

TDT4120

Algoritmer og datastrukturer

Introduksjon til øvingsopplegg og
programmering i Python

Basert på foiler av
Åsmund Eldhuset

Flikket på og
presentert av
Jon Marius Venstad

iDag

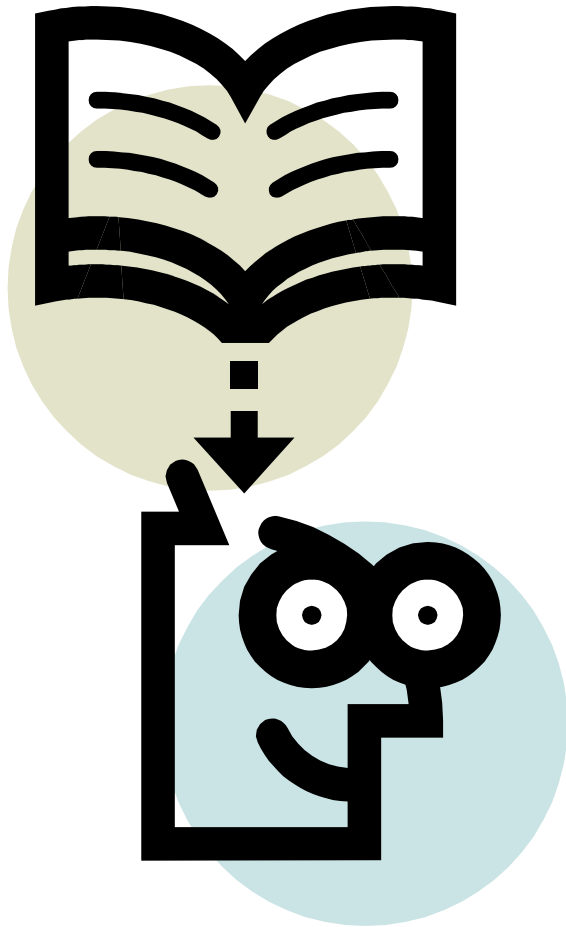
Alg.dat. og de
7 (*3)
gode hjelpere

Øvingsopplegget

Python
vs. Java
Instant hacking



def learnAlgDat(self):



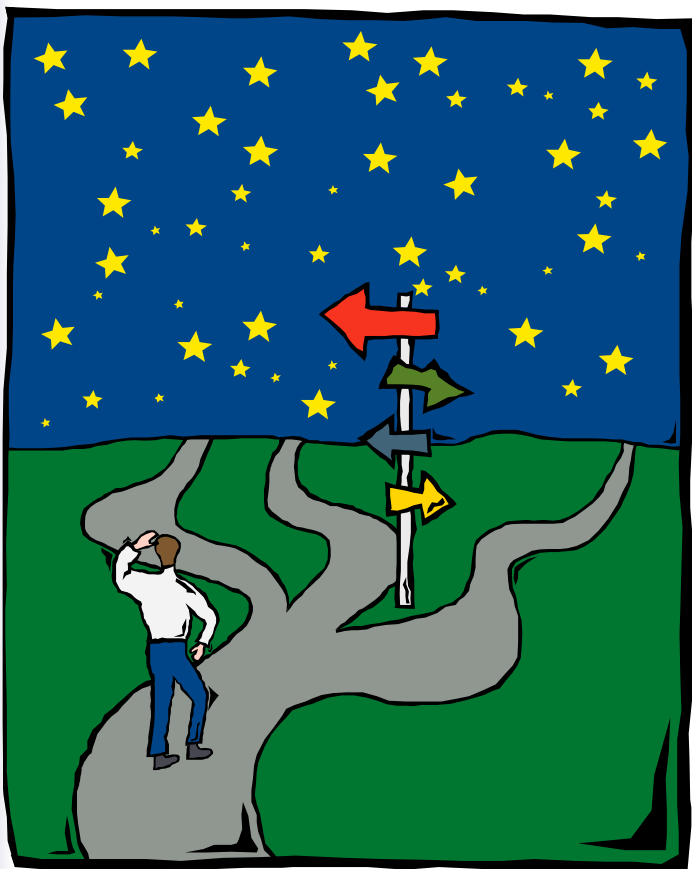
Alg.dat. er et vanskelig fag?

Ja... men vi har noen tips for hvordan man lærer det:

1. Gå på forelesning og få idéene presentert for deg. (Aha-delen)
2. Sett deg ned og implementer algoritmer og strukturer som har vært på forelesning! (Kosedelen)
3. Gjør praksis-øvingene v.h.a. dine implementasjoner. (Egodelen)
4. Nå har du skjont teorien, så gjør teoriøvingene også. (AHA!-delen)

Hvis du står fast...

stud.- og und.-ass.-ene kan



Basics

Teknisk



Idéer og teori

Magic?



Årets und.ass.-er

- Magnus Botnan
- Geir-Arne Fuglstad
- Torbjørn Morland
- Børge Rødsjø
- Jon Marius Venstad
- Kristian Veøy

e-post med spm. kan sendes til din favoritt-ass. eller til algdat@idi.ntnu.no (alle ass. + foreleser)

Studasstimer

- Studasser vil sitte på sal på P15, rom 411 og 424:
 - Mandag 13-16
 - Tirsdag 12-14
 - Onsdag 12-15 (bare rom 424)
 - Torsdag 11-13
 - Fredag 13-16
- Ingen faste grupper; bare huk tak i nærmeste ledige studass
- Studassene vil ha ballonger

Øvingsopplegg

- Én teoriøving (multiple choice) og én praksisøving (programmeringsoppgave) hver uke
- Begge rettes automatisk
- Poengkrav for å få gå opp til eksamen: 30p0 poeng fra hver halvdel av semesteret
 - Hver teoriøving gir opptil 50 poeng
 - Hver praksisøving gir opptil 50 poeng
 - 40 av disse går på korrektheten til programmet
 - 10 av disse er hastighetspoeng som deles ut til de som har skrevet de raskeste programmene
- Det anbefales å gjøre mer enn påkrevd
- Både teoriøvingene og praksisøvingene er eksamensrelevante!

Øvingsopplegg

- Frist for innlevering av øvinger er fredager kl. 10:00 (rett før øvingsforelesningen)
- Program:
 - Tirsdag uke x : Hovedforelesning som introduserer ukens tema
 - Fredag uke x : Øvingen knyttet til teamet presenteres på øvingsforelesningen, og temaet foreleses dypere
 - Fredag uke $x + 1$: Øvingen skal inn; løsningen gjennomgås
- Øving 1 skal inn om en uke!
- Det er lov å gjøre noe helt annet enn det oppgaven sier (det er ofte nødvendig for å være med å kjempe om hastighetspoengene), men du kan *ikke* bruke innebygde funksjoner som gjør deler av arbeidet ditt, f.eks. `sort()`.

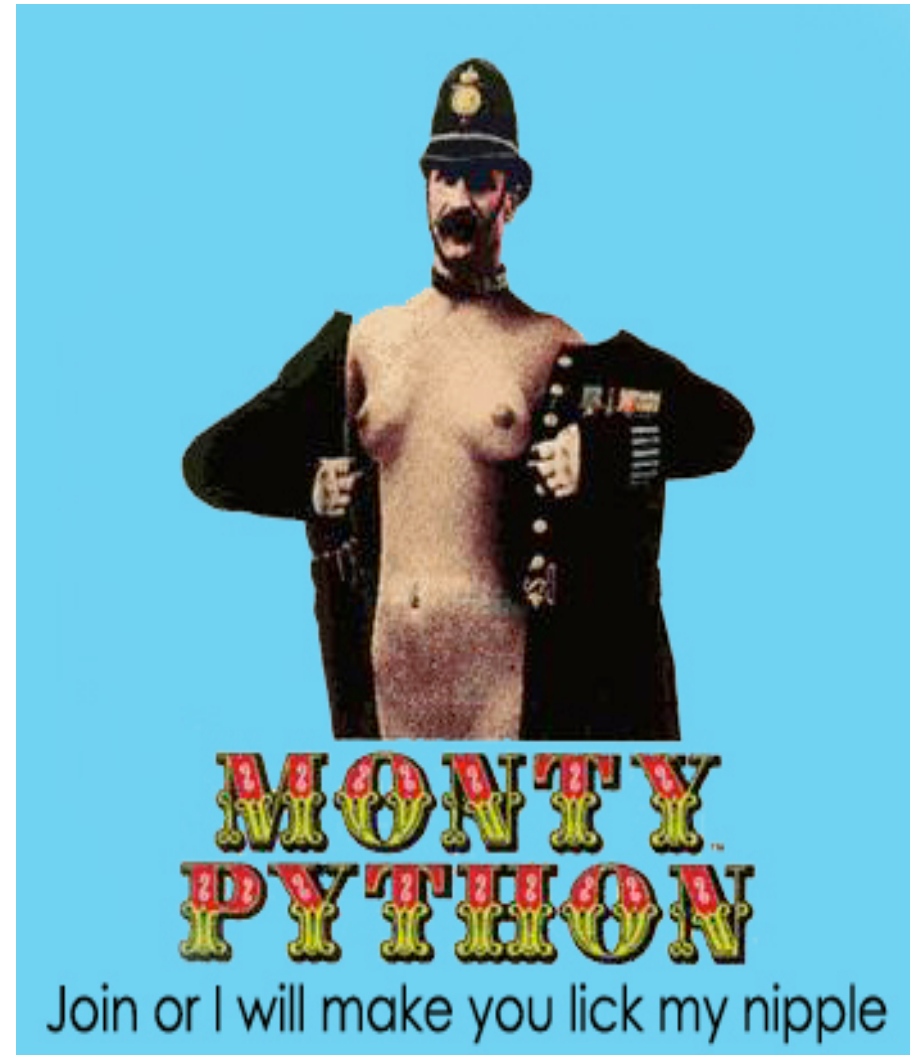
Python!

"A C program is like a fast dance on a newly waxed dance floor by people carrying razors."

"C++: Hard to learn, and built to stay that way."

"Java is, in many ways, C++--"

"And now for something completely different..."



Hello World i Java

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello world!");  
    }  
}
```

Hello World i Python

```
print "Hello world!"
```

Grunnleggende om Python

- Språket er tolket (interpreted), ikke kompilert
 - Koden oversettes til maskinkode mens den kjøres. Dette fører til at Python er langsomt
 - Syntaksfeil oppdages ikke før man "treffer" dem under kjøring
- Dynamisk typing
 - Variabler deklarereres ikke, men opprettes automatisk første gangen de brukes
 - Typen til en variabel kan endre seg underveis, avhengig av hva slags data du legger i den
- Relativt enkel syntaks

Generell syntaks

- Kommentarer startes med #
- Semikolon brukes ikke
- Generelt sett færre parenteser enn i Java
- Indentering og linjeskift har noe å si
 - Må du dele opp en linje, gjøres det ved å sette en \ før linjeskiftet
- Blokker ({ } i Java) startes med et kolon, og indenteringen er det eneste som avgjør hvor blokken slutter
 - All kode på samme nivå må ha samme indentering. *Ikke bland mellomrom og tab!* Pass på at når du kopierer eksempelkode fra øvingene, får du med deg mellomrom!

Om interpreteren

- Startes ved å logge inn på stud og skrive `python`
- Her kan du skrive inn og utføre én kodelinje av gangen, eller større blokker
- Fin til å eksperimentere med
- Veldig fin kalkulator!
- Avsluttes med `Ctrl-D` (`Ctrl-Z`, `Enter` i Windows)
- Skal du lage et større program, bør du lagre det i en tekstfil, f.eks. `program.py`, og kjøre det slik:
`python program.py`
- Hvis du vil kjøre programmet mot den samme input'en flere ganger, kan du lagre input'en i en tekstfil og kjøre det slik:
`python program.py < input.txt` (Python-programmet spiser input-fila, nam nam :)
- I Windows må du skrive `python.exe` i stedet for `python`

Utskrift til skjermen

- `print x` skriver ut verdien av `x` etterfulgt av linjeskift
- `print a, b, c` skriver ut `a`, `b` og `c` med mellomrom imellom på samme linje, etterfulgt av linjeskift
- `print a, b, c,` gjør det samme uten å lage linjeskift etterpå
- hvis du vil unngå mellomrommene, kan du bygge en string av dataene og skrive den ut:
`print str(a) + str(b) + str(c)`
- `print` skriver også ut lister på en fin måte

Input

- Bruker som regel `stdin.readline()`, som leser inn en linje fra terminalen som en string
- Må i så fall skrive `from sys import stdin` øverst i programmet
- `s.strip()` fjerner mellomrom på begge ender
- `s.split()` "deler opp" den originale strengen der den finner whitespace og lager en liste av delene
- `int(s)` konverterer en string til et heltall;
`float(s)` konverterer til et flyttall

if-setninger

- Fungerer på samme måte som i Java, unntatt...
 - Trenger ikke parenteser rundt uttrykkene
 - `elif` i stedet for `else if`
- Støtter sammenligninger av mer enn to operander på en gang
 - Java:
`if (a < b && b == c && c <= d && d < e)`
 - Python:
`if a < b == c <= d < e:`
- Conditionals (kjekt å kunne)
 - Java:
`C ? a : b`
`int min(int a, int b) {return a < b ? a : b;}`
 - Python:
`a if C else b`
`def min(a, b): return a if a < b else b`

if-setninger

- **True** og **False** staves med stor forbokstav
 - De er ikke alltid definerte – skriv i så fall **True = 1** og **False = 0** øverst i koden
- **0**, **[]**, **""** og **None** tolkes som **False**; alt annet tolkes som **True**
- Boolske operatører er annerledes
 - Java: **&&**, **||**, **!**
 - Python: **and**, **or**, **not**
 - **and** og **or** er kortsluttet, akkurat som i Java (hvis mulig, evalueres bare venstresiden)

Funksjoner

- Funksjoner defineres med **def**
- Returtype og argumenttyper spesifiseres ikke, så man kan i utgangspunktet sende hva som helst når man bruker funksjonen (men det er sjelden lurt)

```
def printArguments (a, b) :  
    print a  
    print b
```
- Hvis man ikke returnerer noe selv, returneres **None** (tilsvarer **null** i Java)

Funksjoner

- Alle parametre sendes som referanser – det er altså trygt å sende lister og store objekter
- Tall og strings er immutable (de kan ikke endres), så når man modifierer slike variabler opprettes det nye objekter. "Endringer" av slike variabler inni en funksjon vil derfor ikke synes utenfor
- Lister, dictionaries, egendefinerte objekter og det meste annet er mutable, og endringer av slike variabler inni en funksjon vil endre det originale objektet

Funksjoner

- Fallgruve: Bruker du et variabelnavn på venstre side av et likhetstegn inni en funksjon vil variabelen opprettes dersom den ikke allerede finnes – selv om det finnes en global variabel med samme navn!

- `x = 5`

- `def foo(a)`

- `x = a # Dette er IKKE den samme x'en!`

- Trenger du å bruke en global variabel `x` på den måten, må du skrive

- `global x`

- øverst i funksjonen

Lists

- Som array i Java
- 0-indekserte
- Kan inneholde hva som helst
- Opprette tom liste: *list = []*
- Opprette utfylt liste: *list = [a, b, c, ..., z]*
- Endre element: *list[index] = element*
- Hente element: *list[index]*
- Finne lengde: **len** (*list*)
- Koble på en annen liste: *listA.extend(listB)*

Lists

- Legge til et element bakerst: *list.append(element)*
 - Hvis *element* er en liste, vil hele listen bli satt inn som ett element
- Legge til et element på en bestemt plass (ineffektivt):
list.insert(index, element)
- Lese og fjerne element bakerst: *lst.pop()*
- Lese og fjerne element på en bestemt plass (ineffektivt):
list.pop(index)
- Finne indeksen til første forekomst av et bestemt element (ineffektivt, og krasjer hvis elementet ikke er der):
list.index(element)
- Fjerne første forekomst av et bestemt element (ineffektivt, og krasjer hvis elementet ikke er der):
list.remove(element)

for-løkker og iterasjon

- Vi kan ikke lage for-løkker i Java/C-stil
- Vi kan kun iterere over lister, dvs. gå gjennom alle elementer i en liste
- Syntaks: `for element in list:`
- Løkken vil kjøre like mange ganger som det er elementer i listen, og hver gang vil `element` ha verdien til tilsvarende element i listen
- ```
for e in [0, 42, "hei", [1, 2, 3]]:
 print e
```

# for-løkker og iterasjon

- Hvis vi ønsker en "vanlig" **for**-løkke, kan vi bruke **range ()** til å lage en liste med heltall
- Tre utgaver:
  - **range (end)**  
[0, 1, 2, ..., end - 1]
  - **range (start, end)**  
[start, start + 1, ..., end - 1]
  - **range (start, end, step)**  
[start, start + step, ..., start + x \* step]  
slutter på den siste verdien som er mindre enn end, eller siste verdi som er større enn end hvis step er negativ

# for-løkker og iterasjon

- Iterasjon over elementene i en liste gjør man altså slik:

```
list = [1, 5, 2, 8, 9]
for e in list:
 print e
```

- Du kan da ikke finne indeksene ut fra elementene!
- Iterasjon over indeksene gjør man slik:

```
list = [1, 5, 2, 8, 9]
for i in range(len(list)):
 print i, ":", list[i]
```

- Bruk `xrange` for bedre ytelse (unngår å faktisk lage selve listen, men kan bare brukes i løkker)

# continue, break og else

- **continue** hopper til neste iterasjon av den innerste **for**- eller **while**-løkka
- **break** avbryter den innerste **for**- eller **while**-løkka
- Man kan plassere en **else**-blokk etter en **for**- eller **while**-løkke
  - Den vil bli utført dersom løkka avsluttes på naturlig måte, men den vil ikke bli utført dersom løkka avsluttes med **break**

# Dictionaries

---

- Brukes for å knytte nøkler til verdier
- Kan ses på som et array der indeksene kan være hva som helst (som er immutable)
- Implementert som hashmaps
  - Består egentlig av et array. Ut fra nøkkelen beregnes plasseringen i arrayet. Virkemåten til hashmaps dekkes senere i faget.

# Dictionaries

- Opprette tomt dictionary: `dict = {}`
- Opprette utfylt dictionary: `dict = {key1 : value1, key2 : value2, ...}`
- Lagring av verdi: `dict[key] = value`
- Henting av verdi: `dict[key]` (vil krasje om `key` ikke finnes)
- Sjekke om nøkkel finnes: `dict.has_key(key)`
- Iterere gjennom verdier: `for v in dict.values() :`
- Iterere gjennom nøkler: `for k in dict.keys() :` eller bare `for k in dict :`

# Tuples

- En slags list som ikke kan endres etter at det er opprettet
- Kjekt for å returnere mer enn én verdi fra en funksjon
- `def minAndMax(list) :`
  - `min = list[0]`
  - `max = list[0]`
  - `for x in list:`
    - `if x < min:`
      - `min = x`
    - `if x > max:`
      - `max = x`
  - `return (min, max)`

# Strings

---

- Rammes inn med " eller ' (ingen forskjell)
- Tegn kan escapes på samme måte som i Java (\", \', \n, \t osv.)
- Strings er immutable, så alle "endringer" oppretter i virkeligheten en ny string

# Bruke array som stack

---

- bruk **append** (*element*) for å pushe på stacken (legger element til på slutten)
- bruk **pop** () for å poppe fra stacken (fjerner og returnerer elementet på slutten)

# Bruke array som queue

---

- `pop(0)` fjerner det første elementet
- Men dette er ineffektivt, fordi alle de andre elementene må flyttes ett hakk frem

# Listebehandling

- Sortere stigende: `list.sort()` – bruk dette gjerne til testing, men *ikke* bruk det i praksisøvingene!
- Reversere: `list.reverse()`
- Hente ut en delliste: `list[start : end]`
  - end-elementet blir ikke med
  - `list[: end]` tilsvarer `list[0 : end]`
  - `list[start : ]` tilsvarer `list[start : len(list) ]`
  - `list[: ]` (og alle varianter (over)) gir en kopi av lista!
  - `list[i : i + 1]` lager en liste som bare består av element *i*

# Listebehandling

- Hvis man bruker negative tall i  $[ : ]$ -notasjonen angir man da antall elementer fra slutten av lista
- $a[-x : -y]$  gir fra og med  $x$ te siste element til, men ikke med,  $y$ ende siste element
- Disse kan godt blandes, f.eks. som  $a[x : -x]$ , som gir hele lista minus de  $x$  første og de  $x$  siste

# Mergesort

---

```
def msort(L):
 if len(L) < 2:
 return L
 left = msort(L[: len(L) / 2])
 right = msort(L[len(L) / 2 :])
 return merge(left, right)

def merge(a, b):
 r = []
 while a and b:
 if a[0] < b[0]:
 r.append(a.pop(0))
 else:
 r.append(b.pop(0))
 r.extend(a)
 r.extend(b)
 return r
```

# Listebehandling

- Elegant generering av lister:

```
list = [listElement for element in list if condition]
```

- Tilsvarende kode:

```
resultingList = []
```

```
for element in list:
```

```
 if condition:
```

```
 resultingList.append(resultingElement)
```

- Eksempel:

```
def squareRoots(list):
```

```
 return [sqrt(x) for x in list if x >= 0]
```

# Quicksort

- Fancy quicksort-implementasjon som utnytter dette (hentet fra Wikipedia):
- ```
def qsort(L) :  
    if L == []: return []  
    return \  
    qsort([x for x in L[1:] if x<L[0]]) \  
    + L[0:1] \  
    + qsort([x for x in L[1:] if x>=L[0]])
```
- Velger første element som pivotelement og genererer en liste med alle tallene som er mindre og en med alle tallene som er større
- Kjører qsort på de to listene og kombinerer

Klasser

- `class classname:`

 - `variable1 = startValue1`

 - `variable2 = startValue2`

 - `...`

 - `def methodA(self, arg, ...)`

 - `...`

 - `def methodB(self, arg, ...)`

 - `...`

- Variablene trenger ikke å listes opp på forhånd (men det er lurt å gjøre det!)
- Vi kommer til å bruke klasser kun for å lagre data, ikke for å gjøre fancy objektorientering

Klasser

- Python sitt svar på `this` er `self`
 - `self` må listes som første parameter i metoder
 - `self` må alltid brukes når man skal ha tak i objektvariabler
- Constructoren heter alltid `__init__` (to understreker på hver side)
- Til fysmat og de andre som har lært C++: Python, i likhet med Java, bruker pekere bak kulissene, men eksponerer dem ikke for programmererne. Det finnes derfor ikke noe som tilsvarer C++-operatorene `&` og `*`, og `.` brukes i stedet for `->`

Eksempelklasse

```
class Kubbe:
    vekt = None
    neste = None
    def __init__(self, vekt):
        self.vekt = vekt
        self.neste = None
    def hentNeste(self):
        return self.neste
```

#bruk:

```
k = Kubbe(12)
```

```
m = Kubbe(8)
```

```
k.neste = m
```

```
n = k.neste
```

```
n = k.hentNeste()
```

#evt. neste linje, for å illustrere klasse-funksjon

#*k* sendes automatisk som parameteren *self*

Eksempelklasse

- Tilsvarende klasse i Java:

```
public class Kubbe {
    int vekt;
    Kubbe neste;
    public Kubbe(int vekt) {
        this.vekt = vekt;
        this.neste = null;
    }
    public Kubbe hentNeste() {
        return neste;
    }
}
```

Ressurser

- <http://www.python.org> – Python's hjemmeside
- <http://hetland.org/python/instant-hacking.php>
– Magnus Lie Hetland sin Python-tutorial
- <http://ocw.mit.edu/> generelt, og spesielt <http://ocw.mit.edu/OcwWeb/Electrical-Engineering-and-Computer-Science/6-046JFall-2005/VideoLectures/index.htm> – Open CourseWare fra MIT, med videoopptak av Leiserson som foreleser. Sitat algdatt-studenten som tipset oss: "***MYYYYE*** bedre enn øvingsforelesningene!"