101:

- <https://kramster.it/>
- <https://www.intoit.io/>
- <https://quizlet.com/subject/itgk/>

Mange av teorioppgavene går igjen fra år til år, det er derfor **veldig** lurt å pugge gamle teorisett.





# Generelle tips til teori-pugging

- Pugg alle stegene i Fetch/Execute
- Pugg (litt) hvordan du skal tolke fargekoder (RGB og HEX)
- Pugg antall bits, IPv4/IPv6 og forskjellene der osv.

# Omregning mellom tallsystemer

Desimal - Binær - Heksadesimal

“There are 10 kinds of people in this world:

Those who understand binary and those who don’t.”



# Omregning: binær -> heksadesimal

- Ofte blir man gitt en binærstrenge på dette formatet:
  - 0110 0001 1010 0000
- Hvis ikke er det bare å skrive strengen på det formatet. Legg til en eller flere 0 på starten (venstre) dersom det ikke er partall/ikke går opp.
  - Eks: 11001 -> 1 1001 -> 0001 1001
- Ta fire tall av gangen og regn om til heksadesimal, erstatt tallene med den heksadesimale verdien:
  - 0110 ->  $0^3 + 2^2 + 2^1 + 0^0 = (6)_{16}$



0110 0001 1010 0000



6 1 A 0

**TLDR;** Ta fire og fire tall (fra høyre -> venstre), regn om til heksadesimalt og erstatt posisjonene til de fire tallene med det heksadesimale tallet.

- Hva hvis vi vil gå andre veien? (heksadesimal -> binær)
  - Ta hvert tall, finn det binære ekvivalente tallet (Eks:  $5_{16} \rightarrow 0101_2$ ) og erstatt med dette.



## Omregning: binær -> desimal

- Litt mer tungvint, men helt essensielt å kunne!
- Starter med å omgjøre uttrykket:
  - a. 0 1 1 0 0 1 0 1
  - b.  $0^7 + 2^6 + 2^5 + 0^4 + 0^3 + 2^2 + 0^1 + 2^0$
- Summer dette sammen og du er good:
  - $= (101)_{10}$
  - Kan ikke dele opp uttrykket som fra binær -> hex



# Omregning: desimal -> binær

1. Finn største toer-potens ( $n$ ) som er mindre enn tallet, trekk fra denne potensen og sett på et **1**-tall fra venstre.
2. Ta resterende sum og trekk fra neste toerpotens ( $n-1$ ) hvis denne går opp.
3. Dersom en toerpotens ikke går opp setter du inn **0** for denne posisjonen.
4. Rinse and repeat.



## Eksempel på tankegang

- $101 - 128 (2^7)$  går **ikke** opp, setter en **0** på posisjon 7.
- $101 - 64 (2^6)$  går opp, trekker fra og setter et **1-tall** på posisjon 6.
- $37 - 32 (2^5)$  går opp, trekker fra og setter et **1-tall** på posisjon 5.
- $5 - 16 (2^4)$  går **ikke** opp, setter **0** på posisjon 4.
- $5 - 8 (2^3)$  går **ikke** opp, setter **0** på posisjon 3.
- $5 - 4 (2^2)$  går opp, trekker fra og setter et **1-tall** på posisjon 2.
- $1 - 2 (2^1)$  går **ikke** opp, setter **0** på posisjon 1.
- $1 - 1 (2^0)$  går opp, trekker fra og setter et **1-tall** på posisjon 0.
  - = **0110 0101**



Men hvor er formelen for å regne fra heksadesimal til desimal????

- Lite vits i å pugge alt for mange regler og formler, hvis du skal regne fra heksadesimal til desimal, konverter først til binær og deretter til desimal.
- Samme gjelder (omvendt) hvis vi skal regne fra desimal til heksadesimal





# Cheat codes for å spare tid

- Husk på disse så sparar du gjerne tid når du skal regne om!
- $(\text{1010})_2 = (\text{10})_{10} = (\text{A})_{16}$ 
  - 10 i titallsystemet er lik 1010 i totallsystemet!
- $(0\text{101})_2 = (\text{5})_{10} = (\text{5})_{16}$ 
  - 5 i titallsystemet er det omvendte av 10 i totallsystemet!
- $(\text{1111})_2 = (\text{15})_{10} = (\text{F})_{16}$ 
  - Hvis vi “fletter” de binære tallene for 5 og 10 ender vi opp med 1111, som er  $10 + 5 = 15$ !



# Toerkomplement

- En effektiv måte å representer negative tall på binært
- Fremgangsmåte for å konverte binært -> toerkomplement
  - **Ta det binære tallet, inverter alle siffer og legg på 1**
  - Eksempel:
    - 0101 1010 <- Utgangspunkt
    - 1010 0101 <- Invertert
    - 1010 0110 <- Pluss 1
- Samme fremgangsmåte når vi skal gå fra toerkomplement til binær

## Eksempel: binær til toerkomplement

0000 1000 0000 0001



(0 -> 1 og 1 -> 0)

1111 0111 1111 1110



(+1)

1111 0111 1111 1111

# Tilbake fra toerkomplement til binær

1111 0111 1111 1111



(0 -> 1 og 1 -> 0)

0000 1000 0000 0000



(+1)

0000 1000 0000 0001

← Vi er der vi startet!

# Kjapt om fargekoder

- Hvordan skal vi løse en slik oppgave?
- Husk at tallene (heksadesimal) representerer RGB (red, green, blue):
  - R - G - B -> #RRGGBB
  - 0 = “Inaktiv” farge og F = Fullt “aktiv” farge
- Alle farver aktiv =
  - Hvit
- Alle farver inaktiv =
  - Svart

27

17) Hvilken webfargekode (heksadesimalfargekode) representerer hvit?

- #808080
- #AAAAAA
- #000000
- #FFFFFF

# Lei av å grave deg ned i teoriboka?

The screenshot shows the homepage of the YouTube channel 'Computerphile'. The channel's name, 'Computerphile', is displayed in large green letters against a background of binary code. Below the channel name, there are social media links for Facebook and Twitter. The channel has 1,226,498 subscribers, as indicated by a red 'SUBSCRIBE' button. The navigation bar includes links for HOME, VIDEOS (which is currently selected), PLAYLISTS, COMMUNITY, CHANNELS, ABOUT, and a search icon. The 'VIDEOS' section displays five thumbnail images for recent uploads, each with a title and duration. The thumbnails are:

- 'The Problem with Time & Timezones - Computerphile' (10:13)
- 'How NOT to Store Passwords! - Computerphile' (9:24)
- 'Why Electronic Voting is a BAD Idea - Computerphile' (8:21)
- 'Password Cracking - Computerphile' (20:20)
- 'Hacking Websites with SQL Injection - Computerphile' (8:59)

Below each thumbnail, the video title and duration are listed. The channel also features a 'Uploads' dropdown menu and a 'PLAY ALL' button.



# Anbefalte (eksamensrelevante!) videoer

- Forklaring av flyttall
  - <https://www.youtube.com/watch?v=PZRI1IfStY0>
- Chars og Unicode
  - <https://www.youtube.com/watch?v=MijmeoH9LT>
- Rekursjon (for spesielt interesserte)
  - <https://www.youtube.com/watch?v=Mv9NEXX1VHc>



Minner om...

Still spørsmål på Sembly: <https://s.ntnu.no/itgk-sporsmal>

- Gi tilbakemelding med “give advice”
- **Si ifra** hvis det går for raskt framover! (“Confused?” på Sembly)

Hent slides fra Drive: <https://s.ntnu.no/itgk-slides>

# Variabler og datatyper

“The best thing about a boolean is  
even if you are wrong, you are only off by a bit”



# Datatyper

- Fem grunnleggende datatyper du må kunne:
  - **Heltall** (integer)
  - **Flyttall** (float)
  - **Tegn** (char)
  - **Strenger** (string)
  - **Sannhetsvariabel** / boolsk variabel (boolean)



# Variabler

- Vi kan lagre alle datatypene i **variabler**.
- Vi kan også lagre flere av disse typene av gangen i samme variabel vha. **lister/arrays**
- Obs! Dersom man “kopierer” et objekt ved å legge det i en ny variabel blir det ikke *faktisk* kopiert. Vi lager bare en ny “peker” eller “referanse” til samme objekt.
  - Dvs: Alle endringer gjort i “kopien” vil også gjenspeiles i den originale variabelen

# Boolean

- To alternativer
  - **Sant og Usant**
- Kan *evaluere* om noe er sant eller usant vha. *logiske uttrykk*





# Heltall og flyttall

(engelsk: Integers & floats)

598,0321

- **Integer**
  - Heltall - kan ikke inneholde desimaler.
  - Kan være både negativ og positiv
- **Float**
  - Flyttall, inneholder *alltid* desimaler
    - Veldig forenklet forklaring, bør kunne litt mer på teoridelen!
  - Høy presisjon i utregninger (men ikke veldig relevant for eksamen)
  - Kan rundes av vha. **round(tall, <antall desimaler>)**

# Regneregler og operatorer



- + - \* / (Disse kan dere)
- Kan også sette de foran likhetstegn:
  - += og -= Legger til og trekker fra verdien
  - \*= og /= Multiplisere, dividere

- //

  - Heltallsdivisjon - “hvor mange ganger går dette tallet opp i det andre?”
  - Gir oss et heltall som svar, kontra divisjon som kan gi oss flyttall
  - Eks:  $10 // 4 = 2$  (Vi får plass til fire 2 ganger i ti)

- %

  - Modulo, gir oss resten etter en heltallsdivisjon
    - Eks:  $5 \% 4 = 1$
    - (Hvis vi trekker fra 4 fra 5 står vi igjen med 1 (rest))

- \*\*

  - Opphøyning.
    - Eks:  $5^{**2} = 25$



# Tegn

(engelsk: Char)

- Byggesteinene for tekst
- Kan konvertere enkelttegn til en tallverdi og tilbake:
  - **ord( 'a' )**
    - Konverterer tegnet 'a' til tallverdi: 97
  - **chr(97)**
    - Konverterer fra et tall tilbake til tegnet 'a'



# Strenger

(engelsk: Strings)

- Streng = tekst
- En streng består av flere tegn/char
- Kan sette sammen strenger:
  - `"Hello", "World"` - Skriver ut `"Hello World"`
  - `"Hello" + "World"` - Skriver ut `"HelloWorld"`
- Legge til tekst på slutten av en variabel (append):
  - `tekst += "slutt."`



Minner om...

Still spørsmål på Sembly: <https://s.ntnu.no/itgk-sporsmal>

- Gi tilbakemelding med “give advice”
- **Si ifra** hvis det går for raskt framover! (“Confused?” på Sembly)

Hent slides fra Drive: <https://s.ntnu.no/itgk-slides>

# Betingelser og logiske uttrykk

```
if clueless:  
    facebook.close()  
elif sleepy:  
    coffee.drink()  
else:  
    print("Good job!")
```



# Logiske uttrykk

- Uttrykk er enten sanne eller usanne.
- Tallet 0 er også **False**, alle andre tall er **True**
- `>`, `<`, `==`, `>=`, `<=` Større enn, mindre enn, er lik
- **not**
  - inverterer
- **or**
  - en eller begge
- **and**
  - begge



# if <betingelse>:

- Det første vi tester. Hvis betingelsen er sant kjøres kun dette og ingenting annet i samme struktur, altså kode i ELIF- og ELSE-statements, vil bli kjørt.
- Koden vil heller ikke evaluere videre ELIF-betingelser hvis IF er sann
- Det er helt unødvendig å skrive `if <betingelse> == True`!
  - Bare skriv `if <betingelse>`, betingelsen er i seg selv implisitt enten `True` eller `False`
- `if <betingelse> == False` er ekvivalent med `if not <betingelse>`



## `elif <annen_betingelse>:`

- Forkortelse for “**else if**”
- Evalueres dersom **IF** er usann.
- Kan ha så mange **ELIF**-er som du bare vil.
- Flere **ELIF**-statements evalueres i kronologisk rekkefølge
  - Fra topp til bunn.



## else:

- “If all else fails”
- Kjøres sist av alt, og kan fungere som en catch-all.
- Kan ikke inneholde en betingelse, bruk **ELIF** til dette.

# Alternativ måte å skrive logiske uttrykk

```
if 6 > 5:  
    variabel = "Sant"  
  
else:  
    variabel = "Usant"
```



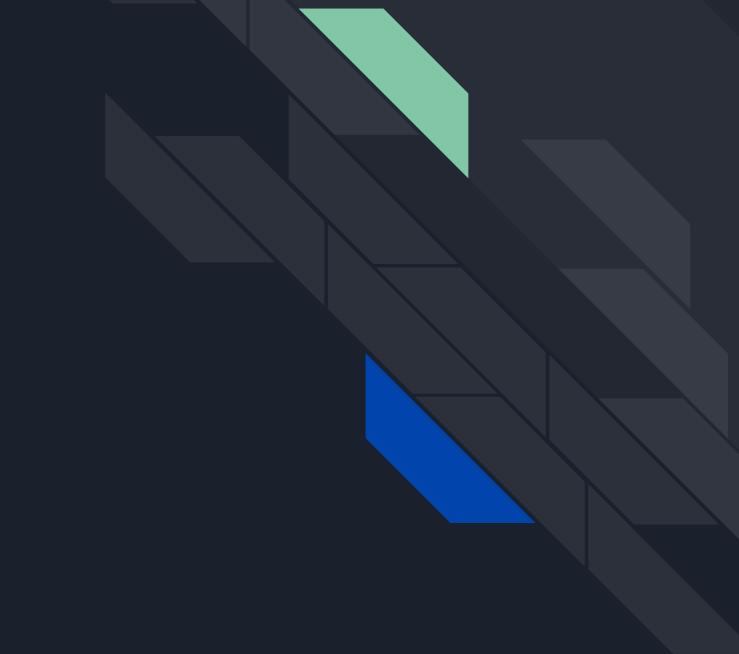
```
variabel = "Sant" if 6 > 5 else "Usant"
```



# Løkker

FOR, WHILE og nøstede løkker

```
while karakter < 'E':  
    pugg_mer()
```





# Løkker Q&A

- **Hva gjør egentlig en løkke?**
  - Kjører samme kodesnutt flere ganger.
- **Hva er en iterasjon?**
  - En iterasjon er når vi kjører koden én gang. Hvis vi for eksempel har en løkke som kjører koden ti ganger vil det si at løkken har ti iterasjoner.





# FOR-løkker

```
for i in range(n):
```

- Kjører et visst antall ganger (iterasjoner), som vi definerer på forhånd.

```
for i in range(10):  
    print(i)
```

- Variabelen **i** er tellevariabelen. Denne økes med 1 (inkrementeres) for hver **itasjon**.



# range(n)

- Definerer hvor mange iterasjoner vi ønsker.
- Kan ta inn ett parameter:
  - **range(10)**, FOR-løkken vil da alltid starte med  $i = 0$  og gå til (men ikke med)  $i = 10$
- Kan også ta inn to parametre:
  - **range(0, 10)**, første parameter definerer da startverdi, siste definerer sluttverdien



# Alternativ FOR-løkke

(Foreach)

- Kan iterere uten tellevariabel.
- Kan iterere over en liste, og trekke ut ett element av gangen fra denne.
  - Løkken stopper når vi har nådd enden av listen.

```
liste = ["Første", "Andre", "Tredje"]
for element in liste:
    print element
```

# WHILE-løkker

`while <betingelse>:`

- Kjører i utgangspunktet ikke et gitt antall ganger, men slutter å kjøre når en **betingelse** er nådd:

```
while not randomNum == 3:  
    randomNum = randInt(1, 5)
```

- Løkken kjører så lenge **randomNum** ikke er lik 3.

# FOR vs. WHILE

- WHILE-løkker er egentlig bare en enklere form for FOR-løkker, vi kan skrive en tilsvarende FOR-løkke ved å bruke WHILE:

```
i = 0  
while i < 10:  
    print(i)  
    i += 1
```

<=>

```
for i in range(10):  
    print(i)
```



# continue, break og return

- **continue**
  - Hopper til neste iterasjon i løkken.
- **break**
  - Stopper hele løkken.
- **return**
  - Hopper ut av løkken, og hopper også ut av funksjonen dersom løkken ligger i en funksjon



# Nøstede løkker

- Løkker som ligger i andre løkker:

```
for i in range(100):  
    for j in range(100):  
        print("indre: " + str(j) + ", ytre: " + str(i))
```

- Løkken på innsiden kjører 100 iterasjoner før den ytre løkken går over på neste iterasjon
- Hvor mange linjer har vi printet etter iterasjonen **i = 5 og j = 20?**



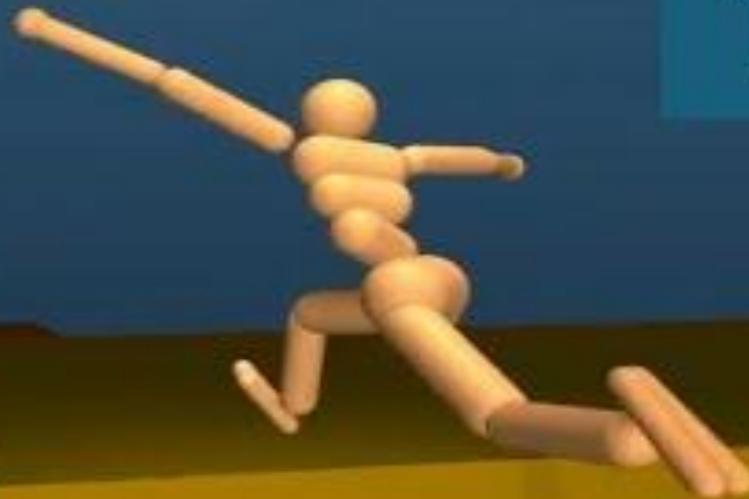
Minner om...

Still spørsmål på Sembly: <https://s.ntnu.no/itgk-sporsmal>

- Gi tilbakemelding med “give advice”
- **Sí ifra** hvis det går for raskt framover! (“Confused?” på Sembly)

Hent slides fra Drive: <https://s.ntnu.no/itgk-slides>

Ti



DEEPMIND AI  
LEARNED HOW TO WALK

# Lister, Mengder, Tupler og Dictionaries

“Why did the programmer quit his job?

- Because he didn't get arrays”



# Lister Q&A

- **Hva bruker vi de til?**
  - Lar oss for eksempel lagre flere forskjellige data/variabler i én variabel
- **Hvorfor ikke bare bruke flere variabler?**
  - Hvis man har verdier som “hører sammen” er det tungvint å manuelt gi de egne navn.
  - Eks: Vi har en liste med målinger som skal brukes til i en graf. Hvis vi skulle gjort det manuelt måtte vi gitt hver enkelt verdi navn som “value\_1”, “value\_2”, etc.
  - Løsning: Legge alle målingene i variabelen “value”, og hente ut verdier vha. **indeksering:** `value[1] = <verdi>`



# Liste / Array



- Kan endre innhold og størrelse (mutable)
- Én liste kan inneholde elementer av forskjellig type, f.eks. tekst, tall og tegn.
  - `min_liste = ["Første", 2, "Tredje", '4']`
- Henter ut verdier ved å bruke indeks:
  - `print(min_liste[0])` skriver ut `"Første"`
- Setter flere lister sammen til én vha. pluss-tegnet:
  - `[1, 2, 3] + [4, 5, 6] => [1, 2, 3, 4, 5, 6]`



# Lister - funksjoner

- `min_liste.append("element")`
  - Legger til "element" på slutten av listen
- `min_liste.insert(index, "element")`
  - Legger til "element" i listen på gitt indeks og flytter elementene til høyre ett hakk
- `element = min_liste.pop(index)`
  - Fjerner elementet på gitt indeks fra min\_liste, og legger verdien inn i variabelen **element**.
- `min_liste.remove("element")`
  - Fjerner første forekomst av "element" i min\_liste



# Mengde / Set

- Fungerer på lik måte som i matematikken, de kan med andre ord ikke inneholde duplikater (to like elementer)
- To måter å definere en mengde på i Python:
  - `mitt_set = {"Første", "Andre", "Tredje"}`
  - `mitt_set_2 = set(["Første", "Andre", "Tredje"])`
- **MEN:** dersom du skal lage en tom mengde er du nødt til å bruke `set()`-funksjonen.



# Litt mer om sets

- Sets kan **ikke** indekseres. Det er derfor ikke veldig lett å hente ut enkeltelementer
- Man kan likevel iterere over sets ved å bruke følgende syntaks:

```
for elem in mitt_set:  
    print(elem) # printer alle elementene i et set
```

- *Veldig* sjeldent man er **nødt** til å bruke sets i dette faget, men kan være greit å vite dersom oppgaven skulle kreve det



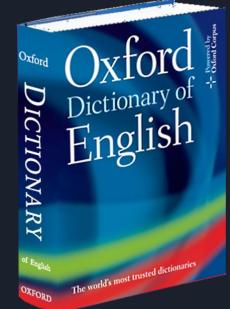
# Tupler

- VELDIG likt lister, hovedforskjellen er at de ikke kan endre størrelse (immutable)
- Defineres slik:
  - `min_tuppel = ("Første", "Andre", "Tredje")`
- Kan for eksempel brukes til å returnere to verdier i en funksjon:
  - `return (first_arg, second_arg)`
- Henter ut verdier ved å bruke indeks (likt som i en liste):
  - `print(min_tuppel[0])` -> skriver ut "Første"



# Kule ting man kan gjøre med lister/tupler/etc.

- `if element in min_liste:`
  - Sjekker om et element ligger i listen.
- `[x for x in range(1, 11)]`
  - Genererer en liste med tall fra 1 til 10
- `[i for i in range(10) if i%2 == 0]`
  - Genererer en liste med kun partall fra 0 til (*men ikke med*) 10
- `min_liste[-1]`
  - Motsatt indeksering, -1 lar oss hente ut siste element i listen, -2 nest siste, osv.



# Dictionary

- Lages vha. krøllparentes (likt som sets).

```
min_dictionary = {  
    "fornavn": "Ola",  
    "etternavn": "Nordmann",  
    "tlf": 1881,  
    "kjonn": 'M'  
}
```

- Det til venstre før kolonet er nøkkelen, det til høyre er selve verdien. Man bruker nøkkelen for å hente ut en verdi (slik som vi bruker indeks i lister og tupler)



# Dictionary

- Hvorfor trenger vi dictionaries?
  - Gir mening når vi må kunne identifisere verdier basert på unike nøkler
  - Nyttig med bursdager, høyde, etc.
- Hvordan bruker vi dictionaries?

```
min_dictionary = {}  
min_dictionary["Magnus"] = "20.10.1996"  
print(min_dictionary["Magnus"]) => skriver ut "20.10.1996"
```

- Kan bruke nesten hva som helst som nøkkel:
  - Tall, Strenger, Tuples, men ikke lister eller dictionaries

# Dictionary .get()

- Kan bruke .get() på en dictionary for å hente ut verdien med en nøkkel:

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
x = car.get("model")
```

=> "Mustang"

- Kan også sette default-verdier i tilfelle nøkkelen ikke eksisterer

```
car = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}  
  
x = car.get("price", 15000)
```

=> 15000



# Hendige funksjoner å bruke med dictionaries

- `if 'name' in dict.keys()`
  - Sjekker om 'name' er en nøkkel i dictionarien "dict"
- `if 'a' in dict.values()`
  - Sjekker om 'John" er en verdi i dict
- `for key, value in dict.items():`
  - Itererer over nøkkel og verdi i en dictionary
  - Kan også iterere over kun nøkler/verdier vha. `keys()` og `values()`



# Todimensjonale Lister

- En liste hvor hvert enkelt element i seg selv er en liste.
- Kan også kalles for en matrise og brukes generelt på lik måte som i matematikken. (Hei Diskmat!)

```
matrise = [  
    [1, 2, 3],  
    [4, 5, 6],  
    [7, 8, 9],  
]
```

- Hvordan indeksere, gitt følgende liste:
  - `list_2D = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]`
  - For å finne elementer i en todimensjonal liste må vi først velge indeks i den ytre (**gule**) listen:
    - `list_2D[0]`
  - Deretter må vi velge hvilken indeks i den indre listen (**grønn**) vi vil hente ut verdien fra:
    - `list_2D[0][2]`
  - Dette henter ut verdien på posisjon **0** i den ytre listen og posisjon **2** i den indre:
    - `list_2D[0][2]` => returnerer **3** som verdi



# Hvordan itererer jeg over en 2D-liste med en løkke?

- Nøstede løkker
  - To indekser -> to tellevariabler -> to løkker
- Hva hvis vi har en **3D**-liste (eller dictionary/tuple etc.)?
  - Samme greia:
    - Tre indekser -> tre tellevariabler -> tre løkker
- Osv.





# Eksempel med 2D-liste og nøstede løkker

- Gitt følgende liste:

```
list_2D = [ [1, 2, 3], [4, 5, 6], [7, 8, 9] ]
```

- I hvilken rekkefølge printes tallene ut i koden nedenfor?

```
for i in range(3):
    for j in range(3):
        print(list_2D[j][i])
```



## Eksempel på “2D-liste”: Liste med dictionaries

```
person = [  
    {"name": "Ola", "Age": 22, "Gender": 'M'},  
    {"name": "Kari", "Age": 45, "Gender": 'F'}  
]
```

```
print(person[0]["name"])      # = "Ola"  
print(person[1]["Age"])       # = 45
```

# Dictionary med liste i feltene

```
student = {  
    "name": "Ola"  
    "age": 22  
    "courses": ["ITGK", "Ex.phil.", "Matte 1"]  
}  
  
print(student["name"])          # = "Ola"  
print(student["courses"][0])    # = "ITGK"  
print(student["courses"][2])    # = "Matte 1"
```

# Viktig!

- Husk at når man “kopierer” variabler så lager man egentlig bare en ny peker til samme objekt. Dette har ikke så mye å si når vi bruker tall (skalar), men med lister (vektor) er det viktig å huske på!
  - Her regnes også strenger som skalar
- Test selv hva som printes ut i de tre kodesnuttene nedenfor

```
a = [1, 2, 3]
b = a
b.pop()
print(a)
```

```
c = 5
d = c
d += 5
print(c)
```

```
e = "Hello"
f = e
f += " world"
print(e)
```



# Hvordan kopiere lister skikkelig

- Bruk **list()**-funksjonen!
  - Lager en **ny** liste, men med samme innhold

```
g = [1, 2, 3]
h = list(g)
h.pop()
print(g)
```



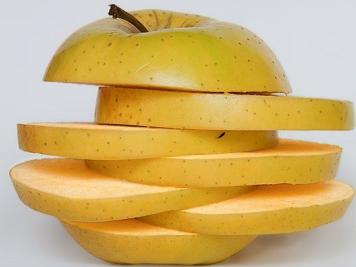
Minner om...

Still spørsmål på Sembly: <https://s.ntnu.no/itgk-sporsmal>

- Gi tilbakemelding med “give advice”
- **Si ifra** hvis det går for raskt framover! (“Confused?” på Sembly)

Hent slides fra Drive: <https://s.ntnu.no/itgk-slides>

# Strenger, slicing og formatering



```
print("Hello World!")
```

# Mer om strenger



- Bruker **str()**-funksjonen for å formatere tall til en streng.
  - fra streng til tall igjen bruker man **int()** eller **float()**
- En streng består av mange *chars* -> dvs. trenger ikke konvertere hvis man skal legge til en char i en streng.
- **lower()**-funksjonen konverterer alle store bokstaver til små.
- Kan bruke **input()**-funksjonen for å ta inn **tekst** fra brukeren via konsollen
  - `navn = input("hva heter du: ")`
  - `alder = int(input("hvor gammel er du: "))`
    - (*må formatere tekst->tall*)



# Streng ≈ Liste

- En streng fungerer ganske likt som en liste
  - Hvert tegn/posisjon i strengen kan sees på som et element/index i en liste
  - Kan iterere med løkker over en streng på samme måte som liste!
  - Kan lage en liste med elementer som tilsvarer hvert enkelt tegn i en streng vha. `list(<streng>)`-funksjonen.
  - Bruk `" ".join(<liste>)` for å få en streng igjen
- Kan ikke endre enkelttegn i en streng, pga. streng er *immutable*. Konverter først til en liste, endre elementet og konverter tilbake.
  - Evt. kan man bruke slicing

# Slicing



- “Kutter” en liste eller en streng. Deler opp i seksjoner
- Hvorfor trenger vi slicing?
  - Trenger dette for å (f.eks.) kutte ut deler av en liste, hente de siste tre elementene osv.
- Slicing gjøres ved å bruke klammeparentes og kolon [:]
  - **liste[x:]** tar med alt fra og med x i listen,
  - **[ :x]** alt fra start og til men ikke med x
  - **[x:y]** fra og med x og til, men ikke med, y.



# Slicing

- Kan også bruke motsatte indekser på slicing:
  - `liste[-3:]` tar fra og med 3. siste element, (siste 3 elementer)
  - `liste[-1:]` tar kun med siste element
- Dobbel kolon `[::]` lar deg gjøre fancy greier
  - `liste[::-1]` - reverserer listen
  - `liste[::2]` - tar kun med hvert 2. element
- Husk at reglene for slicing gjelder både lister og strenger!
  - Ettersom streng er sånn *ish* det samme som en liste med chars



# format( )-funksjonen

- Gir bedre kontroll over formatering av tekst
- Må stå *etter* en tekststreng/variabel, etter punktum.
- Først gir spesifiserer man *hvor* variabler skal stå i en tekst, deretter definerer man *verdiene*
- `"{} er {} år".format("Magnus", 22)`
  - Skriver ut “Magnus er 22 år”
  - Lukket krøllparentes definerer plassering
  - Parameterne i format er verdiene, disse må stå i riktig rekkefølge



# Formateringstegn

- Kan definere typer inne i klammeparentesen og formtere disse:
  - `{:d}` formaterer som heltall
  - `{:s}` formaterer som tekst (streng)
  - `{:f}` formaterer som flyttall
- Dette kan du få bruk for hvis du blir bedt om å kun ta med **n** antall desimaler
  - `{:.2f}` gir flyttall med kun to desimaler osv.
  - `"{:.2f}".format(4.34555)`
    - Returnerer **4.35**
    - Dette runder også av tallet!



## Andre funksjoner som dere kan bruke på strenger (ikke pugg)

- "Tekst".lower() => "tekst"
- "Tekst".upper() => "TEKST"
- len("tekst") => 5
- "Dette,er,en,tekst,med,komma".split(",")
  - => [ "Dette", "er", "en" , "tekst", "med", "komma" ]
- "Dette er en tekst".replace("er", "var")
  - => "Dette var en tekst"

# Funksjoner

+ litt om scope og return

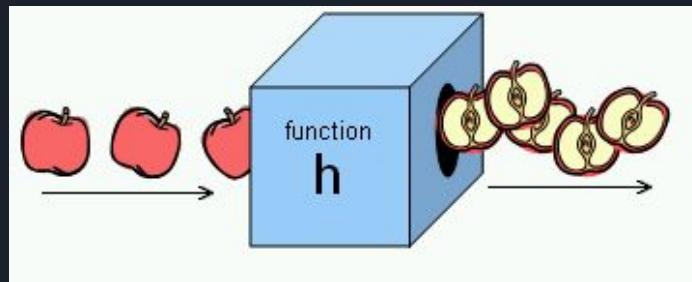
“Functions should do one thing.

They should do it well. They should do it only.”

- Robert C. Martin

# Funksjoner

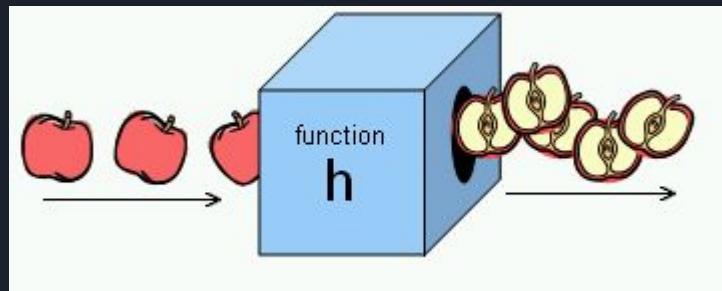
- Funksjoner er nyttige for å dele opp problemer i mindre delproblemer
- Hvis vi har kode vi vet vi kommer til å bruke mange ganger gjør funksjoner koden lettere å lese og mer logisk
- Vi “kaller” på funksjoner for at de skal utføre oppgaver for oss



```
def function(input):  
    output = input / 2  
    return output
```



input      function      output





# Funksjoner - scope

- Scope eller “omfang” på godt norsk
- Funksjoner skal i utgangspunktet bare bruke variablene som blir gitt som input-parametre, eller de som opprettes **lokalt** i funksjonen.
- Verdier kan returneres - “sendes ut” - av funksjonen ved å bruke **return**.



# Flere parametre og retur-variable

- Funksjoner kan ta inn flere parametre, separert med komma:
  - `def function(param1, param2, param3, ...)`
- Funksjoner kan også returnere mer enn én verdi, separert med komma :
  - `return value1, value2, value3`
- Dette er egentlig en tuppel, vi har bare fjernet parentesene ettersom disse er valgfrie.
- For å hente ut de forskjellige returvariablene må man kalle på returverdien med indeks, som en vanlig tuppel/liste.

# Random

- Gir oss *pseudo*-tilfeldige tall.
  - `tall = random.random()`
    - Tilfeldig flyttall mellom 0 og 1
  - `tall = random.randint(0, 5)`
    - Tilfeldig *heltall* mellom 0 og 5
- Hvis du trenger større tall:
  - `tall = random.random()*100`
    - Gir oss tilfeldige flyttall mellom 0 og 100
- Husk `import random`



# Noen hendige innebygde funksjoner

- `max(<liste>)`, `max(var1, var2)`
  - Gir oss største verdi i en liste eller største verdi av to eller flere parametre.
- `min(<liste>)`, `min(var1, var2)`
  - Gir oss minste verdi i en liste eller minste verdi av to eller flere parametre.
- `sum(<liste>)`
  - Returnerer summen av alle tallene i en liste
- *Disse får du mest sannsynlig oppgitt på eksamen (hvis relevante)*



Kan du se noe feil her?



```
def list_multiplier(list_1, list_2):  
    new_list = []  
  
    for i in range(len(list_2)):  
        new_list += list_1[i] + list_2[i]  
  
    return new_list
```



# Husk innrykk!!!

- Dette er alfa-omega i Python!
  - Vær spesielt varsom når du jobber med funksjoner og lister
- Ha kontroll på hvor **return** står



Minner om...

Still spørsmål på Sembly: <https://s.ntnu.no/itgk-sporsmal>

- Gi tilbakemelding med “give advice”
- Si ifra hvis det går for raskt framover! (“Confused?” på Sembly)

Hent slides fra Drive: <https://s.ntnu.no/itgk-slides>

# Filbehandling

Tekst og bytes

“Why did the computer show up at work late?

- It had a hard drive.”



# Filbehandling

- Filbehandling er viktig å kunne og kommer ofte på eksamen
- Pugg én generell metode for å lese fra og skrive til fil, og gjør deg godt kjent med den. Det er en smal sak å endre variabeltype ol. i Python, f.eks. fra tekst til heltall, så gjør det enkelt:
  - Les inn en streng fra fil, og skriv streng til fil.
- Oppgavene om filbehandling er som regel ganske like og handler om å lagre/lese tekst.



# Lesing og skriving til fil

- Du lager et fil objekt med `open()`-funksjonen
  - `f = open("filename", "<r/w/a/>")`
  - **filename** er her navnet på filen og det andre argumentet er *modus*.
  - *Moduser*: “r” leser, “w” skriver over alt som finnes fra før i filen og “a” legger til på av slutten av filen (append).
- Husk å lukke filen etterpå, ved å kjøre `close()` på variabelen vi åpner filen med: `f.close()`

# Gjør det enkelt: filbehandling med `with`

- Skrive til fil:

```
with open("filnavn.txt", 'w') as f:  
    f.write("Hello World")
```

- Lese fra fil:

```
with open("filnavn.txt", 'r') as f:  
    data = f.readlines()
```

- Slipper å bruke `f.close()` med `with!`





# Eksempel: Lese fra fil og printe ut, linje for linje

- Henter inn en fil, leser linje for linje og printer det ut:

```
with open("filnavn.txt", 'r') as fil:  
    for line in fil:  
        print(line)
```

- Kan også ha flere verdier som hører sammen på samme linje, ved å splitte strengen med et skilletegn (for eksempel “ | ”).
- Kan også lese inn alle linjer som en liste med strenger ved å bruke `readlines()`, må da legge det inn i en variabel



# Eksempel: Skrive til fil, linje for linje

- Enkel måte, dersom vi bare skal legge til en ny linje:

```
with open("filnavn.txt", 'a') as fil:  
    fil.write("Dette legges til på en ny linje!")
```

- Kan skrive flere linjer av gangen ved å bruke

```
fil.writelines(liste_med_strenger)
```

- Bruk 'w' i stedet for 'a' hvis du ikke skal ta vare på tidligere innhold i filen!



# Unntaket som bekrefter regelen: fjorårets eksamen

12

## Oppgave 3e – Programmering (5%)

Skriv funksjonen **save\_auction\_data** som tar inn-parameteren **data**. Parameteren data er en to-dimensjonal liste med auksjonsdata formatert slik som beskrevet i den innledende beskrivelsen til Oppgave 3. Funksjonen skal lagre resultatet (return-verdien) av funksjonskallet **all\_winners\_dict(data)** til binærfilen *auction\_winners.db* som ligger i den samme mappen/katalogen som auksjonsprogrammet. Brukeren skal få beskjed om hvorvidt det har lyktes å skrive dataene til binærfilen eller ikke (se eksempel under).

Eksempel på vellykket lagring:

```
>>> save_auction_data(data)
```

*Dictionary data successfully written to file.*

Eksempel på feil ved lagring:

```
>>> save_auction_data(data)
```

*Could not write dictionary data to file.*

```
# Oppgave 3E: 5%
# Save a dictionary to a binary file:
def save_auction_data(data): # 5%
    dict = all_winners_dict(data)
    import pickle
    try:
        f = open("auction_winners.db", 'wb')
        pickle.dump(dict,f)
        f.close()
    except:
        print("Could not write dictionary data to file.")
    else:
        print("Dictionary data successfully written to file.")
```

# Kort og godt hvordan man bruker pickle-biblioteket

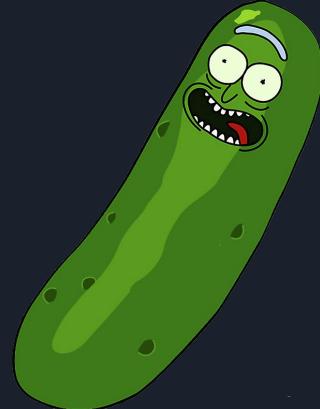
## Pickle Example

[Toggle line numbers](#)

```
1 # Save a dictionary into a pickle file.  
2 import pickle  
3  
4 favorite_color = { "lion": "yellow", "kitty": "red" }  
5  
6 pickle.dump( favorite_color, open( "save.p", "wb" ) )
```

[Toggle line numbers](#)

```
1 # Load the dictionary back from the pickle file.  
2 import pickle  
3  
4 favorite_color = pickle.load( open( "save.p", "rb" ) )  
5 # favorite_color is now { "lion": "yellow", "kitty": "red" }
```



# Exceptions / Unntakshåndtering

“There are two ways to write error-free programs;  
only the third works.”

- Alan J. Perlis



# Try/Except

- Fungerer ganske likt som IF/ELSE
  - IF ≈ Try - betingelsen er at koden kjører uten feilmelding
  - ELSE ≈ Except - dette kjøres dersom en feilmelding blir kastet i Try
- I dette faget bruker vi det egentlig kun i forbindelse med lesing av og skriving til fil.



# Eksempel: Lesing fra fil med Try/Except

```
try:  
    f = open("filnavn.txt")  
    s = f.readline()  
except IOError as e:  
    print "I/O error({0}): {1}".format(e errno, e strerror)
```

# MEN:

- Det er sjeldent veldig stort fokus på oppgaver med try/except, og dersom det ikke står oppført i oppgaven kan du ofte anta at du ikke trenger ta hensyn til dette.
- Unntaket er såklart hvis oppgaven **spesifikt** krever unntakshåndtering!
- Det kan også komme kodeforståelsesoppgaver om dette, så det er lurt å kjenne til!





The Kahoot! logo is centered on a background featuring a stylized silhouette of the world map divided into four horizontal bands of color: red (top), blue, yellow (bottom left), and green (bottom right). The word "Kahoot!" is written in a bold, white, sans-serif font, with an exclamation mark at the end.

Kahoot!



Minner om...

Still spørsmål på Sembly: <https://s.ntnu.no/itgk-sporsmal>

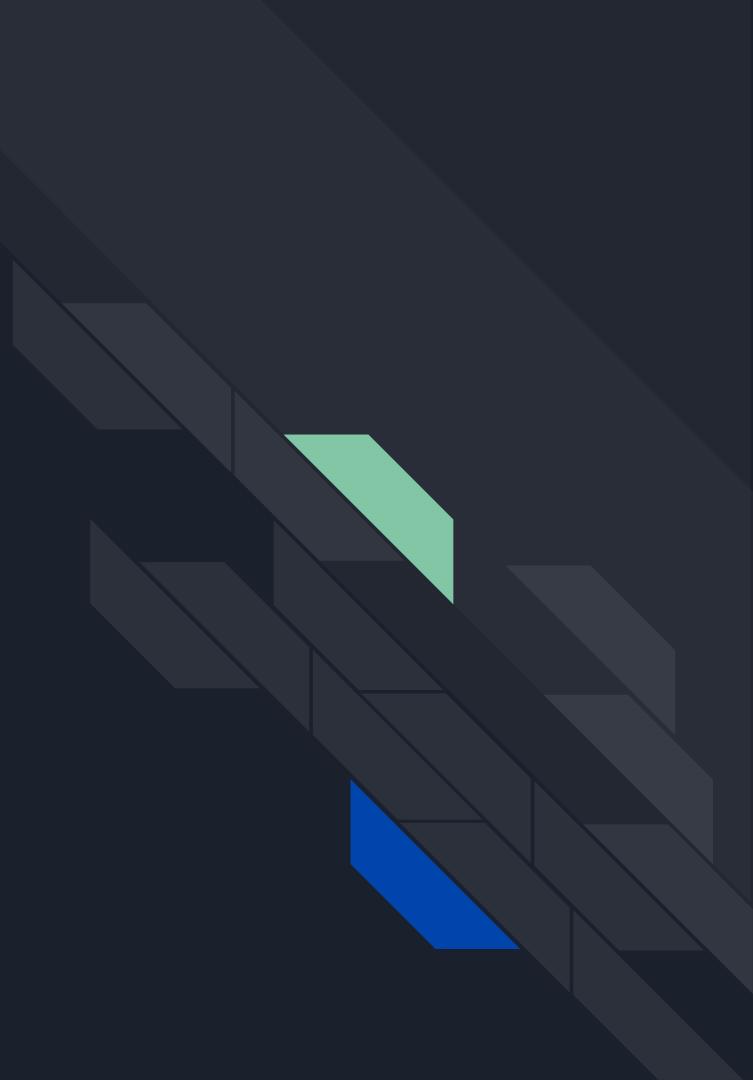
- Gi tilbakemelding med “give advice”

Hent slides fra Drive: <https://s.ntnu.no/itgk-slides>

# Rekursjon

og rekursjon  
og rekursjon

“In order to understand recursion,  
you must first understand recursion”



LEONARDO DICAPRIO

JOSÉPH WATANABE GORDON-LEVITT MARION COTILLARD ELLEN PAGE TOM HARDY CILLIAN MURPHY TOM BERENGER MICHAEL CAINE

YOUR MIND IS THE SCENE OF THE CRIME.



WARNER BROS. PICTURES PRESENTS  
IN ASSOCIATION WITH LEGENDARY PICTURES A SYNCOPY PRODUCTION A FILM BY CHRISTOPHER NOLAN LEONARDO DICAPRIO "INCEPTION" KEN WATANABE JOSEPH GORDON-LEVITT  
MARION COTILLARD ELLEN PAGE TOM HARDY CILLIAN MURPHY TOM BERENGER AND MICHAEL CAINE MUSIC BY HANS ZIMMER EDITED BY LEE SMITH AGE  
PRESERVE  
PRODUCTION DESIGNER GUY HENRY DAVIS DIRECTOR OF PHOTOGRAPHY WALLY PASTERLAK EXECUTIVE PRODUCER CHRIS BRIGHAM THOMAS DULL PRODUCED BY EMMA THOMAS CHRISTOPHER NOLAN  
DESIGNED BY GUY HENRY DAVIS DIRECTOR OF PHOTOGRAPHY WALLY PASTERLAK EXECUTIVE PRODUCER CHRIS BRIGHAM THOMAS DULL PRODUCED BY EMMA THOMAS CHRISTOPHER NOLAN  
WRITTEN AND DIRECTED BY CHRISTOPHER NOLAN INCEPTIONMOVIE.COM

EXPERIENCE IT JULY 16 IN THEATERS AND IMAX



rekursjon



Alle

Google Maps

Videoer

Bilder

Nyheter

Mer

Innstillinger

Verktøy

Omtrent 46 600 resultater (0,56 sekunder)

Mente du: *rekursjon*

## Rekursjon – Wikipedia

<https://no.wikipedia.org/wiki/Rekursjon> ▾

Rekursjon, eller rekursiv funksjon, er i matematikk og informatikk en måte å definere en funksjon på, der funksjonen selv blir anvendt i sin egen definisjon.

## rekursjon – språkvitenskap – Store norske leksikon

[https://snl.no/.../Språkvitenskapens\\_fagdisipliner/Språkvitenskapens\\_grunnbegreper](https://snl.no/.../Språkvitenskapens_fagdisipliner/Språkvitenskapens_grunnbegreper) ▾

12. jan. 2018 - Rekursjon er i språkvitenskapen et syntaktisk fenomen som innebærer at to elementer kan kombineres til en enhet som igjen kan kombineres ...

# Rekursjon Q&A

- **Hva er rekursjon innenfor programmering?**
  - Når en funksjon kaller på seg selv
- **Når må vi bruke det?**
  - I dette faget: kun dersom oppgaven spør om det





## Eksempel: regne ut fakultet

```
def factorial(n):  
    if n == 1:  
        return 1  
    else:  
        return n * factorial(n-1)
```



# Rekursjon vs. iterasjon

- ALL rekursiv kode kan også skrives “iterativt”
  - iterasjon: bruker FOR, WHILE-løkker
- Med mindre oppgaven **spesifikt** spør om det er det like greit å bare bruke løkker i stedet
  - Ofte er det mye vanskeligere å tenke ut rekursive løsninger kontra iterative.
  - Lite å hente på å bruke mye tid på eksamen på å klekke ut fancy løsninger



```
def fibonacci(n):
    if n < 2:
        return n
    else:
        return fibonacci(n-1) + fibonacci(n-2)
```



```
def fibonacci(n):
    a, b = 0, 1
    for i in range(0, n):
        a, b = b, (a + b)
    return a
```

# Hva gjør følgende kode?

```
def recursion(nums):
    if not nums[-1] == 5:
        nums.pop()
        return recursion(nums)
    else:
        return nums

print(recursion([x for x in range(1, 11)]))
```



# Algoritmer

**algorithm** (*noun.*)

Word used by programmers when they do not want to explain what they did.



# Algoritmer Q&A

- **Hva i alle dager er en algoritme?**
  - En slags oppskrift på hvordan man skal løse et problem eller gjøre noe med kode.
- **Hva er “kjøretid”?**
  - Tiden / antall operasjoner det tar for en algoritme å kjøre og behandle et datasett.
  - Worst case: Tiden det tar å kjøre koden, dersom man er uheldig.
  - Average case: Gjennomsnittlig hvor lang tid det tar å kjøre koden.
  - Best case: Tiden det tar å kjøre koden, hvis man er heldig.





# Men hva sører betyr $\Theta(n)$ ?

- En måte å sette grenser for funksjoner i matematikken.
- Brukes også for å betegne kjøretid.



Nå har vi nevnt  $\Theta(n)$ , hva er greia med  $O(n)$ ?

- Folk glemmer hva som er hva, derfor blandes disse ofte sammen
- Ofte når noen sier  $O(n)$ , mener de egentlig  $\Theta(n)$ .
- $\Theta(n)$  brukes i boken. Bare forhold dere til dette.
- Spar dere hodebryet med forskjellene her til algdat

- 
- Hvordan bruker vi denne notasjonen i ITGK?
    - Hvis vi har en løkke som skal gå gjennom en liste med  $n$  antall elementer
      - Kjøretid:  $\Theta(n)$
    - Hvis vi har en løkke som går gjennom en liste og sammenligner hvert element med hverandre :
      - Da trenger vi en nøstet løkke, som betyr at vi må gå gjennom  $n * n$  ganger (her er vi ikke nøyaktig)
      - Kjøretid:  $\Theta(n^2)$



```
a=5  
b=6  
c=10
```

```
for i in range(n):  
    for j in range(n):  
        x = i * i
```

$\Theta(n)$

$\Theta(n)$

$\Theta(n^2)$

```
for k in range(n):  
    w = a*k + 45  
d = 33
```

$\Theta(n)$



# Kjøretid / Tidskompleksitet

- Fra raskest (øverst) til saktest:
  - $\Theta(1)$  - Konstant
  - $\Theta(\log n)$  - Logaritmisk
  - $\Theta(n)$  - Lineær
  - $\Theta(n \log n)$  - Lineær logaritmisk
  - $\Theta(n^2)$  - Kvadratisk / Polynomisk
  - $\Theta(n^3)$  - Kubisk / Polynomisk
  - $\Theta(k^n)$  - Eksponentiell
  - $\Theta(n!)$  - Faktoriell

# Eksempel fra teoridel - eksamen H2016

19.

$$f(n) = 2n^3$$

Hva blir tidskompleksiteten for denne funksjonen?

- a.  $\Theta(n)$
- b.  $\Theta(n \log n)$
- c.  $\Theta(n^2)$
- d.  $\Theta(n^3)$

- Skal finne tidskompleksitet og er gitt et funksjonsuttrykk. Her er det en enkel måte å se svaret på med en gang:
  - Tidskompleksitet = “største” ledd i funksjonen.
    - Merk at dette ikke nødvendigvis trenger å gjelde for små verdier av  $n$ , men det bryr vi oss ikke om





# Bubble sort

- Kjøretid:  $\Theta(n^2)$
- Sammenligner to og to elementer av gangen
- De sorterte elementene “bobler” gradvis til overflaten (høyre side)
- Enkel algoritme, men svært lite effektiv
- Har vært lite om denne på eksamen i det siste



```
def bubble_sort(liste):  
  
    unsorted = True  
  
    while unsorted:  
  
        unsorted = False  
  
        for x in range(0, len(liste)-1):  
  
            if (liste[x] > liste[x+1]):  
  
                liste[x], liste[x+1] = liste[x+1], liste[x]  
  
                unsorted = True  
  
    return liste
```



# MEN:

- Ikke veldig viktig å pugge koden
  - bare husk hvordan algoritmene fungerer!
  - Pseudokode er din venn
- Kan få kodeforståelsesoppgave om f.eks. å kjenne igjen en algoritme

$a[0] \ a[1] \ a[2] \ a[3] \ a[4] \ a[5] \ a[6] \ a[7] \ a[8] \ a[9]$





# Innstikksortering / insertion sort

- Kjøretid:  $\Theta(n^2)$
- Bygger opp en sortert liste fra venstre, ett og ett element av gangen
- Denne har vært mye i forelesning og på eksamen, lurt å kjenne til denne!



```
def insertion_sort(liste):
    for i in range(1, len(liste)):
        element = liste[i]
        position = i
        while position > 0 and liste[position-1]>element:
            liste[position]=liste[position-1]
            position = position-1
        liste[position]=element
    return liste
```

a[0] a[1] a[2] a[3] a[4] a[5] a[6] a[7] a[8] a[9]





# Binærsøk

- Kjøretid:  $\Theta(\log n)$
- Algoritme for å søke etter spesifikke elementer i en **sortert** liste.
- Mer effektivt enn å bare søke fra start -> slutt
- Deler listen i to, og ser etter hvilken halvdel elementet må ligge i.
- Listen må være sortert for at vi skal kunne bruke binærsøk
- Dere bør kunne pseudokoden for denne!

```
def bin_search(liste, verdi, imin, imax):  
    # Heltallsdivisjon  
  
    imid = (imin+imax) // 2  
    if (verdi<liste[imid]):  
        return bin_search(liste, verdi, imin, imid-1)  
    elif (verdi>liste[imid]):  
        return bin_search(liste, verdi, imid+1, imax)  
    else:  
        return imid
```

```
def bin_search(liste, verdi, imin, imax):  
    # Implementation of binary search  
  
    imid = (imin+imax) // 2  
    if (verdi<liste[imid]):  
        return bin_search(liste, verdi, imin, imid-1)  
    elif (verdi>liste[imid]):  
        return bin_search(liste, verdi, imid+1, imax)  
    else:  
        return imid
```

Har vi sett dette før?

Who's my match?



# Repetisjon

I will not kok from LF  
I will not kok from LF





# Hva skrives ut?

```
def mystery(x, y):  
    z = ""  
    for i in range(0, len(x)):  
        if (i % 2) == 1:  
            z += x[i]  
        else:  
            z += y[i]  
    return z
```

```
A = "SUNEAILSUN"  
B = "JALTNCESAES"  
D = mystery(A, B)  
print (D)
```

# Hvilken algoritme er dette?

```
def mystery(liste, verdi, imin, imax):  
    if(imax < imin):  
        return False  
    else:  
        imid = (imin+imax)// 2  
        if (verdi<liste[imid]):  
            return mystery(liste,verdi,imin,imid-1)  
        elif (verdi>liste[imid]):  
            return mystery(liste,verdi,imid+1,imax)  
        else:  
            return imid
```

# Hva med denne?

```
def mystery(liste):  
  
    for i in range(1, len(liste)):  
  
        element = liste[i]  
  
        position = i  
  
        while position > 0 and liste[position-1]>element:  
  
            liste[position]=liste[position-1]  
  
            position = position-1  
  
        liste[position]=element  
  
    return liste
```

# Eksempel fra eksamen 2018

- Start med å se på hvert enkelt parameter som tas inn i funksjonkallet nederst (separert med komma)
- Se deretter innerst i parantesen og evaluér dette.
- Se deretter på if/elif/else-setningen og se hva som blir sant

## Oppgave 2a - Kodeforståelse

```
def myst(val1, val2):
    if(val1 and val2):
        return 'a'
    elif(val1 and not val2):
        return 'b'
    elif(not val1 and val2):
        return 'c'
    else:
        return 'd'
```

Hva returneres ved funksjonskallet under?

```
myst(not(not False and True), not(not False or False))
```



## CASE:

- Vi skal lage en funksjon som tar inn en liste med tall og som skal returnere **False** hvis et av tallene er negativt.
  - Hvordan løser vi dette?



## CASE:

- Vi skal finne største primtall *under* 12 352 og skrive det ut.
  - Hvordan løser vi dette?



## Del opp i delproblem:

1. Hvordan skal vi sjekke om et tall er et primtall?
2. Hvordan skal vi gå gjennom alle tall under 12352?
3. Hvordan skal vi sørge for å returnere et primtall?

# Helt til slutt

“The most disastrous thing that you can ever learn  
is your first programming language.”

- Alan Kay



# “Jeg forstår jo ikke LF engang!”

- Husk at LF = Løsnings*forslag*
- En del LF-er er lite pedagogiske, dvs. de bruker avanserte metoder for å gjøre koden kortere.
- Som regel kan man løse problemet med mye enklere “ordforråd” men litt lengre kode.



# Slapp av!

- De fleste problemer du møter på eksamen kan kokes ned til IF/ELIF/ELSE-statements samt FOR og WHILE-løkker. Er du rutta på disse samt funksjoner kommer du langt bare med dette.
- Det er ingen svar som er for lange! Men forsøk å begrens deg så du ikke bruker all tiden på én oppgave.
- Ingen gyldige svar er for kompliserte eller dårlige! Har du funnet en løsning som funker så får du i utgangspunktet full pott.





## Sist men ikke minst

- ITGK og programmering generelt er et modningsfag.
- Det krever mye mengdetrenings for å bli god.
- Ikke fortvil og gi opp selv om du ikke får det til nå
  - En dårlig karakter i ITGK betyr ikke automatisk at du ikke passer til å holde på med IT.



# Tilbakemelding ønskes!

<https://s.ntnu.no/itgk-sporsmal>

- Trykk på give advice



# Dette klarer du!

- Vi i fagstaben heier på dere!

