Prosedural del av pensum

- **Vandring gjennom læreboken**
- Hva kreves utover å lese læreboken
  - Hva vil det si å kunne programmere?
  - Løsing og diskusjon av konkrete eksempler på oppgaver

Kapittel 1: Introduction

- Innholder først og fremst bakgrunnsinformasjon for det som kommer senere. Nyttig også i senere emner, og antakelig lettere å forstå nå enn da dere startet.
- «Programmering er problemløsing» [PFE: 1.7]
- Nyttig lærdom: Det første viktige steget i programmering er å omforme problemet til en algoritme, dvs gi det en form som datamaskinen kan løse. Så kan algoritmen skrives i Python

Kapittel 2: Programming with numbers and strings

- [PFE: 2.1] Variabler og tilordninger, numeriske uttrykk og typer
- [PFE: 2.2] Lite vekt på behandling av matematiske uttrykk i IN1000 – men viktig og nyttig for noen av dere senere
- [PFE: 2.4] Strenger, konvertering og streng-metoder
- [PFE: 2.5] Innlesing fra tastatur (input)
- [PFE: 2.6] Enkel grafikk - ikke pensum

Kapittel 3: Decisions

- Dette kapittelet tar for seg det som har med valg å gjøre
  - [PFE: 3.1] if-setninger
  - [PFE: 3.2] Relasjonsoperatorer (<, >, == osv..)
  - [PFE: 3.3] Nøstede if-setninger
  - [PFE: 3.7] Boolske operatorer (and, or), uttrykk og variable
- Alle må kunne bruke if-setninger i programmering!
- [PFE: 3.5] Flytskjemaer - ikke direkte pensum i seg selv, men kan være nyttig for å presist forstå kodeflyten
- Merk at det er flere special topics i kapittelet som ikke er pensum (men som likevel kan være nyttige)

Kapittel 4: Loops

- Dette er uunnværlige redskaper i en programmerers verktøykasse - nesten alle programmer inneholder en løkke
- [PFE: 4.1] while-løkker går så lenge en betingelse gjelder
- [PFE: 4.3] Viktig å skjønne hvordan kode presist kjører!
  - *(bruk gjerne blyant og papir selv, eller PythonTutor)*
- [PFE: 4.6] for-løkker går gjennom hvert element i en samling (f.eks. en liste)
  - Et spesialtilfelle er å gå gjennom en samling tall som kan brukes som indekser i en annen liste: range(0,len(min_liste))
- [PFE: 4.7] Nøstede løkker *(vi så på det ifbm data fra fil)*
- [PFE: 4.3/5/8-11] Det er mange "applications" og lignende som ikke er direkte pensum, men som kan være nyttig

Kapittel 5: Functions

- Nå begynner det å bli mer avansert. Alt i kapittelet er sentralt pensum (unntatt rekursjon [PFE: 5.10])
- [PFE: 5.1] om funksjoner som «svarte bokser» for lettere å holde oversikten over programmet.
- [PFE: 5.2] Hvordan lager og bruker vi funksjoner
- [PFE: 5.3] Parametre - noe av det aller viktigste i hele kurset!
- [BJ: 5.4–5.5] Returverdier (det som skiller funksjoner fra prosedyrer)
- [PFE: 5.7] Hvordan funksjoner bør være.

Kapittel 5: Functions *(forts)*

- [PFE: 5.7] Problemløsing med stegvis forfining beskriver en god teknikk til å la et program bli til litt etter litt.
- [PFE: 5.8] En variabels skop er delen av et program hvor en variabel er tilgjengelig.
  - Enkelt sagt: En lokal variabel i en funksjon er bare tilgjengelig når denne funksjonen utføres
  - Tenk også i forhold til OO

Kapittel 6: Lists

- Lister er en veldig mye brukt datastruktur.
- [PFE: 6.1] introdusere lister.
- [PFE: 6.2] viser ulike operasjoner på lister (hvor kun appending Elements er direkte pensum)
- [PFE: 6.3] gir eksempler på bruk av lister
- [PFE: 6.4-6.7] er ikke direkte pensum, men
  - poenget i ST3 (i 6.4) må man ha fått med seg (at f.eks. en liste sendt inn som parameter kan bli endret)
  - Kapittel 6.6 er veldig nyttig (også for eksamen) om hvordan tenke når man skal finne løsning til et problem.

Kapittel 7: Files and exceptions

- [PFE: 7.1-7.2] Hvordan lese fra og skrive til filer. Vi har i hovedsak basert oss på iterering av linjer (7.2.1)
- [PFE: 7.3-7.6] Ikke pensum. (kommandolinje-argumenter, binærfiler, unntakshåndtering)

Kapittel 8: mengder (set) og ordbøker (dict)

- [PFE: 8.1] Mengder er ikke viktig del av pensum, men selvsagt lov å bruke og kan være nyttig
- [PFE: 8.2] Ordbøker (dict) er viktig del av pensum, og nyttig i mange sammenhenger
- [PFE: 8.3] Mer komplekse strukturer, f.eks. ordbøker med lister som verdier. Også brukt sammen med OO (mer detaljer i senere slides).

Prosedural del av pensum

- Vandring gjennom læreboken
- **Hva kreves utover å lese læreboken**
  - Hva vil det si å kunne programmere?
  - Løsing og diskusjon av konkrete eksempler på oppgaver

Hva vil det si å kunne programmere?

- Mer spesifikt:
  - Hva er det dere forventes å kunne etter IN1000?
- Eller sagt på annen måte:
  - Hva er det dere må kunne for å gjøre det godt på eksamen?

Programmering er en ferdighet!

- Målet er å kunne anvende programmering til å løse problemer
  - Å forstå de ulike begrepene (som if og while) er en forutsetning, men ikke tilstrekkelig
  - Å lese boka er ikke nok - man må også trene på å løse mange ulike problemer
  - Alle skriftlige læremidler kan tas med på eksamen - ferdigheten må man ha opparbeidet selv

Programmeringens natur

- Fra første time:
  - Software development happens in your head, not in an editor. Programming is all about problem solving. It requires creativity, ingenuity, and invention”
    (Andy Hunt)
  - "Computer science education cannot make anybody an expert programmer, any more than studying brushes and pigment can make somebody an expert painter"
    (Eric Raymond)

Eksempel på oppgave som involverer kreativitet

- Skriv en funksjon **forkort_lagliste(lagliste)** som tar som argument en liste av strenger som er lagnavn og returnerer en ny liste hvor ingen lagnavn finnes flere ganger i lista. Med andre ord, dersom samme streng opptrer flere ganger i lista som funksjonen mottar som argument, skal denne bare opptre én gang i lista som blir returnert. For eksempel skal kallet f*orkort_lagliste(["Brann", "Molde", "Brann"])* returnere en liste *["Brann", "Molde"]*.

(fra tidligere eksamen)

Eksempel på oppgave som kun krever presis forståelse

Hva skrives ut på skjermen når følgende kode utføres?

```
class Tall:
    def __init__(self, a):
        self._a = a
    def m1(self, c):
        self._a = self._a + c
    def m2(self):
        self._a = self._a * 2
    def m3(self):
        return self._a + 10

t1 = Tall(5)
t2 = Tall(2)
t1. m2()
t2.m1( t1.m3() )
print( t2.m3() )
```

Prøv selv

(lett omskrevet fra eksamen INF1000, høst 2013)

- *Du skal nå skrive en funksjon som har tre parametre som tar imot flyttalls-verdier. Funksjonen skal finne den minste av de tre parameterverdiene og returnere denne. Hvis metoden heter minst, så skal f.eks. programsetningen*
  v = minst(3, 1.3, 2.6)
  føre til at variabelen v blir tilordnet verdien 1.3.

- Prøv selv å skrive et komplett svar på denne (5 min)!

Et mulig svar

```
def minst(a, b, c):
    svar=a
    if b < svar:
        svar = b

if c < svar:
    svar = c

return svar
```

Et annet mulig svar

```
def minst(a, b, c):
    svar=a
    if a <= b and a <= c:
        svar = a

    if b <= a and b <= c:
        svar = b

    if c <= a and c <= b:
        svar = c

    return svar
```

Et tredje mulig svar

```
def minst(a, b, c):
    svar=a
    if a <= b and a <= c:
        svar = a
    elif b <= c:
        svar = b
    else:
        svar = c

    return svar
```